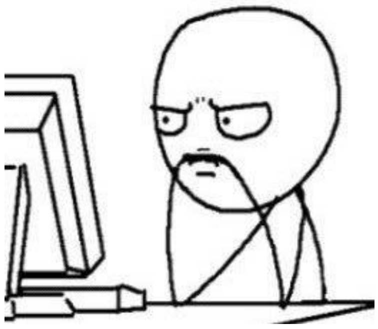
Wondering why do we need the body for the methods that have been overridden in subclasses..?? Well let's see how to get rid of it without affecting inheritance or polymorphism.



Abstraction in Java

As per dictionary, **abstraction** is the quality of dealing with ideas rather than events. For example, when you consider the case of e-mail, complex details such as what happens as soon as you send an e-mail, the protocol your e-mail server uses are **hidden from the user**. Therefore, to send an e-mail you just need to type the content, mention the address of the receiver, and click send.

Likewise in Object-oriented programming, **abstraction is a process of hiding the implementation details from the user**, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it.

In simpler words, data **abstraction** is the process of **hiding certain details and showing only essential information** to the user.

Abstraction can be achieved with either **abstract classes** or **interfaces** (which you will learn more about in the next couple of classes).

The **abstract** keyword is a **non-access modifier**, used for classes and methods:

- **Abstract class:** is a restricted class that **cannot be used to create objects** (to access it, it must be inherited from another class).
- **Abstract method:** can only be used in an abstract class, and **it does not have a body**. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods.

Example

```
abstract class Plane
{
    abstract void takeOff();
    abstract void fly();
    abstract void land();
}
class CargoPlane extends Plane
{
    void takeOff()
    {
        System.out.println
            ("CargoPlane is taking off from a long sized runway.");
    }
    void fly()
    {
        System.out.println
            ("CargoPlane is flying at low heights.");
    }
    void land()
    {
        System.out.println
            ("CargoPlane is landing on long sized runway.");
    }
}

class PassengerPlane extends Plane
{
    void takeOff()
    {
        System.out.println
            ("PassengerPlane is taking off from a medium sized runway.");
    }
    void fly()
    {
        System.out.println
            ("PassengerPlane is flying at medium heights.");
    }
    void land()
    {
        System.out.println
            ("PassengerPlane is landing on medium sized runway.");
    }
}
class FighterPlane extends Plane
{
    void takeOff()
    {
        System.out.println
            ("FighterPlane is taking off from a short sized runway.");
    }
```

```java
    void fly()
    {
        System.out.println
            ("FighterPlane is flying at great heights.");
    }
    void land()
    {
        System.out.println
            ("FighterPlane is landing on short sized runway.");
    }
}

class Airport
{
    void permit(Plane ref)
    {
        ref.takeOff();
        ref.fly();
        ref.land();
    }
}
class Demo
{
    public static void main(String[] args)
    {
        CargoPlane cp = new CargoPlane();
        PassengerPlane pp = new PassengerPlane();
        FighterPlane fp = new FighterPlane();

        Airport a = new Airport();
        a.permit(cp);
        a.permit(pp);
        a.permit(fp);
    }
}
```

Output

```
CargoPlane is taking off from a long sized runway.
CargoPlane is flying at low heights.
CargoPlane is landing on long sized runway.
PassengerPlane is taking off from a medium sized runway.
PassengerPlane is flying at medium heights.
PassengerPlane is landing on medium sized runway.
FighterPlane is taking off from a short sized runway.
FighterPlane is flying at great heights.
FighterPlane is landing on short sized runway.
```

Why And When To Use Abstract Classes and Methods?

To achieve security - hide certain details and only show the important details of an object.

Note: Abstraction can also be achieved with Interfaces, which you will learn more about in the next chapter.

Example

```
 import java.util.Scanner;
 abstract class Shape
 {
     float area;
     abstract void acceptInput();
     abstract void calcArea();
     void dispArea()
     {
         System.out.println(area);
     }
 }
 class Square extends Shape
 {
     float side;

     void acceptInput()
     {
         Scanner scan = new Scanner(System.in);
         System.out.println("Enter the length of the side:");
         side = scan.nextFloat();
     }
     void calcArea()
     {
         area = side*side;
     }
 }
class Rectangle extends Shape
{
     float length;
     float breadth;

     void acceptInput()
     {
         Scanner scan = new Scanner(System.in);
         System.out.println("Enter the length:");
         length = scan.nextFloat();
         System.out.println("Enter the breadth:");
         breadth = scan.nextFloat();
     }
     void calcArea()
     {
         area = length*breadth;
     }
}
```

```java
class Circle extends Shape
{
    float radius;

    void acceptInput()
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the radius:");
        radius = scan.nextFloat();
    }
    void calcArea()
    {
        area = 3.141f*radius*radius;
    }
}
class Geometry
{
    void permit(Shape ref)
    {
        ref.acceptInput();
        ref.calcArea();
        ref.dispArea();
    }
}

class Demo
{
    public static void main(String[] args)
    {
        Square s = new Square();
        Rectangle r = new Rectangle();
        Circle c = new Circle();

        Geometry g = new Geometry();
        g.permit(s);
        g.permit(r);
        g.permit(c);
    }
}
```

Output

```
Enter the length of the side:
5
25.0
Enter the length:
10
Enter the breadth:
20
200.0
Enter the radius:
5
78.525
```