**Constructor Overloading in Java with examples**

[**OOPS CONCEPT**](http://beginnersbook.com/category/oops-concept/)

Like methods, [**constructors**](http://beginnersbook.com/2013/03/constructors-in-java/) can also be overloaded. We will see constructor[**overloading**](http://beginnersbook.com/2013/05/method-overloading/) with the help of an example using this() and parameterized constructor. Before we got through the source code and examples lets discuss why we need to overload a constructor:

Constructor overloading is way of having more than one constructor which does different-2 tasks. For e.g. Vector class has 4 types of constructors. If you do not want to specify the initial capacity and capacity increment then you can simply use default constructor of [**Vector class**](http://beginnersbook.com/2013/12/vector-in-java/) like this Vector v = new Vector(); however if you need to specify the capacity and increment then you call the parameterized constructor with two int args like this:Vector v= new Vector(10, 5);

You must have understood the need to overloading. Lets see how to overload a constructor with the help of below **example program**:

package beginnersbook.com;

public class StudentData

{

private int stuID;

private String stuName;

private int stuAge;

StudentData()

{

//Default constructor

stuID = 100;

stuName = "New Student";

stuAge = 18;

}

StudentData(int num1, String str, int num2)

{

//Parameterized constructor

stuID = num1;

stuName = str;

stuAge = num2;

}

//Getter and setter methods

public int getStuID() {

return stuID;

}

public void setStuID(int stuID) {

this.stuID = stuID;

}

public String getStuName() {

return stuName;

}

public void setStuName(String stuName) {

this.stuName = stuName;

}

public int getStuAge() {

return stuAge;

}

public void setStuAge(int stuAge) {

this.stuAge = stuAge;

}

}

class TestOverloading

{

public static void main(String args[])

{

//This object creation would call the default constructor

StudentData myobj = new StudentData();

System.out.println("Student Name is: "+myobj.getStuName());

System.out.println("Student Age is: "+myobj.getStuAge());

System.out.println("Student ID is: "+myobj.getStuID());

/\*This object creation would call the parameterized

\* constructor StudentData(int, String, int)\*/

StudentData myobj2 = new StudentData(555, "Chaitanya", 25);

System.out.println("Student Name is: "+myobj2.getStuName());

System.out.println("Student Age is: "+myobj2.getStuAge());

System.out.println("Student ID is: "+myobj2.getStuID());

}

}

Output:

Student Name is: New Student

Student Age is: 18

Student ID is: 100

Student Name is: Chaitanya

Student Age is: 25

Student ID is: 555

As you can see in the above example that while creating the instance myobj, default constructor (StudentData()) gets called however during the creating of myobj2, the arg-constructor (StudentDate(int, String, int)) being called.Since both the constructors are having different initialization code the variables value are different in each case as shown in the output of the program.

**Let’s see role of this () in constructor overloading**

package beginnersbook.com;

public class ConstOverloading

{

private int rollNum;

ConstOverloading()

{

rollNum =100;

}

ConstOverloading(int rnum)

{

this();

/\*this() is used for calling the default

\* constructor from parameterized constructor.

\* It should always be the first statement

\* in constructor body.

\*/

rollNum = rollNum+ rnum;

}

public int getRollNum() {

return rollNum;

}

public void setRollNum(int rollNum) {

this.rollNum = rollNum;

}

}

class TestDemo{

public static void main(String args[])

{

ConstOverloading obj = new ConstOverloading(12);

System.out.println(obj.getRollNum());

}

}

Output:

112

As you can see in the above program that we called arg-constructor during object creation (ConstOverloading obj = new ConstOverloading(12);). However since we have placed the this() statement inside it, the default constructor implicitly being called from it.  
Test your skills – Guess the output of below program

package beginnersbook.com;

public class ConstOverloading

{

private int rollNum;

ConstOverloading()

{

rollNum =100;

}

ConstOverloading(int rnum)

{

rollNum = rollNum+ rnum;

this();

}

public int getRollNum() {

return rollNum;

}

public void setRollNum(int rollNum) {

this.rollNum = rollNum;

}

}

class TestDemo{

public static void main(String args[])

{

ConstOverloading obj = new ConstOverloading(12);

System.out.println(obj.getRollNum());

}

}

Output:

Exception in thread "main" java.lang.Error: Unresolved compilation

problem:Constructor call must be the first statement in a constructor

Program caused a compilation error. **Reason**: this() should be the first statement inside a constructor.

