تصحيح أخطاء تطبيقات C#

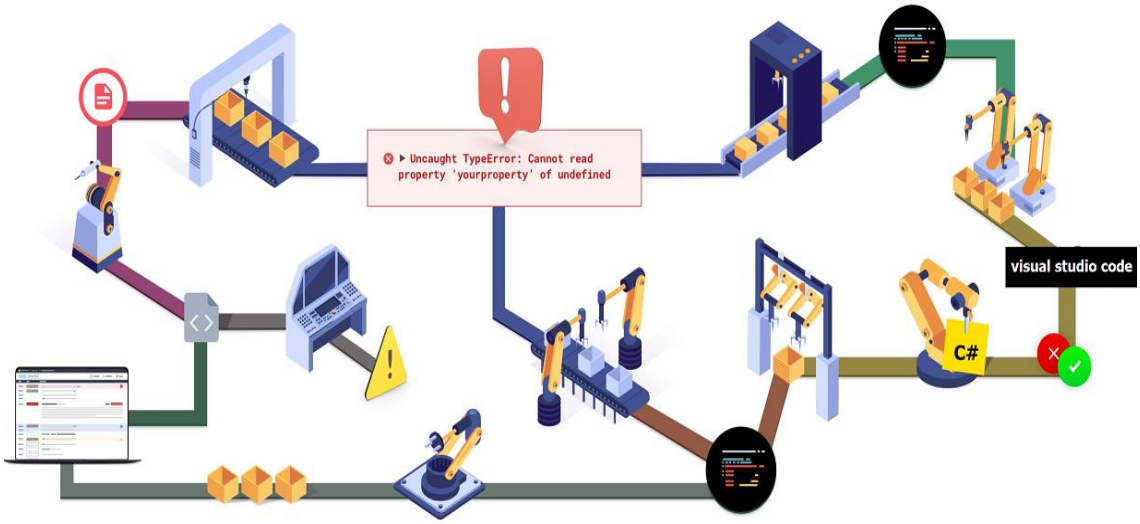
debugger tools and exception handling

C Sharp بالعربي

مجتمع المبرمجين العرب

تعرف على كيفية تصحيح أخطاء تطبيقات وحدة التحكم C# في Visual Studio Code وكيفية تنفيذ معالجة الاستثناءات exception handling باستخدام نمط محاولة الالتقاط try-catch قم بتكوين أدوات مصحح الأخطاء C# واستخدم أدوات مصحح الأخطاء debugger tools لعزل المشكلات المنطقية وإصلاحها fix logic issues

أفحص أنواع الاستثناءات التي يوفرها .NET. وخصائص كائنات الاستثناء exception objects ثم قم بتنفيذ معالجة الاستثناءات، وطرح "تحيدها" كائنات الاستثناء المخصصة throw customized exception objects



المتطلبات الأساسية

- تثبيت Visual Studio Code لتطوير تطبيق C#
- القدرة على تطوير تطبيقات وحدة تحكم C# التي تنفذ منطق العمل باستخدام عبارات التكرار، وعبارات التحديد، والأساليب المخصصة.

محتويات الكتاب:

الوحدة الأولي

مراجعة مبادئ تصحيح أخطاء التعليمات البرمجية code debugging ومعالجة الاستثناءات

افحص عملية تصحيح أخطاء التعليمات البرمجية the code debugging والفوائد التي توفرها مصححات أخطاء التعليمات البرمجية، وتعرف على الاستثناء exception وكيفية استخدام الاستثناءات في أحد التطبيقات.

الأهداف التعليمية

- راجع مسؤوليات اختبار البرامج software testing وتصحيح الأخطاء debugging ومعالجة الاستثناءات exception handling
- افحص عملية تصحيح أخطاء التعليمات البرمجية والفوائد التي توفرها أدوات مصحح أخطاء التعليمات البرمجية code debugger tools
- افحص ما هو الاستثناء وخيارات إدارة الاستثناءات managing exceptions في التعليمات البرمجية.

محتويات الوحدة: -

١- مقدمة

٢- البدء في الاختبار testing وتصحيح الأخطاء debugging ومعالجة الاستثناءات exception handling

٣- فحص نهج مصحح أخطاء التعليمات البرمجية the code debugger في تصحيح الأخطاء.

٤- فحص الاستثناءات exceptions وكيفية استخدام الاستثناءات.

٥- التحقق من المعرفة.

٦- الملخص.

١ المقدمة

عند الإعداد لتطوير تطبيق، تكون كتابة التعليمات البرمجية هي الخطوة الأولى فقطن تبدأ عملية التحقق من أن التعليمات البرمجية تعمل كما هو متوقع، بعد وقت قصير من كتابة الأسطر الأولى من التعليمات، في تطوير البرامج، يتضمن التحقق من التعليمات الاختبار testing وتصحيح الأخطاء debugging ومعالجة الاستثناءات exception handling

لنفترض أنك تطور تطبيقاً، تبين أن تنفيذ المنطق الخاص بميزات معينة أكثر تعقيدًا مما كنت تتوقعه، أنت قادر على إنشاء التعليمات البرمجية وتشغيلها، ولكنك ترى نتائج غير متوقعة ومن الصعب تحديد مكان ظهور المشكلات، بالإضافة إلى ذلك، لاحظت أن تمرير البيانات التي قدمها المستخدم كمعلمة لاستدعاء أسلوب معين، يمكن أن تؤدي إلى أخطاء وقت التشغيل runtime errors إذا لم تجد نهجاً أفضل لتصحيح منطق التعليمات البرمجية code logic وإدارة أخطاء وقت التشغيل managing runtime errors فقد لا تكمل المشروع في الوقت المحدد.

عندما تطلب المشورة من زميلك، فإنهم يذكرونك بأن Visual Studio Code يوفر أدوات تصحيح الأخطاء، وأن C# تتضمن دعماً لمعالجة الاستثناء، قررت أنه حان الوقت لبدء التعرف على تصحيح أخطاء التعليمات البرمجية، ومعالجة الاستثناءات.

في هذه الوحدة، ستتعلم الفرق بين الاختبار testing وتصحيح الأخطاء debugging ومعالجة الاستثناء exception handling يمكنك فحص عملية تصحيح أخطاء التعليمات البرمجية، والفوائد التي توفرها مصححات أخطاء التعليمات البرمجية code debuggers يمكنك أيضاً التعرف على الاستثناءات exceptions وكيفية استخدام الاستثناءات في أحد التطبيقات.

في نهاية هذه الوحدة، ستتمكن من شرح فوائد مصححات التعليمات البرمجية، ومعالجة الاستثناءات.

٢ البدء في الاختبار testing وتصحيح الأخطاء debugging ومعالجة الاستثناءات exception handling

يحتاج كل مطور برامج إلى إكمال مستوى ما من الاختبار، وتصحيح الأخطاء، عند تطوير تعليماته البرمجية، وغالباً ما يكون التعامل مع الاستثناءات مطلوباً، ولكن كيف ترتبط هذه المهام الثلاث، ومتى ينبغي تنفيذها؟

الاختبار وتصحيح الأخطاء ومعالجة الاستثناء

يرتبط تصحيح أخطاء التعليمات البرمجية بشكل واضح بتطوير التعليمات البرمجية واختبارها، بعد كل شيء، يمكنك إجراء تصحيحات على منطق التعليمات أثناء تطوير تطبيقك، ويمكنك أيضاً تشغيل التعليمات بشكل دوري، للتحقق من صحة بناء جملة التعليمات البرمجية code syntax والمنطق logic ولكن هل تحديث منطق التعليمات البرمجية أثناء عملية التطوير، هو المقصود فعلاً بتصحيح الأخطاء debugging؟ وهل التحقق من بناء التعليمات وتشغيلها، هو المقصود بالاختبار testing؟ لا ليس كذلك.

كيف ترتبط معالجة الاستثناء exception handling بتطوير التعليمات البرمجية واختبارها؟ في الواقع، ماذا يعني "معالجة الاستثناء" وهل يتوقع من المطور القيام بذلك؟ في تطوير C# يشار إلى الأخطاء التي تحدث أثناء وقت تشغيل التطبيق runtime على أنها استثناءات **(تختلف عن أخطاء البناء التي تحدث أثناء عملية بناء البرنامج)** و"معالجة الاستثناء" هي العملية التي يستخدمها المطور لإدارة استثناءات وقت التشغيل هذه، ضمن التعليمات البرمجية الخاصة به.

قد تتساءل عن كيفية ارتباط معالجة الاستثناء بتطوير التعليمات البرمجية واختبارها، في الواقع، ماذا يعني "معالجة الاستثناء" وهل من المتوقع أن يقوم المطور بذلك؟ في تطوير C# يُشار إلى الأخطاء التي تحدث أثناء تشغيل التطبيق على أنها استثناءات، يشير مصطلح "exception handling"

معالجة الاستثناء إلى العملية التي يستخدمها المطور لإدارة استثناءات وقت التشغيل هذه، داخل التعليمات البرمجية الخاصة به.

يشار إلى الأخطاء التي تحدث أثناء عملية الإنشاء على أنها أخطاء errors ولا تعد جزءاً من عملية معالجة الاستثناء.

تصف الأقسام التالية دور المطور في الاختبار وتصحيح الأخطاء ومعالجة الاستثناءات.

اختبار البرامج ومسؤوليات المطور

يمكن أن تتضمن عملية تطوير البرامج الكثير من الاختبارات، في الواقع، اختبار البرمجيات له تخصصه الخاص، ويلعب مختبرو البرمجيات دورًا مهمًا في تطوير التطبيقات الكبيرة، حتى أن هناك نهج/طرقًا لعملية تطوير البرمجيات التي تستند إلى الاختبار، مثل التطوير القائم على الاختبار.

يمكن تنظيم فئات اختبار البرامج ضمن أنواع الاختبار types of testing أو نهج الاختبار the approaches to testing أو مزيج من كليهما، إحدى الطرق لتصنيف أنواع الاختبار هي تقسيم الاختبار إلى اختبار وظيفي testing into Functional وغير وظيفي. Nonfunctional testing تتضمن الفئات الوظيفية وغير الوظيفية فئات فرعية للاختبار، على سبيل المثال، يمكن تقسيم الاختبار الوظيفي وغير الوظيفي إلى الفئات الفرعية التالية:

- الاختبار الوظيفي Functional testing - اختبار الوحدة Unit testing - اختبار التكامل Integration testing - اختبار النظام System testing - اختبار القبول Acceptance testing
- اختبار غير وظيفي Nonfunctional testing - اختبار الحماية Security testing - اختبار الأداء Performance testing – اختبار سهولة الاستخدام Usability testing - اختبار التوافق Compatibility testing

على الرغم من أن معظم المطورين ربما لا يعتبرون أنفسهم مختبرين، إلا أنه من المتوقع إجراء مستوى من الاختبار قبل أن يسلم المطورين عملهم. عندما يتم تعيين دور رسمي للمطورين في عملية الاختبار، فغالبًا ما يكون ذلك على مستوى اختبار الوحدة unit testing

ملاحظة

نظرا لأن اختبار البرامج هو موضوع كبير، وغالباً يتم تنفيذه من خلال دور وظيفي منفصل، فلن تتم مناقشة الطرق الرسمية لاختبار البرامج في هذه الدروس

تصحيح أخطاء التعليمات البرمجية ومسؤوليات المطور

تصحيح أخطاء التعليمات البرمجية Code debugging هي عملية يستخدمها المطورون لعزل مشكلة، وتحديد طريقة واحدة أو أكثر لإصلاحها، قد تكون المشكلة مرتبطة إما بمنطق التعليمات البرمجية أو استثناء، وفي كلتا الحالتين، يمكنك العمل على تصحيح أخطاء التعليمات البرمجية عندما لا تعمل بالطريقة التي تريدها، بشكل عام، يتم حجز مصطلح تصحيح الأخطاء debugging لمشكلات وقت التشغيل runtime issues التي ليس من السهل عزلها، لذلك، إصلاح مشكلات بناء الجملة مثل علامة ";" مفقودة في نهاية عبارة، لا يعتبر عادةً تصحيحًا للأخطاء.

خذ بعين الاعتبار نموذج التعليمات البرمجية التالي:

```
string[] students = new string[] {"Sophia",
"Nicolas", "Zahirah", "Jeong"};

int studentCount = students.Length;

Console.WriteLine("The final name is: " +
students[studentCount]);
```

يهدف نموذج التعليمات البرمجية إلى إنجاز ما يلي:

- تعريف مصفوفة string باسم `students` تحتوي المصفوفة `students` على أسماء الطلاب.
- تعريف متغير عدد صحيح يسمى `studentCount` تستخدم التعليمات البرمجية خاصية `Length` للمصفوفة، لتعيين قيمة إلى `studentCount`
- اطبع اسم الطالب النهائي على وحدة التحكم، تستخدم التعليمات البرمجية `studentCount` للوصول إلى الاسم النهائي في مصفوفة `students` وتستخدم الأسلوب `Console.WriteLine()` لطباعة المعلومات إلى وحدة التحكم.

للوهلة الأولى، كل شيء يبدو جيداً، ومع ذلك، تنشئ هذه التعليمة البرمجية استثناء عند محاولة طباعة اسم الطالب إلى وحدة التحكم، نسي المطور أن فهرس المصفوفات يستند إلى الصفر، يجب الوصول إلى الاسم النهائي في المصفوفة باستخدام `students[studentCount - 1]`

تصحيح أخطاء التعليمات البرمجية هو بالتأكيد مسؤولية المطور، في نموذج التعليمات البرمجية هذا، ربما تكون قد تعرفت على المشكلة على الفور، ومع ذلك، في سيناريوهات الترميز الأكثر تعقيداً، لا يكون العثور على مشكلة أمراً سهلاً دائماً، لا تقلق، هناك أدوات وطرق يمكنك استخدامها لتعقب المشكلات التي يصعب العثور عليها.

معالجة الاستثناءات ومسؤوليات المطور

كما قرأت سابقاً، يشار إلى الأخطاء التي تحدث أثناء وقت تشغيل التطبيق على أنها استثناءات، إذا كان أحد التطبيقات ينشئ استثناء، ولم تتم إدارة هذا الاستثناء في التعليمات البرمجية، يمكن أن يؤدي إلى إيقاف تشغيل التطبيق.

التعامل مع الاستثناءات هو بالتأكيد مسؤولية المطور، توفر C# طريقة لتجربة "try" التعليمات البرمجية التي تعرف أنها قد تنشئ استثناء، وطريقة ل التقاط "catch" أي استثناءات تحدث.

خلاصة

فيما يلي بعض الأشياء المهمة التي يجب تذكرها من هذا الدرس:

- الاختبار وتصحيح الأخطاء ومعالجة الاستثناءات كلها مهام مهمة لمطوري البرامج.
- يمكن تصنيف الاختبار إلى اختبار وظيفي وغير وظيفي، ومن المتوقع أن يقوم المطورون بإجراء مستوى معين من الاختبار.
- تصحيح أخطاء التعليمات البرمجية هو عملية عزل المشكلات وتحديد طرق إصلاحها، وهي مسؤولية المطور.
- معالجة الاستثناءات هي عملية إدارة الأخطاء التي تحدث أثناء وقت التشغيل، والمطورين مسؤولون عن معالجة الاستثناءات باستخدام عبارات try and catch في التعليمات البرمجية.

راجع معلوماتك

١- ما هو تصحيح أخطاء debugging التعليمات البرمجية؟

- عملية لتحديد أخطاء بناء جملة التعليمات البرمجية أثناء عملية الإنشاء.
- عملية لعزل المشكلات وإصلاحها في منطق التعليمات البرمجية.
- عملية للتحقق من أن التعليمات البرمجية تبني وتعمل دون أخطاء.

٢. من المسؤول عن اختبار البرامج؟

- مطورو البرامج.
- مختبرو البرامج.
- كل من المطورين والمختبرين.

٣. ماذا يحدث إذا لم تتم إدارة استثناء في التعليمات البرمجية؟

- يستمر تشغيل التعليمات البرمجية دون أي مشكلة.
- ينشئ التطبيق رسالة خطأ ويغلق.
- يتم عرض أخطاء بناء جملة التعليمات البرمجية للمستخدم.

راجع إجابتك

١ عملية لعزل المشكلات وإصلاحها في منطق التعليمات البرمجية

صحيح يتضمن تصحيح أخطاء التعليمات البرمجية عزل وإصلاح مشكلات المنطق في التعليمات البرمجية

٢ كل من المطورين والمختبرين

صحيح يشارك مطورو البرامج والمختبرون مسؤولية اختبار التعليمات البرمجية

٣ ينشئ التطبيق رسالة خطأ ويغلق

صحيح إذا لم تتم إدارة استثناء في التعليمات البرمجية، فسيعرض وقت التشغيل رسالة خطأ وقد يتم إنهاء التطبيق

٣ فحص نهج مصحح أخطاء التعليمات البرمجية the code debugger

يجب على كل مطور التعامل مع الأخطاء البرمجية، على إنها مجرد أسلوب حياة للمطورين، في بعض الأحيان يمكنك اكتشاف الأخطاء بسرعة. بعد كل شيء، لقد كتبت التعليمات، من الجيد العثور على مشكلة وإصلاحها بسرعة، ورغم ذلك، حتماً، ستكون هناك أوقات تجد فيها نفسك تبحث عن خطأ ليس من السهل اكتشافه.

عملية تصحيح أخطاء التعليمات البرمجية Code debugging

عندما تلاحظ وجود خطأ في التعليمات البرمجية، قد يكون من المغري تجربة نهج مباشر، تعرف، هذا الفحص السريع حيث تعتقد أن المشكلة قد تكون، إذا كان يأتي ثماره في أول ٣٠ ثانية، فهذا رائع، لكن لا تدع نفسك تنجرف، لا تستمر في الانتقال إلى المكان التالي، والذي يليه. لا تدع نفسك تضيع الوقت مقابل النهج التالي:

- قراءة التعليمات البرمجية (مرة واحدة أخرى) على أمل أن تجد المشكلة هذه المرة.
- التنقل التفصيلي لعدد قليل من `Console.WriteLine("here")` الرسائل في التعليمات البرمجية لتتبع التقدم من خلال تطبيقك.
- إعادة تشغيل تطبيقك ببيانات مختلفة، على أمل أنه إذا رأيت ما ينجح، فستفهم ما لا ينجح.

ربما تكون قد حققت درجات مختلفة من النجاح باستخدام هذه السُبل، لكن لا تنخدع، هناك طريقة أفضل.

النهج الوحيد الذي ينظر إليه عادة على أنه الأكثر نجاحاً هو استخدام مصحح الأخطاء debugger ولكن ما هو مصحح الأخطاء بالضبط؟

مصحح الأخطاء debugger هو أداة برمجية تستخدم لمراقبة، والتحكم في تدفق التنفيذ من البرنامج، باستخدام نهج تحليلي، تساعدك مصححات الأخطاء في عزل سبب الخطأ، وتساعدك على حله، يتصل مصحح الأخطاء بالتعليمات البرمجية باستخدام أحد نهجين:

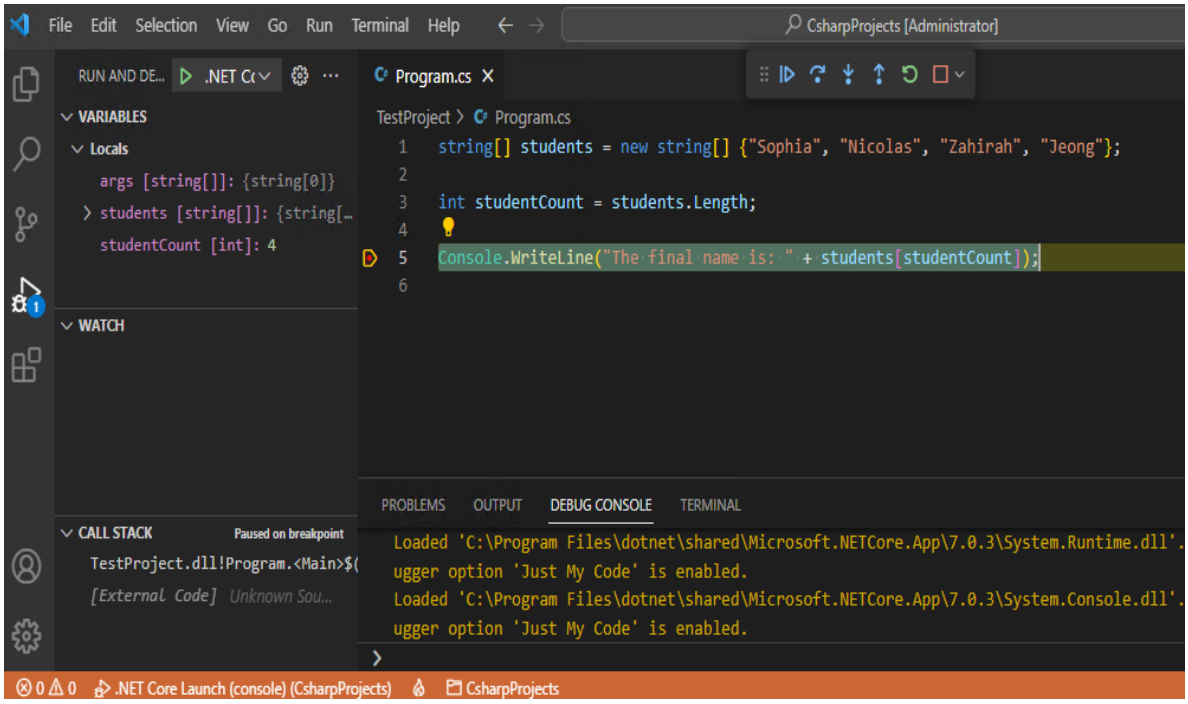
- من خلال استضافة برنامجك في عملية التنفيذ الخاصة به.
- من خلال التشغيل كعملية منفصلة مرتبطة ببرنامجك قيد التشغيل.

تأتي مصححات الأخطاء Debuggers بنكهات مختلفة، يعمل بعضها مباشرة من سطر الأوامر بينما يأتي البعض الآخر مزودًا بواجهة مستخدم رسومية، يدمج Visual Studio Code أدوات مصحح الأخطاء في واجهة المستخدم.

لماذا نستخدم مصحح الأخطاء

إذا كنت لا تقوم بتشغيل التعليمات البرمجية من خلال مصحح أخطاء، فمن المحتمل أنك تخمن ما يحدث في تطبيقك في وقت التشغيل، الفائدة الأساسية من استخدام مصحح الأخطاء هو أنه يمكنك مشاهدة برنامجك قيد التشغيل، يمكنك متابعة تنفيذ البرنامج لسطر واحد، من التعليمات البرمجية في كل مرة، يقلل هذا النهج من فرصة التخمين الخاطئ.

يدعم Visual Studio Code مصححات أخطاء التعليمات البرمجية التي تمكنك من مشاهدة تعليماتك أثناء تشغيلها، تُظهر الصورة التالية تطبيقاً قيد التشغيل، مع إيقاف التنفيذ مؤقتاً على سطر التعليمات البرمجية المميز، يظهر الجانب الأيمن من الشاشة التعليمات البرمجية للبرنامج، بينما يعرض الجانب الأيسر الحالة الحالية للمتغيرات.



لكل مصحح أخطاء مجموعة الميزات الخاصة به، وأهم ميزتين تأتيان مع كل المصححات تقريبًا هما:

- **التحكم في تنفيذ البرنامج**. يمكنك إيقاف البرنامج مؤقتاً وتشغيله خطوة بخطوة، ما يسمح لك بالاطلاع على التعليمات البرمجية التي يتم تنفيذها وكيفية تأثيرها على حالة البرنامج.
- **مراقبة حالة البرنامج الخاص بك**. على سبيل المثال، يمكنك إلقاء نظرة على قيمة المتغيرات، والمعلمات في أي وقت أثناء تنفيذ تعليماتك البرمجية.

يعد إتقان استخدام مصحح أخطاء التعليمات البرمجية مهارة مهمة، لسوء الحظ، إنها مهارة غالباً ما يغفلها المطورون، يساعدك الاستخدام الفعال لمصحح الأخطاء على أن تكون أكثر كفاءة في تتبع الأخطاء في تعليماتك البرمجية، يمكن أن يساعدك مصحح الأخطاء أيضاً على فهم كيفية عمل البرنامج.

خلاصة

فيما يلي بعض الأشياء المهمة التي يجب تذكرها من هذا الدرس:

- تصحيح أخطاء التعليمات البرمجية هي مهارة حاسمة في عملية تطوير البرامج التي يتعلمها كل مطور.
- أفضل طريقة لتصحيح أخطاء تطبيقاتك هي استخدام مصحح أخطاء، وليس إعادة قراءة التعليمات البرمجية خمس مرات أو إضافة عبارات `console.WriteLine()` في التعليمات البرمجية.
- تمكنك مصححات الأخطاء من إيقاف التطبيق مؤقتاً، والتنقل عبر التعليمات البرمجية سطراً سطراً، ومراقبة حالة المتغيرات والمعلمات function parameters

راجع معلوماتك

١- ما هو مصحح الأخطاء؟

- أداة برمجية تستخدم للمراقبة والتحكم في تدفق تنفيذ البرنامج.
- برنامج يقوم بإنشاء التعليمات البرمجية تلقائياً بناءً على إدخال المستخدم.
- أداة تساعد المطورين على كتابة تعليماتهم البرمجية بشكل أسرع.

٢. ما هي الفائدة الأساسية لاستخدام مصحح الأخطاء؟

- يساعد مصحح الأخطاء المطورين على كتابة التعليمات بشكل أسرع.
- يمكّن مصحح الأخطاء المطورين من مشاهدة تطبيقهم قيد التشغيل ومتابعة تنفيذ البرنامج سطرًا واحداً من التعليمات البرمجية في كل مرة.
- يساعد المصحح المطورين على إضافة ميزات جديدة إلى تطبيقهم.

٣. ما هي إحدى أهم ميزات مصحح الأخطاء؟

- إنشاء التعليمات البرمجية.
- مراقبة حالة البرنامج الخاص بك.
- إصلاح الأخطاء تلقائياً.

٤. ما هي أفضل طريقة للعثور على السبب الجذري للخطأ؟

- تشغيل تطبيقك ببيانات مختلفة.
- التنقل بين بعض الرسائل (Console.WriteLine("here" في التعليمات البرمجية.
- استخدام مصحح أخطاء.

راجع إجابتك

١ أداة برمجية تستخدم للمراقبة والتحكم في تدفق تنفيذ البرنامج.

صحيح مصحح الأخطاء هو أداة برمجية تستخدم نهجاً تحليلياً للمراقبة والتحكم في تدفق تنفيذ البرنامج

٢ يمكّن مصحح الأخطاء المطورين من مشاهدة تطبيقهم قيد التشغيل، ومتابعة تنفيذ البرنامج سطرًا واحداً من التعليمات البرمجية في كل مرة.

صحيح الفائدة الأساسية لاستخدام مصحح الأخطاء هي مشاهدة تشغيل التعليمات البرمجية للتطبيق، ومتابعة تنفيذ البرنامج سطر واحد من التعليمات البرمجية في كل مرة

٣ مراقبة حالة البرنامج الخاص بك.

صحيح ملاحظة حالة برنامجك هي واحدة من أهم الميزات التي تأتي مع جميع مصححي الأخطاء تقريباً

٤ استخدام مصحح أخطاء.

صحيح استخدام مصحح الأخطاء هو أفضل طريقة للعثور على السبب الجذري للخطأ

٤ فحص الاستثناءات exceptions وكيفية استخدام الاستثناءات

في وقت سابق من هذه الوحدة، تعلمت أن أخطاء وقت التشغيل في C# تسمى استثناءات exceptions وأنك تحتاج إلى التقاطها "catch" قبل تعطل تطبيقك. حقا؟ يبدو التقاط الاستثناءات قبل تعطل التطبيق وكأنه لعبة فيديو أكثر من كتابة تطبيق، إذن ماذا يعني بالضبط التقاط استثناء "catch" an exception؟ للإجابة على هذا السؤال، تحتاج إلى البدء بإلقاء نظرة فاحصة على ما هو الاستثناء.

ما هي الاستثناءات؟

فيما يلي تعريف رسمي إلى حد ما، يصف ما هو الاستثناء، وكيفية استخدام استثناء في تطبيق C#:

في C# يتم نشر الأخطاء الموجودة في البرنامج أثناء وقت التشغيل، من خلال البرنامج باستخدام آلية تسمى الاستثناءات، يتم عزل الاستثناءات بواسطة التعليمات البرمجية التي تواجه خطأً، ويتم اكتشافها بواسطة التعليمات البرمجية التي يمكنها تصحيح الخطأ، يمكن عزل الاستثناءات بواسطة وقت تشغيل NET. أو بواسطة التعليمات البرمجية في برنامج، يتم تمثيل الاستثناءات بواسطة فئات مشتقة من استثناء classes derived from Exception تحدد كل فئة نوع الاستثناء وتحتوي على خصائص تحتوي على تفاصيل حول الاستثناء.

هام

لا يتطلب هذا التدريب فهماً عميقاً لفئات NET. لا تقلق إذا كان هذا التعريف مربكاً بعض الشيء، يمكنك استخدام الاستثناءات في التعليمات البرمجية الخاصة بك دون فهم عميق للفئات.

تتناول وثائق Microsoft المتعلقة بالاستثناءات قدرًا كبيرًا من التفاصيل، ورغم ذلك، فإن هذا التعريف يوفر المعلومات التي تحتاجها الآن، وعلى وجه التحديد، عليك أن تفهم شيئين:

- يجب أن تفهم ما هي الاستثناءات.
- يجب أن تفهم كيفية استخدام الاستثناءات في تطبيقاتك.

يمكنك التفكير في استثناء كمتغير له إمكانيات إضافية، يمكنك القيام بنفس نوع الأشياء مع الاستثناءات التي تقوم بها مع المتغيرات، على سبيل المثال:

- يمكنك إنشاء أنواع مختلفة من الاستثناءات.
- يمكنك الوصول إلى محتويات استثناء.

ماذا يعني طرح/عزل "throw" والتقاط "catch" استثناء؟

يمكن شرح مصطلحي "throw" و"catch" من خلال تقييم تعريف الاستثناء.

الجملة الثانية من التعريف تقول "يتم طرح الاستثناءات بواسطة التعليمات البرمجية التي تواجه خطأ ويتم اكتشافها بواسطة التعليمات البرمجية التي يمكنها تصحيح الخطأ" يخبرك الجزء الأول من هذه الجملة أنه يتم إنشاء استثناءات بواسطة وقت تشغيل NET. عند حدوث خطأ في التعليمات البرمجية، يخبرك الجزء الثاني من الجملة أنه يمكنك كتابة التعليمات البرمجية لالتقاط استثناء تم طرحه، بالإضافة إلى ذلك، يمكن استخدام التعليمات البرمجية التي تلتقط الاستثناء لإكمال إجراء تصحيحي، على أمل تخفيف الموقف الناجم عن التعليمات البرمجية التي أدت إلى الخطأ، بمعنى آخر، يمكنك كتابة التعليمات البرمجية التي تحمي تطبيقك عند حدوث خطأ.

بعد تقييم تلك الجملة الثانية من التعريف، تعرف ما يلي:

- يتم إنشاء استثناء في وقت التشغيل عندما تنتج التعليمات البرمجية خطأ.
- يمكن التعامل مع الاستثناء كمتغير يحتوي على بعض الإمكانات الإضافية.

- يمكنك كتابة التعليمات البرمجية التي تصل إلى الاستثناء وتتخذ إجراء تصحيحياً.

يخبرك الجزء المتبقي من التعريف أنه إذا اكتشف وقت تشغيل NET. خطأ، فإنه ينشئ الاستثناء، يحتوي الاستثناء الذي تم إنشاؤه على معلومات حول الخطأ الذي حدث، يمكن لتعليماتك البرمجية التقاط استثناء، وتصحيح المشكلة باستخدام المعلومات المخزنة في الاستثناء.

خلاصة

فيما يلي بعض الأشياء المهمة التي يجب تذكرها من هذا الدرس:

- يتم استخدام الاستثناءات في C# لنشر الأخطاء في وقت التشغيل، بواسطة فئات مشتقة من فئة Exception
- يتم طرح/عزل الاستثناءات بواسطة التعليمات البرمجية التي تواجه خطأ، ويتم التقاطها بواسطة التعليمات البرمجية التي يمكنها تصحيح الخطأ.
- عند اكتشاف استثناء، يمكن للتعليمات البرمجية الوصول إلى محتوياته واتخاذ إجراء تصحيحي للتخفيف من الخطأ.
- ينشئ وقت تشغيل NET. استثناءات عندما يكتشف خطأ، ويحتوي الاستثناء على معلومات حول نوع الخطأ الذي حدث.

راجع معلوماتك

١. ما الغرض من طرح/عزل استثناء في C#؟

- لتجاهل الأخطاء التي تحدث في برنامج.
- لاتخاذ إجراء تصحيحي عند حدوث خطأ في برنامج.
- لإنشاء أخطاء إضافية في برنامج.

٢. هل يمكن للمطور الوصول إلى محتويات استثناء في وقت التشغيل؟

- لا يمكن الوصول إلى الاستثناءات.
- نعم، يمكن الوصول إلى الاستثناءات واستخدامها لاتخاذ إجراء تصحيحي.
- يمكن الوصول إلى الاستثناءات، ولكن ليس أثناء وقت تشغيل التطبيق.

٣. ماذا يحدث عندما ينتج عن تنفيذ تطبيق C# خطأ في النظام؟

- يتعطل البرنامج ويغلق.
- يتم إنشاء استثناء وطرحه بواسطة وقت تشغيل NET.
- يتم تجاهل الخطأ بصمت ويستمر تشغيل البرنامج.

٤. ما هي العلاقة بين نوع الاستثناء والمعلومات التي يحتوي عليها؟

- نوع الاستثناء والمعلومات التي يحتوي عليها غير مرتبطين.
- يحدد نوع الاستثناء المعلومات التي يحتوي عليها.
- تحدد المعلومات الواردة في استثناء نوع الاستثناء.

راجع إجابتك

١ لاتخاذ إجراء تصحيحي عند حدوث خطأ في برنامج.

صحيح الغرض من التقاط استثناء هو اتخاذ إجراء تصحيحي عند حدوث خطأ

٢ نعم، يمكن الوصول إلى الاستثناءات واستخدامها لاتخاذ إجراء تصحيحي.

صحيح الاستثناءات هي كائنات يمكن الوصول إليها، يمكن استخدام خصائص الاستثناء للمساعدة في تحديد الإجراء التصحيحي

٣ يتم إنشاء استثناء وطرحه بواسطة وقت تشغيل .NET

صحيح يطرح وقت تشغيل .NET. استثناء عندما ينشئ تطبيق C# خطأ في النظام

٤ يحدد نوع الاستثناء المعلومات التي يحتوي عليها.

صحيح يحدد نوع الاستثناء المعلومات التي يحتوي عليها

٦ التحقق من المعرفة

اختبر معلوماتك

١. ما هو تصحيح أخطاء التعليمات البرمجية code debugging؟

- عملية لتحديد أخطاء بناء جملة التعليمات البرمجية أثناء عملية الإنشاء.
- عملية لعزل المشكلات وإصلاحها في منطق التعليمات البرمجية.
- عملية للتحقق من أن التعليمات البرمجية تبني وتعمل دون أخطاء.

٢. ما هو مصحح الأخطاء debugger؟

- أداة برمجية تستخدم للمراقبة والتحكم في تدفق تنفيذ البرنامج.
- برنامج يقوم بإنشاء التعليمات البرمجية تلقائياً بناءً على إدخال المستخدم.
- أداة تساعد المطورين على كتابة تعليماتهم البرمجية بشكل أسرع.

٣. ماذا يحدث عندما ينتج عن تنفيذ تطبيق C# خطأ في النظام؟

- يتعطل البرنامج ويغلق.
- يقوم وقت تشغيل .NET بإنشاء استثناء وطرحه.
- يتم تجاهل الخطأ بصمت ويستمر تشغيل البرنامج.

٤. ما الغرض من اصطياد catching استثناء في C#؟

- لتجاهل الأخطاء التي تحدث في البرنامج.
- لإنشاء استثناءات أخرى في البرنامج.
- لاتخاذ الإجراءات التصحيحية عند حدوث خطأ في البرنامج.

٥. ما هي العلاقة بين نوع الاستثناء والمعلومات التي يحتوي عليها؟

- يحدد نوع الاستثناء المعلومات التي يحتوي عليها.
- نوع الاستثناء والمعلومات التي يحتوي عليها غير مرتبطين.
- تحدد المعلومات الواردة في استثناء نوع الاستثناء.

راجع إجابتك

١ عملية لعزل المشكلات وإصلاحها في منطق التعليمات البرمجية.

صحيح يتضمن تصحيح أخطاء التعليمات البرمجية عزل وإصلاح مشكلات المنطق في التعليمات البرمجية

٢ أداة برمجية تستخدم للمراقبة والتحكم في تدفق تنفيذ البرنامج.

صحيح مصحح الأخطاء هو أداة برمجية تستخدم نهجاً تحليلياً للمراقبة والتحكم في تدفق تنفيذ البرنامج

٣ يقوم وقت تشغيل NET. بإنشاء استثناء وطرحه.

صحيح يطرح وقت تشغيل NET. استثناء عندما ينشئ تطبيق C# خطأ في النظام

٤ لاتخاذ إجراء تصحيحي عند حدوث خطأ في البرنامج.

صحيح الغرض من التقاط استثناء هو اتخاذ إجراء تصحيحي عند حدوث خطأ

٥ يحدد نوع الاستثناء المعلومات التي يحتوي عليها.

صحيح يحدد نوع الاستثناء المعلومات التي يحتوي عليها

٦ الملخص

كان هدفك هو التعرف على عمليات تصحيح أخطاء التعليمات البرمجية code debugging ومعالجة الاستثناءات exception handling

من خلال فحص تصحيح التعليمات البرمجية، وعمليات معالجة الاستثناء، تعرفت على الأدوات والآليات التي تمكن العمليات من العمل، لقد تعلمت ما يفعله مصحح أخطاء التعليمات البرمجية a code debugger والفوائد المقدمة باستخدام مصحح أخطاء التعليمات البرمجية.

تعرفت أيضاً على العلاقة بين الأخطاء errors والاستثناءات exceptions وكيفية طرح الاستثناءات وإلقاءها thrown and caught أثناء تنفيذ التعليمات البرمجية.

بدون المعرفة المفاهيمية التي اكتسبتها، لن تكون مستعداً لبدء استخدام تصحيح أخطاء التعليمات البرمجية code debugging ومعالجة الاستثناء exception handling في تطبيقات C# الخاصة بك.

الوحدة الثانية

تنفيذ أدوات تصحيح الأخطاء Visual Studio Code

تعرف على كيفية تصحيح أخطاء برامج C# بشكل فعال في Visual Studio Code باستخدام نقاط التوقف breakpoints وأدوات تصحيح الأخطاء الأخرى debugging tools مثل الموارد resources في طريقة عرض RUN AND DEBUG

الأهداف التعليمية

- تكوين مصحح أخطاء Visual Studio Code لبرنامج C#
- إنشاء نقاط توقف breakpoints والتنقل عبر التعليمات البرمجية لعزل المشكلات.
- افحص حالة البرنامج في أي خطوة تنفيذ.
- استخدم مكدس الاستدعاءات the call stack للعثور على مصدر استثناء.

محتويات الوحدة: -

١ المقدمة

كلما اكتشفت الأخطاء وتعرفت عليها بشكل أسرع، زادت سرعة تثبيت التعليمات البرمجية وتحريرها، يدعم Visual Studio Code تصحيح أخطاء التعليمات البرمجية ل C# ومعظم لغات تطوير البرامج الأخرى من خلال استخدام الملحقات، بمجرد أن تتعلم استخدام أدوات تصحيح أخطاء Visual Studio Code ستقضي وقتاً أقل في التساؤل عن سبب توقف التعليمات البرمجية عن العمل، وقضاء المزيد من الوقت في تطوير تطبيقات رائعة.

لنفترض أنك تستخدم Visual Studio Code لتطوير تطبيق وحدة تحكم C# الغرض الأساسي من التطبيق هو معالجة بيانات العملاء بناءً على قواعد العمل، يمكنك تطوير التطبيق باستخدام مجموعة بيانات، عينة صغيرة، ويتم تشغيله بدون أخطاء، ومع ذلك، عند تشغيل التعليمات البرمجية باستخدام مجموعة بيانات أكبر، تنتج التعليمات البرمجية الخاصة بك بعض النتائج غير المتوقعة، قرأت التعليمات البرمجية عدة مرات ولكن من الصعب العثور على الأخطاء في المنطق الخاص بك، وسمعت أن Visual Studio Code يحتوي على أدوات مصحح أخطاء جيدة، ولكنك لم تضطر إلى استخدامها من قبل، وقررت أن تعلم أدوات مصحح الأخطاء هو أفضل فرصة لك لإكمال المشروع في الوقت المحدد.

في هذه الوحدة، ستتعلم كيفية تصحيح أخطاء برامج C# بشكل فعال في Visual Studio Code باستخدام نقاط التوقف breakpoints وأدوات تصحيح الأخطاء الأخرى، مثل الموارد resources في طريقة عرض RUN AND DEBUG

في نهاية هذه الوحدة، ستتمكن من تكوين واستخدام أدوات مصحح أخطاء Visual Studio Code ل C# وستتمكن من عزل أخطاء التعليمات البرمجية بكفاءة، باستخدام أدوات مصحح الأخطاء، ولن تحتاج إلى الاعتماد على Console.WriteLine بعد الآن.

٢ فحص واجهة مصحح أخطاء Visual Studio Code

توفر واجهة مستخدم Visual Studio Code عدة طرق لتكوين خيارات تتبع الأخطاء، وبدء جلسات تصحيح الأخطاء.

ميزات تصحيح الأخطاء في واجهة مستخدم Visual Studio Code

يتضمن Visual Studio Code العديد من ميزات واجهة المستخدم التي تساعدك على تكوين جلسات تصحيح الأخطاء وبدء تشغيلها وإدارتها:

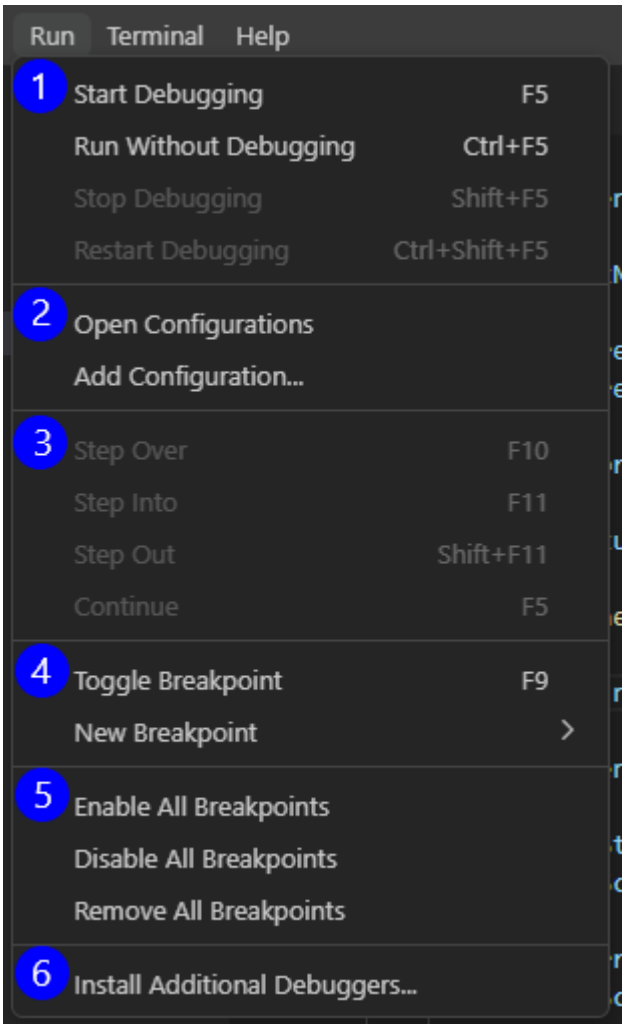
- تكوين مصحح الأخطاء وتشغيله: يمكن استخدام قائمة التشغيل Run menu وعرض RUN AND DEBUG لتكوين جلسات تصحيح الأخطاء وتشغيلها.
- فحص حالة التطبيق: تتضمن طريقة العرض RUN AND DEBUG واجهة قوية، تعرض جوانب مختلفة من حالة التطبيق أثناء جلسة تصحيح الأخطاء.
- التحكم في تنفيذ وقت التشغيل: يوفر شريط أدوات تتبع الأخطاء Debug toolbar عناصر تحكم عالية المستوى لوقت التشغيل، أثناء تنفيذ التعليمات البرمجية.

ملاحظة

تعرفك هذه الوحدة على الكثير من أدوات تصحيح الأخطاء، والمصطلحات، يرجى أن تضع في اعتبارك أن هذه هي أول نظرة على هذه الأدوات، وليست الأخيرة، ستتوفر لديك فرصة لإكمال الأنشطة العملية مع معظم هذه الأدوات أثناء هذه الوحدة، حاول ألا تشعر بالإرهاق، بسبب حجم المعلومات المقدمة.

