# OOP

OOP……. Not OOPS!!

## Object Oriented Programming

### Object-Oriented Programming Concepts

- What is an Object?
- What is a Class?
- What is a Message?
- Encapsulation?
- Inheritance?
- Polymorphism/Dynamic Binding?
- Data Hiding?
- Data Abstraction?

## Java Primitive Data Types

- Primitive Data Types:

- `char` unicode (16 bits)
- `byte` signed 8 bit integer
- `short` signed 16 bit integer
- `int` signed 32 bit integer
- `long` signed 64 bit integer
- `float,double` floating point values

## Other Data Types

- Reference types (composite)
  - objects
  - arrays
- strings are supported by a built-in class named `String (java.lang.String)`.
- string literals are supported by the language

## Attributes

- String color = “Red” ;
- int borderWidth = 5 ;
- //……
- System.out.println(borderWidth) ;

## Custom Data Type

```
public class Shape {
    private String color = null;
    private int borderWidth = 0;

    public int getBorderWidth() {
        return borderWidth;
```

Member variables

# Method Definition

```
public class Shape {
    ..
    public void setBorderWidth(int bw)
    {
       borderWidth = bw;
    }
}
```

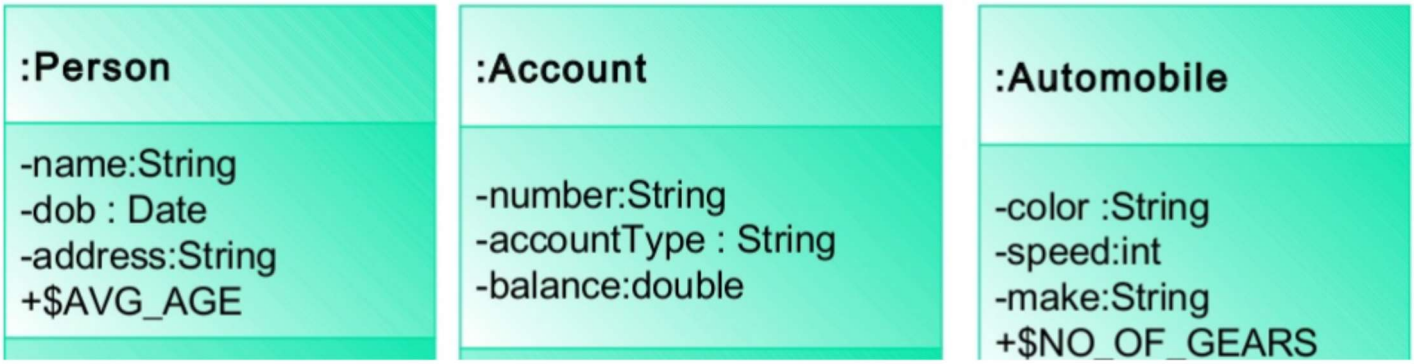
Method

# Define attribute/variable

- **public class** TestShape {
  - **public static void main**(String[] args){
    - Shape s; //Declaration

S is an object here

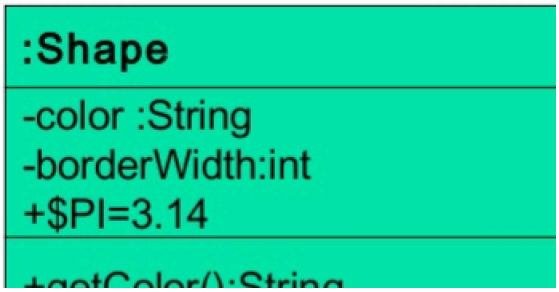
# Real World Entities – More Classes




# Define A Class - Shape

```
public class Shape {
    private string color = null;
    private int borderWidth = 0;
    public static final float PI = 3.14;

    public int getBorderWidth() {
```



public void setBorderWidth(int bw) { +setBorderWidth()

## Constructor

```
Shape s = new Shape();
```

```
public class Shape {
    private String color = null;
    private int borderWidth = 0;

    public Shape(){
    System.out.println(
    "This is default constructor");
    }
.....
```

- Constructor is just like a method.
- It does not have return type.
- Its name is same as Class name.
- It is called at the time of object instantiation (**new** Shape()).
- Constructors are used to initialize instance/class variables.
- A class may have multiple constructors with different number of parameters.

## Multiple Constructors

- One class may have more than one constructors.
- Multiple constructors are used to initialize different sets of class attributes.

