Polymorphism

Objectives

- Overloading and Overriding
- Interface
- Abstract class

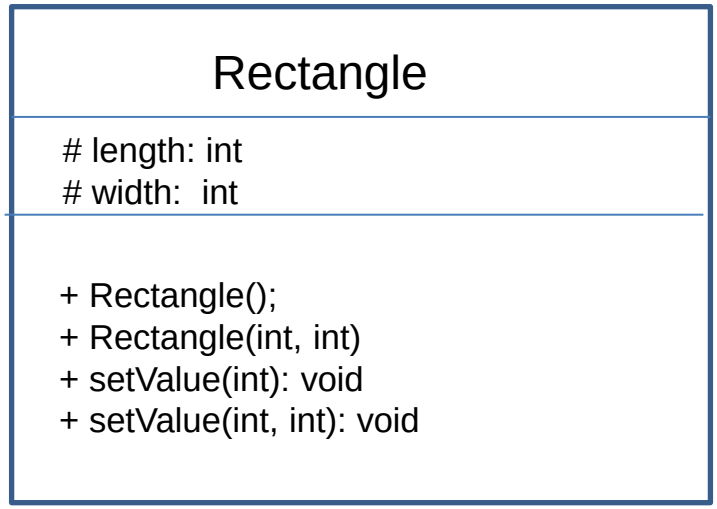
Polymorphism

Ability allows many versions of a method based on overloading and overriding methods techniques.

Overloading: A class can have some methods which have the same name but their parameter types are different.

Overriding: A method in the father class can be overridden in its derived classes (body of a method can be replaced in derived classes).

Overloading



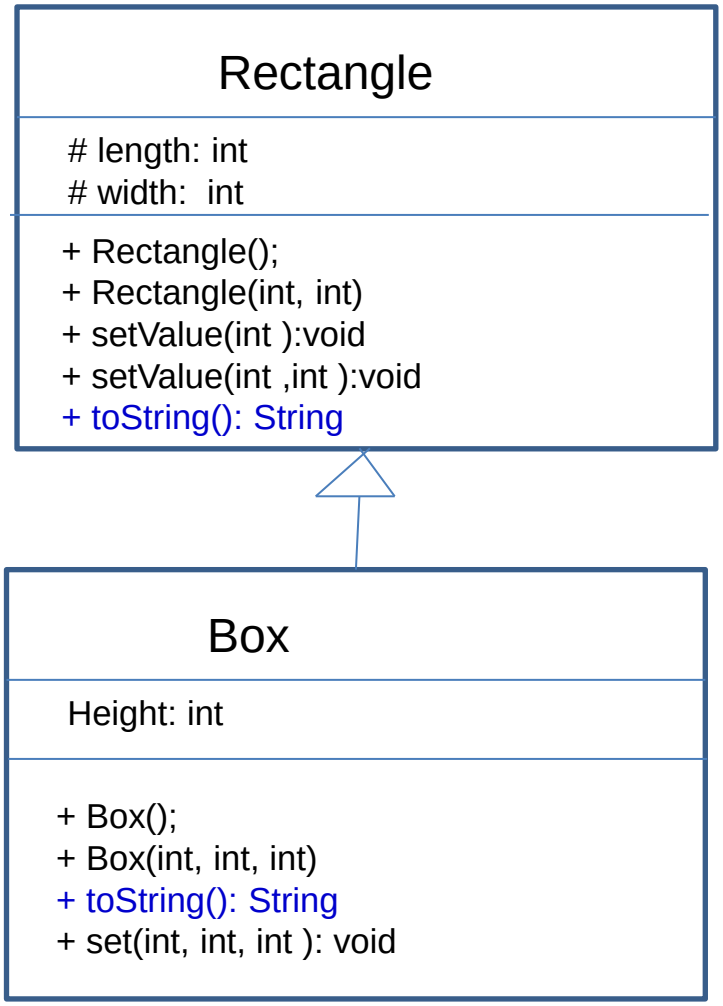
- overloading with constructors

```
public Rectangle(){…}
public Rectangle(int length, int  width){… }
```

- Overloading also extends to general methods.

