

Object Oriented Programming (OOP)

Mohamed Ezz

Lecture 7

Lecture Objectives

- ✓ Understand Dynamic Binding (Polymorphism)
- ✓ Understand how to achieve SW extensible and maintainable using polymorphism .
- ✓ Practice different use of polymorphism
- ✓ Practice how to override Object Method
- ✓ Understand Final Class /Method & Variable

Review : Website (facebook, mail,...) Class-1

[illegible]

Review:Website (facebook, mail,...) Class-2



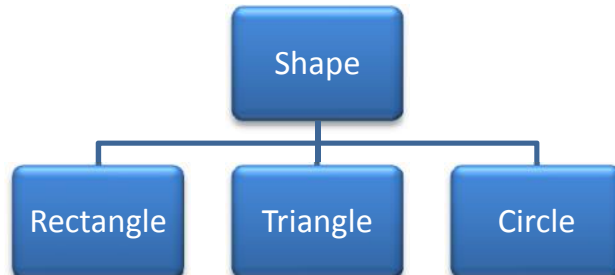
```
public boolean login (String email, String password){
```

```
}  
}
```

Review: Shape Inheritance Example

```
class Shape{  
    protected int color = 0;  
    public void setColor(int color){  
        this.color=color;  
    }  
    public int getColor(){  
        return color;  
    }  
    public float computeArea (){  
        return 0;  
    }  
}
```

```
class Circle extends Shape{  
    private int radius = 0;  
  
    public Circle (int r){  
        radius =r;  
    }  
  
    public float computeArea (){  
        return 22 /7* radius* radius;  
    }  
  
    public void doubleSize (){  
        radius= 2 * radius;  
    }  
}
```



Review: Shape Inheritance Example

```
class Triangle extends Shape{
    private int base = 0;
    private int height = 0;

    public Triangle (int h, int b){
        base=b;
        height=h;
    }
    public float computeArea (){
        return 0.5 * base * height;
    }
}
```

```
class Rectangle extends Shape{
    private int width = 0;
    private int height = 0;

    public Rectangle(int h, int w){
        width=w;
        height=h;
    }

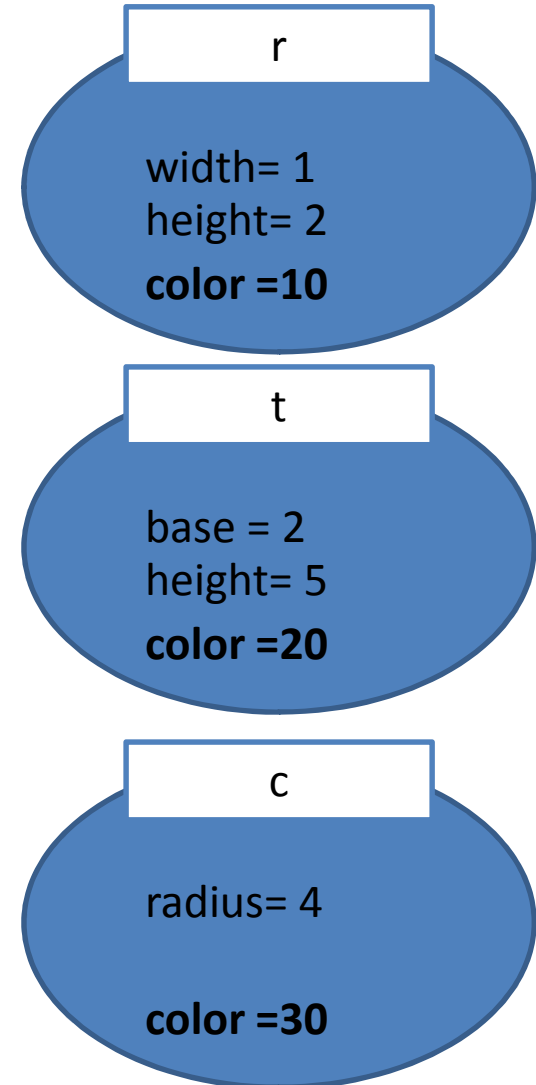
    public float computeArea (){
        return width* height;
    }

    public void swap(){
        int i= width;
        width = height;
        height = i;
    }
}
```

Review: Shape Inheritance Example

```
class TestShape {  
    public static void main(String arg[]){  
        Rectangle r= new Rectangle (1,2);  
        r.setColor(10);  
        Triangle t= new Triangle(2,5);  
        t.setColor(20);  
        Circle c= new Circle (4);  
        c.setColor(30);  
        float totalArea=0;  
        totalArea +=s.computeArea ();  
        totalArea +=t.computeArea ();  
        totalArea +=c.computeArea ();  
        System.out.println(totalArea);  
    }  
}
```

What is the problem if I have 100 objects?



