Learn C#

App Interfaces

The team at Computron Computing has asked you to join their product team to develop the hottest new Computron computer. You’ll be responsible for building some of the standard apps on this new machine, specifically the to-do list and the password manager.

At this point in development you have two classes started: TodoList representing the to-do list application and PasswordManager representing the password manager. In order to work within the Computron system, every app must have a display and reset feature. In other words, each class will need to implement the IDisplayable and IResetable interfaces.

Classes can implement multiple interfaces using the colon and comma syntax:

class TodoList : IDisplayable, IResetable  
{}

Let’s get started! Make sure to save every file and test your code in the console with the command:

dotnet run

**Tasks**

13/13 Complete

Mark the tasks as complete by checking them off

**Learn the Classes**

1.

Get comfortable with the two classes TodoList and PasswordManager. You can see in **Program.cs** an instance of each class constructed.

* In **TodoList.cs**, read the definition of the constructor and Add() method
* In **PasswordManager.cs**, read the definition of the constructor

**IDisplayable**

2.

Every app must have a display feature. In **IDisplayable.cs**, define an empty IDisplayable interface.

The skeleton of an interface looks like this:

namespace SavingInterface  
{  
  interface IFakeFace  
  {  
  }  
}

3.

Within IDisplayable declare one method that:

* is named Display()
* returns nothing

We only need to define the signature, not body, of the method. Here’s an example:

interface IFakeFace  
{  
  int FakeMethod();  
}

4.

In **TodoList.cs**, declare that the class implements the IDisplayable interface.

Use the colon syntax at the top of the file.

5.

The class must actually implement the interface by defining a Display() method.

Define a method that:

* satisfies the interface requirements
* uses Console.WriteLine() to write each to-do item to the console

By “satisfy the interface requirements” we mean that TodoList must have a Display() method with the same signature as Display() in IDisplayable:

void Display();

Use a foreach to loop through the Todos array.

6.

Repeat the process for PasswordManager.

Declare that it implements IDisplayable.

Use the colon syntax at the top of the file.

7.

Define a Display() method that:

* satisfies the interface requirements
* uses Console.WriteLine() to print the actual password if Hidden is false. If Hidden is true, print asterisks \*

By “satisfy the interface requirements” we mean that TodoList must have a Display() method with the same signature as Display() in IDisplayable:

void Display();

In the method body, use an if-else statement that depends on the value of Hidden.

8.

Make sure that both classes are printable.

In **Program.cs**, call Display() on tdl and pm. Run the program with:

dotnet run

You should see both objects printed to the console.

**IResetable**

9.

Every app must have a reset feature. In **IResetable.cs**, define an IResetable interface with one method:

* name it Reset()
* it should return nothing

We only need to define the signature, not body, of the method. Here’s an example:

interface IFakeFace  
{  
  int FakeMethod();  
}

10.

In **TodoList.cs**, implement the interface:

* Declare that it implements IResetable
* Define a Reset() method to satisfy the interface. It should set the Todos property to an empty array of length 5 and set nextOpenIndex to 0.

A class can implement multiple interfaces, for example:

class Dog: IBarkable, IWalkable  
{}

Recall the syntax for declaring a new, empty array. In this case, numbers is an array of ints, with length 10.

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

11.

In **PasswordManager**, implement the interface:

* Declare that it implements IResetable
* Define a Reset() method to satisfy the interface. It should set Password to an empty string and set Hidden to false.

12.

In **Program.cs**, call Reset() and Display() with tdl and pm. In other words, both objects should display, then reset, then display again.

Run the program with:

dotnet run

Your code should contain these lines:

tdl.Display();  
tdl.Reset();  
tdl.Display();

And these lines:

pm.Display();  
pm.Reset();  
pm.Display();

**Extensions**

13.

Well done! The Computron is a better computer thanks to your work. If you’d like extra practice, try these optional challenges:

Add a get-only property to IDisplayable called HeaderSymbol. This should be used in Display() as a header. For example, if the header symbol is -, then the apps should be displayed as:

Todos  
--------  
Eat  
Sleep  
Code  
   
Password  
-----------  
\*\*\*

If you add more than five to-dos to the TodoList, it throws an error! Extend the Add() method so that it doesn’t add any more than five items to the Todos array.

Currently a blank line is printed for each empty index in Todos. Change Display() in TodoList so that it prints [] instead of a blank line. You’ll need to use [the static method String.IsNullOrEmpty()](https://docs.microsoft.com/en-us/dotnet/api/system.string.isnullorempty).

Add a method ChangePassword() to PasswordManager. It should:

* have two string parameters
* if the first argument matches the existing Password, reset the Password to the second argument
* return true if the password change was a success (the first argument matched the old password), and false otherwise

Extend the Password property in PasswordManager:

* Change the set method so that it requires the password to be at least eight characters in length

Adding HeaderSymbol to IDisplayable: Here’s what a get-only Sound property looks like in an interface:

interface ISoundable  
{  
  string Sound { get; }  
}

Extending TodoList.Add(): To extend the Add() method, use an if-else statement that checks if the Todos.Length is less than 5.

Extending TodoList.Display(): Within the foreach loop, check is each string is null or empty by using String.IsNullOrEmpty(). If this returns true, print []. Otherwise print the string.

Adding PasswordManager.ChangePassword(): Here’s the signature for ChangePassword():

public bool ChangePassword(string oldPass, string newPass)

Extending the setter for Password: remember that you can add a backing field for a property and define your own set method. Here’s an example of a legs field and Legs property that must be at least 6.

class CreepyCrawler  
{  
  int legs;  
   
  public int Legs  
  {  
    get  
    {   
      return legs;  
    }  
    set  
    {  
      if (value >= 6) { legs = value; }  
      else { legs = 6; }  
    }  
  }  
}