

# Core Java 8

## Lesson 02 : Object Orientation



# Lesson Objectives

After completing this lesson, participants will be able to -

- Understand concept of Inheritance and Polymorphism
- Implement inheritance in java programs
- Implement different types of polymorphism

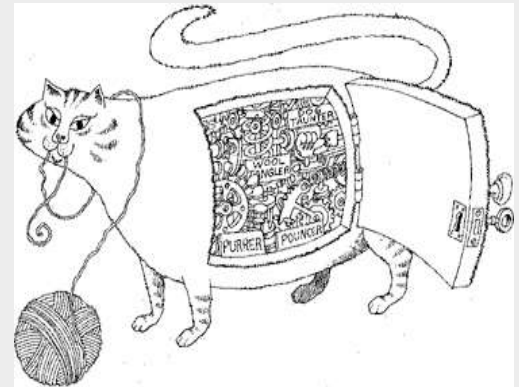




# Encapsulation

“To Hide” details of structure and implementation

- **For example:** It does not matter what algorithm is implemented internally so that the customer gets to view Account status in Sorted Order of Account Number.





# Inheritance, Is-A ,Has- A



**Basic  
TV**



**Smart  
TV**

Smart TV's are inherited from basic television which apart from multimedia functionality of TV allows us to do more like streaming video contents from Internet.



# Inheritance

Inheritance allows programmers to reuse of existing classes and make them extendible either for enhancement or alteration

Allows creation of hierarchical classification

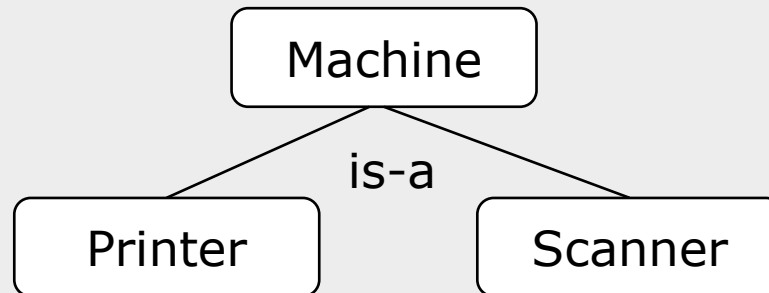
Advantage is reusability of the code:

- A class, once defined and debugged, can be used to create further derived classes

Extend existing code to adapt to different situations

Inheritance is ideal for those classes which has "is-a" relationship

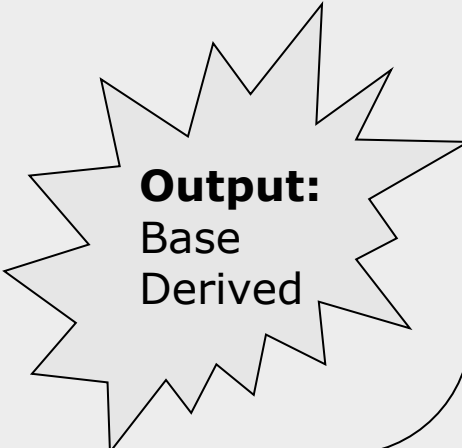
"Object" class is the ultimate superclass in Java





# Inheritance : Example

```
class Base {  
    public void baseMethod() {  
        System.out.println("Base");  
    }  
}  
class Derived extends Base {  
    public void derivedMethod() {  
        super. baseMethod ();  
        System.out.println("Derived");  
    }  
}  
class Test {  
    public static void main(String args[]){  
        Derived derived=new Derived();  
        derived. derivedMethod();  
    }  
}
```

A grey starburst shape with a black outline, containing the output text.

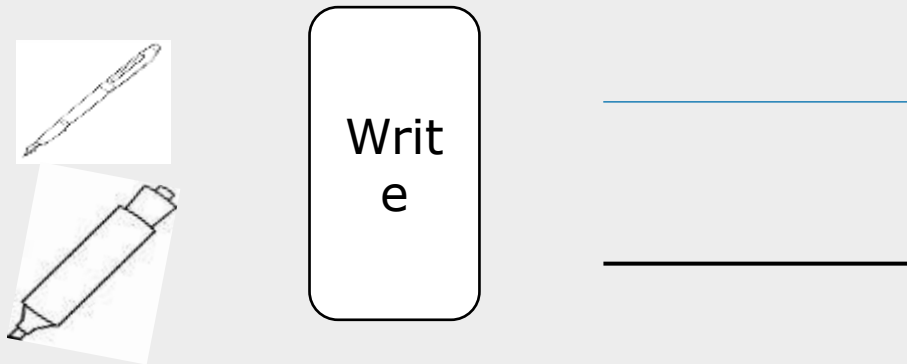
**Output:**  
Base  
Derived



# Polymorphism

Poly meaning “many” and morph means “forms”

It's capability of method to do different things based on the object used for invoking method



Polymorphism also enables an object to determine which method implementation to invoke upon receiving a method call

Java implements polymorphism in two ways

- Method Overloading
- Method Overriding



# Method Overloading

Two or more methods within the same class share the *same* name.

