UNIT II – Control Statements and Constructors

Control structures – Arrays - Objects and classes: Classes – Access Specifiers – methods and attributes - **constructors: Default Constructor** – Parameterized Constructor – Copy Constructor- Garbage collection.

Constructors: Default Constructor

Definition

- A constructor is a block of codes similar to the method.
- It is called when an instance of the class is created.
- At the time of calling constructor, memory for the object is allocated in the memory.
- It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called. It calls a **default constructor** if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Rules for creating constructor in java

There are two rules defined for the constructor.

1. Constructor name **must be the same as its class name**
2. A Constructor must have **no explicit return type**
3. A Java constructor **cannot be abstract, static, final, and synchronized**

Note - can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

Types of constructors in java

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor
3. Copy Constructor

Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax

```
<class_name>()
{
}
```

Example 1

```
class Bike1{
Bike1()
{
System.out.println("Bike is created");
}
public static void main(String args[])
{
Bike1 b=new Bike1();
}
}
```

Example 2

```
class Main
{
int i;
private Main()
{
i = 5;
System.out.println("Constructor is called");
}
public static void main(String[] args)
{
Main obj = new Main();
System.out.println("Value of i: " + obj.i);
}
}
```

```
Constructor is called
Value of i: 5
```

Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used **to provide different values to distinct objects**. However, you can provide the same values also.

Example

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

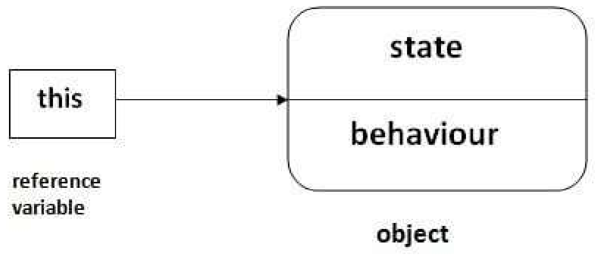
```
class Student4{
   int id;
   String name;
  Student4(int i,String n)
 {
   id = i;
   name = n;
   }
   void display(){System.out.println(id+" "+name);}
   public static void main(String args[]){
   Student4 s1 = new Student4(111,"Karan");
   Student4 s2 = new Student4(222,"Aryan");
   s1.display();
   s2.display();
  }
}
```

```
111 Karan
222 Aryan
```

```
class Box {
  double width;
  double height;
  double depth;

  // This is the constructor for Box.
  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }

  // compute and return volume
  double volume() {

    return width * height * depth;
  }
}

class BoxDemo7 {
  public static void main(String args[]) {
    // declare, allocate, and initialize Box objects
    Box mybox1 = new Box(10, 20, 15);
    Box mybox2 = new Box(3, 6, 9);

    double vol;

    // get volume of first box
    vol = mybox1.volume();
    System.out.println("Volume is " + vol);

    // get volume of second box
    vol = mybox2.volume();
    System.out.println("Volume is " + vol);
  }
}
```

this keyword

this is a reference variable that refers to the current object.



Usage of Java this keyword

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

Example

Without using this keyword

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
rollno=rollno;
name=name;
fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis1{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output:

```
0 null 0.0
0 null 0.0
```

Using this keyword

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output:

```
111 ankit 5000.0
112 sumit 6000.0
```

If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program:

Program where this keyword is not required

```
class Student{
int rollno;
String name;
float fee;
Student(int r,String n,float f){
rollno=r;
name=n;
fee=f;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis3{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output:

```
111 ankit 5000.0
112 sumit 6000.0
```

Copy Constructor

There **is no copy constructor in Java**. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in Java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

By Constructor

```
class Student6{
  int id;
  String name;
  Student6(int i,String n){
  id = i;
```

```
    name = n;
    }
    //constructor to initialize another object
    Student6(Student6 s)
   {
    id = s.id;
    name =s.name;
    }
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    Student6 s1 = new Student6(111,"Karan");
    Student6 s2 = new Student6(s1);
    s1.display();
    s2.display();
   }
  }
```

```
111 Karan
111 Karan
```

Copying values without constructor

```
  class Student7{
    int id;
    String name;
    Student7(int i,String n){
    id = i;
    name = n;
    }
    Student7(){}
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    Student7 s1 = new Student7(111,"Karan");
    Student7 s2 = new Student7();
    s2.id=s1.id;
    s2.name=s1.name;
    s1.display();
```

```
111 Karan
111 Karan
```

```
s2.display();
 }
}
```

Difference between constructor and method

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

Constructor overloading in Java

- It is a technique of having more than one constructor with different parameter lists.
- They are arranged in a way that each constructor performs a different task.
- They are differentiated by the compiler by the number of parameters in the list and their types.

Example

```
class Student5{
int id;
String name;
int age;
//creating two arg constructor
Student5(int i,String n){
id = i;
name = n;
}
//creating three arg constructor
Student5(int i,String n,int a){
id = i;
```

```
111 Karan 0
222 Aryan 25
```

```
name = n;
age=a;
}
void display(){System.out.println(id+" "+name+" "+age);}
public static void main(String args[]){
Student5 s1 = new Student5(111,"Karan");
Student5 s2 = new Student5(222,"Aryan",25);
s1.display();
s2.display();
}
}
```

How Java Constructors are Different From Java Methods?

- Constructors must have the same name as the class within which it is defined it is not necessary for the method in Java.
- Constructors do not return any type while method(s) have the return type or **void** if does not return any value.
- Constructors are called only once at the time of Object creation while method(s) can be called any number of times.

Garbage collection

- garbage means unreferenced objects.
- Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using

- free() function in C language and
- delete() in C++.
- But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector (a part of JVM) so we don't need to make extra efforts.

How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By anonymous object etc.

By nulling a reference:

```
Employee e=new Employee();
e=null;
```

By assigning a reference to another:

```
Employee e1=new Employee();
Employee e2=new Employee();
e1=e2;//now the first object referred by e1 is available for garbage collection
```

By anonymous object:

```
new Employee();
```

finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing.

This method is defined in Object class as:

```
protected void finalize(){}
```

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

```
public static void gc(){}
```

Example

```
public class TestGarbage1{
 public void finalize(){System.out.println("object is garbage collected");}
 public static void main(String args[]){
  TestGarbage1 s1=new TestGarbage1();
  TestGarbage1 s2=new TestGarbage1();
  s1=null;
  s2=null;
  System.gc();  } }
```

```
object is garbage collected
object is garbage collected
```