100 XP

Exercise - Build sample data and menu selection loops

25 minutes

In this exercise, you review the code in the Starter project, perform some code clean-up tasks, and then start adding features to your application. The tasks that you complete during this exercise are:

1. Code review: review the contents of the Program.cs file.
2. Sample data: convert the if-elseif-else structure to switch-case structure that improves readability.
3. Menu loop: enclose the main menu and menu item selection in a loop that iterates until the user enters "exit".
4. Menu selections: write the code for a switch-case structure that establishes separate code branches for each menu option.
5. Code branches: write placeholder within the menu item branches that provides user feedback until app features are developed.
6. Verification test: perform verification tests for the code that you develop in this exercise.

Review the contents of the Program.cs file

In this task, you complete a walkthrough of the Starter project code. The Program.cs file contains a preliminary version of the application that you update during this module. The existing code generates sample data for the application and displays a list of menu options. The menu options represent the main features of your application.

1. Ensure that the **GuidedProject** folder is open in Visual Studio Code.

   The previous unit includes a Setup section that describes how to open the Starter project. If necessary, go back and follow the Setup instructions.

2. In the **EXPLORER** view, expand the **Starter** folder, then select **Program.cs**.

   When you select the Program.cs file, the file contents open in the Visual Studio Code Editor.

If the EXPLORER view isn't open, select EXPLORER from the Visual Studio Code Activity Bar. The EXPLORER button is at the top of the Activity Bar.

3. Take a few minutes to review the code in the Program.cs file.

4. Notice that the top portion of your code begins with some variable declarations.

```c#
// the ourAnimals array will store the following:
string animalSpecies = "";
string animalID = "";
string animalAge = "";
string animalPhysicalDescription = "";
string animalPersonalityDescription = "";
string animalNickname = "";

// variables that support data entry
int maxPets = 8;
string? readResult;
string menuSelection = "";

// array used to store runtime data, there is no persisted data
string[,] ourAnimals = new string[maxPets, 6];
```

At the top of the file, you see a comment line and a list of variables. These variables, `animalSpecies` through `animalNickname`, are used to hold the values of the pet characteristics. Later in the code you assign the characteristic values to a multidimensional string array named `ourAnimals`. Each of these variables is initialized to contain a zero length string `""`. The `ourAnimals` array is declared a little further down in the code.

The next group of variables is a mix of `string` and `int` variables that you'll use to generate sample data, read user input, and establish exit criteria for your main program loop. You may have noticed the code line `string? readResult;`. When used in a variable declaration like this, the `?` character defines a nullable type variable. When reading user entered values with the `Console.ReadLine()` method, it's best to use a nullable type.

The final variable is a two-dimensional string array named `ourAnimals`. Since you're instantiating the array without initializing any values, you use the `new` operator (the `new` operator is used to create a new instance of a type). The number of rows is defined by `maxPets`, which has been initialized to eight. The number of characteristics that you're storing is six, the string variables that you examined above.

5. Scroll down to examine the `for` loop that contains an `if-elseif-else` selection construct.

   > ⓘ **Note**
   >
   > If you see a `switch` statement instead of an `if-elseif-else` inside the code block of your `for` loop, ensure that you are reviewing the Program.cs file in the **Starter** folder, not the Program.cs file in the **Final** folder. You don't want to make changes to the code in the Final folder. You may need to use the Final folder as a reference later in this module.

6. Notice that the `for` loop uses the `maxPets` variable to establish an upper bound on the number of iterations.

7. Notice that the `if-elseif-else` construct selectively branches your code based on pet characteristics.

   The `if-elseif-else` construct is used to define different values for the first four iterations of the `for` loop. After the fourth iteration, all characteristics are assigned an empty, or zero-length, string.

   The values of the animal characteristic variables are assigned to the ourAnimals array at the bottom of the `for` loop.

8. Scroll to the bottom of the code file, and then examine the code that's used to display the menu options and read the user selection.

9. Notice that you assign the value returned by the `Console.ReadLine()` method to the nullable string `readResult`.

   Using a nullable string is best practice for capturing input from the ReadLine() method. Once you verify that the input value isn't null, you assign the value to a standard string variable named `menuSelection`. This process enables you to evaluate the menu selection value without a concern for null values. Many of the methods that accept strings as an input parameter will generate an error if they're passed a null value. If you don't follow this input pattern, the code compiler is likely to generate a warning when you build your project.

   The final lines of the Program.cs file echo the menu option selection and then pause execution until the Enter key is pressed.

