Ankita Srivastava

Cs department

constructor overloading and overriding in java

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.The compiler differentiates these constructors by taking into account the number of parameters in the list and their type

Overriding and Overloading are two very important concepts in Java. They are confusing for Java novice programmers.

Overriding means having two methods with the same method name and parameters (i.e., method signature). One of the methods is in the parent class and the other is in the child class. Overriding allows a child class to provide a specific implementation of a method that is already provided its parent class.

Rules for creating a Java Constructor

1. It has the same name as the class
2. It should not return a value not even void

Example 1: Create your First Constructor Java

Step 1) Type following code in your editor.

```
class Demo{
     int  value1;
     int  value2;
     Demo(){
       value1 = 10;
       value2 = 20;
       System.out.println("Inside Constructor");
    }

    public void display(){
      System.out.println("Value1 === "+value1);
      System.out.println("Value2 === "+value2);
   }

  public static void main(String args[]){
      Demo d1 = new Demo();
     d1.display();
  }
}
```

Step 2) Save , Run & Compile the code.

Output:

```
Inside Constructor
Value1 === 10
Value2 === 20
```

2. Overriding vs. Overloading

Here are some important facts about Overriding and Overloading:

1). The real object type in the run-time, not the reference variable's type, determines which overridden method is used at runtime. In contrast, reference type determines which overloaded method will be used at compile time.
2). Polymorphism applies to overriding, not to overloading.
3). Overriding is a run-time concept while overloading is a compile-time concept.

CONSTRUCTOR OVERRIDE EXAMPLE

```
class Test

{

  public Test()

  {

    System.out.println("Hello 1");

  }

}


public class Demo extends Test

{
  public Test()
  {
    System.out.println("Hello 2");
  }
}
```

1. a constructor cannot be overridden.
2. a constructor cannot be called with an object but method can be called. To call a constructor, an object must be created.
3. in case of methods, we say, the "subclass method can call super class method". But it cannot be said with constructors and inturn we should say "subclass constructor can access super class constructor"; here, correct word is "access" and not "call". It is for the reason, a constructor cannot be overridden.
4. we cannot call a super class constructor with super keyword. For constructors, specially one more term is created by
   Designers – super().
5. in method overriding, the super class method and subclass method should be of the same – same name, same return type and same parameter list. In constructors, same name constructor cannot be written in another class; it is compilation error.

An Example of Overriding

```
class Dog{
   public void bark(){
      System.out.println("woof ");
   }
}
class Hound extends Dog{
   public void sniff(){
      System.out.println("sniff ");
```

```
    }

    public void bark(){
        System.out.println("bowl");
    }
}

public class OverridingTest{
    public static void main(String [] args){
        Dog dog = new Hound();
        dog.bark();
    }
}
```

Output:

```
bowl
```

In the example above, the `dog` variable is declared to be a `Dog`. During compile time, the compiler checks if the `Dog` class has the `bark()` method. As long as the `Dog` class has the `bark()` method, the code compilers. At run-time, a `Hound` is created and assigned to `dog`. The JVM knows that `dog` is referring to the object of `Hound`, so it calls the `bark()` method of `Hound`. This is called Dynamic Polymorphism.

4. An Example of Overloading

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

    //overloading method
    public void bark(int num){
                for(int i=0; i<num; i++)
                        System.out.println("woof ");
    }
}
```

In this overloading example, the two `bark` method can be invoked by using different parameters. Compiler know they are different because they have different method signature (method name and method parameter list).

Note--

The most basic difference is that overloading is being done in the same class while for overriding base and child classes are required. Overriding is all about giving a specific implementation to the inherited method of parent class. ... Argument list should be same in method Overriding….