خيارات قائمة تشغيل Run

توفر قائمة Visual Studio Code Run وصولاً سهلاً إلى بعض أوامر التشغيل، وتصحيح الأخطاء الشائعة.



توفر قائمة Run خيارات القائمة التي يتم تجميعها في ستة أقسام.

١. **بدء وإيقاف التطبيقات** Start and stop applications يتضمن هذا القسم من القائمة خيارات لبدء وإيقاف تنفيذ التعليمات البرمجية، مع أو بدون إرفاق مصحح الأخطاء

٢. **تشغيل التكوينات** Launch configurations يوفر هذا القسم من القائمة إمكانية الوصول لفحص تكوينات التشغيل أو إنشائها.

٣. **التحكم في وقت التشغيل** Runtime control يمكّن هذا القسم من القائمة المطور من التحكم بالطريقة التي يريدها للتقدم من خلال التعليمات البرمجية، يتم تمكين عناصر التحكم عند توقف التنفيذ مؤقتًا أثناء جلسة تصحيح الأخطاء.

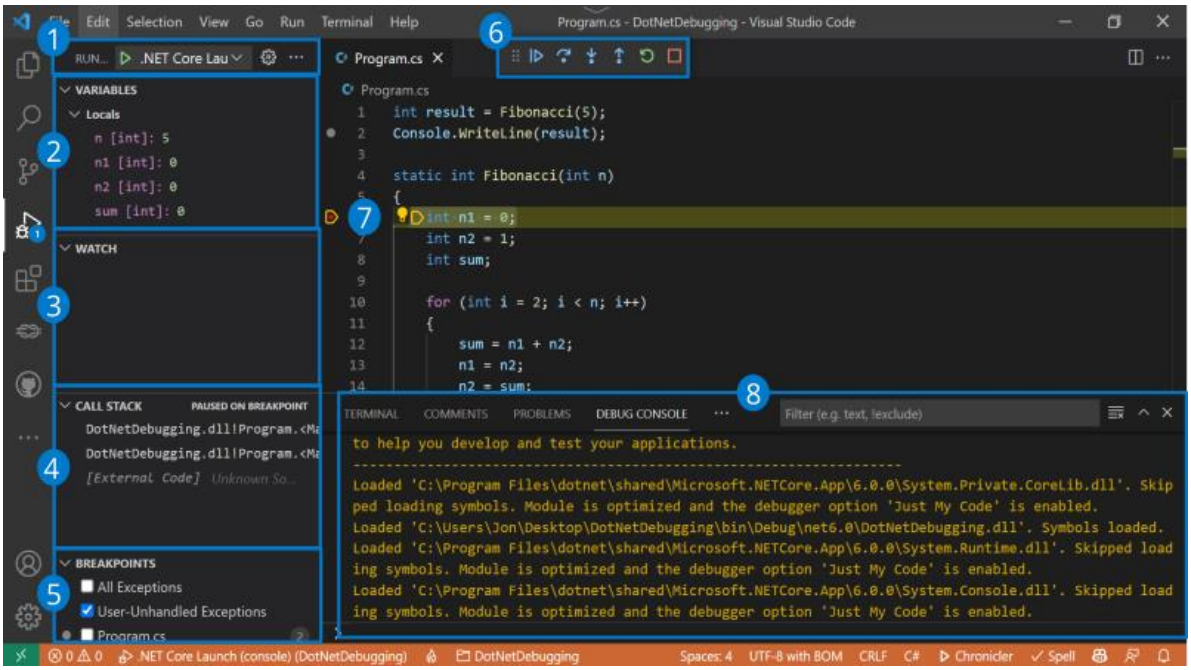
٤. **تعيين نقاط التوقف** Set Breakpoints يمكّن هذا القسم من القائمة المطور من تعيين نقاط التوقف عند أسطر محددة من التعليمات البرمجية، يتوقف تنفيذ التعليمات مؤقتاً عند نقاط التوقف، أثناء جلسة تصحيح الأخطاء.

٥. **إدارة نقاط التوقف** Manage Breakpoints يمكّن هذا القسم من القائمة المطور من إدارة نقاط التوقف، بشكل مجمع وليس بشكل فردي، كلاً على حدة.

٦. **تثبيت مصححات الأخطاء** Install Debuggers يفتح هذا القسم من القائمة، افتح قائمة الملحقات EXTENSIONS وتصفيتها على ملحقات مصححات أخطاء التعليمات البرمجية.

واجهة مستخدم قائمة التشغيل والتصحيح RUN AND DEBUG

توفر قائمة RUN AND DEBUG الوصول إلى أدوات وقت التشغيل، التي يمكن أن تكون لا تقدر بثمن أثناء عملية التصحيح.



١. لوحة تحكم، التشغيل والتصحيح Run and Debug controls panel يستخدم لتكوين جلسة تصحيح الأخطاء، وبدء تشغيلها.

٢. قسم المتغيرات VARIABLES section يستخدم لعرض وإدارة حالة المتغير، أثناء جلسة تصحيح الأخطاء.

٣. قسم المشاهدة WATCH section يستخدم لمراقبة المتغيرات أو التعبيرات، على سبيل المثال، يمكنك تكوين تعبير باستخدام متغير واحد أو أكثر، ومشاهدته لمعرفة متى يتم استيفاء شرط معين.

٤. قسم مكدس الاستدعاءات CALL STACK section يُستخدم لتتبع نقطة التنفيذ الحالية داخل التطبيق قيد التشغيل، بدءًا من نقطة الدخول الأولية إلى التطبيق، يُظهر مكدس الاستدعاءات الأسلوب الذي يتم تنفيذه حاليًا، بالإضافة إلى الأسلوب أو الأساليب الموجودة في مسار التنفيذ، والتي أدت إلى نقطة التنفيذ الحالية (السطر الحالي من التعليمات البرمجية).

٥. قسم نقاط التوقف BREAKPOINTS section يعرض إعدادات نقطة التوقف الحالية.

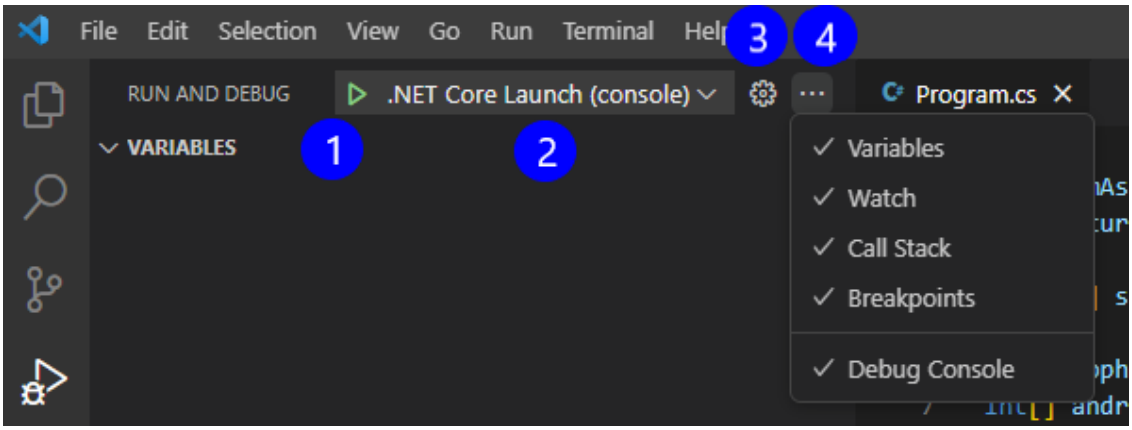
٦. شريط أدوات تتبع الأخطاء Debug toolbar ويسمي أيضاً أدوات التصحيح، يستخدم للتحكم في تنفيذ التعليمات البرمجية أثناء عملية التصحيح. يتم عرض شريط الأدوات هذا فقط أثناء تشغيل التطبيق.

٧. خطوة التنفيذ الحالية Current execution step يُستخدم لتحديد خطوة التنفيذ الحالية من خلال تمييزها في المحرر، في هذه الحالة، خطوة التنفيذ الحالية، خطوة التنفيذ الحالية بمثابة نقطة توقف (يتم تمييز نقاط التوقف بنقطة حمراء على يسار رقم السطر).

٨. وحدة تحكم تتبع الأخطاء DEBUG CONSOLE يستخدم لعرض الرسائل من مصحح الأخطاء، لوحة DEBUG CONSOLE هي وحدة التحكم الافتراضية لتطبيقات وحدة التحكم console applications وهي قادرة على عرض الإخراج من `Console.WriteLine()` وأساليب الإخراج ذات الصلة `Console`

لوحة تحكم، التشغيل والتصحيح Run and Debug controls panel

في الجزء العلوي من قائمة عرض RUN AND DEBUG يمكنك العثور على عناصر التحكم في التشغيل:



١. بدء تصحيح الأخطاء ل يستخدم هذا الزر (سهم أخضر) لبدء جلسة تصحيح الأخطاء.

٢. تشغيل التكوينات Launch configurations توفر هذه القائمة المنسدلة الوصول إلى تكوينات التشغيل، يتم عرض الخيار المحدد.

٣. افتح 'launch.json' يمكن استخدام هذا الزر (شكل ترس) لفتح ملف `launch.json` حيث يمكنك تحرير تكوين التشغيل إذا لزم الأمر.

٤. طرق العرض والمزيد من الإجراءات Views and More Actions يمكنك هذا الزر (علامة القطع an ellipsis) من إظهار/إخفاء مقاطع من لوحة التصحيح debug بالإضافة إلى لوحة وحدة تحكم DEBUG CONSOLE

شريط أدوات تتبع الأخطاء Debug toolbar

يوفر شريط أدوات تتبع الأخطاء عناصر تحكم في التنفيذ، أثناء تشغيل التطبيق.

١. إيقاف مؤقت/متابعة Pause/Continue يمكن استخدام هذا الزر لإيقاف التنفيذ مؤقتاً عند تشغيل التعليمات البرمجية، والمتابعة عند إيقاف تنفيذ التعليمات البرمجية مؤقتاً.

٢. خطوة إلى الأمام Step Over يمكن استخدام هذا الزر لتنفيذ الأسلوب التالي كأمر واحد، دون فحص أو اتباع خطوات مكوناته.

٣. خطوة إلى Step Into يمكن استخدام هذا الزر لإدخال الأسلوب التالي أو سطر التعليمات البرمجية، ومراقبة خطوات التنفيذ سطرًا تلو الآخر.

٣. خطوة إلى الخارج Step Out عند وجوده داخل أسلوب، يمكن استخدام هذا الزر للعودة إلى سياق التنفيذ السابق، عن طريق إكمال كافة الأسطر المتبقية من الأسلوب الحالي، كما لو كانت أمراً واحدًا.

٤. إعادة التشغيل Restart يمكن استخدام هذا الزر لإنهاء تنفيذ البرنامج الحالي، وبدء تصحيح الأخطاء مرة أخرى باستخدام التكوين الحالي.

٥. الإيقاف Stop يمكن استخدام هذا الزر لإنهاء تنفيذ البرنامج الحالي.

بالإضافة إلى ستة عناصر تحكم في التنفيذ، يوفر شريط أدوات تتبع الأخطاء "handle" على الجانب الأيسر، يمكّن المطور من تغيير موضع شريط الأدوات، والقائمة المنسدلة المزيد "More" على الجانب الأيمن، يمكّن المطور من قطع اتصال مصحح الأخطاء.

ملاحظة

يمكنك استخدام الإعداد `debug.toolBarLocation` للتحكم في موقع شريط أدوات التصحيح، يمكن أن يكون عائماً (الافتراضي) أو مثبتًا في طريقة العرض RUN AND DEBUG أو مخفياً، يمكن سحب شريط أدوات التصحيح العائم أفقياً، وأسفل إلى منطقة المحرر.

خلاصة

فيما يلي بعض الأشياء المهمة التي يجب تذكرها من هذا الدرس:

- يمكن استخدام واجهة مستخدم Visual Studio Code لتكوين جلسات تصحيح الأخطاء، وبدء تشغيلها، وإدارتها، يحتوي الملف `launch.json` على تكوينات التشغيل لتطبيقك.
- توفر قائمة التشغيل Run وصولاً سهلاً إلى أوامر التشغيل والتصحيح الشائعة المجمعة في ستة أقسام.
- توفر طريقة العرض RUN AND DEBUG الوصول إلى أدوات وقت التشغيل، بما في ذلك لوحة عناصر التحكم Run and Debug controls أقسام طريقة عرض RUN AND DEBUG هي VARIABLES, WATCH, CALL STACK, BREAKPOINTS
- يوفر شريط أدوات تتبع الأخطاء Debug toolbar عناصر تحكم في التنفيذ أثناء تشغيل تطبيقك، مثل pause/continue, step over, step into, step out, restart, stop
- يتم استخدام وحدة تحكم تتبع الأخطاء DEBUG CONSOLE لعرض الرسائل من مصحح الأخطاء، يمكن لوحدة تحكم DEBUG أيضاً عرض إخراج وحدة التحكم من تطبيقك.

اختبر معلوماتك

١. أي قسم من القائمة تشغيل **Run** يمكّن المطور من تحرير تكوينات التشغيل أو إضافتها؟

- قسم بدء وإيقاف start and stop التطبيقات.
- قسم تكوينات التشغيل Launch configurations
- قسم مصححات أخطاء التثبيت install debuggers

٢. أي قسم من طريقة عرض RUN AND DEBUG يستخدم لتعقب نقطة التنفيذ الحالية داخل التطبيق قيد التشغيل؟

- قسم المتغيرات VARIABLES
- قسم مكدس الاستدعاءات CALL STACK
- قسم المشاهدة WATCH

٣. ما هو الزر الموجود على شريط أدوات تتبع الأخطاء الذي يمكن استخدامه لاستئناف تنفيذ التعليمات البرمجية العادية؟

- الزر إيقاف مؤقت/متابعة Pause/Continue
- الزر خطوة إلى الأمام The Step Over
- زر إعادة التشغيل Restart

٤. أي قسم من قائمة Run يمكّن المطور من تعيين نقاط التوقف على أسطر التعليمات البرمجية؟

- قسم إدارة نقاط التوقف Manage Breakpoints
- قسم بدء وتشغيل التطبيق Start and stop applications
- قسم تعيين نقاط التوقف Set Breakpoints

٥. ما هي الإمكانية التي توفرها القائمة المنسدلة المزيد More على الجانب الأيمن من شريط أدوات التصحيح؟

- خيار لقطع اتصال مصحح الأخطاء.
- خيار لتحرير تكوينات التشغيل.
- خيار لإدارة نقاط التوقف.

٦. أي مما يلي يمكن استخدامه لتكوين جلسة تصحيح الأخطاء، وبدء تشغيلها؟

- شريط أدوات تتبع الأخطاء Debug toolbar
- قسم المتغيرات في طريقة عرض RUN AND DEBUG
- لوحة عناصر التحكم Run and Debug

راجع إجابتك

١ قسم تكوينات التشغيل Launch configurations

صحيح يوفر قسم التكوينات في قائمة Run الوصول إلى تحرير تكوينات التشغيل أو إضافتها في Visual Studio Code

٢ قسم مكدس الاستدعاءات CALL STACK

صحيح يتم استخدام قسم CALL STACK لتتبع نقطة التنفيذ الحالية داخل التطبيق قيد التشغيل، بدءاً من نقطة الإدخال الأولية إلى التطبيق

٣ الزر إيقاف مؤقت/متابعة Pause/Continue

صحيح يمكن استخدام الزر إيقاف مؤقت/متابعة على شريط أدوات التصحيح لإيقاف التنفيذ مؤقتاً عند تشغيل التعليمات البرمجية والمتابعة عند إيقاف تنفيذ التعليمات البرمجية مؤقتاً

٤ قسم تعيين نقاط التوقف Set Breakpoints

صحيح يتيح قسم Set Breakpoints في قائمة Run للمطور تعيين نقاط التوقف على أسطر التعليمات البرمجية

٥ خيار لقطع اتصال مصحح الأخطاء.

صحيح تتيح القائمة المنسدلة "المزيد" على الجانب الأيسر من شريط أدوات التصحيح للمطور قطع اتصال مصحح الأخطاء بالعملية الحالية

٦ لوحة عناصر التحكم Run and Debug

صحيح يتم استخدام لوحة عناصر التحكم Run and Debug لتكوين جلسة تصحيح الأخطاء وبدؤها، من طريقة عرض RUN AND DEBUG في Visual Studio Code

٣ تشغيل التعليمات البرمجية في بيئة التصحيح

تمكن واجهة مستخدم Visual Studio Code المطورين من تشغيل التعليمات البرمجية الخاصة بهم في بيئة تصحيح الأخطاء، يتم توفير دعم تصحيح الأخطاء بواسطة الملحقات، وبالنسبة لمطوري C# يتم توفير دعم مصحح الأخطاء من خلال نفس الملحق، الذي يوفر الدعم لتطوير التعليمات البرمجية وIntelliSense

مصحح الأخطاء وتفاعل التطبيق

يمكن استخدام مصحح الأخطاء لإيقاف تنفيذ التعليمات البرمجية مؤقتاً واستئنافه، وفحص حالة المتغير، وحتى تغيير القيم المعينة للمتغيرات في وقت التشغيل، قد تتساءل، كيف يمكن لمصحح الأخطاء التحكم في تطبيق قيد التشغيل وتعديله؟ الجواب المختصر هو أن مصحح الأخطاء لديه حق الوصول إلى بيئة وقت تشغيل التطبيق، والتعليمات البرمجية القابلة للتنفيذ.

ملاحظة

يعد تفاعل مصحح الأخطاء مع بيئة وقت التشغيل موضوعاً متقدماً، بالإضافة إلى ذلك، فهم كيفية عمل مصحح الأخطاء خلف الكواليس ليس شرطاً لاستخدام مصحح الأخطاء، ومع ذلك، قد يلبي الوصف التالي فضولك.

يستخدم مصحح أخطاء Visual Studio Code لتشغيل أحد التطبيقات والتفاعل معه .NET Core runtime. عند بدء تشغيل مصحح الأخطاء، فإنه ينشئ مثيلاً جديدًا لوقت التشغيل، ويشغل التطبيق داخل هذا المثيل، يتضمن وقت التشغيل واجهة برمجة التطبيقات (API) والتي يستخدمها مصحح الأخطاء لإرفاقها بعملية التشغيل الحالية (التطبيق الخاص بك).

بمجرد تشغيل تطبيقك وإرفاق مصحح الأخطاء، يتصل مصحح الأخطاء بعملية التشغيل الحالية باستخدام واجهات برمجة التطبيقات APIs لتصحيح

أخطاء وقت التشغيل .NET Core runtime's debugging. وبروتوكول تتبع الأخطاء القياسي standard debug protocol

يمكن أن يتفاعل مصحح الأخطاء مع العملية (التطبيق الذي يعمل ضمن مثيل وقت التشغيل .NET.) عن طريق تعيين نقاط التوقف، والتنقل خلال التعليمات البرمجية، وفحص المتغيرات، تمكنك واجهة مصحح الأخطاء الخاصة بـ Visual Studio Code من التنقل في التعليمات البرمجية المصدر، وعرض مكدسات الاستدعاءات وتقييم التعبيرات.

الطريقة الأكثر شيوعاً لتحديد جلسة تصحيح الأخطاء هي تكوين التشغيل في ملف launch.json هذه الطريقة هي الخيار الافتراضي الذي تم تمكينه بواسطة أدوات مصحح الأخطاء، على سبيل المثال، إذا قمت بإنشاء تطبيق وحدة تحكم C# وحددت بدء تصحيح الأخطاء Start Debugging من قائمة تشغيل Run يستخدم مصحح الأخطاء هذه الطريقة لتشغيل تطبيقك وإرفاقه والتفاعل معه.

إنشاء مشروع تعليمات برمجية جديدة

الخطوة الأولى في تعلم أدوات مصحح الأخطاء هي إنشاء مشروع تعليمات برمجية يمكنك تشغيله في مصحح الأخطاء.

١. افتح مثيلاً جديدًا من Visual Studio Code

٢. في القائمة ملف، حدد فتح مجلد.

٣. في مربع الحوار فتح مجلد، انتقل إلى مجلد سطح مكتب Windows

٤. في مربع الحوار فتح مجلد، حدد مجلد جديد.

٥. قم بتسمية المجلد الجديد Debug101 ثم حدد Select Folder

٦. في القائمة المحطة الطرفية، حدد محطة طرفية جديدة.

يمكن استخدام أمر .NET CLI. لإنشاء تطبيق وحدة تحكم جديد.

٧. في موجه أوامر لوحة TERMINAL أدخل الأمر التالي:

```
dotnet new console
```

٨. أغلق لوحة TERMINAL

فحص تكوينات التشغيل launch configurations لتصحيح الأخطاء

يستخدم Visual Studio Code ملف تكوين التشغيل launch configuration file لتحديد التطبيق الذي يتم تشغيله في بيئة تصحيح الأخطاء debug environment

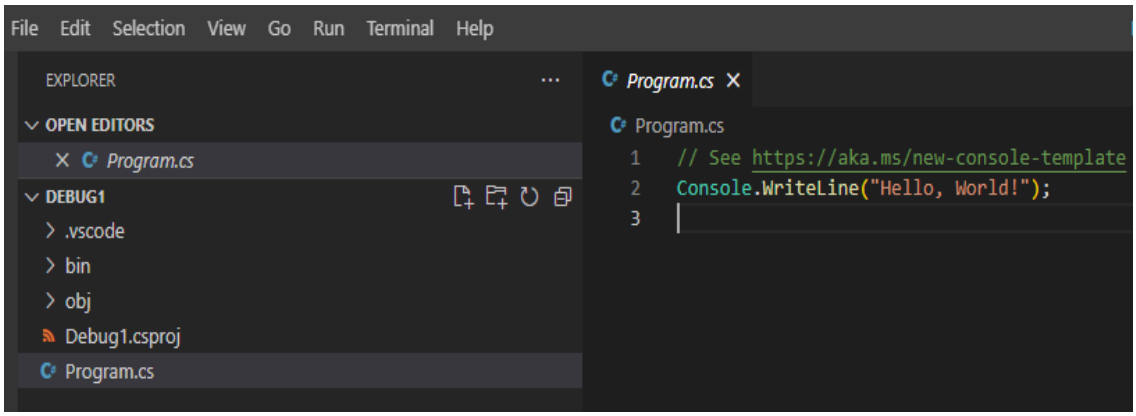
١. إذا لم يتضمن المجلد Debug101 ملف Debug101.sln فحدد **Program.cs** ثم تحقق من إنشاء ملف sln.

يؤدي فتح ملف التعليمات البرمجية C# إلى مطالبة بيئة التصحيح debug environment بالتحقق من ملفات المشروع، ملف sln. هو ملف حل يستخدمه Visual Studio لإدارة المشاريع، وعادةً ما يتم إنشاؤه تلقائيًا عند إنشاء مشروع جديد في Visual Studio Code يستخدم مصحح الأخطاء ملف sln. لتحديد المشروع الذي يجب تشغيله في بيئة التصحيح.

٢. في القائمة عرض View حدد لوحة الأوامر Command Palette

٣. في موجه الأوامر أدخل g :net. ثم حدد . NET: Generate Assets for Build and Debug

٤. لاحظ المجلد الجديد `.vscode` الذي تمت إضافته إلى مجلد المشروع



يحتوي المجلد `.vscode` على الملفات المستخدمة لتكوين بيئة تتبع الأخطاء.

٥. قم بتوسيع المجلد `.vscode` ثم حدد ملف launch.json

٦. خذ دقيقة لفحص ملف launch.json

يمكن أن يتضمن ملف تكوينات التشغيل launch configurations file تكوينات متعددة، يتضمن كل تكوين مجموعة من السمات المستخدمة لتعريف هذا التكوين.

٧. لاحظ أن سمة prelaunchTask تحدد مهمة بناء build

٨. في المجلد `.vscode` حدد tasks.json

٩. لاحظ أن ملف tasks.json يحتوي على مهمة البناء build task لمشروع التعليمات البرمجية.

١٠. أغلق ملفات launch.json وملفات tasks.json

يمكنك إلقاء نظرة فاحصة على سمات تكوين التشغيل لاحقاً في هذه الوحدة.

تشغيل التعليمات البرمجية من قائمة تشغيل Run

توفر القائمة Run في Visual Studio Code خيار تشغيل التعليمات البرمجية الخاصة بك مع مصحح الأخطاء أو بدونه.

١. افتح ملف Program.cs

٢. استبدل محتويات ملف Program.cs بالتعليمات البرمجية التالية:

```
/*
This code uses a names array and corresponding
methods to display
greeting messages
*/

string[] names = new string[] { "Sophia",
"Andrew", "AllGreetings" };

string messageText = "";

foreach (string name in names)
{
```

```
    if (name == "Sophia")
        messageText = SophiaMessage();
    else if (name == "Andrew")
        messageText = AndrewMessage();
    else if (name == "AllGreetings")
        messageText = SophiaMessage();
        messageText = messageText + "\n\r" +
AndrewMessage();

    Console.WriteLine(messageText + "\n\r");
}

bool pauseCode = true;
while (pauseCode == true);

static string SophiaMessage()
{
    return "Hello, my name is Sophia.";
}

static string AndrewMessage()
{
    return "Hi, my name is Andrew. Good to meet
you.";
}
```

٣. في القائمة ملف File حدد حفظ Save

٤. افتح قائمة تشغيل Run

لاحظ أن قائمة Run توفر خيارات لتشغيل التعليمات البرمجية الخاصة بك مع تصحيح الأخطاء أو بدونه.

٥. في قائمة Run حدد Run Without Debugging

٦. لاحظ أن لوحة DEBUG CONSOLE تعرض إخراج وحدة التحكم، وأن شريط أدوات تتبع الأخطاء Debug toolbar يعرض عناصر التحكم في التنفيذ.

يجب عرض لوحة DEBUG CONSOLE أسفل محرر التعليمات البرمجية، بشكل افتراضي، يقع شريط أدوات تتبع الأخطاء Debug toolbar (شريط الأدوات الصغير الذي يعرض عناصر تحكم تنفيذ التعليمات البرمجية) فوق محرر التعليمات البرمجية، ويتم توسيطه أفقياً في نافذة Visual Studio Code

٧. في شريط أدوات تتبع الأخطاء Debug toolbar حدد إيقاف Stop

بدء جلسة تصحيح الأخطاء debug session من قائمة تشغيل Run

تتضمن قائمة Run خيار بدء جلسة تصحيح الأخطاء.

١. في قائمة Run حدد Start Debugging

٢. خذ دقيقة لمراجعة الرسائل المعروضة في لوحة DEBUG CONSOLE

الإخراج من التطبيق الخاص بك هو نفسه عند تشغيله بدون تصحيح الأخطاء، ولكن يتم عرض الرسائل الأخرى المتعلقة بإعداد بيئة تصحيح الأخطاء.

٣. لاحظ الرسائل حول تحميل موارد .NET resources وتطبيق Debug101

تشير أول رسالتين بتحميل مكتبة .NET Core library. ثم تطبيق Debug101

```
Loaded 'C:\Program
Files\dotnet\shared\Microsoft.NETCore.App\7.0.4
\System.Private.CoreLib.dll'. Skipped loading
```

```
symbols. Module is optimized and the debugger
option 'Just My Code' is enabled.
Loaded
'C:\Users\someuser\Desktop\Debug101\bin\Debug\n
et7.0\Debug101.dll'. Symbols loaded.
```

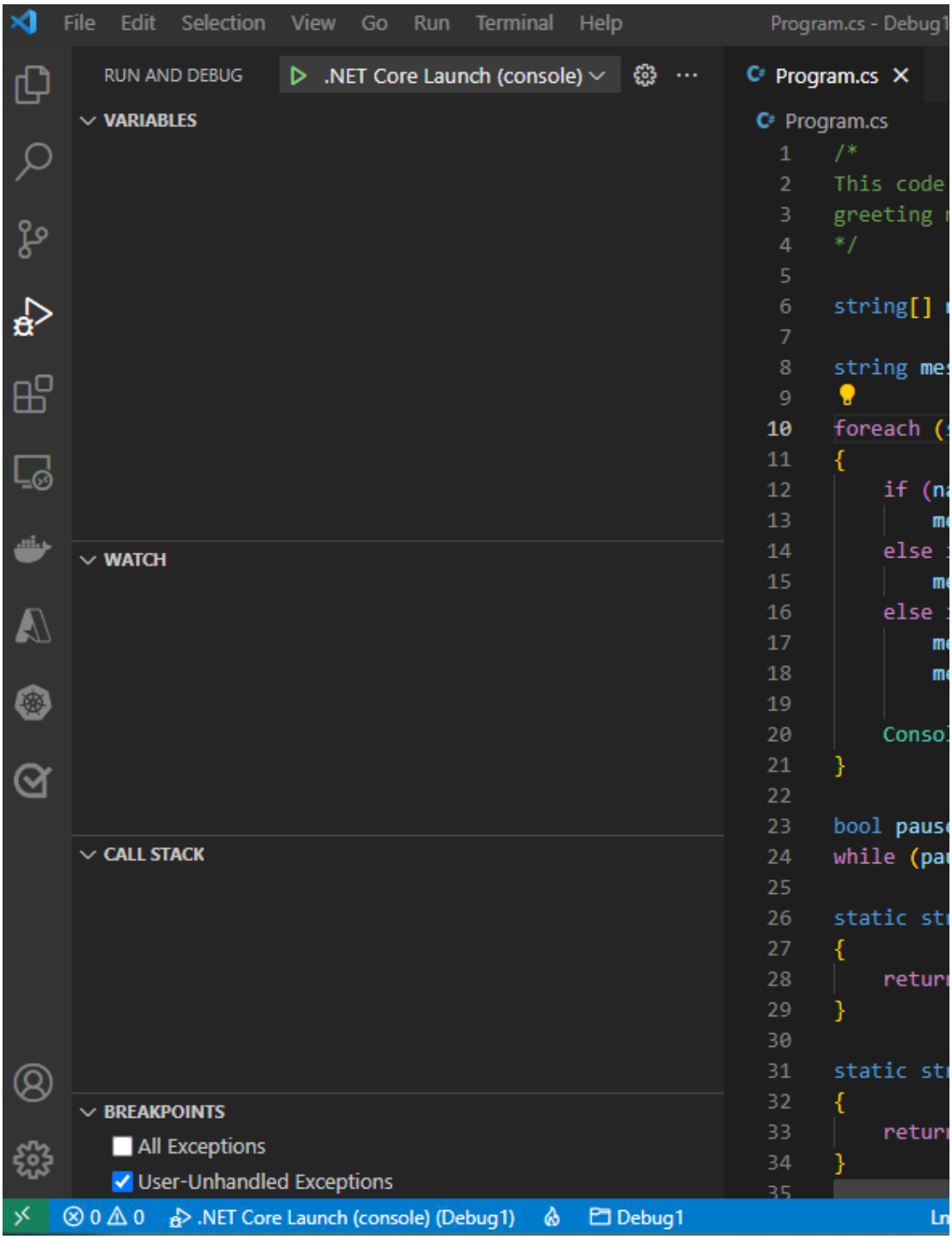
يستخدم مصحح الأخطاء نسخة خاصة من وقت تشغيل .NET runtime للتحكم في تنفيذ تطبيقك، وتقييم حالة التطبيق.

٤. في شريط أدوات تتبع الأخطاء Debug toolbar حدد إيقاف Stop

تشغيل التعليمات البرمجية من طريقة العرض/قائمة Run and Debug

تدعم طريقة العرض RUN AND DEBUG في Visual Studio Code تجربة تصحيح أخطاء ثرية.

١. قم بالتبديل إلى طريقة عرض RUN AND DEBUG



٢. في طريقة العرض RUN AND DEBUG حدد Start Debugging

زر بدء تصحيح الأخطاء Start Debugging هو السهم الأخضر على لوحة التحكم في أعلى طريقة العرض.

٣. لاحظ أن لوحة DEBUG CONSOLE تعرض نفس الرسائل حول تكوين مصحح الأخطاء، الذي تم عرضه عند بدء عملية تصحيح الأخطاء من القائمة تشغيل Run

٤. في شريط أدوات تتبع الأخطاء Debug toolbar حدد إيقاف Stop

فحص الإخراج output من التطبيق الخاص بك

١. قبل إغلاق لوحة DEBUG CONSOLE خذ دقيقة لمراجعة الإخراج الذي أنتجته التعليمات البرمجية الخاصة بك.

٢. لاحظ أن رسالة ترحيب Andrew تتكرر بشكل غير متوقع.

خلال الجزء المتبقي من هذه الوحدة، ستستخدم أدوات مصحح أخطاء Visual Studio Code للتحقيق في مشكلات الترميز.

خلاصة

فيما يلي بعض الأشياء المهمة التي يجب تذكرها من هذا الدرس:

- يستخدم مصحح أخطاء Visual Studio Code ل C# وقت تشغيل .NET Core. لتشغيل أحد التطبيقات والتفاعل معه.
- تحتوي قائمة Visual Studio Code Run على خيارات لبدء تشغيل تطبيق مع مصحح الأخطاء debugger المرفق وبدونه.
- يتضمن شريط أدوات تتبع الأخطاء Debug toolbar زرا لإيقاف عملية قيد التشغيل.
- تتضمن طريقة العرض RUN AND DEBUG خياراً لبدء تصحيح أخطاء debugging أحد التطبيقات.

٤ فحص خيارات تكوين نقطة التوقف breakpoint configuration

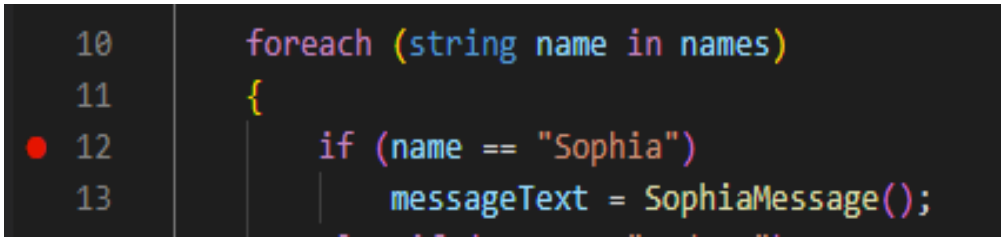
تستخدم مصححات الأخطاء Debuggers لمساعدتك في تحليل التعليمات البرمجية، ويمكن استخدامها للتحكم في تنفيذ وقت تشغيل البرنامج، عندما تبدأ تشغيل مصحح أخطاء Visual Studio Code فإنه يبدأ مباشرة في تنفيذ التعليمات البرمجية، نظراً لأن التعليمات البرمجية الخاصة بك تنفذ في أجزاء من الثانية، فإن تصحيح أخطاء التعليمات البرمجية الفعال يعتمد على قدرتك على إيقاف البرنامج مؤقتاً، على أي عبارة داخل تعليماتك البرمجية، يتم استخدام نقاط التوقف Breakpoints لتحديد مكان توقف تنفيذ التعليمات البرمجية مؤقتاً.

تعيين نقاط التوقف Set breakpoints

يوفر Visual Studio Code عدة طرق لتكوين نقاط التوقف في التعليمات البرمجية، على سبيل المثال:

- **محرر التعليمات البرمجية Code Editor**: يمكنك تعيين نقطة توقف في محرر التعليمات البرمجية Visual Studio بالنقر في العمود إلى يسار رقم السطر.
- **قائمة تشغيل Run menu**: يمكنك تبديل نقطة توقف إلى تشغيل/إيقاف تشغيل من قائمة تشغيل، يحدد سطر التعليمات البرمجية الحالي في المحرر مكان تطبيق إجراء تبديل نقطة التوقف Toggle Breakpoint

عند تعيين نقطة توقف، يتم عرض دائرة حمراء على يسار رقم السطر في المحرر، عند تشغيل التعليمات البرمجية في مصحح الأخطاء، يتوقف التنفيذ مؤقتاً عند نقطة التوقف.

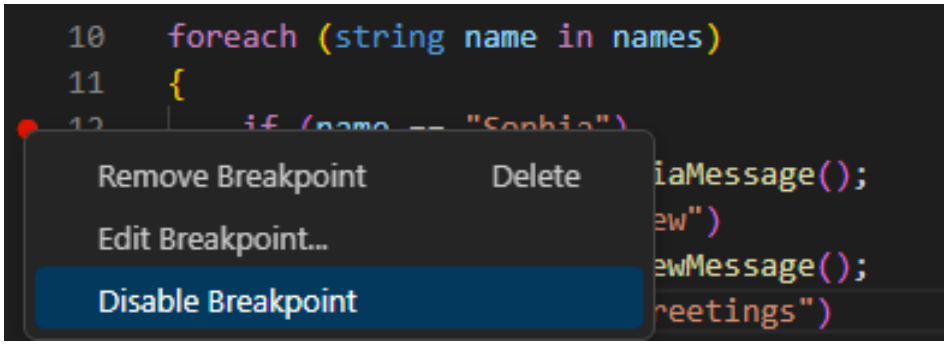


إزالة نقاط التوقف وتعطيلها وتمكينها Remove, disable, and enable breakpoints

بعد تعيين نقاط التوقف في تطبيقك، واستخدامها لعزل مشكلة، قد تحتاج إلى إزالة نقاط التوقف أو تعطيلها.

لإزالة نقطة توقف، كرر الإجراء المستخدم لتعيين نقطة توقف. على سبيل المثال، انقر فوق الدائرة الحمراء الموجودة على يسار رقم السطر أو استخدم خيار تبديل نقطة التوقف the toggle breakpoint في قائمة تشغيل Run

ماذا لو كنت تريد الاحتفاظ بموقع نقطة توقف، ولكنك لا تريد تشغيله أثناء جلسة تصحيح الأخطاء التالية؟ يمكّنك Visual Studio Code من تعطيل "disable" نقطة توقف بدلاً من إزالتها تماماً، لتعطيل نقطة توقف نشطة، انقر بزر الماوس الأيمن فوق النقطة الحمراء إلى يسار رقم الخط، ثم حدد تعطيل نقطة التوقف Disable Breakpoint من قائمة السياق.



عند تعطيل نقطة توقف، يتم تغيير النقطة الحمراء على يسار رقم الخط إلى نقطة رمادية.

ملاحظة

تتضمن قائمة السياق التي تظهر عند النقر بزر الماوس الأيمن فوق نقطة توقف خيارات إزالة نقطة التوقف (حذف) Remove Breakpoint (Delete) وتحرير نقطة التوقف Edit Breakpoint يتم فحص خيار تحرير نقطة التوقف Edit Breakpoint في قسم نقاط التوقف الشرطية Conditional breakpoints ونقاط السجل Logpoints لاحقاً في هذا الدرس.

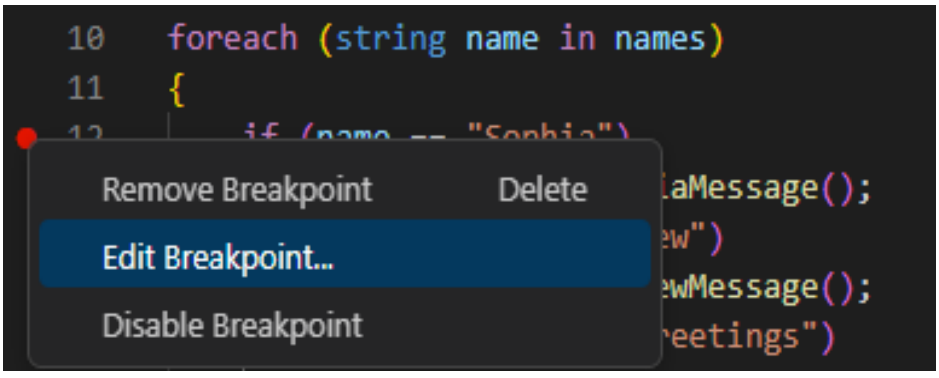
بالإضافة إلى إدارة نقاط التوقف الفردية في المحرر، توفر قائمة Run خيارات لتنفيذ العمليات المجمعة التي تعمل على جميع نقاط التوقف:

- **تمكين كافة نقاط التوقف Enable All Breakpoints**: استخدم هذا الخيار لتمكين كافة نقاط التوقف المعطلة.
- **تعطيل كافة نقاط التوقف Disable All Breakpoints**: استخدم هذا الخيار لتعطيل كافة نقاط التوقف.
- **إزالة كافة نقاط التوقف Remove All Breakpoints**: استخدم هذا الخيار لإزالة جميع نقاط التوقف (تتم إزالة كل من نقاط التوقف الممكّنة والمعطلة)

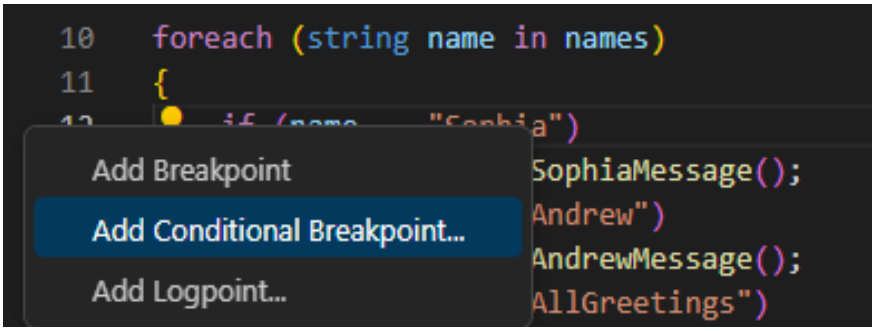
نقاط التوقف الشرطية Conditional breakpoints

نقطة التوقف الشرطية هي نوع خاص من نقاط التوقف التي يتم تشغيلها فقط عند استيفاء شرط محدد، على سبيل المثال، يمكنك إنشاء نقطة توقف شرطية توقف التنفيذ مؤقتاً عندما يكون المتغير المسمى `numItems` أكبر من 5

لقد رأيت بالفعل أن النقر بزر الماوس الأيمن فوق نقطة توقف يؤدي إلى فتح قائمة سياق تتضمن الخيار تحرير نقطة التوقف Edit Breakpoint يتيح لك تحديد تحرير نقطة التوقف، تغيير نقطة توقف قياسية إلى نقطة توقف شرطية.



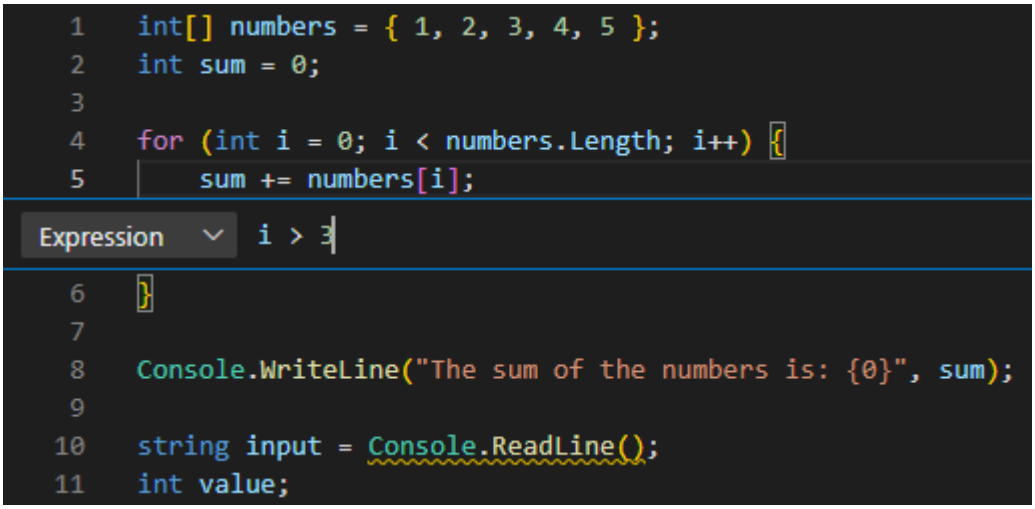
بالإضافة إلى تحرير نقطة توقف موجودة، يمكنك أيضًا تعيين نقطة توقف مشروطة مباشرةً، إذا قمت بالنقر بزر الماوس الأيمن (بدلاً من النقر بزر الماوس الأيسر) لتعيين نقطة توقف جديدة، فيمكنك اختيار إنشاء نقطة توقف مشروطة.



عند إنشاء نقطة توقف شرطية conditional breakpoint تحتاج إلى تحديد تعبير يمثل الشرط.

في كل مرة يواجه فيها مصحح الأخطاء نقطة التوقف الشرطية، فإنه يقيم التعبير، إذا تم تقييم التعبير على أنه true يتم تشغيل نقطة التوقف، ويتوقف التنفيذ مؤقتاً، إذا تم تقييم التعبير على أنه false يستمر التنفيذ كما لو لم يكن هناك نقطة توقف.

على سبيل المثال، افترض أنك بحاجة إلى تصحيح بعض التعليمات البرمجية الموجودة داخل كتلة التعليمات البرمجية for لقد لاحظت أن المشكلة التي تقوم بتصحيحها تحدث فقط بعد انتهاء الحلقة من عدة تكرارات، قررت أنك تريد تشغيل نقطة التوقف عندما يكون متغير التحكم في التكرار للحلقة i أكبر من ثلاثة، يمكنك إنشاء نقطة توقف شرطية وتحديد التعبير `i > 3`



عند تشغيل التعليمات البرمجية في مصحح الأخطاء، فإنه يتخطى نقطة التوقف الخاصة بك، حتى يتم تقييم التكرار عندما يكون `i > 3` صحيحاً، عندما يكون `i = 4` يتوقف التنفيذ مؤقتًا عند نقطة التوقف الشرطية.

