100 XP

Exercise - Write code to read and save new ourAnimals array data

20 minutes

In this exercise, you develop the data entry validation loops for each pet characteristic and then store the new `ourAnimals` array data. The detailed tasks that you complete during this exercise are:

1. Read and validate species: build a loop and the inner code structure used to enter and validate the pet species.
2. Construct pet ID: write the code that uses petCount and the species name to construct the petID value.
3. Read and validate age: build a loop and the inner code structure used to enter and validate the pet's age.
4. Read and validate physical description: build a loop and the inner code structure used to enter a physical description of the pet.
5. Read and validate personality description: build a loop and the inner code structure used to enter a description of the pet's personality.
6. Read and validate nickname: build a loop and the inner code structure used to enter a nickname for the pet.
7. Verification test: perform verification tests for the code that you develop in this exercise.

Important

You must complete the previous exercise in this module before starting this exercise.

Build loop to read and validate the pet species

In this task, you create a `do` loop that iterates until the user enters a valid species name, either **dog** or **cat**. You reuse your voidable string `readResult` to capture the `Console.ReadLine()` input. You also reuse the `animalSpecies` string variable that you used when generating your sample data. You add a new Boolean variable named `validEntry` to your app. You use `validEntry` in an expression that's evaluated as an exit criteria for your `do` loop.

1. Ensure that Visual Studio Code is open, and that your Program.cs file is visible in the Editor.

2. Locate the `while (anotherPet == "y" && petCount < maxPets)` statement, and then add a blank code line at the top code block.

3. On the blank code line that you created, to declare `validEntry` with an initial value of `false`, enter the following code:

   C#
   ```
   bool validEntry = false;
   ```

4. On the line below the declaration of `validEntry`, to create a `do` loop for species data entry, enter the following code:

   C#
   ```
   // get species (cat or dog) - string animalSpecies is a required field
   do
   {
   } while (validEntry == false);
   ```

5. Inside the code block of your `do` statement, to create a display prompt and read the user input, enter the following code:

   C#
   ```
   Console.WriteLine("\n\rEnter 'dog' or 'cat' to begin a new entry");
   readResult = Console.ReadLine();
   ```

6. To ensure that the value of `readResult` is NOT null before assigning the value of `readResult` to `animalSpecies`, enter the following code:

   C#
   ```
   if (readResult != null)
   {
       animalSpecies = readResult.ToLower();
   ```

```
}
```

7. On the line below the `animalSpecies` value assignment, to ensure that `animalSpecies` contains a valid species name, enter the following code:

```c#
if (animalSpecies != "dog" && animalSpecies != "cat")
{
    validEntry = false;
}
else
{
    validEntry = true;
}
```

8. Compare your completed species name data entry loop with the following code:

```c#
// get species (cat or dog) - string animalSpecies is a required field
do
{
    Console.WriteLine("\n\rEnter 'dog' or 'cat' to begin a new entry");
    readResult = Console.ReadLine();
    if (readResult != null)
    {
        animalSpecies = readResult.ToLower();
        if (animalSpecies != "dog" && animalSpecies != "cat")
        {
            //Console.WriteLine($"You entered: {animalSpecies}.");
            validEntry = false;
        }
        else
        {
            validEntry = true;
        }
    }
} while (validEntry == false);
```

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

10. Open the Integrated Terminal panel in Visual Studio Code and enter the command to Build your program.

If any Build errors or warnings were reported, fix the issues before continuing.

Construct the animal ID value

In this task, you use the `animalSpecies` and `petCount` variables to create the value that you assign to `animalID`.

1. Add a blank code line below the code block of your species name data entry loop.

2. To create and assign the `animalID` value, enter the following code:

```c#
// build the animal the ID number - for example C1, C2, D3 (for Cat 1, Cat 2, Dog 3)
animalID = animalSpecies.Substring(0, 1) + (petCount + 1).ToString();
```

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

Build loop to read and validate the pet's age

In this task, you create a `do` loop that iterates until the user enters either a `?` or a valid integer that represents the age of the pet in years. You reuse the voidable string `readResult` to capture the `Console.ReadLine()` input. You also reuse the `animalAge` string variable that you used when generating the sample data. To test whether or not the `animalAge` string represents a valid integer, you use the `validEntry` Boolean. You declare a new integer variable named `petAge` to store the numeric value. Once again, the `validEntry` Boolean is used in the expression that's evaluated as an exit criteria for our `do` loop.