Convert if statement to switch statement

In this task, you convert the existing `if-elseif-else` construct to a `switch-case` construct. A `switch` statement improves the readability of your code.

1. Scroll up to the start of the `for` loop that's used to generate your sample data.

2. Notice that the `if-elseif-else` construct begins with the following `if` statement and code block:

```c#
if (i == 0)
{
    animalSpecies = "dog";
    animalID = "d1";
    animalAge = "2";
    animalPhysicalDescription = "medium sized cream colored female golden re-
triever weighing about 65 pounds. housebroken.";
    animalPersonalityDescription = "loves to have her belly rubbed and likes to
chase her tail. gives lots of kisses.";
    animalNickname = "lola";
}
```

3. Replace the initial `if` statement and code block with the following code:

```c#
switch (i)
{
    case 0:
        animalSpecies = "dog";
        animalID = "d1";
        animalAge = "2";
        animalPhysicalDescription = "medium sized cream colored female golden
retriever weighing about 65 pounds. housebroken.";
        animalPersonalityDescription = "loves to have her belly rubbed and likes
to chase her tail. gives lots of kisses.";
        animalNickname = "lola";
        break;
```

The `case 0:` code performs the same selection as the `if (i == 0)` selection that it replaces. You'll be making corresponding replacements to complete the conversion from `if-elseif-`

`else` construct to a `switch-case` construct.

4. Notice that a red squiggly line symbol now appears under the `;` at the end of the `break` statement.

   Visual Studio Code uses a red squiggly line to help you spot issues in your code. In this case, there are a few issues. First, you haven't closed the code block for your `switch` statement. Also, you have an `else if` without the `if`, which isn't allowed. You fix each of these issues as you complete the conversion from an `if` to a `switch`.

5. Replace the `else if (i == 1)` statement and code block with the following code:

```
c#

case 1:
    animalSpecies = "dog";
    animalID = "d2";
    animalAge = "9";
    animalPhysicalDescription = "large reddish-brown male golden retriever
weighing about 85 pounds. housebroken.";
    animalPersonalityDescription = "loves to have his ears rubbed when he greets
you at the door, or at any time! loves to lean-in and give doggy hugs.";
    animalNickname = "loki";
    break;
```

6. Replace the `else if (i == 2)` statement and code block with the following code:

```
c#

case 2:
    animalSpecies = "cat";
    animalID = "c3";
    animalAge = "1";
    animalPhysicalDescription = "small white female weighing about 8 pounds.
litter box trained.";
    animalPersonalityDescription = "friendly";
    animalNickname = "Puss";
    break;
```

7. Replace the `else if (i == 3)` statement and code block with the following code:

```
c#
```

```
case 3:
    animalSpecies = "cat";
    animalID = "c4";
    animalAge = "?";
    animalPhysicalDescription = "";
    animalPersonalityDescription = "";
    animalNickname = "";
    break;
```

8. Replace the `else` statement and code block with the following code:

```c#
default:
    animalSpecies = "";
    animalID = "";
    animalAge = "";
    animalPhysicalDescription = "";
    animalPersonalityDescription = "";
    animalNickname = "";
    break;

}
```

9. Notice that the `if-elseif-else` construct is now completely replaced by a `switch-case` construct and that the red squiggly line symbol is gone.

   If portions of your `if-elseif-else` construct still remain, or if your `switch` statement is incomplete, check to see if you missed a step.

10. Your completed `switch-case` construct should have a structure similar to the following code:

Note

The variables used to assign the pet characteristic values have been removed from the code sample below to shorten the number of code lines. Your code should include the string variable assignments for each case pattern, including the default case.

```c#
switch (i)
{
    case 0:
        // variable assignments were removed for this view of the structure
        break;

    case 1:
        // variable assignments were removed for this view of the structure
        break;

    case 2:
        // variable assignments were removed for this view of the structure
        break;

    case 3:
        // variable assignments were removed for this view of the structure
        break;

    default:
        // variable assignments were removed for this view of the structure
        break;
}
```

Recall that execution of a switch section isn't permitted to "fall through" to the next switch section. The `break` statement is used to ensure that "fall through" doesn't occur.

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

12. In the EXPLORER view, right-click **Starter**, and then select **Open in Integrated Terminal**.

    A TERMINAL panel should open below the code Editor area.

    There are several ways to open Visual Studio Code's integrated terminal. For example, the top menu provides access to the TERMINAL panel from both the **View** menu and the **Terminal** menu. You may also learn keyboard shortcuts that open the TERMINAL panel. Each method is acceptable.

