Contents

Objects and References (Solutions)

Questions

1. What is the difference between **ref** and **out**?

- ☐ **ref** variables are "read-only", their value cannot change inside a method.
- ☐ **ref** is a keyword, **out** is not.
- ☐ There isn't any: they are both used to pass a reference to a method.
- ☒ **out** variables may not be initialized going into the method, but have to receive a value inside the method.
- ☐ There isn't any: they are both used to pass a value to a method.

Warm-up Exercises

1. Consider the following code:

```
using System;

class Program
{
  static void Main()
  {
    int x = 1;
    int y = 2;
    int z;
    char c = Foo(x, ref y, out z);
    char d = Foo(x, ref y, out z, '%');
  }

  static char Foo(
    int x,
    ref int y,
    out int z,
    char symb = '*'
  )
  {
    x++;
    y--;
    z = x + y;
    return symb;
  }
}
```

(a) What are the values of x, y and z

   i. Before the Foo method is called?
   ii. Inside the Foo method?
   iii. After the Foo method executed?

(b) What is the value of c?

(c) What is the value of d?

Solution

Before the Foo method is executed: 1, 2, and z is not set.

Inside the Foo method: 2, 1 and 3.

After the Foo method: 1, 0, and 2

c holds `'*'`, d holds `%`.

Problems

1. Write the AddRev method (header included) such that the following:

   ```
   int x0 = 4,
     y0 = 3;
   AddRev(ref x0, ref y0);
   Console.WriteLine($"x0 is {x0}, y0 is {y0}.");
   ```

   would display

   ```
   x0 is 7, y0 is 1.
   ```

   Solution

   ```
   void AddRev(ref int xP0, ref int yP0)
   {
     int temp = xP0;
     xP0 = xP0 + yP0;
     yP0 = temp - yP0;
   }
   ```

2. Write the AddLog method (header included) such that the following:

```
    string log;
    int x1 = 4,
      y1 = 3;
    int result = AddLog(x1, y1, out log);
    Console.WriteLine(log + "\n" + result);
```

would display

```
4 + 3 = 7.
7
```

Solution

```
    int AddLog(int xP1, int yP1, out string logP)
    {
      logP = xP1 + " + " + yP1 + " = " + (xP1 + yP1) +
 ↪  ".";
      return xP1 + yP1;
    }
```

3. Write the AddReset method (header included) such that the following:

```
    int x2 = 2,
      y2 = 3,
      z2;
    AddReset(ref x2, ref y2, out z2);
    Console.WriteLine($"x2 = {x2}, y2 = {y2}, z2 =
 ↪  {z2}.");
```

would display

```
x2 = 0, y2 = 0, z2 = 5.
```

Solution

```
    void AddReset(ref int xP2, ref int yP2, out int
     ↪  zP2)
    {
      zP2 = xP2 + yP2;
      xP2 = 0;
      yP2 = 0;
    }
```

4. Consider the "regular" implementation of the Rectangle class:

```
using System;
class Rectangle
{
    private int length;
    public int Length
    {
        get { return length; }
        set { if (value < 0) { throw new
         ↪  ArgumentNullException(); } else length =
         ↪  value; }
    }

    private int width;
    public int Width
    {
        get { return width; }
        set { if (value < 0) { throw new
         ↪  ArgumentNullException(); } else width =
         ↪  value; }
    }

    public Rectangle(int wP, int lP)
    {
        Width = wP;
        Length = lP;
    }

    public override string ToString()
    {
        return $"Width: {Width}\nLength: {Length}";
    }
}
```

And try to answer the following questions.

Solution

A possible solution to those questions is available[1].

(a) Write a `Draw` method that takes one *optional* `char` parameter and draw a rectangle of the calling object's width and length using that character if provided, `*` otherwise. If your method is correctly implemented, then

```
Rectangle r0 = new Rectangle(3, 2);

r0.Draw();
r0.Draw('-');
```

should display

```
***
***
```

[1]https:/princomp.github.io/code/projects/RectangleReferences.zip

```
---
---
```

Solution

A possible solution is: `public void Draw(char symb = '*')  {       string drawin`

(b) Write a `Copy` method that does not take arguments, and return *a copy* of the calling object. If your method is correctly implemented, then

```
Rectangle original = new Rectangle(5, 10);
Rectangle copy = original.Copy();
Console.WriteLine("Original:\n" + original +
 ↪  "\nCopy:\n"+ copy + "\n");
copy.Length = 12;
Console.WriteLine("\nOriginal:\n" + original +
 ↪  "\nCopy:\n" + copy + "\n");
```

should display

```
Original:
Width: 5
Length: 10
Copy:
Width: 5
Length: 10


Original:
Width: 5
Length: 10
Copy:
Width: 5
Length: 12
```

If the length of the original object changed after `copy.Length = 12;` was executed, then your method makes a *shallow* copy instead of a "deep" copy.

Solution

A possible solution is: `public Rectangle Copy()  {       return new Rectangle(Wi`

(c) Write an `Equals` method that return **`true`** if the calling object and the argument are both non-null rectangles with the same length and width, **`false`** otherwise. If your method is correctly implemented, then

```
Rectangle r1 = new Rectangle(5, 10);
Rectangle r2 = new Rectangle(5, 10);
Rectangle r3 = null;
Rectangle r4 = r1;
Rectangle r5 = new Rectangle(10, 5);

Console.WriteLine(
    "r1 and r2 identical: " + r1?.Equals(r2)
    + "\nr1 and r3 identical: " + r1?.Equals(r3)
    + "\nr3 and r1 identical: " + r3?.Equals(r1)
    + "\nr3 and r3 identical: " + r3?.Equals(r3)
    + "\nr1 and r4 identical: " + r1?.Equals(r4)
    + "\nr1 and r5 identical: " + r1?.Equals(r5)
    );
```

should display

```
r1 and r2 identical: True
r1 and r3 identical: False
r3 and r1 identical:
r3 and r3 identical:
r1 and r4 identical: True
r1 and r5 identical: False
```

Solution

A possible solution is: `public bool Equals(Rectangle rP)  {      if (rP == null`