1. Add a blank code line below the line used to assign a value to our `animalID` variable.

2. To create a `do` loop for age data entry, enter the following code:

```c#
// get the pet's age. can be ? at initial entry.
do
{
} while (validEntry == false);
```

3. Inside the code block of your `do` statement, to declare an integer variable named `petAge`, enter the following code:

```c#
int petAge;
```

4. On the line below the declaration of `petAge`, to display a message prompt and read the user input, enter the following code:

```c#
Console.WriteLine("Enter the pet's age or enter ? if unknown");
readResult = Console.ReadLine();
```

5. To ensure that the value of `readResult` isn't null before assigning the value of `readResult` to `animalAge`, enter the following code:

```c#
if (readResult != null)
{
    animalAge = readResult;

}
```

6. On the line below the `animalAge` value assignment, to check whether the user entered `?` before testing for a valid integer, enter the following code:

```c#
if (animalAge != "?")
{
    validEntry = int.TryParse(animalAge, out petAge);
}
else
{
    validEntry = true;
}
```

7. Compare your completed age data entry loop with the following code:

```c#
// get the pet's age. can be ? at initial entry.
do
{
    int petAge;
    Console.WriteLine("Enter the pet's age or enter ? if unknown");
    readResult = Console.ReadLine();
    if (readResult != null)
    {
        animalAge = readResult;
        if (animalAge != "?")
        {
            validEntry = int.TryParse(animalAge, out petAge);
        }
        else
        {
            validEntry = true;
        }
    }
} while (validEntry == false);
```

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

9. Open the Integrated Terminal panel in Visual Studio Code and enter the command to Build your program.

   If any Build errors or warnings were reported, fix the issues before continuing.

Build loop to read and validate the pet's physical description

In this task, you create a `do` loop that iterates until the user enters a string value that represents a physical description of a pet. You reuse the voidable string `readResult` to capture the `Console.ReadLine()` input. You also reuse the `animalPhysicalDescription` string variable that you used when generating the sample data. You use the value assigned to `animalPhysicalDescription` in the expression that's evaluated as an exit criteria for our `do` loop.

1. Add a blank code line below the code block of your age data entry loop.

2. To create a `do` loop for physical description data entry, enter the following code:

   C#

   ```
   // get a description of the pet's physical appearance/condition - animalPhysi-
   calDescription can be blank.
   do
   {
   } while (animalPhysicalDescription == "");
   ```

3. Inside the code block of your `do` statement, to create a display prompt and read the user input, enter the following code:

   C#

   ```
   Console.WriteLine("Enter a physical description of the pet (size, color, gender,
   weight, housebroken)");
   readResult = Console.ReadLine();
   ```

4. To ensure that the value of `readResult` isn't null before assigning the value of `readResult` to `animalPhysicalDescription`, enter the following code:

   C#

   ```
   if (readResult != null)
   {
       animalPhysicalDescription = readResult.ToLower();

   }
   ```

5. To assign a value of `"tbd"` to `animalPhysicalDescription` when the value entered is `""`, enter the following code:

   C#

   ```
   if (animalPhysicalDescription == "")
   {
       animalPhysicalDescription = "tbd";
   }
   ```

6. Compare your completed physical description data entry loop with the following code:

```
c#

// get a description of the pet's physical appearance/condition - animalPhysi-
calDescription can be blank.
do
{
    Console.WriteLine("Enter a physical description of the pet (size, color,
gender, weight, housebroken)");
    readResult = Console.ReadLine();
    if (readResult != null)
    {
        animalPhysicalDescription = readResult.ToLower();
        if (animalPhysicalDescription == "")
        {
            animalPhysicalDescription = "tbd";
        }
    }
} while (animalPhysicalDescription == "");
```

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

8. Open the Integrated Terminal panel in Visual Studio Code and enter the command to Build your program.

   If any Build errors or warnings were reported, fix the issues before continuing.