13. Notice that the TERMINAL panel includes a command line prompt, and that the prompt shows the current folder path.

    For example:

    Output

    ```
    C:\Users\someuser\Desktop\GuidedProject\Starter>
    ```

You can use the TERMINAL panel to run Command Line Interface (CLI) commands, such as `dotnet build` and `dotnet run`. The `dotnet build` command will compile your code and display error and warning messages related to your code syntax.

> **Important**
>
> You need to ensure that the terminal command prompt is open to the root of your project workspace. In this case, the root of your project workspace is the Starter folder, where your Starter.csproj and Program.cs files are located. When you run commands in the terminal, the commands will perform actions using current folder location. If you run the `dotnet build` or `dotnet run` command from a folder location that does not contain your files, the command will generate error messages.

14. At the TERMINAL command prompt, to build your project code, enter the following command: `dotnet build`

    After a couple seconds, you should see a message telling you that your build succeeded, and that you have 0 Warning(s) and 0 Error(s).

    Output

    ```
    Determining projects to restore...
    All projects are up-to-date for restore.
    Starter ->
    C:\Users\someuser\Desktop\GuidedProject\Starter\bin\Debug\net6.0\Starter.dll

    Build succeeded.
        0 Warning(s)
        0 Error(s)
    ```

15. If you see Error or Warning messages, you need to fix them before continuing.

    Error and Warning messages list the code line where the issue can be found. The following message is an example of a `Build FAILED` error message:

    `C:\Users\someuser\Desktop\GuidedProject\Starter\Program.cs(53,18): error CS1002: ; expected [C:\Users\someuser\Desktop\GuidedProject\Starter\Starter.csproj]`

    This message tells you the type of error that was detected and where to find it. In this case, the message tells you that the Program.cs file contains an error - `error CS1002: ; expected`. The `; expected` suggests that you forgot to include a `;` at the end of a statement. The

`Program.cs(53,18)` portion of the message tells you that the error is located on code line 53, at a position 18 characters in from the left.

A syntax error like this prevents your Build from succeeding (Build FAILED). Some Build messages provide a "Warning" instead of an "Error", which means there's something to be concerned with, but you can try running the program anyway (Build succeeded).

Once you've fixed the issues and saved your updates, you can run the `dotnet build` command again. Continue until you have 0 Warning(s) and 0 Error(s).

> **Note**
>
> If you have trouble resolving an issue on your own, you can examine the Program.cs code in the Final folder that's included as part of the download that you completed during Setup. The Program.cs code in the Final folder represents the conclusion of all exercises in this module, so it will include code that you haven't created yet. It may look considerably different than the Program.cs code that you've developed at this point in the Guided project. However, you can try examining the Program.cs code in Final to help you isolate and fix an issue in your code. Avoid using the code in the Final folder as a guide. Remember that developers learn from their mistakes and that every developer spends time finding and fixing errors.

16. Close the Terminal panel.

Create program menu loop

In this task, you build a `do` loop that surrounds the menu options and the code that reads user input. This loop ensures that the main menu is refreshed for the user each time they make a menu selection. This loop iterates until the user chooses to exit the program.

1. In the Visual Studio Code Editor, at the top of the code that's used to display the menu options, locate the following comment:

```c#
// display the top-level menu options
```

2. Create a blank code line above the comment, and then enter the following code:

```c#
do
{
```

By enclosing the program's menu options inside a `do` loop, you ensure that the user sees the menu options at least one time.

3. Scroll down to the bottom of the code file.

4. Create a blank code line below the code that pauses code execution, and then update the code as follows:

```c#
// pause code execution
readResult = Console.ReadLine();

} while ();
```

Notice that you've closed the code block for the `do` loop, and that you've begun the construction of your `while` statement.

5. To specify the Boolean expression that's evaluated by the `while` statement, update the `while` statement as follows:

```c#
while (menuSelection != "exit");
```

This expression will ensure that your `do` loop iterates until the user chooses to "exit" the application. Each iteration will display the menu and read the user's menu selection.

6. Notice that the code line indentation isn't what you expect to see inside the code block of your `do` loop.

7. To have Visual Studio Code "fix" the indentation of your code lines, right-click inside the code Editor, and then select **Format Document**.

   In addition to properly indenting code lines, the **Format Document** command will apply other code formatting rules. However, it doesn't fix syntax errors in your code, and it can have unexpected results if your code includes incomplete or malformed code blocks. The **Format Document** command is helpful as long as you're careful.