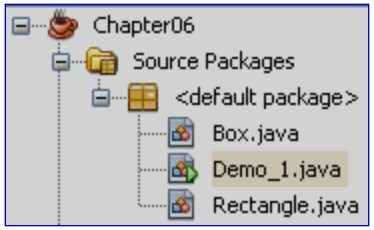
```
public void setValue(int len){
    length= (len>0)?1:0;
}
public void setValue (int  len, int wi){
      length= (len>0)? 1: 0;
      width= (wi>0)? wi:0;
}
```

Overriding


Rectangle
length: int
width: int
+ Rectangle();
+ Rectangle(int, int)
+ setValue(int):void
+ setValue(int ,int):void
+ toString(): String
Box
Height: int
+ Box();
+ Box(int, int, int)
+ toString(): String
+ set(int, int, int): void

Overriding Inherited Methods

Overridden method: An inherited method is re-written




```
1   public class Box extends Rectangle {
2     int height=0;
3     public void set (int l, int w, int h)
4     {  super.setValue(l, w);
5        height = h>0? h: 0;
6     }
7     // Overriding the toString method
8     // of the Rectangle class
      public String toString()
10    { return "[" + length + "," + width +
11                  "," + height + "]";
12    }
13  }
```

```
1   public class Rectangle {
2      protected int length=0, width=0;
3      // Overloading methods
4      public void setValue(int l)
5      {  length = l>0?l:0; }
6      public void setValue(int l, int w)
7      {  length = l>0? l: 0;
8         width= w>0? w: 0;
9      }
10     // Overriding the toString method of the java.lang.Object class
       public String toString()
12     {  return "[" + length + "," + width + "]}";
13     }
14  }
15
```

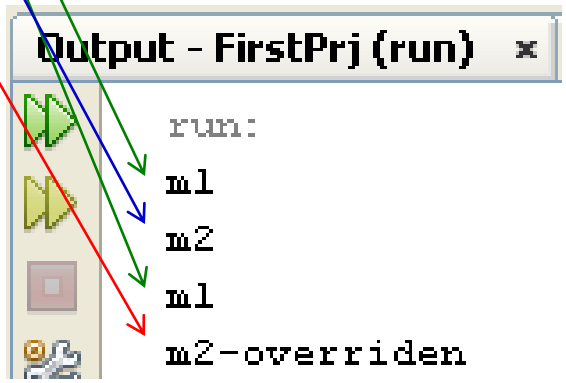
```
Output - Chapter06 (run)
run:
[5,0]}
[10,20]}
[5,10,15]
```

```
public class Demo_1 {
  public static void main  (String[] args)
  {  Rectangle r= new Rectangle();
     r.setValue(5);
     System.out.println(r.toString());
     r.setValue(10,20);
     System.out.println(r.toString());
     Box b= new Box();
     b.set(5,10,15);
     System.out.println(b.toString());
  }
}
```

