

JAVA - OVERRIDING

In the previous chapter, we talked about super classes and sub classes. If a class inherits a method from its super class, then there is a chance to override the method provided that it is not marked final.

The benefit of overriding is: ability to define a behavior that's specific to the subclass type which means a subclass can implement a parent class method based on its requirement.

In object-oriented terms, overriding means to override the functionality of an existing method.

Example:

Let us look at an example.

```
class Animal{
    public void move(){
        System.out.println("Animals can move");
    }
}

class Dog extends Animal{
    public void move(){
        System.out.println("Dogs can walk and run");
    }
}

public class TestDog{
    public static void main(String args[]){
        Animal a = new Animal(); // Animal reference and object
        Animal b = new Dog(); // Animal reference but Dog object

        a.move();// runs the method in Animal class

        b.move();//Runs the method in Dog class
    }
}
```

This would produce the following result:

```
Animals can move
Dogs can walk and run
```

In the above example, you can see that the even though **b** is a type of Animal it runs the move method in the Dog class. The reason for this is: In compile time, the check is made on the reference type. However, in the runtime, JVM figures out the object type and would run the method that belongs to that particular object.

Therefore, in the above example, the program will compile properly since Animal class has the method move. Then, at the runtime, it runs the method specific for that object.

Consider the following example :

```
class Animal{
    public void move(){
        System.out.println("Animals can move");
    }
}

class Dog extends Animal{
    public void move(){
```

```

        System.out.println("Dogs can walk and run");
    }
    public void bark(){
        System.out.println("Dogs can bark");
    }
}

public class TestDog{

    public static void main(String args[]){
        Animal a = new Animal(); // Animal reference and object
        Animal b = new Dog(); // Animal reference but Dog object

        a.move();// runs the method in Animal class
        b.move();//Runs the method in Dog class
        b.bark();
    }
}

```

This would produce the following result:

```

TestDog.java:30: cannot find symbol
symbol   : method bark()
location: class Animal
        b.bark();
        ^

```

This program will throw a compile time error since b's reference type Animal doesn't have a method by the name of bark.

Rules for method overriding:

- The argument list should be exactly the same as that of the overridden method.
- The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass.
- The access level cannot be more restrictive than the overridden method's access level. For example: if the superclass method is declared public then the overriding method in the subclass cannot be either private or protected.
- Instance methods can be overridden only if they are inherited by the subclass.
- A method declared final cannot be overridden.
- A method declared static cannot be overridden but can be re-declared.
- If a method cannot be inherited, then it cannot be overridden.
- A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
- A subclass in a different package can only override the non-final methods declared public or protected.
- An overriding method can throw any unchecked exceptions, regardless of whether the overridden method throws exceptions or not. However the overriding method should not throw checked exceptions that are new or broader than the ones declared by the overridden method. The overriding method can throw narrower or fewer exceptions than the overridden method.
- Constructors cannot be overridden.

Using the super keyword:

When invoking a superclass version of an overridden method the **super** keyword is used.

```

class Animal{

    public void move(){

```

```
        System.out.println("Animals can move");
    }
}

class Dog extends Animal{

    public void move(){
        super.move(); // invokes the super class method
        System.out.println("Dogs can walk and run");
    }
}

public class TestDog{

    public static void main(String args[]){

        Animal b = new Dog(); // Animal reference but Dog object
        b.move(); //Runs the method in Dog class

    }
}
```

This would produce the following result:

```
Animals can move
Dogs can walk and run
```