دعم عدد ظهور نقاط التوقف Hit Count ونقاط السجل Logpoints

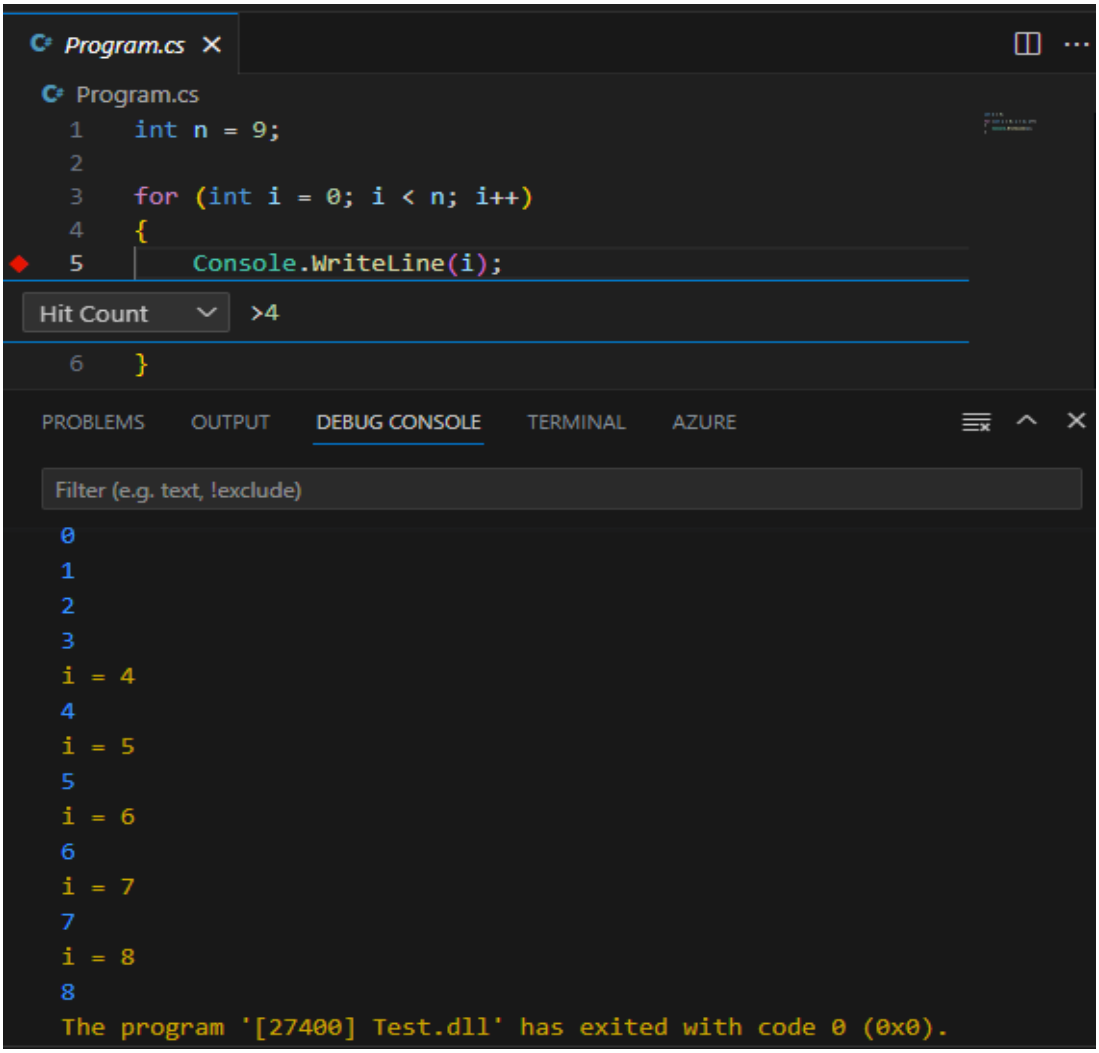
يدعم مصحح أخطاء Visual Studio Code ل C# أيضاً عدد ظهور نقاط التوقف ونقاط السجل Hit Count breakpoints and Logpoints

يمكن استخدام نقطة التوقف "عدد الظهور Hit Count" لتحديد عدد المرات التي يجب أن تتم فيها مواجهة نقطة التوقف قبل تنفيذ التوقف 'break' يمكنك تحديد قيمة عدد نتائج الظهور hit count value عند إنشاء نقطة توقف جديدة (باستخدام إجراء إضافة نقطة توقف مشروطة Conditional Breakpoint) أو عند تعديل قيمة موجودة (باستخدام إجراء تحرير الشرط Edit Condition) في كلتا الحالتين، يفتح مربع نص مضمّن مع قائمة منسدلة، حيث يمكنك إدخال قيمة عدد مرات الظهور/الوصول.

تعد نقطة التسجيل 'Logpoint' أحد أشكال نقطة التوقف التي لا تقطع "break"مصحح الأخطاء، ولكنها تقوم بدلاً من ذلك بتسجيل رسالة إلى وحدة التحكم، تعتبر نقاط السجل مفيدة بشكل خاص لإدخال التسجيل أثناء تصحيح أخطاء بيئات الإنتاج التي لا يمكن إيقافها مؤقتًا paused أو إيقافها stopped

يتم تمثيل نقطة السجل Logpoint بواسطة أيقونة على شكل "diamond" بدلاً من دائرة مملوءة، يتم تمثيل الدخول برمز على شكل "ماس" بدلاً من دائرة مملوءة، تعد رسائل السجل Log messages نصًا عاديًا، ولكن يمكن أن تتضمن تعبيرات ليتم تقييمها داخل الأقواس المتعرجة ('{}').

يمكن أن تتضمن نقاط السجل تعبيراً شرطياً و/أو عدد مرات الدخول conditional 'expression' and/or 'hit count' لمزيد من التحكم عند إنشاء رسائل التسجيل logging messages على سبيل المثال، يمكنك دمج رسالة Logpoint الخاصة بـ `i = {i}` مع شرط عدد نتائج الظهور Hit Count condition `>4` لإنشاء رسائل السجل log messages كما يلي:



خلاصة

فيما يلي بعض الأشياء المهمة التي يجب تذكرها من هذا الدرس:

- يتيح Visual Studio Code إمكانية تحديد نقاط التوقف في محرر التعليمات أو من قائمة Run يتم تمييز خطوط تعليمات نقطة التوقف، بنقطة حمراء على يسار رقم السطر.
- يمكن إزالة نقاط التوقف أو تعطيلها بنفس الخيارات المستخدمة لتحديدها، تتوفر العمليات المجمعة التي تؤثر على كافة نقاط التوقف في قائمة Run
- يمكن استخدام نقاط التوقف الشرطية لإيقاف التنفيذ مؤقتاً، عند استيفاء شرط محدد أو عند الوصول إلى عدد مرات الظهور.
- يمكن استخدام نقاط السجل لتسجيل المعلومات إلى المحطة الطرفية دون إيقاف التنفيذ مؤقتاً أو إدراج التعليمات البرمجية.

اختبر معلوماتك

١. أي من الخيارات التالية يمكن استخدامها لتعيين نقطة توقف في Visual Studio Code؟

- انقر بزر الماوس الأيسر في العمود الموجود يسار رقم سطر في محرر التعليمات البرمجية.
- حدد تبديل نقطة التوقف Toggle Breakpoint في قائمة Edit
- انقر بزر الماوس الأيمن في وسط سطر من التعليمات البرمجية، ثم حدد تبديل نقطة التوقف Toggle Breakpoint

٢. كيف يمكن للمطور تعطيل نقطة توقف في Visual Studio Code؟

- انقر فوق إزالة نقطة التوقف Remove Breakpoint من قائمة Run
- انقر بزر الماوس الأيمن فوق النقطة الحمراء يسار رقم السطر وحدد تعطيل نقطة التوقف Disable Breakpoint
- انقر بزر الماوس الأيسر فوق النقطة الحمراء يسار رقم السطر.

٣. ماذا يحدث عندما يقوم مطور بتعطيل نقطة توقف في Visual Studio Code؟

- تتم إزالة نقطة التوقف من التعليمات البرمجية بالكامل.
- يتم تخطي نقطة التوقف أثناء تصحيح الأخطاء.
- لن يتم تشغيل نقطة التوقف إلا إذا تم تغيير النقطة الموجودة على يسار رقم السطر إلى نقطة رمادية.

٤. ما هي نقطة التوقف الشرطية في Visual Studio Code؟

- نقطة توقف يتم تشغيلها فقط عند استيفاء شرط محدد.
- نقطة توقف يتم تشغيلها في كل مرة يتم فيها تشغيل التعليمات البرمجية.
- نقطة توقف مرئية فقط في المحرر ولا تؤثر على تصحيح الأخطاء.

٥. كيف يمكن للمطور إنشاء نقطة توقف شرطية في Visual Studio Code؟

- انقر بزر الماوس الأيسر في العمود الموجود يسار رقم سطر في محرر التعليمات البرمجية.
- حدد تبديل نقطة التوقف الشرطية Toggle Conditional Breakpoint في قائمة Run
- انقر بزر الماوس الأيمن فوق العمود الموجود يسار رقم السطر، ثم حدد إضافة نقطة توقف شرطية Add Conditional Breakpoint

راجع إجابتك

١ انقر بزر الماوس الأيسر في العمود الموجود يسار رقم سطر في محرر التعليمات البرمجية.

صحيح يمكن تعيين نقطة توقف عن طريق وضع مؤشر الماوس في العمود يسار رقم السطر ثم النقر بزر الماوس الأيسر.

٢ انقر بزر الماوس الأيمن فوق النقطة الحمراء يسار رقم السطر وحدد تعطيل نقطة التوقف Disable Breakpoint

صحيح يمكن تعطيل نقطة توقف بالنقر بزر الماوس الأيمن فوق النقطة الحمراء التي تمثل نقطة توقف ثم تحديد تعطيل نقطة التوقف.

٣ يتم تخطي نقطة التوقف أثناء تصحيح الأخطاء.

صحيح عند تعطيل نقطة توقف، يتم تخطيها أثناء تصحيح الأخطاء، في واجهة المستخدم، يتم تغيير النقطة الحمراء التي تمثل نقطة التوقف إلى نقطة رمادية.

٤ نقطة توقف يتم تشغيلها فقط عند استيفاء شرط محدد.

صحيح نقطة التوقف الشرطية هي نوع خاص من نقاط التوقف التي يتم تشغيلها فقط عند استيفاء شرط محدد.

٥ انقر بزر الماوس الأيمن فوق العمود الموجود يسار رقم السطر، ثم حدد إضافة نقطة توقف شرطية Add Conditional Breakpoint

صحيح يمكن للمطور إنشاء نقطة توقف شرطية بالنقر بزر الماوس الأيمن فوق العمود إلى يسار رقم السطر، ثم تحديد إضافة نقطة توقف شرطية.

٥ تمرين ـ تحديد نقاط التوقف Set breakpoints

يتم استخدام نقاط التوقف أثناء تنفيذ إيقاف عملية التصحيح مؤقتاً، يمكنك هذا من تعقب المتغيرات وفحص التسلسل الذي يتم فيه تنفيذ التعليمات البرمجية الخاصة بك، تعد نقاط التوقف طريقة رائعة لبدء عملية تصحيح الأخطاء.

تعيين نقطة توقف Set breakpoints

في وقت سابق من هذه الوحدة، أكملت تمرين حيث قمت بتشغيل تطبيق في مصحح الأخطاء، عرض التطبيق رسائل الترحيب " greeting messages" في لوحة DEBUG CONSOLE في نهاية التمرين، لاحظت أن التعليمات البرمجية تكرر تحية أندرو Andrew's greeting بطريقة غير متوقعة.

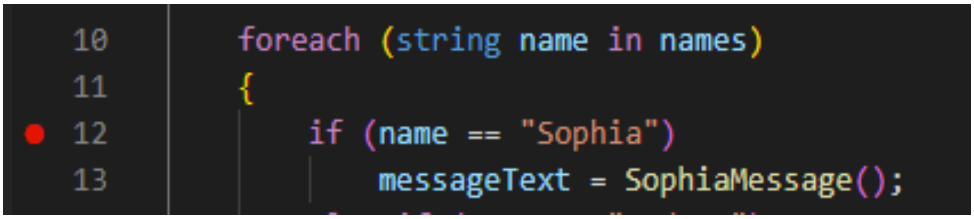
في هذا التمرين، ستستخدم نقطة توقف لمساعدتك في تحديد المشكلة.

١. تأكد من أن ملف Program.cs يحتوي على نموذج التعليمات البرمجية التالي:

```
/*
This code uses a names array and corresponding
methods to display
greeting messages
*/

string[] names = new string[] { "Sophia",
"Andrew", "AllGreetings" };

string messageText = "";

foreach (string name in names)
{
    if (name == "Sophia")
        messageText = SophiaMessage();
```

```
    else if (name == "Andrew")
        messageText = AndrewMessage();
    else if (name == "AllGreetings")
        messageText = SophiaMessage();
        messageText = messageText + "\n\r" +
AndrewMessage();

    Console.WriteLine(messageText + "\n\r");
}

bool pauseCode = true;
while (pauseCode == true);

static string SophiaMessage()
{
    return "Hello, my name is Sophia.";
}

static string AndrewMessage()
{
    return "Hi, my name is Andrew. Good to meet
you.";
}
```

٢. استخدم أدوات مصحح أخطاء debugger tools لتعيين نقطة توقف set a breakpoint على سطر التعليمات البرمجية الأول داخل الحلقة foreach



تلميح

أحد الخيارات السهلة للتبديل بين تشغيل/إيقاف on/off نقطة توقف هو تحديد (النقر بزر الماوس الأيسر) المنطقة الموجودة على يسار رقم السطر، يمكن أيضا تعيين نقاط التوقف باستخدام القائمة Run وباستخدام اختصارات لوحة المفاتيح

٣. في قائمة Run حدد Start Debugging

٤. لاحظ أن تنفيذ التعليمات البرمجية يتوقف مؤقتاً عند نقطة التوقف، وأن سطر التعليمات البرمجية الحالي مميز في المحرر.

٥. في شريط أدوات عناصر التحكم Debug controls في تتبع الأخطاء، حدد خطوة إلى الأمام Step Into

يمكنك تمرير مؤشر الماوس فوق الأزرار الموجودة على شريط أدوات عناصر التحكم في تتبع الأخطاء لعرض تسميات الأزرار.

٦. لاحظ أن تنفيذ التعليمات البرمجية يتقدم إلى سطر التعليمات التالي، ويتوقف مؤقتاً:

```
messageText = SophiaMessage();
```

يعين سطر التعليمات البرمجية هذا القيمة المُرجعة return value للأسلوب `SophiaMessage` إلى متغير السلسلة `messageText`

٧. خذ لحظة للنظر في سبب تحديد Step Into الذي أدى إلى هذه النتيجة.

- يتم استخدام الزر خطوة إلى الأمام Step Into للتقدم إلى العبارة القابلة للتنفيذ التالية.
- نظراً لأن العنصر الأول في المصفوفة `names` هو `Sophia` والعبارة if تتحقق من الاسم `Sophia` يتم تقييم التعبير إلى true وينتقل تنفيذ التعليمات البرمجية إلى كتلة العبارة if

٨. في شريط أدوات عناصر التحكم Debug controls في تتبع الأخطاء، حدد خطوة إلى الأمام Step Into

٩. لاحظ أن تنفيذ التعليمات البرمجية يتقدم إلى الأسلوب `SophiaMessage` ويتوقف مؤقتاً.

لقد تقدم الزر خطوة إلى الأمام Step Into إلى سطر التعليمات التالي القابل للتنفيذ، سطر التعليمات القابل للتنفيذ التالي ليس رقم السطر التالي في الملف، بل العبارة التالية في مسار التنفيذ، في هذه الحالة، العبارة القابلة للتنفيذ التالية هي نقطة الإدخال إلى الأسلوب `SophiaMessage`

١٠. في شريط أدوات عناصر التحكم في تتبع الأخطاء، حدد خطوة إلى الخارج Step Out

١١. لاحظ أن تنفيذ التعليمات البرمجية يعود إلى سطر التعليمات الذي يستدعي الأسلوب `SophiaMessage` ويتوقف مؤقتاً pauses

١٢. خذ لحظة للنظر في سبب قيام تحديد Step Out بإنتاج هذه النتيجة.

عند دخول أسلوب، يكمل الزر خطوة إلى الخارج Step Out الأسطر المتبقية من الأسلوب الحالي، ثم يعود إلى سياق التنفيذ الذي استدعى الأسلوب.

١٣. في شريط أدوات عناصر التحكم في تتبع الأخطاء، حدد خطوة إلى الأمام Step Into

١٤. لاحظ أن تنفيذ التعليمات يتقدم إلى سطر التعليمات التالي ويتوقف مؤقتاً:

```
messageText = messageText + "\n\r" + AndrewMessage();
```

١٥. خذ لحظة للنظر في سبب تقدم التنفيذ إلى سطر التعليمات البرمجية هذا.

على الرغم من أن المسافة البادئة للتعليمات البرمجية تعني أن سطر التعليمات البرمجية هذا جزء من كتلة العبارة else if إلا أنه ليست كذلك، قد يساعد استخدام الأقواس المتعرجة {} لتعريف كتل التعليمات لهذه البنية `if - else if` على تجنب هذا الخطأ، أثناء كتابة التعليمات البرمجية، ستتم إضافة رسالة أندرو إلى `messageText` كل مرة تتكرر فيها الحلقة.

التحقق من تحديثات التعليمات البرمجية

بمجرد عزل مشكلة في تعليماتك، يجب تحديث التعليمات ثم التحقق من إصلاح المشكلة.

١. في شريط أدوات عناصر التحكم Debug controls في تتبع الأخطاء، حدد إيقاف Stop

٢. خذ دقيقة لإصلاح منطق التعليمات البرمجية.

لديك بعض الخيارات لإصلاح المشكلة المحددة في التعليمات البرمجية الخاصة بك، على سبيل المثال:

- يمكنك الاحتفاظ بأسطر التعليمات البرمجية الموجودة وإضافة أقواس متعرجة {} إلى بنية if لكل كتلة تعليمات برمجية.
- يمكنك دمج سطري التعليمات البرمجية اللذين يتبعان العبارة النهائية else if وتشكيل عبارة واحدة كما يلي:

```
else if (name == "AllGreetings")
    messageText = SophiaMessage() + "\n\r" +
AndrewMessage();
```

وفي كلتا الحالتين، يجب أن تتضمن التعليمات البرمجية المحدثة استدعاء `AndrewMessage` داخل كتلة التعليمات البرمجية عندما يكون `name == "AllGreetings"`

٣. في القائمة ملف File حدد حفظ Save

٤. استخدم أدوات واجهة مستخدم مصحح الأخطاء debugger UI لمسح نقطة التوقف التي قمت بتعيينها مسبقاً.

٥. في قائمة Run حدد Start Debugging

٦. تحقق من أن التعليمات البرمجية الخاصة بك تنتج الآن النتائج المتوقعة

```
Hello, my name is Sophia.
```

```
Hi, my name is Andrew. Good to meet you.
```

```
Hello, my name is Sophia.
Hi, my name is Andrew. Good to meet you.
```

٧. في شريط أدوات عناصر التحكم في تتبع الأخطاء، حدد إيقاف.

تهانينا! لقد استخدمت مصحح أخطاء Visual Studio Code بنجاح لمساعدتك في عزل مشكلة منطقية وتصحيحها.

خلاصة

فيما يلي بعض الأشياء المهمة التي يجب تذكرها من هذا الدرس:

- استخدم نقاط التوقف لإيقاف تنفيذ التعليمات مؤقتاً breakpoints to pause code أثناء جلسة تصحيح الأخطاء.
- استخدم Step Into من شريط أدوات عناصر تحكم تتبع الأخطاء، لمراقبة سطر التعليمات التالي القابل للتنفيذ.
- استخدم Step Out من شريط أدوات عناصر تحكم تتبع الأخطاء، للتقدم عبر الأسلوب الحالي، والعودة إلى سطر التعليمات البرمجية الذي يستدعي الأسلوب.

٦ فحص ملف تكوينات التشغيل launch configurations file

لقد رأيت بالفعل أن Visual Studio Code يستخدم ملف launch.json لتكوين مصحح الأخطاء، إذا كنت تقوم بإنشاء تطبيق وحدة تحكم C# بسيط، فمن المحتمل أن يقوم Visual Studio Code بإنشاء ملف launch.json يحتوي على جميع المعلومات التي تحتاجها لتصحيح التعليمات البرمجية بنجاح، ومع ذلك، هناك حالات تحتاج فيها إلى تعديل تكوين تشغيل modify a launch configuration لذلك من المهم فهم سمات تكوين التشغيل.

سمات تكوين التشغيل Attributes of a launch configuration

يتضمن ملف launch.json تكوين تشغيل واحد أو أكثر في القائمة `configurations list` تستخدم تكوينات التشغيل السمات لدعم سيناريوهات تصحيح الأخطاء المختلفة، السمات التالية إلزامية لكل تكوين تشغيل:

- name: الاسم المألوف "السهل" للقارئ المعين لتكوين التشغيل.
- type: نوع مصحح الأخطاء الذي يجب استخدامه لتكوين التشغيل.
- request: نوع الطلب لتكوين التشغيل.

launch.json

.vscode > launch.json > ...

```jsonc
{
    "version": "0.2.0",
    "configurations": [
        {
            // Use IntelliSense to find out which attributes exist for C# debugging
            // Use hover for the description of the existing attributes
            // For further information visit https://github.com/OmniSharp/omnisharp-vscode/blob/master/debugger-launchjson.md
            "name": ".NET Core Launch (console)",
            "type": "coreclr",
            "request": "launch",
            "preLaunchTask": "build",
            // If you have changed target frameworks, make sure to update the program path.
            "program": "${workspaceFolder}/bin/Debug/net7.0/Debug101.dll",
            "args": [],
            "cwd": "${workspaceFolder}",
            // For more information about the 'console' field, see https://aka.ms/VSCode-CS-LaunchJson-Console
            "console": "internalConsole",
            "stopAtEntry": false
        },
        {
            "name": ".NET Core Attach",
            "type": "coreclr",
            "request": "attach"
        }
    ]
}
```

يحدد هذا القسم بعض السمات التي قد تواجهها.

الاسم Name

تحدد السمة `name` اسم العرض لتكوين التشغيل، تظهر القيمة المعينة ل name في القائمة المنسدلة تكوينات التشغيل launch configurations (في لوحة عناصر التحكم في أعلى طريقة عرض RUN AND DEBUG)

نوع Type

تحدد السمة `type` نوع مصحح الأخطاء لاستخدامه لتكوين التشغيل، تحدد قيمة نوع `codeclr` مصحح الأخطاء لتطبيقات .NET Core and .NET 5+. (بما في ذلك تطبيقات C#)

الطلب Request

تحدد سمة `request` الطلب نوع الطلب لتكوين التشغيل، حاليًا، يتم دعم إطلاق القيم وإرفاقها `launch` and `attach`

مهمة تشغيل PreLaunchTask

تحدد السمة `preLaunchTask` مهمة ليتم تشغيلها قبل تصحيح أخطاء البرنامج، يمكن العثور على المهمة نفسها في ملف tasks.json الموجود في المجلد `.vscode` مع ملف launch.json يؤدي تحديد مهمة التشغيل المسبق `build` إلى تشغيل أمر `dotnet build` قبل تشغيل التطبيق.

البرنامج Program

يتم تعيين سمة `program` على مسار التطبيق dll أو مضيف .NET Core القابل للتنفيذ لبدء التشغيل.

عادة ما تأخذ هذه الخاصية النموذج:

```
${workspaceFolder}/bin/Debug/<target-framework>/<project-name.dll>
```

- `<target-framework>` هو الإطار الذي تم إنشاء مشروع تصحيح الأخطاء من أجله، يتم العثور على هذه القيمة عادةً في ملف المشروع كخاصية TargetFramework
- `<project-name.dll>` هو اسم ملف dll لمخرجات بناء المشروع الذي تم تصحيحه، عادة ما تكون هذه الخاصية هي نفس اسم ملف المشروع ولكن مع ملحق `.dll`.

على سبيل المثال:

```
${workspaceFolder}/bin/Debug/net7.0/Debug101.dll
```

ملاحظة

يشير ملحق **`.dll`** إلى أن هذا الملف عبارة عن ملف مكتبة ارتباط ديناميكي (dll) إذا كان اسم مشروعك Debug101 فسيتم إنشاء ملف باسم Debug101.dll عندما تقوم مهمة بناء بتجميع برنامجك باستخدام ملفات Program.cs وDebug101.csproj يمكنك العثور على الملف Debug101.dll في قائمة EXPLORER عن طريق توسيع المجلدين "bin" و"Debug" ثم فتح مجلد يمثل إطار عمل .NET. الذي يستخدمه مشروع التعليمات البرمجية، مثل "net7.0" تم تحديد إصدار .NET Framework في ملف `.csproj` الخاص بك.

Cwd

تحدد السمة `cwd` دليل العمل للعملية المستهدفة.

وسائط Args

تحدد السمة `args` الوسيطات التي يتم تمريرها إلى البرنامج عند التشغيل. لا توجد وسيطات بشكل افتراضي.

وحدة تحكم Console

تحدد السمة `console` نوع وحدة التحكم المستخدمة عند تشغيل التطبيق، الخيارات هي `internalConsole`, `integratedTerminal`, and `externalTerminal` الإعداد الافتراضي هو `internalConsole` يتم تعريف أنواع وحدة التحكم على النحو التالي:

- يتوافق الإعداد `internalConsole` مع لوحة DEBUG CONSOLE في منطقة اللوحات الموجودة أسفل محرر Visual Studio Code
- يتوافق الإعداد `integratedTerminal` مع لوحة OUTPUT في منطقة اللوحات الموجودة أسفل محرر Visual Studio Code
- يتوافق الإعداد `externalTerminal` مع نافذة محطة طرفية خارجية، يعد تطبيق موجه الأوامر الذي يأتي مع Windows مثالاً على النافذة الطرفية.

هام

لا تدعم لوحة DEBUG CONSOLE إدخال وحدة التحكم، على سبيل المثال، لا يمكن استخدام وحدة DEBUG CONSOLE إذا كان التطبيق يتضمن عبارة Console.ReadLine() عندما تعمل على تطبيق C# console application الذي يقرأ مدخلات المستخدم، يجب تعيين إعداد وحدة التحكم console إما على integratedTerminal or externalTerminal يمكن لتطبيقات وحدة التحكم التي تكتب إلى console ولكن لا تقرأ الإدخال من console استخدام أي من الإعدادات console الثلاثة.

التوقف عند الإدخال Stop at Entry

إذا كنت بحاجة إلى التوقف عند نقطة إدخال الهدف، يمكنك اختياريًا تعيين `stopAtEntry` إلى `true`

تحرير تكوين التشغيل launch configuration

هناك الكثير من السيناريوهات التي قد تحتاج فيها إلى تخصيص ملف تكوين التشغيل، تتضمن العديد من هذه السيناريوهات سيناريوهات مشروع متقدمة أو معقدة، تركز هذه الوحدة على سيناريوهين بسيطين عند الحاجة إلى تحديث ملف تكوين الإطلاق:

- يقرأ تطبيق وحدة تحكم C# الإدخال من وحدة التحكم.
- تتضمن مساحة عمل المشروع أكثر من تطبيق واحد.

تحديث تكوين التشغيل لاستيعاب إدخال وحدة التحكم

كما قرأت سابقاً، لا تدعم لوحة DEBUG CONSOLE إدخال وحدة التحكم console input إذا كنت تقوم بتصحيح تطبيق وحدة تحكم يعتمد على إدخال المستخدم، فستحتاج إلى تحديث السمة console في تكوين التشغيل المقترن.

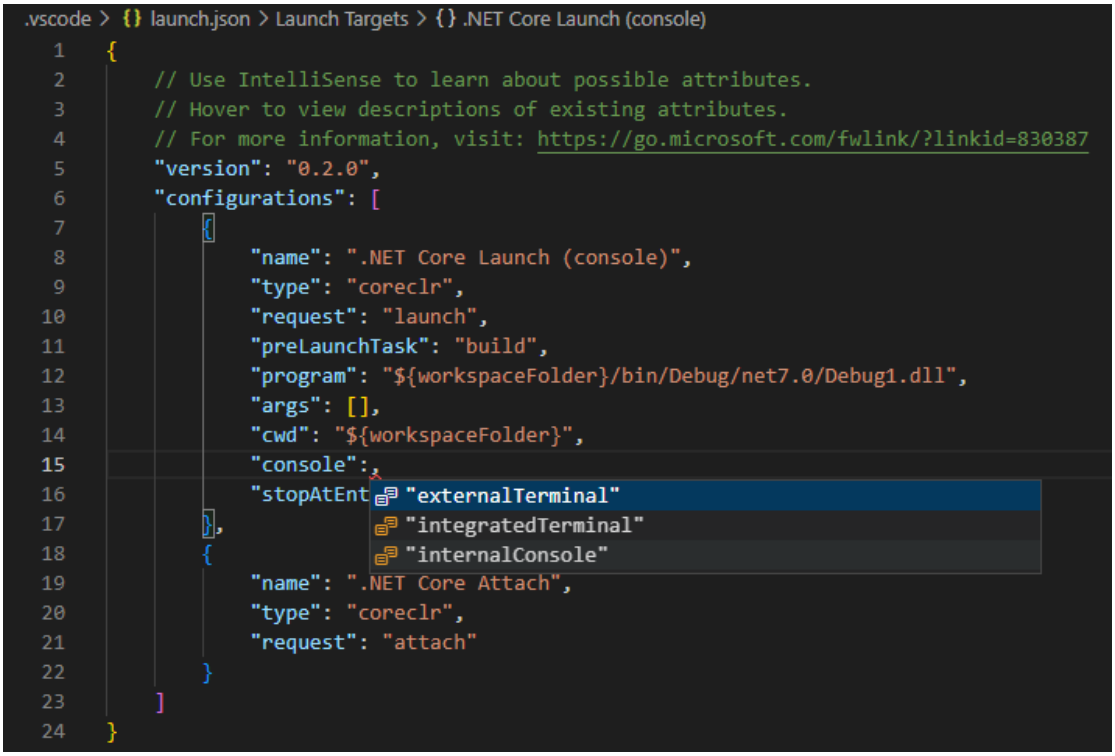
لتحرير السمة console:

١. افتح ملف launch.json في محرر Visual Studio Code

٢. حدد موقع سمة وحدة التحكم **console** attribute

٣. حدد النقطتين (:) والقيمة المعينة، ثم أدخل حرف نقطتين (:)

٤. لاحظ عند الكتابة فوق المعلومات الموجودة باستخدام علامة النقطتين، يعرض Visual Studio Code IntelliSense الخيارات الثلاثة في قائمة منسدلة.



٥. حدد إما integratedTerminal أو externalTerminal
٦. احفظ ملف launch.json

تحديث تكوين التشغيل لاستيعاب تطبيقات متعددة

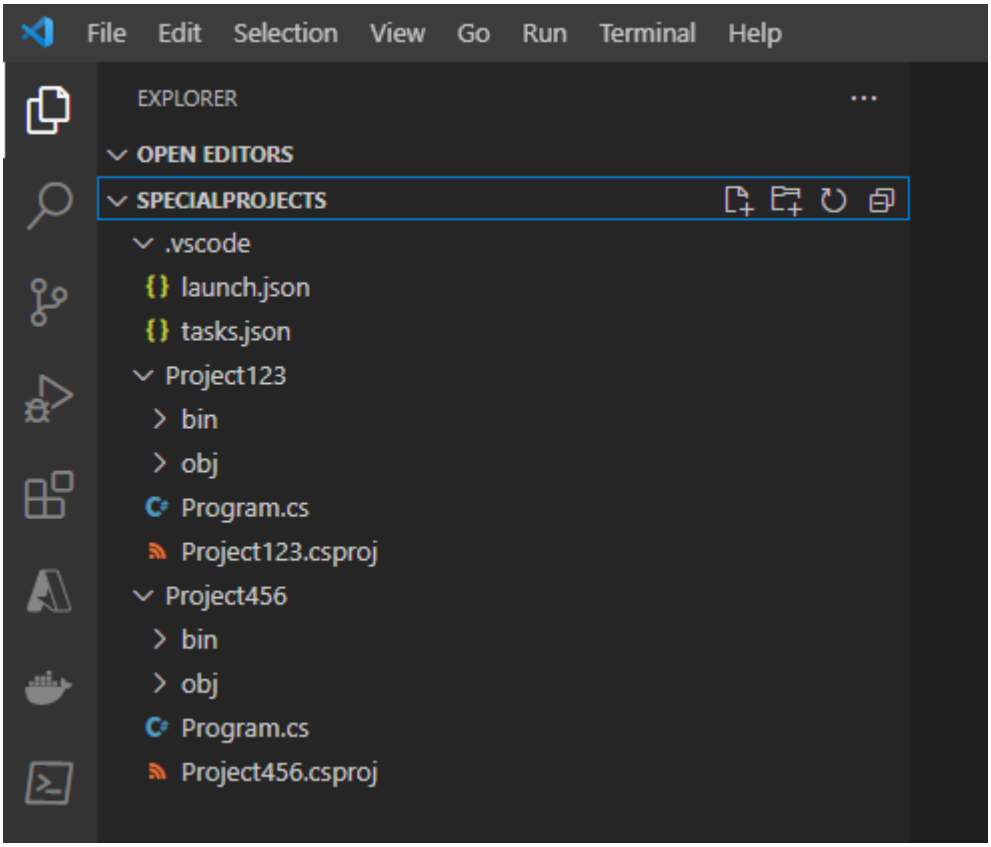
إذا كانت مساحة العمل الخاصة بك تحتوي على مشروع واحد فقط قابل للتشغيل، فسيقوم ملحق C# تلقائيًا بإنشاء ملف launch.json إذا كان لديك أكثر من مشروع واحد قابل للتشغيل، فستحتاج إلى تعديل ملف launch.json يدويًا، يقوم Visual Studio Code بإنشاء ملف launch.json باستخدام القالب الأساسي الذي يمكنك تحديثه.

في هذا السيناريو، يمكنك إنشاء تكوينات منفصلة لكل تطبيق تريد تصحيحه، يمكن إنشاء مهام التشغيل المسبق، مثل مهمة البناء، في ملف المهام tasks.json

لنفترض أنك تعمل على مشروع ترميز يتضمن العديد من تطبيقات وحدة التحكم، مجلد المشروع الجذر SpecialProjects هو مجلد مساحة العمل الذي تفتحه في Visual Studio Code عند العمل على التعليمات البرمجية، لديك تطبيقان تقوم بتطويرهما Project123 وProject456 يمكنك استخدام قائمة RUN AND DEBUG لتصحيح أخطاء التطبيقات، تريد تحديد التطبيق الذي تقوم بتصحيح أخطائه من واجهة المستخدم، أنت تريد أيضًا تجميع أي تحديثات للتعليمات البرمجية المحفوظة قبل إرفاق مصحح الأخطاء بتطبيقك.

يمكنك تحقيق متطلبات هذا السيناريو عن طريق تحديث ملفات launch.json and tasks.json

تظهر لقطة الشاشة التالية قائمة EXPLORER وبنية المجلد التي تحتوي على Project123 and Project456



لاحظ أن المجلد `.vscode` الذي يحتوي على ملفات launch.json and tasks.json مقترن بمجلد مساحة العمل SpecialProjects وليس مجلدات المشروع الفردية.

يوضح المثال التالي كيف يمكنك تكوين ملف launch.json لتضمين تكوينات لكل من تطبيقي "Project123" and "Project456"

```
"version": "0.2.0",
"configurations": [
    {
        "name": "Launch Project123",
        "type": "coreclr",
        "request": "launch",
        "preLaunchTask": "buildProject123",
        "program":
"${workspaceFolder}/Project123/bin/Debug/net7.0
/Project123.dll",
        "args": [],
```

```
            "cwd": "${workspaceFolder}/Project123",
            "console": "internalConsole",
            "stopAtEntry": false
        },
        {
            "name": "Launch Project456",
            "type": "coreclr",
            "request": "launch",
            "preLaunchTask": "buildProject456",
            "program":
"${workspaceFolder}/Project456/bin/Debug/net7.0
/Project456.dll",
            "args": [],
            "cwd": "${workspaceFolder}/Project456",
            "console": "internalConsole",
            "stopAtEntry": false
        }
]
```

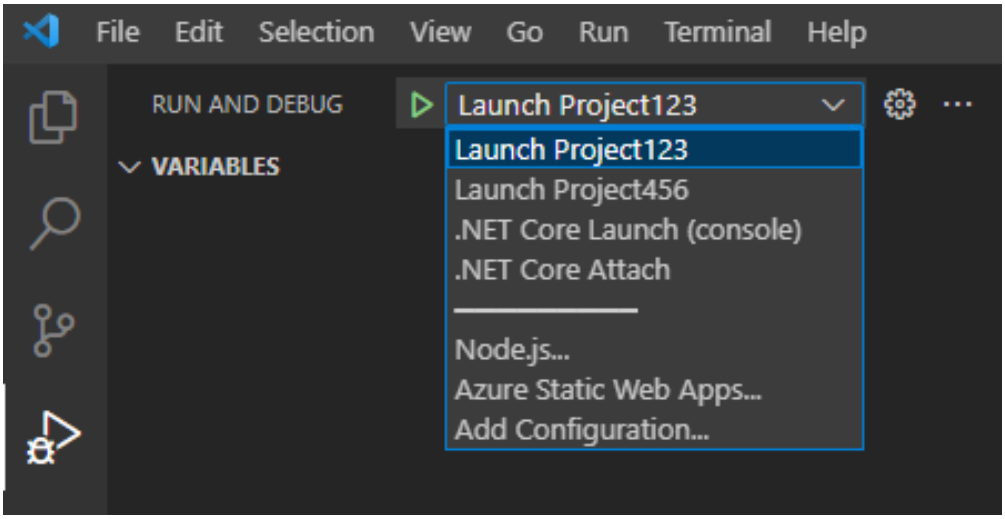
لاحظ أن حقول name, preLaunchTask, and program كلها مكونة لتطبيق معين.

تحدد سمة name خيار التشغيل القابل للتحديد الذي يتم عرضه في واجهة مستخدم، قائمة RUN AND DEBUG وتحدد سمة البرنامج المسار إلى التطبيق الخاص بك، يتم استخدام السمة preLaunchTask لتحديد اسم المهمة التي تم تنفيذها قبل تشغيل مصحح الأخطاء، يحتوي ملف tasks.json على المهام المسماة والمعلومات المطلوبة لإكمال المهمة.

يوضح المثال التالي كيف يمكنك تكوين ملف tasks.json في هذه الحالة، تحدد المهام المسماة عمليات البناء الخاصة بتطبيقات "Project123" و"Project456" تضمن مهمة البناء تجميع أي عمليات تحرير محفوظة وتمثيلها في ملف dll. المقابل المرفق بمصحح الأخطاء.

```
"version": "2.0.0",
"tasks": [
    {
        "label": "buildProject123",
        "command": "dotnet",
        "type": "process",
        "args": [
            "build",

"${workspaceFolder}/Project123/Project123.csproj",
            "/property:GenerateFullPaths=true",

"/consoleloggerparameters:NoSummary"
        ],
        "problemMatcher": "$msCompile"
    },
    {
        "label": "buildProject456",
        "command": "dotnet",
        "type": "process",
        "args": [
            "build",

"${workspaceFolder}/Project456/Project456.csproj",
            "/property:GenerateFullPaths=true",

"/consoleloggerparameters:NoSummary"
        ],
        "problemMatcher": "$msCompile"
    }
]
```

مع وجود تحديثات ملفات launch.json and tasks.json في مكانها، تعرض قائمة RUN AND DEBUG خيارات التشغيل لتصحيح أخطاء تطبيق Project123 أو Project456 تظهر لقطة الشاشة التالية أسماء تكوينات التشغيل المعروضة في القائمة المنسدلة لتكوين التشغيل:



خلاصة

فيما يلي أمران مهمان يجب تذكرهما من هذا الدرس:

- تُستخدم تكوينات التشغيل لتحديد سمات مثل `name`, `type`, `request`, `preLaunchTask`, `program`, and `console`
- يمكن للمطورين تحرير تكوين تشغيل لاستيعاب متطلبات المشروع.

راجع معلوماتك

١. يقوم المطور بتحديث تكوين التشغيل ما هي سمة name المستخدمة؟

- لتحديد نوع مصحح الأخطاء المستخدم في تكوين التشغيل.
- لتحديد نوع الطلب لتكوين التشغيل.
- لتحديد اسم العرض للتكوين.

٢. يعمل مطور على تكوين تشغيل ما هي السمة preLaunchTask المستخدمة؟

- لتحديد مسار dll للتطبيق أو مضيف NET Core. القابل للتنفيذ لتشغيله.
- لتحديد مهمة لتشغيلها قبل تصحيح أخطاء البرنامج.
- لتحديد نوع وحدة التحكم المستخدمة عند تشغيل التطبيق.

٣. ما هي سمة console لتكوين التشغيل المستخدمة؟

- تحديد دليل العمل للعملية الهدف.
- تحديد نوع مصحح الأخطاء الذي يجب استخدامه لتكوين التشغيل هذا.
- تحديد نوع وحدة التحكم المستخدمة عند تشغيل التطبيق.

راجع إجابتك

١ لتحديد اسم العرض للتكوين.

صحيح تحدد سمة name اسم العرض للتكوين. تظهر القيمة المعينة للاسم في لوحة عناصر التحكم أعلى قائمة RUN AND DEBUG

٢ لتحديد مهمة لتشغيلها قبل تصحيح أخطاء البرنامج.

صحيح تحدد السمة preLaunchTask مهمة لتشغيلها قبل تصحيح أخطاء البرنامج، يمكن العثور على المهمة نفسها في ملف tasks.json الموجود في المجلد vscode. مع ملف launch.json يؤدي تحديد مهمة البدء المسبق للبناء إلى تشغيل أمر dotnet build قبل بدء تشغيل التطبيق.

٣ تحديد نوع وحدة التحكم المستخدمة عند تشغيل التطبيق.

صحيح تحدد سمة Console نوع وحدة التحكم المستخدمة عند تشغيل التطبيق، الخيارات internalConsole, integratedTerminal, and externalTerminal الإعداد الافتراضي هو InternalConsole

٧ تكوين نقاط التوقف الشرطية في C#

يدعم مصحح أخطاء Visual Studio Code ل C# خيار تكوين نقطة توقف يتم تشغيلها فقط إذا تم استيفاء شرط، يسمى هذا النوع من نقاط التوقف نقطة توقف شرطية conditional breakpoint يمكن تكوين نقاط التوقف الشرطية مباشرة أو عن طريق تحرير نقطة توقف موجودة.

