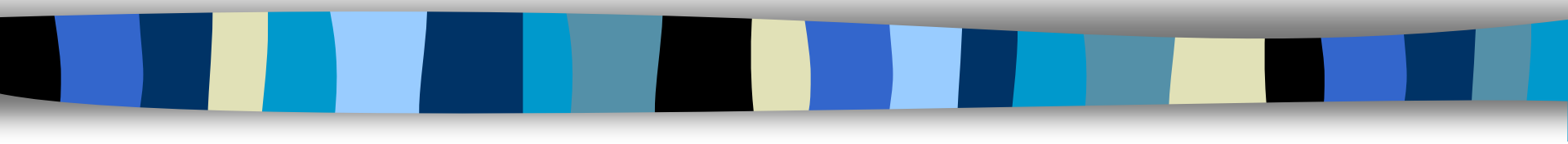
Object Orientation with Design Patterns



Lecture 4 : Polymorphism - Revision

Polymorphism

- When we **extend a superclass** we can **redefine one** or some of its **methods in the subclass**. It can then be said that the **new subclass's method *overides*** the method in the superclass.

- In the following example the Java Applet class has been extended by the class Overloader. The Java Applet class has a method called ***paint*** which it uses to do any output to the applet window.

- If we extend the Java Applet class and **do not** *overload/override* the paint method we should get the following output.

Polymorphism

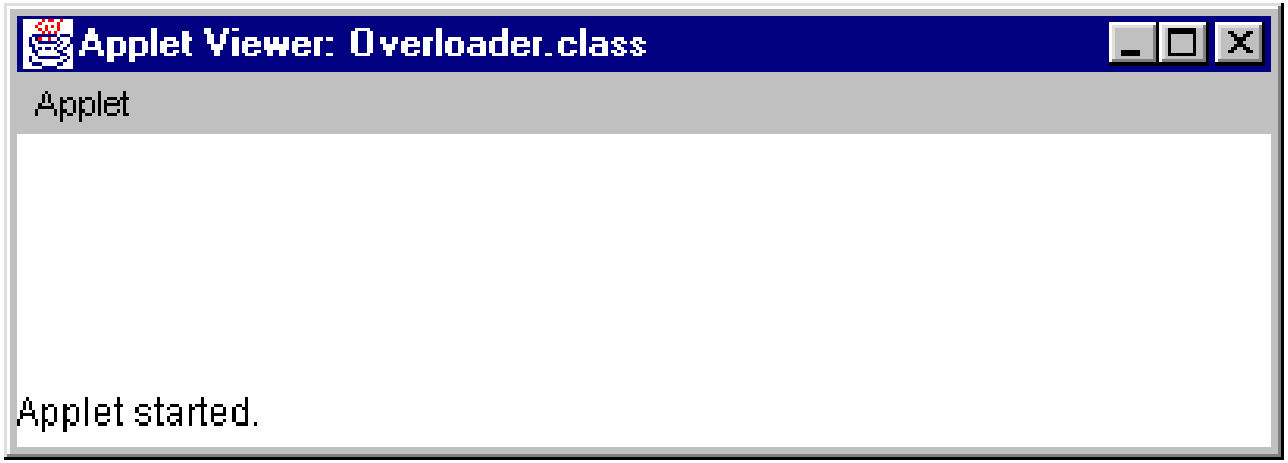
- Code below shows Overloader class.

```
import java.io.*;
import java.applet.*;


public class Overloader extends Applet
{

}
```

- This should just give us a blank Applet window.