The keyboard shortcuts to invoke this command are:

- On Windows Shift + Alt + F
- On Mac Shift + Option + F
- On Linux Ctrl + Shift + I

This command can also be found using the Command Palette. To find the Format Document command from the Command Palette:

- On the **View** menu, select **Command Palette**, and then enter **Format D** at the prompt.
- The filtered list of commands will show **Format Document** as a selectable command

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

9. Open the Integrated Terminal panel.

   You can open Terminal panel by right-clicking the **Starter** folder in the EXPLORER, and then selecting **Open in Integrated Terminal**.

10. At the Terminal command prompt, enter the `dotnet build` command to build your updated code.

    You should see a message reporting 0 Warning(s) and 0 Error(s)

11. Fix any Build errors or warnings that you see reported before continuing.

12. At the Terminal command prompt, enter the `dotnet run` command to run your updated code.

    Your application will start running.

13. Verify that the main menu options are displayed in the Terminal panel as expected.

    You should see the following displayed in the TERMINAL panel.

Output

```
Welcome to the Contoso PetFriends app. Your main menu options are:
1. List all of our current pet information
2. Add a new animal friend to the ourAnimals array
3. Ensure animal ages and physical descriptions are complete
4. Ensure animal nicknames and personality descriptions are complete
```

```
5. Edit an animal's age
6. Edit an animal's personality description
7. Display all cats with a specified characteristic
8. Display all dogs with a specified characteristic

Enter your selection number (or type Exit to exit the program)
```

14. At the Terminal command prompt, to select menu option 1, enter **1**

    If your app is paused with the cursor located to the right of the "1", press Enter. To enter the value, you need to press the Enter key after typing the **1** character.

15. Verify that your code echoes back the menu selection that you entered.

    You should see the following output displayed in the terminal.

    Output

    ```
    You selected menu option 1.
    Press the Enter key to continue
    ```

    If you don't see this message, ensure that you saved your code updates before running the `dotnet build` command. If needed, save your code and then try running your application again.

16. At the Terminal command prompt, press the Enter key to continue.

    Pressing Enter enables your code to proceed past the `Console.ReadLine()` method that is located before the `while` expression is evaluated at the end of the application.

17. To verify that your code continues to accept additional menu selections, try entering one or more of the other menu item numbers.

    If your code stopped running after you entered "1", make sure that you saved your code updates. If needed, save your code and then try running your application again.

    You may notice that you can enter any text or numeric value, or even a combination of letters and numbers. You'll address this issue when you develop the code that manages user input.

18. At the Terminal command prompt, to verify that the exit criteria for the loop is evaluated correctly, enter either **Exit** or **exit**.

When you enter either "Exit" or "exit", the `while` expression should evaluate to `false` and the code execution should end.

> ⓘ **Important**
>
> If code execution does not stop as expected when you enter "exit", you can press **Ctrl-C** in the Integrated Terminal to force code execution to stop.

19. Close the Terminal panel.

Write switch statement for menu selections

In this task, you write the code for a `switch` statement that branches your code execution based on the value assigned to `menuSelection`. You create a `switch` label that corresponds to each menu item number. Creating separate `switch` labels ensures that each menu item is managed separately. A `switch` statement does a good job of implementing the intended logic of your application.

1. In the Visual Studio Code Editor, locate the following `Console.WriteLine()` code line that's used to echo back the menu selection.

```c#
Console.WriteLine($"You selected menu option {menuSelection}.");
```

2. To begin the development of your `switch` statement, update your code as follows:

```c#
//Console.WriteLine($"You selected menu option {menuSelection}.");
//Console.WriteLine("Press the Enter key to continue");

// pause code execution
//readResult = Console.ReadLine();

switch(menuSelection)
{

}
```

3. To add the first `case` pattern to your `switch` statement, update your `switch` statement code as follows:

   c#

   ```
   switch(menuSelection)
   {
       case "1":
           break;

   }
   ```

   You may have noticed that the case pattern used here looks slightly different from the case pattern used when building the sample data earlier in the application (`case 1:` versus `case "1":`). The explanation is simple. You were checking integer values previously and now you're checking string values.

4. To create selection branches in your `switch-case` construct for each of the remaining menu selection numbers, add additional `case` options as follows:

   c#

   ```
   switch(menuSelection)
   {
       case "1":
           break;

       case "2":
           break;

       case "3":
           break;

       case "4":
           break;

       case "5":
           break;

       case "6":
           break;

       case "7":
           break;

       case "8":
   ```

```
        break;

}
```

5. Notice that you can add the optional `default` case to the end of your selection list as follows:

```c#
switch(menuSelection)
{
    // case selections 1-7 have been removed to simplify this sample code

    case "8":
        break;

    default:
        break;
}
```

6. On the Visual Studio Code **File** menu, select **Save**.
7. Open the Integrated Terminal panel and then run the command to Build the program.