ملاحظة

يدعم Visual Studio Code أيضا نقطة توقف شرطية يتم تشغيلها استنادًا إلى عدد المرات التي تم فيها الظهور/الوصول إلى نقطة التوقف "hit"

لنفترض أنك تقوم بتصحيح أخطاء تطبيق يعالج معلومات المنتج في مصفوفة سلسلة متعدد الأبعاد، تتضمن المصفوفة آلاف نقاط البيانات، يبدو أن المشكلة التي تقوم بتصحيحها تحدث للمنتجات التي تم وضع علامة عليها `new` تعالج التعليمات البرمجية المصفوفة داخل `for` تحتاج إلى تعيين نقطة توقف داخل الحلقة، ولكنك تريد التوقف مؤقتًا فقط عندما تكون المنتجات `new`

استخدام نقطة توقف قياسية لفحص تطبيق معالجة البيانات

١. استبدل محتويات ملف Program.cs بالتعليمات البرمجية التالية:

```
int productCount = 2000;
string[,] products = new string[productCount, 2];

LoadProducts(products, productCount);

for (int i = 0; i < productCount; i++)
{
    string result;
    result = Process1(products, i);
```

```
    if (result != "obsolete")
    {
        result = Process2(products, i);
    }
}

bool pauseCode = true;
while (pauseCode == true) ;
```

تستخدم هذه التعليمة البرمجية أسلوب `LoadProducts` لتحميل البيانات في مصفوفة `products` بعد تحميل البيانات، تتكرر التعليمات البرمجية من خلال المصفوفة وتستدعي الأساليب المسماة Process1 and Process2

٢. لإنشاء بيانات لعمليات المحاكاة، أضف الأسلوب التالي إلى نهاية ملف Program.cs الخاص بك:

```
static void LoadProducts(string[,] products,
int productCount)
{
    Random rand = new Random();

    for (int i = 0; i < productCount; i++)
    {
        int num1 = rand.Next(1, 10000) + 10000;
        int num2 = rand.Next(1, 101);

        string prodID = num1.ToString();

        if (num2 < 91)
        {
            products[i, 1] = "existing";
        }
```

```
        else if (num2 == 91)
        {
            products[i, 1] = "new";
            prodID = prodID + "-n";
        }
        else
        {
            products[i, 1] = "obsolete";
            prodID = prodID + "-0";
        }

        products[i, 0] = prodID;
    }
}
```

ينشئ الأسلوب `LoadProducts` بإنشاء ٢٠٠٠ معرف منتج عشوائي وتعيين قيمة `existing`، `new`، `obsolete` لحقل وصف المنتج، هناك احتمال بنسبة ١٪ تقريبًا أن يتم وضع علامة على المنتجات بأنها `new`

لمحاكاة معالجة البيانات، أضف الطرق التالية إلى نهاية ملف Program.cs:

```
static string Process1(string[,] products, int
item)
{
    Console.WriteLine($"Process1 message -
working on {products[item, 1]} product");

    return products[item, 1];
}

static string Process2(string[,] products, int item)
{
```

```
    Console.WriteLine($"Process2 message -
working on product ID #: {products[item, 0]}");
    if (products[item, 1] == "new")
        Process3(products, item);

    return "continue";
}

static void Process3(string[,] products, int item)
{
    Console.WriteLine($"Process3 message -
processing product information for 'new'
product");
}
```

تعرض الأساليب Process1 and Process2 رسائل التقدم وترجع سلسلة return a string

لاحظ أن الأسلوب `Process2` يستدعي `Process3` إذا كان المنتج هو `new`

٤. في قائمة **File** Visual Studio Code حدد **Save**

٥. بالقرب من أعلى ملف Program.cs قم بتعيين نقطة توقف على سطر التعليمات البرمجية التالي:

```
result = Process1(products, i);
```

٦. افتح قائمة RUN AND DEBUG ثم حدد Start Debugging

٧. استخدم Step Into للتنقل عبر التعليمات البرمجية ل Process1 and Process2

لاحظ التحديثات لمقاطع VARIABLES and CALL STACK في قائمة RUN AND DEBUG

٨. استمر في استخدام Step Into للسير عبر التعليمات البرمجية حتى ترى أن i يساوي ٣

يعرض قسم VARIABLES في قائمة RUN AND DEBUG القيمة المعينة إلى i

لاحظ أن Process1 and Process2 يعرضان رسائل إلى لوحة DEBUG CONSOLE قد يتطلب التطبيق الحقيقي تفاعلات المستخدم أثناء معالجة البيانات، قد تعتمد بعض التفاعلات على البيانات التي تتم معالجتها

٩. استخدم الزر Stop لإيقاف تنفيذ التعليمات البرمجية.

تكوين نقطة توقف شرطية باستخدام تعبير

تعد نقطة التوقف القياسية رائعة للتنقل عبر تطبيق معالجة البيانات، ومع ذلك، في هذه الحالة كنت مهتماً بالمنتجات new ولا تريد الاطلاع على تحليل كل منتج للعثور على تلك التي هي new يعد هذا السيناريو مثالاً جيداً على الوقت الذي يجب فيه استخدام نقاط التوقف الشرطية.

١. انقر بزر الماوس الأيمن فوق نقطة التوقف الموجودة، ثم حدد تحرير نقطة التوقف Edit Breakpoint

٢. أدخل التعبير التالي:

```
products[i,1] == "new";
```

٣.لاحظ أنه لم يعد يتم عرض التعبير بعد الضغط على مفتاح الإدخال Enter

٤. لعرض التعبير مؤقتا، مرر مؤشر الماوس فوق نقطة التوقف (نقطة حمراء)

٥. لتشغيل التطبيق الخاص بك مع تكوين نقطة التوقف الشرطية، حدد بدء تصحيح الأخطاء Start Debugging

٦. انتظر حتى يتوقف التطبيق مؤقتا عند نقطة التوقف الشرطية conditional breakpoint

٧. لاحظ قيمة المعروضة i ضمن قسم VARIABLES

٨. في شريط أدوات عناصر التحكم في تتبع الأخطاء Debug controls حدد متابعة Continue

٩. لاحظ أنه تم تحديث قيمة `i` قسم VARIABLES

١٠. حدد خطوة إلى الأمام Step Into

١١. تابع تحديد Step Into حتى يتم عرض رسالة `Process1`

١٢. لاحظ أن `Process1` يُبلغ أنه يعمل على منتج جديد.

١٣. خذ لحظة للنظر في الميزة التي تقدمها نقاط التوقف الشرطية.

في هذا السيناريو الذي تمت محاكاته لمعالجة البيانات، هناك احتمال بنسبة ١٪ تقريبًا أن يكون المنتج جديدًا `new` إذا كنت تستخدم نقطة توقف قياسية لتصحيح المشكلة، فستحتاج إلى الاطلاع على تحليل حوالي ١٠٠ منتج للعثور على أحد المنتجات الجديدة التي تهتم بها.

يمكن أن توفر لك نقاط التوقف الشرطية الكثير من الوقت عندما تقوم بتصحيح أحد التطبيقات.

١٤. استخدم الزر Stop لإيقاف تنفيذ التعليمات البرمجية.

تهانينا! لقد قمت بتكوين نقطة توقف شرطية بنجاح.

خلاصة

فيما يلي أمران مهمان يجب تذكرهما من هذا الدرس:

- استخدم نقطة توقف قياسية standard breakpoint لإيقاف تطبيق مؤقتاً في كل مرة تتم فيها مصادفة نقطة توقف.
- استخدم نقطة توقف شرطية conditional breakpoint لإيقاف تطبيق مؤقتاً عند تقييم تعبير منطقي إلى true

٨ تمرين ـ مراقبة المتغيرات وتدفق التنفيذ

توفر قائمة RUN AND DEBUG للمطورين طريقة سهلة لمراقبة المتغيرات variables والتعبيرات expressions ومراقبة تدفق التنفيذ observe execution flow وإدارة نقاط التوقف أثناء عملية التصحيح.

فحص أقسام قائمة تشغيل وتصحيح الأخطاء Run and Debug

يوفر كل قسم من قائمة RUN AND DEBUG قدرات فريدة، غالباً ما يكون استخدام مجموعة من هذه الأقسام أثناء عملية التصحيح مفيدًا.

قسم المتغيرات VARIABLES

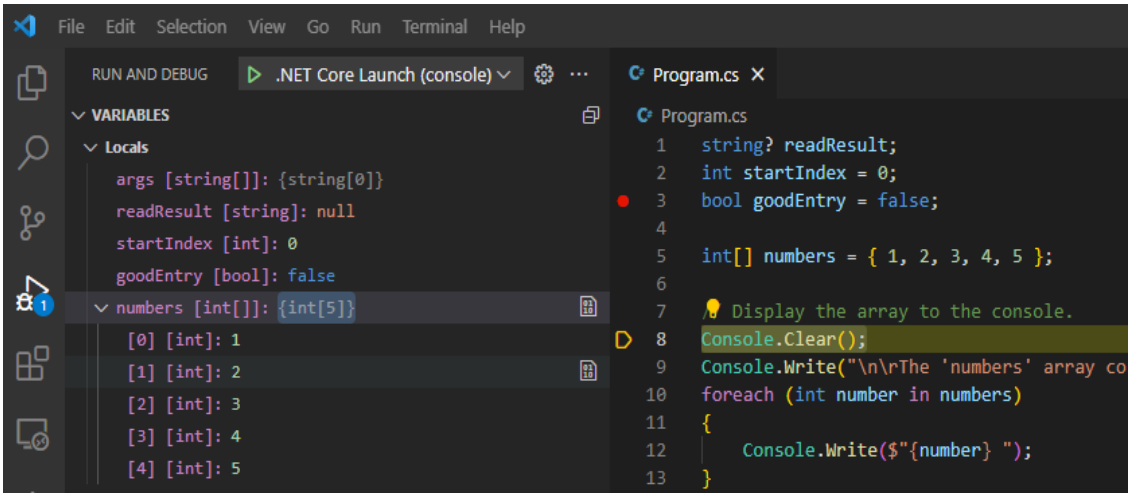
مراقبة حالة المتغير هو جانب مهم من تصحيح أخطاء التعليمات البرمجية، غالباً ما تساعد التغييرات غير المتوقعة في حالة المتغير، على تحديد الأخطاء المنطقية في التعليمات البرمجية.

يقوم قسم المتغيرات بتنظيم متغيراتك حسب النطاق، يعرض `Locals` نطاق المتغيرات في النطاق الحالي (الأسلوب الحالي)

ملاحظة

يعتبر قسم عبارات المستوى الأعلى top-level تطبيق وحدة التحكم أسلوبه الخاص، أسلوب يسمى Main

يمكنك فتح (expand) توسيع النطاقات المعروضة عن طريق تحديد السهم إلى يسار اسم النطاق، يمكنك أيضًا فتح المتغيرات والكائنات variables and objects تُظهر لقطة الشاشة التالية مصفوفة `numbers` المكشوفة ضمن النطاق `Locals`



من الممكن أيضاً تغيير قيمة متغير في وقت التشغيل، باستخدام قسم VARIABLES يمكنك النقر نقراً مزدوجاً فوق اسم المتغير ثم إدخال قيمة جديدة.

قسم المشاهدة أو المراقبة WATCH

ماذا لو كنت تريد تعقب حالة متغيرة عبر الوقت أو أساليب مختلفة؟ قد يكون البحث عن المتغير في كل مرة أمرًا شاقًا، هذا هو المكان الذي يكون فيه قسم المراقبة WATCH مفيدًا.

يمكنك تحديد الزر إضافة تعبير Add Expression (يظهر كعلامة الجمع +) لإدخال اسم متغير أو تعبير لمراقبته، كبديل، يمكنك النقر بزر الماوس الأيمن فوق متغير في قسم VARIABLES وتحديد Add to watch

سيتم تحديث جميع التعبيرات داخل قسم WATCH تلقائياً أثناء تشغيل التعليمات البرمجية.

قسم مكدس الاستدعاءات CALL STACK

في كل مرة تدخل فيها التعليمات البرمجية أسلوبًا من أسلوب آخر، تتم إضافة طبقة استدعاء إلى مكدس استدعاء التطبيق، عندما يصبح التطبيق الخاص بك معقدًا ولديك قائمة طويلة من الأساليب التي يتم استدعاؤها، بواسطة أساليب أخرى، يمثل مكدس الاستدعاءات CALL STACK مسار استدعاءات الأسلوب.

يكون قسم CALL STACK مفيدًا عندما تحاول العثور على الموقع المصدر لاستثناء أو تعبير WATCH إذا طرح تطبيقك استثناء غير متوقع، فسترى غالبًا رسالة في وحدة التحكم تشبه ما يلي:

```
Exception has occurred: CLR/System.DivideByZeroException
An unhandled exception of type 'System.DivideByZeroException' occurred in Debug1.dll: 'Attempted to divide by zero.'
    at Program.<<Main>$>g__WriteMessage|0_1() in C:\Users\howdc\Desktop\Debug1\Program.cs:line 27
    at Program.<<Main>$>g__Process1|0_0() in C:\Users\howdc\Desktop\Debug1\Program.cs:line 16
    at Program.<Main>$(String[] args) in C:\Users\howdc\Desktop\Debug1\Program.cs:line 10
```

تسمى المجموعة ذات المسافة البادئة في البرنامج at Program ... الأسطر الموجودة أسفل رسالة الخطأ تتبع المكدس stack trace يسرد تتبع المكدس stack trace اسم وأصل كل أسلوب تم استدعاؤه قبل حدوث الاستثناء، ومع ذلك، قد يكون من الصعب بعض الشيء فك تشفير المعلومات، لأنها قد تتضمن أيضًا معلومات من وقت تشغيل NET. في هذا المثال، يكون تتبع المكدس stack trace نظيفًا جدًا، ويمكنك رؤية حدوث هذا الاستثناء في أسلوب يسمى `WriteMessage` يتم إنشاء المكدس بأسلوب يسمى `Main` وهي قسم عبارات المستوى الأعلى top-level statements لتطبيق وحدة التحكم.

يمكن أن يساعدك قسم CALL STACK على تجنب صعوبة فك تتبع المكدس، المشوش بمعلومات وقت تشغيل NET. يقوم بتصفية المعلومات غير المرغوب فيها لإظهار الأساليب ذات الصلة فقط من التعليمات البرمجية، بشكل افتراضي، يمكنك إلغاء مكدس الاستدعاءات يدوياً لمعرفة مكان إنشاء الاستثناء.

قسم BREAKPOINTS

يعرض قسم BREAKPOINTS إعدادات نقطة التوقف الحالية، ويمكن استخدامه لتمكين نقاط توقف معينة أو تعطيلها أثناء جلسة تصحيح الأخطاء.

تكوين التطبيق الخاص بك وتشغيل التكوين

عند العمل على تطبيق وحدة تحكم يقرأ إدخال المستخدم، ستحتاج على الأرجح إلى تحديث ملف تكوين التشغيل launch configuration

١. تحديث التعليمات البرمجية في ملف Program.cs كما يلي:

```
string? readResult;
int startIndex = 0;
bool goodEntry = false;

int[] numbers = { 1, 2, 3, 4, 5 };

// Display the array to the console.
Console.Clear();
Console.Write("\n\rThe 'numbers' array
contains: { ");
foreach (int number in numbers)
{
    Console.Write($"{number} ");
}

// To calculate a sum of array elements,
//  prompt the user for the starting element
number.
Console.WriteLine($"}}\n\r\n\rTo sum values 'n'
through 5, enter a value for 'n':");
while (goodEntry == false)
{
```

```
    readResult = Console.ReadLine();
    goodEntry = int.TryParse(readResult, out
startIndex);

    if (startIndex > 5)
    {
        goodEntry = false;
        Console.WriteLine("\n\rEnter an integer
value between 1 and 5");
    }
}

// Display the sum and then pause.
Console.WriteLine($"\n\rThe sum of numbers
{startIndex} through {numbers.Length} is:
{SumValues(numbers, startIndex)}");

Console.WriteLine("press Enter to exit");
readResult = Console.ReadLine();

// This method returns the sum of elements n
through 5
static int SumValues(int[] numbers, int n)
{
    int sum = 0;
    for (int i = n; i < numbers.Length; i++)
    {
        sum += numbers[i];
    }
    return sum;
}
```

٢. خذ دقيقة لمراجعة التعليمات البرمجية.

لاحظ ما يلي:

- تحدد التعليمات البرمجية مصفوفة عدد صحيح يحتوي على خمسة أرقام.
- تعرض التعليمات البرمجية الإخراج في وحدة التحكم.
- تطالب التعليمات المستخدم بإدخال رقم عنصر البداية `n` الذي يستخدمه لجمع عناصر المصفوفة من `n` إلى `5`
- تحسب التعليمات البرمجية المجموع في أسلوب، وتعرض النتائج في وحدة التحكم، ثم تتوقف مؤقتًا.

ملاحظة

لا تدعم لوحة DEBUG CONSOLE إدخال المستخدم من وحدة التحكم.

٣. في قائمة Visual Studio Code File حدد Save

٤. في قائمة Run حدد Remove All Breakpoints

يؤدي ذلك إلى إزالة أي نقاط توقف متبقية من التمرين السابق.

٥ في قائمة RUN AND DEBUG حدد Start Debugging

٦. لاحظ حدوث خطأ عند تنفيذ سطر التعليمات `Console.Clear();`

٧. في شريط أدوات تتبع الأخطاء Debug toolbar حدد إيقاف Stop

٨. قم بالتبديل إلى قائمة EXPLORER ثم افتح ملف launch.json في المحرر.

٩. تحديث قيمة السمة console كما يلي:

```
"console":"integratedTerminal",
```

١٠. في قائمة Visual Studio Code File حدد Save ثم أغلق ملف launch.json

مراجعة إخراج التطبيق وتحديد المشكلات

يمكن أن تكشف مراجعة إخراج تطبيقك عن مشكلات المنطق التي تجاهلتها عند كتابة التعليمات البرمجية.

١. قم بالتبديل مرة أخرى إلى قائمة RUN AND DEBUG

٢. في قائمة RUN AND DEBUG حدد Start Debugging

تظهر الرسائل المعروضة على لوحة DEBUG CONSOLE مصحح الأخطاء المرفق بالتطبيق `Debug101.dll`

٣. لاحظ أنه لا يتم عرض رسائل خطأ.

أدى تغيير قيمة السمة console من InternalConsole إلى IntegratedTerminal في ملف تكوين التشغيل إلى إصلاح خطأ وحدة التحكم، لكنك الآن بحاجة إلى تحديد موقع وحدة التحكم التي تحتوي على مخرجاتك.

٤. في منطقة اللوحات أسفل المحرر، قم بالتبديل من لوحة DEBUG CONSOLE إلى لوحة TERMINAL

٥. لاحظ أن تنفيذ التعليمات البرمجية قد توقف مؤقتًا عند الرسالة التي تطالب المستخدم بإدخال قيمة ل `n`

يجب أن يبدو الإخراج على لوحة TERMINAL كما يلي:

```
The 'numbers' array contains: { 1 2 3 4 5 }

To sum values 'n' through 5, enter a value for 'n':
```

٦. في موجه الأوامر TERMINAL أدخل 3

٧. راجع الإخراج من التطبيق.

يجب أن يبدو الإخراج على لوحة TERMINAL كما يلي:

```
The 'numbers' array contains: { 1 2 3 4 5 }

To sum values 'n' through 5, enter a value for 'n':
3
```

```
The sum of numbers 3 through 5 is: 9
press Enter to exit
```

٨. خذ دقيقة للنظر في القيمة المبلغ عنها ل `sum` وقيم عناصر المصفوفة من 3 إلى 5 المعروضة في أعلى وحدة التحكم.

الرسالة تقول: `The sum of numbers 3 through 5 is: 9` ومع ذلك، فإن عناصر المصفوفة من 3 إلى 5 وهي 3, 4, 5 ألا يجب أن يكون المجموع المبلغ عنه 12؟

يمكنك استخدام قسم VARIABLES في قائمة RUN AND DEBUG للتحقيق في المشكلة.

مراقبة حالة المتغير variable state

في بعض الحالات، ببساطة مراقبة حالة المتغير كافية لتحديد مشكلة المنطق في التطبيق.

١. تعيين نقطة توقف على سطر التعليمات البرمجية التالي:

```
Console.WriteLine($"\n\rThe sum of numbers
{startIndex} through {numbers.Length} is:
{SumValues(numbers, startIndex)}");
```

٢. في قائمة RUN AND DEBUG حدد Start Debugging

٣. التبديل من لوحة DEBUG CONSOLE إلى لوحة TERMINAL

٤. في موجه الأوامر TERMINAL أدخل 3

سيتم إيقاف تنفيذ التعليمات البرمجية مؤقتًا عند نقطة التوقف.

٥. خذ دقيقة لمراجعة قسم VARIABLES في قائمة RUN AND DEBUG

لاحظ أنه تم تعيين القيمة التي أدخلتها ل `startIndex` وهي 3

٦. حدد خطوة إلى الأمام Step Into

٧. لاحظ أنه يتم تحديث قسمي VARIABLES and CALL STACK

يوضح قسم CALL STACK أن تنفيذ التعليمات البرمجية قد انتقل إلى الأسلوب `SumValues`

يُظهر قسم المتغيرات VARIABLES الذي يسرد المتغيرات المحلية، قيمة العدد الصحيح `n` يتم تعيين قيمة معلمة الأسلوب `n` من وسيطة استدعاء الأسلوب `startIndex` في هذه الحالة، يؤدي التغيير إلى أسماء المتغيرات إلى توضيح أن القيمة قد تم تمريرها، وليس مؤشرًا مرجعيًا reference pointer

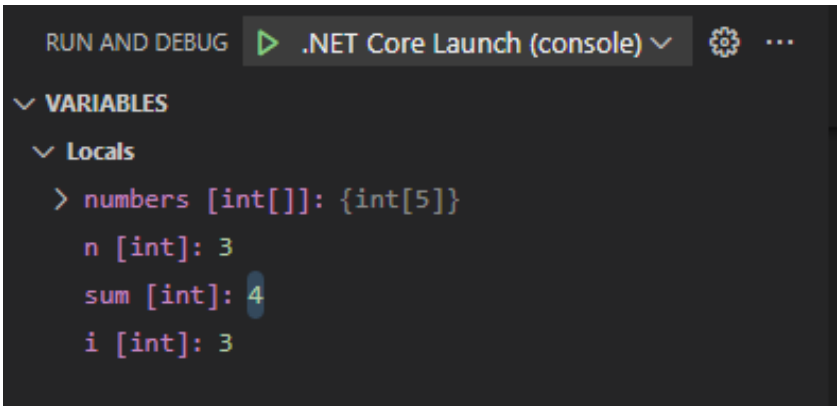
ملاحظة

في هذه الحالة، يمكنك رؤية معظم التعليمات البرمجية في المحرر، لذلك قد لا تحتاج إلى قسم CALL STACK ولكن عندما تعمل على تطبيقات أكبر باستخدام استدعاءات أسلوب متداخلة ومترابطة للغاية، يمكن أن يكون مسار التنفيذ الموضح في قسم CALL STACK مفيدًا للغاية

٨. تابع تحديد Step Into حتى لا تعود القيمة المعينة إلى sum هي 0

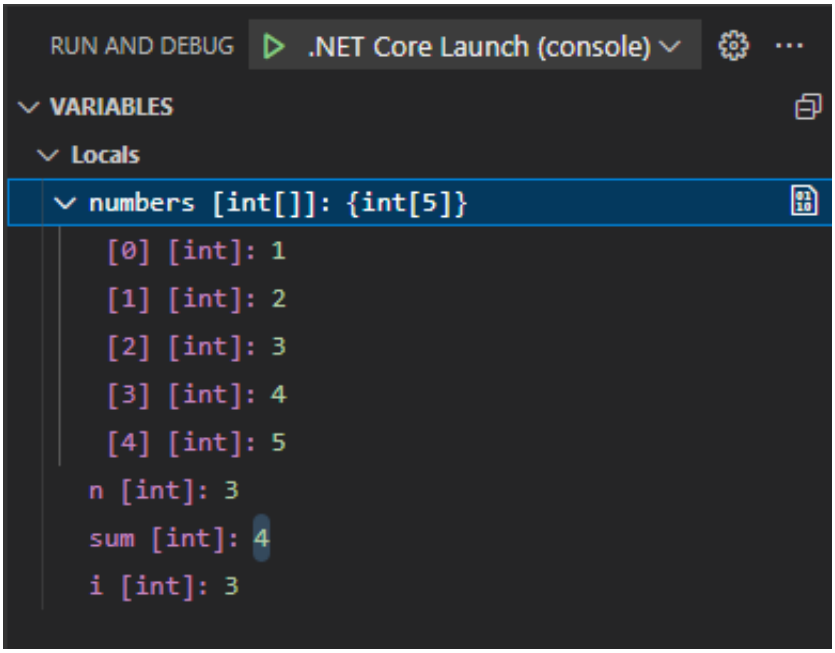
٩. خذ دقيقة لمراجعة المعلومات الموضحة في قسم VARIABLES

يجب أن ترى ما يلي:



لاحظ أن القيمة المعينة ل sum انتقلت من 0 إلى 4

١٠. لتوسيع المصفوفة `numbers` حدد numbers [int[]]



١١. تذكر أنه يتم الوصول إلى عناصر المصفوفة باستخدام أرقام الفهرس المستندة إلى الصفر.

في هذه الحالة، يكون خطأ المنطق هو تعارض بين الإرشادات الموجودة في واجهة المستخدم، والتعليمات البرمجية الأساسية، تشير واجهة المستخدم إلى عناصر المصفوفة 5-1 ومع ذلك، تستخدم التعليمات البرمجية القيمة التي أدخلها المستخدم للوصول إلى عناصر المصفوفة المستندة إلى الصفر، يخزن عنصر المصفوفة الذي يحتوي على فهرس 3 قيمة 4 لا تعوض التعليمات البرمجية عن أرقام الفهرس المستندة إلى الصفر.

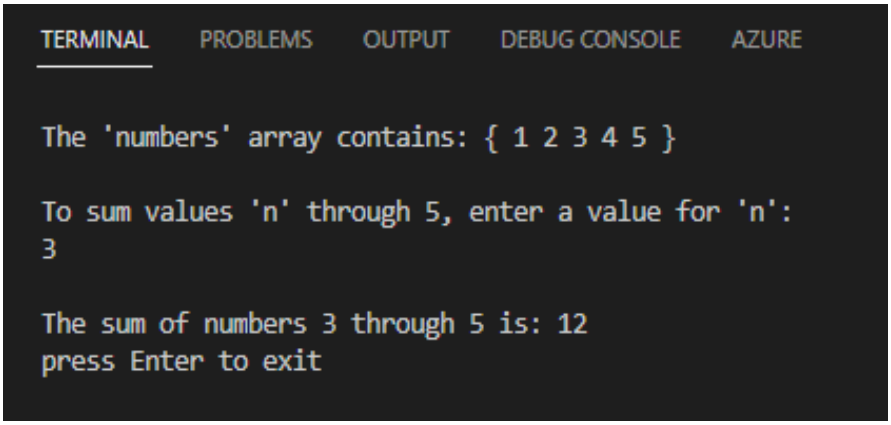
١٢. لإنهاء جلسة تصحيح الأخطاء، حدد إيقاف Stop

١٣. خذ دقيقة للنظر في كيفية إصلاح المشكلة.

يمكن تصحيح هذه المشكلة في واجهة المستخدم عن طريق مطالبة المستخدم بإدخال قيمة بين 0 و 4 يمكن أيضاً تصحيحه في التعليمات البرمجية عن طريق طرح 1 من القيمة التي تم إدخالها، بشكل عام، يجب أن يكون هدفك واضحاً وسهلاً لاتباع واجهة المستخدم، في هذه الحالة قد يكون من الأفضل تحديث التعليمات البرمجية كما يلي:

```
Console.WriteLine($"\n\rThe sum of numbers {startIndex} through {numbers.Length} is: {SumValues(numbers, startIndex - 1)}");
```

سيؤدي تشغيل التعليمات البرمجية المحدثة إلى إنتاج الإخراج التالي:



١٤. قم بتحديث التعليمات البرمجية باستخدام الأسلوب المقترح، ثم احفظ ملف Program.cs

١٥. قم بإلغاء تحديد نقطة التوقف، وأعد تشغيل التطبيق في مصحح الأخطاء، وتحقق من عرض النتيجة المقصودة في TERMINAL

لقد استخدمت للتو حالة المتغير variable state لتحديد مشكلة منطقية وإصلاحها! أحسنت.

مراقبة تعبيرات المراقبة

يمكن استخدام قسم WATCH لمراقبة التعبيرات التي تستند إلى متغير واحد أو أكثر.

لنفترض أنك تعمل على تطبيق يقوم بإجراء حسابات رقمية على مجموعة بيانات، تعتقد أن التعليمات البرمجية الخاصة بك تنتج نتائج غير موثوق بها، عندما تكون النسبة بين متغيرين رقميين أكبر من 5 يمكنك استخدام قسم WATCH لمراقبة النسبة المحسوبة.

١. تحديث ملف Program.cs بالتعليمات البرمجية التالية:

```
bool exit = false;
var rand = new Random();
int num1 = 5;
int num2 = 5;

do
```

```
{
    num1 = rand.Next(1, 11);
    num2 = num1 + rand.Next(1, 51);

} while (exit == false);
```

٢. احفظ ملف Program.cs

٣. تعيين نقطة توقف على سطر التعليمات البرمجية النهائي.

٤. تعيين تعبير WATCH التالي:

```
num2 / num1 > 5
```

٥. في قائمة RUN AND DEBUG حدد Start Debugging

٦. لاحظ القيم المعروضة في قسمي VARIABLES و WATCH

٧. حدد متابعة Continue حتى ترى تعبير WATCH يقيم إلى true

إذا تم تقييم تعبير WATCH إلى true في التكرار الأول، فحدد متابعة Continue عدة مرات أخرى، أو حتى ترى true مرة ثانية.

٨. خذ دقيقة للنظر في كيفية استخدام قسم WATCH

في هذا السيناريو، حددت أن التعليمات البرمجية تنتج نتائج غير موثوق بها، عندما تكون النسبة بين متغيرين رقميين أكبر من 5 لقد أنشأت تعبيراً في قسم WATCH يمثل هذا الشرط، يمكنك الآن استخدام قسم WATCH لتعقب هذا الشرط.

تعديل القيمة المعينة إلى متغير في قسم VARIABLES

قد تكون هناك أوقات تريد فيها إنشاء شرط برمجي يدوياً، يمكنك قسم VARIABLES في قائمة RUN AND DEBUG من القيام بذلك عن طريق تغيير حالة متغير في وقت التشغيل.

١. خذ دقيقة لمراجعة التعليمات البرمجية التي تقوم بتشغيلها.

لاحظ أن التعليمات البرمجية لن تخرج أبدًا من الحلقة do لأن exit لن تكون true أبدًا، هذا ليس شرطاً برمجياً تحتاج إلى تغييره في تطبيق حقيقي، ولكنه يوضح الإمكانية.

٢. في قسم VARIABLES انقر بزر الماوس الأيمن فوق exit [bool] ثم حدد Set Value

يمكنك قسم VARIABLES من تغيير القيمة المعينة إلى متغير في وقت التشغيل، يمكن أن يكون هذا مفيدًا عندما تريد معرفة كيفية تفاعل التعليمات البرمجية مع شرط معين.

٣. لتعيين قيمة exit إلى true اكتب true ثم اضغط على Enter

في هذه الحالة، سيؤدي تغيير قيمة exit إلى true إلى إغلاق التطبيق عند تنفيذ العبارة while

٤. حدد متابعة Continue

٥. لاحظ أن لوحة DEBUG CONSOLE تعرض رسالة تعلمك بإنهاء البرنامج.

تهانينا! لقد استخدمت بنجاح قسمي VARIABLES و WATCH في قائمة RUN AND DEBUG

خلاصة

فيما يلي بعض الأشياء المهمة التي يجب تذكرها من هذا الدرس:

- مراقبة حالة المتغير باستخدام قسم VARIABLES في قائمة RUN AND DEBUG
- تعقب تعبير عبر الوقت أو أساليب مختلفة باستخدام قسم WATCH في قائمة RUN AND DEBUG
- استخدم قسم CALL STACK في قائمة RUN AND DEBUG للعثور على الموقع المصدر لاستثناء أو تعبير WATCH
- استخدم قسم VARIABLES لتغيير القيمة المعينة للمتغير في وقت التشغيل.

٩ تمرين ـ تحدي إكمال نشاط باستخدام مصحح الأخطاء

تستخدم تحديات التعليمات البرمجية في هذا التدريب، لتعزيز ما تعلمته، ومساعدتك على اكتساب بعض الثقة قبل المتابعة.

تحدي حالة المتغير

في هذا التحدي، يتم تزويدك بنموذج تعليمات برمجية، لا تنتج النتيجة المتوقعة، تحتاج إلى استخدام قسمي نقاط التوقف breakpoints والمتغيرات VARIABLES في قائمة RUN AND DEBUG لمساعدتك في معرفة المشكلات.

١. أدخل نموذج التعليمات البرمجية التالي في محرر Visual Studio

```
/*
This code instantiates a value and then calls
the ChangeValue method
to update the value. The code then prints the
updated value to the console.
    هذه التعليمات مثيل value ثم تقوم باستدعاء الأسلوب ChangeValue
  لتحديث value تقوم التعليمات بعد ذلك بطباعة القيمة المحدثة إلى وحدة التحكم
*/
int x = 5;

ChangeValue(x);

Console.WriteLine(x);

void ChangeValue(int value)
{
    value = 10;
}
```

٢. يصف تعليق التعليمات البرمجية الوظيفة المطلوبة.

٣. قم بتشغيل التطبيق في مصحح أخطاء Visual Studio Code

٤. تحقق من الإخراج الذي تم إنتاجه.

٥. استخدم أدوات مصحح الأخطاء C# لعزل المشكلة.

٦. ضع في اعتبارك كيفية تحديث التعليمات البرمجية لمطابقة الوظيفة المطلوبة.

١٠ مراجعة حل تحدي إكمال نشاط باستخدام مصحح الأخطاء

المثال التالي لعملية تتبع الأخطاء هو أحد الحلول الممكنة للتحدي من الدرس السابق.

تنفيذ أدوات مصحح الأخطاء C# لتحديد المشكلة

تنفذ عملية التصحيح التالية نقطة توقف ثم تراقب قيمة `x` في قسم VARIABLES في قائمة RUN AND DEBUG

١. تعيين نقطة توقف على سطر التعليمات البرمجية التالي:

```
int x = 5;
```

٢. افتح قائمة RUN AND DEBUG

٣. في أعلى قائمة RUN AND DEBUG حدد Start Debugging

٤. في قسم VARIABLES في قائمة Run and Debug دون القيمة المعينة إلى `x`

٥. في شريط أدوات التحكم في تتبع الأخطاء Debug control حدد خطوة إلى الأمام Step Into

٦. تعقب القيمة المعينة إلى `x` أثناء التنقل عبر كل سطر تعليمة برمجية.

٧. لاحظ أن قيمة `x` لا تتغير عند إدخال التنفيذ وإنهاء الأسلوب `ChangeValue`

يتم تمرير الأسلوب `ChangeValue` قيمة `x` بدلاً من مرجع reference إلى `x` وبالتالي فإن التغيير في value داخل الأسلوب لا يؤثر على المتغير الأصلي `x`

النظر في تحديث التعليمات البرمجية بناءً على نتائج تصحيح الأخطاء

إذا كنت تريد أن يغير الأسلوب `ChangeValue` القيمة value في التعليمة البرمجية للاستدعاء، فستحتاج إلى تحديث التعليمات، إحدى الطرق لتحقيق النتيجة المقصودة هي تحديث الأسلوب `ChangeValue` لإرجاع قيمة عدد صحيح، وتحديث التعليمات البرمجية التي تستدعي `ChangeValue` بحيث تعين القيمة المرجعة إلى `x`

على سبيل المثال:

```
int x = 5;
x = ChangeValue(x);
Console.WriteLine(x);

int ChangeValue(int value)
{
    value = 10;
    return value;
}
```

إذا نجحت في هذا التحدي، تهانينا! تابع لاختبار المعلومات في الدرس التالي.

هام

إذا كان لديك مشكلة في إكمال هذا التحدي، ربما يجب عليك مراجعة الدروس السابقة قبل المتابعة.

١١ التحقق من المعرفة

١. أي قسم من قائمة RUN AND DEBUG يستخدم لتعقب نقطة التنفيذ الحالية داخل التطبيق قيد التشغيل؟

- قسم المتغيرات VARIABLES
- قسم CALL STACK
- قسم المشاهدة WATCH

٢. أي من الخيارات التالية يمكن استخدامها لتعيين نقطة توقف set a breakpoint في Visual Studio Code؟

- انقر بزر الماوس الأيسر في العمود الموجود على يسار رقم السطر في محرر التعليمات البرمجية.
- حدد تبديل نقطة التوقف Toggle Breakpoint في القائمة Edit
- انقر بزر الماوس الأيمن في منتصف سطر التعليمات البرمجية، ثم حدد تبديل نقطة التوقف Toggle Breakpoint

٣. كيف يمكن للمطور تعطيل نقطة توقف disable a breakpoint في Visual Studio Code؟

- انقر فوق Remove Breakpoint من القائمة Run
- انقر بزر الماوس الأيمن فوق النقطة الحمراء إلى يسار رقم السطر وحدد تعطيل نقطة التوقف Disable Breakpoint
- انقر بزر الماوس الأيسر فوق النقطة الحمراء إلى يسار رقم السطر.

٤. ما هي نقطة التوقف الشرطية conditional breakpoint في Visual Studio Code؟

- نقطة توقف يتم تشغيلها فقط عند استيفاء شرط محدد.
- نقطة توقف يتم تشغيلها في كل مرة يتم فيها تشغيل التعليمات البرمجية.
- نقطة توقف مرئية فقط في المحرر ولا تؤثر على تصحيح الأخطاء.

٥. كيف يمكن للمطور إنشاء نقطة توقف شرطية conditional breakpoint في Visual Studio Code؟

- انقر بزر الماوس الأيسر في العمود إلى يسار رقم سطر في محرر التعليمات البرمجية.
- حدد تبديل نقطة التوقف الشرطية Toggle Conditional Breakpoint في قائمة Run
- انقر بزر الماوس الأيمن فوق العمود على يسار رقم السطر، ثم حدد إضافة نقطة توقف شرطية Add Conditional Breakpoint

٦. في ملف تكوين التشغيل launch configuration ما هي سمة وحدة التحكم console attribute المستخدمة؟

- تحديد دليل العمل للعملية المستهدفة.
- تحديد نوع مصحح الأخطاء الذي يجب استخدامه لتكوين التشغيل هذا.
- تحديد نوع وحدة التحكم المستخدمة عند تشغيل التطبيق.

راجع إجابتك

١ قسم CALL STACK

صحيح يتم استخدام قسم CALL STACK لتتبع نقطة التنفيذ الحالية داخل التطبيق قيد التشغيل، بدءًا من نقطة الإدخال الأولية إلى التطبيق

٢ انقر بزر الماوس الأيسر في العمود الموجود على يسار رقم سطر في محرر التعليمات البرمجية.

صحيح يمكن تعيين نقطة توقف عن طريق وضع مؤشر الماوس في العمود على يسار رقم السطر ثم النقر بزر الماوس الأيسر

٣ انقر بزر الماوس الأيمن فوق النقطة الحمراء إلى يسار رقم السطر وحدد تعطيل نقطة التوقف Disable Breakpoint

صحيح يمكن تعطيل نقطة توقف بالنقر بزر الماوس الأيمن فوق النقطة الحمراء التي تمثل نقطة توقف ثم تحديد Disable Breakpoint

٤ نقطة توقف يتم تشغيلها فقط عند استيفاء شرط محدد.

صحيح نقطة التوقف الشرطية هي نوع خاص من نقاط التوقف التي يتم تشغيلها فقط عند استيفاء شرط محدد

٥ انقر بزر الماوس الأيمن فوق العمود على يسار رقم السطر، ثم حدد إضافة نقطة توقف شرطية Add Conditional Breakpoint

صحيح يمكن للمطور إنشاء نقطة توقف شرطية بالنقر بزر الماوس الأيمن فوق العمود إلى يسار رقم السطر، ثم تحديد Add Conditional Breakpoint

٦ تحديد نوع وحدة التحكم المستخدمة عند تشغيل التطبيق.

صحيح تحدد سمة console attribute نوع وحدة التحكم المستخدمة عند تشغيل التطبيق، الخيارات هي internalConsole, integratedTerminal, externalTerminal الإعداد الافتراضي هو internalConsole

١٢ الملخص

كان هدفك هو اكتساب خبرة في تصحيح أخطاء تطبيقات C# باستخدام Visual Studio Code

من خلال تكوين debugger environment بيئة مصحح أخطاء Visual Studio Code لسيناريوهات تطبيق نموذجية مختلفة، اكتسبت خبرة في كل جانب تقريباً من واجهة مصحح الأخطاء، لقد استخدمت نقاط التوقف breakpoints ونقاط التوقف الشرطية conditional breakpoints لإيقاف التنفيذ مؤقتاً pause في المواقع الهامة داخل التعليمات البرمجية.

أثناء إيقاف التنفيذ مؤقتاً، استخدمت أقسام VARIABLES, WATCH, CALL STACK في قائمة RUN AND DEBUG لتقييم حالة المتغيرات والتعبيرات والتطبيق الكلي، كما استخدمت عناصر التحكم في التنفيذ على **شريط** أدوات Debug toolbar تتبع الأخطاء عبر خطوط التعليمات البرمجية، وتجاوزها أثناء تشغيل التطبيق.

حتى أنك استخدمت ملف launch.json لتكوين وحدة التحكم التي تريد استخدامها ولتكوين بيئة تصحيح الأخطاء للمشاريع التي تتضمن أكثر من ملف قابل للتنفيذ.

بدون القدرة على تكوين أدوات مصحح أخطاء Visual Studio Code واستخدامها، لن تتمكن من عزل أخطاء التعليمات البرمجية وإصلاحها في الوقت المناسب.

المصادر:

- يمكنك العثور على معلومات إضافية حول مصحح أخطاء Visual Studio Code وتشغيل التكوينات: debugging
- يمكنك العثور على معلومات إضافية حول تصحيح أخطاء .NET في debugger Visual Studio Code
- يمكنك العثور على معلومات إضافية حول تكوين مصحح أخطاء C# debugger-settings

الوحدة الثالثة

تنفيذ معالجة الاستثناءات في تطبيقات وحدة تحكم C#

تعرف على الاستثناءات exceptions وعملية معالجة الاستثناء exception handling التي يدعمها C# ثم قم بتنفيذ أنماط معالجة الاستثناءات لسيناريوهات الترميز المختلفة.

الأهداف التعليمية

- فحص الفئات الأساسية للاستثناءات، ومراجعة بعض استثناءات النظام الشائعة.
- فحص أدوات معالجة الاستثناء المضمنة مع C# وتقنيات استخدام هذه الأدوات.
- تنفيذ النمط `try-catch` ضمن سيناريوهات ترميز مختلفة.

محتويات الوحدة: -

١- مقدمة

٢- فحص الاستثناءات exceptions وعملية معالجة الاستثناء

٣- فحص الاستثناءات التي تم إنشاؤها بواسطة المترجم -compiler generated exceptions

٤- تمرين - تنفيذ معالجة استثناء try-catch

٥- تمرين - إكمال نشاط تحدي try-catch

٦- مراجعة حل تحدي try-catch

٧- التقاط أنواع استثناء محددة Catch specific exception

٨- تمرين - إكمال نشاط تحدي catch لالتقاط استثناءات محددة

٩- مراجعة حل تحدي catch التقاط استثناءات محددة

١٠- التحقق من المعرفة

١١- الملخص

١ المقدمة

تتضمن لغة C# ميزات معالجة الاستثناء exception handling التي تساعدك على إدارة أخطاء وقت التشغيل، التي تحدث بسبب مواقف غير متوقعة أو استثنائية، يتم تنفيذ معالجة الاستثناءات في C# باستخدام الكلمات الأساسية `try`, `catch`, `finally` تمكن هذه الكلمات الأساسية التعليمات البرمجية من تجربة الإجراءات التي قد لا تنجح، ومعالجة حالات الفشل عندما تقرر أنه من المعقول القيام بذلك، وتنظيف الموارد بعد ذلك، يمكن إنشاء الاستثناءات بواسطة وقت تشغيل اللغة الشائعة (CLR) أو بواسطة NET. أو مكتبات الجهات الخارجية أو بواسطة التعليمات البرمجية للتطبيق.

لنفترض أنك تطور تطبيقًا لإدارة المشتريات والمخازن باستخدام C# يعالج التطبيق عمليات الشراء المباشرة، ويدير بيانات المخزون المرتبطة باستخدام مزيج من إدخال المستخدم، ومنطق العمل المضمن، يظهر الاختبار الأولي أن المخالفات في بيانات أمر الشراء يمكن أن تؤدي إلى حالات خطأ غير متوقعة، غالباً ما تؤدي الأخطاء إلى فقدان البيانات أو سوء إدارتها، تحتاج إلى تنفيذ معالجة الاستثناء والتأكد من أن تطبيقك ينفذ بطريقة متوقعة قبل إصداره.

في هذه الوحدة، ستتعرف على الاستثناءات exceptions وعملية معالجة الاستثناء exception handling وأنماط معالجة الاستثناء التي تدعمها C# يمكنك أيضاً معرفة كيفية تنفيذ معالجة الاستثناء لسيناريوهات الترميز المختلفة، خلال الوحدة، ستكمل أنشطة التحدي التي تعزز ما تعلمته.

في نهاية الوحدة، ستتمكن من تطوير تطبيقات C# التي تدير مشكلات وقت التشغيل غير المتوقعة، وتقديم تجربة مستخدم أكثر استقراراً.

٢ فحص الاستثناءات exceptions وعملية معالجة الاستثناء

تتم إدارة أخطاء وقت التشغيل في تطبيق C# باستخدام آلية تسمى الاستثناءات exceptions توفر الاستثناءات طريقة منظمة وموحدة وآمنة للنوع لمعالجة كل من مستوى النظام، وظروف الخطأ على مستوى التطبيق، يتم إنشاء الاستثناءات بواسطة وقت تشغيل .NET. أو بواسطة التعليمات البرمجية في أحد التطبيقات.

السيناريوهات الشائعة التي تتطلب معالجة الاستثناء

هناك العديد من سيناريوهات البرمجة التي تتطلب معالجة الاستثناء، تتضمن العديد من هذه السيناريوهات شكلاً من أشكال الحصول على البيانات، على الرغم من أن بعض السيناريوهات تتضمن تقنيات ترميز خارج نطاق هذا التدريب، إلا أنها لا تزال جديرة بالملاحظة.

