Task 06/05/2020

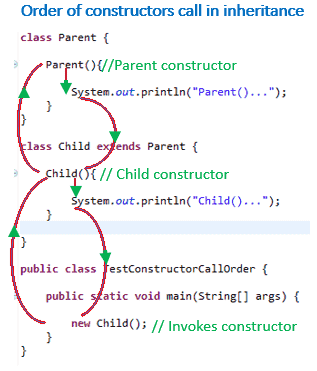
1. Constructor invocation in inheritance hierarchy

***In inheritance relationship, when we create an object of a child class, then first base class constructor and then derived class constructor get called implicitly.***

**Example-**

Here are the steps how the program control flows in the image.

1. You create an object of child class in main() as “new Child()”
2. Control goes to child constructor, but, its body is not getting executed.
3. Control goes to parent constructor, body of it get executed.
4. Then, control comes back to child constructor and body get executed.
5. Then, the controls come back to “new Child()” statement and exit.



1. Difference between no arg constructor and default constructor

**Default Constructor in java:**

* When we write a class without any constructor then at compilation time java compiler creates a default constructor in our class.
* The accessibility modifier of the default constructor is same as accessibility modifier of class.
* The allowed accessibility modifier are public and default.
* Default constructor added by java compiler this constructor does not have anything except super (); call.
* If our class have any constructor then java compiler does not create default constructor

**No-argument Constructor in java:**

* As a developer we can create our own constructor with no arguments is known as no-argument constructor.
* It can have all four accessibility modifiers as it is defined by developer.
* So allowed accessibility modifiers are public, private, protected and default
* It can have logic including super call.
* The common point between default and no-argument constructor
* Both does not have any arguments.
* And one more point we need to remember that in no-argument constructor also by default first statement will be super () call which is added by java compiler if it does not have.

1. If my base class method is static, can I override that method in sub class?

We can declare static methods with same signature in subclass, but it is not considered overriding as there won’t be any run-time polymorphism. Hence the answer is ‘No’.  
If a derived class defines a static method with same signature as a static method in base class, the method in the derived class hides the method in the base class.

Example-

/\* Java program to show that if static method is redefined by

a derived class, then it is not overriding. \*/

// Superclass

class Base {

// Static method in base class which will be hidden in subclass

public static void display() {

System.out.println("Static or class method from Base");

}

// Non-static method which will be overridden in derived class

public void print() {

System.out.println("Non-static or Instance method from Base");

}

}

// Subclass

class Derived extends Base {

// This method hides display() in Base

public static void display() {

System.out.println("Static or class method from Derived");

}

// This method overrides print() in Base

public void print() {

System.out.println("Non-static or Instance method from Derived");

}

}

// Driver class

public class Test {

public static void main(String args[ ]) {

Base obj1 = new Derived();

// As per overriding rules this should call to class Derive's static

// overridden method. Since static method can not be overridden, it

// calls Base's display()

obj1.display();

// Here overriding works and Derive's print() is called

obj1.print();

}

}

Output:

Static or class method from Base

Non-static or Instance method from Derived