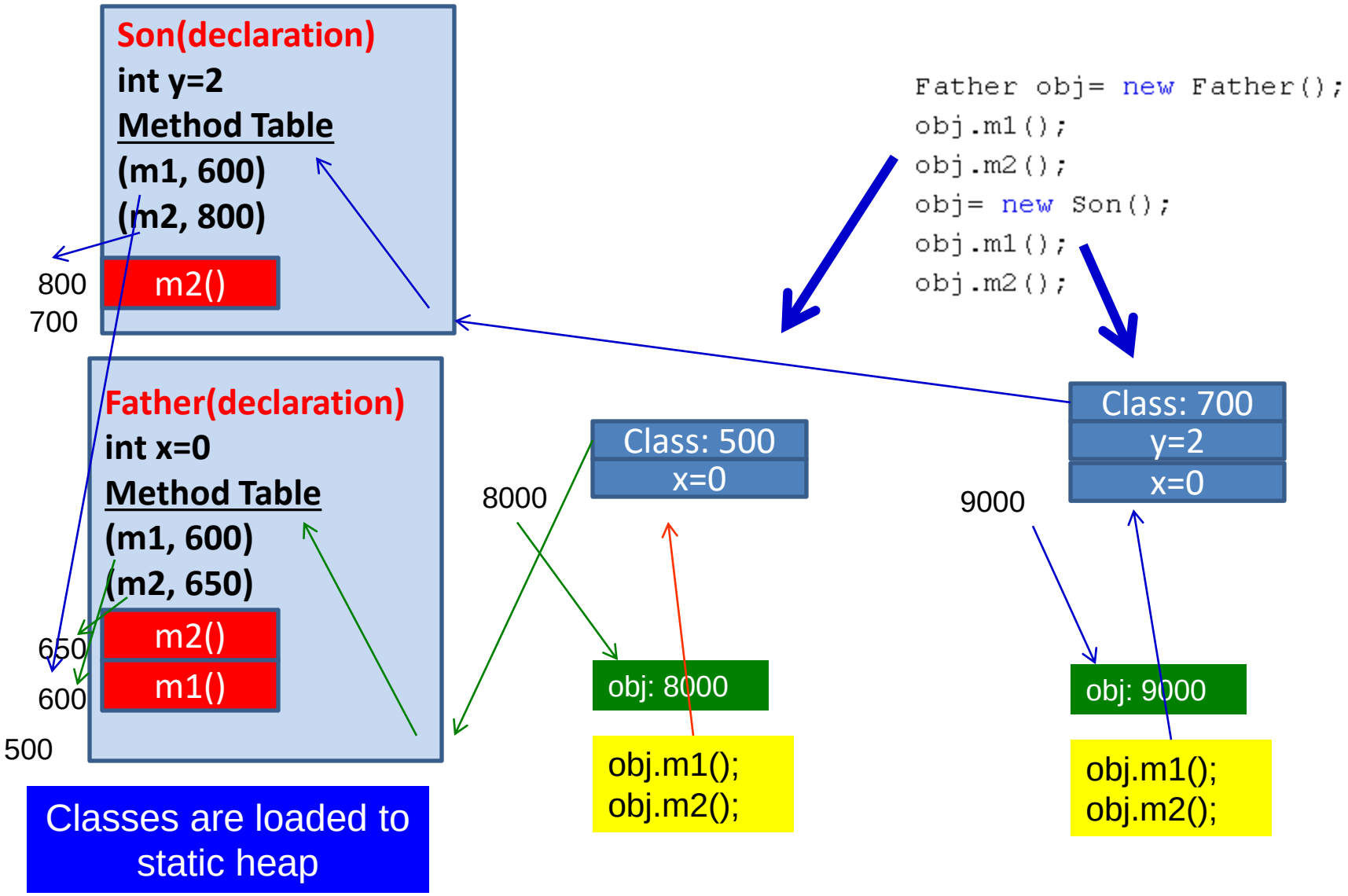
Overloaded methods: Methods have the same name but their parameters are different in a class

How Can Overridden Method be Determined?

```
class Father{
    int x=0;
    void m1() { System.out.println("m1");}
    void m2() { System.out.println("m2");}
}
class Son extends Father {
    int y=2;
    void m2() { System.out.println("m2-overriden");}
}
public class CallOverriddenMethod {
    public static void main(String[] args){
        Father obj= new Father();
        obj.m1();
        obj.m2();
        obj= new Son();
        obj.m1();
        obj.m2();
    }
}
```



How Can Overridden Methods be Determined?



Interfaces

- An *interface* is a reference type, similar to a class, that can contain *only* constants, initialized fields, static methods, prototypes (abstract methods, default methods), static methods, and nested types.
- It will be the **core** of some classes
- Interfaces cannot be instantiated because they have no-body methods.
- Interfaces can only be *implemented* by classes or *extended* by other interfaces.

WHY AND WHEN TO USE INTERFACES?