Polymorphism

- Polymorphism
 - the ability for a variable (of a superclass type) to contain (**Refer to**) different objects of a subclass type at different points in time
 - results in different functionality being executed for the same method call
 - allows for run-time (dynamic) instead of compile-time (static) binding
 - Subclass object has “multiple identities”, based on its class inheritance tree


Polymorphism (contd.)

```
Shape someShape;
```

```
someShape = new Rectangle (1,2);
```

```
someShape.computeArea ();
```

calls computeArea()
in Rectangle

A horizontal arrow pointing from the text box to the `computeArea ()` method call in the line above.

```
someShape = new Square (4);
```

```
someShape.computeArea();
```

calls computeArea()
in Square

A horizontal arrow pointing from the text box to the `computeArea()` method call in the line above.

- computeArea is a method of Shape, Rectangle and Square

Polymorphism (contd.)

— big deal!?

```
Shape anyShape[] = new Shape[100];
```

```
anyShape[0] = new Rectangle(2,3);
```

```
anyShape[1] = new Triangle (2,5);
```

```
anyShape[2] = new Circle(7);
```

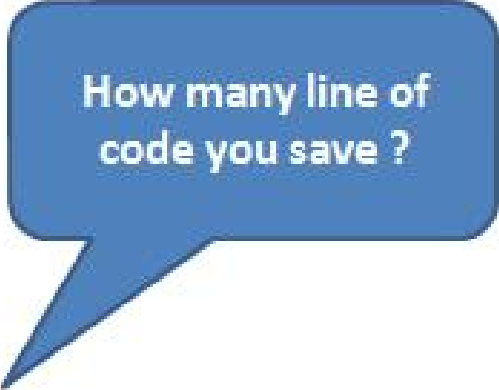
```
.....
```

```
float totalArea=0;
```

```
for (int j =0; j<100; j++)
```

```
    totalArea += anyShape[j].computeArea();
```

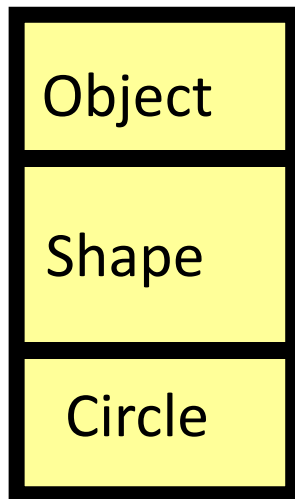
```
// I don't have to worry if it is a Rectangle or Triangle or Square
```



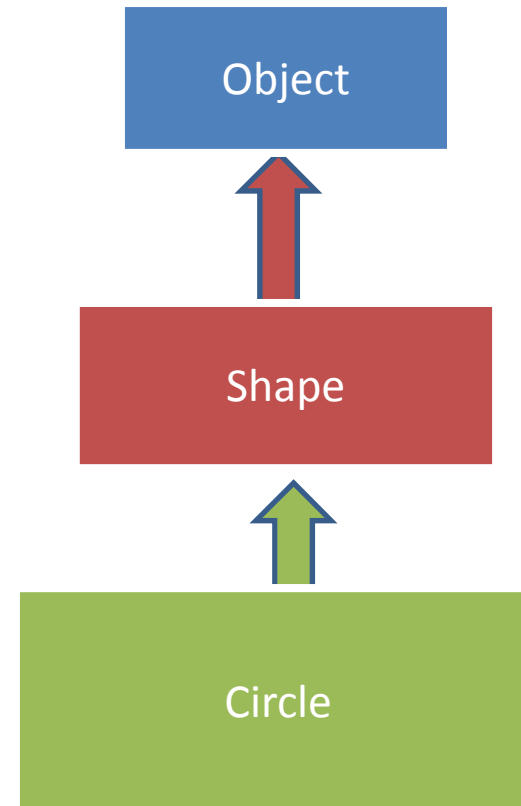
How many line of
code you save ?

Polymorphism

- An object has “multiple identities”, based on its class inheritance tree
- It can be used in different ways
- A Circle is-a Shape is-a Object



