Contents

Warm-up Exercises

1. Consider the following partial class definition:

```
public class Book
{
    private string title;
    private string author;
    private string publisher;
    private int copiesSold;
}
```

1. Write a statement that would create a `Book` object.
2. Write a "getter" and a "setter" for the `title` attribute.
3. Write a constructor for the `Book` class taking at least one argument.

Solution for Part 1

```
Book myBook = new Book();
```

Solution for Part 2

```
public string GetTitle()
{
    return title;
}

public void SetTitle(titleP)
{
    title = titleP;
}
```

Solution for Part 3

```
public Book(string titleP, string authorP)
{
    title = titleP;
    author = authorP;
}
```

Questions

1. How do you make reference to a public property Name outside of the class.

☐ *Name
☐ +Name
☒ .Name
☐ neither of these

1. In C#, you should think of the class's properties as the class's attributes.

☒ Yes
☐ No

1. The property notation allows the client to directly manipulate the private instance variable.

☐ Yes
☒ No

1. Consider the code:

```
public void SetName(string tempAccountName)
{
name = tempAccountName; // store the account name
}
```

Which of the following statements is false? - ( ) The first line of each method declaration is the method header. - ( ) The method's return type specifies the type of data the method returns to its caller after performing its task. - ( ) The return type void indicates that when SetName() completes its task, it does not return any information to its calling method. - (x) All methods require at least one parameter to provide data to perform tasks.

1. A return type of _____ is specified for a method that does not return a value.

☐ int
☐ double
☒ void
☐ None of the above.

1. Methods are called by writing the name of the method followed by _____ enclosed in parentheses.

☐ a condition
☒ argument(s)
☐ a counter
☐ None of the above.

1. The parameter list in the method header and the arguments in the method call must agree in:

☐ Number
☐ Type
☐ Order
☒ All of the above

1. Suppose method1 is declared as

```
public void method1(int a, float b, string c)
```

Which of the following methods does not overload method1? - (x) void method2(int a, float b, char c) - ( ) int method1(float a, int b, string c) - ( ) float method1(int a, float b) - ( ) string method1(string a, float b, int c)

1. Write a get method for an instance variable named `total` of type **int**.

Solution

```
public int GetTotal()
{
    return total;
}
```

1. Write a getter for an attribute of type **string** named `myName`.

Solution

```
public string GetMyName()
{
    return myName;
}
```

1. Write a setter for an attribute of type **int** named `myAge`.

Solution

```
public void SetMyAge(int age)
{
    myAge = age;
}
```

1. Assuming name is a **string** instance variable, there is a problem with the following setter. Fix it.

```
public int SetName1(string var){
    name = var;
}
```

Solution

```
public int SetName1(string nameVar)
{
    name = nameVar;
}
```

1. Is it possible to have more than one constructor defined for a class? If yes, how can C# know which one is called?

Solution

Yes, C# can identify which constructor is called based on that constructor's method signature, that is, the combination of parameters associated with it.

1. What is the name of a constructor method? What is the return type of a constructor?

Solution

The name of a constructor method is the name of the class that contains it, and a constructor's return type *is* the class that contains it.

1. Write a constructor for a `Soda` class with one **string** attribute called `name`.

Solution

```
public Soda(string nameP)
{
    name = nameP;
}
```

1. What is the "default" constructor? Do we always have the possibility of using it?

Solution

The default constructor is one without any parameters. The only case in which it may not be called is if it has not been explicitly defined while other constructors have been defined.

1. Why would one want to define a constructor for a class?

Solution

By defining a constructor for a class, one can specify which values to assign to the instance variables upon instantiation.

Problems