تتضمن السيناريوهات الشائعة التي تتطلب معالجة الاستثناء ما يلي:

- إدخال المستخدم: يمكن أن تحدث استثناءات عندما تعالج التعليمات إدخال المستخدم، على سبيل المثال، تحدث استثناءات عندما تكون قيمة الإدخال بتنسيق خاطئ أو خارج النطاق.
- معالجة البيانات والحسابات: يمكن أن تحدث الاستثناءات عندما تقوم التعليمات بإجراء عمليات حسابية للبيانات أو تحويلات، على سبيل المثال، تحدث استثناءات عندما تحاول التعليمات البرمجية القسمة على صفر، أو تحويلها إلى نوع غير معتمد، أو تعيين قيمة خارج النطاق.
- عمليات إدخال/إخراج الملف: يمكن أن تحدث استثناءات عند قراءة التعليمات من ملف أو كتابته، على سبيل المثال، تحدث استثناءات عندما لا يكون الملف موجودًا، أو لا يملك البرنامج الإذن للوصول إلى الملف، أو يكون الملف قيد الاستخدام بواسطة عملية أخرى.
- عمليات قاعدة البيانات: يمكن أن تحدث استثناءات عندما تتفاعل التعليمات مع قاعدة بيانات، على سبيل المثال، تحدث استثناءات عند فقدان اتصال قاعدة البيانات، أو حدوث خطأ في بناء الجملة في عبارة SQL أو حدوث انتهاك قيد.

- اتصال الشبكة: يمكن أن تحدث استثناءات عندما تتصل التعليمات عبر شبكة، على سبيل المثال، تحدث الاستثناءات عند فقدان اتصال الشبكة، أو حدوث مهلة أو إرجاع الخادم البعيد لخطأ.
- الموارد الخارجية الأخرى: يمكن أن تحدث استثناءات عندما تتصل التعليمات بالموارد الخارجية الأخرى، يمكن لخدمات الويب أو واجهات برمجة تطبيقات REST أو مكتبات الجهات الخارجية طرح استثناءات لأسباب مختلفة، على سبيل المثال، تحدث استثناءات بسبب مشكلات اتصالات الشبكة، والبيانات التي تم تكوينها بشكل غير جيد، وما إلى ذلك.

معالجة الاستثناء للكلمات الأساسية keywords وكتل التعليمات البرمجية code blocks والأنماط patterns

يتم تنفيذ معالجة الاستثناء في C# باستخدام الكلمات الأساسية `try`, `catch`, `finally` تحتوي كل كلمة من هذه الكلمات الأساسية على كتلة تعليمات برمجية مقترنة، ويمكن استخدامها لتلبية هدف معين في نهجك لمعالجة الاستثناء، على سبيل المثال:

```
try
{
   // try code block - code that may generate
an exception
}
catch
{
   // catch code block - code to handle an
exception
}
finally
{
   // finally code block - code to clean up
resources
}
```

ملاحظة

تمكن لغة C# أيضاً التعليمات البرمجية من إنشاء كائن استثناء `exception` `object` باستخدام الكلمة الأساسية `throw` تتم تغطية سيناريوهات معالجة الاستثناء التي تتضمن استخدام الكلمة الأساسية `throw` لإنشاء استثناءات في وحدة منفصلة في هذا المسار التعليمي.

تحتوي كتلة `try` على التعليمات البرمجية المحمية التي قد تسبب استثناء، إذا تسببت التعليمات البرمجية داخل كتلة `try` استثناء، فستتم معالجة الاستثناء بواسطة كتلة `catch` المقابلة.

تحتوي كتلة `catch` على التعليمات البرمجية التي يتم تنفيذها عند اكتشاف استثناء، يمكن للكتلة `catch` معالجة الاستثناء أو تسجيله أو تجاهله، يمكن تكوين كتلة `catch` لتنفيذها عند حدوث أي نوع استثناء، أو فقط عند حدوث نوع معين من الاستثناء.

تحتوي كتلة `finally` على تعليمات برمجية يتم تنفيذها سواء حدث استثناء أم لا، غالباً ما يتم استخدام الكتلة `finally` لتنظيف أي موارد يتم تخصيصها في كتلة `try` على سبيل المثال، التأكد من أن المتغير يحتوي على القيمة الصحيحة أو المطلوبة المعينة له.

يتم تنفيذ معالجة الاستثناءات في تطبيق C# بشكل عام باستخدام نمط واحد أو أكثر من الأنماط التالية:

- يتكون نمط `try-catch` من كتلة `try` متبوعة بواحدة أو أكثر من عبارات `catch` يتم استخدام كل كتلة `catch` لتحديد معالجات لاستثناءات مختلفة.
- يتكون نمط `try-finally` من كتلة `try` متبوعة بكتلة `finally` عادة يتم تشغيل عبارات كتلة `finally` عندما يترك عنصر التحكم عبارة `try`
- ينفذ نمط `try-catch-finally` جميع الأنواع الثلاثة من كتل معالجة الاستثناءات، أحد السيناريوهات الشائعة لنمط `try-catch-finally` هو عندما يتم الحصول على الموارد واستخدامها في كتلة `try` وتتم إدارة الظروف الاستثنائية في كتلة `catch` ويتم تحرير الموارد أو إدارتها بطريقة أخرى في كتلة `finally`

كيف يتم تمثيل الاستثناءات في التعليمات البرمجية؟

يتم تمثيل الاستثناءات في التعليمات البرمجية ككائنات، ما يعني أنها مثيل لفئة class توفر مكتبة .NET class. فئات استثناء يتم الوصول إليها في التعليمات تماماً مثل فئات .NET class. الأخرى، مثال آخر لفئة .NET. المستخدمة كعنصر في التعليمات الفئة Random (المستخدمة لإنشاء أرقام عشوائية)

وبشكل أكثر دقة، الاستثناءات هي أنواع، ممثلة بفئات مشتقة جميعاً في نهاية المطاف من `System.Exception` تتضمن فئة الاستثناء المشتقة من Exception معلومات تحدد نوع الاستثناء وتحتوي على خصائص توفر تفاصيل حول الاستثناء، يتم تضمين فحص أكثر تفصيلاً للفئة Exception لاحقا في هذه الوحدة.

يشار إلى مثيل وقت التشغيل لفئة بشكل عام كعنصر object لذلك غالباً ما يشار إلى الاستثناءات كعناصر استثناء exception objects

ملاحظة

على الرغم من أنهما يستخدمان أحيانًا بالتبادل، إلا أن الفئة class والكائن object هما شيئان مختلفان، تحدد الفئة نوع الكائن، ولكنها ليست كائناً بحد ذاته، الكائن هو كيان ملموس يعتمد على فئة.

عملية معالجة الاستثناء

عند حدوث استثناء، يبحث وقت تشغيل .NET. عن أقرب عبارة catch يمكنها معالجة الاستثناء، تبدأ العملية بالأسلوب الذي تسبب في طرح الاستثناء، أولاً، يتم فحص الأسلوب لمعرفة ما إذا كانت التعليمات البرمجية التي تسببت في الاستثناء موجودة داخل الكتلة البرمجية try إذا كانت التعليمات داخل كتلة try فسيتم مراعاة ترتيب عبارات catch المقترنة ب try

إذا كانت العبارات catch غير قادرة على معالجة الاستثناء، فسيتم البحث عن الأسلوب الذي يستدعي الأسلوب الحالي، يتم فحص هذا الأسلوب لتحديد ما إذا كان استدعاء الأسلوب (إلى الأسلوب الأول) داخل كتلة try إذا كان

الاستدعاء داخل كتلة try يتم النظر في العبارات المقترنة catch تستمر عملية البحث هذه حتى يتم العثور على عبارة catch يمكنها معالجة الاستثناء الحالي.

بمجرد العثور على عبارة catch يمكنها معالجة الاستثناء، يستعد وقت التشغيل لنقل عنصر التحكم إلى العبارة الأولى للكتلة catch ومع ذلك، قبل بدء تنفيذ الكتلة catch ينفذ وقت التشغيل أي كتل finally مقترنة بعبارات try التي تم العثور عليها أثناء البحث، إذا تم العثور على أكثر من كتلة واحدة finally فسيتم تنفيذها بالترتيب، بدءاً من الأقرب إلى التعليمات البرمجية التي تسببت في طرح الاستثناء.

إذا لم يتم العثور على عبارة catch لمعالجة الاستثناء، ينهي وقت التشغيل التطبيق، ويعرض رسالة خطأ للمستخدم.

خذ بعين الاعتبار نموذج التعليمات البرمجية التالي الذي يتضمن نمطا try-finally متداخلاً داخل نمط try-catch

```
try
{
    // Step 1: code execution begins
    try
    {
        // Step 2: an exception occurs here
    }
    finally
    {
        // Step 4: the system executes the
finally code block associated with the try
statement where the exception occurred
    }

}
catch // Step 3: the system finds a catch
clause that can handle the exception
{
   // Step 5: the system transfers control to
the first line of the catch code block
}
```

في هذا المثال، تحدث العملية التالية:

١. يبدأ التنفيذ في كتلة التعليمات البرمجية من العبارة الخارجية try

٢. يتم طرح استثناء في كتلة التعليمات البرمجية من العبارة الداخلية try

٣. يعثر وقت التشغيل على العبارة catch المقترنة بالجملة الخارجية try

٤. قبل أن ينقل وقت التشغيل عنصر التحكم إلى السطر الأول من كتلة catch ينفذ العبارة finally المقترنة بالجملة الداخلية try

٥. ثم ينقل وقت التشغيل عنصر التحكم إلى السطر الأول من كتلة catch وينفذ التعليمات البرمجية التي تعالج الاستثناء.

في هذا المثال البسيط، توجد أنماط try-catch and try-finally المتداخلة، ضمن أسلوب واحد، ولكن يمكن نشر عدة أنماط try-catch and try-finally بين الأساليب التي تستدعي أساليب أخرى.

معالجة الاستثناء ومكدس الاستدعاءات call stack

غالباً ما ترى مصطلح إلغاء تشغيل مكدس الاستدعاءات "call stack unwinding" عند القراءة حول معالجة الاستثناء، وعملية معالجة الاستثناء، لفهم هذا المصطلح، تحتاج إلى فهم مكدس الاستدعاءات وكيفية استخدامه ل تتبع مكدس track the stack استدعاءات الأسلوب أثناء تنفيذ التعليمات البرمجية.

يمكنك التفكير في مكدس الاستدعاءات مثل برج كتل برمجية، عندما تبني برجاً، تبدأ بكتلة واحدة فقط، في كل مرة تضيف كتلة إلى البرج، تضعها فوق الكتل الموجودة، عند بدء تشغيل التطبيق في مصحح الأخطاء، تكون نقطة الإدخال إلى التطبيق هي الطبقة الأولى المضافة إلى مكدس الاستدعاءات (الكتلة الأولى من البرج) في كل مرة يستدعي أسلوب أسلوباً آخر، تتم إضافة الأسلوب الجديد إلى أعلى المكدس، عند خروج التعليمات البرمجية من أسلوب، تتم إزالة الأسلوب من مكدس الاستدعاءات.

ملاحظة

بالنسبة لتطبيق وحدة التحكم، نقطة الإدخال إلى تطبيقك هي عبارات المستوى الأعلى، في مكدس استدعاء `Visual Studio Code` يشار إلى نقطة الإدخال هذه باسم الأسلوب `Main`

إلغاء مكدس الاستدعاءات هو العملية التي يستخدمها وقت تشغيل .NET. عندما يواجه برنامج C# خطأ، نفس العملية التي قمت بمراجعتها للتو.

بالعودة إلى قياس برج الكتلة، عندما تحتاج إلى إزالة كتلة من البرج، تبدأ من الأعلى وتزيل كل كتلة حتى تصل إلى الكتلة التي تحتاج إليها، تشبه هذه العملية كيفية عمل إلغاء مكدس الاستدعاءات، حيث تكون كل طبقة استدعاء في المكدس مثل كتلة في البرج، عندما يحتاج وقت التشغيل إلى فك مكدس الاستدعاءات، فإنه يبدأ من الأعلى ويزيل كل طبقة استدعاء حتى تصل إلى تلك التي تحتوي على ما تحتاجه، في هذه الحالة، طبقة الاستدعاء التي تحتاجها هي الطريقة التي تحتوي على عبارة `catch` يمكنها معالجة الاستثناء الذي حدث.

خلاصة

فيما يلي بعض الأشياء المهمة التي يجب تذكرها من هذا الدرس:

- تتضمن السيناريوهات الشائعة التي قد تتطلب معالجة الاستثناء إدخال المستخدم، ومعالجة البيانات، وعمليات إدخال/إخراج الملفات، وعمليات قاعدة البيانات، واتصال الشبكة.
- يتم تنفيذ معالجة الاستثناءات في C# باستخدام الكلمات الأساسية try, catch, finally تحتوي كل كلمة أساسية على كتلة تعليمات برمجية مقترنة تخدم غرضاً محددًا.
- يتم تمثيل الاستثناءات كأنواع ومشتقة من فئة `System.Exception` في .NET. تحتوي الاستثناءات على معلومات تحدد نوع الاستثناء، والخصائص التي توفر تفاصيل إضافية.
- عند حدوث استثناء، يبحث وقت تشغيل .NET. عن أقرب عبارة `catch` يمكنها معالجة الاستثناء، يبدأ البحث بالأسلوب حيث تم طرح الاستثناء، وينتقل لأسفل مكدس الاستدعاءات إذا لزم الأمر.

اختبر معلوماتك

١- أي من خيارات الإجابات التالية يسرد الكلمات الأساسية C# المستخدمة لمعالجة الاستثناء؟

- try, catch, and finally
- try, catch, finally, and throw
- try, catch, and throw

٢. ما الذي تحتوي عليه كتلة التعليمات البرمجية try؟

- التعليمات التي تنظف الموارد ويتم تنفيذها سواء حدث استثناء أم لا.
- التعليمات البرمجية التي تطرح كائن استثناء exception object
- التعليمات البرمجية التي قد تسبب استثناء.

٣. ما الغرض من كتلة التعليمات البرمجية catch؟

- لتنظيف أي موارد يتم تخصيصها في كتلة try
- لمعالجة الاستثناء، القيام بتسجيله أو تجاهله.
- لتنفيذ التعليمات البرمجية التي قد تسبب استثناء.

٤. ما هو الغرض من الكتلة finally؟

- لتنفيذ التعليمات البرمجية التي قد تسبب استثناء.
- لمعالجة الاستثناء، القيام بتسجيله أو تجاهله.
- لتنظيف أي موارد يتم تخصيصها في كتلة try

راجع إجابتك

١ try, catch, finally, and throw

صحيح يتم استخدام الكلمات الأساسية try, catch, finally, throw لمعالجة الاستثناء في C#

٢ التعليمات البرمجية التي قد تسبب استثناء.

صحيح تحتوي كتلة try على التعليمات البرمجية المحمية التي قد تسبب استثناء

٣ لمعالجة الاستثناء، القيام بتسجيله أو تجاهله.

صحيح تحتوي كتلة catch على التعليمات البرمجية التي يتم تنفيذها عند اكتشاف استثناء، يمكن للكتلة catch معالجة الاستثناء أو تسجيله أو تجاهله

٤ لتنظيف أي موارد يتم تخصيصها في كتلة try

صحيح غالباً ما يتم استخدام الكتلة finally لتنظيف أي موارد يتم تخصيصها في كتلة try تحتوي كتلة finally على التعليمات البرمجية التي يتم تنفيذها سواء حدث استثناء أم لا

٣ فحص الاستثناءات التي تم إنشاؤها بواسطة المترجم compiler-generated exceptions

يتم إنشاء الاستثناءات بواسطة وقت تشغيل .NET. أو بواسطة التعليمات البرمجية في تطبيق، يعتمد نوع الاستثناء على التعليمات البرمجية التي تسبب الاستثناء.

الاستثناءات التي تم إنشاؤها بواسطة المحول البرمجي

يطرح وقت تشغيل .NET. استثناءات عند فشل العمليات الأساسية، فيما يلي قائمة قصيرة باستثناءات وقت التشغيل وشروط الخطأ الخاصة بها:

`ArrayTypeMismatchException` يتم طرحه عندما لا يمكن لمصفوفة تخزين عنصر معين، لأن النوع الفعلي للعنصر غير متوافق مع النوع الفعلي للمصفوفة.

`DivideByZeroException` يتم طرحه عند إجراء محاولة لتقسيم قيمة متكاملة integral value على صفر.

`FormatException` يتم طرحه عندما يكون تنسيق وسيطة argument غير صالح.

`IndexOutOfRangeException` يتم طرحه عند محاولة فهرسة مصفوفة، عندما يكون الفهرس أقل من صفر أو خارج حدود المصفوفة.

`InvalidCastException` يتم طرحه عند فشل تحويل صريح من نوع أساسي إلى واجهة أو إلى نوع مشتق في وقت التشغيل.

`NullReferenceException` يتم طرحه عند إجراء محاولة للإشارة إلى كائن قيمته فارغة.

`OverflowException` يتم طرحه عند تجاوز عملية حسابية في سياق محدد.

نماذج التعليمات البرمجية للاستثناءات التي تم إنشاؤها بواسطة المحول البرمجي

تعرض نماذج التعليمات البرمجية التالية مثالاً على التعليمات البرمجية التي تتسبب في استثناء، تم إنشاؤه بواسطة المحول البرمجي.

ArrayTypeMismatchException

يتم طرح استثناء من النوع ArrayTypeMismatchException عند إجراء محاولة لتخزين عنصر من النوع الخطأ داخل مصفوفة، يطرح المثال التالي استثناء ArrayTypeMismatchException عند محاولة تخزين قيمة رقمية في مصفوفة سلسلة.

```
string[] names = { "Dog", "Cat", "Fish" };
Object[] objs = (Object[])names;

Object obj = (Object)13;
objs[2] = obj; // ArrayTypeMismatchException occurs
```

DivideByZeroException

يحدث استثناء من النوع DivideByZeroException عند محاولة قسمة عدد صحيح أو رقم عشري على صفر، يطرح المثال التالي استثناء DivideByZeroException عند إجراء قسمة عدد صحيح.

```
int number1 = 3000;
int number2 = 0;
Console.WriteLine(number1 / number2); // DivideByZeroException occurs
```

ملاحظة

لا تؤدي قسمة قيمة الفاصلة العائمة floating-point إلى حدوث استثناء؛ ينتج عنه ما لا نهاية إيجابية أو لا نهاية سالبة أو رقم (NaN) وفقاً لقواعد حساب IEEE 754

FormatException

يحدث استثناء من النوع FormatException عندما يكون تنسيق الوسيطة غير صالح، أو عندما لا يتم تشكيل تنسيق مركب لسلسلة بشكل جيد، يطرح المثال التالي استثناء FormatException عند محاولة تحويل سلسلة إلى عدد صحيح.

```
int valueEntered;
string userValue = "two";
valueEntered    =    int.Parse(userValue);    //
FormatException occurs
```

IndexOutOfRangeException

يتم طرح استثناء من النوع IndexOutOfRangeException عند محاولة الوصول إلى عنصر من مصفوفة أو مجموعة باستخدام فهرس خارج حدوده، يطرح المثال التالي استثناء IndexOutOfRangeException عند محاولة تعيين العنصر الأخير من المصفوفة `values1` إلى العنصر الأخير من المصفوفة `values2`

```
int[] values1 = { 3, 6, 9, 12, 15, 18, 21 };
int[] values2 = new int[6];

values2[values1.Length       -       1]       =
values1[values1.Length       -       1];       //
IndexOutOfRangeException occurs
```

InvalidCastException

يتم طرح استثناء من النوع InvalidCastException عند محاولة تحويل غير صالح أو تحويل صريح، يطرح المثال التالي استثناء InvalidCastException عند محاولة تحويل object من نوع string إلى متغير int

```
object obj = "This is a string";
int num = (int)obj;
```

NullReferenceException

يتم طرح استثناء من النوع NullReferenceException عند محاولة الوصول إلى عضو على نوع تكون قيمته فارغة، يظهر مثالان، في المثال الأول يتم طرح NullReferenceException عند محاولة الوصول إلى عنصر من مصفوفة فارغة null array المثال الثاني يطرح NullReferenceException عند محاولة الوصول إلى أسلوب سلسلة فارغة null string

مثال:

```
int[] values = null;
for (int i = 0; i <= 9; i++)
    values[i] = i * 2;
```

مثال:

```
string? lowCaseString = null;
Console.WriteLine(lowCaseString.ToUpper());
```

OverflowException

يحدث استثناء من النوع OverflowException عندما تحاول عملية حسابية تعيين نتيجة خارج نطاق نوع البيانات المستهدفة، يطرح المثال التالي استثناء OverflowException عند محاولة تحويل decimal قيمة 400 إلى متغير byte

```
decimal x = 400;
byte i;

i = (byte)x; // OverflowException occurs
Console.WriteLine(i);
```

خلاصة

فيما يلي بعض الأشياء المهمة التي يجب تذكرها من هذا الدرس:

- يطرح وقت تشغيل NET runtime. استثناء عند فشل عملية.
- يعتمد نوع الاستثناء على التعليمات البرمجية التي تسبب الاستثناء.
- يجب أن يلتقط تطبيقك استثناءات وقت التشغيل.

اختبر معلوماتك

١- متى يتم طرح استثناء ArrayTypeMismatchException؟

- يتم طرح استثناء ArrayTypeMismatchException عند إجراء محاولة لتقسيم قيمة متكاملة على صفر.
- يتم طرح استثناء ArrayTypeMismatchException عند محاولة فهرسة مصفوفة خارج حدود المصفوفة.
- يتم طرح استثناء ArrayTypeMismatchException عند محاولة تخزين قيمة من نوع محدد في مصفوفة من نوع آخر.

٢. متى يتم طرح استثناء DivideByZeroException؟

- يتم طرح استثناء DivideByZeroException عند إجراء محاولة لتقسيم قيمة متكاملة على صفر.
- يتم طرح استثناء DivideByZeroException عند محاولة فهرسة مصفوفة خارج حدود المصفوفة.
- يتم طرح استثناء DivideByZeroException عند محاولة تخزين قيمة من نوع محدد في مصفوفة من نوع آخر.

٣. متى يتم طرح استثناء IndexOutOfRangeException؟

- يتم طرح استثناء IndexOutOfRangeException عند إجراء محاولة لتقسيم قيمة متكاملة على صفر.
- يتم طرح استثناء IndexOutOfRangeException عند محاولة فهرسة مصفوفة خارج حدود المصفوفة.
- يتم طرح استثناء IndexOutOfRangeException عند محاولة تخزين قيمة من نوع محدد في مصفوفة من نوع آخر.

٤. متى يتم طرح استثناء InvalidCastException؟

- يتم طرح استثناء InvalidCastException عند إجراء محاولة لتقسيم قيمة متكاملة على صفر.
- يتم طرح استثناء InvalidCastException عند محاولة تخزين قيمة من نوع محدد في مصفوفة من نوع آخر.
- يتم طرح استثناء InvalidCastException عند محاولة تحويل غير صالح أو تحويل صريح.

٥. متى يتم طرح استثناء NullReferenceException؟

- يتم طرح استثناء NullReferenceException عند محاولة تحويل غير صالح أو تحويل صريح.
- يتم طرح استثناء NullReferenceException عند محاولة تخزين قيمة من نوع محدد في مصفوفة من نوع آخر.
- يتم طرح استثناء NullReferenceException عند محاولة الوصول إلى عضو على نوع قيمته فارغة.

راجع إجابتك

١ يتم طرح استثناء ArrayTypeMismatchException عند محاولة تخزين قيمة من نوع محدد في مصفوفة من نوع آخر.

صحيح يتم طرح الاستثناء ArrayTypeMismatchException عند إجراء محاولة لتخزين عنصر من النوع الخطأ داخل مصفوفة

٢ يتم طرح استثناء DivideByZeroException عند إجراء محاولة لتقسيم قيمة متكاملة على صفر.

صحيح يتم طرح DivideByZeroException عند محاولة تقسيم عدد صحيح أو رقم عشري على صفر

٣ يتم طرح استثناء IndexOutOfRangeException عند محاولة فهرسة مصفوفة خارج حدود المصفوفة.

صحيح يتم طرح استثناء عند إجراء محاولة للوصول إلى عنصر من مصفوفة أو مجموعة باستخدام فهرس خارج حدوده

٤ يتم طرح استثناء InvalidCastException عند محاولة تحويل غير صالح أو تحويل صريح.

صحيح يتم طرح استثناء InvalidCastException عند محاولة تحويل غير صالح أو تحويل صريح

٥ يتم طرح استثناء NullReferenceException عند محاولة الوصول إلى عضو على نوع قيمته فارغة.

صحيح يتم طرح استثناء NullReferenceException عند محاولة الوصول إلى عضو على نوع قيمته فارغة

٤ تمرين - تنفيذ معالجة استثناء try-catch

يتكون نمط `try-catch` من كتلة `try` متبوعة بواحدة أو أكثر من عبارات `catch` تحدد كل عبارة `catch` المعالج لنوع استثناء مختلف.

عندما يتم طرح استثناء، يبحث وقت تشغيل اللغة الشائعة (CLR) عن عبارة `catch` يمكنها معالجة الاستثناء، إذا كان الأسلوب المنفذ حالياً لا يحتوي على عبارة `catch` يمكنها التعامل مع نوع الاستثناء الذي تم طرحه، فسيقوم CLR بالبحث عن الأسلوب الذي يستدعي الأسلوب الحالي، يستمر البحث عبر مكدس الاستدعاءات حتى يتم العثور على عبارة `catch` المناسبة، إذا لم يتم العثور على عبارة `catch` لمعالجة الاستثناء، فسيعرض CLR رسالة استثناء غير معالج للمستخدم ويوقف تنفيذ البرنامج.

في هذا التمرين، ستقوم بتنفيذ نمط أساسي `try-catch`

إنشاء مشروع تعليمات برمجية جديدة

خطوتك الأولى هي إنشاء مشروع تعليمات برمجية تستخدمه أثناء هذا الدرس.

١. افتح مثيلاً جديدًا من Visual Studio Code

٢. في القائمة ملف File حدد فتح مجلد Open Folder

٣. في مربع الحوار فتح مجلد Open Folder انتقل إلى مجلد Desktop سطح مكتب Windows

٤. في مربع الحوار فتح مجلد Open Folder حدد مجلد جديد New folder

٥. قم بتسمية المجلد الجديد Exceptions101 ثم حدد Select Folder

٦. في القائمة المحطة الطرفية Terminal حدد محطة طرفية جديدة New Terminal

ستستخدم أمر .NET CLI command لإنشاء تطبيق وحدة تحكم جديد.

٧. في موجه أوامر لوحة TERMINAL أدخل الأوامر التالية:

```
dotnet new console
```

٨. أغلق لوحة TERMINAL

تنفيذ تجربة بسيطة

١. استخدم قائمة Visual Studio Code EXPLORER لفتح ملف Program.cs

٢. في القائمة العرض View حدد لوحة الأوامر Command Palette

٣. في موجه الأوامر command prompt أدخل .net: g ثم حدد .NET: Generate Assets for Build and Debug

٤. استبدل محتوى ملف Program.cs بالتعليمات البرمجية التالية:

```
double float1 = 3000.0;
double float2 = 0.0;
int number1 = 3000;
int number2 = 0;

Console.WriteLine(float1 / float2);
Console.WriteLine(number1 / number2);
Console.WriteLine("Exit program");
```

٥. خذ دقيقة لفحص التعليمات البرمجية.

لاحظ أن التطبيق يستخدم نوعين من المتغيرات الرقمية double, int تقوم التعليمات البرمجية بإجراء عملية حسابية للقسمة باستخدام كلا النوعين الرقميين.

يستخدم المطورون متغير نوع double للحسابات عندما تكون القيم الكسرية الدقيقة مهمة.

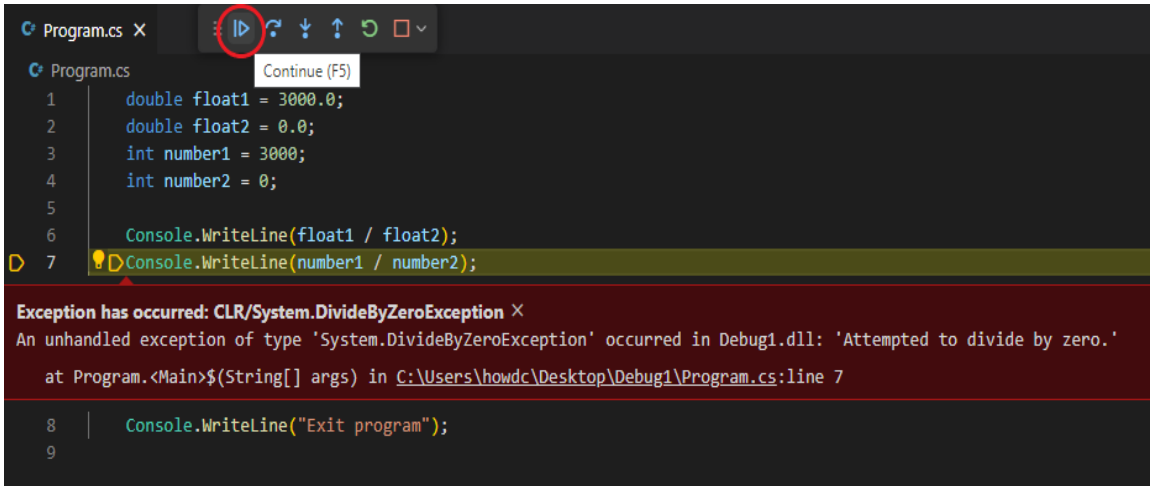
٦. في قائمة Run حدد Start Debugging

لاحظ حدوث استثناء `DivideByZeroException` عند تقسيم قيم العدد الصحيح.

ملاحظة

ربما لاحظت أن المعادلة التي تستخدم متغيرات من النوع double قادرة على الإكمال دون التسبب في حدوث خطأ، تؤدي عملية القسمة على الصفر باستخدام متغيرات النوع double بإرجاع نتيجة مساوية لما لا نهاية infinity أو "ليس رقمًا" لا يعني هذا أنه يجب عليك دائما استخدام متغيرات النوع double بدلاً من الأنواع int أو decimal النهج الصحيح هو استخدام متغيرات من النوع المناسب، وتنفيذ معالجة الاستثناءات لاكتشاف أي أخطاء قد تحدث

٧. في شريط أدوات تتبع الأخطاء Debug toolbar حدد متابعة Continue



٨. خذ دقيقة من وقتك لفحص مخرجات الرسالة لتطبيقك.

∞

```
Unhandled exception. System.DivideByZeroException:
Attempted to divide by zero.
   at Program.<Main>$(String[] args) in
C:\Users\msuser\Desktop\Exceptions101\Program.cs:l
ine 7
```

لاحظ أن الاستثناء غير المعالج تسبب في إيقاف تشغيل التطبيق، بعد اكتمال العبارة الأولى Console.WriteLine()

ملاحظة

افتراضيًا، يستخدم Visual Studio Code نصًا بلون مختلف لعرض الرسائل التي تم إنشاؤها بواسطة مصحح الأخطاء، يساعد هذا المطور على التمييز بين مخرجات التطبيق، ورسائل مصحح الأخطاء، إذا كنت تريد عرضًا أوضح لمخرجات تطبيقك، فيمكنك تكوين ملف Launch.json لاستخدام وحدة تحكم مختلفة، على سبيل المثال، قم بتعيين وحدة التحكم console على IntegratedTerminal لاستخدام لوحة TERMINAL لإخراج التطبيق، يتم دائمًا عرض رسائل مصحح الأخطاء في لوحة DEBUG CONSOLE

٩. قم بتضمين الحسابين داخل كتلة التعليمات الخاصة بعبارة try كما يلي:

```
double float1 = 3000.0;
double float2 = 0.0;
int number1 = 3000;
int number2 = 0;

try
{
    Console.WriteLine(float1 / float2);
    Console.WriteLine(number1 / number2);
}

Console.WriteLine("Exit program");
```

١٠. لاحظ الخط الأحمر المتعرج أسفل قوس الإغلاق للكتلة try

يتطلب بناء جملة C# عبارة catch أو finally عند استخدام عبارة try

١١. إنشاء كتلة تعليمة برمجية catch أسفل كتلة التعليمات البرمجية try كما يلي:

```
try
{
    Console.WriteLine(float1 / float2);
    Console.WriteLine(number1 / number2);
}
catch
{
    Console.WriteLine("An exception has been
caught");
}
```

١٢. في قائمة Visual Studio Code File حدد Save

١٣. في قائمة Run حدد Start Debugging

١٤. خذ دقيقة لفحص الإخراج الذي أنتجه تطبيقك.

```
∞
An exception has been caught
Exit program
```

١٥. لاحظ أنه على الرغم من أن الاستثناء لا يزال يحدث، فإن التطبيق قادر الآن على إنهاء تنفيذ أسطر التعليمات البرمجية المتبقية قبل الإغلاق.

تمكنك معالجة الاستثناء من التحكم في تنفيذ التعليمات البرمجية عند حدوث استثناءات، تساعد معالجة الاستثناء على التأكد من أن التعليمات البرمجية الخاصة بك مستقرة، وتنتج النتائج المتوقعة.

التقاط الاستثناءات التي تم طرحها في الأساليب المستدعاه

في كثير من الحالات، يتم اكتشاف استثناء على مستوى مكدس الاستدعاءات stack أقل من المستوى الذي تم طرحه فيه.

عند طرح استثناء ولا يلتقط الأسلوب الحالي الاستثناء، سيقوم وقت تشغيل اللغة الشائعة common language runtime بفك المكدس، بحثاً عن أسلوب يحتوي على عبارة catch يمكنها معالجة الاستثناء، سيتم تنفيذ العبارة الأولى catch التي تم العثور عليها والتي يمكنها التعامل مع الاستثناء، إذا

لم يتم العثور على عبارة مناسبة، في أي مكان في مكدس الاستدعاءات، فسينهي وقت تشغيل اللغة الشائعة العملية، ويعرض رسالة خطأ للمستخدم.

استبدل التعليمات البرمجية في ملف Program.cs بالتعليمات التالية:

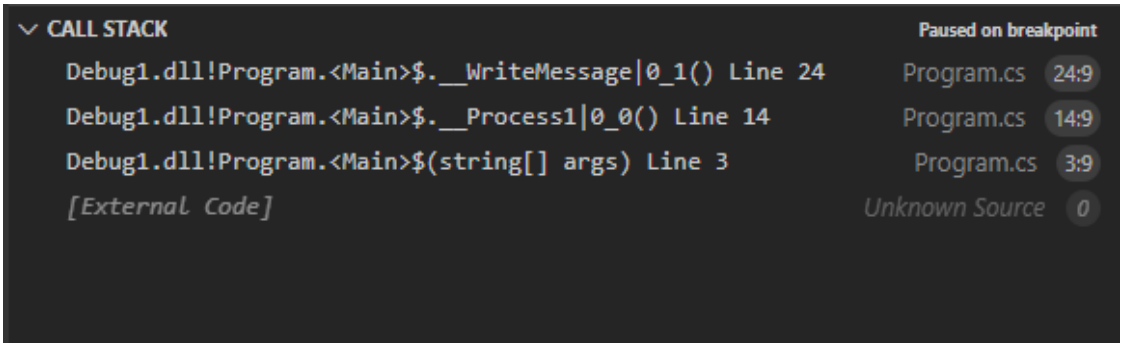
```
try
{
    Process1();
}
catch
{
    Console.WriteLine("An exception has
occurred");
}

Console.WriteLine("Exit program");

static void Process1()
{
    WriteMessage();
}

static void WriteMessage()
{
    double float1 = 3000.0;
    double float2 = 0.0;
    int number1 = 3000;
    int number2 = 0;

    Console.WriteLine(float1 / float2);
    Console.WriteLine(number1 / number2);
}
```

٢. خذ دقيقة لمراجعة التعليمات البرمجية المحدثة.

- تتضمن عبارات المستوى الأعلى top-level كتلة `try` التي تستدعي الأسلوب `Process1()`

- يستدعي الأسلوب `Process1()` الأسلوب `WriteMessage()`
- يحتوي الأسلوب `WriteMessage()` على التعليمات حيث سيتم طرح الاستثناء `DivideByZeroException`

لاحظ أنه سيتم إنشاء الاستثناء بطريقة تحتوي على مستويين من مكدس الاستدعاءات، فوق كتل التعليمات البرمجية try و catch



يتم تمثيل عبارات المستوى الأعلى كأسلوب مسمى `Main` في مكدس الاستدعاءات.

٣. في قائمة Visual Studio Code File حدد Save

٤. في قائمة Run حدد Start Debugging

٥. خذ دقيقة لفحص الإخراج الذي أنتجه التطبيق.

```
∞
An exception has occurred
Exit program
```

خلاصة

فيما يلي بعض الأشياء المهمة التي يجب تذكرها من هذا الدرس:

- تنفيذ نمط `try-catch` لتجربة أسطر التعليمات البرمجية `try` المحددة داخل التطبيق والتقاط استثناءات `catch` التي تحدث ضمن نطاق كتلة التعليمات البرمجية `try`
- استخدم عبارة `catch` لالتقاط استثناء تم طرحه في نفس مستوى مكدس الاستدعاءات.
- استخدم عبارة `catch` لالتقاط استثناء تم طرحه على مستوى أعلى من مكدس الاستدعاءات.

٥ تمرين - إكمال نشاط تحدي try-catch

تستخدم تحديات التعليمات البرمجية في هذه الوحدة، لتعزيز ما تعلمته ومساعدتك على اكتساب بعض الثقة قبل المتابعة.

تحدي Try-Catch

تنفيذ معالجة الاستثناء لتلبية معلمات التحدي التالية:

١. ابدأ بتعليمات التطبيق التالي:

```
try
{
    Process1();
}
catch
{
    Console.WriteLine("An exception has
occurred");
}

Console.WriteLine("Exit program");

static void Process1()
{
    WriteMessage();
}

static void WriteMessage()
{
    double float1 = 3000.0;
    double float2 = 0.0;
    int number1 = 3000;
    int number2 = 0;

    Console.WriteLine(float1 / float2);
```

```
    Console.WriteLine(number1 / number2);
}
```

٢. قم بتحديث الأسلوب `Process1` لالتقاط الاستثناء الذي تم طرحه في الأسلوب `WriteMessage`

٣. يجب أن يطبع الأسلوب `Process1` الرسالة التالية إلى وحدة التحكم عند اكتشاف الاستثناء:

```
Exception caught in Process1
```

٤. لا تغير أي تعليمة برمجية خارج الأسلوب `Process1`

٥. عند تشغيل التطبيق المحدث، يجب أن يقوم بإنشاء الإخراج التالي:

```
∞
Exception caught in Process1
Exit program
```

لأغراض هذا التحدي، يمكنك تجاهل رسائل الإخراج التي أنشأها مصحح الأخطاء، على سبيل المثال، يمكنك تجاهل الرسائل التالية:

```
Exception thrown: 'System.DivideByZeroException'
in Exceptions101.dll

The program '[436] Exceptions101.dll' has exited
with code 0 (0x0).
```

٦ مراجعة حل تحدي try-catch

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الدرس السابق:

```
try
{
    Process1();
}
catch
{
    Console.WriteLine("An exception has
occurred");
}

Console.WriteLine("Exit program");

static void Process1()
{
    try
    {
        WriteMessage();
    }
    catch
    {
        Console.WriteLine("Exception caught in
Process1");
    }

}

static void WriteMessage()
{
    double float1 = 3000.0;
    double float2 = 0.0;
    int number1 = 3000;
    int number2 = 0;
```

```
    Console.WriteLine(float1 / float2);
    Console.WriteLine(number1 / number2);
}
```

في هذا الحل تم تحديث الأسلوب `Process1` لاستخدام النمط try-catch يتم استدعاء الأسلوب `WriteMessage` في كتلة `try` مما يمكّن `Process1` من التقاط الاستثناء قبل أن يتم اكتشافه بواسطة عبارة catch في عبارات المستوى الأعلى.

يؤدي تشغيل هذا التطبيق إلى إنشاء الإخراج المطلوب:

```
∞
Exception caught in Process1
Exit program
```

لاحظ، نظراً لاكتشاف الاستثناء داخل Process1 لا يتم تنفيذ كتلة `catch` في عبارات المستوى الأعلى، تصبح الفوائد المكتسبة من خلال التقاط الاستثناءات على مستويات مختلفة في مكدس الاستدعاءات، أكثر وضوحاً عند اكتشاف أنواع استثناءات محددة، ستفحص أنواع الاستثناءات في الدرس التالي.

إذا نجحت في هذا التحدي، تهانينا! تابع إلى الدرس التالي.

هام

إذا كان لديك مشكلة في إكمال هذا التحدي، ربما يجب عليك مراجعة الدروس السابقة قبل المتابعة.

٧ التقاط أنواع استثناء محددة Catch specific exception

في وقت سابق من هذه الوحدة، تعلمت أن كائنات الاستثناء exception objects التي تم التقاطها بواسطة تطبيق C# هي مثيلات لفئة استثناء exception class بشكل عام، ستكون التعليمات البرمجية `catch` واحدة مما يلي:

- كائن استثناء هو مثيل للفئة `System.Exception` الأساسية.
- كائن استثناء يمثل مثيلاً لنوع الاستثناء الذي يرث من الفئة الأساسية، على سبيل المثال، مثيل لفئة `InvalidCastException`

فحص خصائص الاستثناء

`System.Exception` هي الفئة الأساسية التي ترث منها كافة أنواع الاستثناءات المشتقة، يرث كل نوع استثناء من الفئة الأساسية، من خلال تسلسل هرمي معين للفئة، على سبيل المثال، التسلسل الهرمي للفئة `InvalidCastException` هو كما يلي:

```
Object
    Exception
        SystemException
            InvalidCastException
```

معظم فئات الاستثناء التي ترث من `Exception` لا تضيف أي وظائف إضافية؛ إنهم ببساطة يرثون من `Exception` ولذلك، فإن فحص خصائص فئة `Exception` يمكّنك من فهم معظم الاستثناءات، وكيف يمكنك استخدام استثناء في التعليمات البرمجية.

فيما يلي خصائص الفئة Exception

- Data تحتوي خاصية Data على بيانات عشوائية في أزواج قيم المفاتيح.
- HelpLink يمكن استخدام خاصية HelpLink للاحتفاظ بعنوان URL (أو URN) لملف تعليمات يوفر معلومات شاملة حول سبب الاستثناء.
- HResult يمكن استخدام خاصية HResult للوصول إلى قيمة رقمية مشفرة تم تعيينها لاستثناء محدد.
- InnerException يمكن استخدام الخاصية InnerException لإنشاء سلسلة من الاستثناءات والاحتفاظ بها أثناء معالجة الاستثناءات.
- Message توفر الخاصية Message تفاصيل حول سبب الاستثناء.
- Source يمكن استخدام الخاصية Source للوصول إلى اسم التطبيق أو العنصر الذي يسبب الخطأ.
- StackTrace تحتوي الخاصية StackTrace على تتبع المكدس الذي يمكن استخدامه لتحديد مكان حدوث خطأ، يمكن استخدام الخاصية للحصول على الأسلوب الذي يطرح الاستثناء الحالي.
- TargetSite يمكن استخدام الخاصية TargetSite للحصول على الأسلوب التي يطرح الاستثناء الحالي.

لا بأس إذا كنت تشعر بالإرهاق قليلاً من خلال هذا الفحص لخصائص الاستثناء exception والفئات الأساسية base classes والوراثة inheritance لا تقلق، فإن اكتشاف الاستثناءات في التعليمات البرمجية والوصول إلى خصائص الاستثناء أسهل من شرح كيفية عمل الاستثناءات وخصائص الاستثناء.

ملاحظة

في هذه الوحدة، ستركز على استخدام خاصية رسالة الاستثناء exception's message للإبلاغ عن الاستثناء في واجهة مستخدم تطبيقك.

الوصول إلى خصائص كائن استثناء

الآن بعد أن فهمت كائنات الاستثناء وخصائصها، حان الوقت لبدء الترميز.

١. تحديث ملف Program.cs كما يلي:

```
try
{
    Process1();
}
catch
{
    Console.WriteLine("An exception has
occurred");
}

Console.WriteLine("Exit program");

static void Process1()
{
    try
    {
        WriteMessage();
    }
    catch
    {
        Console.WriteLine("Exception caught in
Process1");
    }
}

static void WriteMessage()
```

```
{
    double float1 = 3000.0;
    double float2 = 0.0;
    int number1 = 3000;
    int number2 = 0;

    Console.WriteLine(float1 / float2);
    Console.WriteLine(number1 / number2);
}
```

٢. خذ دقيقة لمراجعة التعليمات البرمجية.

هذه هي نفس التعليمات البرمجية التي رأيتها في الدرس السابق، أنت تعرف أنه يتم طرح استثناء أثناء الأسلوب `WriteMessage` تعرف أيضاً أن الاستثناء يتم اكتشافه في الأسلوب `Process1` ستستخدم هذه التعليمة البرمجية لفحص كائنات الاستثناء exception objects وأنواع الاستثناءات المحددة specific exception types

٣. حدّث الأسلوب `Process1` على النحو التالي:

```
static void Process1()
{
    try
    {
        WriteMessage();
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Exception caught in
Process1: {ex.Message}");
    }
}
```

٤. خذ دقيقة لفحص تحديثاتك.