- Constructors those receive parameters are called Parameterized Constructors

## Constructors Overloading

```
public class Shape {
    private String color = null;
    private int borderWidth = 0;

    public Shape(){
    System.out.println("This is
       default constuctor")
    }

    public Shape (String c, int w){
        color=c;
        borderWidth=w;
    }
```

```
Shape s = new
   Shape()

s.setColor("Red");
s.setBorderWidth(5);

Or

Shape s = new
   Shape("Red",5)
```

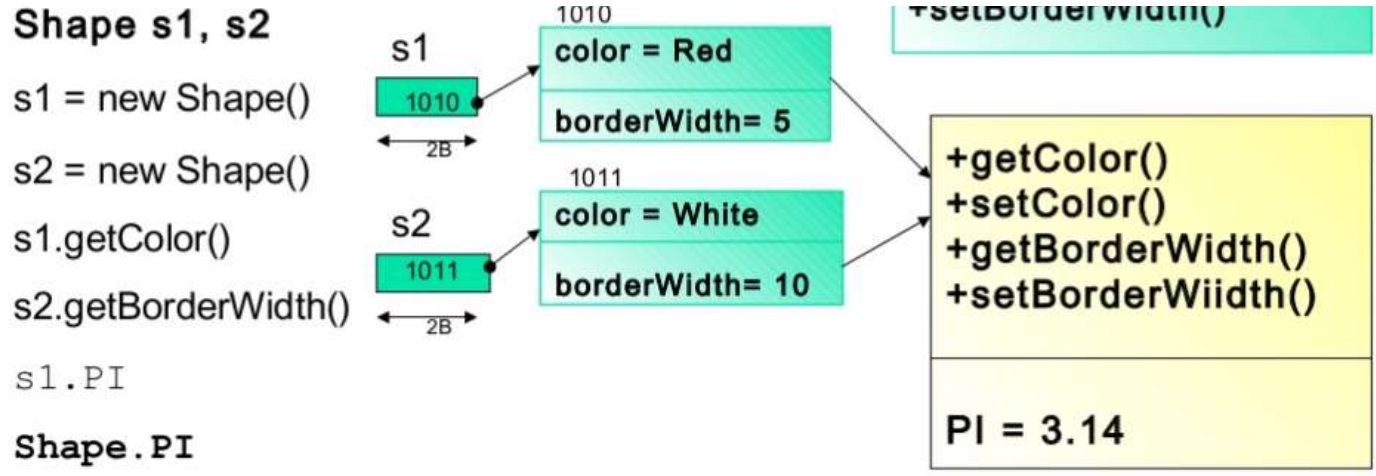
## Default Constructor

- Default constructor does not receive any parameter.
  - public Shape(){ .. }
- If User does not define any constructor then Default Constructor will be created by Java Compiler

# Declare an Instance/Object

**Declare Primitive**

# Instance vs static attributes




# OOP Key Concepts

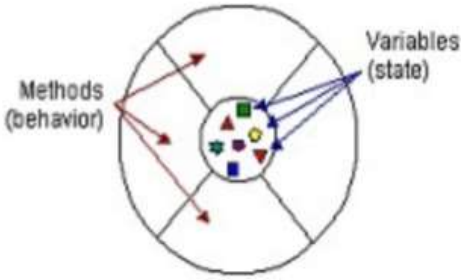
- Encapsulation:
  - Creates Expert Classes.
- Inheritance:
  - Creates Specialized Classes.
- Polymorphism:
  - Provides Dynamic behaviour at Runtime.

