Method Overloading and constructors

```
class MyClass {
   int height;
   MyClass()
 {     System.out.println("bricks");
      height = 0;
   }
   MyClass(int i) {
   System.out.println("Building new House
that is "      + i + " feet tall");
      height = i;
   }
   void info() {
      System.out.println("House is " + height
+ " feet tall");
   }
   void info(String s) {
      System.out.println(s + ": House is "      +
height + " feet tall");
   }
}
```

```
public class samle {
   public static void main(String[] args)
   {
      MyClass t = new MyClass(0);
      t.info();
      t.info("overloaded method");
}
}
```

<u>Output:</u>

```
Building new House that is 0 feet tall
House is 0 feet tall
overloaded method: House is 0 feet tall
```

Note:

1.Functions only differ in return types cannot be overloaded.

Method Overloading and constructors

```
class Rectangle {
    double length;
    double breadth;

    void area(int length, int width) {
  int areaOfRectangle = length * width;
        System.out.println("Area of Rectangle : " + areaOfRectangle);
    }

    void area(double length, double width) {
  double areaOfRectangle = length * width;

 System.out.println("Area of Rectangle : " + areaOfRectangle);
    }

}
```

```
class RectangleDemo {
    public static void main(String args[]) {

        Rectangle r1 = new Rectangle();

        r1.area(10, 20);
        r1.area(10.50, 20.50);
    }
    }
```

Output:

```
Area of Rectangle : 200
Area of Rectangle : 215.25
```

Method Overloading and constructors

```
class Rectangle {
    double length;
    double breadth;

    void area(int length, int width) {
  int areaOfRectangle = length * width;
        System.out.println("Area of Rectangle : 
" + areaOfRectangle);
    }

   /* void area(double length, double width)
{
  double areaOfRectangle = length * width;

 System.out.println("Area of Rectangle : " +
areaOfRectangle);*/
    }

}
```

```
class RectangleDemo {
    public static void main(String args[]) {

        Rectangle r1 = new Rectangle();

        r1.area(10, 20);
        r1.area(10.50, 20.50); //exception

    }
    }
```

Method Overloading and constructors

```
class Rectangle {
    double length;
    double breadth;

  /*  void area(int length, int width) {
  int areaOfRectangle = length * width;
        System.out.println("Area of Rectangle :
" + areaOfRectangle);
    }*/

    void area(double length, double width)
{
  double areaOfRectangle = length * width;

 System.out.println("Area of Rectangle : " +
areaOfRectangle);
    }

}
```

```
class RectangleDemo {
    public static void main(String args[]) {

        Rectangle r1 = new Rectangle();

        r1.area(10, 20);//automatic type conversion
        r1.area(10.50, 20.50);

    }
    }
```

Output:

```
Area of Rectangle : 200.0
Area of Rectangle : 215.25
```

Using Objects as parameters

```
class Rectangle {
    int length;
    int width;
    Rectangle(int l, int b) {
        length = l;
        width = b;
    }

    void area(Rectangle r1) {
    int areaOfRectangle = r1.length * r1.width;
   System.out.println("Area of Rectangle : "
                               + areaOfRectangle);
    }
}

class t1 {
    public static void main(String args[]) {
        Rectangle r1 = new Rectangle(10, 20);
        r1.area(r1);
    }
}
```

Output:

```
Area of Rectangle : 200
```

Returning Objects

```
class Rectangle
 {
     int length;
     int breadth;
     Rectangle(int l,int b) {
      length = l;
      breadth = b;
     }
   Rectangle getRectangleObject() {
   Rectangle rect = new Rectangle(10,20);
      return rect;
     }
}
```

```
class f {
      public static void main(String args[]) {
        Rectangle ob1 = new Rectangle(40,50);
        Rectangle ob2;
        ob2 = ob1.getRectangleObject();
        System.out.println("ob1.length : " + ob1.length);
        System.out.println("ob1.breadth: " + ob1.breadth
        System.out.println("ob2.length : " + ob2.length);
        System.out.println("ob2.breadth: " + ob2.breadth
        }}
```

Output:

```
ob1.length : 40
 ob1.breadth: 50
ob2.length : 10
ob2.breadth: 20
```