لاحظ أن عبارتك المحدثة `catch` تلتقط مثيلاً للفئة Exception في كائن يسمى `ex` لاحظ أيضاً أن الأسلوب Console.WriteLine() يستخدم `ex`

للوصول إلى خاصية الكائن Message وعرض رسالة الخطأ إلى وحدة التحكم.

على الرغم من أنه يمكن استخدام عبارة catch بدون وسيطات، إلا أنه لا يوصى بهذه الطريقة، إذا لم تحدد وسيطة، فسيتم اكتشاف جميع أنواع الاستثناءات، ولا يمكنك التمييز بينها.

بشكل عام، يجب عليك فقط التقاط الاستثناءات التي تعرف التعليمات البرمجية كيفية التعامل معها، لذلك، يجب أن تحدد عبارة catch وسيطة كائن مشتقة من System.Exception يجب أن يكون نوع الاستثناء محددًا قدر الإمكان، يساعد هذا على تجنب التقاط الاستثناءات التي لا يتمكن معالج الاستثناء من حلها، ستقوم بتحديث التعليمات البرمجية لاكتشاف نوع استثناء محدد، لاحقًا في هذا التمرين.

٥. في القائمة ملف File حدد حفظ Save

٦. تعيين نقطة توقف Set breakpoint على سطر التعليمات البرمجية التالي:

```
Console.WriteLine($"Exception caught in Process1: {ex.Message}");
```

٧. في قائمة Run حدد Start Debugging

يجب أن يتوقف تنفيذ التعليمات البرمجية مؤقتًا عند نقطة التوقف.

٨. مرر مؤشر الماوس فوق ex

لاحظ أن IntelliSense يعرض نفس خصائص الاستثناء التي فحصتها سابقاً.

٩. خذ دقيقة لفحص المعلومات التي تصف كائن الاستثناء ex

لاحظ أن الاستثناء System.DivideByZeroException هو من نوع وتم تعيين الخاصية Message إلى "محاولة القسمة على صفر" Attempted to divide by zero.

لاحظ أن الخاصية StackTrace تُبلغ عن الأسلوب ورقم السطر حيث حدث الخطأ، بالإضافة إلى تسلسل استدعاءات الأسلوب (وأرقام الأسطر) التي أدت إلى الخطأ.

١٠. في شريط أدوات تتبع الأخطاء Debug toolbar حدد متابعة Continue

١١. خذ دقيقة لفحص إخراج وحدة التحكم.

لاحظ أن خاصية الاستثناء `Message` مضمنة في الإخراج الذي تم إنشاؤه بواسطة التطبيق الخاص بك:

```
∞
Exception caught in Process1: Attempted to
divide by zero.
Exit program
```

التقاط نوع استثناء معين specific exception

الآن بعد أن عرفت نوع الاستثناء الذي يجب التقاطه، يمكنك تحديث العبارة `catch` للتعامل مع نوع الاستثناء المحدد هذا.

١. حدّث الأسلوب `Process1` على النحو التالي:

```
static void Process1()
{
    try
    {
        WriteMessage();
    }
    catch (DivideByZeroException ex)
    {
        Console.WriteLine($"Exception caught in
Process1: {ex.Message}");
    }
}
```

٢. احفظ التعليمات البرمجية، ثم ابدأ جلسة تصحيح الأخطاء debug

٣. لاحظ أن التطبيق المحدث يبلغ عن نفس الرسائل إلى وحدة التحكم.

على الرغم من أن الرسائل التي تم الإبلاغ عنها هي نفسها، إلا أن هناك اختلافا مهماً، سيلتقط الأسلوب `Process1` استثناءات من النوع المحدد، الذي هو مستعد للتعامل معه فقط.

٤. لإنشاء نوع استثناء مختلف، قم بتحديث الأسلوب `WriteMessage` كما يلي:

```
static void WriteMessage()
{
    double float1 = 3000.0;
    double float2 = 0.0;
    int number1 = 3000;
    int number2 = 0;
    byte smallNumber;

    Console.WriteLine(float1 / float2);
    // Console.WriteLine(number1 / number2);
    checked
    {
        smallNumber = (byte)number1;
    }
}
```

٥. لاحظ استخدام العبارة `checked`

عند إجراء عمليات حسابية للنوع المتكامل integral type تقوم بتعيين قيمة نوع متكامل إلى نوع متكامل آخر، تعتمد النتيجة على سياق التحقق من الفائض، في سياق `checked` ينجح التحويل إذا كانت قيمة المصدر ضمن نطاق نوع الوجهة، وإلا، فسيتم طرح `OverflowException` في سياق غير محدد، ينجح التحويل دائمًا، ويستمر كما يلي:

- إذا كان نوع المصدر source type أكبر من نوع الوجهة destination type فسيتم اقتطاع قيمة المصدر عن طريق تجاهل البتات bits "الإضافية" الأكثر أهمية، ثم يتم التعامل مع النتيجة كقيمة لنوع الوجهة.
- إذا كان نوع المصدر source type أصغر من نوع الوجهة destination type فإن قيمة المصدر إما ممتدة بالإشارة أو ممتدة صفرياً، بحيث تكون بنفس حجم نوع الوجهة، يتم استخدام ملحق التوقيع

إذا تم توقيع نوع المصدر؛ يتم استخدام الامتداد الصفري إذا كان نوع المصدر غير موقع، ثم يتم التعامل مع النتيجة كقيمة لنوع الوجهة.

- إذا كان نوع المصدر هو نفس حجم نوع الوجهة، فسيتم التعامل مع قيمة المصدر كقيمة لنوع الوجهة.

ملاحظة

يتم التعامل مع حسابات النوع المتكامل غير الموجودة داخل كتلة checked كما لو كانت داخل كتلة التعليمات البرمجية unchecked

٦. احفظ التعليمات البرمجية، ثم ابدأ جلسة تصحيح الأخطاء debug

٧. لاحظ أنه يتم اكتشاف نوع الاستثناء الجديد بواسطة عبارة `catch` في عبارات المستوى الأعلى بدلاً من داخل الأسلوب `Process1`

يقوم التطبيق بطباعة الرسائل التالية إلى وحدة التحكم:

```
∞
An exception has occurred
Exit program
```

ملاحظة

لم يتم تنفيذ الكتلة catch في `Process1` هذا هو السلوك الذي أردته، التقاط الاستثناءات التي تكون التعليمات البرمجية مستعدًا للتعامل معها.

التقاط استثناءات متعددة في كتلة التعليمات البرمجية

في هذه المرحلة، قد تتساءل عما يحدث عند حدوث استثناءات متعددة في كتلة تعليمة برمجية واحدة، هل ستسفر التعليمات البرمجية catch هل سيكتشف كل استثناء عند حدوثه؟

١. حدّث الأسلوب `WriteMessage` على النحو التالي:

```
static void WriteMessage()
{
    double float1 = 3000.0;
    double float2 = 0.0;
    int number1 = 3000;
    int number2 = 0;
    byte smallNumber;

    Console.WriteLine(float1 / float2);
    Console.WriteLine(number1 / number2);
    checked
    {
        smallNumber = (byte)number1;
    }
}
```

٢. تعيين نقطة توقف داخل الأسلوب `WriteMessage` على سطر التعليمات البرمجية التالي:

```
Console.WriteLine(float1 / float2);
```

٣. احفظ التعليمات البرمجية، ثم ابدأ جلسة تصحيح الأخطاء.

٤. تنقل عبر التعليمات البرمجية سطراً واحدًا في كل مرة، ولاحظ ما يحدث بعد أن تعالج التعليمات البرمجية الاستثناء الأول.

عند حدوث الاستثناء الأول، يتم تمرير عنصر التحكم إلى العبارة الأولى catch التي يمكنها معالجة الاستثناء، لا يتم الوصول إلى التعليمات البرمجية التي من شأنها إنشاء الاستثناء الثاني أبدًا، وهذا يعني أن بعض التعليمات البرمجية لا يتم تنفيذها، ويمكن أن يؤدي ذلك إلى مشاكل خطيرة.

٥. خذ دقيقة للنظر في كيفية إدارة استثناءات متعددة، ومتى/لماذا قد لا تقوم التعليمات البرمجية بإدارة استثناءات متعددة.

لقد تعلمت سابقاً في هذه الوحدة أنه يجب اكتشاف الاستثناءات بالقرب من مكان حدوثها قدر الإمكان، مع مراعاة ذلك، يمكنك اختيار تحديث الأسلوب `WriteMessage` لالتقاط الاستثناءات، باستخدام try-catch على سبيل المثال:

```
static void WriteMessage()
{
    double float1 = 3000.0;
    double float2 = 0.0;
    int number1 = 3000;
    int number2 = 0;
    byte smallNumber;

    try
    {
        Console.WriteLine(float1 / float2);
        Console.WriteLine(number1 / number2);
    }
    catch (DivideByZeroException ex)
    {
        Console.WriteLine($"Exception caught in
WriteMessage: {ex.Message}");
    }
    checked
    {
        smallNumber = (byte)number1;
    }
}
```

يمكنك أيضاً تغليف التعليمات البرمجية التي تسبب OverflowException في منفصل try-catch داخل الأسلوب `WriteMessage`

```
checked
{
    try
    {
        smallNumber = (byte)number1;
    }
    catch (OverflowException ex)
    {
        Console.WriteLine($"Exception caught in
WriteMessage: {ex.Message}");
    }
}
```

٦. وفي أي ظروف سيكون من غير المرغوب فيه الحصول على استثناءات لاحقة؟

ضع في اعتبارك الحالة عندما يقوم الأسلوب (أو كتلة التعليمات البرمجية) بإكمال عملية مكونة من جزأين، افترض أن الجزء الثاني من العملية يعتمد على إكمال الجزء الأول، إذا تعذر إكمال الجزء الأول من العملية بنجاح، فلا داعي للاستمرار في الجزء الثاني من العملية، في هذه الحالة، من الأفضل غالبًا تقديم رسالة للمستخدم توضح حالة الخطأ، دون محاولة تنفيذ الجزء أو الأجزاء المتبقية من العملية الأكبر.

التقاط أنواع استثناء منفصلة separate exception في كتلة التعليمات البرمجية

هناك أوقات قد تتسبب فيها الاختلافات في بياناتك، في أنواع مختلفة من الاستثناءات.

١. امسح نقاط التوقف، ثم استبدل محتويات ملف Program.cs بالتعليمات البرمجية التالية:

```
// inputValues is used to store numeric values
entered by a user
string[] inputValues = new string[]{"three",
"9999999999", "0", "2" };

foreach (string inputValue in inputValues)
{
    int numValue = 0;
    try
    {
        numValue = int.Parse(inputValue);
    }
    catch (FormatException)
    {
```

```
        Console.WriteLine("Invalid readResult.
Please enter a valid number.");
    }
    catch (OverflowException)
    {
        Console.WriteLine("The number you
entered is too large or too small.");
    }
    catch(Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

٢. خذ دقيقة لمراجعة هذه التعليمات البرمجية.

أولا، تنشئ التعليمات البرمجية مصفوفة string تسمى inputValues تهدف البيانات الموجودة في المصفوفة إلى تمثيل قيم الإدخال التي أدخلها مستخدم، طلب منه إدخال قيم رقمية، اعتمادًا على القيمة التي تم إدخالها، قد تحدث أنواع استثناء مختلفة.

لاحظ أن التعليمات البرمجية تستخدم الأسلوب `int.Parse` لتحويل convert قيم "إدخال" string إلى أعداد صحيحة integers يتم وضع التعليمات `int.Parse` داخل كتلة try

٣. تعيين نقطة توقف على سطر التعليمات البرمجية التالي:

```
int numValue = 0;
```

٤. احفظ التعليمات البرمجية، ثم ابدأ جلسة تصحيح الأخطاء.

٥. تنقل عبر التعليمات سطراً تلو الآخر، ولاحظ أنه تم اكتشاف أنواع مختلفة من الاستثناءات.

خلاصة

فيما يلي بعض الأشياء المهمة التي يجب تذكرها من هذا الدرس:

- يجب تكوين العبارة `catch` لالتقاط نوع استثناء محدد، على سبيل المثال، نوع الاستثناء `DivideByZeroException`
- يمكن الوصول إلى خصائص كائن استثناء داخل الكتلة `catch` على سبيل المثال، يمكنك استخدام الخاصية `Message` لإعلام مستخدم التطبيق بوجود مشكلة.
- يمكنك تحديد بندين أو أكثر من `catch` عندما تحتاج إلى التقاط أكثر من نوع استثناء واحد.

٨ تمرين - إكمال نشاط تحدي catch لالتقاط استثناءات محددة

تستخدم تحديات التعليمات البرمجية، لتعزيز ما تعلمته ومساعدتك على اكتساب بعض الثقة قبل المتابعة.

تحدي التقاط استثناءات محددة Catch specific exceptions challenge

في هذا التحدي، يتم تزويدك بنموذج تعليمات برمجية، التي تنشئ عدة أنواع استثناء مختلفة، تحتوي كتلة الفردية try على التعليمات التي تنشئ الاستثناءات، يتم تضمين عبارات catch متعددة لمعالجة أنواع استثناءات محددة.

تحتاج إلى تحديث نموذج التعليمات البرمجية، بحيث يتم التقاط كل استثناء، وعرض رسالة الخطأ المقابلة إلى وحدة التحكم.

فيما يلي متطلبات هذا التحدي:

١. تأكد من أن ملف Program.cs يحتوي على نموذج التعليمات البرمجية التالي:

```
try
{
    int num1 = int.MaxValue;
    int num2 = int.MaxValue;
    int result = num1 + num2;
    Console.WriteLine("Result: " + result);

    string str = null;
    int length = str.Length;
    Console.WriteLine("String Length: " +
length);

    int[] numbers = new int[5];
    numbers[5] = 10;
```

```
    Console.WriteLine("Number at index 5: " +
numbers[5]);

    int num3 = 10;
    int num4 = 0;
    int result2 = num3 / num4;
    Console.WriteLine("Result: " + result2);
}
catch (OverflowException ex)
{
    Console.WriteLine("Error: The number is too
large to be represented as an integer." +
ex.Message);
}
catch (NullReferenceException ex)
{
    Console.WriteLine("Error: The reference is
null." + ex.Message);
}
catch (IndexOutOfRangeException ex)
{
    Console.WriteLine("Error: Index out of
range." + ex.Message);
}
catch (DivideByZeroException ex)
{
    Console.WriteLine("Error: Cannot divide by
zero." + ex.Message);
}

Console.WriteLine("Exiting program.");
```

٢. قم بتحديث التعليمات البرمجية، بحيث يتم عرض كل رسالة خطأ إلى وحدة التحكم عند حدوث هذا النوع من الاستثناء.

٣. تحقق من أن التعليمات البرمجية المحدثة، تطبع الرسائل التالية إلى وحدة التحكم:

```
Error: The number is too large to be represented as an integer. Arithmetic operation resulted in an overflow.
Error: The reference is null. Object reference not set to an instance of an object.
Error: Index out of range. Index was outside the bounds of the array.
Error: Cannot divide by zero. Attempted to divide by zero.
Exiting program.
```

٩ مراجعة حل تحدي catch التقاط استثناءات محددة

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الدرس السابق:

```
checked
{
    try
    {
        int num1 = int.MaxValue;
        int num2 = int.MaxValue;
        int result = num1 + num2;
        Console.WriteLine("Result: " + result);
    }
    catch (OverflowException ex)
    {
        Console.WriteLine("Error: The number is
too large to be represented as an integer. " +
ex.Message);
    }
}

try
{
    string? str = null;
    int length = str.Length;
    Console.WriteLine("String Length: " +
length);
}
catch (NullReferenceException ex)
{
    Console.WriteLine("Error: The reference is
null. " + ex.Message);
```

```
}

try
{
    int[] numbers = new int[5];
    numbers[5] = 10;
    Console.WriteLine("Number at index 5: " +
numbers[5]);
}
catch (IndexOutOfRangeException ex)
{
    Console.WriteLine("Error: Index out of
range. " + ex.Message);
}

try
{
    int num3 = 10;
    int num4 = 0;
    int result2 = num3 / num4;
    Console.WriteLine("Result: " + result2);
}
catch (DivideByZeroException ex)
{
    Console.WriteLine("Error: Cannot divide by
zero. " + ex.Message);
}

Console.WriteLine("Exiting program.");
```

عند تشغيل هذا التطبيق، سيقوم بطباعة الرسائل المطلوبة إلى وحدة التحكم كإخراج:

```
Error: The number is too large to be represented as an integer. Arithmetic operation resulted in an overflow.
Error: The reference is null. Object reference not set to an instance of an object.
Error: Index out of range. Index was outside the bounds of the array.
Error: Cannot divide by zero. Attempted to divide by zero.
Exiting program.
```

هذه التعليمة البرمجية هي مجرد "أحد الحلول الممكنة" نظراً لوجود العديد من الطرق لتحديث منطق معالجة الاستثناءات، طالما أنك حصلت على النتائج الصحيحة وفقًا لقواعد التحدي، فقد قمت بعمل رائع!

نهنئك في حال نجاحك! تابع لاختبار المعلومات في الدرس التالي.

هام

إذا كان لديك مشكلة في إكمال هذا التحدي، ربما يجب عليك مراجعة الدروس السابقة قبل المتابعة.

١٠ التحقق من المعرفة

١- ما الذي تحتوي عليه كتلة التعليمات البرمجية try؟

- التعليمات البرمجية التي تقوم بتنظيف الموارد، يتم تنفيذها سواء حدث استثناء أم لا
- التعليمات البرمجية التي تطرح كائن استثناء
- التعليمات البرمجية التي قد تسبب استثناء

٢- ما الغرض من كتلة التعليمات البرمجية catch؟

- لتنظيف أي موارد يتم تخصيصها في كتلة try
- لمعالجة الاستثناء، القيام بتسجيله أو تجاهله
- لتنفيذ التعليمات البرمجية التي قد تسبب استثناء

٣- متى يتم طرح استثناء IndexOutOfRangeException؟

- يتم طرح استثناء IndexOutOfRangeException عند محاولة تخزين قيمة من نوع محدد في مصفوفة من نوع آخر
- يتم طرح استثناء IndexOutOfRangeException عند محاولة تحويل غير صالح أو تحويل صريح
- يتم طرح استثناء IndexOutOfRangeException عند محاولة فهرسة مصفوفة خارج حدود المصفوفة

٤- متى يتم طرح استثناء ArrayTypeMismatchException؟

- يتم طرح استثناء ArrayTypeMismatchException عند محاولة تخزين قيمة من نوع محدد في مصفوفة من نوع آخر.
- يتم طرح استثناء ArrayTypeMismatchException عند محاولة الوصول إلى عضو على نوع قيمته فارغة.
- يتم طرح استثناء ArrayTypeMismatchException عند محاولة فهرسة مصفوفة خارج حدود المصفوفة.

٥- ما هو اسم الفئة الأساسية التي ترث منها كافة أنواع الاستثناءات المشتقة؟

- Object
- System
- Exception

٦- ما هي خاصية كائن الاستثناء التي يمكن استخدامها لتحديد مكان حدوث خطأ؟

- InnerException
- StackTrace
- TargetSite

٧. ما هو النهج الموصي به لاكتشاف الاستثناءات في C#؟

- التقاط أي نوع من الاستثناء دون تحديد وسيطة كائن.
- التقاط الاستثناءات التي تستطيع التعليمات البرمجية التعامل معها، معالجتها.
- التقاط الاستثناءات غير المشتقة من System.Exception فقط

٨. ما هو اسم نوع الاستثناء المحدد الذي يحدث عند محاولة القسمة على صفر في C#؟

- ArithmeticException
- InvalidCastException
- DivideByZeroException

راجع إجابتك

١ التعليمات البرمجية التي قد تسبب استثناء.

صحيح تحتوي الكتلة البرمجية try على التعليمات المحمية التي قد تسبب استثناء

٢ لمعالجة الاستثناء، القيام بتسجيله أو تجاهله.

صحيح تحتوي الكتلة البرمجية catch على التعليمات التي يتم تنفيذها عند اكتشاف استثناء، يمكن للكتلة catch معالجة الاستثناء أو تسجيله أو تجاهله

٣ يتم طرح استثناء IndexOutOfRangeException عند محاولة فهرسة مصفوفة خارج حدود المصفوفة

صحيح يتم طرح استثناء IndexOutOfRangeException عند إجراء محاولة للوصول إلى عنصر من مصفوفة أو مجموعة باستخدام فهرس خارج حدوده.

٤ يتم طرح استثناء ArrayTypeMismatchException عند محاولة تخزين قيمة من نوع محدد في مصفوفة من نوع آخر.

صحيح يتم طرح الاستثناء ArrayTypeMismatchException عند إجراء محاولة لتخزين عنصر من النوع الخطأ داخل مصفوفة

٥ Exception

صحيح في System.Exception هي الفئة الأساسية التي ترث منها جميع أنواع الاستثناءات المشتقة

٦ StackTrace

صحيح تحتوي الخاصية StackTrace على تتبع مكدس يمكن استخدامه لتحديد مكان حدوث خطأ

٧ التقاط الاستثناءات التي تستطيع التعليمات البرمجية التعامل معها، معالجتها.

صحيح النهج الموصي به هو التقاط الاستثناءات التي تعرف التعليمات البرمجية التعامل معها فقط

٨ DivideByZeroException

صحيح في C# يكون نوع الاستثناء المحدد الذي يحدث عند محاولة القسمة على صفر هو DivideByZeroException

١١ الملخص

كان هدفك هو اكتساب خبرة في تنفيذ معالجة الاستثناء في تطبيقات C# باستخدام Visual Studio Code

من خلال فحص خصائص أنواع الاستثناءات الشائعة وتجربة النمط try-catch اكتسبت خبرة في التقاط استثناءات وقت التشغيل، لقد استخدمت معالجة الاستثناء لالتقاط الاستثناءات في الأسلوب حيث حدثت، وعلى مستوى أقل من مكدس الاستدعاءات.

لقد مارست أيضاً استخدام بندين أو أكثر من catch لالتقاط أنواع استثناء مختلفة، مقترنة بكتلة تعليمات برمجية واحدة try

بدون القدرة على تنفيذ معالجة الاستثناء، لن تتمكن من تقديم تطبيقات C# مستقرة وموثوقة.

المصادر:

يمكنك العثور على معلومات إضافية حول خصائص الاستثناء هنا:
exception-class-and-properties أو system.exception

يمكنك العثور على معلومات إضافية حول الاستثناءات هنا:
exceptions

يمكنك العثور على معلومات إضافية حول استخدام أنواع استثناء معينة هنا:
specific-exceptions

يمكنك العثور على معلومات إضافية حول الأنماط try-catch-finally هنا:
try-catch-finally

الوحدة الرابعة

إنشاء استثناءات وطرحها في تطبيقات وحدة تحكم C#

تعرف على كيفية إنشاء كائنات الاستثناء exception objects وكيفية تخصيص خصائص كائن الاستثناء، وعملية إنشاء كائنات الاستثناء المخصصة custom exception objects وطرحها throwing والتقاط catching كائنات الاستثناء المخصصة في تطبيق C#

الأهداف التعليمية

- تعرف على كيفية إنشاء كائنات الاستثناء، وكيفية تخصيص إعدادات خصائصها.
- تطوير تطبيقات وحدة تحكم C# التي تقوم بإنشاء وطرح كائنات استثناء مخصصة.
- تطوير تطبيقات وحدة تحكم C# التي تلتقط كائنات الاستثناء المخصصة، وتدير معلومات خاصية الاستثناء.

محتويات الوحدة: -

١- مقدمة

٢- فحص كيفية إنشاء الاستثناءات وطرحها throw

٣- تمرين - إنشاء استثناء وطرحه throw

٤- تمرين - إكمال نشاط تحدي لإنشاء الاستثناءات وطرحها throw

٥- مراجعة حل تحدي إنشاء الاستثناءات وطرحها throw

٦- التحقق من المعرفة

٧- الملخص

١ المقدمة

يمكن طرح الاستثناءات بواسطة التعليمات البرمجية عند مواجهة مشكلة أو حالة خطأ، يتم إنشاء كائنات الاستثناء التي تصف الخطأ، ثم طرحها باستخدام الكلمة الأساسية throw عندما يتم طرح استثناء بواسطة التعليمات البرمجية، يبحث وقت التشغيل عن أقرب عبارة catch يمكنها معالجة الاستثناء.

لنفترض أنك تعمل على تطبيق معالجة بيانات لشركة، يعتمد التطبيق على قواعد العمل، والمواصفات لضمان اكتمال معالجة أمر الشراء، ومهام إدارة المخزون بشكل مناسب، بالإضافة إلى ذلك، يجب أن يستخدم التطبيق لغة معينة لإعلام المستخدم، عند مواجهة حالات شذوذ البيانات، وغيرها من المشكلات. يجب طرح استثناءات مخصصة وضبطها، وتنعكس في واجهة مستخدم التطبيق عند مواجهة المشكلات، توفر قواعد العمل إرشادات محددة في الفئات التالية:

- متطلبات إدخال البيانات للعمليات.
- معايير النجاح والفشل للعمليات.
- متطلبات ترتيب التسلسل للعمليات.
- الإبلاغ عن فشل العملية ومتطلبات التخفيف من المخاطر.

في هذه الوحدة، ستتعلم كيفية إنشاء كائنات استثناء exception objects وكيفية تخصيص خصائص كائن استثناء properties of an exception object وعملية إنشاء كائنات استثناء مخصصة، وطرحها throwing والتقاطها catching في تطبيق C#

في نهاية هذه الوحدة، ستتمكن من إنشاء عناصر استثناء وتخصيصها، وطرحها والتقاطها، التي تفي بمتطلبات تطبيقك.

٢ فحص كيفية إنشاء الاستثناءات وطرحها throw

يوفر .NET تسلسلاً هرمياً لفئات الاستثناء، التي يتم اشتقاقها من الفئة الأساسية `System.Exception` يمكن لتطبيقات C# إنشاء استثناءات من أي نوع استثناء وطرحها throw exceptions يمكن للمطورين أيضاً تخصيص كائنات الاستثناء بمعلومات خاصة بالتطبيق، عن طريق تعيين قيم الخصائص.

ملاحظة

يركز هذا الدرس على إنشاء الاستثناءات وطرحها، وتخصيص كائنات الاستثناءات، إنشاء فئات استثناء مخصصة خارج نطاق هذا الدرس.

إنشاء كائن استثناء exception object

يعد إنشاء استثناءات وطرحها Creating and throwing exceptions من داخل التعليمات البرمجية، جانباً مهماً من جوانب برمجة C# تساعدك القدرة على إنشاء استثناء استجابة لحالة أو مشكلة أو خطأ معين، على ضمان استقرار تطبيقك.

يعتمد نوع الاستثناء الذي تقوم بإنشائه على مشكلة الترميز، ويجب أن يتطابق مع الغرض المقصود من الاستثناء قدر الإمكان.

على سبيل المثال، افترض أنك تقوم بإنشاء أسلوب اسمه `GraphData` يقوم بتحليل البيانات، يتلقى الأسلوب مصفوفة بيانات كمعلمة إدخال، يتوقع الأسلوب أن تكون بيانات الإدخال في نطاق معين، إذا كان الأسلوب يتلقى بيانات خارج النطاق المتوقع، فإنه ينشئ ويطرح استثناء من النوع `ArgumentException` سيتم التعامل مع الاستثناء في مكان ما أسفل مكدس الاستدعاءات call stack بواسطة التعليمات البرمجية المسؤولة عن توفير البيانات.

فيما يلي بعض أنواع الاستثناءات الشائعة التي قد تستخدمها عند إنشاء استثناء:

- `ArgumentException` أو `ArgumentNullException` استخدم هذا النوع من الاستثناء عند استدعاء أسلوب أو منشئ بقيمة وسيطة غير صحيحة أو مرجع فارغ invalid argument value or null reference
- `InvalidOperationException` استخدم هذا النوع من الاستثناء عندما لا تدعم ظروف تشغيل إحدى الأساليب الإكمال الناجح، لاستدعاء أسلوب معين method call
- `NotSupportedException` استخدم هذا النوع من الاستثناء عندما تكون عملية أو ميزة غير معتمدة operation or feature is not supported
- `IOException` استخدم هذا النوع من الاستثناء عند فشل عملية إدخال/إخراج input/output operation fails
- `FormatException` استخدم هذا النوع من الاستثناء عندما يكون تنسيق سلسلة أو بيانات غير صحيح format of a string or data is incorrect

يتم استخدام الكلمة الأساسية `new` لإنشاء مثيل للاستثناء، على سبيل المثال، يمكنك إنشاء مثيل لنوع الاستثناء `ArgumentException` كما يلي:

```
ArgumentException invalidArgumentException = new ArgumentException();
```

تكوين الاستثناءات المخصصة وطرحها Configure and throw customized exceptions

تتضمن عملية طرح كائن استثناء throwing an exception object إنشاء مثيل لفئة مشتقة من الاستثناء، وتكوين خصائص الاستثناء اختيارياً، ثم طرح الكائن باستخدام الكلمة الأساسية `throw`

غالباً ما يكون من المفيد تخصيص استثناء بمعلومات سياقية قبل طرحه، يمكنك توفير معلومات خاصة بالتطبيق داخل كائن استثناء، عن طريق تكوين خصائصه، على سبيل المثال، تنشئ التعليمات البرمجية التالية كائن استثناء

يسمى `invalidArgumentException` بخاصية مخصصة `Message` ثم تطرح الاستثناء:

```
ArgumentException invalidArgumentException = new ArgumentException("ArgumentException: The 'GraphData' method received data outside the expected range.");
throw invalidArgumentException;
```

ملاحظة

خاصية الاستثناء Message للقراءة فقط، لذلك، يجب تعيين خاصية مخصصة Message عند إنشاء مثيل للكائن.

عند تخصيص كائن استثناء، من المهم توفير رسائل خطأ واضحة، تصف المشكلة وكيفية حلها، يمكنك أيضاً تضمين معلومات إضافية مثل تتبع المكدس stack traces ورموز الأخطاء، لمساعدة المستخدمين على تصحيح المشكلة.

يمكن أيضاً إنشاء كائن استثناء مباشرة داخل عبارة throw على سبيل المثال:

```
throw new FormatException("FormatException: Calculations in process XYZ have been cancelled due to invalid data format.");
```

يجب معرفة بعض الاعتبارات، عند طرح استثناء تتضمن:

- يجب أن توضح خاصية Message سبب الاستثناء، ومع ذلك، لا ينبغي إدراج المعلومات الحساسة أو التي تمثل مشكلة أمنية في نص الرسالة.
- غالبًا ما يتم استخدام خاصية StackTrace لتتبع أصل الاستثناء، تحتوي خاصية السلسلة هذه على اسم الأساليب الموجودة في مكدس الاستدعاءات call stack الحالي، بالإضافة إلى اسم الملف file name ورقم السطر line number في كل أسلوب مرتبط بالاستثناء، يتم إنشاء كائن StackTrace تلقائيًا بواسطة وقت تشغيل اللغة العامة

(CLR) من نقطة عبارة throw يجب طرح الاستثناءات من النقطة التي يجب أن يبدأ فيها تتبع المكدس

متى يتم طرح استثناء throw an exception

يجب أن تطرح الأساليب استثناءً كلما لم تتمكن من إكمال الغرض المقصود منها، يجب أن يستند الاستثناء الذي تم طرحه إلى الاستثناء الأكثر تحديدًا، المتوفر، والذي يناسب شروط الخطأ.

ضع في اعتبارك سيناريو يعمل فيه مطور على تطبيق يقوم بتنفيذ عملية تجارية، تعتمد عملية العمل على إدخال المستخدم، إذا لم يتطابق الإدخال مع نوع البيانات المتوقع، فإن الأسلوب الذي ينفذ عملية العمل ينشئ ويطرح استثناء، يمكن تكوين كائن الاستثناء بمعلومات خاصة بالتطبيق في قيم الخاصية، يوضح نموذج التعليمات البرمجية التالي هذا السيناريو:

```
string[][] userEnteredValues = new string[][]
{
        new string[] { "1", "two", "3"},
        new string[] { "0", "1", "2"}
};

foreach (string[] userEntries in
userEnteredValues)
{
    try
    {
        BusinessProcess1(userEntries);
    }
    catch (Exception ex)
    {
        if
(ex.StackTrace.Contains("BusinessProcess1") &&
(ex is FormatException))
        {
            Console.WriteLine(ex.Message);
        }
```

```
    }
}

static void BusinessProcess1(string[]
userEntries)
{
    int valueEntered;

    foreach (string userValue in userEntries)
    {
        try
        {
            valueEntered =
int.Parse(userValue);

            // completes required calculations
based on userValue
            // ...
        }
        catch (FormatException)
        {
            FormatException
invalidFormatException = new
FormatException("FormatException: User input
values in 'BusinessProcess1' must be valid
integers");
            throw invalidFormatException;
        }
    }
}
```

في هذا النموذج من التعليمات البرمجية، تستدعي عبارات المستوى الأعلى top-level statements الأسلوب `BusinessProcess1` وتمرير مصفوفة سلسلة تحتوي على قيم أدخلها المستخدم، يتوقع الأسلوب `BusinessProcess1` قيم إدخال المستخدم التي يمكن تحويلها إلى عدد صحيح، عندما يواجه الأسلوب بيانات بتنسيق غير صالح، فإنه ينشئ مثيلاً

لنوع الاستثناء FormatException باستخدام خاصية Message مخصصة، ثم يطرح الأسلوب الاستثناء، يتم التقاط الاستثناء في عبارات المستوى الأعلى ككائن يسمى ex يتم فحص خصائص الكائن ex قبل عرض رسالة الاستثناء للمستخدم، أولاً، تفحص التعليمات البرمجية الخاصية StackTrace لمعرفة ما إذا كانت تحتوي على "BusinessProcess1" ثانياً، يتم التحقق من أن كائن الاستثناء ex من النوع FormatException

إعادة طرح الاستثناءات Re-throwing exceptions

بالإضافة إلى طرح استثناء جديد، يمكن استخدام throw لإعادة طرح استثناء من داخل كتلة التعليمات البرمجية catch في هذه الحالة لا تأخذ throw معامل استثناء.

```
catch (Exception ex)
{
    // handle or partially handle the exception
    // ...

    // re-throw the original exception object
for further handling down the call stack
    throw;
}
```

عند إعادة طرح استثناء، يتم استخدام كائن الاستثناء الأصلي، لذلك لا تفقد أي معلومات حول الاستثناء، إذا كنت تريد إنشاء كائن استثناء جديد يغلف الاستثناء الأصلي، فيمكنك تمرير الاستثناء الأصلي كوسيطة إلى مُنشئ كائن الاستثناء الجديد، على سبيل المثال:

```
catch (Exception ex)
{
    // handle or partially handle the exception
    // ...

    // create a new exception object that wraps
the original exception
    throw new ApplicationException("An error
occurred", ex);
}
```

بالنسبة لسيناريو تطبيق "BusinessProcess1" ضع في اعتبارك التحديثات التالية:

- تم تحديث الأسلوب `BusinessProcess1` ليشمل تفاصيل إضافية، يواجه `BusinessProcess1` الآن مشكلتين، ويجب إنشاء استثناء لكل مشكلة.
- تم تحديث عبارات المستوى الأعلى top-level statements تستدعي عبارات المستوى الأعلى الآن الأسلوب `OperatingProcedure1` يستدعي الأسلوب `OperatingProcedure1` الأسلوب `BusinessProcess1` داخل كتلة التعليمات البرمجية try
- الأسلوب `OperatingProcedure1` قادر على معالجة أحد أنواع الاستثناءات، ومعالجة الآخر جزئياً، بمجرد معالجة الاستثناء الذي تمت معالجته جزئياً، يجب أن يقوم `OperatingProcedure1` بإعادة طرح الاستثناء الأصلي.

يوضح نموذج التعليمات البرمجية التالي السيناريو المحدث:

```
try
{
    OperatingProcedure1();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
    Console.WriteLine("Exiting application.");
}

static void OperatingProcedure1()
{
    string[][] userEnteredValues = new string[][]
    {
        new string[] { "1", "two", "3"},
        new string[] { "0", "1", "2"}
    };
```

```
    foreach(string[] userEntries in
userEnteredValues)
    {
        try
        {
            BusinessProcess1(userEntries);
        }
        catch (Exception ex)
        {
            if
(ex.StackTrace.Contains("BusinessProcess1"))
            {
                if (ex is FormatException)
                {

Console.WriteLine(ex.Message);

Console.WriteLine("Corrective action taken in
OperatingProcedure1");
                }
                else if (ex is
DivideByZeroException)
                {

Console.WriteLine(ex.Message);
                    Console.WriteLine("Partial
correction in OperatingProcedure1 - further
action required");

                // re-throw the original exception
                    throw;
                }
                else
                {
                    // create a new exception
object that wraps the original exception
```

```
                    throw new
ApplicationException("An error occurred - ", ex);
                }
            }
        }

    }
}

static void BusinessProcess1(string[] userEntries)
{
    int valueEntered;

    foreach (string userValue in userEntries)
    {
        try
        {
            valueEntered = int.Parse(userValue);

            checked
            {
             int calculatedValue = 4 / valueEntered;
            }
        }
        catch (FormatException)
        {
            FormatException
invalidFormatException = new
FormatException("FormatException: User input
values in 'BusinessProcess1' must be valid
integers");
            throw invalidFormatException;
        }
        catch (DivideByZeroException)
        {
            DivideByZeroException
unexpectedDivideByZeroException = new
```

```
DivideByZeroException("DivideByZeroException:
Calculation in 'BusinessProcess1' encountered
an unexpected divide by zero");
            throw
unexpectedDivideByZeroException;

        }
    }
}
```

ينتج عن نموذج التعليمات البرمجية المحدث الإخراج التالي:

```
FormatException: User input values in
'BusinessProcess1' must be valid integers
Corrective action taken in OperatingProcedure1
DivideByZeroException: Calculation in
'BusinessProcess1' encountered an unexpected
divide by zero
Partial correction in OperatingProcedure1 -
further action required
DivideByZeroException: Calculation in
'BusinessProcess1' encountered an unexpected
divide by zero
Exiting application.
```

أشياء يجب تجنبها عند طرح استثناءات

تحدد القائمة التالية الممارسات التي يجب تجنبها عند طرح الاستثناءات:

- لا تستخدم الاستثناءات لتغيير تدفق البرنامج كجزء من التنفيذ العادي. استخدم الاستثناءات للإبلاغ عن حالات الخطأ ومعالجتها.
- لا يجب إرجاع الاستثناءات كقيمة إرجاع أو معلمة بدلاً من طرحها.
- لا تقم بطرح `System.IndexOutOfRangeException`، `System.NullReferenceException`، `System.Exception`، `System.SystemException`، عن قصد من التعليمات البرمجية المصدر.

- لا تقم بإنشاء استثناءات يمكن طرحها في وضع تصحيح الأخطاء ولكن ليس وضع الإصدار release mode لتحديد أخطاء وقت التشغيل أثناء مرحلة التطوير، استخدم `Debug.Assert` بدلاً من ذلك.

ملاحظة

يعد أسلوب Debug.Assert أداة لاكتشاف الأخطاء المنطقية أثناء التطوير، بشكل افتراضي، يعمل الأسلوب Debug.Assert فقط في بنيات التصحيح، يمكنك استخدام Debug.Assert في جلسات تصحيح الأخطاء للتحقق من وجود شرط يجب ألا يحدث أبدًا، يأخذ الأسلوب معلمتين: شرط منطقي للتحقق منه، ورسالة سلسلة اختيارية لعرضها إذا كان الشرط خاطئاً false لا ينبغي استخدام Debug.Assert بدلاً من طرح استثناء، وهو طريقة للتعامل مع المواقف الاستثنائية أثناء التنفيذ العادي للتعليمات البرمجية، يجب عليك استخدام Debug.Assert لالتقاط الأخطاء التي يجب ألا تحدث أبدًا، واستخدام الاستثناءات لمعالجة الأخطاء التي قد تحدث أثناء التنفيذ العادي لبرنامجك.

خلاصة

فيما يلي بعض الأشياء المهمة التي يجب تذكرها من هذا الدرس:

- عند إنشاء استثناء وطرحه، يجب أن يتطابق نوع الاستثناء مع الغرض المقصود من الاستثناء قدر الإمكان.
- لطرح استثناء throw an exception يمكنك إنشاء مثيل لفئة مشتقة من الاستثناء، وتكوين خصائصه، ثم استخدام الكلمة الأساسية throw
- عند إنشاء كائن استثناء exception object من المهم توفير رسائل خطأ واضحة، ومعلومات إضافية لمساعدة المستخدمين على تصحيح المشكلة.

اختبر معلوماتك

١. ما هو الغرض من خاصية StackTrace لكائن الاستثناء؟

- لتتبع أصل الاستثناء track the origin of the exception
- لتكوين خصائص الاستثناء configure the properties of the exception
- لطرح كائن الاستثناء throw the exception object

٢. أي من أسطر التعليمات البرمجية التالية يوفر مثالاً لتخصيص استثناء بمعلومات سياقية؟

- `throw new FormatException();`
- `ArgumentException invalidArgumentException = new ArgumentException("ArgumentException: The 'GraphData' method received data outside the expected range.");`
- `ArgumentNullException argumentNullException = new ArgumentNullException();`

راجع إجابتك

١ لتتبع أصل الاستثناء track the origin of the exception

صحيح غالباً ما تستخدم الخاصية StackTrace لتتبع أصل الاستثناء

٢
```
ArgumentException invalidArgumentException = new ArgumentException("ArgumentException: The 'GraphData' method received data outside the expected range.");
```

صحيح يوضح نموذج التعليمات البرمجية هذا كيفية إنشاء كائن استثناء بخاصية Message مخصصة

٣ تمرين - إنشاء استثناء وطرحه throw

غالباً ما يحتاج المطورون إلى إنشاء استثناءات وطرحها من داخل أسلوب، ثم التقاط هذه الاستثناءات بشكل أكبر أسفل مكدس الاستدعاءات حيث يمكن التعامل معها، تساعدك معالجة الاستثناء على ضمان استقرار تطبيقاتك.

في هذا التمرين، ستبدأ بنموذج تطبيق يتضمن حالة خطأ محتملة داخل أسلوب مستدعي، سيطرح الأسلوب المحدث استثناء عند اكتشاف المشكلة، ستتم معالجة الاستثناء في كتلة catch من التعليمات التي تستدعي الأسلوب، والنتيجة هي تطبيق يوفر تجربة مستخدم أفضل.

إنشاء مشروع تعليمات برمجية جديدة

الخطوة الأولى هي إنشاء مشروع تعليمات برمجية يمكنك استخدامه أثناء هذا الدرس:

١. افتح مثيلاً جديدًا Open a new instance من Visual Studio Code

٢. في القائمة ملف File حدد فتح مجلد Open Folder

٣. في مربع الحوار فتح مجلد، انتقل إلى مجلد سطح مكتب Desktop Windows

٤. في مربع الحوار فتح مجلد، حدد مجلد جديد New folder

٥. قم بتسمية المجلد الجديد ThrowExceptions101 ثم حدد Select Folder

٦. في القائمة المحطة الطرفية Terminal حدد محطة طرفية جديدة New Terminal

ستستخدم أمر .NET CLI. لإنشاء تطبيق وحدة تحكم جديد new console app

٧. في موجه أوامر لوحة TERMINAL أدخل الأمر التالي:

```
dotnet new console
```

٨. أغلق لوحة TERMINAL

مراجعة نموذج التطبيق

استخدم الخطوات التالية لتحميل نموذج التطبيق ومراجعته.

١. افتح ملف Program.cs

٢. في القائمة عرض View حدد لوحة الأوامر Command Palette

٣. في موجه الأوامر command prompt أدخل **net: g** ثم حدد **.NET: Generate Assets for Build and Debug**

٤. استبدل محتويات ملف Program.cs بالتعليمات البرمجية التالية:

```
// Prompt the user for the lower and upper
bounds
Console.Write("Enter the lower bound: ");
int lowerBound = int.Parse(Console.ReadLine());

Console.Write("Enter the upper bound: ");
int upperBound = int.Parse(Console.ReadLine());

decimal averageValue = 0;

// Calculate the sum of the even numbers
between the bounds
averageValue = AverageOfEvenNumbers(lowerBound,
upperBound);

// Display the value returned by
AverageOfEvenNumbers in the console
Console.WriteLine($"The average of even numbers
between {lowerBound} and {upperBound} is
{averageValue}.");