Parameter declarations are different

You can overload Constructors and Normal Methods

```
class Box {  
    Box(){  
        //1. default no-argument constructor  
    }  
    Box(dbl dblValue){  
        // 2. constructor with 1 arg  
    }  
    public static void main(String[] args){  
Box  boxObj1 = new Box(); // calls constructor 1  
Box  boxObj2 = new Box(30); // calls constructor 2  
    } }  

```






# Constructor Overloading

If class has more than one constructors, then they are called as overloaded constructors.

When constructors are overloaded, they must differ in

- Number of parameters
- Type of parameters
- Order of parameters



A same model car can be constructed in different ways as per the requirement. Few of them are basic, while the other differ in fuel type, color, engine power, CNG, A.C. and or other accessories.





# Method Overriding

In a class hierarchy, when a method in a subclass has the same *name* and *type signature* as a method in its super class, then the subclass method overrides the super class method

Overridden methods allow Java to support run-time polymorphism



Normal Swap Machine



Chip card Machine which **overrides** the card reading for better security



## @Override Annotation

The @Override annotation informs the compiler that the element is meant to override an element declared in a superclass

It applies to only methods

```
public class Employee {  
    @override  
    public String toString() {  
        //statements  
        return "EmpName is:"+empname;  
    }  
}
```

Above code will throw a compilation error as it is not overriding the toString method of Object Class



## Reference Variable Casting

- One class types involved must be the same class or a subclass of the other class type
- Assignment to different class types is allowed only if a value of the class type is assigned to a variable of its superclass type
- Assignment to a variable of the subclass type needs explicit casting:

```
String StrObj = Obj;
```

- Explicit casting is not needed for the following:

```
String StrObj = new String("Hello");  
Object Obj = StrObj;
```



# Implementing Interfaces

```
public interface SimpleCalc {
```

```
    int add(int a, int b);
```

abstract method

```
    int i = 10;
```

By default is public, static and final

```
}
```

//Interfaces are to be implemented.

```
class Calculator implements SimpleCalc {
```

```
    int add(int a, int b){
```

```
        return a + b;
```

```
    }
```

```
}
```



