

# Java Programing

**Java:** - Java is a Programming language (Heavy weight language) It is a universal language

- It was created in 1995 by “Sunmicro systems”
- Now it’s owned by “Oracle”

We are learning core java because our main intension is to learn a tool viz, selenium web driver which understood core java

**Java is used for:**

- Mobile Applications
- Desktop Applications
- Web Applications
- Web Servers and Application Servers
- Games
- Database Connection and much, much more

**We are using Java because**

- Java works on different platforms (windows, Mac, Linux etc)
- It is one of the most popular programming languages in the world
- It is easy to learn and simple to use
- It is open source and free
- It is secure, fast and powerful
- The OS by default can’t understood java we can write or type the java but we can’t compile and run the java in windows OS by default
- So, for that we need to install the (engine) java ‘JDK’ (Java Development)

**How to check Java is installed in our system?**

Ans: Go to run window (press window + R) button and type ‘cmd’ and click ok button, type java and press Enter button

**How to install Java JDK?**

- Go to Google and search for “Download Java JDK 11”
- Click on the Oracle link (Java is Owned by Oracle)
- Select the respective installer based on your OS and click on the installer to download it
- Accept the license agreement, register to Oracle website and download the “.exe”  
([jdk-11.0.19\\_windows-x64\\_bin.exe](#)) file
- Install it

**To check Java version:**

Go to command prompt and type command → java -version (java space -version) and press Enter

**Note:** Java is highly case sensitive

**JDK:** JDK is a Java Development Kit, it contains everything that will be required to develop and run Java application

**JRE:** JRE is a Java Runtime Environment, it helps in execution of java program, it contains everything to run java application which has already been compiled, it doesn't contain the code library required to develop java application

**JVM:** JVM is a Java Virtual Machine, it helps in compilation, JVM which works on top of your operating system to provide recommended environment to your compiled java code, JVM only works with bytecode hence you need to compile your java application so that it can be converted to bytecode format (also known as the class file)

- Java is a programming language hence it is basically a compiler, it can also act as interpreter
- Hence java is both compiler and interpreter
- .java (dot Java) is the extension for the java source file

### **To write Java in notepad:**

Open notepad -> type Java code -> save as <class name>.java and save as -> type should be "all files (\*)"

### **How to set system Variables**

#### **Step1:**

- Create Java\_Home variable
- Variable name: Java\_Home
- Variable value: c:\program files\Java\jdk-11

#### **Step2:** Update/edit path variable for JDK

- Variable name: path
- Variable value: c:\program files\Java\jdk-11\bin

### **How to get/navigate the "Environment Variables":**

- Type Window + E -> Windows explorer / Explorer
- In the left-hand side select "This PC" option and right click on "This PC" and click on properties
- Open settings page
- Click on Advanced system settings, it displays system properties window
- In system properties window click on "Environment Variables" button, it displays Environment Variable Window
- Navigate to system variable section then click on new button and type the field using Variable name and value
- Then click on "Path" and click on Edit and fill the fields (copy and paste Java-JDK-11>bin path)

To compile java program first we need to write program in notepad and save it any drive using new folder (Example)

## Creating Source File:

### Example1:

```
class Test
{
    Public static void main (String [] args)
    {
        System.out.println ("Welcome To Core Java");
    }
}
```

### Output:

Welcome To Core Java

## How to compile and run the Java Program:

### Ans:

C:\Users\Admin>cd\

C:\>D:

D:\>cd Example

D:\Example>javac Test.java

D:\Example>java Test

**Welcome To Core Java**

**or**

C:\Users\admin> cd C:\ Program files\Java\JDK-11\bin

C:\program files\Java\JDK-11\bin >javac G:\Example\Test.java

C:\program files\Java\JDK-11\bin >java \_cp G:\Example Test

**Welcome To Core Java**

Especially the .class file (byte code) which are created upon compilation are platform independent

**Class Path:** It is a process of making the compiler global by setting the compiler path in system environment variable

With the help of class path, we can directly compile and run the java programme from any directory

### Note:

- JDK, JRE and JVM are platform independent
- Java is a secured language because the class file which are created upon compilation are very difficult to decode it and the class file will never get affected by virus

### Assignment:

- **WAP to display five Fruit names**
- **WAP to display five Vegetable names**
- **WAP to display five City names**
- **WAP to display five Friends names**
- **WAP to display five River names**

**class:** It is an inbuilt keyword, it must be always in lower case, it indicates the definition of the class

or

It's a keyword used to define the class name in java code.

In java the class can be declared either with public OR default access modifier OR Abstract OR final keyword

**class <class name>:** It indicates name of the class and it will be user defined, as per java coding standard the first character of the class name should be in upper case

**Example:**

**Sample:** It is a name given to the class. The class name must start with capital letter. class name is an identifier in java; hence we cannot create a duplicate class within the same package.

**{ (open flower bracket):** It indicates begin of the block

**} (close flower bracket):** it indicates end of the block

Or

```
{  
}
```

block. Every artifact in java will have open and closed brackets.

**public:** public is an access specifier, if any members of the class are a public it can be accessible irrespective of any package

or

It is an access modifier in java. The public members can be accessed anywhere within the project.

**static:** It is a keyword to define the static members, In order to executes the static member object or instance doesn't require, all static members allocate memory only during the runtime these members are at class level

or

- It is a keyword to define the static members.
- Static members belong to the class hence we no need to create an object to access the static members.
- for static members, the memory allocation takes place at runtime.
- We use static members for better memory management.
- static members include:
  - (1) static variables
  - (2) static methods
  - (3) static block

**void:** Void keyword can be used only with methods (presided by method name) those methods never "Return" any value

**main:** main method is an entry point, it acts as thread, it is available in java.lang package (it is default package in core java)

or

- It is a built-in method inside java.lang package.
- It is a Thread in java. Hence it acts as an entry point in java

- It is mainly used to run the java program

**String args []:** It is a parameter to a main method; it's datatype is String so the parameter we can call it as array of String parameter

Or

**String:** It is a final built-in class & as well as datatype in java.

**args []:** it is a single dimensional array

**Ex: String args []:** args is an array which accepts String as a value.

main is a static method with public scope & it won't return any value. It accepts array of String object as an argument.

**System.out.println:** It is an output statement in core java

Or

**System:** it is a pre-defined final class in java. lang package.

**out:** it is a static object inside System class

**println OR print:** it is an overloaded method in out object. It is a stream object.

**Difference between println and print:**

#### println

- After printing the statement, it applies new line Character
- Example:
- ```
class Demo
{
    public static void main (String [] args)
    {
        System.out.println("Mango");
        System.out.println("Orange");
    }
}
```
- Output:
- ```
Mango
Orange
```

#### print

- \* After printing the statement it doesn't applies new line character
- Example:
- ```
class Demo
{
    public static void main (String [] args)
    {
        System.out.println("Mango");
        System.out.println("Orange");
    }
}
```
- Output:
- ```
MangoOrange
```

**Special cases:**

**Case1: Can we change the name of args?**

**Ans:** yes, if we give kk or any name instead of args it acceptable because it considers as name of the array

Example:

```

class Demo
{
    public static void main (String kk [])
    {
        System.out.println("Mango");
    }
}

```

**Output:**

Mango

**Case2: Can we change name of main method or can we write main method in upper case?**

**Ans:** No, this program compile successfully but it won't execute because of absence of main method and main should be in lower case

Example:

```

class Demo
{
    public static void Main (String kk [])
    {
        System.out.println("Mango");
    }
}

```

**Output:**

Error

**Case3: Should we need to maintain class name and file name as same?**

**Ans:** yes, if class name and file name is different, after successful compilation it generates .class file having the name similar to the class name

If there are hundreds of .java file having the different class name and file name once after compiling all the .java files it is a challenging for the programmer to match .class files vs .java files

**Case4: In a main method can we interchange public and static keywords?**

**Ans:** Yes, we can interchange public and static keywords in a main method because it won't affect to main method

Example:

```

class Demo
{
    static public void main (String kk [])
    {
        System.out.println("Mango");
    }
}

```

**Output:**

Mango

**Variables:** variables are used to store the value, the value stored in a variable can change during the runtime

**Or**

Variables are the placeholders to store the respective data, in java the variables are tightly coupled with datatypes, while declaring the datatypes we must mention the type

Example:

```
class Demo
{
    public static void main (String kk [])
    {
        int x=25;
        System.out.println(x);
        x=50;
        System.out.println(x);
        x=100;
        System.out.println(x);
    }
}
```

**Output:**

```
25
50
100
```

**Datatypes:** Based on the datatypes we can store the specific value \ respective value in a variable

**Below are the Datatypes available in java:**

**1) Number/integer/Numeric datatypes:**

- a) byte → 1 byte (1byte= 8bits and range is  $(2^8-1)$ )
- b) short → 2 bytes
- c) int → 4 bytes
- d) long → 8 bytes

**2) Decimal datatypes:**

- a) float → 4 bytes
- b) double → 8 bytes

**3) boolean** → 1 byte → it accepts true or false only

**4) char** → 2 bytes

**5) String**

**Note:** All the datatypes in java are primitive datatypes except String because String is a class/objective and it is a derived datatype

**Different ways of declaring the datatypes:**

**Case1: declare the datatypes first and later allocates the value**

Example:

```
a) int age;  
   age = 29;  
b) String name;  
   name = "Ranav";
```

### **case2: Declare and allocate the value at same time**

Example:

```
a) int age = 29;  
b) String name = "Ranav";
```

### **Case3: Declare all the types together and later add the value**

Example:

```
String name, city, state;  
Name = "Ranav";  
City = "Tumakuru";  
State = "Karnataka";
```

**Primitive Datatypes:** The datatypes which are non-object are called as primitive datatypes

Example: byte, short, int, long, float, double, boolean, char

**Note:** as java is oops (Object Oriented Programming) sometimes we can't use primitive datatypes, as it is especially in case of collection so we may need to convert the primitive type into object

### **How to convert primitive types into object?**

**Ans:** we can convert the primitive type datatypes into object using wrapper classes

### **Wrapper Classes:**

Wrapper classes are the pre-defined classes in java which are used to convert the primitive datatypes into object/collection

Example:	Primitive type	Object type
	-byte	-Byte
	-short	-Short
	-int	-Integer
	-long	-Long
	-float	-Float
	-double	-Double
	-boolean	-Boolean
	-char	-Character



## Operators Used in Java:

---

### 1) Arithmetic Operator:

- a) + → Addition
- b) - → Subtraction
- c) \* → Multiplication
- d) / → Division
- e) % → Modulus (Mod)

<Operand>	<Operator>	<Operand>	
10	+	10	
20	-	10	
30	*	25	
50	/	5	
12	%	2	

### 2) Relational/Comparison Operator:

- a) > → Greater Than
- b) >= → Greater Than Equal To
- c) < → Less Than
- d) <= → Less Than Equal To
- e) == → Equal To
- f) != → Not Equal To

The result of all Relational/Comparison Operator will be “**boolean value**”

### 3) Assignment Operator:

= → Assignment Operator

### 4) Concatenation Operator:

+ → Concatenation Operator

<Operand>	<Operator>	<Operand>
“XYZ”	+	25
20	+	“ABC”
“ABC”	+	“XYZ”

**Note:** in concatenation operator the any one operand must be a String

### 5) Logical Operator:

- a) && → “AND” Operator

A	B	A&&B
0	0	0
0	1	0
1	0	0
1	1	1

b) `||` → “OR” Operator

A	B	A    B
0	0	0
0	1	1
1	0	1
1	1	1

c) `!` → “NOT” Operator

A	!A
0	1
1	0

d) **Increment/Decrement Operator:**

`++` → Increment Operator

```
int i=10;  
i++; → i=11;
```

`--` → Decrement Operator

```
int i=10;  
i--; → i= 9;
```

e) **Arithmetic Assignment Operator:**

`+=` → Addition Assignment Operator

```
int i=10;  
i=i+5; → 15  
Or  
int i=10;  
i+=5; → 15
```

`--` → Subtraction Assignment Operator

```
int i=10;  
i=i-5; → 5  
or  
int i=10;  
i-=5; → 5
```

`*` → Multiplication Assignment Operator

```
int i=5;  
i=i*3; → 15  
or  
int i=5;  
i*=3; → 15
```

`/` → Division Operator

```
int i=50;  
i=i/5; → 10  
or
```

```
int i=50;  
i/=5;    →10
```

%--> Modulus Assignment Operator

```
int i=10;  
i=i%2;  →0  
or  
int i=10;  
i%=2;   →0
```

### **Command line arguments:**

-----  
By using command line arguments, we can provide input to our program

Or

It is a process of passing the parameters through command prompt with the help of main method arguments (args [])

Example:

```
class CommandLineArgumentsDemo  
{  
    public static void main(String args [])  
    {  
        String s1=args[0];  
        System.out.println(s1);  
        String s2=args[1];  
        System.out.println(s2);  
    }  
}
```

### **Command Prompt Query:**

G:\Java\_Notes\Example\CommandLineARguments>javac CommandLineArgumentsDemo.java

G:\Java\_Notes\Example\CommandLineARguments>java CommandLineArgumentsDemo Ranav Rakshi

### **Output:**

```
Ranav  
Rakshi
```

### **Case1: By using command line arguments perform addition of 2 integers**

```
class AdditionDemo  
{  
    public static void main (String args [])  
    {  
        int x=Integer.parseInt (args [0]);  
        int y=Integer.parseInt (args [1]);  
        int res=(x+y);  
        System.out.println(res);  
    }  
}
```

### Command Prompt Query:

```
G:\Java_Notes\Example\CommandLineARguments>javac  
AdditionCommandLineArgumentsUsingInt.java
```

```
G:\Java_Notes\Example\CommandLineARguments>java AdditionCommandLineArgumentsUsingInt 10  
50
```

### Output:

60

### How to get character as a input?

#### Example:

```
class CharDemo  
{  
    public static void main (String args [])  
    {  
        char ch = args [0]. charAt (0);  
        System.out.println(ch);  
    }  
}
```

### Output:

```
G:\Java_Notes\Example\CommandLineARguments>javac CharDemo.java
```

```
G:\Java_Notes\Example\CommandLineARguments>java CharDemo A
```

A

### Datatype Casting:

-----

It is a type of casting when you assign a value of one primitive datatype to another types

There are two types of casting

- 1) **Implicit or Widening casting:** This type of casting happens automatically in java, here we are storing or converting a smaller type to a larger type size

**Example:** byte→short→char→int→long→float→double

#### Rules for Implicit Casting:

- a) Both the datatypes should be compatible
- b) The source datatypes should be smaller than the destination datatypes

Example: byte num1 = 10;  
Short num2 = num1;

#### Source code:

```
class ImplicitAddition  
{  
    public static void main (String args [])  
    {  
        byte num1 = 10;
```

```

        short num2 = num1;
        System.out.println (num2);

        int num3 = 10000;
        long num4 = num3;
        System.out.println (num3);

        char ch = 'A';
        int num5 = ch;
        System.out.println (num5);
    }
}

```

**Output:**

```

10
10000
65

```

- 2) **Explicit or Narrowing Casting:** This type of casting we have to do manually by placing the datatype in parenthesis in front of the value

Example: double→float→long→int→char→short→byte

**Rules for Explicit Casting:**

- a) Both the datatypes should be compatible
- b) The source datatypes should be bigger than the destination datatypes

**Note:** Here we might see the data loss (not always)

**Source code:**

```

class ExplicitAddition
{
    Public static void main (String args [])
    {
        Short num1 =10;
        byte num2 =(byte) num1;
        System.out.println(num2);

        long num3 =10000;
        int num4 =(int) num3;
        System.out.println(num4);

        int num5 =65;
        char num6 =(char) num5;
        System.out.println(num6);
    }
}

```

**Output:** Here it shows an error when we run so I do manually by placing the type parenthesis in front the value i.e. (byte), (int), (char) ..etc

**Conditional Statements in Java:**

-----

To apply the validation, we need the conditional statements

There are two types of conditional statements are in java

- 1) if
- 2) switch

**Note:** The “if” statement can be written in 4 possible ways

- 1) if
- 2) if else / else if
- 3) ladder if
- 4) nested if

**1) if:** if is used to validate only the matching or true condition

**syntax:**

```
if (condition)
{
    <statements>;
}
```

**Example:**

**a) Verify the given number is even**

```
class EvenNumber
{
    public static void main (String args [])
    {
        int num = Integer.parseInt(args [0]);
        if (num%2==0)
        {
            System.out.println(num+" is a Even Number");
        }
    }
}
```

**Output:**

G:\Java\_Notes\Example\ConditionalStatement\IfStatement>javac EvenNumber.java

G:\Java\_Notes\Example\ConditionalStatement\IfStatement>java EvenNumber 2

**2 is a Even Number**

**b) Validate the given number is equal to 10?**

```
class IfStatements
{
    public static void main (String args [])
    {
        int num=10;
        if(num==10)
        {
            System.out.println("Number is 10");
        }
    }
}
```

**Output:**

Number is 10

**Note:** During optional validation we have to go for if statement

**2) If else:** It is used to validate both the matching/true as well as non-matching/false condition

**Syntax:**

```
If (condition)
{
    <statement>
}
else
{
    <non-matching statement>
}
```

**When to go for if else statement?**

**Ans:** To validate both the matching/true as well as non-matching/false condition

**Example:**

**a) Validate the person is eligible for voting or not?**

**Ans:**

```
class Voting
{
    public static void main (String args [])
    {
        Int age =Integer.parseInt(args [0]);
        If(age>=18)
        {
            System.out.println("Eligible for Voting");
        }
        else
        {
            System.out.println("Not Eligible for Voting");
        }
    }
}
```

a) **Input:** 22

**Output:** Eligible for Voting

b) **Input:** 15

**Output:** Not Eligible for Voting

**b) Validate the given city is Tumakuru or not?**

**Ans:**

```
package ifstatements
public class IfStatements
{
    public static void main (String args [])
    {
        String cityname = "Tumakuru";
        If (cityname.equals ("Tumakuru"))
```

```

        {
            System.out.println("Yes, cityname is Tumakuru");
        }
        else
        {
            System.out.println("No cityname is Something else");
        }
    }
}

```

a) **Input:** Tumakuru

**Output:** Yes, cityname is Tumakuru

b) **Input:** Bengaluru

**Output:** No, cityname is Something else

3) **Ladder if:** It is used to apply multiple validations for the single input

**Syntax:**

```

If (Condition)
{
    <statement>;
}
else if (condition)
{
    <statement>;
}
else if (condition)
{
    <statement>;
}
else
{
    <non-matching statement>;
}

```

**When to go for Ladder if?**

**Ans:** It is used to apply multiple possible validations for the single/multiple inputs

Example:

1) **Validate the student grade based on the marks?**

**Ans:**

```

class StudentMarks
{
    public static void main (String args [])
    {
        int marks =Integer.parseInt(args [0]);
        if ((marks<=100) && (marks>=70))
        {
            System.out.println("The Result is Distinction");
        }
        else if ((marks<70) && (marks>=60))
        {
            System.out.println("The Result is First Class");
        }
        else if ((marks<60) && (marks>=50))

```



```

        {
            System.out.println("The Result is Second Class");
        }
        else if ((marks<50) && (marks>=35))
        {
            System.out.println("The Result is Pass Class");
        }
        else if ((marks<35) && (marks>=0))
        {
            System.out.println("The Result has Failed");
        }
        else
        {
            System.out.println("Invalid Marks input");
        }
    }
}

```

**Output:**

- a) **Input:** 80  
**Result:** The Result is Distinction
- b) **Input:** 25  
**Result:** The Result has Failed

4) **Nested if:** It is used to apply validations for the values which are depending upon each other's

**Example:** State is depending on Country, Rose is depending on Flower etc.

**Syntax:**

```

If (condition)
{
    <statement>;
}
If (condition)
{
    <statement>;
}
else
{
    <non-matching statement>;
}

```

**When to go for Nested if?**

**Ans:** It is used to apply validations for the value which are depending on each other's

**Example:**

**Validate the flower and their depending category?**

```

class FlowerValidation
{
    public static void main (String args [])
    {
        String item = "Rose";
        String category = "Flower";
    }
}

```

```

        If (category.equals("Flower"))
        {
            System.out.println("Flower category");
        }
        If (item.equals("Rose"))
        {
            System.out.println("Queen of Flowers");
        }
        else if (item.equals("Tube Rose"))
        {
            System.out.println("King of Flowers");
        }
        else
        {
            System.out.println("Invalid item" + item + "Under flower Category");
        }
        }
        else
        {
            System.out.println("Invalid Category;" + Category);
        }
    }
}

```

#### Output:

- a) **Input:** Rose  
**Result:** Queen of Flowers
- b) **Input:** Tube Rose  
**Result:** King of Flowers

#### Assignments:

- 1) WAP to verify the given number is odd or not?
- 2) WAP to verify the given number is divisible by 4 or not?
- 3) WAP to verify largest number among the given two integer numbers?
- 4) WAP to verify the given number is positive or not?
- 5) WAP to verify the largest number among the Three given Numbers?
- 6) WAP to verify the given character is vowel or not?
- 7) WAP to display the number of digits in a given number?
- 8) WAP to verify the given number is divisible by 9 or not?
- 9) WAP, based on given integer value in between 1 to 7, it has to display appropriate weekday name?
- 10) WAP to display month name based on the given integer value in between 1 to 12?