A Circle object really has 3 parts

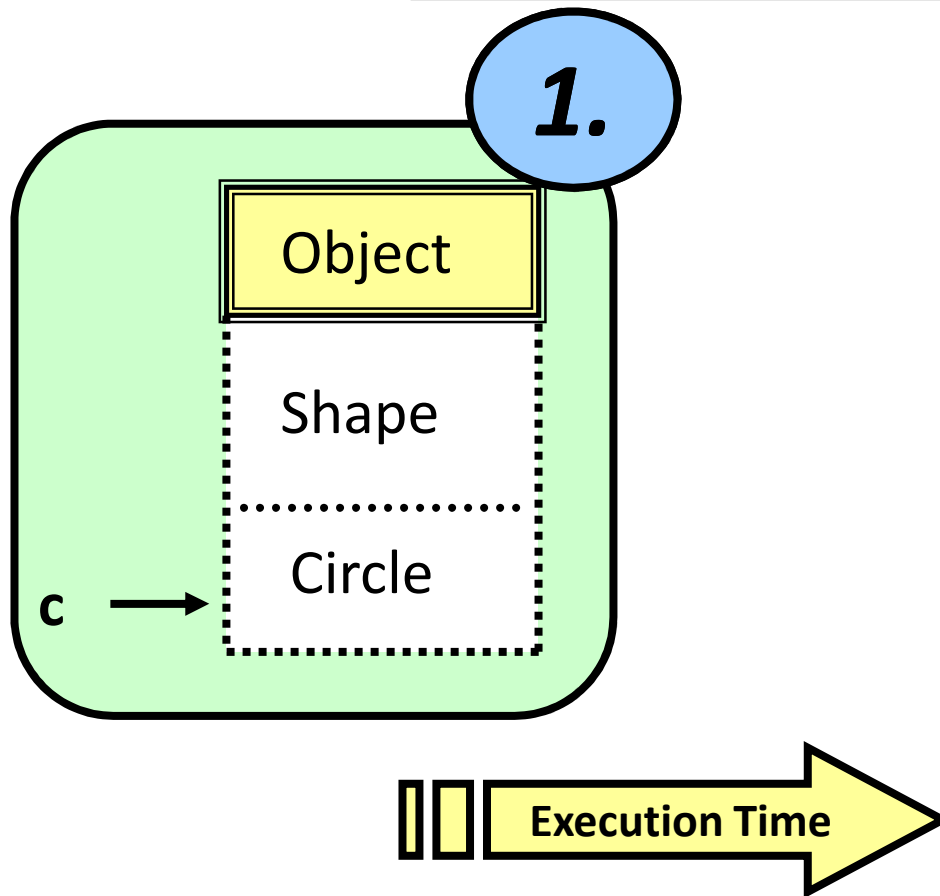


How Objects are Created

```
Circle c = new Circle(3);
```

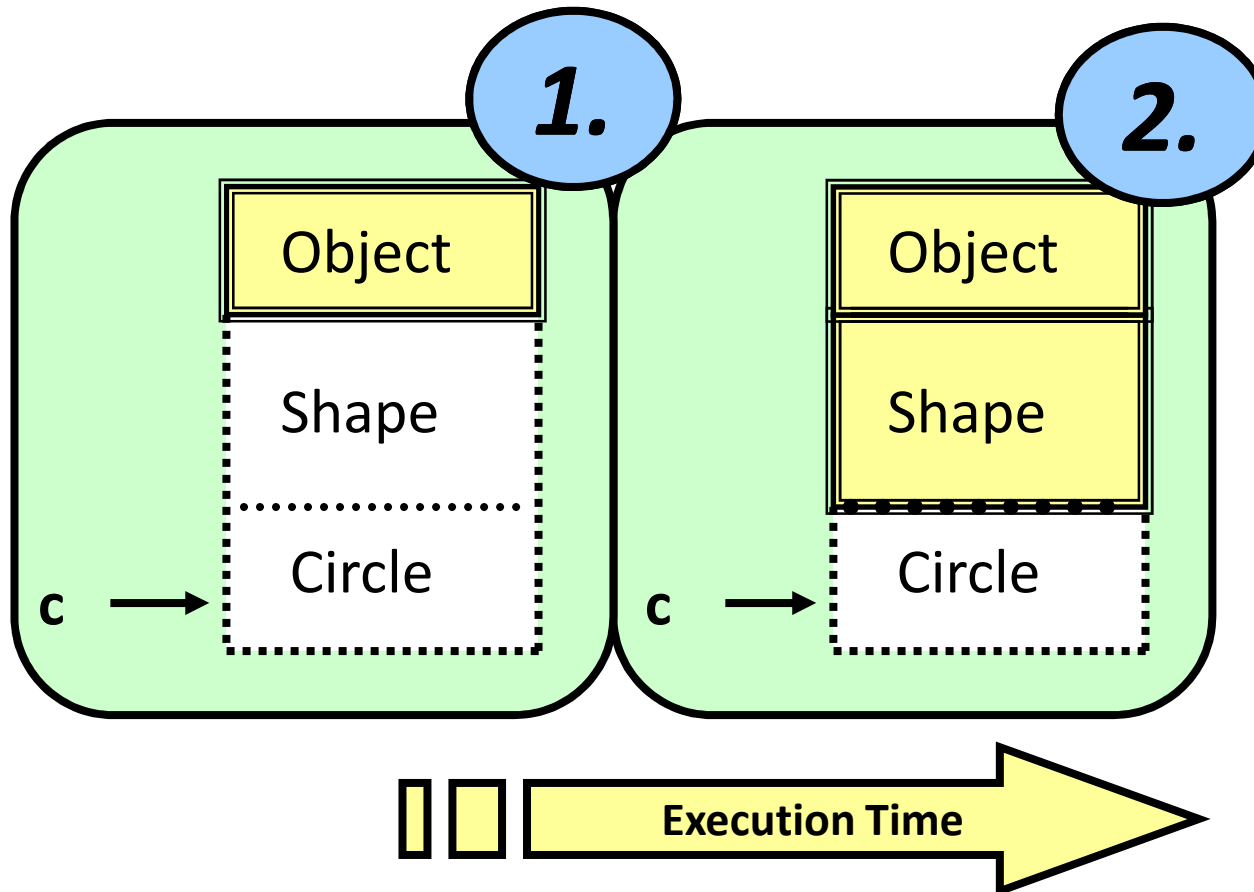
How Objects are Created

```
Circle c = new Circle(3);
```