- To achieve security - hide certain details and only show the important details of an object (interface).
- Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can implement multiple interfaces

Interfaces…

```
1   public interface InterfaceDemo {
2       final int MAXN=100; // constant
3       int n=0; // Fields in interface must be initialized
4       static public int sqr(int x){ return x*x;}
5       public abstract void m1(); // abstract methods
6       abstract public void m2();
7       void m3(); // default methods
8       void m4();
9   }
10
11  class UseIt{
12      public static void main(String args[]){
            InterfaceDemo obj= new InterfaceDemo();
14      }
15  }
```

Interfaces…

```
public interface InterfaceDemo {
    final int MAXN=100; // constant
    int n=0; // Fields in interface must be initialized
    static public int sqr(int x){ return x*x;}
    public abstract void m1(); // abstract methods
    abstract public void m2();
    void m3(); // default methods
    void m4();
}
class A implements InterfaceDemo{
    // overriding methods
    public void m1() { System.out.println("M1");}
    public void m2() { System.out.println("M2");}
    void m3() { System.out.println("M3");}
    void m4() { System.out.println("M4");}
}
```

m3(), m4() in A cannot implement m3(), m4() in InterfaceDemo, attempting to assign weaker access privileges, were public

Default methods of an interface must be overridden as public methods in concrete classes.

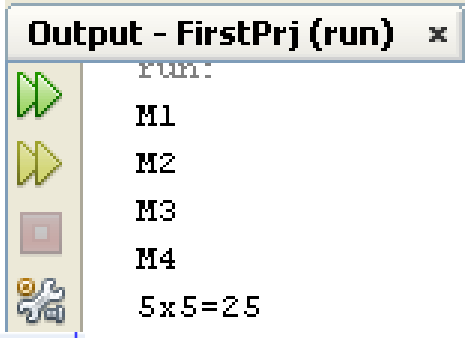
Interfaces

...

```
public interface InterfaceDemo {
    final int MAXN=100; // constant
    int n=0; // Fields in interface must be initialized
    static public int sqr(int x){ return x*x;}
    public abstract void m1(); // abstract methods
    abstract public void m2();
    void m3(); // default methods
    void m4();
}
```

```
class A implements InterfaceDemo{
    // overriding methods
    public void m1() { System.out.println("M1");}
    public void m2() { System.out.println("M2");}
    public void m3() { System.out.println("M3");}
    public void m4() { System.out.println("M4");}
}
```

```
class UseIt{
    public static void main(String args[]){
        InterfaceDemo obj= new A();
        obj.m1();
        obj.m2();
        obj.m3();
        obj.m4();
        int s= InterfaceDemo.sqr(5);
        System.out.println("5x5=" + s);
    }
}
```



Abstract Classes

- Used to define ***what*** **behaviors a class is required to perform without having to provide an explicit implementation.**
- **It is the result of so-high generalization**
- **Syntax to define a abstract class**
  - *public abstract class className{ ... }*
- It isn't necessary for all of the methods in an abstract class to be abstract.
- An abstract class can also declare implemented methods.

Abstract Classes…

```
1   package shapes;
2   public abstract class Shape {
3       abstract public double circumstance();
4       abstract public double area();
5   }
6   class Circle extends Shape {
7       double r;
8       public Circle (double rr) { r=rr; }
9       public double circumstance() { return 2*Math.PI*r; }
10      public double area() { return Math.PI*r*r; }
11  }
12  class Rect extends Shape {
13      double l,w;
14      public Rect(double ll, double ww){
15          l = ll; w = ww;
16      }
17      public double circumstance() { return 2*(l+w); }
18      public double area() { return l*w; }
19  }
20  class Program {
21      public static void main(String[] args) {
22          Shape s = new Shape ();
23      }
24  }
```

```
20  class Program {
21      public static void main(String[] args) {
22          Shape s = new Circle(5);
23          System.out.println(s.area());
24      }
25  }
```

Modified

```
Output - Chapter06 (run)
run:
78.53981633974483
```

Abstract Classes…

```
1   public abstract class AbstractDemo2 {
2       void m1() // It is not abstract class
3       { System.out.println("m1");
4       }
5       void m2() // It is not abstract class
6       { // empty body
7       }
8       public static void main(String[] args)
        {   AbstractDemo2 obj = new AbstractDemo2();
10      }
11  }
```

This class have no abstract method but it is declared as an abstract class. So, we can not initiate an object of this class.