#### switch Statement:

-----

If the given input is required to match to available cases, we have to prefer switch block

**switch statement can be written in two possible ways**

- a) switch and default
- b) nested switch

- 1) **switch and default:** To validate both matching and non-matching statements

#### Syntax:

```

Switch (expression)
{
Case <value1>:
    <statement>;
    break;
case <value2>:
    <statement>;
    break;
-----
-----
Case <valueN>:
    <statement>;
    break;
default:
    <default statement>;
}

```

### Example1:

```

class VowelDemo
{
    public static void main (String args [])
    {
        char ch = args [0].charAt(0);
        switch (ch)
        {
            Case 'A':
                System.out.println(ch+ "is a Vowel");
                break;
            Case 'E':
                System.out.println(ch+ "is a Vowel");
                break;
            Case 'I':
                System.out.println(ch+ "is a Vowel");
                break;
            Case 'O':
                System.out.println(ch+ "is a Vowel");
                break;

            Case 'U':
                System.out.println(ch+ "is a Vowel");
                break;
            default:
                System.out.println(ch+ "is not a Vowel");
        }
    }
}

```

### Output:

- a) **input:** A  
**Result:** A is a Vowel
- b) **input:** Y  
**Result:** Y is not a Vowel

**Note:** If multiple cases are performing the same task instead of writing as a separate case, you can include multiple cases in a single line

```

class VowelDemo
{
    public static void main (String args [])
    {
        char ch = args [0]. charAt (0);
        switch (ch)
        {
            case 'A', case 'E', case 'I', case 'O', case 'U':
                System.out.println(ch+ "is a vowel");
                break;
            default:
                System.out.println(ch+ "is not a Vowel");
        }
    }
}

```

**Output:**

a) **input:** A

**Result:** A is a Vowel

b) **input:** Y

**Result:** Y is not a Vowel

**Example2: Validate the colour and colour code**

```

class ColourValidation
{
    public static void main (String args [])
    {
        String colour = "Black";
        switch (colour)
        {
            Case "Red":
                System.out.println("DANGER");
                break;
            Case "Blue":
                System.out.println("OCEAN");
                break;
            Case "Black":
                System.out.println("Fundamental Colour");
                break;
            default:
                System.out.println("Invalid colour" + colour);
        }
    }
}

```

**Output:** Fundamental colour

2) **Nested switch:** To validate the statements with multiple values where one value is depending on another value

**Syntax:**

```

switch (expression)
{
    case <value1>:
        <statement1>;

```

```

        break;
    case <value2>:
        <statement>;
        break;
    -----
    -----
    default:
        <Non-matching statement>;
}

break;
default:
    <non-matching statement>;
}

```

### Example: Validate the item and category

```

class ValidationOfItemUsingNestedSwitch
{
    public static void main (String args [])
    {
        String category = "Flower";
        String item = "Tube Rose";
        switch (category)
        {
            case "Flower":
                switch (item)
                {
                    case "Rose":
                        System.out.println("Queen of Flowers");
                        break;
                    case "Tube Rose":
                        System.out.println("King of Flowers");
                        break;
                    default:
                        System.out.println("Invalid item" + item + "in Flower category");
                }
                break;
            default:
                System.out.println("Invalid category" + category);
        }
    }
}

```

**Output:** King of Flowers

### Assignment:

- 1) Based on the given number in between 1 to 7, it has to display the weekday name?
- 2) Based on the given number in between 1 to 12, it has to display the month name?

### Difference Between if and switch statements:

-----

- | <b>if</b>  |
|--|
| 1) It is a conditional statement                 |
| 2) Supports all logical and comparison operators |

- | <b>switch</b>  |
|--|
| 1) It is a conditional statement                     |
| 2) doesn't supports logical and comparison operators |

- 3) It is slow compare to 'switch' because it checks every statement is matching until it gets the false one
- 4) 'break' keyword is not required
- 5) 'else' is used to validate the non-matching statement.

Example:

```
if (condition)
{
    <statement>;
}
else
{
    <non-matching statement>;
}
```

- 3) it is fast compare to 'if' because it directly switch to the matching statement
- 4) 'break' keyword is required
- 5) 'default' is used to validate non-matching statement.

Example:

```
switch (expression)
{
    case (value):
        <statement>;
        break;
    default:
        <Invalid statement>;
}
```

**Note: What is the role of break statement in switch case?**

**Ans:** if the break statement is missing in switch case, in this case it continuing execution with upcoming case block.

**Example:**

```
int num =7;
switch (num)
{
    case 5:
        System.out.println("FIVE");

    case 7:
        System.out.println("SEVEN");
    default:
        System.out.println("Invalid Number");
    case 3:
        System.out.println("THREE");
    case 9:
        System.out.println("NINE");
}
```

**Output:**

```
SEVEN
Invalid Number
THREE
NINE
```

**Note:** Here it checks the matching condition and write matching and below all the conditions because of we didn't provide "break" point.

\*\*\* "if" is better than "switch" because "if" is used everywhere and there is no limitations or drawback

**Looping Statement:**

-----

If user wants to perform/executes any statements or activities repeatedly then go for looping statements.

Example: 1-10, 1-100,

**There are “Four” looping statements are available:**

- a) for loop
- b) for each or enhanced loop
- c) while loop
- d) do while loop

### **Case1: for loop**

~~~~~

If you know the exact count in this case we have to prefer for loop, it is also called as finite (exact) looping statement.

### **Syntax:**

```
for (Initialization; condition; increment/decrement)
{
    <statement>;
}
```

### **How “for loop” works?**

**Ans:** It checks the condition first, if the condition is true then it will enter the loop block and it will run the loop block until the condition become false.

### **Example:**

```
class LoopingDemo
{
    public static void main (String args [])
    {
        for (int i=10; i<=20; i++)
        {
            System.out.println(i);
        }
    }
}
```

### **Output:**

```
10
11
12
13
14
15
16
17
18
19
20
```

### **How for loop works?**

**Ans:**

```
class LoopingDemo
{
    public static void main (String args [])
    {
        int i=10;
        for (; i<20;)
        {
            System.out.println(i);
            i++;
        }
    }
}
```

```

    }
}

```

**Note:** Here initialization takes place first then it checks the condition and it takes place every time later increment or decrement takes place

### WAP to display numbers in reverse order?

```

class LoopingReverse
{
    public static void main (String args [])
    {
        for (int i=10; i>=1; i--)
        {
            System.out.println(i);
        }
    }
}

```

### Output:

```

10
9
8
7
6
5
4
3
2
1

```

### WAP to print numbers from 100 to 50?

```

class LoopingReverse1
{
    public static void main (String args [])
    {
        for (int i=100; i>=50; i--)
        {
            System.out.println(i);
        }
    }
}

```

### WAP to display square result of first 10 numbers?

```

class LoopingSquare
{
    public static void main (String args [])
    {
        for (int i=1; i<=10; i++)
        {
            int b=i*i;
            System.out.println(b);
        }
    }
}

```



### WAP to display number from 0 to 10 show only even numbers?

```
class LoopingEven
{
    public static void main (String args [])
    {
        for (int i=0; i<=10; i=i+2)
        {
            System.out.println(i);
        }
    }
}
```

#### Output:

```
0
2
4
6
8
10
```

**Note:** This is the one way to find even/odd numbers but modulus/mod (%) is the right way to validate even/odd numbers

```
class LoopingEven
{
    public static void main (String args [])
    {
        for (int i=0; i<=10; i++)
        {
            if ((i%2) == 0)
                System.out.println(i);
        }
    }
}
```

### WAP to display numbers from 0 to 5 show only odd numbers?

```
class LoopingOdd
{
    public static void main (String args [])
    {
        for (int i=0; i<=10; i++)
        {
            if ((i%2) != 0)
                System.out.println(i);
        }
    }
}
```

#### Nested for loop:

##### Syntax:

```
for (Initialization; condition; increment/decrement)
{
    for (Initialization; condition; increment/decrement)
    {
```

```

        <statement>;
    }
    <statement>;
}

```

### How Nested for loop works?

**Ans:** For every outer for loop iteration the complete inner for loop will get execute

**Example: print the pattern**

```

***
***
***
***
***

```

**Ans:**

```

class NestedLoopDemo
{
    Public static void main (String args [])
    {
        for (int i=1; i<=5; i++)
        {
            for (int j=1; j<=3; j++)
            {
                System.out.println("* ");
            }
            System.out.println();
        }
    }
}

```

**Output:**

```

***
***
***
***
***

```

**Note:** Always remember outer for loop is for rows and inner for loop is for columns

**Interview Purpose:**

```

int I;
for (i=1; i<=5; i++)
{
    System.out.println(i);
}
System.out.println("i value outside for loop:" + i);

```

**Output:**

I value outside for loop: 6

**Note:** in this code case it run first I value from 1 to 5, then it will enter outside I value is greater than 5 condition will match, so it will print I value 6 and exit

\* if I declare I inside for loop it will show compilation error because we created one outside loop so it will throw error

```

for (int i=1; i<=5; i++)
{
    System.out.println(i);
}

```

```

    }
    System.out.println("I value outside for loop:" + i);

```

**Output:**

Compilation error

**Case2: for each or enhanced loop**

**Syntax:**

```

    for (<datatype><variable>:<array/object/collection>)
    {
        <statement>;
    }

```

**How for each works?**

**Ans:** it first checks the right-side value that is should be either array/object/collection, if right side is matching it will pick one by one value and store it in the variable

**Note:** we call for each loop so smart because we don't mention the size of array/object/collection, when we write it is a array/object/collection it enters and read the array/object/collection then write it's size in variable so we called it is so smart/advanced/enhanced loop

**Example:**

```

    Int arr [] = {10,20,30,40,50};
    for (int x: arr)
    {
        System.out.println(x);
    }

```

**Note:** This is for single dimensional array

**Nested for each loop:**

**Syntax:**

```

    for (<datatype><variable> []: <array/object/collection>)
    {
        for (<datatype><variable> []: <variable> [])
        {
            <statement>;
        }
    }

```

**Note:** Here we mention "<variable> []" because it is 2-dimensional array so variable [] with array we write like "<variable> []"

**Example:**

```

    Int arr [] [] = {{ 10,20}, {30,40}, {50,60}};
    for (int x []: arr)
    {
        for (int x1: x)
        {
            System.out.print(x1 + " ");
        }
        System.out.println();
    }

```

**Output:**

10 20  
30 40  
50 60

**Finite loops:** These loops will have definite number of iterations

Example: for loop, for each loop

**Infinite loops:** These loops will not have definite number of iterations, these loops run infinitely

Example: while loop, do while loop

**Case3: while loop:** it is an infinite loop, first it verifies the condition if the condition is true it enters into the body of the while loop after executing the whole body again it verifies the condition, if the condition is true, it executes the body of the loop again in this way it executes till condition become false.

**Syntax:**

```
initialization;
while (condition)
{
    <statement>;
    (increment/decrement);
}
```

**How while loops work?**

**Ans:** it checks the condition first, if the condition is true then it will enter and execute the block until it gets the false condition

**Example:**

**WAP to print the numbers from 10 to 20?**

```
class WhileLoopDemo
{
    public static void main (String args [])
    {
        int i=10;
        while (i<=20)
        {
            System.out.println(i);
            i++;
        }
    }
}
```

**Output:**

10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20

### WAP to display the numbers 10 to 1?

```
class WhileReverse
{
    public static void main (String args [])
    {
        int i=10;
        while (i>=1)
        {
            System.out.println(i);
            i--;
        }
    }
}
```

**Case4: do while loop:** irrespective of condition it executes the body of the loop at least one time

#### Syntax:

```
initialization;
do
{
    <statement>;
    <increment/decrement>;
}
while (condition);
```

#### How do while loop works?

**Ans:** It will execute and enter the block without checking the condition, once it enters the block then it checks the condition, if the condition is true, it will continue otherwise it will stop

#### Example:

### WAP to print the numbers from 20 to 30?

```
class DoWhileLoopDemo
{
    public static void main (String args [])
    {
        int i=20;
        do
        {
            System.out.println(i);
            i++;
        }
        while (i<=30);
    }
}
```

#### Output:

```
20
21
22
23
24
25
26
27
28
```

**WAP to display numbers from 10 to 1?**

```
class DoWhileLoopReverse
{
    public static void main (String args [])
    {
        int i=10;
        do
        {
            System.out.println(i);
            i--;
        }
        while (i>=1);
    }
}
```

**WAP to display alphabets from A to Z?**

```
class DoWhileAlphabets
{
    public static void main (String args [])
    {
        int i=0;
        do
        {
            int val =(i+65);
            char ch = (char) val;
            System.out.println(ch);
            i++;
        }
        while (i<=26);
    }
}
```

Or

```
class DoWhileAlphabets
{
    public static void main (String args [])
    {
        int i=0;
        do
        {
            System.out.println((char) (i+65));
            i++;
        }
        while (i<=26);
    }
}
```

**Assignment:**

**The below assignments should be executes in for, while and do while loops**

**1) WAP to display the complete 10<sup>th</sup> table?**

- 2) WAP to display the complete 12<sup>th</sup> table in a reverse order?
- 3) WAP to display even numbers from 40 to 80?
- 4) WAP to display odd numbers from 91 to 61?
- 5) WAP to display numbers in between 100 to 1 which are divisible by 9?
- 6) WAP to display only the result of 5<sup>th</sup> table in reverse order?
- 7) WAP to display the cube of numbers from 20 to 1?
- 8) WAP to display odd numbers in between 200 to 100?
- 9) WAP to display the complete 3<sup>rd</sup> table from 1 to 20?

**Break:** A break statement terminates a loop or a block

**Example:**

```
class BreakDemo
{
    Public static void main (String args [])
    {
        int i=1;
        while (true)
        {
            System.out.println(i);
            if(i==15)
            {
                break;
            }
            i++;
        }
    }
}
```

**WAP it should run 10 times and once it reaches 5<sup>th</sup> iteration the execution get stop?**

```
class BreakDemo1
{
    public static void main (String args [])
    {
        for (int i=1; i<=10; i++)
        {
            System.out.println ("Inside the loop:" + i);
            if (i==5)
            {
                System.out.println("Condition matched...");
                break;
            }
            System.out.println ("*****");
        }
    }
}
```

**Output:**

```
Inside the loop: 1
*****
Inside the loop: 2
*****
-
-
Inside the loop: 5
```

Condition matched...  
\*\*\*\*\*

**Continue:** Continue statement skip or terminates iterations

**Example:**

```
class ContinueDemo
{
    public static void main (String args [])
    {
        for (int i=10; i<=30; i++)
        {
            if (i%2==0)
            {
                continue;
            }
            System.out.println(i);
        }
    }
}
```

**WAP it should run 10 times and once it reaches 5<sup>th</sup> iteration, skip the 5<sup>th</sup> and continue with complete iterations?**

```
class ContinueDemo1
{
    public static void main (String args [])
    {
        for (int i=1; i<=10; i++)
        {
            if (i==5)
            {
                System.out.println("Condition matched...");
                continue;
            }
            System.out.println ("Inside the loop:" + i);
            System.out.println ("*****");
        }
    }
}
```

**Output:**

```
Inside the loop: 1
*****
Inside the loop: 2
*****
-
-
Condition matched...
*****
Inside the loop: 6
*****
Inside the loop: 7
*****
-
-
```



Inside the loop: 10  
\*\*\*\*\*

**Note:** if we are using “if” statement within the body of any loop we shouldn’t perform increment or decrement operations within the body of the statement

**WAP to find the count of numbers in between 100 to 20 which are divisible by 5?**

**Step1:** make sure can you print numbers in between 100 to 20

**Step2:** from step1, try to print numbers which are divisible by 5

**Step3:** from step3, apply logic to get count of numbers

```
class ForLoopCountNumbersDemo
{
    public static void main (String args [])
    {
        int count =0;
        for (int i=100; i>=20; i--)
        {
            if (i%5==0)
            {
                count++;
            }
        }
        System.out.println(count);
    }
}
```

**Count:**  
17

**WAP to find the sum of numbers in between 1 to 10?**

```
class ForLoopSumOfNumbers
{
    public static void main (String args [])
    {
        int sum=0;
        for (int i=1; i<=10; i++)
        {
            sum = sum+i;
        }
        System.out.println(sum);
    }
}
```

**Output:**  
55

**Note:** To terminate infinitely running loop in command prompt use “ctrl + c”

- break and continue are loop

## Difference between break and continue:

### break

- a) It is a keyword; it is used in switch and looping statements
- b) It will stop the whole loop/block completely
- c) It must be the last statement in the block, otherwise, it will throw the unreachable code error.

### continue

- a) It is a keyword; it is used in looping statement only
- b) It stops particular iterations and continue with remaining iterations
- c) It must be the last statement in the block, otherwise, it will throw unreachable code error.

## Assignment:

Below assignment are done by for, while and do while loops

- 1) WAP to find the count of odd numbers in between 250 to 500?
- 2) WAP to find the factorial of a given number?
- 3) WAP to find the numbers in between 200 to 100 which are divisible by 7?
- 4) WAP to find the square of sum of first 20 numbers?
- 5) WAP to find cube of sum of numbers in between 10 to 20?

## Patterns:

-----

WAP to display the pattern

```
* * * * *
* * * * *
* * * * *
```

Ans: Using for loop:

```
class PatternsDemo
{
    public static void main (String args [])
    {
        for (int i=1; i<=3; i++)
        {
            for (int j=1; j<=5; j++)
            {
                System.out.print("* ");
            }
            System.out.println();
        }
    }
}
```

Using while loop:

```
class PatternsDemo
{
    public static void main (String args [])
    {
        int i=1;
        while(i<=3)
        {
            int j=1;
            while(j<=5)
```

```

        {
            System.out.print("* ");
            j++;
        }
        System.out.println();
        i++;
    }
}

```

### Using dowhile loop:

```

class PatternsDemo
{
    public static void main (String args [])
    {
        int i=1;
        do
        {
            int j=1;
            do
            {
                System.out.print("* ");
                j++;
            }
            while(j<=5)
            System.out.println();
            i++;
        }
        while(i<=3)
    }
}

```

### Output:

```

* * * * *
* * * * *
* * * * *

```

### Assignments:

The below assignments should be resolve in for, while and dowhile loops

- |               |                |       |                |       |
|---------------|----------------|-------|----------------|-------|
| 1) #####      | 2) 88888       | 3) 3  | 4) 1           | 5) *  |
| ####          | 8888           | 33    | 2 3            | **    |
| ###           | 888            | 333   | 4 5 6          | ***   |
| ##            | 88             | 3333  | 7 8 9 10       | ****  |
| #             | 8              | 33333 | 11 12 13 14 15 | ***** |
| 6) 1          | 7) 4           | 8) 1  | 9) 55555       | 10) * |
| 3 5           | 8 12           | 2 2   | 4444           | **    |
| 7 9 11        | 16 20 24       | 333   | 333            | ***   |
| 13 15 17 19   | 28 32 36 40    | 4444  | 22             | ****  |
| 21 3 25 27 29 | 44 48 52 56 60 | 55555 | 1              | ***** |

a) 1                      b) \* \* \* \* \*

2 2                      \*                      \*

3 3 3                    \*                      \*

4 4 4 4                \*                      \*

5 5 5                    \* \* \* \* \*

6 6

7

**IDE's:** Integrated Development Environment are used to write and run the java code

There are two types of IDE's:

- a) Eclipse
- b) IntelliJ

### Installation of Eclipse:

- a) Go to google and search "Download Eclipse"
- b) Click on the first link: <https://www.eclipse.org/downloads/>
- c) Click on download packages and select "Eclipse IDE for Enterprise Java and Web Developers"
- d) Click on "Windows x86\_64"
- e) Download and install the installer

### Eclipse folder structure:

- a) Install Eclipse
- b) Create a workspace (a empty folder) in any drive
- c) Open Eclipse folder and browse and launch the Eclipse with the workspace
- d) Once Eclipse opens for the workspace, we need to create the project in the eclipse  
to create project in Eclipse we need to follow some steps:  
File → New → Project → JavaProject → give the name to the project (without space) → Finish  
Project names should like for practice:  
DemoJavaProject/ SampleJavaProject/ PracticeJavaProject/ TestJavaProject....  
SampleJavaPrograms/ DemoJavaPrograms/ SamplePracticePrograms....

**Note:** Project will have

- 1) src
- 2) JRE system libraries

- e) Inside the Project right click on src → New → package. give package name (all small letters without space)  
like domainname.organizationname.projectname  
Example: - com.sgtesting.programs  
              - com.sgtesting.forloopassignment
- f) inside the Project right click on package → New → class. give class name (first letter should be in  
uppercase remain all are small and without space)  
Example: Test, Programs, Example

### Arrays:

-----

Arrays are used to store the elements based on the index, here the index always starts with zero, an array can store size-1 elements

Suppose the array has five elements to store it stores elements from the index 0 to 4 (size-1)

Or

Java Array is an object which contains elements of a similar data types or An array is a container object that holds a fixed number of values of a single datatype

## There are 2 types of arrays:

- a) single dimensional array
- b) two-dimensional array or multi-dimensional array

### Note:

- java doesn't support dynamic array
- array always stores homogeneous values means single type of values
- array stores the values based on index, the index always starts from zero, the last index of the array will be always size-1.
- To find the size of the array we can use:  
`<array name>.length` for single dimensional array
- To find the size of the two-dimensional array we can use:  
For rows: `<array name>.length;`  
For column: `<array name>[row index].length;`
- Array can store both primitive and object
- Data manipulation via... adding values, removal, modification is not possible in array at runtime because java supports only static/ fixed array

### Single dimensional array:

-----

single dimensional array represents storing the elements in the memory in the form of single column and multiple rows.

**Note:** in core java arrays are fixed in nature

**We can declare single dimensional array as follows:**

**Case1: Declare single dimensional array, assign elements into single dimensional array and read the elements from single dimensional array**