// Wait for user input
Console.ReadLine();
```

```
static decimal AverageOfEvenNumbers(int
lowerBound, int upperBound)
{
    int sum = 0;
    int count = 0;
    decimal average = 0;

    for (int i = lowerBound; i <= upperBound; i++)
    {
        if (i % 2 == 0)
        {
            sum += i;
            count++;
        }
    }

    average = (decimal)sum / count;

    return average;
}
```

٥. خذ دقيقة لمراجعة التعليمات البرمجية.

لاحظ أن التطبيق ينفذ المهام التالية:

- تستخدم عبارات المستوى الأعلى عبارات `Console.ReadLine()` للحصول على قيم ل `lowerBound` and `upperBound`
- تقوم عبارات المستوى الأعلى بتمرير `lowerBound`, `upperBound` كوسيطات عند استدعاء الأسلوب `AverageOfEvenNumbers`

- يقوم الأسلوب `AverageOfEvenNumbers` بتنفيذ المهام التالية:
  - يعلن المتغيرات المحلية المستخدمة في العمليات الحسابية
  - يستخدم حلقة for لجمع الأرقام الزوجية بين `lowerBound`, `upperBound` يتم تخزين المبلغ في `sum`
  - يحسب عدد الأرقام المضمنة في المجموع sum يتم تخزين العدد في `count`
  - يخزن متوسط الأرقام التي تم جمعها في متغير يسمى `average` يتم إرجاع قيمة `average`
- تقوم عبارات المستوى الأعلى بطباعة القيمة التي تم إرجاعها بواسطة `AverageOfEvenNumbers` إلى وحدة التحكم ثم تقوم بإيقاف التنفيذ مؤقتًا.

تكوين بيئة تتبع الأخطاء

يقرأ نموذج التطبيق إدخال المستخدم من وحدة التحكم، لا تدعم لوحة DEBUG CONSOLE قراءة الإدخال من وحدة التحكم، تحتاج إلى تحديث ملف launch.json قبل أن تتمكن من تشغيل هذا التطبيق في مصحح الأخطاء.

١. استخدم قائمة EXPLORER لفتح ملف launch.json

٢. في ملف launch.json قم بتحديث السمة console كما يلي:

```
// For more information about the 'console' field,
see https://aka.ms/VSCode-CS-LaunchJson-Console
"console":"integratedTerminal",
```

القيمة الافتراضية للسمة console هي internalConsole والتي تتوافق مع لوحة DEBUG CONSOLE لسوء الحظ، لا تدعم لوحة DEBUG CONSOLE إدخال وحدة التحكم، يتوافق integratedTerminal الإعداد مع لوحة TERMINAL التي تدعم إدخال وحدة التحكم والإخراج.

٣. احفظ التغييرات التي أجريتها على ملف launch.json ثم أغلق الملف.

٤. في قائمة Run حدد Start Debugging

٥. قم بالتبديل إلى لوحة TERMINAL

٦. في المطالبة "الحد الأدنى lower bound" أدخل 3

٧. في المطالبة "الحد الأعلى upper bound" أدخل 11

٨. لاحظ أن التطبيق يعرض الرسالة التالية ثم يتوقف مؤقتاً:

```
The average of even numbers between 3 and 11 is 7.
```

٩. للخروج من التطبيق، اضغط على مفتاح الإدخال Enter

طرح استثناء في الأسلوب AverageOfEvenNumbers

يتوقع الأسلوب `AverageOfEvenNumbers` حدًا أعلى أكبر من الحد الأدنى، يحدث خطأ `DivideByZero` إذا كان الحد الأدنى أكبر من الحد الأعلى أو يساويه.

تحتاج إلى تحديث الأسلوب `AverageOfEvenNumbers` لطرح استثناء، عندما يكون الحد الأدنى أكبر من الحد الأعلى أو مساوياً له.

١. خذ دقيقة للنظر في الطريقة التي تريد بها معالجة المشكلة.

أحد الخيارات هو التفاف/تغليف حساب `average` داخل كتلة `try` والتقاط `catch` استثناء `DivideByZero` عند حدوثه، يمكنك إعادة طرح الاستثناء rethrow ثم التعامل معه في تعليمات الاتصال.

خيار آخر وهو تقييم معلمات الإدخال قبل بدء العمليات الحسابية، إذا كان الحد الأدنى `lowerBound` أكبر من أو يساوي الحد العلوي `upperBound` فيمكنك طرح استثناء.

يعد تقييم المعلمات Evaluating the parameters وطرح استثناء throwing an exception قبل بدء الحسابات هو الخيار الأفضل.

٢. ضع في اعتبارك نوع الاستثناء الذي يجب طرحه.

هناك نوعان من الاستثناءات التي تتوافق مع المشكلة:

- `ArgumentOutOfRangeException` يجب طرح نوع استثناء `ArgumentOutOfRangeException` فقط عندما تكون قيمة الوسيطة خارج نطاق القيم المسموح بها، كما هو محدد بواسطة الأسلوب الذي تم استدعاؤه، على الرغم من أن `AverageOfEvenNumbers` لا يحدد بشكل صريح نطاقًا مسموحًا به لـ `lowerBound` أو `upperBound` فإن قيمة `lowerBound` تشير ضمنًا إلى النطاق المسموح به لـ `upperBound`
- `InvalidOperationException` يجب طرح نوع الاستثناء `InvalidOperationException` فقط عندما لا تدعم ظروف تشغيل إحدى الأساليب الإكمال الناجح لاستدعاء أسلوب محدد، في هذه الحالة، يتم تحديد ظروف التشغيل من خلال معلمات الإدخال للأسلوب.

عندما يكون لديك نوعان أو أكثر من أنواع الاستثناءات للاختيار من بينها، حدد نوع الاستثناء الذي يناسب المشكلة بشكل أدق، في هذه الحالة، يتم محاذاة نوعي الاستثناء إلى المشكلة بالتساوي.

عندما يكون لديك نوعان أو أكثر من الاستثناءات المتوافقة مع المشكلة بشكل متساوٍ، حدد نوع الاستثناء الأضيق نطاقًا، يتم تحديد نطاق نوع الاستثناء `ArgumentOutOfRangeException` إلى الوسيطات التي تم تمريرها إلى الأسلوب، يتم تحديد نطاق نوع الاستثناء `InvalidOperationException` وفقًا لظروف تشغيل الأسلوب، في هذه الحالة، يكون نوع الاستثناء `ArgumentOutOfRangeException` ذو نطاق أضيق من نوع الاستثناء `InvalidOperationException`

يجب أن يطرح الأسلوب AverageOfEvenNumbers استثناء ArgumentOutOfRangeException

٣. في الجزء العلوي من الأسلوب AverageOfEvenNumbers للكشف عن مشكلة الحد الأعلى، قم بتحديث التعليمات البرمجية كما يلي:

```
if (lowerBound >= upperBound)
{

}

int sum = 0;
```

٤. لإنشاء استثناء ArgumentOutOfRangeException وطرحه، قم بتحديث كتلة if كما يلي:

```
if (lowerBound >= upperBound)
{
    throw new
ArgumentOutOfRangeException("upperBound",
"ArgumentOutOfRangeException: upper bound must
be greater than lower bound.");
}
```

يقوم سطر التعليمات البرمجية هذا بتهيئة مثيل جديد للفئة ArgumentOutOfRangeException باسم معلمة الإدخال التي تتسبب في الاستثناء ورسالة خطأ محددة.

التقاط الاستثناء في التعليمات البرمجية للاستدعاء Catch the exception in the calling code

كلما أمكن، يجب اكتشاف الاستثناءات على مستوى مكدس الاستدعاءات call stack حيث يمكن معالجتها handled في نموذج التطبيق هذا، يمكن إدارة معلمات الأسلوب AverageOfEvenNumbers في أسلوب الاستدعاء (عبارات المستوى الأعلى the top-level statements)

١. مرر لأعلى إلى عبارات المستوى الأعلى top-level statements

٢. لإحاطة استدعاء الأسلوب AverageOfEvenNumbers وعبارة Console.WriteLine داخل كتلة try قم بتحديث التعليمات كما يلي:

```
try
{
    // Calculate the sum of the even numbers
between the bounds
    averageValue =
AverageOfEvenNumbers(lowerBound, upperBound);

    // Display the result to the user
    Console.WriteLine($"The average of even
numbers between {lowerBound} and {upperBound}
is {averageValue}.");
}
```

٣. لإنشاء العبارة catch المقترنة، أدخل التعليمات التالية:

```
catch(ArgumentOutOfRangeException ex)
{

}
```

٤. خذ دقيقة للنظر في كيفية التعامل مع الاستثناء.

لمعالجة هذا الاستثناء، يجب أن تقوم التعليمات البرمجية بما يلي:

- شرح المشكلة للمستخدم.
- الحصول على قيمة جديدة ل upperBound
- اتصال `AverageOfEvenNumbers` باستخدام الجديد upperBound
- استمر إلى catch لالتقاط الاستثناء، إذا كان الحد العلوي الجديد `upperBound` المقدم أقل من الحد الأدنى `lowerBound` أو يساويه.

الاستمرار في catch التقاط الاستثناء يتطلب حلقة تكرار، نظراً لأنك تريد استدعاء الأسلوب AverageOfEvenNumbers مرة واحدة على الأقل، فيجب استخدام حلقة do

٥. لتضمين كتل `try` and `catch` داخل حلقة do قم بتحديث التعليمات البرمجية كما يلي:

```
do
{
    try
    {
        // Calculate the sum of the even
numbers between the bounds
        averageValue =
AverageOfEvenNumbers(lowerBound, upperBound);

        // Display the result to the user
        Console.WriteLine($"The average of even
numbers between {lowerBound} and {upperBound}
is {averageValue}.");
    }
    catch (ArgumentOutOfRangeException ex)
    {

    }
}
```

مطلوب تعبير while لتعريف شرط الخروج من حلقة تكرار do من الصعب تحديد الشرط قبل تعريف محتويات كتلة التعليمات البرمجية do سيساعدك إكمال كتلة catch على تحديد تعبير while المطلوب.

٦. لشرح المشكلة للمستخدم والحصول على upperBound جديد، قم بتحديث كتلة التعليمات البرمجية catch كما يلي:

```
catch (ArgumentOutOfRangeException ex)
{
   Console.WriteLine("An error has occurred.");
   Console.WriteLine(ex.Message);
   Console.WriteLine($"The upper bound must be
greater than {lowerBound}");
   Console.Write($"Enter a new upper bound: ");
    upperBound = int.Parse(Console.ReadLine());
}
```

تصف الكتلة البرمجية catch المحدثة، المشكلة، وتتطلب من المستخدم إدخال حد أعلى جديد new upper bound ومع ذلك، ماذا لو لم يكن لدى المستخدم قيمة حد أعلى صالحة لإدخالها؟ ماذا لو احتاج المستخدم إلى إنهاء التكرار الحلقي بدلاً من إدخال قيمة؟

٧. لتزويد المستخدم بخيار للخروج من الحلقة بدلاً من إدخال حد أعلى جديد، قم بتحديث كتلة catch كما يلي:

```
catch (ArgumentOutOfRangeException ex)
{
    Console.WriteLine("An error has
occurred.");
    Console.WriteLine(ex.Message);
    Console.WriteLine($"The upper bound must be
greater than {lowerBound}");
    Console.Write($"Enter a new upper bound (or
enter Exit to quit): ");
    string? userResponse = Console.ReadLine();
    if
(userResponse.ToLower().Contains("exit"))
    {

    }
    else
    {
        upperBound = int.Parse(userResponse);
    }
}
```

تتضمن كتلة catch المحدثة مسارين، مسار إنهاء "exit" ومسار حد أعلى جديد "new upper bound"

٨. خذ دقيقة للنظر في تعبير while المطلوب للحلقة do

إذا أدخل المستخدم عند المطالبة "Exit" يجب أن تخرج التعليمات من حلقة التكرار، إذا أدخل المستخدم حد أعلى جديداً، يجب أن تستمر الحلقة، يمكن استخدام تعبير while ليقيم قيمة منطقية، على سبيل المثال:

```
while (exit == false);
```

سينشئ التعبير while المقترح السلوك التالي:

- ستستمر حلقة do في التكرار طالما أن القيمة المنطقية exit تساوي false
- ستتوقف الحلقة do عن التكرار بمجرد أن تكون القيمة المنطقية exit مساوياً true

٩. لإنشاء مثيل لمتغير منطقي Boolean variable يسمى exit واستخدام exit لتعيين شرط الخروج من تكرار do قم بتحديث التعليمات البرمجية كما يلي:

```
bool exit = false;

do
{
    try
    {
        // Calculate the sum of the even
numbers between the bounds
        averageValue =
AverageOfEvenNumbers(lowerBound, upperBound);

        // Display the result to the user
        Console.WriteLine($"The average of even
numbers between {lowerBound} and {upperBound} is
{averageValue}.");

        exit = true;
    }
    catch (ArgumentOutOfRangeException ex)
```

```
    {
        Console.WriteLine("An error has occurred.");
        Console.WriteLine(ex.Message);
        Console.WriteLine($"The upper bound
must be greater than {lowerBound}");
        Console.Write($"Enter a new upper bound
(or enter Exit to quit): ");
        string? userResponse = Console.ReadLine();
        if
(userResponse.ToLower().Contains("exit"))
        {
            exit = true;
        }
        else
        {
            exit = false;
            upperBound = int.Parse(userResponse);
        }
    }
} while (exit == false);
```

١٠. احفظ التعليمات البرمجية المحدثة.

١١. في قائمة Run حدد Start Debugging

١٢. قم بالتبديل إلى لوحة TERMINAL

١٣. في المطالبة "الحد الأدنى lower bound" أدخل 3

١٤. في المطالبة "الحد الأعلى upper bound" أدخل 3

١٥. لاحظ أنه يتم عرض الإخراج التالي في لوحة TERMINAL:

```
Enter the lower bound: 3
Enter the upper bound: 3
An error has occurred.
```

```
ArgumentOutOfRangeException: upper bound must be greater than lower bound. (Parameter 'upperBound')
The upper bound must be greater than 3
Enter a new upper bound (or enter Exit to quit):
```

١٦. في المطالبة new upper bound أدخل 11

١٧. لاحظ أنه يتم عرض الإخراج التالي في لوحة TERMINAL:

```
Enter the lower bound: 3
Enter the upper bound: 3
An error has occurred.
ArgumentOutOfRangeException: upper bound must be greater than lower bound. (Parameter 'upperBound')
The upper bound must be greater than 3
Enter a new upper bound (or enter Exit to quit): 11
The average of even numbers between 3 and 11 is 7.
```

١٨. للخروج من التطبيق، اضغط على مفتاح الإدخال Enter

تهانينا! لقد نجحت في طرح استثناء واكتشافه والتعامل معه. thrown, caught, and handled an exception

خلاصة

فيما يلي بعض الأشياء المهمة التي يجب تذكرها من هذا الدرس:

- تأكد من تكوين بيئة تتبع الأخطاء debug environment الخاصة بك لدعم متطلبات تطبيقك.
- يجب أن تطرح التعليمات البرمجية للأسلوب استثناءً throw an exception عند اكتشاف مشكلة أو شرط.
- يجب اكتشاف الاستثناءات عند مستوى في مكدس الاستدعاءات call stack حيث يمكن حلها.

٤ تمرين - إكمال نشاط تحدي لإنشاء الاستثناءات وطرحها throw

تستخدم تحديات التعليمات البرمجية في هذا الدرس، لتعزيز ما تعلمته ومساعدتك على اكتساب بعض الثقة قبل المتابعة.

تحدي إنشاء الاستثناءات وطرحها Create and throw exceptions

في هذا التحدي، تبدأ بنموذج تطبيق يستخدم سلسلة من استدعاءات الأسلوب لمعالجة البيانات، تنشئ عبارات المستوى الأعلى مصفوفة من قيم مدخلات المستخدم، وتستدعي أسلوباً يسمى `Workflow1` يمثل `Workflow1` سير عمل عالي المستوى يتكرر عبر المصفوفة، ويمرر قيم إدخال المستخدم إلى أسلوب يسمى `Process1` يستخدم `Process1` بيانات مدخلات المستخدم لحساب قيمة.

حاليًا، عندما يواجه `Process1` مشكلة أو خطأ، فإنه يقوم بإرجاع سلسلة تصف المشكلة بدلاً من طرح استثناء، يتمثل التحدي الذي تواجهه في تنفيذ معالجة الاستثناء في نموذج التطبيق.

استخدم الخطوات التالية لإكمال التحدي:

١. استخدم التعليمات البرمجية التالية كنموذج أولي للتطبيق:

```
string[][] userEnteredValues = new string[][]
{
            new string[] { "1", "2", "3"},
            new string[] { "1", "two", "3"},
            new string[] { "0", "1", "2"}
};

string overallStatusMessage = "";

overallStatusMessage =
Workflow1(userEnteredValues);
```

```
if (overallStatusMessage == "operating
procedure complete")
{
    Console.WriteLine("'Workflow1' completed
successfully.");
}
else
{
    Console.WriteLine("An error occurred during
'Workflow1'.");
    Console.WriteLine(overallStatusMessage);
}

static string Workflow1(string[][]
userEnteredValues)
{
    string operationStatusMessage = "good";
    string processStatusMessage = "";

    foreach (string[] userEntries in
userEnteredValues)
    {
        processStatusMessage =
Process1(userEntries);

        if (processStatusMessage == "process
complete")
        {
            Console.WriteLine("'Process1'
completed successfully.");
            Console.WriteLine();
        }
```

```
        else
        {
            Console.WriteLine("'Process1' encountered an issue, process aborted.");

Console.WriteLine(processStatusMessage);
            Console.WriteLine();
            operationStatusMessage = processStatusMessage;
        }
    }

    if (operationStatusMessage == "good")
    {
        operationStatusMessage = "operating procedure complete";
    }

    return operationStatusMessage;
}

static string Process1(String[] userEntries)
{
    string processStatus = "clean";
    string returnMessage = "";
    int valueEntered;

    foreach (string userValue in userEntries)
    {
        bool integerFormat = int.TryParse(userValue, out valueEntered);
```

```
        if (integerFormat == true)
        {
            if (valueEntered != 0)
            {
                checked
                {
                    int calculatedValue = 4 /
valueEntered;
                }
            }
            else
            {
                returnMessage = "Invalid data.
User input values must be non-zero values.";
                processStatus = "error";
            }
        }
        else
        {
            returnMessage = "Invalid data. User
input values must be valid integers.";
            processStatus = "error";
        }
    }

    if (processStatus == "clean")
    {
        returnMessage = "process complete";
    }

    return returnMessage;
}
```

٢. تحقق من أن نموذج التطبيق يطبع الرسائل التالية إلى وحدة التحكم:

'Process1' completed successfully.

'Process1' encountered an issue, process aborted.
Invalid data. User input values must be valid integers.

'Process1' encountered an issue, process aborted.
Invalid data. User input values must be non-zero values.

An error occurred during 'Workflow1'.
Invalid data. User input values must be non-zero values.

٣. تحديث نموذج التطبيق باستخدام المتطلبات التالية:

- يجب تحويل كافة الأساليب من أساليب `static string` إلى أساليب `static void`
- يجب أن يطرح الأسلوب `Process1` استثناءات لكل نوع من المشاكل التي تمت مواجهتها.
- يجب أن يلتقط الأسلوب `Workflow1` الاستثناءات `FormatException` ويعالجها.
- يجب أن تلتقط عبارات المستوى الأعلى الاستثناءات `DivideByZeroException` وتعالجها.
- يجب استخدام خاصية `Message` الخاصة بالاستثناء لإعلام المستخدم بالمشكلة.

٤. يجب أن يطبع الحل المكتمل الرسائل التالية إلى وحدة التحكم:

'Process1' completed successfully.

'Process1' encountered an issue, process aborted.
Invalid data. User input values must be valid integers.

An error occurred during 'Workflow1'.
Invalid data. User input values must be non-zero values.

٥ مراجعة حل تحدي إنشاء الاستثناءات وطرحها throw

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الوحدة السابقة:

```
string[][] userEnteredValues = new string[][]
{
            new string[] { "1", "2", "3"},
            new string[] { "1", "two", "3"},
            new string[] { "0", "1", "2"}
};

try
{
    Workflow1(userEnteredValues);
    Console.WriteLine("'Workflow1' completed
successfully.");

}
catch (DivideByZeroException ex)
{
    Console.WriteLine("An error occurred during
'Workflow1'.");
    Console.WriteLine(ex.Message);
}

static void Workflow1(string[][]
userEnteredValues)
{
    foreach (string[] userEntries in
userEnteredValues)
    {
        try
        {
            Process1(userEntries);
```

```
            Console.WriteLine("'Process1'
completed successfully.");
            Console.WriteLine();
        }
        catch (FormatException ex)
        {
            Console.WriteLine("'Process1'
encountered an issue, process aborted.");
            Console.WriteLine(ex.Message);
            Console.WriteLine();
        }
    }
}

static void Process1(String[] userEntries)
{
    int valueEntered;

    foreach (string userValue in userEntries)
    {
        bool integerFormat =
int.TryParse(userValue, out valueEntered);

        if (integerFormat == true)
        {
            if (valueEntered != 0)
            {
                checked
                {
                    int calculatedValue = 4 /
valueEntered;
                }
            }
            else
            {
```

```
                throw new
DivideByZeroException("Invalid data. User input
values must be non-zero values.");
            }
        }
        else
        {
            throw new FormatException("Invalid
data. User input values must be valid
integers.");
        }
    }
}
```

يؤدي تشغيل هذا التطبيق إلى إنشاء الإخراج المطلوب:

```
'Process1' completed successfully.

'Process1' encountered an issue, process aborted.
Invalid data. User input values must be valid integers.

An error occurred during 'Workflow1'.
Invalid data. User input values must be non-zero values.
```

هذه التعليمة البرمجية هي مجرد حل واحد ممكن لأن هناك العديد من الطرق لإنشاء الاستثناءات وطرحها، طالما حصلت على النتائج الصحيحة، وفقاً لقواعد التحدي، ففعلت ذلك بشكل رائع!

نهنئك في حال نجاحك! تابع لاختبار المعلومات في الدرس التالي.

هام

إذا كان لديك مشكلة في إكمال هذا التحدي، ربما يجب عليك مراجعة الدروس السابقة قبل المتابعة.

٦ التحقق من المعرفة

١- يحتاج المطور إلى إنشاء استثناء وطرحه throw في تطبيق C# أي من الأوصاف التالية صحيح؟

- يجب ألا يتطابق نوع الاستثناء مع الغرض المقصود من الاستثناء.
- يعتمد نوع الاستثناء الذي يقومون بإنشائه على مشكلة الترميز.
- تتضمن عملية طرح كائن استثناء استخدام الكلمة الأساسية catch

٢- متى يجب أن يطرح أسلوب استثناء throw an exception؟

- يجب أن يطرح الأسلوب استثناء عندما يستطع الأسلوب إكمال الغرض المقصود منه.
- يجب أن يطرح الأسلوب استثناء عندما لا يتمكن الأسلوب من إكمال الغرض المقصود منه.
- يجب أن يطرح الأسلوب استثناء عندما يتطابق نوع استثناء مع الغرض المقصود من الأسلوب.

٣- يحتاج المطور إلى إنشاء كائن استثناء يتضمن خاصية Message محددة أي من الجمل التالية صحيحة؟

- خاصية الاستثناء Message قابلة للكتابة.
- يجب ألا توضح الخاصية Message سبب الاستثناء.
- يجب أن توضح الخاصية Message سبب الاستثناء.

راجع إجابتك

١ يعتمد نوع الاستثناء الذي يقومون بإنشائه على مشكلة الترميز.

صحيح يعتمد نوع الاستثناء exception type الذي تم إنشاؤه على مشكلة الترميز، ويجب أن يتطابق مع الغرض المقصود من الاستثناء قدر الإمكان

٢ يجب أن يطرح الأسلوب استثناء عندما لا يتمكن الأسلوب من إكمال الغرض المقصود منه.

صحيح يجب أن تطرح الأساليب استثناء throw an exception كلما لم تتمكن من إكمال الغرض المقصود منها

٣ يجب أن توضح الخاصية Message سبب الاستثناء.

صحيح يجب أن توضح خاصية الاستثناء Message سبب الاستثناء، نظراً لأن Message خاصية للقراءة فقط، يجب تعيينها عند إنشاء مثيل لكائن الاستثناء exception object

٧ الملخص

كان هدفك هو اكتساب خبرة في إنشاء وطرح كائنات استثناء مخصصة creating and throwing customized exception objects يمكن التقاطها ومعالجتها caught and handled داخل كتلة التعليمات البرمجية catch

من خلال إنشاء الاستثناءات المخصصة وطرحها والتقاطها ومعالجتها creating, throwing, catching, and handling customized exceptions يمكنك تبسيط عملية الكشف عن المشكلات البرمجية، وتصحيحها داخل تطبيقك، يتيح لك تخصيص الاستثناءات بمعلومات خاصة بتطبيقك، تحسين شرح المشكلات داخل واجهة مستخدم التطبيق، يساعد طرح استثناءات داخل أسلوب، واستخدام دعم C# المضمن في التقاط كائنات الاستثناءات على أي مستوى داخل مكدس الاستدعاءات، على ضمان اكتشاف المشكلات حيث يمكن معالجتها.

بدون القدرة على إنشاء كائنات استثناء مخصصة وطرحها، سيكون من الصعب الحفاظ على تعليماتك البرمجية، وستعاني تجربة المستخدم.

المصادر:

يمكنك العثور على معلومات إضافية حول استخدام الاستثناءات هنا:

using-exceptions

يمكنك العثور على معلومات إضافية حول الإنشاء والطرح هنا:

creating-and-throwing-exceptions

يمكنك العثور على معلومات إضافية حول إنشاء الاستثناءات المحددة من قبل المستخدم هنا:

create-user-defined-exceptions

يمكنك العثور على معلومات إضافية حول رسائل الاستثناء المترجمة هنا:

create-localized-exception-messages

يمكنك العثور على معلومات إضافية حول طرح الاستثناءات بشكل صريح هنا:

explicitly-throw-exceptions

يمكنك العثور على معلومات إضافية حول استخدام استثناءات مخصصة هنا:

specific-exceptions-in-a-catch-block

الوحدة الخامسة

مشروع موجه ـ تصحيح ومعالجة الاستثناءات في تطبيق وحدة تحكم C# باستخدام Visual Studio Code

ترشدك هذه الوحدة خلال عملية مراجعة التعليمات البرمجية، وتصحيح الأخطاء، بالإضافة إلى عملية إضافة معالجة الاستثناءات إلى أحد التطبيقات.

الأهداف التعليمية

- استخدم أدوات مصحح أخطاء Visual Studio Code debugger tools لتحديد وتصحيح مشكلة، في منطق التعليمات البرمجية.
- تنفيذ معالجة الاستثناء exception handling في تطبيق وحدة تحكم C# باستخدام النمط try-catch
- إنشاء استثناءات وطرحها throw exceptions بخصائص مخصصة customized properties
- التقاط الاستثناءات Catch exceptions على مستوى أقل في مكدس الاستدعاءات call stack

محتويات الوحدة: -

١- مقدمة.

٢- الإعداد.

٣- تمرين - مراجعة واختبار تطبيق وحدة تحكم C# باستخدام نموذج بيانات.

٤- تمرين - طرح الاستثناءات والتقاطها Throw and catch exceptions في تطبيق وحدة تحكم C#

٥- التحقق من المعرفة.

٦- الملخص.

١ المقدمة

مطورو C# مسؤولون عن تصحيح أخطاء التعليمات البرمجية، ومعالجة الاستثناءات في تطبيقاتهم، تدعم لغة C# أنماط معالجة الاستثناء، مثل الأنماط try-catch و try-catch-finally يستخدم Visual Studio Code الملحقات لدعم تصحيح أخطاء التعليمات البرمجية، يمكنك العثور على ملحقات مصحح الأخطاء في Visual Studio Code Marketplace

لنفترض أنك جزء من فريق يقوم بتطوير تطبيقات دعم البيع بالتجزئة، يعمل الفريق على تطبيق تسجيل نقدي يدير معاملات البيع بالتجزئة، تقوم بتطوير طريقة عمل MakeChange التي تدير المال حتى وحساب مقدار التغيير الذي تم إرجاعه إلى العميل، يقوم أسلوب MakeChange بتتبع عدد الأوراق النقدية لكل فئة (1, 5, 10, and 20) المتوفرة في الخزنة "درج النقود" يخطط الفريق لإضافة معالجة استثناء إلى التطبيق، أثناء عملية التحقق من التعليمات البرمجية.

ترشدك هذه الوحدة خلال عملية مراجعة التعليمات البرمجية، وتصحيح الأخطاء، حيث تتم إضافة معالجة الاستثناء أيضاً إلى التطبيق.

في نهاية هذه الوحدة، يمكنك تصحيح أخطاء تطبيق C# باستخدام مصحح أخطاء debugger وتنفيذ نمط try-catch وطرح الاستثناءات throw exceptions التي يتم اكتشافها في مستوى أقل من مكدس الاستدعاءات.

ملاحظة

هذه وحدة مشروع موجهة حيث يمكنك إكمال مشروع شامل، من خلال اتباع الإرشادات خطوة بخطوة.

٢ الإعداد

في هذا المشروع الإرشادي، يمكنك استخدام Visual Studio Code لتحديث تطبيق C# تركز التحديثات على تصحيح أخطاء التعليمات البرمجية، وإضافة معالجة الاستثناءات إلى التطبيق، يمكنك مراجعة التطبيق وتصحيحه، وتنفيذ نمط try-catch في عبارات المستوى الأعلى، ثم طرح استثناءات من داخل أسلوب يتم اكتشافه في عبارات المستوى الأعلى.

نظرة عامة على المشروع

أنت جزء من فريق يعمل على تطبيقات دعم البيع بالتجزئة، التعليمات البرمجية التي تقوم بتطويرها الأسلوب `MakeChange` لإدارة أموال تطبيق تسجيل النقد، يجب أن يفي تطبيقك بالمواصفات التالية:

- تطبيق وحدة تحكم C# يحاكي معاملات الشراء اليومية.
- يستدعي التطبيق الأسلوب `MakeChange` لإدارة الأموال "درج النقود" أثناء المعاملات، يقبل `MakeChange` الدفعات النقدية ويعيد التغيير.
- يتحقق تطبيق الاستدعاء بشكل مستقل من رصيد درج النقود بعد كل معاملة.
- يتم تنفيذ نمط `try-catch` لإدارة الاستثناءات كما يلي:
  - تُستخدم الاستثناءات للإبلاغ عن أي مشكلة تمنع إتمام المعاملة بنجاح ومعالجتها.
  - يتم إنشاء الاستثناءات وإلقائها في أسلوب `MakeChange`
  - يتم اكتشاف الاستثناءات ومعالجتها في تطبيق الاستدعاء.

تم بالفعل تطوير تطبيق يحاكي المعاملات ويستدعي الأسلوب `MakeChange` يتضمن مشروع التعليمات البرمجية المبدئي Starter لهذه الوحدة الإرشادية للمشروع ملف Program.cs الذي يتضمن التعليمات البرمجية التالية:

محاكاة المعاملات: تقوم عبارات المستوى الأعلى بتكوين بيانات التطبيق، ومحاكاة سلسلة من المعاملات، باستخدام أما مصفوفة `testData` أو عدد أكبر من المعاملات التي تم إنشاؤها عشوائياً.

تهيئة "till" درج النقود: يتم استخدام الأسلوب `LoadTillEachMorning` لتكوين السجل النقدي "till" درج النقود مع عدد محدد مسبقاً من الأوراق النقدية في كل فئة.

معاملات العملية: يتم استخدام الأسلوب `MakeChange` لإدارة أموال درج النقود أثناء معاملات الشراء.

تقرير حالة "till" درج النقود: يتم استخدام الأسلوب `LogTillStatus` لعرض عدد الأوراق النقدية من كل فئة موجودة حاليًا في "till" درج النقود.

تقرير رصيد "till" درج النقود: يتم استخدام الأسلوب `TillAmountSummary` لعرض رسالة تظهر المبلغ النقدي في درج النقود.

ملاحظة

للحفاظ على الحسابات بسيطة، جميع تكاليف العناصر أرقامًا صحيحة، وتشمل أي ضريبة أو رسوم، وهذا يحافظ على تركيز مهام الترميز على تصحيح الأخطاء ومعالجة الاستثناءات.

هدفك لهذه الوحدة هو التحقق من أن منطق التطبيق يعمل بشكل صحيح، وعزل وتصحيح أي أخطاء منطقية، وتنفيذ معالجة الاستثناءات، لتحقيق هذا الهدف، سوف تكمل التدريبات التالية:

١. مراجعة وتصحيح التعليمات البرمجية للتطبيق الحالي.

٢. تحديث التطبيق لتنفيذ معالجة الاستثناءات.

الإعداد

استخدم الخطوات التالية للتحضير لتمارين المشروع الإرشادي:

١. لتنزيل ملف مضغوط يحتوي على التعليمات البرمجية للمشروع المبدئي Starter حدد الارتباط التالي Lab Files

٢. فك ضغط ملفات التنزيل.

فك ضغط الملفات في بيئة التطوير الخاصة بك، ضع في اعتبارك استخدام الكمبيوتر كبيئة تطوير حتى تتمكن من الوصول إلى تعليماتك البرمجية بعد إكمال هذا الدرس، إذا كنت لا تستخدم الكمبيوتر كبيئة تطوير، فيمكنك فك ضغط الملفات في بيئة الاختبار المعزولة "وضع الحماية" sandbox أو البيئة المستضيفة hosted environment

- على جهازك المحلي، انتقل إلى مجلد التنزيلات.
- انقر بزر الماوس الأيمن فوق Guided-project-debugging-CSharp-main.zip ثم حدد Extract all
- حدد إظهار الملفات المستخرجة عند اكتمالها Show extracted files when complete ثم حدد استخراج Extract
- دون موقع المجلد المستخرج.

٣. انسخ مجلد GuidedProject المستخرج إلى مجلد سطح مكتب Windows

ملاحظة

إذا كان هناك مجلد باسم GuidedProject موجود بالفعل، يمكنك تحديد استبدال الملفات في الوجهة لإكمال عملية النسخ.

٤. افتح مجلد GuidedProject الجديد في Visual Studio Code

- افتح Visual Studio Code
- في القائمة File حدد Open Folder

- انتقل إلى مجلد سطح مكتب Windows وحدد موقع المجلد "GuidedProject"
- حدد GuidedProject ثم حدد Select Folder

يجب أن تظهر قائمة استكشاف EXPLORER مجلد GuidedProject ومجلدين فرعيين باسم Final وStarter

أنت الآن جاهز لبدء تمارين المشروع الإرشادي. حظ سعيد!

٣ تمرين ـ مراجعة واختبار تطبيق وحدة تحكم C# باستخدام نموذج بيانات

في هذا التمرين، يمكنك مراجعة التعليمات البرمجية، واختبارها في المشروع المبدئي Starter وعزل مشكلة منطقية وإصلاحها، ثم التحقق من أن التطبيق المحدث يعمل كما هو متوقع.

أكمل المهام التالية أثناء هذا التمرين:

١. مراجعة التعليمات البرمجية: راجع محتويات ملف Program.cs

يتضمن Program.cs أقسام التعليمات البرمجية التالية:

- عبارات المستوى الأعلى Top-level statements: تحاكي عبارات المستوى الأعلى سلسلة من المعاملات باستخدام مصفوفة `testData` أو عدد أكبر من المعاملات التي تم إنشاؤها عشوائياً.
- تحميل درج النقود كل صباح LoadTillEachMorning: يستخدم الأسلوب `LoadTillEachMorning` لتكوين السجل النقدي لدرج النقود مع عدد محدد مسبقاً من الأوراق النقدية في كل فئة.
- MakeChange يتم استخدام الأسلوب `MakeChange` لإدارة النقد لدرج النقود أثناء معاملات الشراء.
- سجل حالة درج النقود LogTillStatus: يتم استخدام الأسلوب `LogTillStatus` لعرض عدد الأوراق النقدية لكل فئة حالية في درج النقود.
- المبلغ النقدي في درج النقود TillAmountSummary: يتم استخدام الأسلوب `TillAmountSummary` لعرض رسالة تظهر المبلغ النقدي في درج النقود.

٢. الاختبار الأولي: تأكد من نجاح `MakeChange` في موازنة أموال درج النقود، عند استخدام مصفوفة `testData` لمحاكاة المعاملات.

٣. تصحيح أخطاء التعليمات البرمجية Code debugging: عزل وتصحيح المشكلة المنطقية التي يتم كشفها عند استخدام بيانات تم إنشاؤها عشوائياً.

٤. اختبار التحقق Verification test: قم بإجراء اختبار تحقق على التعليمات البرمجية التي قمت بتطويرها في هذا التمرين.

مراجعة محتويات ملف Program.cs

في هذه المهمة، يمكنك إكمال معاينة التعليمات البرمجية لمشروع Starter يحتوي ملف Program.cs على تطبيق يحاكي شروط المعاملات اليومية، يستدعي التطبيق الأسلوب `MakeChange` لإدارة أموال درج النقود أثناء كل معاملة، يتم استخدام أساليب أخرى لتهيئة درج النقود، وإنشاء رسائل إعداد التقارير.

١. تأكد من فتح مجلد GuidedProject في Visual Studio Code

٢. في قائمة EXPLORER قم بتوسيع المجلدين GuidedProject وStarter

يحتوي مجلد Starter على نموذج التطبيق للمشروع الإرشادي.

٣. افتح ملف Program.cs في محرر التعليمات البرمجية Visual Studio

٤. في القائمة عرض View حدد لوحة الأوامر Command Palette

٥. في موجه الأوامر command prompt أدخل `.net: g` ثم حدد **`.NET: Generate Assets for Build and Debug`**

٦. في المطالبة Select the project to launch حدد مشروع Starter

سيتضمن ملف launch.json الذي تم إنشاؤه تكوينًا لمشروع Starter

٧. استغرق بضع دقائق لمراجعة عبارات المستوى الأعلى لهذا التطبيق:

```
/*
This application manages transactions at a
store check-out line. The
```

```
check-out line has a cash register, and the
register has a cash till
that is prepared with a number of bills each
morning. The till includes
bills of four denominations: $1, $5, $10, and
$20. The till is used
to provide the customer with change during the
transaction. The item
cost is a randomly generated number between 2
and 49. The customer
offers payment based on an algorithm that
determines a number of bills
in each denomination.

Each day, the cash till is loaded at the start
of the day. As transactions
occur, the cash till is managed in a method
named MakeChange (customer
payments go in and the change returned to the
customer comes out). A
separate "safety check" calculation that's used
to verify the amount of
money in the till is performed in the "main
program". This safety check
is used to ensure that logic in the MakeChange
method is working as
expected.
*/

string? readResult = null;
bool useTestData = true;

Console.Clear();

int[] cashTill = new int[] { 0, 0, 0, 0 };
int registerCheckTillTotal = 0;
```

```
// registerDailyStartingCash: $1 x 50, $5 x 20, $10 x 10, $20 x 5 => ($350 total)
int[,] registerDailyStartingCash = new int[,] { { 1, 50 }, { 5, 20 }, { 10, 10 }, { 20, 5 } };

int[] testData = new int[] { 6, 10, 17, 20, 31, 36, 40, 41 };
int testCounter = 0;

LoadTillEachMorning(registerDailyStartingCash, cashTill);

registerCheckTillTotal =
registerDailyStartingCash[0, 0] *
registerDailyStartingCash[0, 1] +
registerDailyStartingCash[1, 0] *
registerDailyStartingCash[1, 1] +
registerDailyStartingCash[2, 0] *
registerDailyStartingCash[2, 1] +
registerDailyStartingCash[3, 0] *
registerDailyStartingCash[3, 1];

// display the number of bills of each denomination currently in the till
LogTillStatus(cashTill);

// display a message showing the amount of cash in the till
Console.WriteLine(TillAmountSummary(cashTill));
```

```
// display the expected
registerDailyStartingCash total
Console.WriteLine($"Expected till value:
{registerCheckTillTotal}\n\r");

var valueGenerator = new
Random((int)DateTime.Now.Ticks);

int transactions = 10;

if (useTestData)
{
    transactions = testData.Length;
}

while (transactions > 0)
{
    transactions -= 1;
    int itemCost = valueGenerator.Next(2, 20);

    if (useTestData)
    {
        itemCost = testData[testCounter];
        testCounter += 1;
    }

    int paymentOnes = itemCost % 2;
// value is 1 when itemCost is odd, value is 0
when itemCost is even
    int paymentFives = (itemCost % 10 > 7) ? 1 : 0;
// value is 1 when itemCost ends with 8 or 9,
otherwise value is 0
```

```
    int paymentTens = (itemCost % 20 > 13) ? 1 : 0;
// value is 1 when 13 < itemCost < 20 OR 33 <
itemCost < 40, otherwise value is 0
    int paymentTwenties = (itemCost < 20) ? 1 : 2;
// value is 1 when itemCost < 20, otherwise
value is 2

    // display messages describing the current
transaction
    Console.WriteLine($"Customer is making a
${itemCost} purchase");
    Console.WriteLine($"\t Using
{paymentTwenties} twenty dollar bills");
    Console.WriteLine($"\t Using {paymentTens}
ten dollar bills");
    Console.WriteLine($"\t Using {paymentFives}
five dollar bills");
    Console.WriteLine($"\t Using {paymentOnes}
one dollar bills");

    // MakeChange manages the transaction and
updates the till
    string transactionMessage =
MakeChange(itemCost, cashTill, paymentTwenties,
paymentTens, paymentFives, paymentOnes);

    // Backup Calculation - each transaction
adds current "itemCost" to the till
    if (transactionMessage == "transaction
succeeded")
    {
```

```
        Console.WriteLine($"Transaction
successfully completed.");
        registerCheckTillTotal += itemCost;
    }
    else
    {
        Console.WriteLine($"Transaction
unsuccessful: {transactionMessage}");
    }

Console.WriteLine(TillAmountSummary(cashTill));
    Console.WriteLine($"Expected till value:
{registerCheckTillTotal}\n\r");
    Console.WriteLine();
}

Console.WriteLine("Press the Enter key to
exit");
do
{
    readResult = Console.ReadLine();

} while (readResult == null);
```

التعليمات البرمجية لعبارات المستوى الأعلى تكمل المهام التالية:

- تكوين بيانات التطبيق ومتغيرات البيئة المستخدمة لاختبار الأسلوب `MakeChange`
- استدعاء أساليب `LoadTillEachMorning()`, `LogTillStatus()`, `TillAmountSummary()` لإعداد أموال درج النقود money till وطباعة رسائل الحالة إلى وحدة التحكم.

- يستخدم حلقة `while` لمحاكاة سلسلة من المعاملات.
- استدعاء الأسلوب `MakeChange` من داخل كتلة التعليمات البرمجية للحلقة `while`
- تقارير عن حالة المال في درج النقود، بعد كل معاملة.

ملاحظة

تتضمن عبارات المستوى الأعلى عبارة Console.ReadLine() سيحتاج الملف launch.json إلى التحديث قبل تصحيح الأخطاء.

٨. خذ لحظة لمراجعة الأسلوب `LoadTillEachMorning()`

```
static void LoadTillEachMorning(int[,]
registerDailyStartingCash, int[] cashTill)
{
    cashTill[0] = registerDailyStartingCash[0, 1];
    cashTill[1] = registerDailyStartingCash[1, 1];
    cashTill[2] = registerDailyStartingCash[2, 1];
    cashTill[3] = registerDailyStartingCash[3, 1];
}
```

٩. خذ لحظة لمراجعة الأسلوب `MakeChange()`

```
static string MakeChange(int cost, int[]
cashTill, int twenties, int tens = 0, int fives
= 0, int ones = 0)
{
    string transactionMessage = "";

    cashTill[3] += twenties;
    cashTill[2] += tens;
    cashTill[1] += fives;
    cashTill[0] += ones;
```

```
    int amountPaid = twenties * 20 + tens * 10
+ fives * 5 + ones;
    int changeNeeded = amountPaid - cost;

    if (changeNeeded < 0)
        transactionMessage = "Not enough money
provided.";

    Console.WriteLine("Cashier Returns:");

  while ((changeNeeded > 19) && (cashTill[3] > 0))
    {
        cashTill[3]--;
        changeNeeded -= 20;
        Console.WriteLine("\t A twenty");
    }

   while ((changeNeeded > 9) && (cashTill[2] > 0))
    {
        cashTill[2]--;
        changeNeeded -= 10;
        Console.WriteLine("\t A ten");
    }

   while ((changeNeeded > 4) && (cashTill[1] > 0))
    {
        cashTill[2]--;
        changeNeeded -= 5;
        Console.WriteLine("\t A five");
    }

   while ((changeNeeded > 0) && (cashTill[0] > 0))
```

```
    {
        cashTill[0]--;
        changeNeeded--;
        Console.WriteLine("\t A one");
    }

    if (changeNeeded > 0)
        transactionMessage = "Can't make
change. Do you have anything smaller?";

    if (transactionMessage == "")
        transactionMessage = "transaction
succeeded";

    return transactionMessage;
}
```

الأسلوب MakeChange يدير أموال درج النقود، أثناء كل عملية شراء، تعتمد عملية المعاملة على الموارد والشروط التالية:

المعاملة النقدية Cash transaction: يقبل الأسلوب MakeChange دفعة نقدية من العميل، ثم يحدد عدد الأوراق النقدية كل فئة يجب إرجاعها إلى العميل في التغيير، يجب على MakeChange التأكد أولاً من أن العميل قد قدم ما يكفي من المال لتغطية المعاملة، إذا كان المبلغ المدفوع كافياً، تبدأ عملية "إجراء التغيير make change" بأكبر فئة من فئات الأوراق النقدية، وتستمر وصولاً إلى أصغر فئة، في كل مرحلة، يضمن MakeChange أن الفئة الحالية أقل من التغيير المستحق، كما يضمن MakeChange أيضًا وجود الأوراق النقدية بالقيمة المطلوبة في درج النقود قبل إضافتها إلى التغيير الذي تم إرجاعه إلى العميل.

معلمات الإدخال Input parameters: يستخدم الأسلوب MakeChange معلمات الإدخال التالية:

- عدد صحيح integer يمثل تكلفة العنصر الذي يتم شراؤه: `itemCost`
- مصفوفة عدد صحيح integer array تحتوي على عدد الأوراق النقدية في درج النقود، لكل فئة نقدية: `cashTill`
- الدفعة التي يقدمها العميل، حيث يتم تحديد عدد الفواتير لكل فئة بشكل منفصل: `paymentTwenties, paymentTens, paymentFives, paymentOnes`

النقد المتوفر في درج النقود Cash available in till: يجب أن يتم تضمين الأوراق النقدية المقدمة للدفع من قبل العميل، في أوراق كل فئة متاحة لإجراء التغيير.

التغيير المستحق للعميل Change owed to customer: يتم حساب التغيير المستحق للعميل باستخدام المبلغ المدفوع من قبل العميل مطروحًا منه تكلفة الصنف.

الدفع الناقص Underpayment: إذا لم يقدم العميل دفعة كافية، فسيقوم `MakeChange` بإرجاع رسالة وصفية، ويتم إلغاء المعاملة.

درج النقود غير قادر Insufficient till: إذا تعذر على درج النقود إجراء تغيير دقيق، يقوم `MakeChange` بإرجاع رسالة وصفية، ويتم إلغاء المعاملة.

١٠. خذ لحظة لمراجعة الأسلوب `LogTillStatus()`