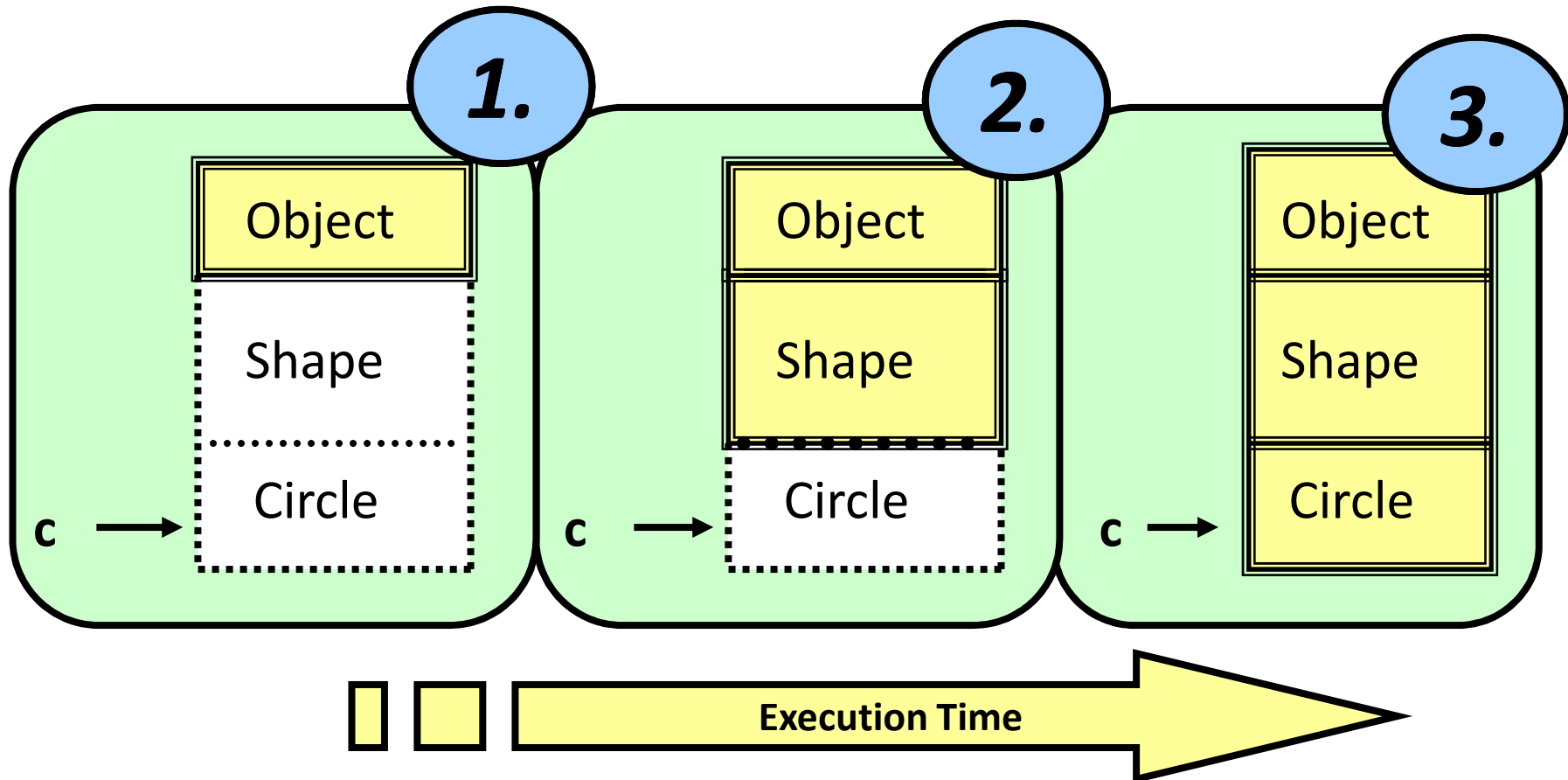
How Objects are Created

```
circle c = new circle(3);
```