   Use the `dotnet build` command to build your updated code.

   You should see the message reporting 0 Warning(s) and 0 Error(s)
8. Fix any Build errors or warnings that you see reported before continuing.
9. Close the Terminal panel.

Write placeholder code for each case of the switch statement

In this task, you update the code execution paths created by your `switch` statement. When you're done, each `switch` section will provide feedback to the user that acknowledges their menu selection. Providing feedback that acknowledges user input is especially important since you won't be able to complete the code for all menu options during this module. The feedback lets the user know that the code recognized their input, even if the application isn't able to process the request.

1. In the Visual Studio Code Editor, locate the code line containing the `case "1":` selection:

```c#
switch (menuSelection)
{
    case "1":
        break;
```

2. To provide a feedback message to the user between the `case "1":` and `break;` code lines, update your code as follows:

```c#
switch (menuSelection)
{
    case "1":
        // List all of our current pet information
        Console.WriteLine("this app feature is coming soon - please check back 
to see progress.");
        Console.WriteLine("Press the Enter key to continue.");
        readResult = Console.ReadLine();
        break;
```

3. Add the same feedback message for the `case "2":` selection as follows:

```c#
switch (menuSelection)
{
    case "1":
        // List all of our current pet information
        Console.WriteLine("this app feature is coming soon - please check back 
to see progress.");
        Console.WriteLine("Press the Enter key to continue.");
        readResult = Console.ReadLine();
        break;

    case "2":
        // List all of our current pet information
        Console.WriteLine("this app feature is coming soon - please check back 
to see progress.");
        Console.WriteLine("Press the Enter key to continue.");
        readResult = Console.ReadLine();
        break;
```

4. Update the Line Comment for the `case "2":` selection to reflect the second menu option as follows:

```c#
case "2":
    // Add a new animal friend to the ourAnimals array
    Console.WriteLine("this app feature is coming soon - please check back to see progress.");
    Console.WriteLine("Press the Enter key to continue.");
    readResult = Console.ReadLine();
    break;
```

5. For the `case "3":` through `case "8":` selections, update your code with a Line Comment that reflects the associated menu item text.

   For example, add `// Ensure animal ages and physical descriptions are complete` below `case "3":`

   You can copy the menu item text from the code lines that display the menu options.

   Note

   You can use the "Format Document" command to clean up your formatting. This may speed up your work and give you more time to focus on the code that you're entering rather than how it looks.

6. For the `case "3":` and `case "4":` selections, add the following feedback message and ReadLine() below the Line Comment:

```c#
Console.WriteLine("Challenge Project - please check back soon to see progress.");
Console.WriteLine("Press the Enter key to continue.");
readResult = Console.ReadLine();
```

7. For the remaining `case` selections, update your code with the following feedback message and ReadLine():

   ```c#
   Console.WriteLine("UNDER CONSTRUCTION - please check back next month to see progress.");
   Console.WriteLine("Press the Enter key to continue.");
   readResult = Console.ReadLine();
   ```

   Note

   If you added the optional `default` case to your selection list, don't add the feedback message to that case.

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

   Recall that Visual Studio Code will format your code if you right-click inside the code Editor and then select **Format Document** from the context menu.

9. Open the Integrated Terminal panel, and then run the command to Build the program.

   Use the `dotnet build` command to build your updated code.

10. Fix any Build errors or warnings that you see reported before continuing.

Check your work

In this task, you run your application from the Integrated Terminal and verify that your `switch` statement is branching your code as intended. You also verify that the expected feedback messages are being displayed for each menu selection. The exit criteria for your main menu loop will also be retested.

1. If necessary, open Visual Studio Code's Integrated Terminal panel.

2. At the Terminal command prompt, enter **dotnet run**

3. At the Terminal command prompt, to select the first menu option, enter **1**

4. Verify that the expected "coming soon" message is displayed.

5. Repeat the previous step, entering each of the other menu option numbers to verify that the expected message is displayed in each case.

6. At the Terminal command prompt, enter either **Exit** or **exit** to verify that your exit criteria is still being evaluated correctly.

   Important

   If code execution does not stop as expected, you can press **Ctrl-C** in the Integrated Terminal to force execution to stop.

7. Close the Terminal panel.

Next unit: Exercise - Write code to display all ourAnimals array data

Continue >