# Encapsulation

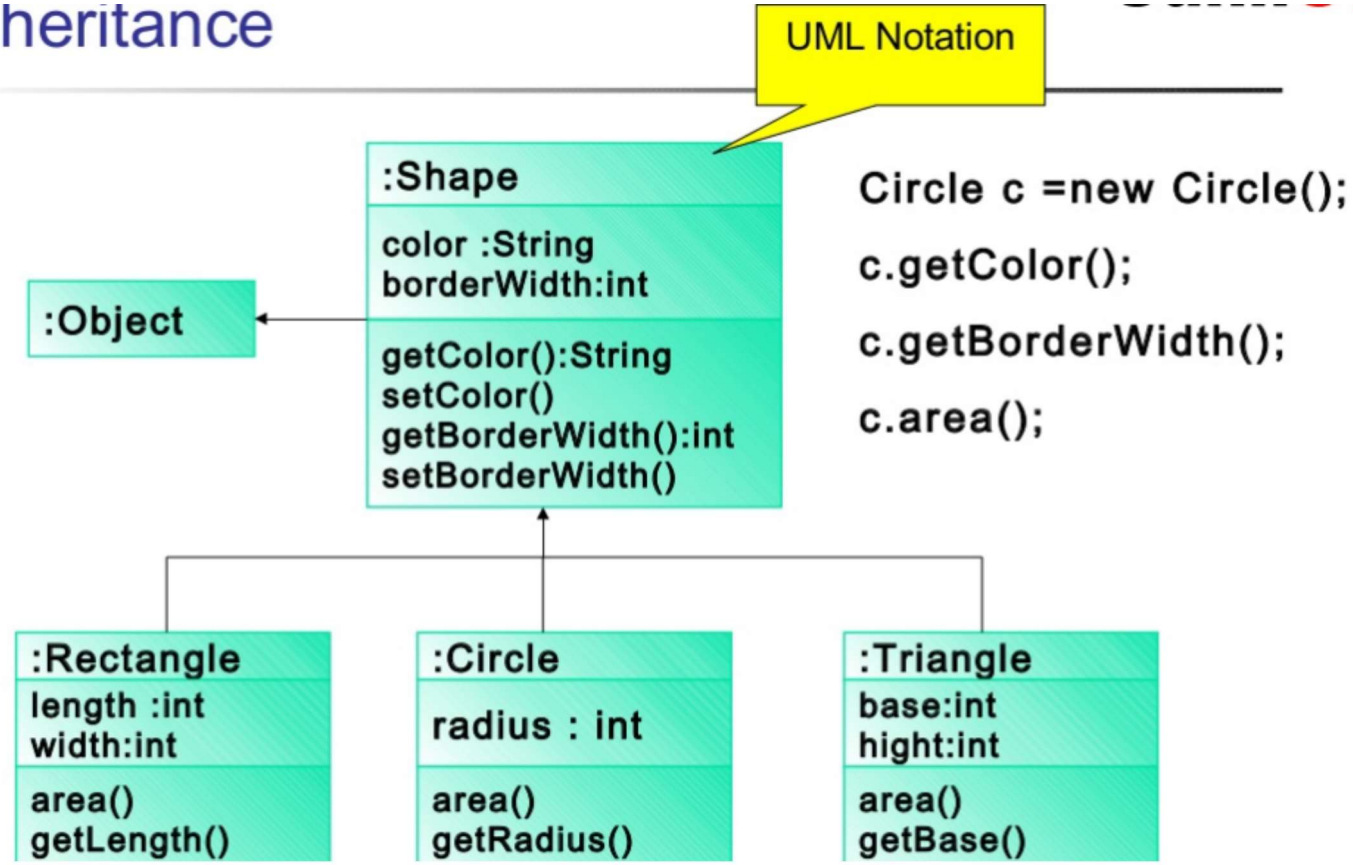
- Often, for practical reasons, an object may wish to expose some of its variables or hide some of its methods.