How Objects are Created

```
circle c = new circle(3);
```

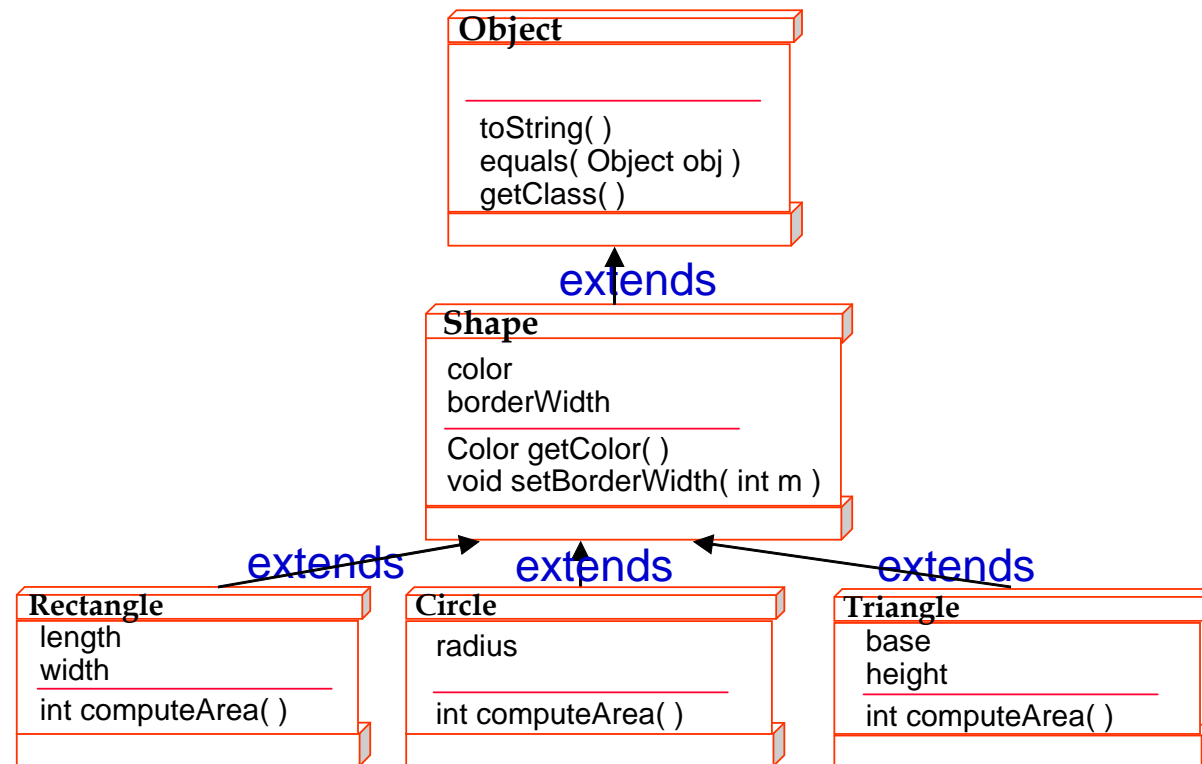


Three Common Uses for Polymorphism

1. Using Polymorphism in Arrays
2. Using Polymorphism for Method Arguments
3. Using Polymorphism for Method Return Type

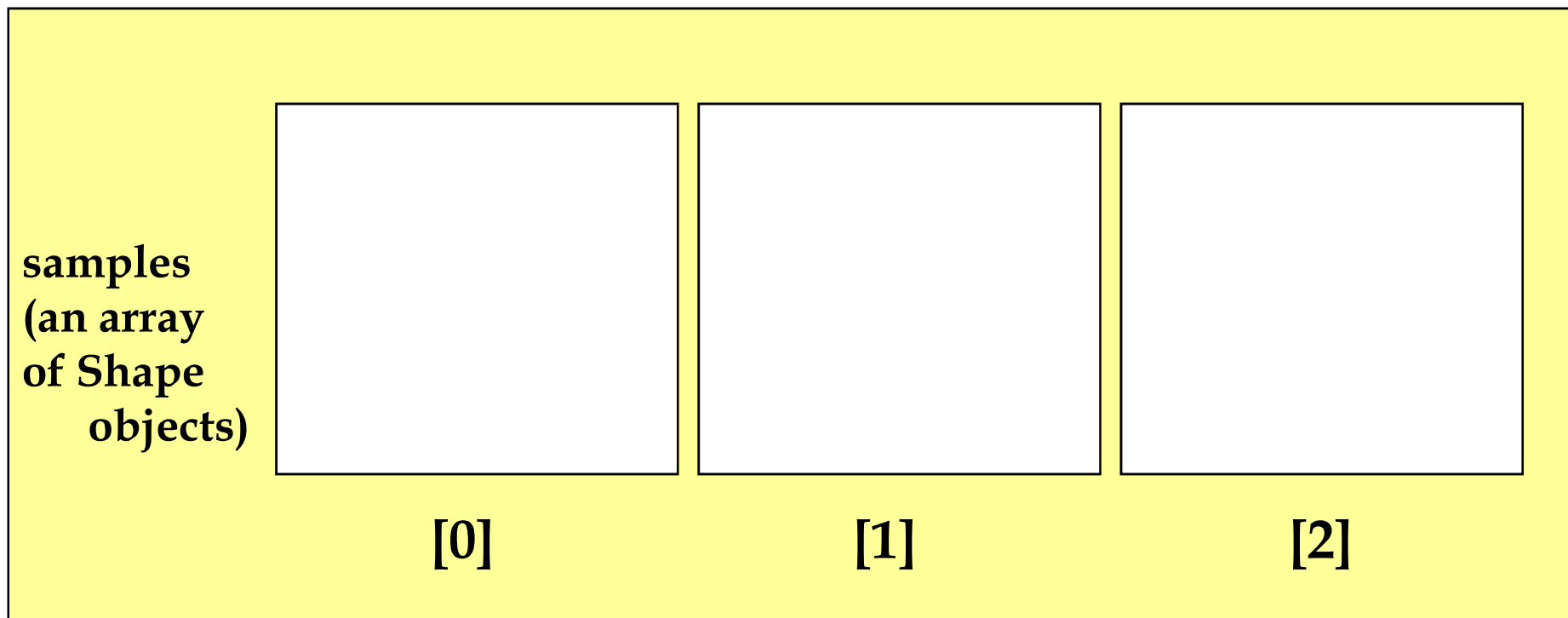