```
package com.sgtesting.arraydemo;

public class SingleDimensionalArrayCase1
{
    Public static void main (String args [])
    {
        // declare an array
        int arr [] = new int [4];

        // assign elements into an array
        arr [0] =10;
        arr [1] =20;
        arr [2] =30;
        arr [3] =40;

        // read the elements from array
        System.out.println(arr [0]);
        System.out.println(arr [1]);
        System.out.println(arr [2]);
        System.out.println(arr [3]);
    }
}
```

**Output:**

```
10
20
30
40
```

### Case2: apply for loop to read the elements from single dimensional array

```
package com.sgtesting.arraydemo;

public class SingleDimensionalArrayCase2
{
    Public static void main (String args [])
    {
        // declare an array
        int arr [] = new int [4];

        // assign elements into an array
        arr [0] =10;
        arr [1] =20;
        arr [2] =30;
        arr [3] =40;

        // read the elements from array using for loop
        for (int i=0; i<arr.length; i++)
        {
            System.out.println(arr [i]);
        }
    }
}
```

#### Output:

```
10
20
30
40
```

### Case3: Assign elements into an array in the declare step itself

```
package com.sgtesting.arraydemo;

public class SingleDimensionalArrayCase3
{
    Public static void main (String args [])
    {
        // declare an array and assign elements in to an array in declare step
        int arr [] = { 100,200,300,400,500};

        // read the elements from array
        for (int i=0; i<arr.length; i++)
        {
            System.out.println(arr [i]);
        }
    }
}
```

#### Output:

```
100
200
300
400
500
```

**Case4: read elements from an array in reverse order**

```

package com.sgtesting.arraydemo;

public class SingleDimensionalArrayCase3
{
    Public static void main (String args [])
    {
        // declare an array and assign elements in to an array in declare step
        int arr [] = { 100,200,300,400,500};

        // read the elements from array
        for (int i=arr.length-1; i>=0; i--)
        {
            System.out.println(arr [i]);
        }
    }
}

```

**Output:**

```

100
200
300
400
500

```

**Case5: read the elements from an array using for each loop**

```

package com.sgtesting.arraydemo;

public class SingleDimensionalArrayCase3
{
    Public static void main (String args [])
    {
        // declare an array and assign elements in to an array in declare step
        int arr [] = { 100,200,300,400,500};

        // read the elements from array using for each loop
        for (int kk: arr [])
        {
            System.out.println(kk);
        }
    }
}

```

**Output:**

```

100
200
300
400
500

```

**Note:** in for each loop, we can't do reverse order

**Assignment:**

- 1) WAP to read the elements from the byte array in reverse order?
- 2) WAP to create the character array and read the elements from character array using while loop?
- 3) WAP to create the double array and read the elements from double array in reverse order using

do while loop?

- 4) WAP to create a string array and read the elements from the string array in reverse order using While loop?
- 5) WAP to create a string array and read first half of the elements?
- 6) WAP to create character array and read second half of the elements from character array?
- 7) WAP to create a float array and read second half of the elements using while loop?
- 8) WAP to create boolean array and read the elements from boolean array using for loop?
- 9) WAP to create long array and read second half of the elements by applying do while loop?
- 10) WAP to create short array and read elements from the short array using do while loop?
- 11) There is an integer array it has some elements, WAP to find the sum of all elements?

Programmatically assign odd numbers in between 20 to 40 into a single dimensional array and read the elements from the array in reverse order?

Ans:

```
package com.sgtesting.arraydemo;

public class OddNumbersReverseDemo
{
    Public static void main (String args [])
    {
        // step1: make sure can we display numbers in between 20 to 40
        // step2: from step1; among numbers 20 to 40 display only odd numbers
        // step3: from step2; find the count of odd numbers in between 20 to 40

        int count=0;
        for (int i=20; i<=40; i++)
        {
            if (i%2==1)
            {
                count++;
            }
        }
        System.out.println("# of elements:" + count);

        // step4: create/declare an array with appropriate size
        int a [] = new int [count];

        // step5: assign odd numbers in between 20 to 40 into array
        int k=0;
        for (int j=20; j<40; j++)
        {
            if (i%2==0)
            {
                a [k] = j;
                k++;
            }
        }

        // step6: read the elements from array in reverse order
        for (int m=a.length-1; m>=0; m--)
        {
            System.out.println(a [m]);
        }
    }
}
```

Output:



39  
37  
35  
33  
31  
29  
27  
25  
23  
21

### Assignments:

- 1) Programmatically assign numbers in between 100 to 50 which are divisible by 4 into 1D array and read the elements from 1D array?
- 2) Programmatically assign numbers, the result of 7<sup>th</sup> table into 1D array in reverse order and read the elements from the array in reverse order?
- 3) Programmatically assign numbers in between 200 to 100 which are divisible by 9 into 1D array and read second half of the elements

### Two-Dimensional Array:

Two-dimensional array represents matrix, it has rows and columns, the row size always starts with zero (0) and column size always starts with zero (0).

Example: `int a [] [] = new int [2] [3];`

|   | 0       | 1       | 2       |
|---|---------|---------|---------|
| 0 | a[0][0] | a[0][1] | a[0][2] |
| 1 | a[1][0] | a[1][1] | a[1][2] |

### Case1: How to declare 2D array, how to assign elements into 2D array, How to read elements from 2D array?

```
package com.sgtesting.arraydemo;
public class TwoDimensionalArrayCase1
{
    public static void main (String args [])
    {
        // Declare/create an 2D array
        int a [] [] = new int [2][3];
        a[0][0] = 100;
        a[0][1] = 200;
        a[0][2] = 300;
        a[1][0] = 400;
        a[1][1] = 500;
        a[1][2] = 600;

        // read elements from 2D array
        System.out.println(a[0][0]);
        System.out.println(a[0][1]);
        System.out.println(a[0][2]);
        System.out.println(a[1][0]);
        System.out.println(a[1][1]);
        System.out.println(a[1][2]);
    }
}
```

```
}
```

**Output:**

```
100
200
300
400
500
600
```

**Case2: How to read elements from 2D array by using looping statements**

```
package com.sgtesting.arraydemo;
public class TwoDimensionalArrayCase2
{
    public static void main (String args [])
    {
        // Declare/create an 2D array
        int a [] [] = new int [2][3];
        a[0][0] = 100;
        a[0][1] = 200;
        a[0][2] = 300;
        a[1][0] = 400;
        a[1][1] = 500;
        a[1][2] = 600;

        // read elements from 2D array using loops
        for (int i=0; i<a.length;i++)
        {
            for (int j=0; j<a[i].length; j++)
            {
                System.out.print(a[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

**Output:**

```
100 200 300
400 500 600
```

**Case3: Read elements from 2D array in reverse order**

```
package com.sgtesting.arraydemo;
public class TwoDimensionalArrayCase2
{
    public static void main (String args [])
    {
        // Declare/create an 2D array
        int a [] [] = new int [2][3];
        a[0][0] = 100;
        a[0][1] = 200;
        a[0][2] = 300;
        a[1][0] = 400;
        a[1][1] = 500;
        a[1][2] = 600;
```

```

        // read elements from 2D array using loops in reverse order
        for (int i= a.length-1; i>=0;i--)
        {
            for (int j= a[i].length; j>=0; j--)
            {
                System.out.print(a[i][j] + " ");
            }
            System.out.println();
        }
    }
}

```

**Output:**

```

600 500 400
300 200 100

```

#### **Case4: Assign elements into 2D array in the declaration step itself**

```

package com.sgtesting.arraydemo;
public class TwoDimensionalArrayCase2
{
    public static void main (String args [])
    {
        // Declare/create an 2D array in declaration step itself
        int a [] [] = {{ 10,20,30},{40,50,60},{70,80,90} };

        // read elements from 2D array
        for (int i=0; i<a.length;i++)
        {
            for (int j=0; j<a[i].length; j++)
            {
                System.out.print(a[i][j] + " ");
            }
            System.out.println();
        }
    }
}

```

**Output:**

```

10 20 30
40 50 60
70 80 90

```

#### **Case5: Read the elements from 2D array using for each loop**

```

package com.sgtesting.arraydemo;
public class TwoDimensionalArrayCase2
{
    public static void main (String args [])
    {
        // Declare/create an 2D array in declaration step itself
        int a [] [] = {{ 10,20,30},{40,50,60} };

        // read elements from 2D array using for each loop
        for (int b[] = a)
        {
            for (int k = b)

```

```

        {
            System.out.print(k + " ");
        }
        System.out.println();
    }
}

```

**Output:**

```

10 20 30
40 50 60

```

## 2D array Examples:

### 1) WAP to read elements from string 2D 3\*3 array

```

package com.sgtesting.twodarrayassignment1;
public class String2DimArray
{
    public static void main (String [] args)
    {
        String a[][]= {{ "Ranav","Rakshi","Ravi"}, {"Thilak","Shine","Thili"}, {"Praveen","Raya","Sille"} };

        for (int i=0; i<a.length; i++)
        {
            for (int j=0; j<a[i].length; j++)
            {
                System.out.print(a[i][j] + " ");
            }
            System.out.println();
        }
    }
}

```

**Output:**

```

Ranav Rakshi Ravi
Thilak Shine Thili
Praveen Raya Sille

```

### 2) WAP to read elements from 2D 4\*3 Character array?

```

package com.sgtesting.twodarrayassignment1;
public class Character2DimArray
{
    public static void main (String [] args)
    {
        char ch[][]= {{ 'A','B','C'}, {'D','E','F'}, {'G','H','T'}, {'J','K','L'} };

        for (int i=0; i<ch.length; i++)
        {
            For (int j=0; j<ch[i].length; j++)
            {
                System.out.print(ch[i][j] + " ");
            }
            System.out.println();
        }
    }
}

```

**Output:**

```
A B C
D E F
G H I
J K L
```

**Assignments:**

- 1) Create 3\*3 2D double array read the elements from the array in reverse order?
- 2) Create 3\*3 2D byte array read the elements from the array using while loop?
- 3) Create integer 3\*3 2D array read only first row of the elements?
- 4) There is a 3\*3 string 2D array, WAP to concatenate all the elements?
- 5) Create 3\*3 short 2D array, WAP to read the last row of the elements alone?
- 6) Create 3\*3 Character 2D array, read first column of the elements
- 7) Create 2\*3 Character 2D array read the elements using do while loop?
- 8) Create 3\*3 string 2D array, WAP to read 2<sup>nd</sup> row of the elements?
- 9) Create 3\*3 long 2D array, display the sum of all the elements?
- 10) Create 3\*3 integer 2D array apply modulus operator on each element and display result in matrix format?
- 11) There is an integer 3\*3 2D array, WAP to transpose the elements (row become column)?

**Can we store values of different datatypes into array?**

**Ans:** yes, in some condition using object

Because object is a superclass of java it can store any datatype into array

**Example:**

```
object arr [] = new object ();
arr [0] =125;
arr [1] = "Ranav";
arr [2] = true;
arr [3] = 12.40;

System.out.println(Arrays.ToString(arr));
```

**Output:**

```
[125, Ranav, true, 12.40]
```

## Part2:

### Oops concept:

#### class in Java:

##### What is class?

**Ans:** class is a template by referring those templates we can create an object or instance, a class can contain variables, methods and constructors these are collectively called as members of the class

or

class is a container/blue print of object which describes states and behavior that its object support

class contains states and behavior:

- a) states mean variables
- b) behavior mean methods

example: \* if class is an Animal, then

- states include height, width, color etc...
- behavior include Running, hunting, jumping etc...

\* if class is a Human, then

- states include Name, job, age etc...
- behavior include Sleeping, Eating, Working etc...

**Object:** It is an instance created to the class, which describes states and behavior that of its object supports

##### Why object is created to the class?

**Ans:** in order to access the class member, we need to create a object to the class

**Note:** for the class object is like a “door”.

##### How to create an object to the class?

**Ans:**

<class name> <object name> = new <constructor name/class name> ();

Example:

```
class Sample
{
    String = "Ranav";
}

Sample sam = new Sample ();
```

here;

**Sample** = it is a class, (class name)

**Sam** = it is a reference variable to class

**New** = new is an operator it allocates memory to a reference variable object

**Sample ();** = it is a default constructor, it is provided by the JVM to create object or instance

Example;

```
package com.sgtesting.encapsulation;
public class Person
{
    String FirstName;
    int age;

    public static void main (String args [])
    {
        Person obj = new Person ();
    }
}
```

```

        obj.FirstName = "Ranav";
        obj.age = 29;
        System.out.println("FirstName:" + obj.FirstName);
        System.out.println("Age:" + obj.age);
    }
}

```

**Output:**

```

FirstName: Ranav
Age: 29

```

**Case1: can we create multiple objects for the same class**

**Ans:** Yes,

```

package com.sgtesting.encapsulation;
public class Person
{
    String FirstName;
    int age;

    public static void main (String args [])
    {
        Person obj = new Person ();
        obj.FirstName = "Ranav";
        obj.age = 29;
        System.out.println("FirstName:" + obj.FirstName);
        System.out.println("Age:" + obj.age);
        System.out.println("*****");
        Person obj1 = new Person ();
        obj1.FirstName = "Rakshith";
        obj1.age = 29;
        System.out.println("FirstName:" + obj1.FirstName);
        System.out.println("Age:" + obj1.age);
    }
}

```

**Output:**

```

FirstName: Ranav
Age: 29
*****
FirstName: Rakshith
Age: 29

```

**Example:**

```

package com.sgtesting.encapsulation;
public class Human
{
    String Firstname;
    int age;
    String ProductName;
    String Manufacturer;
    String Feathercolor;
    int NoOfWings;
    String QuantityOfMilkProvides;
    String BreedName;
    public static void main (String args [])
    {

```

```

        Human sachin = new Human ();
        sachin.Firstname = "Sachin Tendulkar";
        sachin.age = 55;
        sachin.ProductName = "Boost";
        sachin.Manufacturer = "MRF";
        sachin.Feathercolor = "";
    }
}

```

In the above example for the **“object” sachin** some of the information is relevant and some of the details are irrelevant, for that purpose we have to write independent classes those classes contain specific details

```

package com.sgtesting.encapsulation;

class Person1
{
    String FirstName;
    int age;
}

class Product
{
    String ProductName;
    String Manufacturer;
}

class Bird
{
    String FeatherColor;
    int NoOfWings;
}

class DomesticAnimal
{
    String QuantityOfMilkProvides;
    String BreedName;
}

public class MainDemo1
{
    public static void main (String args [])
    {
        Person1 obj1 = new Person1();
        Obj1.Firstriame = "Sachin Tendulkar";
        obj1.age = 29;
        Syso ("Firstname: " + obj1.Finstname);
        Syso ("Age: " + obj1.age);
        Syso ("*****");

        Product1 car = new Product1();
        car.ProductName = "Marthi suzuki";
        car.Manufacturer = "Marthi suzuki";
        Syso ("manuteruer:" + car.ProductName);
        Syso ("Manufacturer:" + car.Manufacturer);
        Syso("*****");
        Bird parrot = New Bird ();
        parrot.Feathercolor = " Green";
        parrot.NoOfwings = 2;
        Syso ("Feathercolor: "+ parrot.Feathercolor),
    }
}

```



```

        Syso("Noofwings:" + parrot.Noofwings);
        Syso ("*****");

        DomesticAnimal cow = new DomesticAnimal ();
        cow.QuantityOfMilkProvides = "5 liters";
        cow.BreedName = "Jersey";
        Syso ("QuantityOfMilkProvides:" + cow.QuantityOfMilkProvides);
        Syso ("BreedName:" + cow.breedName );

    }
}

```

### Assignments:

- 1) create a class for employee, department, insurance and execute each class?
- 2) create a class for product, order, items and execute each class member?
- 3) create a class for student, library, sports, CSdept execute each member of the class?
- 4) create a class for country, state, district, taluk and execute each member of the class?
- 5) create a class for fruits, flowers, vegetables and WAP to execute each member of the class?
- 6) create a class for desktop, laptop, mobile and execute each member of the class?

### class member:

-----

The components of a class, such as its instance variables or methods are called as class members.

class members are 2 types:

- instance or non-static members
- static members

### instance or non-static members:

- a) instance variable
- b) constructor
- c) instance methods
- d) instance block

- to access the instance member of the class we need to create an object to the class
- instance members can be directly accessed inside the constructor or instance methods
- instance members cannot be accessed directly inside the static members, if you want to access you need to create an object to the class

### static members:

- a) static variables
- b) static method
- c) static block

- static members belong to the class, hence no need to create an object to the class to access the static members
- static members can be accessed directly inside the instance member as well as inside static members
- the static member from another class should be accessed by giving class name reference
- static member can also be accessed by creating the object to the class, but it's doesn't recommended
- **static block:** it is used to assign the values to static variables to call the static methods etc...

### instance or non-static members:

#### a) instance variable:

- it is belonging to the instance members
- it can be access anywhere in the class
- to assign the values to the instance variable we use either constructor or methods
- if the instance variable name and local variable name (variable coming from the method/constructor) are same to differentiate them we use "this" keyword to the instance variable

- anything you directly written in the class are instance variable

**example:**

```
package instancevariable;
class variables
{
    String name;
    int age;
    boolean active;
    double salary;

    variables ()
    {
        // explicit default constructor → manually we creating constructor
        name = "Ranav";
        age = 29;
        active = true;
        salary = 10000.99;
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Status: " + active);
        System.out.println("Salary: " + salary);
    }
}

public class InstanceVariables
{
    public static void main (String args [])
    {
        Variables var = new Variables ();
    }
}
```

**Output:**

```
Name: Ranav
Age: 29
Status: true
Salary: 10000.99
```

**Note:**

- Integer.parseInt ();  

↓

↓

class
static method
- Arrays.sort ();  

↓

↓

class
static method
- System.out ();  

↓

↓

class
static method

-     Arrays.ToString ();  
           ↓           ↓  
       class       static method

## b) Constructor:

-----  
 Constructor name must be similar to the class name, the constructor never returns any value but the execution of constructor always provides an object or instance

Or

It is a piece or block of code which can be reused, it is an instance member of the class to access instance members we have to create an object to the class

## Why constructor is required?

Ans:

- it is used to create an object to the class
- it is used to assign the values to the instance variable
- it is used to call and run the instance members
- it is required to reuse the code

## Rules for constructor: \*\*\*

- every class by default will have one constructor which is called as a default constructor
- the default constructor will be created by the compiler in java
- we can create more than one constructor inside/within the class having different set of signatures/parameters (constructor overloading)
- the constructor can be written with arguments (parameterized constructor) or without arguments (default constructor), hence there are 2 types of constructors are there
  - default constructor
  - parameterized constructor
- constructor can't return any value
- the constructor's name is predefined (must be same as class name)
- multiple constructors in the class requires creation of multiple objects in order to access/execute all of them
- if we create a parameterized constructor in the class then the default constructor will not be available, to get it back we need to explicitly create a default constructor

## Java supports three types of constructors:

- default constructor
- no-args constructor
- parameterized constructor

**default constructor:** if a class doesn't have any explicitly specified constructors in this case in order to create an instance or an object JVM provides default constructor

**No-args constructor:** This type of constructor has to explicitly specified in a class and it never accepts any parameters

