What is inheritance and why is it important?

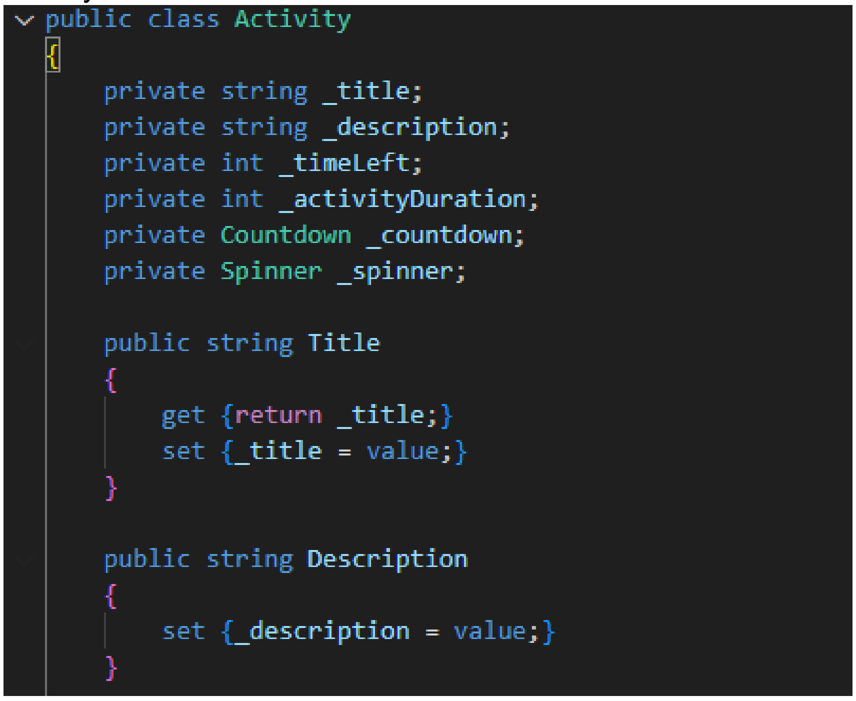
Inheritance is one of the main principles of object-oriented programming. Its use provides us the ability to create classes that take default behaviors and attributes from a “superclass” or “parent” class. Those elements remain linked to the parent class so that when they change in the parent class, they also change in the “child” classes that inherit from them.

The benefits of using inheritance include:

- Reduce code repetition: A superclass acts as a “category” of objects. When we need to create objects that have the same methods and attributes, we can avoid repeating those elements’ creation by having them in a parent class and letting the child classes inherit from it.
- Facilitate code maintenance: Having the main methods and attributes in one place promotes the DRY principle, which makes our code more maintainable because we only need to make changes or fixes in one place.
- Organization: Inheritance helps organize our classes in a hierarchical structure, and we can define the object’s relationships and functionality better using UML diagrams.

An application of inheritance could be e-commerce product categories. Where we can have a top-level class that has attributes like “productName”, “productdescription”, “price”, and “owner”, and methods like “updateStock”, “changePrice”, and “addDescription”. Then we can have subclasses that represent a product category, e.g. for the “Shoes” caterogy, we will inherit the top-level attributes and methods, and we can add additional attributes like “size”, “brand”, “color”, “discipline”, etc. We will have the ability to access the top-level methods and attributes from the “Shoes” caterogy.

Code example:
Activity superclass that we are using for a Mindfullness program. It represents an activity the user can do, such as a Breathing Activity, Reflection Activity, or Listing Activity.

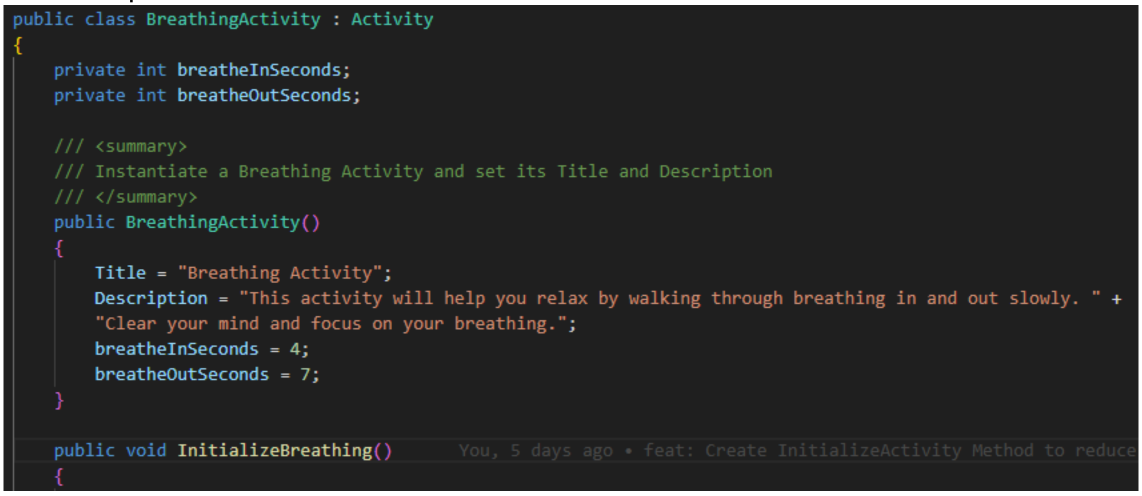


```
public class Activity
{
    private string _title;
    private string _description;
    private int _timeLeft;
    private int _activityDuration;
    private Countdown _countdown;
    private Spinner _spinner;

    public string Title
    {
        get {return _title;}
        set {_title = value;}
    }

    public string Description
    {
        set {_description = value;}
    }
```

For the different types of Activities we need to create 3 extra classes that will contain the specific methods and attributes that the different Activities need. If we don't use inheritance, we would have to repeat the main Activity attributes (_title, _description, etc) and methods (Title, Description, etc) in each of those classes.

By using inheritance, we say to the child classes that they need to take the main Activity methods and attributes so that we don't have to repeat their declaration.

For example:



```
public class BreathingActivity : Activity
{
    private int breatheInSeconds;
    private int breatheOutSeconds;

    /// <summary>
    /// Instantiate a Breathing Activity and set its Title and Description
    /// </summary>
    public BreathingActivity()
    {
        Title = "Breathing Activity";
        Description = "This activity will help you relax by walking through breathing in and out slowly. " +
        "Clear your mind and focus on your breathing.";
        breatheInSeconds = 4;
        breatheOutSeconds = 7;
    }

    public void InitializeBreathing()      You, 5 days ago • feat: Create InitializeActivity Method to reduce
    {
```

The syntax ***"public class NewClass : Superclass"*** tells the program that our new class (in the code example, the BreathingActivity class) will inherit from the superclass (Activity) its methods and attributes, and then we define other elements that will pertain only to the subclass.