```
static void LogTillStatus(int[] cashTill)
{
  Console.WriteLine("The till currently has:");
  Console.WriteLine($"{cashTill[3]   *   20}   in
twenties");
  Console.WriteLine($"{cashTill[2] * 10} in tens");
  Console.WriteLine($"{cashTill[1] * 5} in fives");
  Console.WriteLine($"{cashTill[0]} in ones");
  Console.WriteLine();
}
```

يستخدم أسلوب `LogTillStatus` مصفوفة `cashTill` للإبلاغ عن المحتويات الحالية لدرج النقود.

١١. خذ لحظة لمراجعة الأسلوب `TillAmountSummary()`

```
static string TillAmountSummary(int[] cashTill)
{
    return $"The till has {cashTill[3] * 20 +
cashTill[2] * 10 + cashTill[1] * 5 +
cashTill[0]} dollars";
}
```

يستخدم الأسلوب `TillAmountSummary` مصفوفة `cashTill` لحساب الرصيد النقدي الحالي المتوفر في درج النقود.

بهذا يكتمل مراجعة مشروع التعليمات البرمجية الموجود.

تأكد أن MakeChange يدير الأموال بنجاح عند استخدام مصفوفة testData

في هذه المهمة، يمكنك محاكاة المعاملات باستخدام المصفوفة `testData` والتحقق من نجاح `MakeChange` في موازنة أموال درج النقود money till

١. في قائمة Visual Studio Code Run حدد Start Debugging

٢. لاحظ حدوث خطأ `IOException`

لا تدعم وحدة تحكم تتبع الأخطاء `DEBUG CONSOLE` الأساليب `Console.Clear()` or `Console.ReadLine()` تحتاج إلى تحديث ملف launch.json قبل تصحيح الأخطاء.

٣. في شريط أدوات تتبع الأخطاء Debug toolbar حدد إيقاف Stop

٤. استخدم قائمة استكشاف EXPLORER لفتح ملف launch.json

٥. في ملف launch.json قم بتحديث السمة console كما يلي:

```
// For more information about the 'console' field, see https://aka.ms/VSCode-CS-LaunchJson-Console
"console":"integratedTerminal",
```

القيمة الافتراضية للسمة `console` هي `internalConsole` والتي تتوافق مع لوحة DEBUG CONSOLE، لسوء الحظ لا تدعم لوحة DEBUG CONSOLE بعض أساليب وحدة التحكم، يتوافق الإعداد `integratedTerminal` مع لوحة TERMINAL التي تدعم إدخال وإخراج وحدة التحكم console input and output

٦. احفظ التغييرات التي أجريتها على ملف launch.json

٧. في قائمة Visual Studio Code Run حدد Start Debugging

٨. راجع الإخراج الذي تم إنشاؤه بواسطة التطبيق في لوحة TERMINAL

قم بالتبديل من لوحة DEBUG CONSOLE إلى لوحة TERMINAL لمراجعة الإخراج.

٩. لاحظ أن `MakeChange` يوازن بنجاح درج النقود till عند استخدام المصفوفة `testData` لمحاكاة المعاملات.

يجب أن تشاهد الأسطر التالية مدرجة في أسفل الإخراج المُبلغ عنه:

```
The till has 551 dollars
Expected till value: 551


Press the Enter key to exit
```

لاحظ أن قيمتي درج النقود till المبلغ عنها والمتوقعة تبلغ 551

١٠. للخروج من التطبيق، اضغط على مفتاح الإدخال Enter

تحديد مشكلات المنطق وإصلاحها Identify and fix logic issues

في هذه المهمة، يمكنك استخدام عمليات المحاكاة لعرض مشكلة منطق التعليمات البرمجية، ثم استخدام أدوات مصحح أخطاء Visual Studio Code لعزل المشكلة وإصلاحها.

١. لتشغيل التعليمات البرمجية باستخدام المعاملات التي تم إنشاؤها عشوائياً، قم بتغيير القيمة المعينة `useTestData` إلى `false`

يمكنك العثور على المتغير `useTestData` بالأعلى، بالقرب من عبارات المستوى الأعلى.

٢. احفظ ملف Program.cs ثم قم بتشغيل التطبيق في مصحح الأخطاء.

٣. راجع الإخراج في لوحة TERMINAL

٤. لاحظ التناقض في رصيد درج النقود till

يتم الإبلاغ عن الرصيد النهائي الذي تم حسابه بواسطة MakeChange والرصيد المحتفظ به في كشوف المستوى الأعلى في الجزء السفلي من الإخراج، على سبيل المثال:

```
Transaction successfully completed.
The till has 379 dollars
Expected till value: 434

Press the Enter key to exit
```

٥. للخروج من التطبيق، اضغط على مفتاح الإدخال Enter

٦. أغلق لوحة TERMINAL

تصحيح التعليمات البرمجية Debug the code

في هذه المهمة، يمكنك استخدام أدوات مصحح أخطاء Visual Studio Code لعزل ثم إصلاح مشكلة المنطق isolate and then fix the logic issue

بالقرب من نهاية عبارات المستوى الأعلى، حدد موقع سطر التعليمات البرمجية التالي:

```
Console.WriteLine();
```

٢. تعيين نقطة توقف على سطر التعليمات البرمجية المحدد.

٣. في قائمة Visual Studio Code Run حدد Start Debugging

٤. لاحظ أن تنفيذ التعليمات البرمجية يتوقف مؤقتاً عند نقطة التوقف breakpoint

٥. في شريط أدوات عناصر التحكم Debug controls في تتبع الأخطاء، حدد خطوة إلى الأمام Step Into

٦. راجع الإخراج في لوحة TERMINAL

٧. إذا كانت القيم التي تم الإبلاغ عنها والأحرف المتوقعة متساوية، فحدد متابعة Continue على شريط أدوات عناصر التحكم Debug controls في تتبع الأخطاء.

٨. كرر الخطوة السابقة حتى ترى تعارضاً بين القيم التي تم الإبلاغ عنها والأحرف المتوقعة.

٩. بمجرد أن ترى تعارضاً، خذ دقيقة لفحص تفاصيل المعاملة.

١٠. لاحظ أن المبالغ النقدية المستلمة والتغييرات التي تم إرجاعها صحيحة، ولكن المبلغ قصير بمقدار خمس دولارات.

يشير هذا النقص إلى أنه يتم تحديث مصفوفة `cashTill` بشكل غير صحيح، على الرغم من صحة التقرير.

١١. أوقف جلسة تصحيح الأخطاء وأغلق لوحة TERMINAL

١٢. قم بالتمرير إلى أسفل الأسلوب. `MakeChange`

توجد عبارات while المستخدمة في "إجراء تغيير" في نهاية الأسلوب `MakeChange`

١٣. راجع كتل التعليمات البرمجية الخاصة بعبارة while المستخدمة لإجراء التغيير.

نظرًا لأن رصيد درج النقود متوقف بمقدار خمس دولارات، فمن المحتمل أن تكون المشكلة في كتلة التعليمات البرمجية المستخدمة لإرجاع الأوراق النقدية ذات قيمة خمس دولارات.

١٤. لاحظ التعليمات البرمجية التالية:

```
while ((changeNeeded > 4) && (cashTill[1] > 0))
{
    cashTill[2]--;
    changeNeeded -= 5;
    Console.WriteLine("\t A five");
}
```

يتم استخدام مصفوفة `cashTill[]` لتخزين عدد الفواتير لكل فئة متوفرة حاليًا، يتم استخدام عنصر المصفوفة 1 لإدارة عدد الأوراق النقدية ذات الخمسة دولارات في درج النقود، يشير التعبير الموجود في عبارة while إلى `cashTill[1]` بشكل صحيح، ومع ذلك، فإن العبارة الموجودة داخل مقطع التعليمات البرمجية تقلل من `cashTill[2]` بدلاً من `cashTill[1]` إن تحديد قيمة فهرس 2 يعني أنه تتم إزالة ورقة نقدية بقيمة عشرة دولارات من الورق النقدي بدلاً من الورقة النقدية بقيمة خمسة دولارات.

١٥. تحديث كتلة التعليمات البرمجية while كما يلي:

```
while ((changeNeeded > 4) && (cashTill[1] > 0))
{
    cashTill[1]--;
    changeNeeded -= 5;
    Console.WriteLine("\t A five");
}
```

١٦. احفظ ملف Program.cs

تحقق من عملك

في هذه المهمة، يمكنك تشغيل التطبيق، والتحقق من أن التعليمات البرمجية المحدثة تعمل على النحو المنشود.

١. في قائمة Visual Studio Code Run حدد Remove All Breakpoints

٢. في قائمة Run حدد Start Debugging

٣. راجع الإخراج في لوحة TERMINAL

٤. تحقق من أن قيمة درج النقود المُبلغ عنها، تساوي قيمة الدرج المتوقعة:

يتم احتساب الرصيد النهائي لدرج النقود بواسطة `MakeChange` والرصيد المحتفظ به في عبارات المستوى الأعلى في أسفل الإخراج، على سبيل المثال:

```
Transaction successfully completed.
The till has 452 dollars
Expected till value: 452


Press the Enter key to exit
```

ينشئ التطبيق عشوائياً تكلفة شراء الأصناف، لذلك، فإن قيم درج النقود المُبلغ عنها في إخراجك مختلفة، طالما أن القيمتين متساويتين، فقد نجحت في إصلاح مشكلة المنطق.

٤ تمرين ـ طرح الاستثناءات والتقاطها Throw and catch exceptions في تطبيق وحدة تحكم C#

في هذا التمرين، ستقوم بتطوير كتلة التعليمات البرمجية `try` وعبارة `catch` في عبارات المستوى الأعلى، وإنشاء استثناءات وطرحها create and throw exceptions في الأسلوب `MakeChange` ثم إكمال كتلة التعليمات البرمجية `catch` باستخدام كائن استثناء exception object يجب إكمال المهام التالية أثناء هذا التمرين:

تحديث عبارات المستوى الأعلى: تنفيذ نمط `try-catch` في عبارات المستوى الأعلى، ستحتوي كتلة `try` على استدعاء `MakeChange`

تحديث أسلوب `MakeChange`: إنشاء وطرح استثناءات لمشكلات عدم قدرة درج النقود "Insufficient till" والدفع الناقص "Underpayment"

تحديث كتلة `catch`: لاستخدام خصائص الاستثناء الذي تم طرحه properties of the thrown exception

اختبار التحقق: قم بإجراء اختبارات التحقق للتعليمات البرمجية التي طورتها في هذا التمرين.

إضافة نمط try-catch إلى عبارات المستوى الأعلى

في هذه المهمة، ستقوم بتضمين استدعاء الأسلوب `MakeChange` داخل عبارة try وإنشاء العبارة catch المقابلة.

١. تأكد من أن الملف Program.cs مفتوح في محرر Visual Studio Code

٢. حدد موقع أسطر التعليمات البرمجية التالية:

```
// MakeChange manages the transaction and
updates the till
```

```
string transactionMessage =
MakeChange(itemCost, cashTill, paymentTwenties,
paymentTens, paymentFives, paymentOnes);

// Backup Calculation - each transaction adds
current "itemCost" to the till
if (transactionMessage == "transaction
succeeded")
{
    Console.WriteLine($"Transaction
successfully completed.");
    registerCheckTillTotal += itemCost;
}
else
{
    Console.WriteLine($"Transaction
unsuccessful: {transactionMessage}");
}
```

٣. خذ دقيقة للنظر في الغرض من هذه التعليمة البرمجية.

لاحظ أن `MakeChange` يُرجع قيمة سلسلة، يتم تعيين القيمة المرجعة إلى متغير يسمى `transactionMessage` إذا كان `transactionMessage` يساوي نجاح المعاملة "transaction succeeded" تتم إضافة تكلفة الصنف الذي تم شراؤه إلى `registerCheckTillTotal` يتم استخدام المتغير `registerCheckTillTotal` للتحقق من رصيد درج النقود المحسوب بواسطة الأسلوب `MakeChange`

٤. لإحاطة استدعاء الأسلوب `MakeChange` في كتلة عبارة try قم بتحديث التعليمات البرمجية كما يلي:

```
try
{
```

```
    // MakeChange manages the transaction and
updates the till
    string transactionMessage =
MakeChange(itemCost, cashTill, paymentTwenties,
paymentTens, paymentFives, paymentOnes);
}
```

٥. أضف عبارة catch التالية بعد كتلة التعليمات البرمجية لعبارة try:

```
catch
{
}
```

ستنتهي من تطوير العبارة catch بمجرد إنشاء الاستثناءات وطرحها created and thrown the exceptions

إنشاء استثناءات وطرحها في الأسلوب MakeChange

في هذه المهمة، ستقوم بتحديث MakeChange لإنشاء استثناءات مخصصة وطرحها create and throw custom exceptions عند تعذر إكمال المعاملة.

يتضمن الأسلوب `MakeChange` مسألتين يجب أن تؤديا إلى استثناءات:

مشكلة دفع أقل: تحدث هذه المشكلة عندما يقدم العميل دفعة أقل من تكلفة الصنف، إذا لم يقدم العميل دفعة كافية، يجب أن يطرح `MakeChange` استثناء.

مشكلة درج النقود غير قادر: تحدث هذه المشكلة عندما لا يحتوي الدرج على الأوراق المالية المطلوبة، لإنتاج التغيير الدقيق. إذا كان تعذر إجراء تغيير دقيق، يطرح `MakeChange` استثناء.

١. مرر لأسفل إلى أسلوب `MakeChange`

٢. حدد موقع أسطر التعليمات البرمجية التالية:

```
if (changeNeeded < 0)
    transactionMessage = "Not enough money
provided.";
```

٣. خذ دقيقة للنظر في المشكلة التي تعالجها هذه التعليمة البرمجية.

إذا كان `changeNeeded` أقل من الصفر، فهذا يعني أن العميل لم يقدم ما يكفي من المال لتغطية سعر شراء الصنف الذي يشتريه، يعد سعر الشراء والأموال المقدمة من قبل العميل من معلمات للأسلوب MakeChange الأسلوب غير قادر على إكمال المعاملة، عندما لا يقدم العميل ما يكفي من المال، وبعبارة أخرى، تفشل العملية.

هناك نوعان من الاستثناء يبدوان متطابقين مع هذه الشروط:

- `InvalidOperationException` يجب طرح استثناء InvalidOperationException فقط عندما لا تدعم شروط تشغيل الأسلوب الإكمال الناجح لاستدعاء أسلوب معين، في هذه الحالة يتم تأسيس شروط التشغيل بواسطة المعلمات المتوفرة للأسلوب.
- `ArgumentOutOfRangeException` يجب طرح استثناء ArgumentOutOfRangeException فقط عندما تكون قيمة الوسيطة خارج نطاق القيم المسموح بها، كما هو محدد بواسطة الأسلوب الذي تم استدعاؤه، في هذه الحالة، يجب أن تكون الأموال المقدمة أكبر من تكلفة الصنف.

يمكن أن يعمل أي من نوعي الاستثناء، ولكن `InvalidOperationException` هو تطابق أفضل قليلاً في سياق هذا التطبيق.

٤. حدث التعليمات البرمجية كما يلي:

```
if (changeNeeded < 0)
    throw new
InvalidOperationException("InvalidOperationExce
ption: Not enough money provided to complete
the transaction.");
```

٥. مرر لأسفل لتحديد موقع أسطر التعليمات البرمجية التالية:

```
if (changeNeeded > 0)
    transactionMessage = "Can't make change. Do
you have anything smaller?";
```

٦. خذ دقيقة للنظر في المشكلة التي تعالجها هذه التعليمة البرمجية.

إذا كان `changeNeeded` أكبر من الصفر بعد حلقات while التي تعد التغيير، فإن الأموال النقدية في درج النقود نفذت، التي يمكن استخدامها لإجراء التغيير، الأسلوب غير قادر على إكمال المعاملة عندما يفتقر درج النقود على الأموال المطلوبة لإجراء التغيير، بمعنى آخر، تفشل العملية.

يجب استخدام الاستثناء `InvalidOperationException` لإنشاء الاستثناء.

٧. حدث التعليمات البرمجية كما يلي:

```
if (changeNeeded > 0)
    throw new
InvalidOperationException("InvalidOperationExce
ption: The till is unable to make the correct
change.");
```

إكمال كتلة التعليمات البرمجية catch

في هذه المهمة، ستقوم بتحديث العبارة catch لالتقاط نوع استثناء معين.

١. مرر لأعلى فوق الأسلوب `MakeChange` وحدد موقع التعليمات البرمجية التالية:

```
catch
{
}
```

٢. لالتقاط catch نوع الاستثناء الذي تم طرحه thrown في الأسلوب `MakeChange` قم بتحديث التعليمات البرمجية كما يلي:

```
catch (InvalidOperationException e)
{
    Console.WriteLine($"Could not complete
transaction: {e.Message}");
}
```

سيتم اكتشاف كائن الاستثناء `InvalidOperationException` الذي تم طرحه في `MakeChange` ولكن لن يتم caught التقاط أنواع الاستثناءات الأخرى، نظراً لأنك غير مستعد للتعامل مع أنواع الاستثناءات الأخرى، فمن المهم السماح لها بالالتقاط caught بمستوي أقل في مكدس الاستدعاءات، إذا أدركت أنه من المتوقع وجود أنواع استثناءات أخرى داخل `MakeChange` يمكنك إضافة عبارات catch إضافية.

٣. استخدم القائمة ملف لحفظ التحديثات.

تحويل أسلوب MakeChange من "string" إلى "void" والوصول إلى خصائص الاستثناء

في هذه المهمة، ستقوم بتحديث MakeChange ليكون من النوع void ثم استخدم خصائص الاستثناء exception properties لتوصيل تفاصيل المشكلة للمستخدم.

١. قم بالتمرير إلى أعلى الأسلوب `MakeChange`

٢. لتحويل الأسلوب `MakeChange` من نوع string إلى نوع void قم بتحديث التعليمات البرمجية كما يلي:

```
static void MakeChange(int cost, int[] cashTill,
int twenties, int tens = 0, int fives = 0, int
ones = 0)
```

٣. احذف إعلان المتغير التالي:

```
string transactionMessage = "";
```

٤. قم بالتمرير إلى أسفل الأسلوب `MakeChange`

٥. احذف أسطر التعليمات البرمجية التالية:

```
if (transactionMessage == "")
    transactionMessage = "transaction
succeeded";

return transactionMessage;
```

٦. مرر لأعلى إلى عبارات المستوى الأعلى وحدد موقع كتلة try

٧. حدث كتلة try كما يلي:

```
try
{
    // MakeChange manages the transaction and
updates the till
    MakeChange(itemCost, cashTill,
paymentTwenties, paymentTens, paymentFives,
paymentOnes);

    Console.WriteLine($"Transaction
successfully completed.");
    registerCheckTillTotal += itemCost;
}
```

٨. حدد موقع أسطر التعليمات البرمجية التالية ثم احذفها:

```
// Backup Calculation - each transaction adds
current "itemCost" to the till
if (transactionMessage == "transaction
succeeded")
{
    Console.WriteLine($"Transaction
successfully completed.");
    registerCheckTillTotal += itemCost;
}
else
{
    Console.WriteLine($"Transaction
unsuccessful: {transactionMessage}");
}
```

الآن، تقوم كتل التعليمات البرمجية try and catch بتوصيل رسائل نجاح "success" وفشل "failure" المعاملة إلى المستخدم.

نظرا لأن خاصية الاستثناء `Message` تصف المشكلة، فإن عبارة واحدة `Console.WriteLine()` تعالج كلا المشكلتين، من الأسهل قراءة التعليمات البرمجية وصيانتها بعد هذه التحديثات.

٩. استخدم القائمة ملف لحفظ التحديثات.

تحقق من عملك

في هذه المهمة، ستقوم بتشغيل التطبيق، والتحقق من أن التعليمات البرمجية المحدثة تعمل بكفاءة.

١. مرر لأعلى للعثور على حلقة while في عبارات المستوى الأعلى.

يتم استخدام هذه الحلقة للتكرار من خلال المعاملات.

٢. حدد موقع التعليمات التالية قبل بضعة أسطر من بداية حلقة while

```
int transactions = 10;
```

٣. حدث عدد المعاملات إلى 40 كما يلي:

```
int transactions = 40;
```

٤. حدد موقع سطر التعليمات التالية داخل حلقة while

```
int itemCost = valueGenerator.Next(2, 20);
```

٥. حدث منشئ الأرقام العشوائية `itemCost` كما يلي:

```
int itemCost = valueGenerator.Next(2, 50);
```

يعد نطاق التكلفة هذا مناسبًا بشكل أفضل للأصناف التي سيشتريها العملاء.

٦. استخدم القائمة ملف File لحفظ التحديثات.

٧. في قائمة Run حدد Start Debugging

٨. راجع الإخراج في لوحة TERMINAL

٩. تحقق من عرض الرسائل المقترنة بنوعي الاستثناء:

يجب أن يتضمن تقرير المعاملات رسائل "تعذر إجراء معاملة Could not make transaction" التالية:

```
Customer is making a $42 purchase
         Using 2 twenty dollar bills
         Using 0 ten dollar bills
         Using 0 five dollar bills
         Using 0 one dollar bills
Could       not       make       transaction:
InvalidOperationException:  Not  enough  money
provided to complete the transaction.
```

```
Customer is making a $23 purchase
         Using 2 twenty dollar bills
         Using 0 ten dollar bills
         Using 0 five dollar bills
         Using 1 one dollar bills
Cashier prepares the following change:
         A five
         A five
         A one
         A one
Could       not       make       transaction:
InvalidOperationException: The till is unable to
make change for the cash provided.
```

تهانينا، لقد قمت بتصحيح تطبيق التسجيل النقدي the cash register لإصلاح مشكلة منطق التعليمات البرمجية، وقمت بتحديث التطبيق لاستخدام تقنيات معالجة الاستثناءات المناسبة.

ملاحظة

يُظهر الناتج المُبلغ عنه أن أموال درج النقود till لم تعد متوازنة، هناك أخطاء منطقية إضافية في التعليمات البرمجية، تتوفر وحدة مشروع تحدي إذا كنت مهتماً بإظهار مهاراتك في تصحيح أخطاء Visual Studio Code

٥ التحقق من المعرفة

١- ما الغرض من التقاط استثناء catching an exception في C#؟

- لتجاهل الأخطاء التي تحدث في البرنامج.
- لإنشاء استثناءات أخرى في البرنامج.
- لاتخاذ إجراء تصحيحي عند حدوث خطأ في البرنامج.

٢- ما هي العلاقة بين نوع الاستثناء type of exception والمعلومات التي يحتوي عليها؟

- نوع الاستثناء والمعلومات التي يحتوي عليها غير مرتبطين.
- يحدد نوع الاستثناء المعلومات التي يحتوي عليها.
- تحدد المعلومات الواردة في استثناء نوع الاستثناء.

٣- ما هي نقطة التوقف الشرطية conditional breakpoint في Visual Studio Code؟

- نقطة توقف يتم تشغيلها فقط عند استيفاء شرط محدد.
- نقطة توقف يتم تشغيلها في كل مرة يتم فيها تشغيل التعليمات البرمجية.
- نقطة توقف مرئية فقط في المحرر ولا تؤثر على تصحيح الأخطاء.

٤- متى يتم طرح استثناء `ArgumentOutOfRangeException`؟

- يتم طرح استثناء `ArgumentOutOfRangeException` عند محاولة فهرسة مصفوفة خارج حدود المصفوفة.
- يتم طرح استثناء `ArgumentOutOfRangeException` عندما تكون قيمة وسيطة خارج نطاق القيم المسموح بها كما هو محدد بواسطة الأسلوب.
- يتم طرح استثناء `ArgumentOutOfRangeException` عند محاولة تخزين قيمة من نوع محدد في مصفوفة من نوع آخر.

٥- ما هو النهج الموصي به لالتقاط الاستثناءات catching exceptions في C#؟

- التقاط Catch أي نوع من الاستثناء دون تحديد وسيطة كائن object argument
- التقاط Catch الاستثناءات التي تعرف التعليمات البرمجية الخاصة بك كيفية التعامل معها.
- التقاط Catch الاستثناءات غير المشتقة من System.Exception فقط.

راجع إجابتك

١ لاتخاذ إجراء تصحيحي عند حدوث خطأ في البرنامج.

صحيح الغرض من التقاط استثناء catching an exception هو اتخاذ إجراء تصحيحي عند حدوث خطأ

٢ يحدد نوع الاستثناء المعلومات التي يحتوي عليها.

صحيح يحدد نوع الاستثناء type of exception المعلومات التي يحتوي عليها

٣ نقطة توقف يتم تشغيلها فقط عند استيفاء شرط محدد.

صحيح نقطة التوقف الشرطية conditional breakpoint هي نوع خاص من نقاط التوقف التي يتم تشغيلها فقط عند استيفاء شرط محدد

٤ يتم طرح استثناء `ArgumentOutOfRangeException` عندما تكون قيمة وسيطة خارج نطاق القيم المسموح بها كما هو محدد بواسطة الأسلوب.

صحيح يجب طرح استثناء `ArgumentOutOfRangeException` فقط عندما تكون قيمة الوسيطة خارج نطاق القيم المسموح بها كما هو محدد بواسطة الأسلوب الذي تم استدعاؤه

٥ التقاط Catch الاستثناءات التي تعرف التعليمات البرمجية الخاصة بك كيفية التعامل معها.

صحيح النهج الموصي به هو التقاط الاستثناءات التي تعرف التعليمات البرمجية الخاصة بك كيفية التعامل معها فقط

٦ الملخص

كان هدفك هو اكتساب خبرة مع مصحح أخطاء Visual Studio Code debugger وتنفيذ نمطا `try-catch` وإنشاء وطرح الاستثناءات create and throw exceptions التي يتم اكتشافها في مستوى أقل من مكدس الاستدعاءات call stack

من خلال مراجعة تطبيق التسجيل النقدي، وتصحيح أخطاء الأسلوب `MakeChange` وتنفيذ تقنيات معالجة الاستثناء exception handling في كلاً من الأسلوب واستدعاء التعليمات البرمجية، اكتسبت التجربة التي تريدها.

لقد استخدمت أدوات مصحح الأخطاء debugger tools لتكوين نقطة توقف breakpoint في عبارات المستوى الأعلى للتطبيق، مع إيقاف التنفيذ مؤقتاً execution paused قمت بالتنقل عبر التعليمات البرمجية لعزل مشكلة في منطق التعليمات، لقد نفذت نمطا `try-catch` في عبارات المستوى الأعلى.

وأنشأت وطرحت created and threw استثناء `InvalidOperationException` في الأسلوب `MakeChange` ثم قمت بتحديث عبارة catch لالتقاط نوع الاستثناء `InvalidOperationException` فقط.

تمكنك القدرة على تتبع أخطاء تطبيقات C# وتنفيذ معالجة الاستثناء من تطوير تطبيقات مستقرة وموثوقة.

الوحدة السادسة

مشروع التحدي - تصحيح تطبيق وحدة تحكم C#

في هذه الوحدة، ستقوم بتشغيل تطبيق لتحديد المشكلات المنطقية logic issues ثم استخدام أدوات مصحح أخطاء Visual Studio Code debugger tools لعزل المشكلة وإصلاحها.

الأهداف التعليمية

- أظهر قدرتك على استخدام أدوات مصحح أخطاء Visual Studio Code لتحديد مشكلة في منطق التعليمات البرمجية وتصحيحها.

محتويات الوحدة: -

١- مقدمة

٢- الإعداد

٣- تمرين - استخدم بيانات الاختبار لكشف المشكلات المنطقية في تطبيق وحدة تحكم C#

٤- تمرين - استخدم مصحح الأخطاء لعزل وإصلاح المشكلات المنطقية في تطبيق وحدة تحكم C#

٥- التحقق من المعرفة

٦- الملخص

١ المقدمة

يوفر Visual Studio Code أدوات رائعة لتصحيح أخطاء التعليمات البرمجية لمطوري C# يمكنك العثور على ملحقات مصحح الأخطاء ل C# ولغات البرمجة الأخرى في Visual Studio Code Marketplace

لنفترض أنك جزء من فريق يعمل على طلب تسجيل نقدي، تقوم بتطوير `MakeChange` طريقة تدير المال في الدرج أو الخزينة، تحسب مقدار التغيير الذي تم إرجاعه إلى العميل، يتتبع الأسلوب `MakeChange` عدد الأوراق النقدية لكل فئة (1, 5, 10, and 20) المتوفرة في درج النقود، يتم طرح استثناءات عندما لا تغطي النقود المستلمة تكلفة الصنف، وعندما لا يتمكن الدرج من إجراء تغيير باستخدام الأوراق النقدية المتاحة، اجتاز الأسلوب `MakeChange` اختبارات التحقق من التعليمات البرمجية الأولية، ولكن نموذج بيانات أكبر، يعرض أخطاء منطقية، تحتاج إلى عزلها وإصلاحها قبل أن يتم إصدار التعليمات البرمجية.

في هذه الوحدة، يمكنك إكمال المهام التالية:

- تشغيل تطبيق التسجيل النقدي ومراجعة الإخراج الذي تم إنشاؤه لتحديد المشكلة المنطقية.
- استخدم أدوات مصحح أخطاء debugger tools لعزل المشكلة وإصلاحها.

في نهاية هذه الوحدة، سيوازن الأسلوب `MakeChange` بنجاح الأموال في درج النقود، أثناء محاكاة المعاملات اليومية.

ملاحظة

هذه وحدة مشروع تحدي حيث ستكمل مشروعًا شاملاً كامل المواصفات، تهدف هذه الوحدة إلى أن تكون بمثابة اختبار لمهاراتك؛ هناك القليل من الإرشادات، ولا توجد تعليمات خطوة بخطوة.

٢ الإعداد

في مشروع التحدي هذا، ستستخدم أدوات مصحح أخطاء debugger tools لتصحيح أخطاء تطبيق وحدة تحكم C#

مواصفات المشروع

يتضمن مشروع التعليمات البرمجية المبدئي Starter لهذه الوحدة ملف Program.cs مع ميزات التعليمات البرمجية التالية:

محاكاة المعاملات: تعمل عبارات المستوى الأعلى على تكوين بيانات التطبيق ومحاكاة سلسلة من المعاملات باستخدام إما مصفوفة `testData` أو عدد أكبر من المعاملات التي يتم إنشاؤها عشوائياً.

تهيئة درج النقود: يتم استخدام الأسلوب `LoadTillEachMorning` لتكوين السجل النقدي لدرج النقود till مع عدد محدد مسبقاً من الأوراق النقدية في كل فئة.

معاملات العملية: يتم استخدام الأسلوب `MakeChange` لإدارة النقد في درج النقود till أثناء معاملات الشراء.

تقرير حالة درج النقود: يتم استخدام الأسلوب `LogTillStatus` لعرض عدد الأوراق النقدية لكل فئة موجودة حالياً في الدرج.

تقرير رصيد درج النقود: يتم استخدام الأسلوب `TillAmountSummary` لعرض رسالة توضح المبلغ النقدي الموجود في الدرج.

يحتوي الأسلوب `MakeChange` على مشكلات منطقية، تمنعه من موازنة الأموال بنجاح في الدرج، أثناء معاملات المحاكاة تحتاج إلى استخدام أدوات مصحح أخطاء Visual Studio Code لعزل مشكلات المنطق وإصلاحها.

للتأكد من أن أسلوبك `MakeChange` يعمل بشكل صحيح، يجب أن تتحقق التعليمات من تحقيق توازن ناجح في الدرج، في ظل الشروط التالية:

- تحاكي عبارات المستوى الأعلى المعاملات باستخدام تكاليف العنصر/الصنف التي تم إنشاؤها عشوائياً.

- تنشئ عبارات المستوى الأعلى قيماً عشوائية ل `itemCost` في النطاق `2 - 49`
- تحاكي عبارات المستوى الأعلى 100 معاملة.
- يتم موازنة درج النقود till بنجاح عندما تكون قيمته المُبلغ عنها، مساوية لقيمة الدرج till المتوقعة، على سبيل المثال:

```
The till has 1184 dollars
Expected till value: 1184
```

الإعداد

استخدم الخطوات التالية للتحضير لتمارين مشروع التحدي:

١. لتنزيل ملف مضغوط يحتوي على مشروع Starter حدد الارتباط التالي Lab Files

٢. فك ضغط ملفات التنزيل.

- على جهازك المحلي، انتقل إلى مجلد التنزيلات.
- انقر بزر الماوس الأيمن فوق Challenge-project-debugging-CSharp-main.zip ثم حدد استخراج الكل.
- حدد إظهار الملفات المستخرجة عند اكتمالها، ثم حدد استخراج.
- دون موقع المجلد المستخرج.

٣. انسخ مجلد ChallengeProject المستخرج إلى مجلد سطح مكتب Windows

٤. افتح مجلد ChallengeProject الجديد في Visual Studio Code

- افتح Visual Studio Code في بيئة التطوير الخاصة بك.
- في Visual Studio Code في القائمة File حدد Open Folder
- انتقل إلى مجلد Windows Desktop وحدد موقع مجلد "ChallengeProject"
- حدد ChallengeProject ثم حدد Select Folder

يجب أن تظهر قائمة استكشاف EXPLORER مجلد ChallengeProject ومجلدين فرعيين باسم Final وStarter

٣ تمرين ـ استخدم بيانات الاختبار لكشف المشكلات المنطقية في تطبيق وحدة تحكم C#

الأسلوب `MakeChange` قادر على معالجة المعاملات المحاكية بنجاح عند استخدام المصفوفة `testData` ومع ذلك، يتم كشف مشكلات منطقية عند محاكاة المعاملات باستخدام مجموعة بيانات أكبر من الأصناف المسعرة بشكل عشوائي، توضح هذه المشكلة أهمية اختبار تطبيقاتك بدقة.

المواصفات

في تمرين التحدي هذا، تحتاج إلى تكوين المتغيرات المستخدمة لمحاكاة المعاملات، والتحقق من أن تقرير المعاملات يتضمن التناقضات المتوقعة.

تنطبق المتطلبات التالية على المعاملات المحاكاة:

- تحاكي عبارات المستوى الأعلى المعاملات باستخدام تكاليف الصنف التي تم إنشاؤها عشوائياً.
- تنشئ عبارات المستوى الأعلى قيماً عشوائية ل `itemCost` في النطاق `2 - 49`
- تحاكي عبارات المستوى الأعلى 100 معاملة.

يجب أن يتضمن إخراج المعاملة المبلغ عنها ما يلي:

- سجل 100 معاملة تمت محاولة تنفيذها.
- مثيلات رسالة تفيد تعذر إجراء معاملة:

"Could not make transaction: InvalidOperationException: Not enough money provided to complete the transaction" لا تتوفر أموال كافية لإكمال المعاملة

- مثيلات رسالة تفيد تعذر إجراء معاملة:

"Could not make transaction: InvalidOperationException: The till is unable to make change for the cash provided" يتعذر على الدرج إجراء تغيير على الأموال النقدية المقدمة

- وجود تناقض بين القيم المتوقعة بالدرج التي تم الإبلاغ عنها.

راجع عملك

للتحقق من أن التطبيق ينتج النتائج المتوقعة عند تنفيذ متطلبات المحاكاة المحددة، أكمل الخطوات التالية:

١. تأكد من فتح مجلد ChallengeProject في Visual Studio Code

٢. في قائمة EXPLORER قم بتوسيع المجلدين ChallengeProject وStarter

يحتوي مجلد Starter على نموذج التطبيق لوحدة المشروع الإرشادية هذه.

٣. افتح ملف Program.cs في محرر التعليمات البرمجية Visual Studio

٤. تكوين بيئة تتبع الأخطاء debug environment وتكوين التشغيل launch configuration

٥. تكوين التطبيق لاستخدام المصفوفة `testData` للمعاملات المحاكاة (`useTestData = true`)

٦. تشغيل التطبيق في جلسة تصحيح الأخطاء debug session

٧. تحقق من تساوي قيمتي " المُبلغ عنها reported" و "المُتوقعة expected" لدرج النقود، في تقرير المعاملات الذي تم إنشاؤه بواسطة التطبيق.

٨. تكوين متغيرات التطبيق لتلبية متطلبات المحاكاة في قسم المواصفات Specification

٩. تشغيل التطبيق في جلسة تصحيح الأخطاء.

١٠. تحقق من أن تطبيقك يحقق متطلبات الإخراج التالية:

- يضم تقرير المعاملات سجلاً ب 100 معاملة تمت محاولة تنفيذها.
- يتضمن تقرير المعاملات تناقضًا بين القيم المتوقعة لدرج النقود المُبلغ عنها
- يتضمن تقرير المعاملات مثيلات الرسائل التالية:

```
Could not make transaction: InvalidOperationException: Not enough money provided to complete the transaction.
```

```
Could not make transaction: InvalidOperationException: The till is unable to make change for the cash provided.
```

بمجرد التحقق من صحة نتائج هذا التمرين، انتقل إلى التمرين التالي في هذا التحدي.

٤ تمرين - استخدم مصحح الأخطاء لعزل وإصلاح المشكلات المنطقية في تطبيق وحدة تحكم C#

في تمرين التحدي هذا، تحتاج إلى استخدام أدوات مصحح أخطاء، لعزل وإصلاح المشكلات التي تمنع الأسلوب `MakeChange` من موازنة الدرج بنجاح، بمجرد تحديث التطبيق، تحتاج إلى التحقق من نتائجك.

المواصفات

تنطبق متطلبات المواصفات التالية على المعاملات:

- تحاكي عبارات المستوى الأعلى المعاملات باستخدام تكاليف الأصناف التي تم إنشاؤها عشوائيا.
- تنشئ عبارات المستوى الأعلى قيما عشوائية ل `itemCost` في النطاق `2 - 49`
- تحاكي عبارات المستوى الأعلى 100 معاملة

يجب أن تتضمن مخرجات المعاملة المبلغ عنها ما يلي:

- سجل ١٠٠ معاملة تمت محاولة تنفيذها.
- مثيلات رسالة تفيد تعذر إجراء المعاملة:

"Could not make transaction: InvalidOperationException: Not enough money provided to complete the transaction"

- مثيلات رسالة تفيد تعذر إجراء المعاملة:

"Could not make transaction: InvalidOperationException: The till is unable to make change for the cash provided"

- قيمة درج النقود المبلغ عنها تساوي قيمة الدرج المتوقعة.

تتبع أخطاء التطبيق

استخدم الخطوات التالية لعزل مشكلات المنطق وإصلاحها:

١. تكوين بيئة تتبع الأخطاء.

٢. تشغيل التطبيق في جلسة تصحيح الأخطاء.

٣. استخدم أدوات مصحح أخطاء Visual Studio Code لعزل مشكلات المنطق وإصلاحها.

٤. احفظ التطبيق المحدث.

راجع عملك

للتحقق من أن التطبيق المحدث ينتج النتائج المتوقعة عند تنفيذ متطلبات المحاكاة المحددة، أكمل الخطوات التالية:

١. افتح ملف Program.cs في محرر التعليمات البرمجية Visual Studio

٢. تكوين متغيرات التطبيق لتلبية متطلبات المعاملة المحاكاة في قسم المواصفات.

٣. تشغيل التطبيق المحدث.

٤. تحقق من أن تطبيقك يحقق متطلبات الإخراج التالية:

- يتضمن تقرير المعاملات 100 سجل معاملة تمت محاولة تنفيذها.
- يتضمن تقرير المعاملات قيمة درج النقود المبلغ عنها تساوي قيمة الدرج المتوقعة.
- يتضمن تقرير المعاملات مثيلات الرسائل التالية:

```
Could not make transaction: InvalidOperationException: Not enough money provided to complete the transaction.
```

```
Could not make transaction: InvalidOperationException: The till is unable to make change for the cash provided.
```

تهانينا إذا نجحت في هذا التحدي!

٥ تحقق من المعرفة

١- كيف يمكن للمطور إنشاء نقطة توقف شرطية conditional breakpoint في Visual Studio Code؟

- انقر بزر الماوس الأيسر في العمود إلى يسار رقم سطر في محرر التعليمات البرمجية.
- حدد تبديل نقطة التوقف الشرطية Toggle Conditional Breakpoint في قائمة Run
- انقر بزر الماوس الأيمن فوق العمود على يسار رقم السطر، ثم حدد إضافة نقطة توقف شرطية Add Conditional Breakpoint

٢- في ملف تكوين التشغيل launch configuration ما هي سمة وحدة التحكم console attribute المستخدمة؟

- تحديد دليل العمل للعملية المستهدفة.
- تحديد نوع مصحح الأخطاء الذي يجب استخدامه لتكوين التشغيل هذا.
- تحديد نوع وحدة التحكم المستخدمة عند تشغيل التطبيق.

٣- أي قسم من قائمة RUN AND DEBUG يستخدم لتعقب نقطة التنفيذ الحالية داخل التطبيق قيد التشغيل؟

- قسم المتغيرات VARIABLES
- قسم CALL STACK
- قسم المشاهدة WATCH

٤- أي من الخيارات التالية يمكن استخدامها لتعيين نقطة توقف set a breakpoint في Visual Studio Code؟

- انقر بزر الماوس الأيسر في العمود الموجود على يسار رقم السطر في محرر التعليمات البرمجية.
- حدد تبديل نقطة التوقف Toggle Breakpoint في القائمة Edit
- انقر بزر الماوس الأيمن في منتصف سطر التعليمات البرمجية، ثم حدد تبديل نقطة التوقف Toggle Breakpoint

راجع إجابتك

١ انقر بزر الماوس الأيمن فوق العمود على يسار رقم السطر، ثم حدد إضافة نقطة توقف شرطية Add Conditional Breakpoint

صحيح يمكن للمطور إنشاء نقطة توقف شرطية بالنقر بزر الماوس الأيمن فوق العمود إلى يسار رقم السطر، ثم تحديد Add Conditional Breakpoint

٢ تحديد نوع وحدة التحكم المستخدمة عند تشغيل التطبيق.

صحيح تحدد سمة console attribute نوع وحدة التحكم المستخدمة عند تشغيل التطبيق، الخيارات هي internalConsole, integratedTerminal, externalTerminal الإعداد الافتراضي هو internalConsole

٣ قسم CALL STACK

صحيح يتم استخدام قسم CALL STACK لتتبع نقطة التنفيذ الحالية داخل التطبيق قيد التشغيل، بدءًا من نقطة الإدخال الأولية إلى التطبيق

٤ انقر بزر الماوس الأيسر في العمود الموجود على يسار رقم سطر في محرر التعليمات البرمجية.

صحيح يمكن تعيين نقطة توقف عن طريق وضع مؤشر الماوس في العمود على يسار رقم السطر ثم النقر بزر الماوس الأيسر

٦ الملخص

كان هدفك هو إظهار القدرة على استخدام أدوات مصحح أخطاء Visual Studio Code debugger tools لتطبيقات C#

من خلال تكوين بيئة تصحيح الأخطاء configuring the debug environment وتصحيح أخطاء debugging تطبيق التسجيل النقدي بنجاح، أظهرت قدرتك على استخدام أدوات مصحح الأخطاء debugger tools ل C# لقد بدأت بالتأكد من تكوين ملف launch.json بشكل صحيح لدعم تطبيقك، ثم قمت بتشغيل التطبيق، وراجعت الإخراج الذي تم إنشاؤه، وحددت مشكلة المنطق، بعد ذلك استخدمت أدوات مصحح أخطاء Visual Studio Code لعزل مشكلات المنطق وإصلاحها، عند الانتهاء، قمت بحفظ التطبيق المحدث، وتشغيل التطبيق لإثبات أن التطبيق تم تنفيذه على النحو المنشود.

توضح قدرتك على تصحيح تطبيق التسجيل النقدي قدرتك على استخدام أدوات مصحح أخطاء Visual Studio Code debugger tools ل C#