# Constructors and Initialization

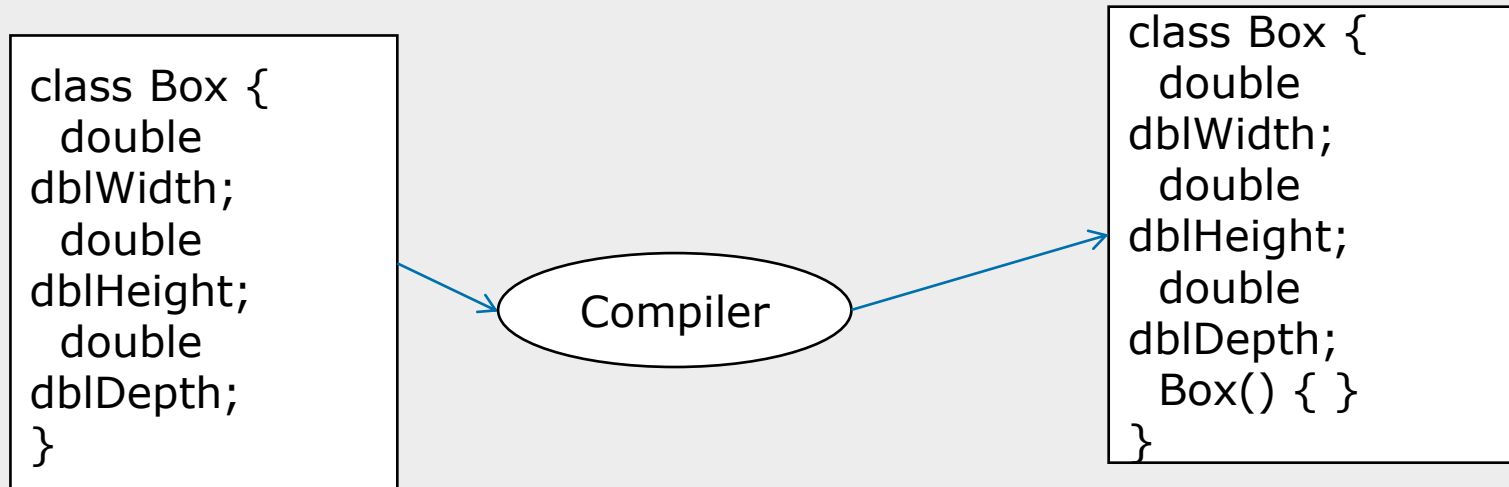
All Java classes have *constructors*

- Constructors initialize a new object of that type

Default no-argument constructor is provided if program has no constructors

Constructors:

- Same name as the class
- No return type, not even void






# Constructor Overloading

If class has more than one constructors, then they are called as overloaded constructors.

When constructors are overloaded, they must differ in

- Number of parameters
- Type of parameters
- Order of parameters



A same model car can be constructed in different ways as per the requirement. Few of them are basic, while the other differ in fuel type, color, engine power, CNG, A.C. and or other accessories.





## Constructor Overloading :Example

```
class Student{  
    int id;  
    String name;  
    //creating a parameterized constructor  
    Student(int i,String n){  
        id = i;  
        name = n;  
    }  
}
```





## Static modifier

Static modifier can be used in conjunction with:

- A variable
- A method

Static members can be accessed before an object of a class is created, by using the class name

Static variable :

- Is shared by all the class members
- Used independently of objects of that class
- Example: `static int intMinBalance = 500;`



# Static Method

## Static methods:

- Can only call other static methods
- Must only access other static data
- Cannot refer to this or super in any way
- Cannot access non-static variables and methods

## Static constructor:

- used to initialize static variables

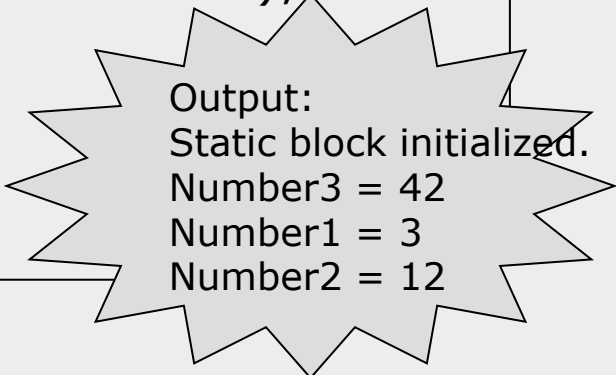
Method `main()` is a static method. It is called by JVM.





## Static modifier -Example

```
// Demonstrate static variables, methods, and blocks.
public class UseStatic {
    static int intNum1 = 3;           // static variable
    static int intNum2;
    static {                          //static constructor
        System.out.println("Static block initialized.");
        intNum2 = intNum1 * 4;
    }
    static void myMethod(int intNum3) { // static method
        System.out.println("Number3 = " + intNum3);
        System.out.println("Number1 = " + intNum1);
        System.out.println("Number2 = " + intNum2);
    }
    public static void main(String args[]) {
        myMethod(42);
    }
}
```



Output:  
Static block initialized.  
Number3 = 42  
Number1 = 3  
Number2 = 12



# Coupling And Cohesion

Coupling :In order to create an effective application that is easy to develop, easy to maintain and easy to update, we need to know the concept of **coupling and cohesion**. **Coupling** refers to the extent to which a class knows about the other class.

There are two types of coupling –

- **Tight Coupling**(*a bad programming design*)
- **Loose Coupling**(*a good programming design*)

Cohesion :**Cohesion** refers to the extent to which a class is defined to do a **specific specialized task**. A class created with high cohesion is targeted towards a single specific purpose, rather than performing many different specific purposes.

There are two types of cohesion –

- **Low cohesion**(*a bad programming design*)
- **High Cohesion**(*a good programming design*)



# Demo

## Polymorphism

- Method Overloading
- Method Overriding





# Summary

In this lesson, you have learnt about:

- Inheritance
- Using super keyword
- InstanceOf Operator
- Method & Constructor overloading
- Method overriding
- @override annotation
- Using final keyword
- Best Practices





# Review Question

Question 1: Which of the following options enable parent class to avoid overriding of its methods.

- extends
- Override
- Final

Question 2: When you want to invoke parent class method from child, it should be written as first statement in child class method

- True/False

Question 3: Which of the following access specifier enables child class residing in different package to access parent class methods?

- private
- public
- Final
- Protected