Abstract Classes…

Error. Why?

```
   public abstract class AbstractDemo2 {
      void m1() // It is not abstract class
3     { System.out.println("m1");
4     }
5     abstract void m2();
6  }
   class Derived extends  AbstractDemo2
   {  public void m1() // override
9     { System.out.println("m1");
10    }
11    public static void main(String[] args)
12    {  Derived obj = new Derived();
13    }
14 }
```

Implementing Abstract Methods

- Derive a class from an abstract superclass, the subclass will inherit all of the superclass's features, all of ***abstract* methods** included**.**
- To replace an inherited abstract method with a concrete version, the subclass need merely override it.
- Abstract classes ***cannot be instantiated***

Anonymous Classes

Anonymous classes are classes which are not named but they are identified automatically by Java compiler.

Where are they? They are identified at initializations of interface/abstract class object but abstract methods are implemented as attachments**.**

Why are they used?

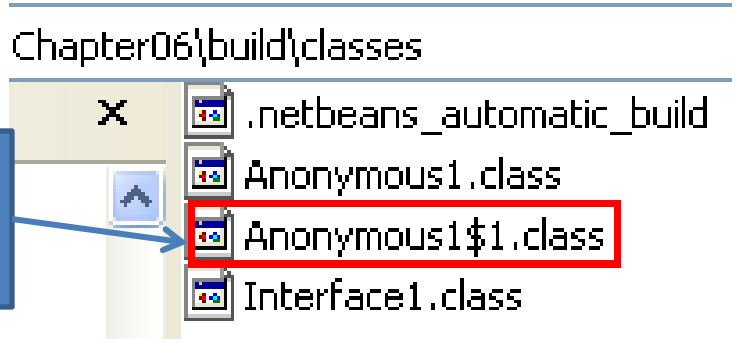
- Enable you to make your code more concise.
- Enable you to declare and instantiate a class at the same time.
- They are like local classes except that they do not have a name.
- Use them if you need to use a local class only once.

Anonymous Class…

```
1   // New - Java Interface
2   public interface Interface1 {
3     void M1();
4     void M2();
5   }
6   class Anonymous1{
7       public static void main(String[] args) {
8           Interface1 obj = new Interface1() {
9               public void M1()
10              { System.out.println("M1");}
11              public void M2()
12              { System.out.println("M2");}
13          };
14          obj.M1();
15          obj.M2();
16      }
17  }
18
```

Anonymous class.

Class name is given by the compiler: **ContainerClass$Number**




```
Output - Chapter06 (run)
run:
M1
M2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Anonymous Class…

```
1   package adapters;
2   // abstract class contains all concrete methods
3   public abstract class MyAdapter {
4       public void M1() { System.out.println("M1");}
5       public void M2() { System.out.println("M2");}
6   }
7   class Program {
8       public static void main(String[] args) {
9           // Overriding one method
10          MyAdapter obj = new MyAdapter ()
            {   public void M1()
12              {   System.out.println("M1 overridden");
13              }
14          };
15          obj.M2();
16          obj.M1();
17      }
18  }
```

Concrete methods but they can not be used because the class is declared as abstract one.

The abstract class can be used only when at least one of it's methods is overridden

```
Output - Chapter06 (run)
run:
M2
M1 overridden
```

Anonymous class is a technique is commonly used to support programmer when only some methods are overridden only especially in event programming.

Summary

- Polymorphism is a concept of object-oriented programming
- Polymorphism is the ability of an object to take on many forms
- Overloading and overriding are a technology to implement polymorphism feature.
- In OOP occurs when a parent class/ interface reference is used to refer to a child class object