*Access Levels:*

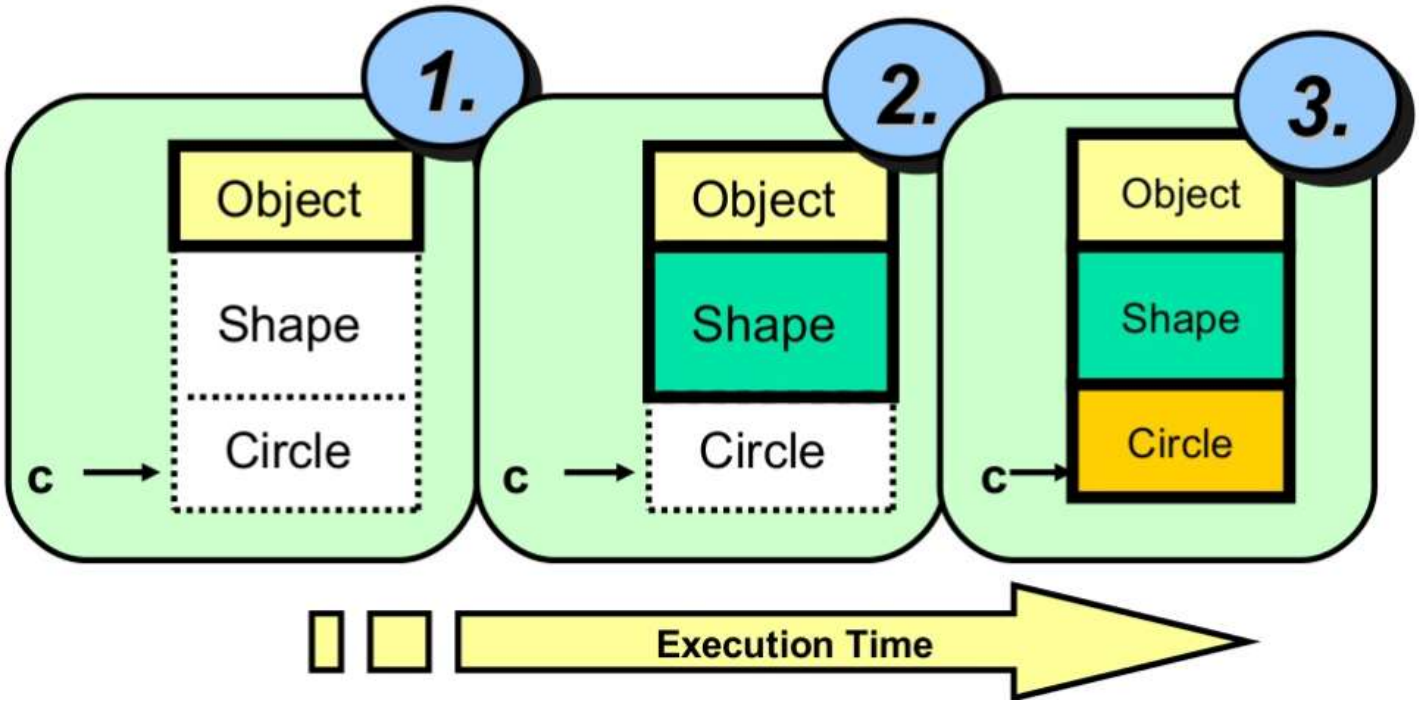
| Modifier | Class | Subclass | Package | World |
|---|---|---|---|---|
| private | X | | | |
| protected | X | X | X | |
| public | X | X | X | X |