## Example:

```
package com.sgtesting.noargsconstructor;

class Student
{
    String StudentName;
    String StudentID;
    String Branch;
    String block;
```

```

String StuLocation;

Student ()
{
    StudentName="Ranav";
    StudentID="1GU101617";
    Branch="Mechanical Engineering";
    block="C-Block";
    StuLocation="Koratagere";
    System.out.println("Student Name:"+StudentName);
    System.out.println("Student ID:"+StudentID);
    System.out.println("Branch Name:"+Branch);
    System.out.println("Block:"+block);
    System.out.println("Stu Location:"+StuLocation);
    System.out.println("*****");
}

}

class Library
{
    String StudentID;
    String LibID;
    String BookName;
    String Author;
    String IssueDate;

    Library ()
    {
        StudentID="1GU101617";
        LibID="1GULib001";
        BookName="Turbo Machines";
        Author="Rakshith";
        IssueDate="15/10/2015";
        System.out.println("Student ID:"+StudentID);
        System.out.println("Lib ID:"+LibID);
        System.out.println("Book Name:"+BookName);
        System.out.println("Author Name:"+Author);
        System.out.println("Issue Date:"+IssueDate);
        System.out.println("*****");
    }
}

class Sports
{
    String StudentID;
    String SportName;
    String Coach;
    String GrondType;
    String SportEquip;

    Sports ()
    {
        StudentID="1GU101617";
        SportName="Cricket";
        Coach="Dhoni";
        GrondType="Outdoor";
        SportEquip="Bat and Ball";
        System.out.println("Student ID:"+StudentID);
        System.out.println("Sport Name:"+SportName);
        System.out.println("Coach Name:"+Coach);
        System.out.println("Ground Type:"+GrondType);
        System.out.println("Sport Equip:"+SportEquip);
        System.out.println("*****");
    }
}

```

```

    }

    class CSdept
    {
        String StudentID;
        String LabName;
        String LabID;
        String GuideName;
        String LabDay;

        CSdept ()
        {
            StudentID="1GU101617";
            LabName="Java";
            LabID="GUCS01";
            GuideName="Prabhakar";
            LabDay="Friday";
            System.out.println("Student ID:"+StudentID);
            System.out.println("Lab Name:"+LabName);
            System.out.println("Lab ID:"+LabID);
            System.out.println("Guide Name:"+GuideName);
            System.out.println("Lab Day:"+LabDay);
        }
    }

    public class ClassStudentLibrarySportsCSDept {

        public static void main(String[] args) {
            // create a class for student, library, sports, csdept and execute each member of the class?

            Student stu=new Student();
            Library lib=new Library();
            Sports spo=new Sports();
            CSdept cs=new CSdept();

        }
    }

```

#### Note:

- here we are segregating all the common class member into another class
- value to the variable should be assigned either inside the method/constructor of the class
- here we have achieved min line of code in the main method but the problem is both the instance are getting the same value
- it is because we are using the default constructor, the constructor should be used when we single instance to the class
- but here we have 2 instances hence go for parameterized constructor

#### Parameterized constructor:

-----

This type of constructor accepts parameters

#### Example:

```

package com.sgtesting.parameterizedconstructor;

class Employee
{
    String EmpNumber;
    String EmpName;

```

```

String EmpJob;
int EmpSalary;
String EmpLocation;

Employee (String EmpNum,String EmpNam,String Empjob,int EmpSal,String EmpLoc)
{
    EmpNumber=EmpNum;
    EmpName=EmpNam;
    EmpJob=Empjob;
    EmpSalary=EmpSal;
    EmpLocation=EmpLoc;
    System.out.println("Emp Number:"+EmpNumber);
    System.out.println("Emp Name:"+EmpName);
    System.out.println("Emp Job:"+EmpJob);
    System.out.println("Emp Salary:"+EmpSalary);
    System.out.println("Emp Location:"+EmpLocation);
    System.out.println("*****");
}

}

class Department
{
    String EmpNumber;
    String DeptID;
    String DeptName;
    String DeptBlock;
    String DeptLocation;

    Department(String EmpNum,String DeptID,String DepNam,String DepBlo,String DepLoc)
    {
        EmpNumber=EmpNum;
        DeptID=DeptID;
        DeptName=DepNam;
        DeptBlock=DepBlo;
        DeptLocation=DepLoc;
        System.out.println("Emp Number:"+EmpNumber);
        System.out.println("Dept ID:"+DeptID);
        System.out.println("Dept Name:"+DeptName);
        System.out.println("Dept Block:"+DeptBlock);
        System.out.println("Dept Location:"+DeptLocation);
        System.out.println("*****");
    }
}

class Insurance
{
    String EmpNumber;
    String PolicyName;
    long PolicyNumber;
    long PolicyAmount;
    String StartDate;
    String EndDate;

    Insurance (String EmpNum,String PloNam,long PolNum,long PolAmo,String StDat,String EndDt)
    {
        EmpNumber=EmpNum;
        PolicyName=PloNam;
        PolicyNumber=PolNum;
        PolicyAmount=PolAmo;
        StartDate=StDat;
        EndDate=EndDt;
        System.out.println("Emp Number:"+EmpNumber);
        System.out.println("Policy Name:"+PolicyName);
        System.out.println("Policy Number:"+PolicyNumber);
    }
}

```

```

        System.out.println("Policy Amount:"+PolicyAmount);
        System.out.println("Start Date:"+StartDate);
        System.out.println("End Date:"+EndDate);
    }
}

public class ClassEmpDeptInsu {

    public static void main (String [] args) {
        // create a class for employee, department, insurance and execute each class members?

        Employee emp= new Employee("1SG001","Ranav","QA-Engineer",65025,"Bengaluru");

        Department dept= new Department("1SG001","1SGDEP03","QA","A-Block","Bengaluru");

        Insurance insu= new Insurance ("1SG001","Health Insurance", 123456,25000, "10/10/2021", "10/12/2024");

    }
}

```

### Output:

```

Emp Number:1SG001
Emp Name: Ranav
Emp Job: QA-Engineer
Emp Salary:65025
Emp Location: Bengaluru
*****

Emp Number:1SG001
Dept ID:1SGDEP03
Dept Name: QA
Dept Block: A-Block
Dept Location: Bengaluru
*****

Emp Number: 1SG001
Policy Name: Health Insurance
Policy Number:123456
Policy Amount:25000
Start Date:10/10/2021
End Date:10/12/2024

```

**Case 1: Without Using instance variables can we use parameterized constructor in a class?**

**Ans:** Yes,

```

class Student
{
    Student (String fname, String branch, int rollNo)
    {
        System.out.println("First Name: " +fname);
        System.out.println("Branch: " +branch);
        System.out.println("Roll Number: " +rollNo);
    }
}

public class StudentDemo
{
    public static void main (String args [])
    {
        Student s1 = new Student ("Ranav", "Mechanical Engineering", 77);
    }
}

```

**Output:**

First Name: Ranav  
Branch Name: Mechanical Engineering  
Roll Number: 77

**Case2: Suppose the values you are supplying to a parameterized constructor the same value you would like to use in a same class and other methods in this case we have to prefer assigning parameter value into a instance variable**

```
class Student2
{
    String FirstName;
    String BranchName;
    int RollNumber;

    Student2 (String fname, String branch, int rollno)
    {
        FirstName=fname;
        BranchName=branch;
        RollNumber=rollno;
        System.out.println("First Name: " +FirstName);
        System.out.println("Branch Name: " +BranchName);
        System.out.println("Roll Number: " +RollNumber);
    }

    void show ()
    {
        System.out.println("In method body !!!!");
        System.out.println("First Name: " +FirstName);
        System.out.println("Branch Name: " +BranchName);
        System.out.println("Roll Number: " +RollNumber);
    }
}

public class Student2Demo
{
    public static void main (String args [])
    {
        Student2 s1=new Student2("Ranav", "Mechanical Engineering",77);
        S1.show ();
    }
}
```

**Output:**

First Name: Ranav  
Branch Name: Mechanical Engineering  
Roll Number: 77  
In method body  
First Name: Ranav  
Branch Name: Mechanical Engineering  
Roll Number: 77

**Case3: If instance variable name and parameter name both are same, in this case the parameter hides the instance variable within the body of the constructor**

```
class Student3
{
    String FirstName;
    String BranchName;
    int RollNumber;

    Student3 (String FirstName, String branch, int RollNumber)
    {
```



```

        FirstName= FirstName; ----- → here it's shown a warning message
        BranchName= branch;
        RollNumber= RollNumber; ----- → here it's shown a warning message
        System.out.println("First Name: " +FirstName);
        System.out.println("Branch Name: " +BranchName);
        System.out.println("Roll Number: " +RollNumber);
    }

    void show ()
    {
        System.out.println("In method body !!!!");
        System.out.println("First Name: " +FirstName);
        System.out.println("Branch Name: " +BranchName);
        System.out.println("Roll Number: " +RollNumber);
    }
}

public class Student3Demo
{
    public static void main (String args [])
    {
        Student3 s1=new Student3("Ranav", "Mechanical Engineering",77);
        S1.show ();
    }
}

```

**Output:**

```

First Name: Ranav
Branch Name: Mechanical Engineering
Roll Number: 77
In method body
First Name: null
Branch Name: Mechanical Engineering
Roll Number: 0

```

**Note:** solution for the case is using “this” operator

- declare parameterized constructor so that different instance of the class can have different value
  - a) without “this” keyword is a local variable
  - b) with “this” keyword is a instance variable
- object → <class name> <instance variable> <constructor>
- public class purpose is to execute/run
- source file can have n number of classes but one must be public in scope
- whatever class is public the same name we must use for the source file
- the public class will have main method, it is mainly used for running the code, the main method must have min line of code (only executable code)
- if the class has multiple instances, then we have to use parameterized constructor to pass different values

**this Operator:**

-----  
 “this” operator refers to the current class alone  
 “this” can be used with:

- instance variable
- constructor
- instance method

**case1: using “this” operator with instance variable level**

```

class Student3
{
    String FirstName;
    String BranchName;
    int RollNumber;
}

```

```

Student3 (String FirstName, String branch, int RollNumber)
{
    this.FirstName = FirstName;
    this.BranchName = branch;
    this.RollNumber = RollNumber;
    System.out.println("First Name: " +FirstName);
    System.out.println("Branch Name: " +BranchName);
    System.out.println("Roll Number: " +RollNumber);
}

void show ()
{
    System.out.println("In method body !!!!");
    System.out.println("First Name: " +FirstName);
    System.out.println("Branch Name: " +BranchName);
    System.out.println("Roll Number: " +RollNumber);
}

}

public class Student3Demo
{
    public static void main (String args [])
    {
        Student3 s1=new Student3("Ranav", "Mechanical Engineering",77);
        S1.show ();
    }
}

```

**Output:**

```

First Name: Ranav
Branch Name: Mechanical Engineering
Roll Number: 77
In method body
First Name: Ranav
Branch Name: Mechanical Engineering
Roll Number: 77

```

**Constructor Overloading:**

Writing more than one constructor inside/within the class where constructor name is same but having different signatures (parameters) is called as constructor overloading

**Rules for constructor overloading:**

- the number of parameters used in the constructor should be different
- the datatype of the parameters used in the constructor should be different
- the sequence of datatype of the arguments (parameters) used in the constructor should be different

**example:**

```

package com.sgtesting.constructoroverloading;
class Employee
{
    Employee (String fname)
    {
        System.out.println("First Name: " + fname);
    }

    Employee (double sal)
    {
        System.out.println("Salary: " + sal);
    }
}

```

```

        Employee (String address, int pincode)
        {
            System.out.println("Address: " + address);
            System.out.println("Pincode: " + pincode);
        }
    }

    public class EmployeeDemo
    {
        public static void main (String args [])
        {
            Employee c1 = new Employee ("Ranav");
            Employee c2 = new Employee (65892.256);
            Employee c3 = new Employee ("Tumakuru", 572101);
        }
    }

```

**Output:**

```

First Name: Ranav
Salary: 65892.256
Address: Tumakuru
Pincode: 572101

```

**Assignment:**

**By using constructor overloading create or write a below program**

- 1) university
- 2) college
- 3) examination

**Constructor chaining:**

-----

By using constructor chaining we can execute constructors in a sequential order

Constructor chaining can be achieved by using "this" operator with constructor execution

**While performing constructor chaining with "this" operator we have to follow the below rules:**

- "this" operator with constructor execution always must be the first statement within the body of the constructor
- "this" operator with constructor execution must be the only one statement within the body of the constructor

**Example:**

```

package com.sgtesting.parameterizedconstructor;

class College
{
    College (String name)
    {
        System.out.println("College Name: " + name);
    }

    College (int branchcount)
    {
        // call the College (String name) here using "this" operator it will execute the whole class bcz
    }
}

```

```

        // it defines the constructor chaining
        this ("CIT Engineering College");
        System.out.println("# of Branches in College: " + branchcount);
    }
}
public class ConstructorChaining
{
    public static void main (String args [])
    {
        College col = new College (6);
    }
}

```

**Output:**

```

College Name: CIT Engineering College
# of Branches in College: 6

```

**Note:** Constructor chaining is nothing but calling the constructor in the other constructor level

- In case we have (String name, int noofstaff) this also execute

**Programming approach on Constructor:**

**Case1: Write a constructor it should accept integer array as a parameter and find the sum of all the elements?**

```

package com.sgtesting.parameterizedconstructor;

class Demo1
{
    // step1: create a integer array that should accepts elements from array
    Demo1 (int a [])
    {
        // step2: find the integer array elements are printing are not
        // step3: find the sum of all the elements from the integer array
        int sum = 0;
        for (int i=0; i<a.length; i++)
        {
            sum = sum + a[i];
        }
        // step4: read the sum from the variable sum
        System.out.println("Sum Result: " + sum);
    }
}

public class SampleDemo
{
    public static void main (String args [])
    {
        // step5: create an integer array to pass the values to the object
        int b [] = {1,2,3,4,5,6};
        Demo1 dem = new Demo1 (b);
    }
}

```

**Output:**

```

21

```

**Case2: WAC it should accept 2D character array as a parameter, in constructor all the elements of 2D array assign into a 1D array and read the elements from 1D character array?**

```

package com.sgtesting.parameterizedconstructor;

```

```

class Demo2
{
    Demo2 (char ch [] [])
    {
        // declare an 1D array
        char b [] = new char [ch.length * ch[0].length];
        char k=0;
        // assign elements into a 1D array from 2D array
        for (int i=0; i<ch.length; i++)
        {
            for(int j=0; j<ch[0].length; j++)
            {
                b [k] = ch [i][j];
                k++;
            }
        }
        // read the elements from the 1D array

        for (char m : b)
        {
            System.out.print(m+ " ")
        }
    }
}

public class SampleDemo2
{
    public static void main (String args [])
    {
        char ch [] [] = {{'A', 'B', 'C'}, {'D', 'E', 'F'}};
        Demo2 dem = new Demo2 (ch);
    }
}

```

**Output:**

A B C D E F

**Case3: Verify the given number is prime number or not?**

**package** com.sgtesting.parameterizedconstructor;

```

class Demo2
{
    Demo2(int num)
    {
        int flag = 0;
        for (int i=2; i<num; i++)
        // here we skip i=1 & taking i<num (means neglecting num) bcz of prime number definition
        {
            if (num%i == 0)
            {
                flag++;
                break;
            }
        }
        if (flag == 0)
        {

```

```

        System.out.println(num + " is a Prime Number");
    }
    else
    {
        System.out.println(num + " is not a Prime Number");
    }
}
}

```

```

public class PrimeNumber
{
    public static void main (String [] args)
    {
        Demo2 dem = new Demo2 (7);
        Demo2 dem1 = new Demo2 (8);
        Demo2 dem2 = new Demo2 (13);
        Demo2 dem3 = new Demo2 (21);
    }
}

```

#### Output:

```

7 is a Prime Number
8 is not a Prime Number
13 is a Prime Number
21 is not a Prime Number

```

#### Case4: WAC to display the Fibonacci series?

```

package com.sgtesting.parameterizedconstructor;

class Demo3
{
    Demo3()
    {
        int fn = 0;
        int sn = 1;
        System.out.println(fn);
        System.out.println(sn);

        for (int i=1; i<=10; i++)
        {
            int tn = fn + sn;
            System.out.println(tn);
            fn = sn;
            sn = tn;
        }
    }
}

public class Fibonacci
{
    public static void main(String[] args)
    {
        Demo3 dem = new Demo3 ();
    }
}

```

#### Output:

0  
1  
1  
2  
3  
5  
8  
13  
21  
34  
55  
89

**Case5: By using constructor overloading and by applying 2 integer parameters, perform basic mathematical activities**

```
package com.sgtesting.constructoroverloading;

class Demo4
{
    Demo4 (int x, int y, String action)
    {
        int res = 0;
        switch (action)
        {
            case "Add":
                res=x+y;
                System.out.println("Addition result: "+res);
                break;
            case "Sub":
                res=x-y;
                System.out.println("Subtraction result: "+res);
                break;
            case "Div":
                res=x/y;
                System.out.println("Division result: "+res);
                break;
            case "Mult":
                res=x*y;
                System.out.println("Multiplication result: "+res);
                break;
        }
    }
}

public class BasicMathsActivities
{
    public static void main (String [] args)
    {
        Demo4 dem = new Demo4 (10,20,"Add");
        Demo4 dem1 = new Demo4 (25,10,"Sub");
        Demo4 dem2 = new Demo4 (100,5,"Div");
        Demo4 dem3 = new Demo4 (10,20,"Mult");
    }
}
```

**Output:**

Addition result: 30  
Subtraction result: 15  
Division result: 20  
Multiplication result: 200

#### Case5:

```
package constructor;

import java.util.Arrays;

class Sample2
{
    //Q: Write a constructor for displaying the missing numbers from the array?
    Sample2 (int arr [])
    {
        Arrays.sort(arr);
        int max = arr[arr.length-1];
        int x = 0;
        for (int i=0; i<max; i++)
        {
            if(i==arr[x]) {
                x++;
            }
            else
            {
                System.out.println("Missing number: " + i);
            }
        }
    }

    //Q: Write a constructor for subtraction of 2 array values?
    Sample2(int a [], int b [])
    {
        if (a.length == b.length)
        {
            int res [] = new int [a.length];
            for (int i=0; i<res.length; i++)
            {
                res[i] = a[i] - b[i];
                System.out.println(res[i]);
            }
            //System.out.println(Arrays.toString(res));
        }
        else
        {
            System.out.println("Subtraction can't be done as the arrays are uneven in the size");
        }
    }

    Sample2 (String city [])
    {
        //Q: Write a constructor to find the city names "Bangalore" is present in the array?
        boolean flag = false;
        for (int i=0; i<city.length; i++)
```



```

        {
            if (city[i].equals ("Bangalore"))
            {
                flag = true;
                break;
            }
        }

        if (flag == true) System.out.println("Bangalore was found");
        else System.out.println("Bangalore is not found");
    }
}

public class QA1
{
    public static void main (String [] args)
    {
        int arr [] = {1, 3, 4, 5, 7, 9, 10};
        Sample2 s1 = new Sample2(arr);
        System.out.println("*****");
        Sample2 s2 = new Sample2(new int[] {10, 20, 30}, new int[] {1, 2, 3});
        System.out.println("*****");
        Sample2 s3 = new Sample2(new String [] {"Gadag", "Raichur", "Mangalore", "Mysore",
        "Hubballi"});
        System.out.println("*****");
        Sample2 s4 = new Sample2(new int [] {1, 2, 8, 9, 10, 11});
    }
}

```

### Drawback/Limitations of constructors:

- Constructor names are predefined
- Constructor will not return any value
- Multiple constructors require creation of multiple objects in order to execute all the constructor
- If we want to perform different tasks using constructor which requires same set of parameters is not possible in constructor
- Constructor overloading is not supported as the constructor's name should be similar to the class name
- Constructor name are ambiguous/confusing in nature

### Methods:

-----

It is a block or piece of code which can be reused

### Difference between Constructor and Method:

| Constructor                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Method                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>• Names are predefined</li> <li>• Will not return any value</li> <li>• Multiple objects are required to run the multiple Constructor</li> <li>• Different tasks with the same set of arguments are not possible</li> <li>• Here we have default constructor concept</li> <li>• If you create object to the class the respective Constructor will get execute</li> <li>• Constructor overriding is not possible, because the</li> </ul> | <ul style="list-style-type: none"> <li>* Names are user defined</li> <li>* it may or may not return the value</li> <li>* Using one object of the class we can run multiple methods</li> <li>* it is possible by changing method names</li> <li>* no default method concept</li> <li>* if we create object to class the method will not get execute, we must call the method to execute it</li> <li>* method overloading and method overriding is</li> </ul> |

Constructor name and class name must be same possible

### Method types:

- a) Methods which don't return value
- b) Methods which return value

### Methods which don't return value:

-----  
These types of methods should be preceded by "void" keyword, here again methods can be classified into 2 types

- Methods which don't accept parameters
- Methods which accept parameters

- **Methods which don't accept parameters:**

-----  
These types of methods never having any parameter, it is similar to the no args constructor

### Syntax:

```
class Demo
{
    void<methodName1> ()
    {
        <statement>;
    }

    void<methodName2> ()
    {
        <statement>;
    }

    -
    -
    -
    void<methodName^th> ()
    {
        <statement>;
    }
}
```

### Example:

```
package com.sgtesting.methoddemo;

class Maths1
{
    void addition()
    {
        int x, y, res;
        x=10;
        y=20;
        res=x+y;
        System.out.println("addition result: "+res);
    }
}

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

```

    {
        Maths1 ma = new Maths1();
        ma.addition ();
        ma.addition ();
    }
}

```

### Output:

addition result: 30  
addition result: 30

- **Methods which accept parameters:**

-----

These types of methods accept parameters and eliminates hard coding of the values

### Syntax:

```

class Demo
{
    void<methodName1> ()
    {
        <statement>;
    }

    void<methodName2> ()
    {
        <statement>;
    }

    -
    -
    -
    void<methodName^th> ()
    {
        <statement>;
    }
}

```

### Example:

```

package com.sgtesting.methoddemo;

class Maths3
{
    void multiplication (int x, int y)
    {
        int res = x*y;
        System.out.println("Multiplication result: "+res);
    }

    void addition (int x, int y)
    {
        int res = x+y;
        System.out.println("addition result: "+res);
    }
}

public class MethodAcceptParam
{

```

```

    public static void main(String[] args)
    {
        Maths3 ma = new Maths3();
        ma.multiplication(8, 10);
        ma.addition(10, 25);
        ma.multiplication(5, 7);
        ma.addition(12, 10);
    }
}

```

**Output:**

```

Multiplication result: 80
addition result: 35
Multiplication result: 35
addition result: 22

```

**1) WAM for the given number it has to find the factorial?**

```

package com.sgtesting.methoddemo;

import java.util.Scanner;

class Factorial
{
    void fact1()
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number: ");
        int num=sc.nextInt ();
        int fact=1;
        for (int i=1; i<=num; i++)
        {
            fact=fact*i;
        }
        System.out.println("Factorial of a Given number is: "+fact);
        sc.close ();
    }
}

public class MethodFactorial
{
    public static void main (String [] args)
    {
        Factorial fac = new Factorial ();
        fac.fact1 ();
    }
}

```

**Output:**

```

Enter the number:
5
Factorial of a Given number is: 120

```

**2) WAM to display number in between 100 to 50 which are divisible by 3?**

```

package com.sgtesting.methoddemo;

class Divide

```

```

{
    void div()
    {
        for(int i=100; i>=50; i--)
        {
            if(i%3==0)
            {
                System.out.println("Number which are divisible by 3 is: "+i);
            }
        }
    }
}

public class MethodDivisibleBy3
{
    public static void main(String[] args)
    {
        Divide d=new Divide();
        d.div();
    }
}

```

### Output:

```

Number which are divisible by 3 is: 99
Number which are divisible by 3 is: 96
Number which are divisible by 3 is: 93
Number which are divisible by 3 is: 90
Number which are divisible by 3 is: 87
Number which are divisible by 3 is: 84
Number which are divisible by 3 is: 81
Number which are divisible by 3 is: 78
Number which are divisible by 3 is: 75
Number which are divisible by 3 is: 72
Number which are divisible by 3 is: 69
Number which are divisible by 3 is: 66
Number which are divisible by 3 is: 63
Number which are divisible by 3 is: 60
Number which are divisible by 3 is: 57
Number which are divisible by 3 is: 54
Number which are divisible by 3 is: 51

```

### 3) WAM to perform addition of 2 matrix and display the result?