1) Using Polymorphism in Arrays

- We can declare an array to be filled with “Shape” objects, then put in Rectangles, Circles, or Triangles



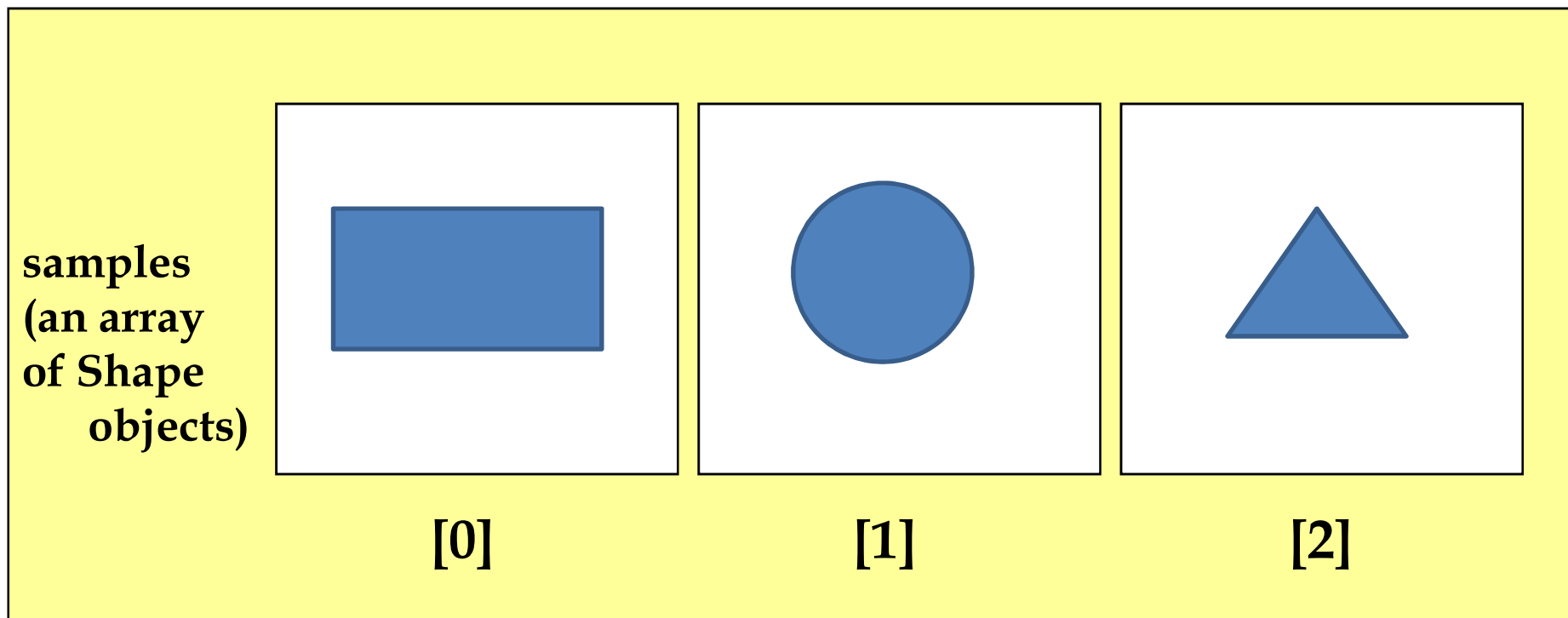
1) Using Polymorphism in Arrays

- We can declare an array to be filled with “Shape” objects, then put in Rectangles, Circles, or Triangles



