Dylan Storts Design Patterns Prof. Retterer

Singleton Pattern

Introduction: For this project we were simple instructed to do the Singleton pattern in a simple application that would prove we knew the pattern and didn't just copy a source.

Analysis: The singleton pattern seemed very straight forward at first as it only has a `getInstance` method and a private constructor for the most part. The way that I decided to portray my solution to demonstrate the pattern was through fishing. The singleton would control how many fish objects there would be, being one in this case, as if one was the daily limit for catching fish. I kept it straight forward with the form only having three buttons as you can see on the right in Illustration 1. The button that reads "Get Another Fish" generates a number in a certain range that represents the length of the fish you just caught and prints that number in the text box labeled so. The button "Keep Fish" does just that and makes the fish object that stores the length of whatever fish you just caught and is the instance that is controlled by the singleton. The kept fish length of the current instance of fish would always be reprinted in the livewell text box whenever the keep fish button was pressed even though it will not change as the singleton protects the current instance from being replaced or edited as that is done only by the private constructor. This all seemed straight forward and a good situation for a singleton but I kept running into an error I could not figure out. This error can be seen in Illustration 2 below…

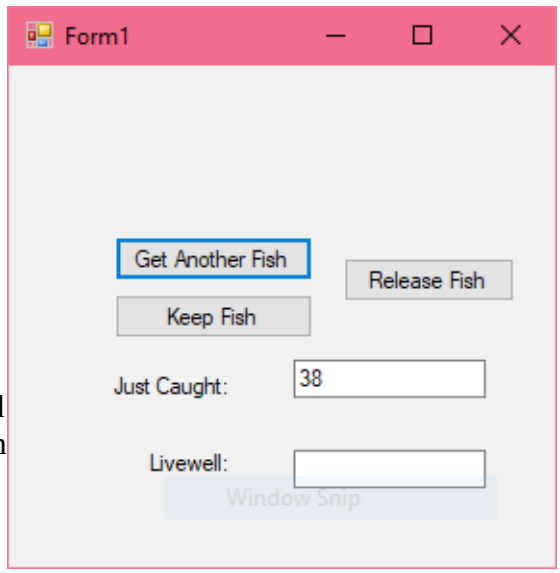

Illustration 1

```
private void Btn_keepfish_Click(object sender, EventArgs e)
{
    //here we actually call upon the singleton class to be demonstrated with this program
    LivewellFish myFish = LivewellFish.getInstance(tb_justcaught.Text);

    //now we can set the livewell box again, either populating it with the new fish length or retur
    tb_livewell.Text = myFish.getFish();

}

private void Btn_getfish_Click(object sen
{
    caughtFish = rnd.Next(smallestFish, l
    //now we have generated a new "fish"
    //this new "caught" fish will now be
    tb_justcaught.Text = caughtFish.ToStr
```

Exception Unhandled

System.NullReferenceException: 'Object reference not set to an instance of an object.'

myFish was null.

View Details | Copy Details

Exception Settings

I understand that the error is caused because apparently the object `myFish` is still null, or at least it is set but the reference isn't set to the object, but I could not figure out how to follow the singleton pattern and get the proper reference to the object. I have provided the code that should have, in theory, actually filled the `null` object and returned the proper reference in Drawing 1 below. In the `getInstance` method the parameter `newFish` is the string version of the next caught fish that the user wants to store. Then `fishInstance`, the single instance of the kept fish, is checked if it is still `null` and populates the instance with the new fish length passing it into the constructor to create the proper instance. After all that, or if the instance already existed, the instance is passed back to the form to be printed. I just could not find my error that kept the reference in the form to null.

```
      private LivewellFish(String fish)
        {//private constructor
            Livewell = fish;
        }

      public static LivewellFish getInstance(String newFish)
        {
            //here is the essence of the singleton where a single instance of it is
only allowed to be in memory at once
            if (fishInstance == null)
            {
                //create a new singleton object with a different string in it if there
is no existing object previously
                LivewellFish yourFish = new LivewellFish(newFish);
            }
            //return the singleton object
            return fishInstance;
        }
```

Drawing 1:

Reflection: I am very confident that even thought my application does not work I do understand how the singleton pattern is implemented into code. It simply lets a single instance of an object be made in memory and if that same kind of object is ever asked to be made again it can not because of the private constructor and instead gets the existing instance referenced back.