200 XP

Exercise - Implement a switch statement

11 minutes

A `switch` statement is a C# selection statement that provides an alternative to an `if-elseif-else` branching construct. The `switch` statement provides advantages over an `if-elseif-else` construct when evaluating a single value against a list of known matching values.

Consider the following scenario:

- You're working on an application related to food nutrition. A section of the code deals with fruits.
- Your code includes a variable named `fruit` that's used to hold the name of different types of fruit.
- You have a list of the 20 fruits that your application is focused on.
- You want to branch your code based on the value assigned to `fruit`.

In this scenario, you can use a `switch` statement to create a separate branch for each type of fruit.

How does a switch statement work?

The `switch` statement chooses one section of code to execute from a list of possible switch sections. The selected *switch section* is chosen based on a pattern match with the statement's match expression.

Consider the following code sample that shows the basic structure of `switch` statement:

C#

```
switch (fruit)
{
    case "apple":
        Console.WriteLine($"App will display information for apple.");
        break;

    case "banana":
        Console.WriteLine($"App will display information for banana.");
        break;
```

```
    case "cherry":
        Console.WriteLine($"App will display information for cherry.");
        break;
}
```

The match expression (which may also be referred to as the *switch expression*) is the value following the `switch` keyword, in this case `(fruit)`. Each *switch section* is defined by a *case pattern*. Case patterns are constructed using the keyword `case` followed by a value. The first case pattern in this example is: `case "apple":`. Case patterns are Boolean expressions that evaluate to either `true` or `false`. Each switch section includes a small number of code lines that will be executed if the case pattern is a match for the match expression. In this example, if `fruit` is assigned a value of "apple", the first case pattern will evaluate as `true` and that switch section will be executed.

A switch statement must include at least one switch section, but will normally contain three or more switch sections.

The switch is best used when:

- You have a single value (variable or expression) that you want to match against many possible values.
- For any given match, you need to execute a couple of lines of code at most.

> **Note**
>
> This first example of a `switch` statement is purposefully simple and your examination of the syntax was brief. You will be examining additional features of the `switch` statement when you work through some more advanced scenarios in the sections below.

It's time to prepare your coding environment and begin developing your own `switch` statements.

Prepare your coding environment

This module includes hands-on activities that guide you through the process of building and running demonstration code. We encourage you to complete these activities using Visual Studio Code as your development environment. Using Visual Studio Code for these activities will help you to become more comfortable writing and running code in a developer environment that's used by professionals worldwide.

1. Open Visual Studio Code.

   You can use the Windows Start menu (or equivalent resource for another OS) to open Visual Studio Code.

2. On the Visual Studio Code **File** menu, select **Open Folder**.

3. In the **Open Folder** dialog, navigate to the Windows Desktop folder.

   If you have a different folder location where you keep code projects, you can use that folder location instead. For this training, the important thing is to have a location that's easy to locate and remember.

4. In the **Open Folder** dialog, select **Select Folder**.

   If you see a security dialog asking if you trust the authors, select **Yes**.

5. On the Visual Studio Code **Terminal** menu, select **New Terminal**.

   Notice that a command prompt in the Terminal panel displays the folder path for the current folder. For example:

   ```dos
   C:\Users\someuser\Desktop>
   ```

   > **Note**
   >
   > If you are working on your own PC rather than in a sandbox or hosted environment and you have completed other Microsoft Learn modules in this C# series, you may have already created a project folder for code samples. If that's the case, you can skip over the next step, which is used to create a console app in the TestProject folder.

6. At the Terminal command prompt, to create a new console application in a specified folder, type **dotnet new console -o ./CsharpProjects/TestProject** and then press Enter.

   This .NET CLI command uses a .NET program template to create a new C# console application project in the specified folder location. The command creates the CsharpProjects and TestProject folders for us, and uses TestProject as the name of our `.csproj` file.

7. In the EXPLORER panel, expand the **CsharpProjects** folder.

You should see the TestProject folder and two files, a C# program file named Program.cs and a C# project file named TestProject.csproj.

8. In the EXPLORER panel, to view your code file in the Editor panel, select **Program.cs**.

9. Delete the existing code lines.

   You'll be using this C# console project to create, build, and run code samples during this module.

10. Close the Terminal panel.

Create and test a switch statement

1. Ensure that you have Visual Studio Code open and Program.cs displayed in the Editor panel.

   > **Note**
   >
   > Program.cs should be empty. If it isn't, select and delete all code lines.

2. Type the following code into the Visual Studio Code Editor:

```c#
int employeeLevel = 200;
string employeeName = "John Smith";

string title = "";

switch (employeeLevel)
{
    case 100:
        title = "Junior Associate";
        break;
    case 200:
        title = "Senior Associate";
        break;
    case 300:
        title = "Manager";
        break;
    case 400:
        title = "Senior Manager";
        break;
    default:
        title = "Associate";
```

```
        break;
}

Console.WriteLine($"{employeeName}, {title}");
```

3. On the Visual Studio Code **File** menu, select **Save**.

   The Program.cs file must be saved before building or running the code.

4. In the EXPLORER panel, to open a Terminal at your TestProject folder location, right-click **TestProject**, and then select **Open in Integrated Terminal**.