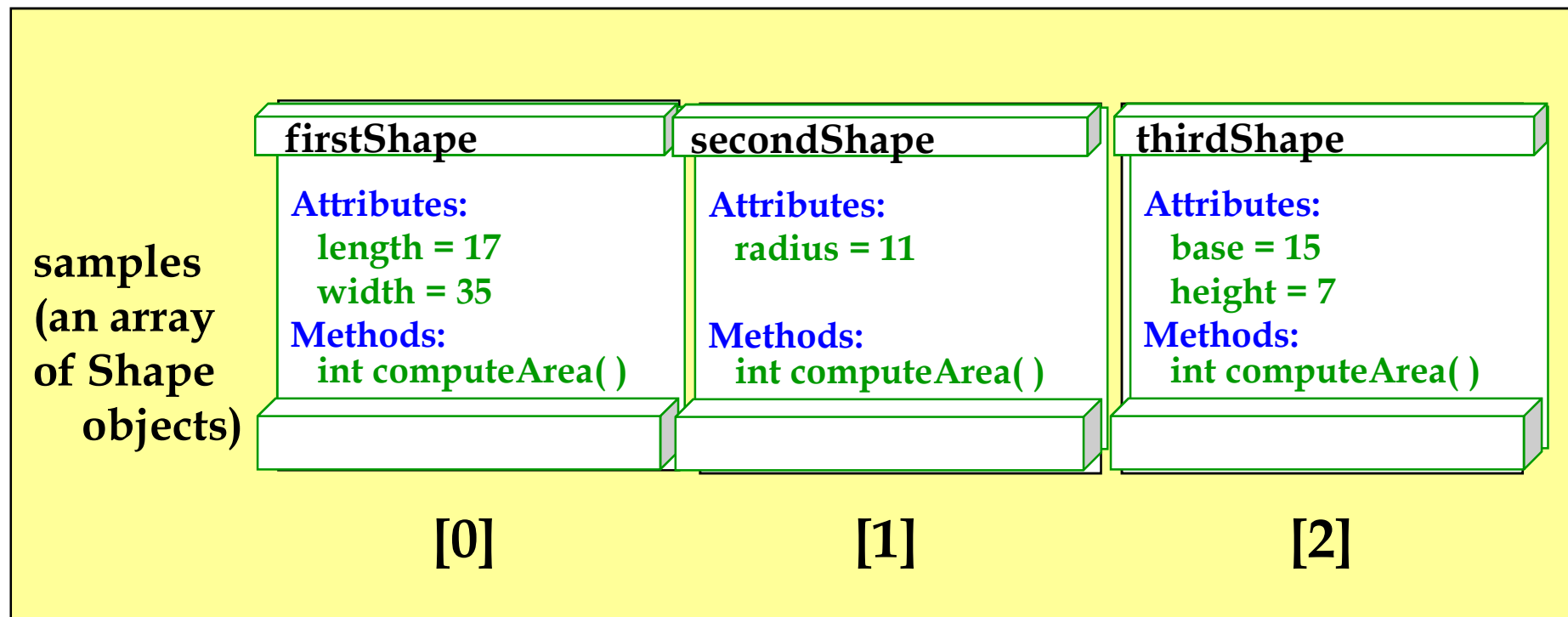
1) Using Polymorphism in Arrays

- We can declare an array to be filled with “Shape” objects, then put in Rectangles, Circles, or Triangles



1) Using Polymorphism in Arrays

- We can declare an array to be filled with “Shape” objects, then put in Rectangles, Circles, or Triangles



Dynamic Method Binding

- Dynamic Method Binding

- At execution time, method calls routed to appropriate version
 - Method called for appropriate class

- Example

- Triangle**, **Circle**, and **Square** all subclasses of **Shape**

- Each has an overridden **computeArea** method

- Call **computeArea** using superclass references

- At static time compiler knows only **computeArea of Superclass**
- At execution time, program determines to which class the reference is actually pointing
- Calls appropriate **computeArea** method

2) Using Polymorphism for Method Arguments

- We can create a procedure that has Shape as the type of its argument, then use it for objects of type Rectangle, Circle, and Triangle

```
public int calculatePaint (Shape myFigure) {  
    final int PRICE = 5;  
  
    int totalCost = PRICE * myFigure.computeArea();  
    return totalCost;  
}
```

The actual definition of **computeArea()** is known only at runtime, not compile time – this is “dynamic binding”

2) Using Polymorphism for Method Arguments

- Polymorphism give us a powerful way of writing code that can handle multiple types of objects, in a unified way

```
public int Test() {  
    Rectangle r = new Rectangle(2,3);  
    Triangle t = new Triangle (2,5);  
    Circle c = new Circle(7);  
    int totalCost = calculatePaint(r) + calculatePaint(t)+  
calculatePaint(c);  
    System.out.println(totalCost );  
}
```