## Inheritance



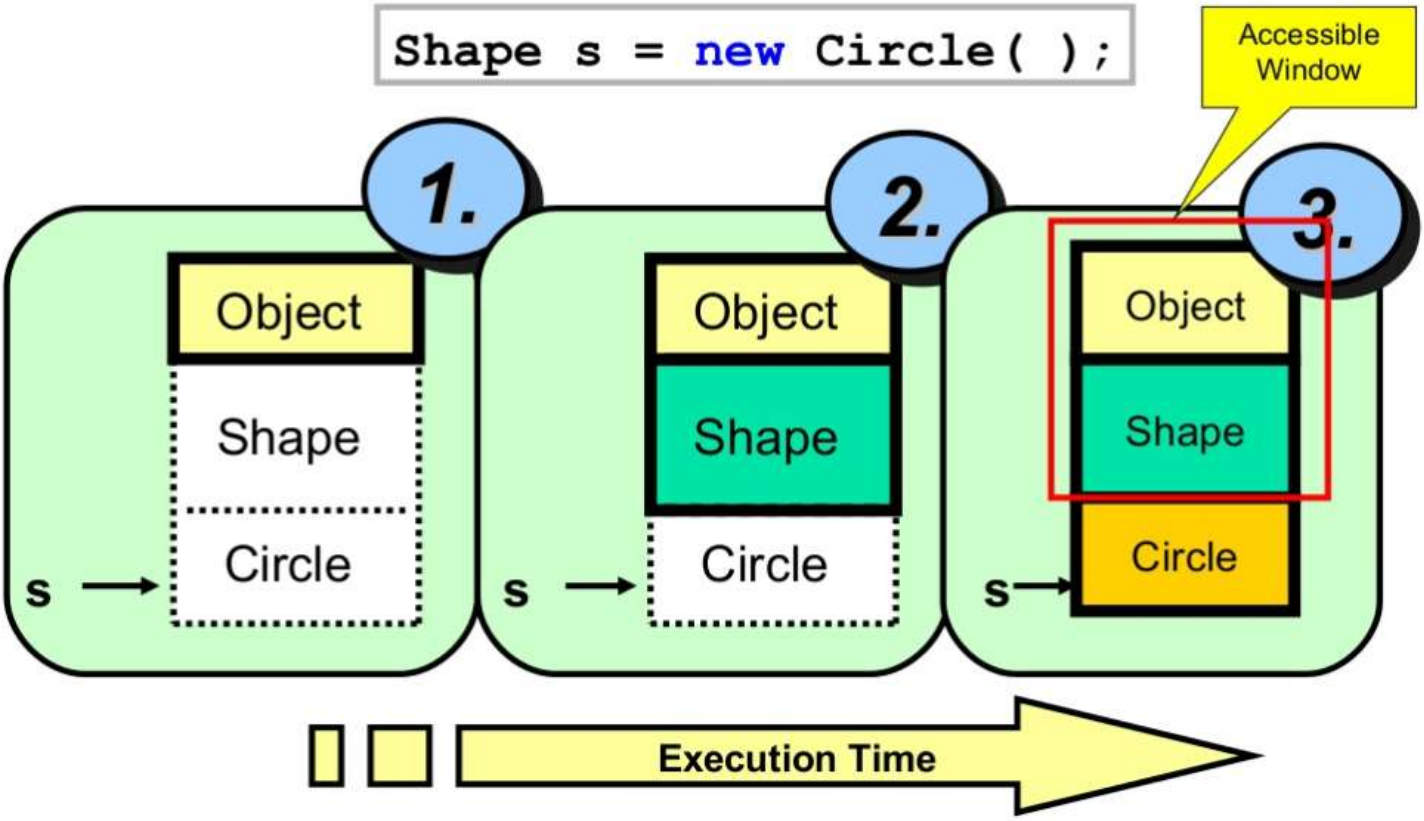

## How Objects are Created



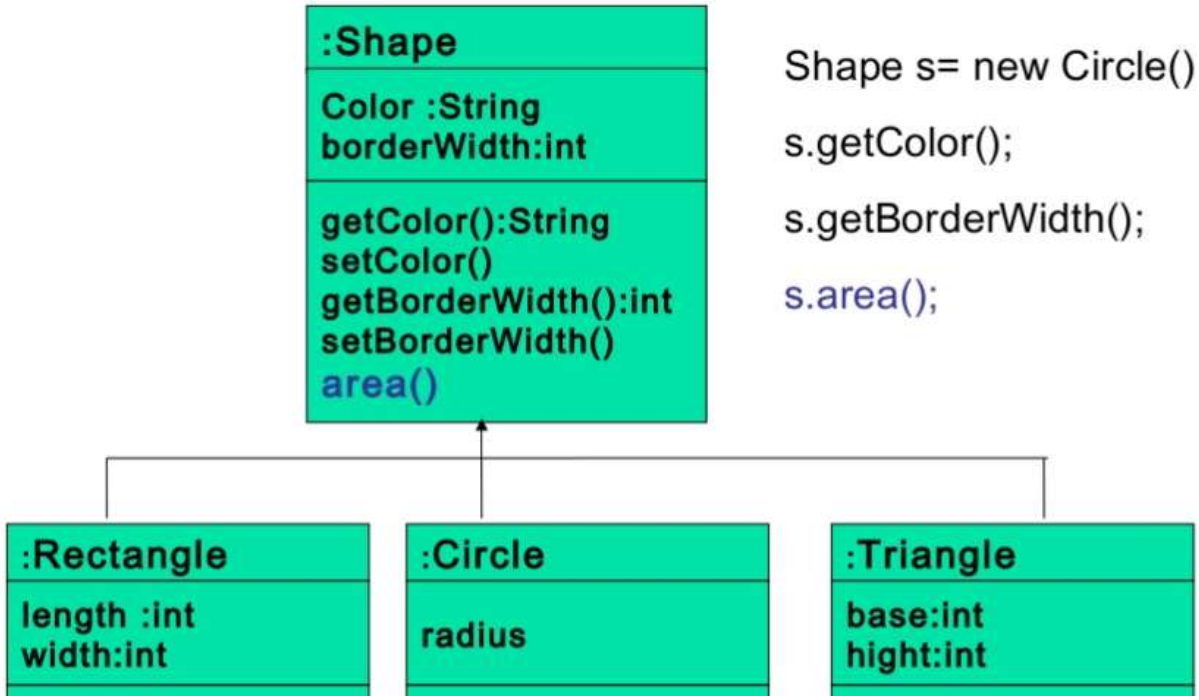

# Parents Can Keep Child's Reference

- Circle c = new Circle();
  - c.getColor()
  - c.getBorderBidth()
  - c.area()
- Shape s = new Circle();
  - s.getColor()
  - s.getBorderBidth()
  - s.area()
- Circle c1 = (Circle) s;
  - c1.getColor()
  - c1.getBorderBidth()
  - c1.area()