Build loop to read and validate the pet's personality description

In this task, you create a `do` loop that iterates until the user enters a string value that represents a description of a pet's personality. You reuse the voidable string `readResult` to capture the `Console.ReadLine()` input. You also reuse the `animalPersonalityDescription` string variable that you used when generating the sample data. You use the value assigned to `animalPersonalityDescription` in the expression that's evaluated as an exit criteria for our `do` loop.

1. Add a blank code line below the code block of your physical description data entry loop.

2. To create a `do` loop for personality description data entry, enter the following code:

```
c#
```

```
// get a description of the pet's personality - animalPersonalityDescription can
be blank.
do
{
} while (animalPersonalityDescription == "");
```

3. Inside the code block of your `do` statement, to create a display prompt and read the user input, enter the following code:

```c#
Console.WriteLine("Enter a description of the pet's personality (likes or dis-
likes, tricks, energy level)");
readResult = Console.ReadLine();
```

4. To ensure that the value of `readResult` isn't null before assigning the value of `readResult` to `animalPersonalityDescription`, enter the following code:

```c#
if (readResult != null)
{
    animalPersonalityDescription = readResult.ToLower();

}
```

5. To assign a value of `"tbd"` to `animalPersonalityDescription` when the value entered is `""`, enter the following code:

```c#
if (animalPersonalityDescription == "")
{
    animalPersonalityDescription = "tbd";
}
```

6. Compare your completed physical description data entry loop with the following code:

c#

```
// get a description of the pet's personality - animalPersonalityDescription can
be blank.
do
{
    Console.WriteLine("Enter a description of the pet's personality (likes or
dislikes, tricks, energy level)");
    readResult = Console.ReadLine();
    if (readResult != null)
    {
        animalPersonalityDescription = readResult.ToLower();
        if (animalPersonalityDescription == "")
        {
            animalPersonalityDescription = "tbd";
        }
    }
} while (animalPersonalityDescription == "");
```

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

8. Open the Integrated Terminal panel in Visual Studio Code and enter the command to Build your program.

   If any Build errors or warnings were reported, fix the issues before continuing.

Build loop to read and validate the pet's nickname

In this task, you create a `do` loop that iterates until the user enters a string value that represents a nickname for a pet. You reuse the voidable string `readResult` to capture the `Console.ReadLine()` input. You also reuse the `animalNickname` string variable that you used when generating the sample data. You use the value assigned to `animalNickname` in the expression that's evaluated as an exit criteria for our `do` loop.

1. Add a blank code line below the code block of your personality description data entry loop.

2. To create a `do` loop for personality description data entry, enter the following code:

```c#
// get the pet's nickname. animalNickname can be blank.
do
{
} while (animalNickname == "");
```

3. Inside the code block of your `do` statement, to create a display prompt and read the user input, enter the following code:

```c#
Console.WriteLine("Enter a nickname for the pet");
readResult = Console.ReadLine();
```

4. To ensure that the value of `readResult` isn't null before assigning the value of `readResult` to `animalNickname`, enter the following code:

```c#
if (readResult != null)
{
    animalNickname = readResult.ToLower();

}
```

5. To assign a value of `"tbd"` to `animalNickname` when the value entered is `""`, enter the following code:

```c#
if (animalNickname == "")
{
    animalNickname = "tbd";
}
```

6. Compare your completed nickname data entry loop with the following code:

```c#
// get the pet's nickname. animalNickname can be blank.
do
{
    Console.WriteLine("Enter a nickname for the pet");
    readResult = Console.ReadLine();
    if (readResult != null)
    {
        animalNickname = readResult.ToLower();
        if (animalNickname == "")
        {
            animalNickname = "tbd";
        }
    }
} while (animalNickname == "");
```

7. On the Visual Studio Code **File** menu, select **Save**.
8. Open the Integrated Terminal panel in Visual Studio Code and enter the command to Build your program.