**Another important point** to note while overloading a constructor is: When we don’t define any constructor, the compiler creates the default constructor(also known as no-arg constructor) by default during compilation however if we have defined a parameterized constructor and didn’t define a no-arg constructor then while calling no-arg constructor the program would fail as in this case compiler doesn’t create a no-arg constructor.

Lets see the above point with the example program:

package beginnersbook.com;

public class Demo

{

private int rollNum;

//We are not defining a no-arg constructor here

Demo(int rnum)

{

rollNum = rollNum+ rnum;

}

//Getter and Setter methods

}

class TestDemo{

public static void main(String args[])

{

//This statement would call no-arg constructor

Demo obj = new Demo();

}

}

Output:

Exception in thread "main" java.lang.Error: Unresolved compilation

problem:The constructor Demo() is undefined

# Vector in Java

[**JAVA COLLECTIONS**](http://beginnersbook.com/category/java-collections/)

Vector implements List Interface. Like ArrayList it also maintains insertion order but it is rarely used in non-thread environment as it is synchronized and due to which it gives poor performance in searching, adding, delete and update of its elements.

#### Three ways to create vector class object:

**Method 1:**  
Vector vec = new Vector();  
It creates an empty Vector with the default initial capacity of 10. It means the Vector will be re-sized when the 11th elements needs to be inserted into the Vector. Note: By default vector doubles its size. i.e. In this case the Vector size would remain 10 till 10 insertions and once we try to insert the 11th element It would become 20 (double of default capacity 10).

**Method 2:**  
Syntax: Vector object= new Vector(int initialCapacity)  
Vector vec = new Vector(3);  
It will create a Vector of initial capacity of 3.

**Method 3:**  
Syntax:  
Vector object= new vector(int initialcapacity, capacityIncrement)  
Vector vec= new Vector(4, 6)  
Here we have provided two arguments. The initial capacity is 4 and capacityIncrement is 6. It means upon insertion of 5th element the size would be 10 (4+6) and on 11th insertion it would be 16(10+6).

## Complete Example of Vector in Java:

import java.util.\*;

public class VectorExample {

public static void main(String args[]) {

/\* Vector of initial capacity(size) of 2 \*/

Vector<String> vec = new Vector<String>(2);

/\* Adding elements to a vector\*/

vec.addElement("Apple");

vec.addElement("Orange");

vec.addElement("Mango");

vec.addElement("Fig");

/\* check size and capacityIncrement\*/

System.out.println("Size is: "+vec.size());

System.out.println("Default capacity increment is: "+vec.capacity());

vec.addElement("fruit1");

vec.addElement("fruit2");

vec.addElement("fruit3");

/\*size and capacityIncrement after two insertions\*/

System.out.println("Size after addition: "+vec.size());

System.out.println("Capacity after increment is: "+vec.capacity());

/\*Display Vector elements\*/

Enumeration en = vec.elements();

System.out.println("\nElements are:");

while(en.hasMoreElements())

System.out.print(en.nextElement() + " ");

}

}

Output:

Size is: 4

Default capacity increment is: 4

Size after addition: 7

Capacity after increment is: 8

Elements are:

Apple Orange Mango Fig fruit1 fruit2 fruit3

## Important methods of Vector Class:

1. **void addElement(Object element):** It inserts the element at the end of the Vector.
2. **int capacity():** This method returns the current capacity of the vector.
3. **int size():** It returns the current size of the vector.
4. **void setSize(int size):** It changes the existing size with the specified size.
5. **boolean contains(Object element):** This method checks whether the specified element is present in the Vector. If the element is been found it returns true else false.
6. **boolean containsAll(Collection c):** It returns true if all the elements of collection c are present in the Vector.
7. **Object elementAt(int index):** It returns the element present at the specified location in Vector.
8. **Object firstElement():** It is used for getting the first element of the vector.
9. **Object lastElement():** Returns the last element of the array.
10. **Object get(int index):** Returns the element at the specified index.
11. **boolean isEmpty():** This method returns true if Vector doesn’t have any element.
12. **boolean removeElement(Object element):** Removes the specifed element from vector.
13. **boolean removeAll(Collection c):** It Removes all those elements from vector which are present in the Collection c.
14. **void setElementAt(Object element, int index):** It updates the element of specifed index with the given element.