```

package com.sgtesting.methoddemo;

class MatrixAddition
{
    void matrixAddition(int x[][],int y[][])
    {
        if (x.length == y.length && x[0].length == y[0].length )
        {
            for (int i=0; i<x.length; i++)
            {
                for (int j=0; j<x[i].length; j++)
                {
                    int res=x[i][j] + y[i][j];
                    System.out.print(res+" ");
                }
            }
        }
    }
}

```

```
}  
    }  
}  
  
System.out.println();
```

## Output:

$$\begin{array}{ccc} 5 & 7 & 9 \\ 11 & 13 & 15 \end{array}$$

#### 4) WAM to verify the given number is prime or not?

```
package com.sgtesting.methoddemo;

class Prime
{
    void primeNum (int num)
    {
        int flag=0;
        for (int i=2; i<num; i++)
        {
            if (num%i == 0)
            {
                flag++;
                break;
            }
        }
        if (flag == 0)
        {
            System.out.println(num+" is a prime number");
        }
        else
        {
            System.out.println(num+" is not a prime number");
        }
    }
}

public class MethodPrimeNumber
{
    public static void main (String [] args)
    {
        Prime pm = new Prime ();
        pm.primeNum (8);
        pm.primeNum (7);
        pm.primeNum (20);
        pm.primeNum (31);
    }
}
```

```
}
```

### Output:

```
8 is not a prime number
7 is a prime number
20 is not a prime number
31 is a prime number
```

### Assignment:

- 1) WAM to display cube of first 10 numbers in reverse order?
- 2) WAM to display prime number in between 50 to 100?
- 3) WAM to find the count of prime numbers in between 1 to 100?
- 4) WAM to find the sum of prime numbers in between 150 to 100?
- 5) WAM to display prime numbers in between 250 to 200?
- 6) WAM to display all complete 20 tables?
- 7) WAM for the given 2D 3\*3 character array and transpose the array?
- 8) WAM for the given 3\*3 character array concatenate all the elements?
- 9) WAM for the given 3\*3 integer array assign 2<sup>nd</sup> row elements into 1D array and display the elements in 1D array?
- 10) WAM to perform subtraction of 2 matrix and transpose the result?
- 11) WAM for the given 3\*3 2D string array concatenate the only 3<sup>rd</sup> row of the elements?
- 12) WAM to perform multiplication of 2 matrix and display the result?

### Methods which return value:

-----

These types of methods “return” values based on return type specified

### Syntax:

```
class Demo
{
    <datatype><methodname1>(parameter)
    {
        <statement>;
        return <value>;
    }

    <datatype><methodname2>(parameter)
    {
        <statement>;
        return <value>;
    }
}
```

### Example:

```
package com.sgtesting.returndemo;

class Demo4
{
    String getName ()
    {
        String s="Ranav";
        return s;
    }

    int getAge ()
    {
```

```

        int age=29;
        return age;
    }

    String [] gethobbies (String hobbies [])
    {
        return hobbies;
    }
}

public class MethodReturnValue1
{
    public static void main (String [] args)
    {
        Demo4 dm = new Demo4();
        String s1= dm.getName ();
        System.out.println(s1);
        int a1= dm.getAge ();
        System.out.println(a1);
        String str [] = {"Playing", "Riding", "Designing"};
        String s2 [] = dm.gethobbies (str);
        for (String kk : s2)
        {
            System.out.println(kk);
        }
    }
}

```

### Output:

```

Name is: Ranav
Age is: 29
Hobbies are:
Playing
Riding
Designing

```

### Example:

```

package com.sgtesting.returndemo;

class Maths
{
    boolean isPrime (int num)
    {
        int flag=0;
        boolean result = false;
        for (int i=2; i<num; i++)
        {
            if(num%i==0)
            {
                flag++;
                break;
            }
        }
        if(flag==0)
        {
            result=true;
        }
    }
}

```



```

        return result;
    }
}

public class MathsDemo {

    public static void main (String [] args) {

        Maths mat = new Maths ();
        boolean b1 = mat.isPrime (7);
        System.out.println(b1);
        boolean b2 = mat.isPrime (21);
        System.out.println(b2);
    }
}

```

### Output:

```

true
false

```

### Example:

```

package com.sgtesting.returndemo;

class Maths2
{
    boolean isPrime(int num)
    {
        int flag=0;
        boolean result=false;
        for(int i=2;i<num;i++)
        {
            if(num%i==0)
            {
                flag++;
                break;
            }
        }
        if(flag==0)
        {
            result=true;
        }
        return result;
    }
}

public class MathsDemo1
{
    public static void main(String[] args)
    {

        Maths2 mat=new Maths2();
        boolean b1=mat.isPrime(7);
        System.out.println(b1);
        System.out.println("*****");
        System.out.println();

        // display prime numbers in between 1 to 50
    }
}

```

```

    for (int i=1;i<50;i++)
    {
        if(mat.isPrime(i)==true)
        {
            System.out.println(i+" is a prime number");
        }
    }
    System.out.println("*****");
    System.out.println();

    // display count of prime numbers in between 50 to 100

    int count=0;
    for(int i=50;i<=100;i++)
    {
        if(mat.isPrime(i)==true)
        {
            count++;
        }
    }
    System.out.println("Count of the prime numbers is: "+count);
    System.out.println("*****");
    System.out.println();

    // display sum of prime numbers in between 50 to 100
    int sum=0;
    for(int i=50;i<=100;i++)
    {
        if(mat.isPrime(i)==true)
        {
            sum+=i;
        }
    }
    System.out.println("Sum pf all the prime numbers is: "+sum);
}
}

```

### Output:

```

true
*****

1 is a prime number
2 is a prime number
3 is a prime number
5 is a prime number
7 is a prime number
11 is a prime number
13 is a prime number
17 is a prime number
19 is a prime number
23 is a prime number
29 is a prime number
31 is a prime number
37 is a prime number
41 is a prime number
43 is a prime number
47 is a prime number
*****

```

Count of the prime numbers is: 10

\*\*\*\*\*

Sum of all the prime numbers is: 1216

### WAM to return the result of addition of 2 matrix?

```
package com.sgtesting.returndemo;

class MatrixAddition
{
    int [][] getMatrixAddition (int x [][] , int y [][] )
    {
        int z[][]=new int [y.length][y[0].length];
        for (int i=0; i<y.length; i++)
        {
            for (int j=0; j<y[i].length; j++)
            {
                z[i][j]=x[i][j]+y[i][j];
            }
        }
        return z;
    }
}

public class MatrixDemo2
{
    public static void main (String [] args)
    {
        MatrixAddition mat=new MatrixAddition();
        int a [][] = {{ 1,2,3}, {4,5,6}, {7,8,9}};
        int b [][] = {{ 10,12,13},{14,15,16},{17,18,19}};
        int res [][] = mat.getMatrixAddition (a, b);

        // display 1st row of the elements
        for (int i=0; i<res.length; i++)
        {
            System.out.println(res[0][i]+" ");
        }
        System.out.println("*****");

        // display 2nd row of the elements
        for (int i=0; i<res.length; i++)
        {
            System.out.println(res[1][i]+" ");
        }
        System.out.println("*****");

        // display 3rd row of the elements
        for (int i=0; i<res.length; i++)
        {
            System.out.println(res[2][i]+" ");
        }
        System.out.println("*****");
        System.out.println();

        // transpose of the result of the return value
    }
}
```

```

        for (int i=0; i<res.length; i++)
        {
            for (int j=0; j<res[i].length; j++)
            {
                System.out.print(res[j][i]+" ");
            }
            System.out.println();
        }
        System.out.println("*****");
        System.out.println();

        // display sum of the elements
        int sum=0;
        for (int i=0; i<res.length; i++)
        {
            for (int j=0; j<res[i].length; j++)
            {
                sum+=res[i][j];
            }
        }
        System.out.println("Sum of all the elements is:" +sum);
        System.out.println();
    }
}

```

### Output:

```

11
14
16
*****
18
20
22
*****
24
26
28
*****

11 18 24
14 20 26
16 22 28
*****

```

Sum of all the elements is: 179

Case1: The return value of a method can be used as a parameter to other method?

```

package com.sgtesting.returndemo;

class Calculation
{
    int addition (int x, int y)
    {
        return (x+y);
    }
}

```

```

    int subtraction (int x, int y)
    {
        int res=(x-y);
        return res;
    }

    void multiplication (int x, int y)
    {
        int res = (x*y);
        System.out.println("multiplication result: "+res);
    }
}

public class CalculationDemo
{
    public static void main (String [] args)
    {
        Calculation cal = new Calculation ();
        cal.multiplication (cal.addition (15, 5), cal.subtraction (120, 115));
    }
}

```

### Output:

multiplication result: 100

Case2: the return value of method we can use within the body of the other method

```

package com.sgtesting.returndemo;

class Calculation1
{
    int addition (int x, int y)
    {
        return (x+y);
    }

    int subtraction (int x, int y)
    {
        int res = (x-y);
        return res;
    }

    void multiplication ()
    {
        int a=this.addition (20, 40);
        int b=this.subtraction (120, 110);
        int res=(a*b);
        System.out.println("Multiplication result: "+res);
    }
}

public class CalculationDemo1
{
    public static void main (String [] args)
    {
        Calculation1 cal = new Calculation1();
        cal.multiplication ();
    }
}

```

```
}
```

### Output:

Multiplication result: 600

### Assignment:

- 1) WAM, the method has to return sum of the first 20 numbers?
- 2) WAM, the method has to return count of add numbers in between 150 to 250?
- 3) WAM, the method has to return sum of numbers in between 100 to 200 which are divisible by 7?
- 4) From the given 2D character array return 1<sup>st</sup> row of the elements?
- 5) WAM for the given 2D string array, it has to return the concatenation result?
- 6) WAM for the given number it has to return the factorial number?

### Call by Value:

-----

There is an involvement of primitive datatype, the changes which are taking place within the body of the method these changes never reflects outside

### Example:

```
package com.sgtesting.encapsulation;

class CallByValue
{
    void Calculate (int x, int y)
    {
        x+=100;
        y*=9;
        System.out.println("In method, value of x: "+x);
        System.out.println("In method, value of y: "+y);
    }
}

public class CallByValueDemo
{
    public static void main (String [] args)
    {
        int x, y;
        x=10;
        y=5;
        System.out.println("The value of x: "+x);
        System.out.println("The value of y: "+y);

        CallByValue cal = new CallByValue ();
        cal.Calculate (x, y);
        System.out.println("After execution of method, value of x: "+x);
        System.out.println("After execution of method, value of y: "+y);
    }
}
```

### Output:

```
The value of x: 10
The value of y: 5
In method, value of x: 110
In method, value of y: 45
After execution of method, the value of x: 10
After execution of method, the value of y: 5
```

## Call By Reference:

In call by reference there is involvement of object or instance, if any changes which are taking place within the body of the method those changes can reflect outside

### Example:

```
package com.sgtesting.encapsulation;

class CallByReference
{
    int x, y;
    void calculate (CallByReference col)
    {
        col.x +=100;
        col.y *=9;
        System.out.println("In method, value of x: "+col.x);
        System.out.println("In method, value of y: "+col.y);
    }
}

public class CallByReferenceDemo
{
    public static void main (String [] args)
    {
        CallByReference col = new CallByReference ();
        col.x=10;
        col.y=5;
        System.out.println("The value of x: "+col.x);
        System.out.println("The value of y: "+col.y);

        col.calculate (col);
        System.out.println("After execution of method, value of x: "+col.x);
        System.out.println("After execution of method, value of y: "+col.y);
    }
}
```

### Output:

```
The value of x: 10
The value of y: 5
In method, value of x: 110
In method, value of y: 45
After execution of method, value of x: 110
After execution of method, value of y: 45
```

## Method Overloading:

Method overloading can be achieved by using the same method name, each method differs with number of parameters, datatype of each parameter, sequence of datatype of each parameter. Based on these approaches we can overload the method

Or

- It is a process of writing multiple methods with same name and having different signatures
- Method overloading takes place in the same (independent) class or in the inheritance (parent and child) class
- In method overloading return type is not considered whereas only method name and signature (arguments) is considered

### Rules for method overloading?

- The number of arguments used in the methods should be different
- The datatype of the arguments used in the methods should be different
- The sequence of datatype of the arguments used in the methods should be different

### Why method overloading?

**Ans:** Same tasks can be done in different/multiple possibilities

### Note:

- Based on the return type we can't overload the method

### Example:

```
package com.sgtesting.encapsulation;

class Maths3
{
    void addition (int x, int y)
    {
        System.out.println("Addition result: "+ (x+y));
    }

    void addition (int x, int y, int z)
    {
        System.out.println("Addition result: "+ (x+ y+ z));
    }

    void addition (int x)
    {
        System.out.println("Addition result: "+ (x+100));
    }
}

public class MethodOverloadingDemo
{
    public static void main (String [] args)
    {
        Maths3 mat = new Maths3();
        mat.addition (10);
        mat.addition (20, 30);
        mat.addition (50, 40, 30);
    }
}
```

### Output:

```
Addition result: 110
Addition result: 50
Addition result: 120
```

### 1) Can a method return multiple times?

**Ans:** no, method can return only once

### 2) Can a method return multiple values?

**Ans:** yes, a method can return multiple or single values

Example: array or collection or object are the best example through which you can return multiple values

### 3) What a method can return?

**Ans:** a method in java can return any type of data

Example: all datatypes, class, interface, abstract class, object, collection, array etc



#### 4) What a method can accept as arguments?

**Ans:** a method in java can accept any type of data

Example: all datatypes, abstract class, interface, object, collection, array etc

#### 5) Can we write nested methods in java?

**Ans:** no, but we can call one method inside another method

#### 6) Can we call the method in its own body?

**Ans:** yes, it is called as recursion (calling the method in its own body is called as recursion)

#### 7) During constructor do we need instance variable?

**Ans:** maybe or may not be

Example:

**may be:** if you want to get the constructor value outside, use the instance variable

**may not be:** if you don't want the constructor value outside then don't use instance variable

#### Recursion:

-----

Recursion means calling the same method itself

Or

calling the method in its own body is called as recursion

#### Example:

##### Case1:

```
package com.sgtesting.methoddemo;

class Maths4
{
    int count=10;
    void displayNumber ()
    {
        if(count<=20)
        {
            System.out.println(count);
            count=count+1;
            displayNumber ();
        }
    }
}

public class RecursionDemo
{
    public static void main (String [] args)
    {
        Maths4 mat = new Maths4();
        mat.displayNumber ();
    }
}
```

#### Output:

```
10
11
12
13
14
15
```

16  
17  
18  
19  
20

### Case2: Returning the value using recursion method

```
package com.sgtesting.methoddemo;

class Maths5
{
    int getFactorial (int num)
    {
        if(num==1)
        {
            return 1;
        }
        return num * getFactorial (num-1);
    }
}

public class RecursionDemo1
{
    public static void main (String [] args)
    {
        Maths5 mat = new Maths5();
        int result = mat.getFactorial (5);
        System.out.println("Factorial of a given number is: "+result);
    }
}
```

### Output:

Factorial of a given number is: 120

### Structure of the above program working:

5 \* getFactorial (5-1)  
5 \* 4 \* getFactorial (4-1)  
5 \* 4 \* 3 \* getFactorial (3-1)  
5 \* 4 \* 3 \* 2 \* getFactorial (2-1)

### Static components:

-----

Static components include

- Static variables
- Static methods
- Static block

### Static Variables:

-----

In order to executes static members object or instance doesn't required

All static members are belonging to class level, the exhibition of the static members allocates memory only during runtime.

**Case1:** if static variables are available in main class

**Rule1:** the instance members we can't access directly in a static method static block of the same class

Example:

```
package com.sgtesting.returndemo;

public class Person
{
    String firstname;
    int age;
    public static void main (String [] args)
    {
        Person obj1 = new Person ();
        obj1.firstname = "Ranav";
        obj1.age =29;
        System.out.println ("First Name: "+obj1.firstname);
        System.out.println ("Age: "+obj1.age);
    }
}
```

Output:

```
First Name: Ranav
Age: 29
```

**Rule2:** In static members we can directly access in a static method or static block of the same class

Example:

```
package com.sgtesting.returndemo;

public class Person1
{
    static String firstname;
    static int age;
    public static void main (String [] args)
    {
        firstname="Ranav";
        age=29;
        System.out.println("First Name: "+firstname);
        System.out.println("Age: "+age);
    }
}
```

Output:

```
First Name: Ranav
Age: 29
```

**Case2:** if static members are available in an independent class

**Ans:** if the static members are available in an independent class, we can access them in a main class → main method using <class name.static method name>;

Example:

```
package com.sgtesting.staticmethoddemo;

class Maths
{
    static String mathstype;

    static void multiplication (int x, int y)
    {
        int res=(x+y);
        System.out.println("Multiplication Result: "+res);
    }
}
```

```

    }
}

public class StaticMethodDemo2
{
    public static void main (String [] args)
    {
        Maths.mathstype="Algebra";
        System.out.println(Maths.mathstype);
        Maths.multiplication(12, 15);
    }
}

```

**Output:**

```

Algebra
Multiplication Result: 27

```

**Case3:** There are 2 classes Demo1, Demo2. Demo1 contains instance method, Demo2 contains static method, can we call instance method of Demo1 in static method of Demo2?

**Ans:** yes,

```

package com.sgtesting.staticmethoddemo;

class Demo1
{
    void addition (int x, int y)
    {
        int res=(x+y);
        System.out.println("Addition Result: "+res);
    }
}

class Demo2
{
    static void multiplication (int x, int y)
    {
        Demo1 dm = new Demo1();
        dm.addition (10, 20);
        int res = (x*y);
        System.out.println("Multiplication Result: "+res);
    }
}

public class StaticDemo3
{
    public static void main (String [] args)
    {
        Demo2.multiplication(15, 20);
    }
}

```

**Output:**

```

Addition Result: 30
Multiplication Result: 300

```

**Case4:** there are 2 classes Demo1 and Demo2, Demo1 contains instance method and Demo2 contains static method can we call static method of Demo2 in instance method of Demo1?

Ans: yes

```
package com.sgtesting.staticmethoddemo;

class Demo1
{
    void addition (int x, int y)
    {
        Demo21.multiplication(15, 20);
        int res = (x+y);
        System.out.println("Addition Result: "+res);
    }
}

class Demo2
{
    static void multiplication (int x, int y)
    {
        int res = (x*y);
        System.out.println("Multiplication Result: "+res);
    }
}

public class StaticDemo4
{
    public static void main (String [] args)
    {
        Demo1 dm = new Demo1();
        dm.addition (100, 200);
    }
}
```

**Output:**

```
Multiplication Result: 300
Addition Result: 300
```

**Case5:** There are 2 classes Demo1 and Demo2, Demo contains static method and Demo2 also contains static method. Can we call Demo1 static method in Demo2 static method

Ans: yes,

```
package com.sgtesting.staticmethoddemo;

class Dmo1
{
    static void addition (int x, int y)
    {
        int res = (x+y);
        System.out.println("Addition Result: "+res);
    }
}

class Demo2
{
    static void multiplication (int x, int y)
    {
        Demo1.addition(10, 100);
        int res = (x*y);
        System.out.println("Multiplication Result: "+res);
    }
}
```

```

    }

    public class StaticMethodDem5
    {
        public static void main (String [] args)
        {
            Demo2.multiplication(10, 20);
        }
    }

```

**Output:**

Addition Result: 110  
 Multiplication Result: 200

**Case6:** There are 2 classes Demo1 and Demo2, Demo1 contains instance method and Demo2 contains instance method. Can we execute instance method of Demo2 inside the instance method of Demo1

**Ans:** yes,

```

package com.sgtesting.staticmethoddemo;

class Demo1
{
    void addition (int x, int y)
    {
        int res = (x+y);
        System.out.println("Addition Result: "+res);
    }
}

class Demo2
{
    void multiplication (int x, int y)
    {
        Demo1 dm= new Demo1();
        dm.addition (10, 200);
        int res = (x*y);
        System.out.println("Multiplication Result: "+res);
    }
}

public class StaticMethodDemo6
{
    public static void main (String [] args)
    {
        Demo2 dm= new Demo2();
        dm.multiplication (10, 20);
    }
}

```

**Output:**

Addition Result: 210  
 Multiplication Result: 200

**Static Block:**

-----

**Case1:** There is a class it has static variables, methods and static block, find out the execution order of static members

**Example:**

```
package com.sgtesting.staticblockdemo;

class Student
{
    static String firstname;
    static int age;

    static
    {
        firstname = "Ranav";
        showFN ();
        showAge ();
    }

    static void showFN ()
    {
        System.out.println("FirstName: "+firstname);
    }

    static void showAge ()
    {
        System.out.println("Age: "+age);
    }
}

public class StaticBlockDemo1
{
    public static void main (String [] args)
    {
        Student.age =29;
        System.out.println("Age in method: "+Student.age);
    }
}
```

**Output:**

```
FirstName: Ranav
Age: 0
Age in method: 29
```

**Case2:** if a class contains static block alone, how to execute the static block?