Returning Objects

```
class Rectangle {
  int length;
  int width;
  Rectangle(int l, int b) {
    length = l;
    width = b;
  }
  Rectangle area()
 {
 Rectangle r1=new Rectangle(11,21);
  length=77;
    return r1;
  }
  void disp()
  {
  System.out.println(length +""+width);
  }}
```

```
class sample{
  public static void main(String args[]) {
  Rectangle r1 = new Rectangle(10, 20);
    Rectangle r2;
    r2=r1.area();
    r1.disp();
    r2.disp();
  }
}
```

Output:

```
77 20
11 21
```

Assigning values with in class

```
class Rectangle {
   int length=15;
   int width=22;
Rectangle(){}
   Rectangle(int l, int b) {
      length = l;
      width = b;
   }
   Rectangle area()
{
Rectangle r1=new Rectangle(); //no arg cons is
//compulsory
        return r1;
   }
   void disp()
   {
   System.out.println(length +" "+width);
   } }
```

```
class sample{
   public static void main(String args[]) {
   Rectangle r1 = new Rectangle();
      Rectangle r2;
      r2=r1.area();
      r1.disp();
      r2.disp();
   }
}
```

Output:

```
15 22
15 22
```

String functions

- Unlike most other computer languages, Java provides built-in support for
- *multithreaded programming.* A multithreaded program contains two or more
- parts that can run concurrently. Each part of such a program is called a *thread,*
- and each thread defines a separate path of execution. Thus, multithreading is a
- specialized form of multitasking.
- You are almost certainly acquainted with multitasking, because it is supported
- by virtually all modern operating systems. However, there are two distinct types
- of multitasking: process-based and thread-based. It is important to understand the
- difference between the two. For most readers, process-based multitasking is the more
- familiar form. A *process* is, in essence, a program that is executing. Thus, *process-based*
- multitasking is the feature that allows your computer to run two or more programs
- concurrently. For example, process-based multitasking enables you to run the Java

Static

- When a member is declared as static ,it can be accessed before any objects are created and without reference to any object.

- Both methods and variables can be declared as static.

- Instance variables declared as static are global for all objects.

- Methods declared as static can access
  - only static methods
  - Only static data
  - They can’t refer to ‘this’.

static

```
class teststat
{
static int a;
int b;
static void disp()
{
System.out.println(a+" ");//Accessing b is an error
f();//if f() non static -error
}
static void f()
{ }
}
```

```
public class test {
public static void main(String[] args) {
teststat r=new teststat();
r.disp();
}
}
```

Static block

```
class teststat
{
static int a=3;
static int b;
static void disp(int x)
{
System.out.println(x+" ");
System.out.println(a+" ");
System.out.println(b+" ");
}
static
{
b=a*4;
}
}
```

```
public class test {
public static void main(String[] args) {
teststat r=new teststat();
r.disp(10);
}
}
```

<u>Output:</u>

```
10
3
12
```

final

- Variable can be declared as final .
- It Prevents its contents from being modified.
- Intialize a final variable at the time of declaration.
- Eg.final int a=3;

Recursion

```
class teststat
{
public int myRecursiveMethod (int aVariable)
    {
      System.out.println(aVariable);
      aVariable–;
      if (aVariable == 0)
        return 0;
      return myRecursiveMethod(aVariable);
    }
}
```

```
public class test {
public static void main(String[] args) {
teststat r=new teststat();
r.myRecursiveMethod(10);
}}
```

Output:

```
10
9
8
7
6
5
4
3
2
1
```

Input

```
import java.util.*;
class Rect {
    double length;
    double breadth;
   void input()
   {
   Scanner s =new Scanner(System.in);
   length=s.nextInt();
   breadth=s.nextInt();
   s.close();
   }
    void disp() {
       System.out.println(length+" "+breadth);

    }
}
```

```
class input {
    public static void main(String args[]) {

        Rect r1 = new Rect();
r1.input();
r1.disp();
}
}
```

Output:

```
2
2
2.0
2.0
```

scannerobj.nextInt(), for integer

scannerobj.nextFloat() for float

scannerobj.nextDouble() for double

scannerobj.next()- for string

scannerobjnextLine() [for string with spaces]

Arrays

- Array within a class

- Array of objects

Eg. Student[] studentArray = new Student[7];

```
public static void main(String[] args) {
        Student[] studentArray = new Student[7];
        studentArray[0] = new Student();
        studentArray[0].marks = 99;
        System.out.println(studentArray[0].marks); // prints 99
        modify(studentArray[0]);
        System.out.println(studentArray[0].marks); // prints 100 and not 99
        // code
     }
     public static void modify(Student s) {
        s.marks = 100;
     }
```