Assign **Rectangle** object to reference **r** is **static binding**

3) Using Polymorphism for Method Return Type

- We can write general code, leaving the type of object to be decided at runtime

```
public Shape createPicture (int i ) {  
    System.out.println("1 for rectangle, " +  
        "2 for circle, 3 for triangle:");
```

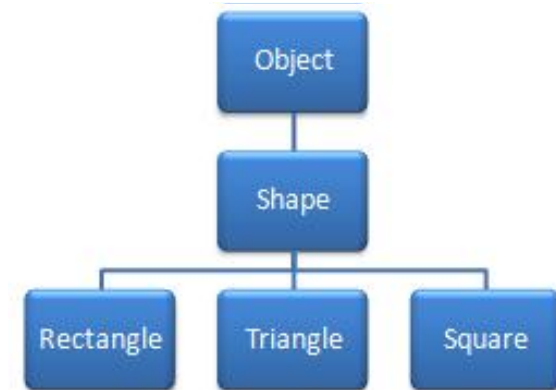
```
    if ( i == 1 ) return new Rectangle(17, 35);  
    if ( i == 2 ) return new Circle(11);  
    if ( i == 3 ) return new Triangle(15, 7);  
}
```

3) Using Polymorphism for Method Return Type

```
public Shape Test() {  
    Shape s1= createPicture(1);  
    Shape s2= createPicture(2);  
    Shape s3= createPicture(3);  
  
    int totalCost = s1.computeArea()+s2.computeArea()+  
s3.computeArea();  
  
    System.out.println(totalCost );  
  
}
```

Object Class

Object class has methods & attribute inherited for all java classes such as:



Methods	Description	Issues
equals()	compares two objects of same type for equality and returns true if equals and false otherwise	shallow compare which mean primitive attributes compared while object not compared(ref only)
toString()	returns a String representation of an object	
clone()	takes no arguments and returns a copy of the object on which it is called	shallow copy which mean primitive attributes copied while object not copied (ref only)

Override Object –methods

```
class Rectangle {  
    private int width = 0;  
    private int height = 0;  
    private Point p= new Point(0,0);  
    public Rectangle(int h, int w){  
        width=w;  
        height=h;  
    }  
    public String toString() {  
        return "width =" + width+ "height =" +  
height+ " x="+p.x + " y="+p.y ;  
    }  
    public boolean equals(Object obj){  
        Rectangle r = (Rectangle )obj;  
        if( width == r. width && height == r. height  
&& p.x == r. p.x && p.y == r.p.y )  
            return true;  
        else return false;  
    } }  
}
```

```
class TestRectangle {  
  
    public static void main(string ar[]){  
        Rectangle r1= new Rectangle (1,2);  
        Rectangle r2= r1.clone();  
        r2..p.setX(5);  
        r2..p.setY(5);  
        System.out.println(r1);  
        System.out.println(r2);  
        System.out.println(r2.equals(r1));  
    }  
}
```

What will happen if we moved r1 ?

Override Object –methods

```
class Rectangle {  
    private int width = 0;  
    private int height = 0;  
    private Point p= new Point(0,0);  
    ....  
    ....  
    ....  
    public Object clone(){  
        Rectangle r = new Rectangle (width,height);  
        r.p.x = p.x ;  
        r.p.y = r.p.y ;  
        return r;  
    }  
}
```

```
class TestRectangle {  
  
    public static void main(string ar[]){  
        Rectangle r1= new Rectangle (1,2);  
        Rectangle r2= r1.clone();  
        r2..p.setX(5);  
        r2..p.setY(5);  
        System.out.println(r1);  
        System.out.println(r2);  
        System.out.println(r2.equals(r1));  
    }  
}
```

Final class/variable/method

- Final class can not be subclasses
 - This is the last class in hierarchy
- Final variable its value can not be changed
 - This consider as constant variable
- Final method can not be override in the subclass
 - The subclass can only use this parent method, but not override
 - Prevent clients from overriding.

Final Example

- **For Class**

```
final public class A{ ... }
```

```
public class B extends A { //this is correct? No
```

- **For variable**

```
public class A{
```

```
    final static int pi = 3.14; // like constant
```

```
    public A(){
```

```
        pi = 3; //this is correct? no
```

```
    }
```

```
}
```

Final Example

- **For method**

```
public class A{  
    public A(){  
        x();  
    }  
    final public void x(){  
    }  
}  
  
public class B extends A {  
    public void x(){ //this correct? no  
    }  
}
```


Finalization

Its similar to destructor

```
void finalize() {  
    System.out.println("Object desctator executed ");  
}
```

This method executed automatically when object destroyed by Java Garbage collection.

Example: Rectangle r = new Rectangle (1,2);
 r= null;
 // now object Rectangle (1,2); are candidate for Garbage
 //collection