```
package com.sgtesting.staticblockdemo;

class Test
{
    static
    {
        System.out.println("It is a static block");
    }
}

public class StaticBlockDemo2
{
    public static void main (String [] args)
    {
        Test te = new Test ();
    }
}
```

```
}
```

**Output:**

It is a static block

**Case3:** If a class contains static block as well as constructor, in this case which will execute first?

```
package com.sgtesting.staticblockdemo;

class Test2
{
    Test2()
    {
        System.out.println("It is a no-args constructor");
    }

    static
    {
        System.out.println("It is a static block");
    }
}

public class StaticBlockDemo3
{
    public static void main (String [] args)
    {
        Test2 dm = new Test2();
    }
}
```

**Output:**

It is a static block  
It is a no-args constructor

**Case4:** If a class contains multiple static blocks, what is the order of execution of the static block?

```
package com.sgtesting.staticblockdemo;

class Test3
{
    static
    {
        System.out.println("It is a first static block");
    }

    static
    {
        System.out.println("It is a second static block");
    }

    static
    {
        System.out.println("It is a third static block");
    }
}

public class StaticBlockDemo4
{
}
```



```

    public static void main (String [] args)
    {
        Test3 te = new Test3();
    }
}

```

### Output:

It is a first static block  
 It is a second static block  
 It is a third static block

**Case5:** whether static method can return a value?

1) **WAM, method should accept 1D integer array and a method should return sum of all elements**

Ans:

```

package com.sgtesting.staticblockdemo;

class Calculation
{
    static int getSumOfElements (int x[])
    {
        int sum=0;
        for (int i=0; i<x.length; i++)
        {
            sum=sum+x [i];
        }
        return sum;
    }
}

public class StaticBlockDemo5
{
    public static void main (String [] args)
    {
        int a [] = {2,4,6,8,10,12,14,16,18,20};
        int res=Calculation.getSumOfElements(a);
        System.out.println("Sum of all the elements: "+res);
    }
}

```

### Output:

Sum of all the elements: 110

### Assignment:

- 1) For the given 2 1D integer array perform the addition of each element and return the result?
- 2) For the given 2 1D integer array, create a resultant array and return the resultant array?
- 3) For the given 2 1D integer array subtract each element and return the result of 2<sup>nd</sup> half of the elements?
- 4) For the given 2D integer array the method as to return all the elements of 2D array in the form of 1D array?
- 5) For the given 2 1D integer array assign elements into another array in reverse order and return the second array?

### Instance Block:

-----

Instance block contains only the body it doesn't have any name

### Example:

```

package com.sgtesting.instanceblock;

```

```

class Demo11
{
    {
        System.out.println("It is an instance block");
    }
}

public class SampleDemo
{
    public static void main (String [] args)
    {
        Demo11 dm = new Demo11();
    }
}

```

### Output:

It is an instance block

### Instance block limitations:

- The limitations of instance block we can't pass any parameters

**Case1:** If a class contains instance block, static block and a constructor which one will execute first?

```

package com.sgtesting.instanceblock;

class Demo12
{
    Demo12()
    {
        System.out.println("It is a constructor");
    }

    {
        System.out.println("It is an instance block");
    }

    static
    {
        System.out.println("It is a static block");
    }
}

public class InstanceBlockDemo
{
    public static void main (String [] args)
    {
        Demo12 dm = new Demo12();
    }
}

```

### Output:

It is a static block  
It is an instance block  
It is a constructor

**Case2:** If you can write a multiple instance block what is the order of execution?

```

package com.sgtesting.instanceblock;

class Demo13
{
    {
        System.out.println("It is a first instance block");
    }

    {
        System.out.println("It is a second instance block");
    }

    {
        System.out.println("It is a third instance block");
    }
}

public class InstanceBlockDemo2
{
    public static void main (String [] args)
    {
        Demo13 dm = new Demo13();
    }
}

```

#### Output:

```

It is a first instance block
It is a second instance block
It is a third instance block

```

#### Note:

- No need to create a method in static
- When you create an object to the class
  - Memory will run
  - Instance block will run
  - Static block will run
  - Corresponding constructor will run

#### Outer and Inner class:

Outer and inner class are also called as nested classes, the members of the outer class can be directly accessed by the inner class but the members of the inner class can't be accessed by the outer class directly, in order to access we have to create inner class object in an outer class

#### Example:

```

package com.sgtesting.outerandinnerclass;

class Outer
{
    String firstname;
    Inner in = new Inner ();
    void displayAge ()
    {
        in.age=29;
        System.out.println("Age: "+in.age);
    }
}

```

```

class Inner
{
    int age;
    void showFN ()
    {
        firstname="Ranav";
        System.out.println("First Name: "+firstname);
    }
}

public class OuterAndInnerClassDemo
{
    public static void main (String [] args)
    {
        Outer out = new Outer ();
        out.displayAge ();
        out.in.showFN ();
    }
}

```

### Output:

```

Age: 29
First Name: Ranav

```

### Encapsulation:

Encapsulation means data hiding, the data can be hidden by using private access specifier the hidden data can be accessible in other classes by using setter and getter methods.

### Example:

```

package com.sgtesting.encapsulation;

class Bank
{
    private String bankname;
    private int accountNo;

    public void setBankName (String bankname)
    {
        this.bankname =bankname;
    }

    public String getBankName ()
    {
        return bankname;
    }

    public void setaccountNo (int accountNo)
    {
        this.accountNo =accountNo;
    }

    public int getAccountNo ()
    {
        return accountNo;
    }
}

```

```

public class EncapsulationBankDemo
{
    public static void main (String [] args)
    {
        Bank ba = new Bank ();
        ba.setBankName("HDFC Bank");
        String v1=ba.getBankName();
        System.out.println(v1);

        ba.setaccountNo(1234456789);
        int v2=ba.getAccountNo();
        System.out.println(v2);
    }
}

```

### Output:

```

HDFC Bank
1234456789

```

### Singleton design pattern:

Singleton design pattern indicates the creation and usage of only one object or instance, here if you can create more than one object instead of creating the new object it utilizes the existing object

### Example:

```

package com.sgtesting.encapsulation;

class BasicMaths
{
    void addition (int x, int y)
    {
        int res = (x+y);
        System.out.println("Addition result: "+res);
    }

    void multiplication (int x, int y)
    {
        int res = (x*y);
        System.out.println("Multiplication result: "+res);
    }
}

public class BeforeSingletonDesignPatternDemo
{
    public static void main (String [] args)
    {
        BasicMaths o1 = new BasicMaths();
        o1.addition(10, 100);
        o1.multiplication(20, 10);

        BasicMaths o2 = new BasicMaths ();
        o2.addition (20, 150);
        o2.multiplication (20, 40);

        BasicMaths o3 = new BasicMaths();
    }
}

```

```

        o3.addition (30, 90);
        o3.multiplication (30, 15);
    }
}

```

### Output:

```

Addition result: 110
Multiplication result: 200
Addition result: 170
Multiplication result: 800
Addition result: 120
Multiplication result: 450

```

In the above example o2 and o3 objects are not required the same actions, we can able to achieve by using first object o1.

Currently if you can create n number of objects the class allows to create n number of object, to avoid unnecessary creation of objects are instances we have to apply singleton design pattern on existing class

### How to apply singleton design pattern?

Ans:

**Step1:** define constructor as private, so it avoids creation of object/instance in a other classes

**Step2:** create a static method, the static method has to return the object (object of the same class)

**Step3:** apply a condition in a static method if object is null then only it has to create new object otherwise it has to return the same existing object

```

package com.sgtesting.encapsulation;

class BasicMaths2
{
    static BasicMaths2 obj = null;
    private BasicMaths2()
    {
        // private constructor
    }

    void addition (int x, int y)
    {
        int res = (x+y);
        System.out.println("Addition result: "+res);
    }

    void multiplication (int x, int y)
    {
        int res = (x*y);
        System.out.println("Multiplication result "+res);
    }

    public static BasicMaths2 getInstance ()
    {
        if(obj==null)
        {
            obj=new BasicMaths2();
        }
        return obj;
    }
}

public class AfterApplyingSingletonDesignPatternDemo

```

```

{
    public static void main (String [] args)
    {
        BasicMaths2 o1 = BasicMaths2.getInstance();
        o1.addition(20, 80);
        o1.multiplication(20, 20);

        BasicMaths2 o2 = BasicMaths2.getInstance();
        o2.addition(100, 200);
        o2.multiplication(15, 100);

        BasicMaths2 o3 = BasicMaths2.getInstance();
        o3.addition(500, 100);
        o3.multiplication(25, 10);

        if (o1==o2 && o2==o3)
        {
            System.out.println("Achieved singleton design pattern");
        }
        else
        {
            System.out.println("Not Achieved singleton design pattern");
        }
    }
}

```

#### Output:

```

Addition result: 100
Multiplication result 400
Addition result: 300
Multiplication result 1500
Addition result: 600
Multiplication result 250
Achieved singleton design pattern

```

#### Debugging in Eclipse:

-----

Following things should be aware of:

- **Toggle break point:** during debug, it is used to pause/halt the test script execution, the importance of toggle break point comes into picture only when you are running your code in debug mode

**Note:** running code in normal mode will not pause/ halt for toggle break point

- **Debug as:** it is one of the run modes in java, during this mode the execution pauses at every toggle breakpoint
- **Step into (f5):** it is used to run the java program line by line, f5 will enter the block if it is used in the method call statement, object creation statement etc.
- **Step over (f6):** it is used to run the java program line by line, but f6 will execute the whole block at a time without entering inside it especially at method call statement, object creation statement etc
- **Step out (f7):** if you entered the block accidentally and you want to come out of the block by executing complete block then press f7
- **Normal mode (f8):** running the execution from debug mode to normal mode we use f8 key, where on f8 will halt/pause at every toggle break points
- **Watch:** it is used to see or inspect the runtime values at runtime

**Note:** to apply the watch, we must select the variable/object/line etc first and then right click and select watch upon applying watch, the runtime value stored in the variable/object/line etc will be displayed in expression tab during runtime

- Alternative for watch is select the variable/object/line and then press “ctrl+shift+i”

### Debug viewer:

-----

- **Variable:**
  - This selection will display all the variables which are executed or located into the memory will be displayed
  - We can modify the variable values during runtime in the variable section
- **Break point:**
  - This section will show all the break points applied and its line number, from here only you can enable or disable the break point just by selecting/unselecting the checkbox
- **Expression:**
  - All the watch values can be seen at expression section

### Note:

! when we declare parameterized constructor the default constructor will gone

! when we run 1<sup>st</sup> main method will run because main method is a thread and entry point

### Inheritance:

-----

It provides reusability of the code

Or

It deals about parent class and child class relationship in java

Or

How the child class inherits the parent class behaviors

### Note:

- The parent class is also called as super class/base class
- The child class is also called as derived class/sub class

### Syntax:

```
Class A
{
}
```

```
Class B extends class A
{
}
```

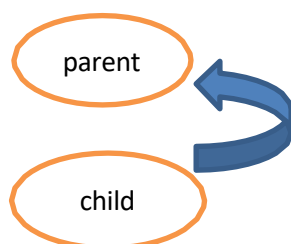
### Types of inheritance:

- Simple inheritance
- Multi-level inheritance
- Hierarchical inheritance
- Hybrid inheritance
- Multiple inheritance

### Simple inheritance:

-----

A single child class extends parent class



Child is referring to parent because child is depending on parent/ dependency inversion principle



### Example:

```
package com.sgtesting.inheritancedemo;

class Maths1
{
    void addition (int x, int y)
    {
        System.out.println("Addition result: "+(x+y));
    }
}

class Maths2 extends Maths1
{
    void subtraction (int a, int b)
    {
        System.out.println("Subtraction result: "+(a-b));
    }
}

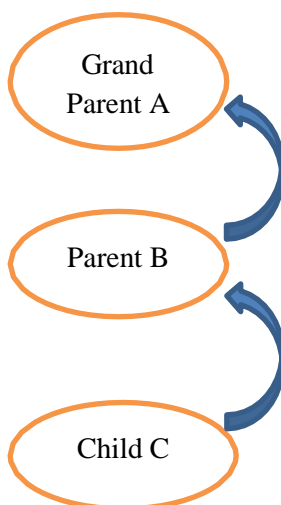
public class SimpleInheritanceDemo
{
    public static void main (String [] args)
    {
        Maths2 mat = new Maths2();
        mat.addition(100, 50);
        mat.subtraction(100, 40);
    }
}
```

### Output:

```
Addition result: 150
Subtraction result: 60
```

### Multi-level inheritance:

-----  
A child class (C) extends its parent (B) class and the parent (B) class has its another Parent (A) class.



### Example:

```
package com.sgtesting.inheritancedemo;
```

```

class GrandParent
{
    void addition (int x, int y)
    {
        System.out.println("Addition result: "+(x+y));
    }
}

class Parent extends GrandParent
{
    void subtraction (int a, int b)
    {
        System.out.println("Subtraction result: "+(a-b));
    }
}

class Child extends Parent
{
    void division (int x, int y)
    {
        System.out.println("Division result: "+(x/y));
    }
}

public class MultiLevelInheritanceDemo
{
    public static void main (String [] args)
    {
        Child chi = new Child ();
        chi.addition(100, 20);
        chi.subtraction(70, 55);
        chi.division(50, 5);
    }
}

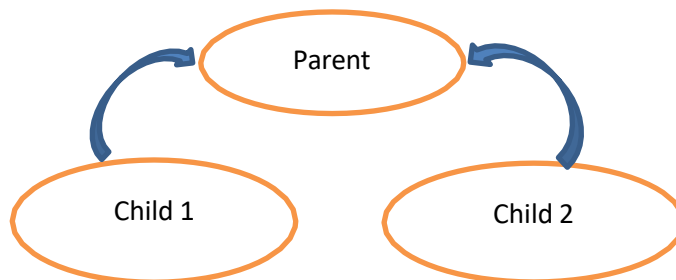
```

### Output:

Addition result: 120  
 Subtraction result: 15  
 Division result: 10

### Hierarchical inheritance:

-----  
 A multiple sub classes extends one super/parent class



### Example:

```

package com.sgtesting.inheritancedemo;

```

```

class Parent1
{
    void addition (int x, int y)
    {
        System.out.println("Addition Result: "+(x+y));
    }
}

class Child1 extends Parent1
{
    void subtraction (int a, int b)
    {
        System.out.println("Subtraction result: "+(a-b));
    }
}

class Child2 extends Parent1
{
    void division (int x, int y)
    {
        System.out.println("Division result: "+(x/y));
    }
}

public class HierarchicalInheritanceDemo
{
    public static void main(String[] args)
    {
        Child2 chi = new Child2();
        chi.addition(10, 200);
        chi.division(50, 5);

        Child1 chi1 = new Child1();
        chi1.addition(200, 50);
        chi1.subtraction(200, 100);
    }
}

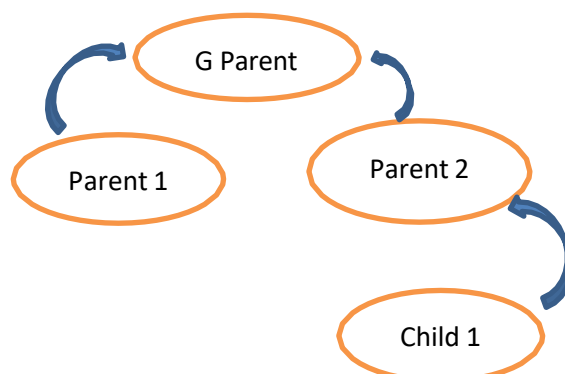
```

### Output:

Addition Result: 210  
 Division result: 10  
 Addition Result: 250  
 Subtraction result: 100

### Hybrid inheritance:

It is a combination of simple, hierarchical and multi-level inheritance



**Example:**

```
package com.sgtesting.inheritanceDemo;

class GrandParent1
{
    void addition (int x, int y)
    {
        System.out.println("Addition Result: "+(x+y));
    }
}

class Parent01 extends GrandParent1
{
    void subtraction (int a, int b)
    {
        System.out.println("Subtraction result: "+(a-b));
    }
}

class Parent02 extends GrandParent1
{
    void division (int x, int y)
    {
        System.out.println("Division Result: "+(x/y));
    }
}

class Child01 extends Parent02
{
    void multiplication (int x, int y)
    {
        System.out.println("Multiplication result: "+(x*y));
    }
}

public class HybridInheritanceDemo
{
    public static void main (String [] args)
    {
        Parent01 par = new Parent01();
        par.addition(200, 50);
        par.subtraction(200, 100);

        Child01 chi = new Child01();
        chi.addition(200, 100);
        chi.division(200, 20);
        chi.multiplication(10, 20);
    }
}
```

**Output:**

```
Addition Result: 250
Subtraction result: 100
Addition Result: 300
Division Result: 10
Multiplication result: 200
```

## Special cases in inheritance:

### Case1: super class contains private members (private variables, private methods)

**Ans:** The private members of the class can be accessible only within that class only, if a class contains private members based on the sub class object, we can't access private members of the class

### Case2: If super class contains parameterized constructor

**Example:**

```
package com.sgtesting.inheritancedemo;

class Student
{
    Student (String fname, String location)
    {
        System.out.println("First Name: "+fname);
        System.out.println("Location: "+location);
    }
}

class Library extends Student
{
    Library ()
    {
        super ("Ranav", "Tumakuru");
    }
}

public class InheritanceCase2Demo
{
    public static void main (String [] args)
    {
        Library lib = new Library ();
    }
}
```

### Output:

```
First Name: Ranav
Location: Tumakuru
```

### Super keyword:

-----  
super keyword with constructor execution can be used in sub class

while writing super keyword with constructor execution it has to satisfy the below 2 rules

- super keyword with constructor execution always must be the first statement within the body of the constructor statement or method
- super keyword with constructor execution always must be the only one statement within the body of the constructor or method

### extension of Case 2:

#### Case2.1:

```
package com.sgtesting.inheritancedemo;

class Student
```

```

{
    Student (String fname, String location)
    {
        System.out.println("First Name: "+fname);
        System.out.println("Location: "+location);
    }
}

class Library extends Student
{
    Library (String fn, String loc)
    {
        super (fn, loc);
    }
}

public class InheritanceCase2Demo
{
    public static void main (String [] args)
    {
        Library lib = new Library ("Ranav", "Tumakuru");
    }
}

```

#### Output:

```

First Name: Ranav
Location: Tumakuru

```

#### Extension of Case2.1:

#### Case2.2:

```

package com.sgtesting.inheritancedemo;

class Student
{
    Student (String fname, String location)
    {
        System.out.println("First Name: "+fname);
        System.out.println("Location: "+location);
    }
}

class Library extends Student
{
    String bookname;
    Library (String fn, String loc, String bookname)
    {
        super(fn, loc);
        this.bookname=bookname;
        System.out.println("Book Name: "+bookname);
    }
}

public class InheritanceCase2Demo
{
    public static void main (String [] args)
    {
        Library lib = new Library ("Ranav", "Tumakuru", "Core Java");
    }
}

```

```
    }  
}
```

**Output:**

First Name: Ranav  
Location: Tumakuru  
Book Name: Core Java

**Case3: In multi-level inheritance find out the order of the execution of constructor**

**Example:**

```
package com.sgtesting.inheritancedemo;  
  
class AA  
{  
    AA ()  
    {  
        System.out.println("It is a AA's class constructor");  
    }  
}  
  
class BB extends AA  
{  
    BB ()  
    {  
        Super ();  
        System.out.println("It is a BB's class constructor");  
    }  
}  
  
class CC extends BB  
{  
    CC ()  
    {  
        Super ();  
        System.out.println("It is a CC's class constructor");  
    }  
}  
  
public class InheritanceCase3Demo  
{  
    public static void main (String [] args)  
    {  
        CC obj = new CC ();  
    }  
}
```

**Output:**

It is a AA's class constructor  
It is a BB's class constructor  
It is a CC's class constructor

**Case4: If super class and sub class having same method name with same signature (Parameters)**

**Example:**

```
package com.sgtesting.inheritancedemo;
```

```

class Employee
{
    void showBonus (int percentage)
    {
        double sal, bonus;
        sal = 50000;
        bonus = (sal*percentage)/100;
        System.out.println("Bonus of the employee: "+bonus);
    }
}

class Department extends Employee
{
    Department (int percentage)
    {
        super.showBonus (percentage);
    }

    void showBonus (int percentage)
    {
        System.out.println("Percentage of bonus for department: "+percentage);
    }
}

public class SameMethodNameWithSameSignatureDemoCase4
{
    public static void main (String [] args)
    {
        Department dep = new Department (10);
        dep.showBonus (20);
    }
}

```

#### Output:

```

Bonus of the employee: 5000.0
Percentage of bonus for department: 20

```

#### Case5: If super class and sub class contains same variable name with same datatype

**Ans:** here the sub class variable hides the super class variable

#### Example:

```

package com.sgtesting.inheritanceDemo;

class State
{
    String name;
    void show ()
    {
        System.out.println("In super class name value: "+name);
    }
}

class Country extends State
{
    String name;
    Country (String name1, String name)
    {

```



```

        super.name=name1;
        this.name=name;
    }

    void display ()
    {
        System.out.println("In sub class name value: "+name);
    }
}

public class SameVariableWithSameDatatypeDemoCase5
{
    public static void main (String [] args)
    {
        Country o = new Country ("Karnataka", "Bharath");
        o.show();
        o.display();
    }
}

```

### Output:

```

In super class name value: Karnataka
In sub class name value: Bharath

```

### Assignment:

- 1) WAP to describe the multi-level inheritance?
- 2) WAP to describe hierarchical inheritance?
- 3) WAP to describe hybrid inheritance?
- 4) WAP for multi-level inheritance in which super class contains parameterized constructor?
- 5) WAP for multi-level inheritance in which super class contains constructor overloading?
- 6) WAP for hierarchical inheritance in which super class contains parameterized constructor?
- 7) WAP for hybrid inheritance in which super class contains constructor overloading?
- 8) WAP for multi-level inheritance in which each class contains same method name with signature?
- 9) WAP for multi-level inheritance in which each class contains same variable name with datatype?

### Note:

- Static components never handled in inheritance

### Abstract class:

-----

Abstract class provides partial implementation, if 'n' number of sub classes is required to implement same behavior, in this case we have to prefer an abstract class

### Abstract class have the following features:

- If a class contains abstract methods in this case the class must be specified as an abstract class
- If a class doesn't contain abstract methods even though class can be specified as an abstract
- An abstract class may contain abstract methods or instance members or static members individually or together
- If a sub class extends an abstract class the sub class responsibility has to implement all the abstract method from the abstract super class

Suppose if sub class missed to implement any abstract method from abstract super class in this case sub class also becomes an abstract class

- We can't create object for an abstract class but we can provide object reference, in this case based on the reference variable we can access members which are available in the abstract class alone

**Example:**

```
package com.sgtesting.abstractclassdemo;

abstract class College
{
    abstract void showCollegeName (String name);
    abstract void showCollegeLocation (String location);
    void displayCourses (String courses [])
    {
        System.out.println("Courses are..... ");
        for (int i=0; i<courses.length; i++)
        {
            System.out.println("Course Name: "+courses[i]);
        }
    }
}

class EngineeringCollege extends College
{
    void showCollegeName (String name)
    {
        System.out.println("Engineering College Name: "+name);
    }

    void showCollegeLocation (String location)
    {
        System.out.println("College Location: "+location);
    }

    void displaycityName (String name)
    {
        System.out.println("City Name: "+name);
    }
}

public class AbstractMainDemo
{
    public static void main (String [] args)
    {
        EngineeringCollege obj=new EngineeringCollege ();
        obj.showCollegeName("CIT");
        obj.showCollegeLocation("Gubbi, Tumakuru");
        String a[]={ "ME", "CSE", "ECE", "EEE", "Civil", "ISE" };
        obj.displayCourses(a);
        obj.displaycityName("Tumakuru");
    }
}
```

**Output:**

```
Engineering College Name: CIT
College Location: Gubbi, Tumakuru
Courses are.....
    Course Name: ME
    Course Name: CSE
    Course Name: ECE
    Course Name: EEE
    Course Name: Civil
```

Course Name: ISE  
City Name: Tumakuru

**Case1:** create instance or object based on the abstract class reference

```
package com.sgtesting.abstractclassdemo;

abstract class College
{
    abstract void showCollegeName (String name);
    abstract void showCollegeLocation (String location);
    void displayCourses (String courses [])
    {
        System.out.println("Courses are..... ");
        for (int i=0; i<courses.length; i++)
        {
            System.out.println("Course Name: "+courses[i]);
        }
    }
}

class EngineeringCollege extends College
{
    void showCollegeName (String name)
    {
        System.out.println("Engineering College Name: "+name);
    }

    void showCollegeLocation (String location)
    {
        System.out.println("College Location: "+location);
    }

    void displaycityName (String name)
    {
        System.out.println("City Name: "+name);
    }
}

public class AbstractMainDemo
{
    public static void main (String [] args)
    {
        College obj=new EngineeringCollege ();
        obj.showCollegeName("CIT");
        obj.showCollegeLocation("Gubbi, Tumakuru");
        String a[]= {"ME", "CSE", "ECE", "EEE", "Civil", "ISE"};
        obj.displayCourses (a);
    }
}
```

**Output:**

```
Engineering College Name: CIT
College Location: Gubbi, Tumakuru
Courses are.....
    Course Name: ME
    Course Name: CSE
    Course Name: ECE
```

Course Name: EEE  
Course Name: Civil  
Course Name: ISE  
City Name: Tumakuru

**Note:**

- Here we can't access to create object for subclass because we are using reference from abstract class

**Assignment:**

- 1) If abstract class contains only static method, WAP to execute static method from abstract class.
- 2) If abstract class contains static block alone, WAP to execute static block from abstract class.
- 3) In a multilevel inheritance superclass is an abstract class and it contains parameterized constructor.
- 4) In multilevel inheritance superclass contains an abstract method based on the sub class object execute all the members.
- 5) In a hybrid inheritance super class contains abstract method, based on the subclass object execute all the methods.
- 6) If abstract class contains only the instance methods, WAP to execute the instance methods.

**Access Specifiers:**

-----

Access specifier or access modifiers are used to specify the boundary or range of a specific member (class, variable, method and constructor).

**Java supports the following access specifier:**

- a) private
- b) package private
- c) protected
- d) public

**private:** if any members of the class are private those members can be accessible only within the same class alone

**package private (default access specifier):** if any member of the class is package private access specifier in this case these members can be accessible within the same package alone

**protected:** if any members of the class are having access specifier as protected those members can be accessible in the same package and as well as in other package class alone can be accessible

**public:** if any members of the class are public those can be accessible irrespective of any package

**example:**

```
package p1;

public class Protection
{
    private int private_a=100;
    protected int protected_b=200;
    int default_c=300;
    public int public_d=400;

    public Protection ()
    {
        System.out.println("It is a protection class constructor");
        System.out.println("private: "+private_a);
    }
}
```

```

        System.out.println("protected: "+protected_b);
        System.out.println("default: "+default_c);
        System.out.println("public: "+public_d);
        System.out.println("*****");
    }
}

----->
package p1;

public class SubClassInP1Package extends Protection
{
    public SubClassInP1Package ()
    {
        System.out.println("It is a SubClassInP2Package class constructor");
        // System.out.println("private: "+private_a);
        System.out.println("protected: "+protected_b);
        System.out.println("default: "+default_c);
        System.out.println("public: "+public_d);
        System.out.println("*****");
    }
}

----->
package p1;

public class IndependentClassP1Package
{
    Protection o=new Protection ();
    public IndependentClassP1Package ()
    {
        System.out.println("It is a Independent class ");
        // System.out.println("private: "+o.private_a);
        System.out.println("protected: "+o.protected_b);
        System.out.println("default: "+o.default_c);
        System.out.println("public: "+o.public_d);
        System.out.println("*****");
    }
}

```

|                  | same protection class<br>in package p1 | sub class of class<br>protection in p1 class | independent class in<br>p1 package |
|------------------|----------------------------------------|----------------------------------------------|------------------------------------|
| <b>private</b>   | yes                                    | no                                           | no                                 |
| <b>protected</b> | yes                                    | yes                                          | yes                                |
| <b>default</b>   | yes                                    | yes                                          | yes                                |
| <b>public</b>    | yes                                    | yes                                          | yes                                |

#### P2 package (another package):

```

----->
package p2;

import p1.Protection;
public class SubClassInP2Package extends Protection
{

```

```

        public SubClassInP2Package ()
        {
            System.out.println("It is a SubClassInP2Package class constructor");
            // System.out.println("private: "+private_a);
            System.out.println("protected: "+protected_b);
            // System.out.println("default: "+default_c);
            System.out.println("public: "+public_d);
            System.out.println("*****");
        }
    }
}

----->
package p2;

import p1.Protection;

public class IndependentClassP2Package
{
    Protection o=new Protection ();
    public IndependentClassP2Package ()
    {
        System.out.println("It is a Independent class ");
        // System.out.println("private: "+o.private_a);
        // System.out.println("protected: "+o.protected_b);
        // System.out.println("default: "+o.default_c);
        System.out.println("public: "+o.public_d);
        System.out.println("*****");
    }
}

```

#### mainpackage execution:

-----

```

--->
package mainpackage;

import p1.IndependentClassP1Package;
import p1.Protection;
import p1.SubClassInP1Package;
import p2.IndependentClassP2Package;
import p2.SubClassInP2Package;

public class MainDemo
{
    public static void main (String [] args)
    {
        // Execute Protection class constructor
        Protection o=new Protection ();

        // Execute SubClassInP1Package class constructor
        SubClassInP1Package o1=new SubClassInP1Package ();

        // Execute IndependentClassP1Package class constructor
        IndependentClassP1Package o2=new IndependentClassP1Package ();

        // Execute SubClassInP2Package class constructor
        SubClassInP2Package o3=new SubClassInP2Package ();
    }
}

```

```

// Execute IndependentClassP2Package class constructor
IndependentClassP2Package o4=new IndependentClassP2Package();
}
}

```

|                  | sub class of class<br>protection in p2 class | independent class in<br>p2 package |
|------------------|----------------------------------------------|------------------------------------|
| <b>private</b>   | no                                           | no                                 |
| <b>protected</b> | yes                                          | no                                 |
| <b>default</b>   | no                                           | no                                 |
| <b>public</b>    | yes                                          | yes                                |

### final Keyword:

-----

final keyword indicates the definition of constant on core java

The final keyword can be used along with:

- a) variables
- b) methods
- c) class

variables:

-----

Example:

```

final int a=120;
final static int b=200;

```

the value stored in final variables never change throughout the execution of the program

with methods:

-----

The final method of a superclass can't be overridden by the subclass method

Example:

```

class Demo1
{
    final void show (String name)
    {
        System.out.println("In superclass name: "+name);
    }
}

class Demo2 extends Demo1
{
    // compilation error because it unable to override the superclass method that is final
}

public class FinalDemo
{
    public static void main (String [] args)
    {
    }
}

```

Case3: the final can't be extended by any other class if any class tries to extend final super class it provides compilation error in extend keyword

### interface (java 1.7):

-----  
interface provides complete implementation

#### The interface has the following features:

- a) interface contains only the abstract method
- b) in interface for abstract methods, abstract keyword is optional
- c) the variables available in interface by default final and static
- d) A single sub class can implement more than one interfaces that indicates multiple inheritance
- e) If a sub class implement an interface the sub class responsibility it has to implement all abstract methods from interface, if sub class has missed to implement any abstract method from interface in this case sub class also becomes an abstract class
- f) one interface can extend another interface the extending can be simple or multi-level or hybrid etc  
example:

```
class A extends class B
interface A extends interface B
class A implements interface C
```

- g) while sub class implementing abstract method from the interface in sub class for those method it must use public access specifier
- h) we can't create an object or instance for an interface we can provide only object reference, in this case the members which are known by the interface or the members which are available in the interface alone can be accessible

#### Example:

##### Case1:

```
package com.sgtesting.interfacedemo;

interface University
{
    void showUniversityName (String name);
}

class EngineeringCollege implements University
{
    public void showUniversityName (String name)
    {
        System.out.println("University name: "+name);
    }

    void displayLocation (String location)
    {
        System.out.println("Location of Engineering College: "+location);
    }
}

public class InterfaceCase1
{
    public static void main (String [] args)
    {
        EngineeringCollege ee = new EngineeringCollege ();
        ee.showUniversityName("CIT");
        ee.displayLocation("Gubbi");
    }
}
```



**Output:**

University name: CIT  
Location of Engineering College: Gubbi

**Case2: multiple inheritance**

```
package com.sgtesting.interfacedemo;

interface University1
{
    void showUniversityName (String name);
}

interface College
{
    abstract void displayCollegeName (String name);
}

class EngineeringCollege1 implements University1,College
{
    public void showUniversityName(String name)
    {
        System.out.println("University Name: "+name);
    }

    public void displayCollegeName (String name)
    {
        System.out.println("Engineering College Name: "+name);
    }

    void displayLocation(String location)
    {
        System.out.println("Location of Engineering College: "+location);
    }
}

public class InterfaceCase2
{
    public static void main (String [] args)
    {
        EngineeringCollege1 ee = new EngineeringCollege1();
        ee.showUniversityName("VTU");
        ee.displayCollegeName("CIT");
        ee.displayLocation("Gubbi");
    }
}
```

**Output:**

University Name: VTU  
Engineering College Name: CIT  
Location of Engineering College: Gubbi

**Case3: one interface can extend another interface**

```
package com.sgtesting.interfacedemo;

interface University2
```

```

{
    void showUniversityName(String name);
}

interface College2 extends University2
{
    abstract void displayCollegeName(String name);
}

class Engineering1 implements College2
{
    public void showUniversityName(String name)
    {
        System.out.println("University Name: "+name);
    }

    public void displayCollegeName(String name)
    {
        System.out.println("Engineering College Name: "+name);
    }

    void displayLocation(String location)
    {
        System.out.println("Location of Engineering College: "+location);
    }
}

public class InterfaceCase3
{
    public static void main (String [] args)
    {
        Engineering1 ee = new Engineering1();
        ee.showUniversityName("VTU");
        ee.displayCollegeName("CIT");
        ee.displayLocation("Gubbi");
    }
}

```

#### Output:

```

University Name: VTU
Engineering College Name: CIT
Location of Engineering College: Gubbi

```

**Case4: we can't create an object or instance for an interface we can provide only object reference, in this case the members which are known by the interface or the members which are available in the interface alone can be accessible**

```

package com.sgtesting.interfacedemo;

interface University3
{
    void showUniversityName(String name);
}

interface College3

```

```

{
    void displayCollegeName(String name);
}

class EngineeringCollege2 implements University, College3
{
    public void showUniversityName(String name)
    {
        System.out.println("University Name: "+name);
    }

    public void displayCollegeName(String name)
    {
        System.out.println("Engineering College Name: "+name);
    }

    void displayLocation(String location)
    {
        System.out.println("Location of Engineering College: "+location);
    }
}

public class InterfaceCase4
{
    public static void main (String [] args)
    {
        College3 cc = new EngineeringCollege2();
        // cc.showUniversityName("VTU");      can't access
        cc.displayCollegeName("CIT");
        // cc.displayLocation("Gubbi");        can't access
    }
}

```

#### Output:

Engineering College Name: CIT

#### Assignment:

- 1) WAP for multiple inheritance in which a sub class implements more than 2 inheritances?
- 2) WAP for multiple inheritance in which sub class extends on abstract class and implements more than 2 inheritances?  
Like (class A extends class B implements C, D, E)
- 3) WAP to demonstrate one interface extends another interface at multi-level?
- 4) WAP to demonstrate one interface extends another interface at hierarchical level?
- 5) WAP to demonstrate an interface contains variables and execute those variables?

#### Polymorphism:

-----

A super class reference variable can refer to each object based on the super class reference variable we are executing the each class specific members that indicates polymorphism it is also called as dynamic method dispatch

#### Example:

```
package com.sgtesting.polymorphism;
```

```

abstract class Figure
{
    abstract void calculateArea ();
}

class Square extends Figure
{
    void calculateArea ()
    {
        int side=5;
        System.out.println("Area of Square: "+(side*side));
    }
}

class Rectangle extends Figure
{
    void calculateArea ()
    {
        int l, b;
        l=8;
        b=12;
        System.out.println("Area of Rectangle: "+(l*b));
    }
}

class Circle extends Figure
{
    void calculateArea ()
    {
        double pi=3.14;
        double r=2.5;
        System.out.println("Area of Circle: "+(pi*r*r));
    }
}

public class PolymorphismDemo
{
    public static void main (String [] args)
    {
        Figure figure=null;

        Square square=new Square ();
        Rectangle rect=new Rectangle ();
        Circle circle=new Circle ();

        figure=square;
        figure.calculateArea();

        figure=rect;
        figure.calculateArea();

        figure=circle;
        figure.calculateArea();
    }
}

```

**Output:**

Area of Square: 25  
Area of Rectangle: 96  
Area of Circle: 19.625

**Assignment:****1) Please consider any 5 examples on polymorphism?****Exception Handling:**

-----  
There are three types errors available in core java

- a) Syntax errors
- b) Logical errors
- c) Runtime errors

**Syntax errors:**

-----

If any programing line is against the predefined syntax it leads to syntax errors

Note: syntax errors always lead to compilation errors

**Logical errors:**

-----

This kind of errors executing the programs properly but logically it provides invalid result

**Runtime errors:**

-----

The program provides an error due to invalid input so these kinds of errors are called runtime errors

**Case1: Before handling the exception, what can be the default behavior of the exception**

```
package com.sgtesting.exceptiondemo;

class Maths
{
    static void addition (int x, int y)
    {
        int res=(x+y);
        System.out.println("Addition result: "+res);
    }

    static void multiplication (int x, int y)
    {
        int res=(x*y);
        System.out.println("Multiplication result: "+res);
    }

    static void subtraction (int x, int y)
    {
        int res=(x-y);
        System.out.println("Subtraction result: "+res);
    }

    static void division (int x, int y)
    {
        int res=(x/y);
        System.out.println("Division result: "+res);
    }
}
```

```

    }

    static void modulus (int x, int y)
    {
        int res = (x % y);
        System.out.println("Modulus result: "+res);
    }

    static void averagevalue (int x, int y)
    {
        int res=((x+y)/2);
        System.out.println("Average value result: "+res);
    }

    static void isEvenNumber (int x)
    {
        if(x%2==0)
        {
            System.out.println(x+" is Even Number");
        }
    }

    static void isOddNumber (int x)
    {
        if (x%2!= 0)
        {
            System.out.println(x+" is Odd Number");
        }
    }

    static void isPositiveNumber (int x)
    {
        if (x>0)
        {
            System.out.println(x+" is Positive Number");
        }
    }

    static void isNegativeNumber (int x)
    {
        if(x<0)
        {
            System.out.println(x+" is Negative Number");
        }
    }
}

public class BeforeExceptionHandling
{
    public static void main (String [] args)
    {
        Maths.addition(100, 200);
        Maths.multiplication(10, 20);
        Maths.subtraction(200, 50);
        Maths.averagevalue(10, 20);
        Maths.modulus(8, 2);
        Maths.division(10, 0);
    }
}

```

