Design Patterns: **Private Class Data**

Structural Pattern

Protecting class state by minimizing the visibility of its attributes (data)

---

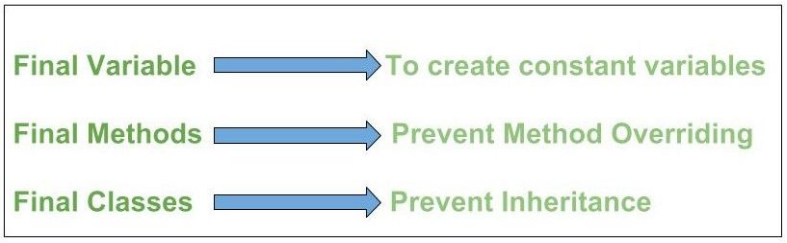
1. Design Pattern Description

Intent

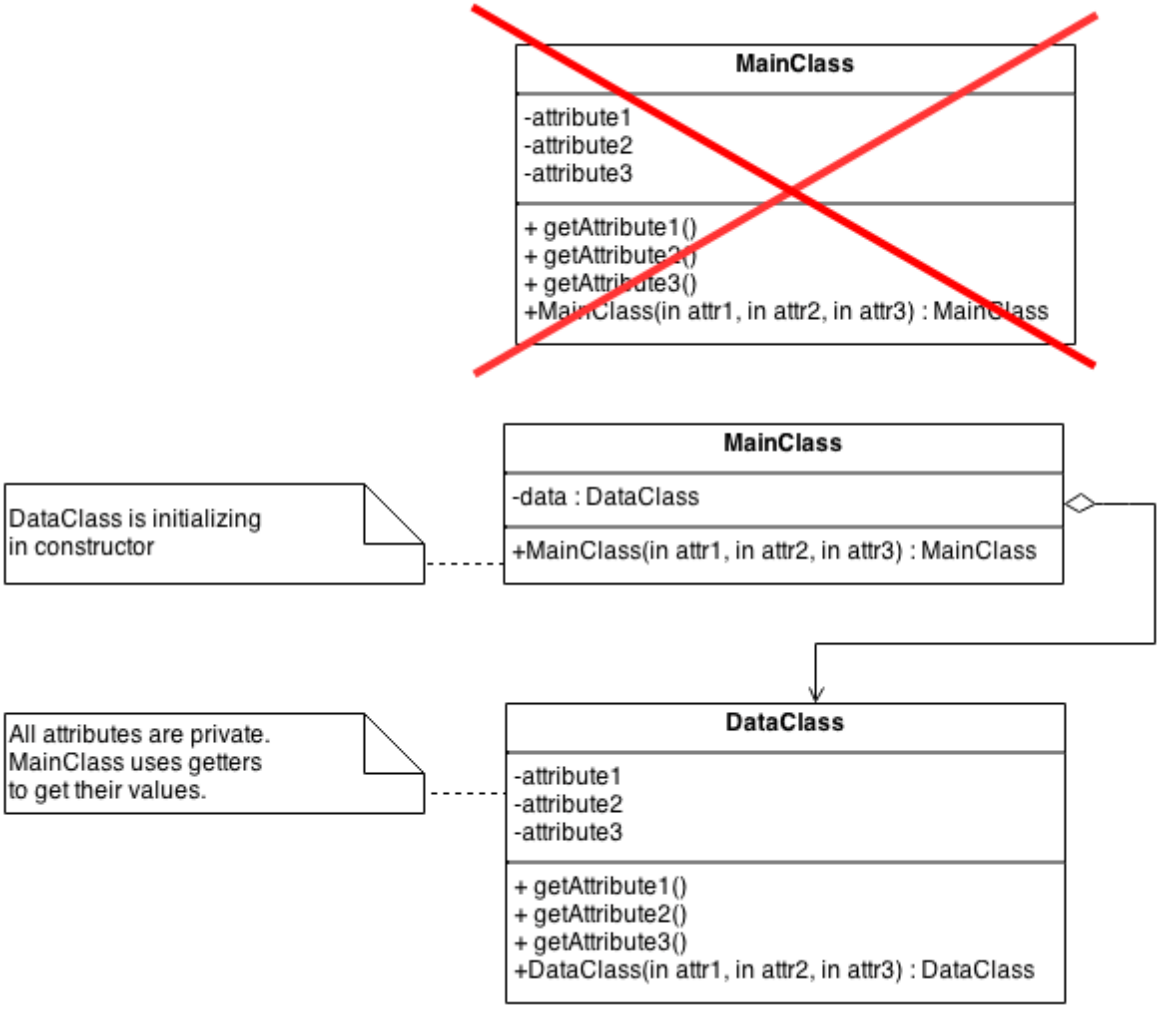
- **Control write access** to class attributes
- **Separate data from methods** that use it
- Encapsulate class data initialization
- Providing **new type of final** - final after constructor (*)

“Final” explanation

(*) `final` is keyword used in some languages meaning (ref. Java):



Structure


MainClass
-attribute1
-attribute2
-attribute3
+ getAttribute1()
+ getAttribute2()
+ getAttribute3()
+MainClass(in attr1, in attr2, in attr3) : MainClass
MainClass
-data : DataClass
+MainClass(in attr1, in attr2, in attr3) : MainClass
DataClass is initializing
in constructor
DataClass
-attribute1
-attribute2
-attribute3
+ getAttribute1()
+ getAttribute2()
+ getAttribute3()
+DataClass(in attr1, in attr2, in attr3) : DataClass
All attributes are private.
MainClass uses getters
to get their values.

2. Design Pattern Example

Before

Attributes radius, color and origin:

- **should not change after constructor**.
- private access limits visibility but **they can still be modified by internal methods**.
- since they have to be set once, **const or final/ReadOnly attributes cannot be set**.

```
public class Circle {
   private double radius;
   private Color color;
   private Point origin;
   public Circle(double radius, Color color, Point origin) {
     this.radius = radius;
     this.color = color;
     this.origin = origin;
   }
   public double Circumference {
     get { return 2 * Math.PI * this.radius; }
   }
   public double Diameter {
     get { return 2 * this.radius; }
   }
   public void Draw(Graphics graphics) {
     //...
   }
 }
```

After

```
public class Circle {
   private double radius;
   private Color color;
   private Point origin;
   public Circle(double radius, Color color, Point origin) {
     this.radius = radius;
     this.color = color;
     this.origin = origin;
   }
   public double Circumference {
     get { return 2 * Math.PI * this.radius; }
   }
```

```
  public double Diameter {
    get { return 2 * this.radius; }
  }
  public void Draw(Graphics graphics) {
    //...
  }
}
```

3. Existing example in SwanLab?

Closest example: **Mesh_Uniftted_Properties**
But it was intended to remove a large list of properties from a “fat” class.

Interesting concept considered in Swan (because of size, users, purpose, etc.):

PROTECT PROPERTIES ALONG THE INTERNAL WORKFLOW

I think the relevance of “final” properties grows with:

- **Number of developers** working in a common area of code.
- **Size of the class.**

4. Design Proposal for SwanLab?

Don’t if it’s a good practice in Swan, at least not a good common one for regular or small classes since it is desirable to have their properties explicitly defined in the class file.

Maybe it could be useful to protect the “mesh”, “backgroundMesh” properties along the workflow of the different UnfittedMeshes, since they can be easily and unconsciously mutated and might be interesting to prevent it.