Polymorphism

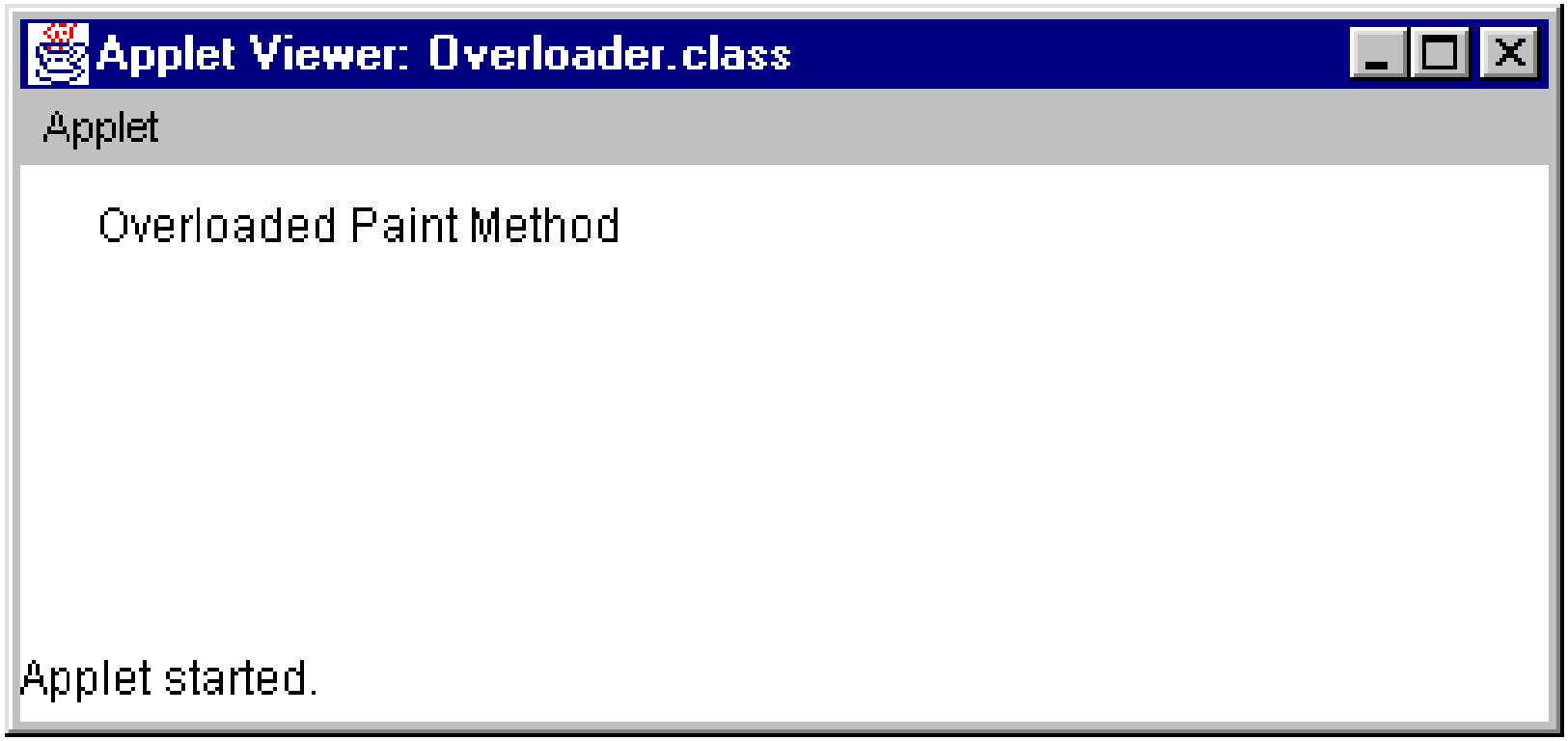
- We can however change this behaviour by overloading the Applet paint method in the subclass.

```
import java.io.*;
import java.awt.*;
import java.applet.*;


public class Overloader extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Overloaded Paint Method", 20,20);
    }
}
```

- The code above shows the new overloaded paint method in the Overloader class. The next slide shows the output when the applet is run.

Polymorphism



Polymorphism

- As you can see the overloaded paint method has changed the behaviour of the Applet program.

- In the first example the paint method was not overloaded so the default paint method in the Java Applet class we called. This default paint method in the superclass does nothing so there was no output. In effect we could think of the paint method in the superclass as having no code in its body.

```
public void paint(Graphics g)
{
    // Do nothing
}
```

- In the second example we overloaded the paint method and put some code inside it.

Polymorphism

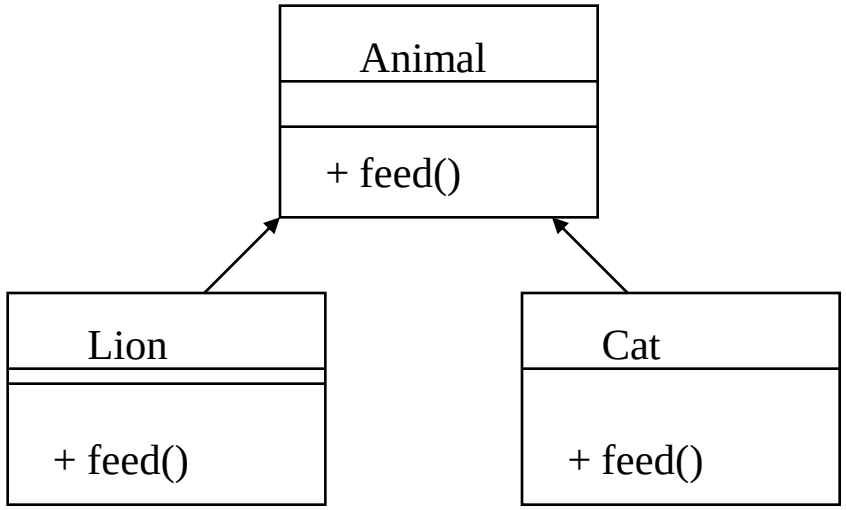
- This is really just a combination of inheritance and method overloading so what has it got to do with polymorphism?
- **Polymorphism is the ability to treat dissimilar but related objects in the same way.**
- The objects in question can be related through inheritance
- The objects in question may be dissimilar because they may want the same method to behave in different ways.

Polymorphism

- Lets take a non-software analogy
  - When a car driver decides to hit the brakes in order to avoid a pedestrian crossing an intersection, the driver is concerned about the concept of quickly slowing down or braking. The brakes will be either drum brakes or disk brakes, which although constructed quite differently both achieve the same effect of slowing a vehicle down by converting the vehicles kinetic energy to heat. The driver who is about to depress the brake pedal to avoid the pedestrian is not (and should not be) thinking about the drum versus disk implementation of the brake but only of the desired action. The system (the car) determines the how the braking action is to be implemented.

Polymorphism

- Now lets take a software analogy



Without Polymorphism

If animal = 'Lion' then
do the Lion feed
Else if animal = 'Cat' then
do the Cat feed
End

With Polymorphism

Do the Animal feed

Polymorphism

- Polymorphism – the ability to hide many different implementations behind a single interface
- **The ability of the system to decide at run time which implementation of feed to call for each animal**
- This ability is called Dynamic Binding
  – Bind the correct version of the operation to the object at runtime.