   If any Build errors or warnings were reported, fix the issues before continuing.

Save the new pet information

In this task, you save the values entered for the pet characteristics to the `ourAnimals` array.

1. Add a blank code line below the code block of your nickname data entry loop.
2. To store the data values specified by the user, enter the following code:

```c#
// store the pet information in the ourAnimals array (zero based)
ourAnimals[petCount, 0] = "ID #: " + animalID;
ourAnimals[petCount, 1] = "Species: " + animalSpecies;
ourAnimals[petCount, 2] = "Age: " + animalAge;
ourAnimals[petCount, 3] = "Nickname: " + animalNickname;
ourAnimals[petCount, 4] = "Physical description: " + animalPhysicalDescription;
ourAnimals[petCount, 5] = "Personality: " + animalPersonalityDescription;
```

3. On the Visual Studio Code **File** menu, select **Save**.
4. Open the Integrated Terminal panel in Visual Studio Code and enter the command to Build your program.

   If any Build errors or warnings were reported, fix the issues before continuing.

Check your work

In this task, you run your application from the Integrated Terminal and verify that pet data entry is working correctly.

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, enter **2**

4. Verify that the Terminal panel has updated to show the following output:

Output

```
We currently have 4 pets that need homes. We can manage 4 more.

Enter 'dog' or 'cat' to begin a new entry
```

5. Enter the following values at the Terminal command prompts and verify that each subsequent prompt is displayed:

   - At the `Enter 'dog' or 'cat' to begin a new entry` prompt, enter **dog**
   - At the `Enter the pet's age or enter ? if unknown` prompt, enter **?**
   - At the `Enter a physical description of the pet (size, color, gender, weight, housebroken)` prompt, press the Enter key.
   - At the `Enter a description of the pet's personality (likes or dislikes, tricks, energy level)` prompt, press the Enter key.
   - At the `Enter a nickname for the pet` prompt, press the Enter key.

   The Terminal panel should be updated as follows:

Output

```
Enter 'dog' or 'cat' to begin a new entry
dog
Enter the pet's age or enter ? if unknown
?
Enter a physical description of the pet (size, color, gender, weight, housebro-
ken)

Enter a description of the pet's personality (likes or dislikes, tricks, energy
level)

Enter a nickname for the pet

Do you want to enter info for another pet (y/n)
```

6. At the Terminal command prompt, enter **n**

7. Verify that the Terminal panel has updated to show the main menu options.

8. At the Terminal command prompt, enter **1**

9. Verify that the Terminal panel has updated to show the following output:

Output

```
ID #: d1
Species: dog
Age: 2
Nickname: lola
Physical description: medium sized cream colored female golden retriever weigh-
ing about 65 pounds. housebroken.
Personality: loves to have her belly rubbed and likes to chase her tail. gives
lots of kisses.

ID #: d2
Species: dog
Age: 9
Nickname: loki
Physical description: large reddish-brown male golden retriever weighing about
85 pounds. housebroken.
Personality: 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.

ID #: c3
Species: cat
Age: 1
Nickname: Puss
Physical description: small white female weighing about 8 pounds. litter box
trained.
Personality: friendly

ID #: c4
Species: cat
Age: ?
Nickname:
Physical description:
Personality:

ID #: d5
Species: dog
Age: ?
Nickname: tbd
Physical description: tbd
Personality: tbd
Press the Enter key to continue
```

If your newly added pet information isn't displayed, check to be sure that you've included the code lines to save the data to the ourAnimals array, and check to be sure that you've included the code line to construct the petID.

10. Verify that you can create additional animal descriptions for dogs and cats and that animal characteristics are being saved to the `ourAnimals` array.

11. Exit the application, and then close the Terminal panel.

Congratulations on completing this Guided project! You've created an application that combines selection and iteration statements to achieve your application design goals. Your application includes over 300 lines and performs tasks that you might find in a professional application. Completing this project represents a significant accomplishment. Keep up the great work!

Next unit: Knowledge check

Continue >