Constructor

Constructor

- a constructor is a **block of codes similar to the method**.
- Its special type of method that is used to initialize the object.
- It is called/ initialized when an **object of the class is created**.
- At the time of **calling constructor, memory for the object is allocated** in the memory.

- It is called constructor because it **constructs the values at the time of object creation.**
- It is not necessary to write a constructor for a class. It is because **java compiler creates a default constructor if your class doesn't have any.**

Rules for defining Constructor

- Constructor name must be **same as the class name.**
- Constructor **should not have any return type.**
- Class can contain **more than one constructor.**
- constructor cannot be **abstract, static, final, and synchronized**
- Its defined with any access specifier.

Types

i) Default constructor

ii) Parameterized constructor

iii) Constructor overloading.

Default constructor

- A constructor is called "Default Constructor" when it doesn't have any parameter.
- A constructor doesn’t contain any parameter / arguments is called Default constructor

Example Program

```
public class vikram {

public vikram()
{
System.out.println("I am vikram ");
}
void diaplay()
{
System.out.println("I an b to cse dept.");
}
public static void main(String[] args) {
vikram o=new vikram();
o.diaplay();
}
}
```

Parameterized constructor

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

Example Program

```
public class paracons {
int roll_no;
String name;

public paracons()                                   //Default constructor
{
System.out.println("hello");
}
public paracons(int r, String n)                    //parameterized constructor
{
roll_no=r;
name=n;
}
```

```
void display()
{
System.out.println("The roll number is "+ roll_no);
System.out.println("The name of the person is " + name);
}

public static void main(String[] args)
 {
paracons o=new paracons();          // Calls Default
paracons o1=new paracons(7,"vikram");    // parameterized
                                                constructor
o.display();
o1.display();
}
```

Output

```
The roll number is 0
The name of the person is null
The roll number is 7
The name of the person is vikram
```

Constructor overloading

- The class contains more than one constructor then it is called as overloaded constructor.
- Same constructor name with different argument.

Example Program

```
public class paracons {
int roll_no; //0
String name;
String dept; //null
public paracons()                        //Default
{
System.out.println("hello");
}
public paracons(int r,String n)          //param. cont with 2 arg
{
roll_no=r;
name=n;
}
public paracons(int r, String n, String d)   / / para.cont 3 arg
{
roll_no=r;
name= n;
dept=d;     }
```

Overloading Constuctor

```
void display()
{
System.out.println("the roll number is "+ roll_no);
System.out.println("the name of the person is " + name);
System.out.println("department is " + dept);
}

public static void main(String[] args) {
paracons o=new paracons();                    // default call
paracons o1=new paracons(7,"vikram"); // call with 2 arguments
paracons o2=new paracons(10,“Raja","IT");   // call with 2 arguments
o.display();        // calls  default constructor display
 o1.display();      // calls for 2 argument  constructor display
o2.display();       //  calls for 3 argument  constructor display

}
```

output

```
hello
the roll number is 0
the name of the person is null
department is null

the roll number is 7
the name of the person is vikram
department is null

the roll number is 10
the name of the person is Raja
department is IT
```

Recursive

- in which a **method** calls itself to solve some problem. A **method** that uses this technique is **recursive**.

- Eg: Factorial.

Final

- Its used with variable, methods and classes.
- If a variable is declared as final It make that variable as constant.
- Its value cannot be changed In a program

Garbage Collection

- 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.

How can an object be unreferenced

- By nulling a reference:

Employee e=**new** Employee();

e=**null**;

- By assigning a reference to another:

Employee e2=**new** Employee();

e2=e;

gc() and finalize()

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

  Method name: protected void finalize(){}
- **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(){}

Program:

```
public class garbagetest {
public static void main(String[] args) {
garbagetest t1=new garbagetest();
garbagetest t2=new garbagetest();
System.out.println("hello");
t1= null;
t2=t1;
System.gc();
}
public void finalize()
{
System.out.println("garbage collected");
}
}
```