   A Terminal panel will open. The Terminal should include a command prompt showing that the Terminal is open to your TestProject folder location.

5. At the Terminal command prompt, to run your code, type **dotnet run** and then press Enter.

   > **Note**
   >
   > If you see a message saying "Couldn't find a project to run", ensure that the Terminal command prompt displays the expected TestProject folder location. For example: `C:\Users\someuser\Desktop\csharpprojects\TestProject>`

   You should see the following output:

   ```
   Output

   John Smith, Senior Associate
   ```

6. Take a minute to review the `switch` statement that you entered.

7. Notice that the `switch` statement defines a single code block.

   The `switch` statement defines a single code block that includes a list of switch sections. To the right of the `switch` keyword is a *switch expression* that's enclosed in parenthesis.

8. Notice the list of switch sections inside the code block.

   The `switch` code block contains a list of *switch sections*, each of which includes one or more switch labels. In addition, each switch section includes a statement list that will execute if the label is equal to the switch expression defined at the top of the switch statement.

The switch expression is evaluated against the case labels from top to bottom until a value that is equal to the switch expression is found. If none of the labels are a match, the statement list for the `default` case will be executed. If no default is included, control is transferred to the end point of the switch statement. Each label must provide a value type that matches the type specified in the switch expression.

> **Note**
>
> The optional `default` label can appear at any position within the list of switch sections. However, most developers choose to put it last because it makes more sense (logically) to position `default` as the final option. Regardless of the position, the `default` section will be evaluated last.

In our example:

- the switch expression is `(employeeLevel)`
- each switch section has a single switch label (`case` or `default`).
- the matching switch section is defined by `case: 200`, since `employeeLevel = 200`.

9. Notice that each switch section is separated from the next.

   Only one switch section is allowed to be executed. This means that execution of a switch section is not permitted to "fall through" to the next switch section. The `break` keyword is one of several ways to end a switch section before it gets to the next section. If you forget the `break` keyword (or, optionally, the `return` keyword) the compiler will generate an error.

Change the level variable value to see how the switch statement evaluates it

To exercise the default case, let's change the employee's level by modifying the value assignment.

1. To modify the value assigned to `employeeLevel`, update your code as follows:

```c#
int employeeLevel = 201;
```

2. Save your code file, and then use Visual Studio Code to run your code.

Enter `dotnet run` from the Terminal command prompt to run your code.

3. Notice that the output has changed.

   Now, when you run the code, you should see the more generic title used.

   Output

   ```
   John Smith, Associate
   ```

   Since the `employeeLevel` doesn't match any labels, the `default` label is matched.

Modify a switch section to include multiple labels

Suppose our company decided to give all level 100 employees the title "Senior Associate" -- the same title as level 200 employees. As the developer, you decide to implement this by removing the first switch section belonging to the label `case 100:`, and instead allow both the `case 100:` and `case 200:` labels to execute the same switch section.

1. To modify the value assigned to `employeeLevel`, update your code as follows:

   c#

   ```
   int employeeLevel = 100;
   ```

2. To assign multiple labels to the first switch section, update your code as follows:

   c#

   ```
   case 100:
   case 200:
       title = "Senior Associate";
       break;
   ```

   When you're finished making changes, your modifications should match the following code:

   c#

   ```
   int employeeLevel = 100;
   string employeeName = "John Smith";

   string title = "";
   ```

```
switch (employeeLevel)
{
    case 100:
    case 200:
        title = "Senior Associate";
        break;
    case 300:
        title = "Manager";
        break;
    case 400:
        title = "Senior Manager";
        break;
    default:
        title = "Associate";
        break;
}

Console.WriteLine($"{employeeName}, {title}");
```

3. Save your code file, and then use Visual Studio Code to run your code.

   Enter `dotnet run` from the Terminal command prompt to run your code.

   You should see the following output:

   Output

   ```
   John Smith, Senior Associate
   ```

   Both of the case labels `100` and `200` are now paired with the switch section that sets the title to the string value `Senior Associate`.

Recap

Here's the main takeaways you learned about the switch statement:

- Use the `switch` statement when you have one value with many possible matches, each match requiring a branch in your code logic.
- A single switch section containing code logic can be matched using one or more labels defined by the `case` keyword.
- Use the optional `default` keyword to create a label and a switch section that will be used when no other case labels match.

Knowledge check

1. A developer writes the code to implement a `switch-case` construct. What is the purpose of the `break` keyword? *

- The `break` keyword tells the runtime to continue evaluating other cases in the `switch` construct.
- The `break` keyword tells the runtime to stop evaluating case patterns and prevents execution of other cases in the `switch` construct.
- The `break` keyword tells the runtime to exit the application.

2. A developer writes the code to implement a `switch-case` construct that evaluates a variable against many possible matching values. They include the `default` keyword as part of their `switch-case` construct. What is the purpose of the `default` keyword? *

- The `default` keyword supplies the default value for the variable if the variable has not been initialized.
- The `default` keyword acts as the matching value when none of the supplied `case` values is a match.
- The `default` keyword supplies a default actions code block that is always executed regardless of the matching case value

3. Which of the following statements about the `switch-case` construct is true? *

- A single switch section can have multiple case labels.
- A switch construct must include a default switch section.
- The colon at the end of the case label is optional.

Check your answers