# Parents Can Keep Child's Reference




# Method Overriding – area()

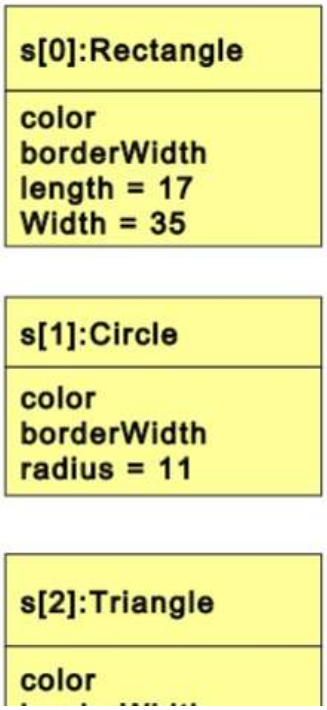


# Polymorphism

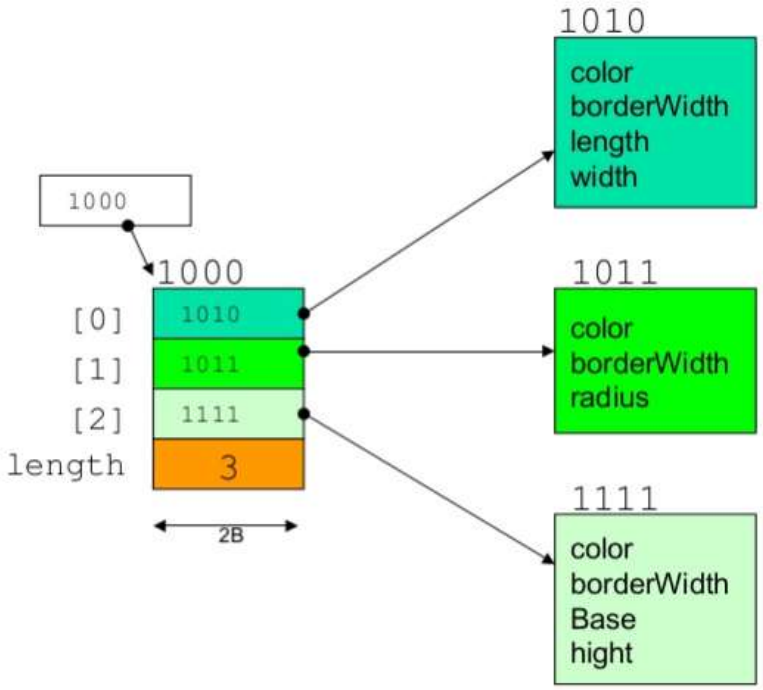
- Three Common Uses of Polymorphism:
  - Using Polymorphism in Arrays.
  - Using Polymorphism for Method Arguments.
  - Using Polymorphism for Method Return Type.
- Ways to Provide polymorphism:
  - Through Interfaces.
  - Method Overriding.
  - Method Overloading.

# 1) Using Polymorphism in Arrays

- Shape s[] = new Shape[3];
- s[0] = new Rectangle()
- s[1] = new Circle()
- s[2] = new Triangle()



# 1) Using Polymorphism in Arrays




- Shape[] s;
- s = new Shape[3];
- s[0] = new Rectangle()
- s[1] = new Circle()
- s[2] = new Triangle()

# 2) Using Polymorphism for Method Arguments

```
public static void main(String[] args) {
    Shape[] s = new Shape[3];
    s[0] = new Rectangle();
    s[1] = new Circle();
    s[2] = new Triangle();
    double totalArea = calcArea(s);
    System.out.println(totalArea);
}
public static double calcArea(Shape[] s) {
    double totalArea = 0;

    for(int i =0;i<s.length; i++){
        totalArea += s[i].area();
    }
    return totalArea;
```

## 3) Polymorphism using Return Type

```
public static Shape getShape(int i) {
    if (i == 1) return new Rectangle();
    if (i == 2) return new Circle();
    if (i == 3) return new Triangle();
}
```

## Method Overloading

- PrintWriter
  - println(String)
  - println(int)
  - println(double)
  - println(boolean)
  - println()

# The Final modifier