```

        Maths.isEvenNumber(10);
        Maths.isOddNumber(5);
        Maths.isPositiveNumber(2);
        Maths.isNegativeNumber(-5);
    }
}

```

#### Output:

```

Addition result: 300
Multiplication result: 200
Subtraction result: 150
Average value result: 15
Modulus result: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
at com.sgtesting.exceptiondemo.Maths.division(BeforeExceptionHandling.java:25)
at com.sgtesting.exceptiondemo.BeforeExceptionHandling.main(BeforeExceptionHandling.java:83)

```

#### case2: after handling the exception

the exception can be handled by try...catch block since the try catch block handling the exception properly the execution continues with upcoming lines instead of stopping the execution there itself

#### example:

```

package com.sgtesting.exceptiondemo;

class Maths2
{
    static void addition (int x, int y)
    {
        int res=(x+y);
        System.out.println("Addition result: "+res);
    }

    static void multiplication (int x, int y)
    {
        int res=(x*y);
        System.out.println("Multiplication result: "+res);
    }

    static void subtraction (int x, int y)
    {
        int res=(x-y);
        System.out.println("Subtraction result: "+res);
    }

    static void division (int x, int y)
    {
        try
        {
            int res=(x/y);
            System.out.println("Division result: "+res);
        }
        catch (Exception e)
        {
            e.printStackTrace ();
        }
    }
}

```

```

    }

    static void modulus (int x, int y)
    {
        int res= (x % y);
        System.out.println("Modulus result: "+res);
    }

    static void averagevalue (int x, int y)
    {
        int res=((x+y)/2);
        System.out.println("Average value result: "+res);
    }

    static void isEvenNumber (int x)
    {
        if(x%2==0)
        {
            System.out.println(x+" is Even Number");
        }
    }

    static void isOddNumber (int x)
    {
        if (x%2!=0)
        {
            System.out.println(x+" is Odd Number");
        }
    }

    static void isPositiveNumber (int x)
    {
        if(x>0)
        {
            System.out.println(x+" is Positive Number");
        }
    }

    static void isNegativeNumber (int x)
    {
        if(x<0)
        {
            System.out.println(x+" is Negative Number");
        }
    }
}

public class AfterHandlingException
{
    public static void main (String [] args)
    {
        Maths2.addition(100, 200);
        Maths2.multiplication(10, 20);
        Maths2.subtraction(200, 50);
        Maths2.averagevalue(10, 20);
        Maths2.modulus(8, 2);
        Maths2.division(10, 0);
    }
}

```



```

        Maths2.isEvenNumber(10);
        Maths2.isOddNumber(5);
        Maths2.isPositiveNumber(2);
        Maths2.isNegativeNumber(-5);
    }

}

```

### Output:

```

Addition result: 300
Multiplication result: 200
Subtraction result: 150
Average value result: 15
Modulus result: 0
java.lang.ArithmeticException: / by zero
at com.sgtesting.exceptiondemo.Maths2.division(AfterHandlingException.java:27)
at com.sgtesting.exceptiondemo.AfterHandlingException.main(AfterHandlingException.java:90)
10 is Even Number
5 is Odd Number
2 is Positive Number
-5 is Negative Number

```

### Exception handling keyword in java:

-----

- a) Try block
- b) Catch block
- c) Finally
- d) Throw
- e) Throws

### Try block:

-----

This block should contain only te executable lines

### Note:

- as per the coding standard variable declaration or specifying the reference variables we should do outside of the try block
- assume if the try block contains 10 lines of code, if execution as araised at 5<sup>th</sup> line in this case the execution comes out from the try block, at any point the execution never return back to try block

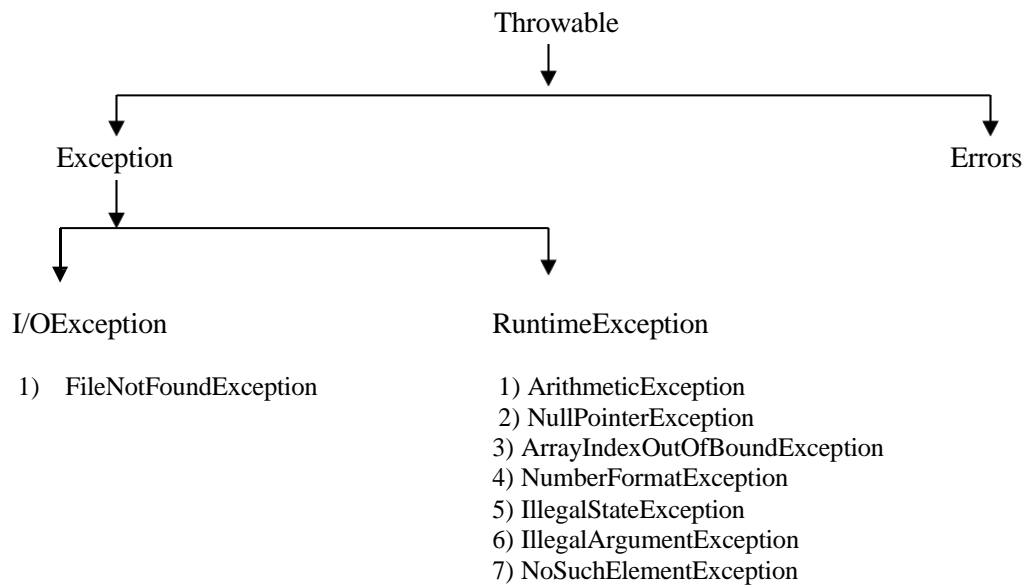
### catch block:

-----

The catch block catches the exception thrown by the try block

Here the exception class specified in the catch block must with exception thrown by the try block, then only it catches the exception

## Exception class Hierarchy:



**Case1:** the exception thrown by the try block, must match with exception class specified in the catch block, in these criteria it will handle the exception

```
static void division (int x, int y)
{
    try
    {
        int res=(x/y);
        System.out.println("Division result: "+res);
    }
    catch (ArithmeticException e)
    {
        e.printStackTrace ();
    }
}
```

**Output:**

```
Addition result: 300
Multiplication result: 200
Subtraction result: 150
Average value result: 15
Modulus result: 0
java.lang.ArithmeticException: / by zero
at com.sgtesting.exceptiondemo.Maths2.division(AfterHandlingException.java:27)
at com.sgtesting.exceptiondemo.ExceptionDemo.main(ExceptionDemo.java:90)
10 is Even Number
5 is Odd Number
2 is Positive Number
-5 is Negative Number
```

**Case2:** if exception thrown by the try block doesn't match with exception class specified in the catch block in this case it doesn't handle the exception

```
static void division(int x, int y)
{
    try
```

```

    {
        int res=(x/y);
        System.out.println("Division result: "+res);
    }
    catch (NullPointerException e)
    {
        e.printStackTrace();
    }
}

```

**Output:**

Addition result: 300  
 Multiplication result: 200  
 Subtraction result: 150  
 Average value result: 15  
 Modulus result: 0  
[java.lang.ArithmeticException: / by zero](#)  
 at com.sgtesting.exceptiondemo.Maths2.division([AfterHandlingException.java:27](#))  
 at com.sgtesting.exceptiondemo.ExceptionDemo.main([ExceptionDemo.java:90](#))

**Case3: Specifies super class exception in catch block, it can handle or catch any type of exception which has thrown by the try block**

```

static void division(int x, int y)
{
    try
    {
        int res=(x/y);
        System.out.println("Division result: "+res);
    }
    catch (Throwable e)
    {
        e.printStackTrace();
    }
}

```

**Output:**

Addition result: 300  
 Multiplication result: 200  
 Subtraction result: 150  
 Average value result: 15  
 Modulus result: 0  
[java.lang.ArithmeticException: / by zero](#)  
 at com.sgtesting.exceptiondemo.Maths3.division([ExceptionDemo.java:27](#))  
 at com.sgtesting.exceptiondemo.ExceptionDemo.main([ExceptionDemo.java:90](#))  
 10 is Even Number  
 5 is Odd Number  
 2 is Positive Number  
 -5 is Negative Number

**Case4: a single try block can be followed by multiple catch block, in this case the catch block which contains super class always must be last catch block**

```

static void division(int x, int y)
{
    try
    {
        int res=(x/y);
    }
}

```

```

        System.out.println("Division result: "+res);
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

```

#### Output:

```

Addition result: 300
Multiplication result: 200
Subtraction result: 150
Average value result: 15
Modulus result: 0
java.lang.ArithmeticException: / by zero
at com.sgtesting.exceptiondemo.Maths3.division(ExceptionDemo.java:27)
at com.sgtesting.exceptiondemo.ExceptionDemo.main(ExceptionDemo.java:90)
10 is Even Number
5 is Odd Number
2 is Positive Number
-5 is Negative Number

```

#### Finally block:

-----

It is an optional block, irrespective of exceptions araised or not this block executes always

#### Example:

```

static void division(int x, int y)
{
    try
    {
        int res=(x/y);
        System.out.println("Division result: "+res);
    }
    catch (ArithmeticException e)
    {
        e.printStackTrace();
    }
    finally
    {
        System.out.println("This block executes always!!!");
    }
}

```

#### Output:

```

Addition result: 300
Multiplication result: 200
Subtraction result: 150
Average value result: 15
Modulus result: 0
java.lang.ArithmeticException: / by zero
This block executes always!!!
at com.sgtesting.exceptiondemo.Maths3.division(ExceptionDemo.java:27)
at com.sgtesting.exceptiondemo.ExceptionDemo.main(ExceptionDemo.java:94)
10 is Even Number
5 is Odd Number
2 is Positive Number
-5 is Negative Number

```

## Throw and Throws:

---

**Throw:** by using “throw” keyword and by using inbuilt exception class we can throw user defined exception messages

**Throws:** the “throws” keyword throws the exception at method level if the same method as called by any other program in this case that should handle the exception by applying “try catch” block

### Example:

```
package com.sgtesting.exceptiondemo;

class Sample
{
    static int getCharactorCount (String str) throws Exception
    {
        if(str==null)
        {
            throw new Exception ("The input is null, please provide the valid String input!!!");
        }
        return str.length();
    }
}

public class ExceptionDemo
{
    public static void main (String [] args)
    {
        try
        {
            int v1=Sample.getCharactorCount("React");
            System.out.println("# of Chars: "+v1);

            int v2=Sample.getCharactorCount(null);
            System.out.println("# of Chars: "+v2);

        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
}
```

### Output:

```
# of Chars: 5
The input is null, please provide the valid String input!!!
```

## Part3: Core Java

### Strings:

String is an immutable for each modification on the string object it creates separate objects in the memory

### When to go for Strings concept?

**Ans:** if the content is fixed and won't change frequently then go for String concept

**Note:** String concept is also thread safe bcz we can't modify the object if we try to modify a new object will create so all immutable objects are thread safe (even wrapper classes)

### Example:

```
package com.sgtesting.stringdemo;

public class StringDemo
{
    public static void main(String[] args)
    {
        String s="Java";
        System.out.println(s);

        s.concat("Programming");
        System.out.println(s);

        s.concat("Language");
        System.out.println(s);
    }
}
```

### Output:

```
Java
Java
Java
```

### Program Explanation:



### Constant pool:

“It is a memory section which is available in a heap memory, it stores only the objects which have created by String literal approach”, at any point constant pool never accept duplicate String object if you trying to create duplicate object instead of creating new one it provides casting object reference

If we can create a String object based on String literals in this case it creates String object in a constant pool

**Note:**

- If you can create a String object based on the new operator in this case it creates String objects in the heap memory
- In heap memory only reference variables are saved

**Example:**

```
package com.sgtesting.stringdemo;

public class StringDemo
{
    public static void main(String[] args)
    {
        String s1="JAVA";
        String s2="java";
        String s3="JAVA";

        String s4=new String("JAVA");
        String s5=new String("java");
        String s6=new String("JAVA");

        System.out.println("(s1==s2): "+(s1==s2));
        System.out.println("(s1==s3): "+(s1==s3));
        System.out.println("(s1==s4): "+(s1==s4));
        System.out.println("(s4==s6): "+(s4==s6));
    }
}
```

**Output:**

```
(s1==s2): false
(s1==s3): true
(s1==s4): false
(s4==s6): false
```

**Explanation:**

String s1= "JAVA";

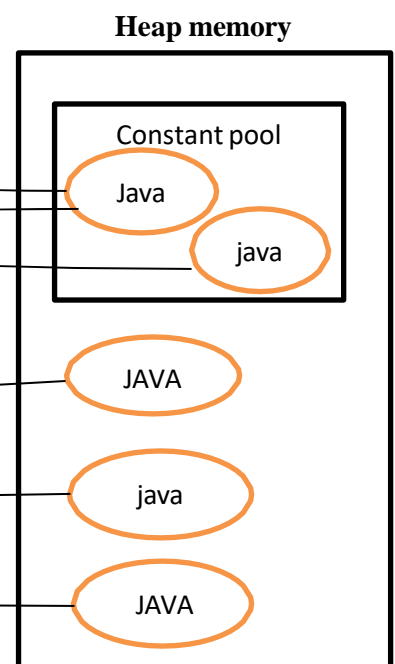
String s2= "java;

String s3= "JAVA";

String s4= new String("JAVA");

String s5=new String("java");

String s6=new String("JAVA");



(s1==s2) → false  
(s1==s3) → true  
(s1==s4) → false  
(s4==s6) → false

### Another Example:

```
String s1=new String ("You cannot change me");  
String s2=new String ("You Cannot change me");  
System.out.println(s1==s2); → false    (because both are not pointing to the same object)
```

```
String s3= "You cannot change me";  
System.out.println(s1==s3); → false    (because both are not pointing to the same object one will in  
                                         heap and other one is in String constant pool (SCP))
```

```
String s4= "You cannot change me";  
System.out.println(s3==s4); → true     (because both are pointing to the same object)
```

```
String s5= "You cannot" + "change me";  
System.out.println(s4==s5); → true
```

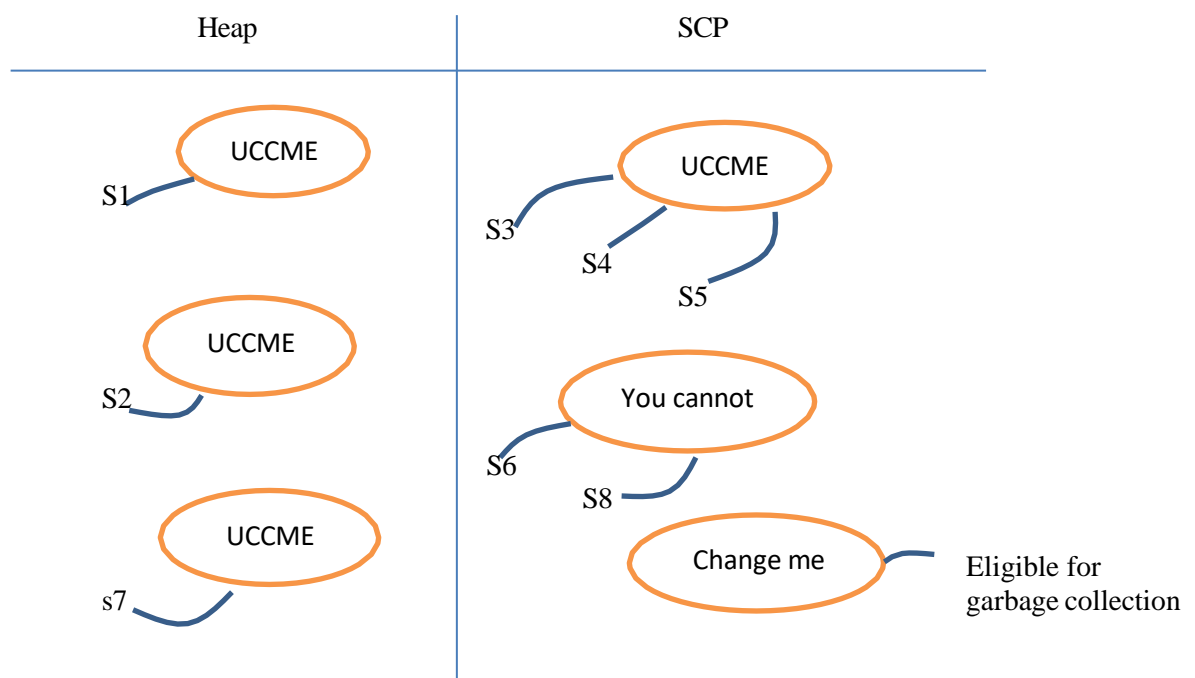
**Note:** if both arguments are constant then JVM will create the object at compile time

```
String s6= "You cannot";  
String s7=s6 + "change me";  
System.out.println(s4==s7); → false
```

**Note:** if at least one variable is there then the operation will be performed at runtime only

```
final String s8= "You cannot";  
String s9=s8 + "change me";  
System.out.println(s4==s9); → true
```

**Note:** every final variable will be replaced by the value at compile time only





### **Advantages/importance of SCP:**

- Instead of creating multiple objects, one object will be referred by multiple so performance and memory utilization will be improved

### **Disadvantage:**

- If anyone tries to change the content it will affect the remaining references to prevent that immutability concept will be introduced

### **FAQ's:**

#### **1) Why SCP concept is available only for string object but not for StringBuffer?**

**Ans:** because most commonly used object in java is String but StringBuffer we may use or may not use so special privilege/memory management is only for String but not for StringBuffer, that's why SCP concept is available only for String object but not for StringBuffer.

**Example:** String is a regular customer to the bar but StringBuffer is an occasional customer so special privilege is not available for StringBuffer

#### **2) Why String objects are immutable whereas StringBuffer objects are mutable?**

**Ans:** because SCP concept is there in String to reuse the same object, by using one object if we are trying to change the content it will affect the remaining references to prevent that immutable concept is required for String object

But not for StringBuffer because reusability concept is not there, for every time new object will be created.

#### **3) In addition to String objects any other objects are immutable in java?**

**Ans:** yes, all wrapper class objects are immutable

### **How to create a String object?**

**Ans:**

These are the following ways to create String object

- 1) `String s = new String ();`  
-----
- 2) `String s = new String ("welcome");`  
-----
- 3) `String s = new String ("welcome");`  
`String s1 = new String (s);`  
-----
- 4) `Char ch [] = {'A', 'B', 'C'};`  
`String s = new String (ch);`  
-----
- 5) `StringBuffer s = new StringBuffer ("welcome");`  
`String s1 = new String (s);`  
-----
- 6) `StringBuilder s = new StringBuilder ("welcome");`  
`String s1 = new String (s);`

### **Important Constructors of String class:**

-----

- 1) `String s=new String ();` → to create an empty string object
- 2) `String s=new String (String literals);`
- 3) `String s=new String (StringBuffer sb);`

- 4) `String s=new String (StringBuilder sb);`
- 5) `String s=new String (char [] ch);`
- 6) `String s=new String (byte [] b);`

### Important methods of String class:

- a) `public char charAt(int index)`
- b) `public boolean equals (object o)` → to check equality of String objects


example:

```
String s= "JAVA";
System.out.println(s.equals("java")); → false
```

- c) `public String concat(String s)`

Example:

```
String s= "Java";
s=s.concat(" Programming");
System.out.println(s); → Java Programming
```



```
s=s + " Programming";
s+= " Programming";
```

- d) `public boolean equalsIgnoreCase(String s)` → to check equality of string objects where case is ignored

example:

```
String s="JAVA";
System.out.println(s.equalsIgnoreCase("java")); → true
```

- e) `public boolean isEmpty()`

example:

```
String s= " ";
System.out.print(s.isEmpty()); → true
```

```
String s= "Java";
System.out.print(s.isEmpty()); → false
```

- f) `public int length()` → to check number of character in the String

example:

```
String s= "Java";
System.out.println(s.length()); → 4
```

**Note:** length variable applicable only for array concept but length () (length method) applicable only for String concept

- g) `public String replace(char old, char new);` → to replace every occurrence of old one with new one

example:

```
String s= "ababab";
System.out.println(s.replace('a', 'b')); → bbbbbb
```

- h) `public String substring(int begin)` → from begin index to end of the string

example:

```
String s= "abcdefg";
System.out.println(s.substring(3)); → defg
```

- i) `public String substring(int begin, int end)` → from begin index to end-1 index

example:

```
String s= "abcdefg";  
System.out.println(s.substring(3,5)); → def
```

j) `public int indexOf(char ch)`  
example:

```
String s= "java";  
System.out.println(s.indexOf('g')); → 3
```

Note:

- if specified character is not available then we are going to get -1 as output
- if specified character is available at multiple times in a string then it provides first occurrence index

k) `public int lastIndexOf(char ch)`  
example:

```
String s= "babbab";  
System.out.println(s.lastIndexOf('a')); → 4
```

l) `public String toLowerCase();`

m) `public String toUpperCase();`

n) `public String trim();` → to remove blank space present at begin and end

note: it not going to trim in between words

### Other String class Operations (String functions):

```
package com.sgtesting.stringdemo;
```

```
public class StringOperations  
{
```

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

```
        stringLength();  
        emptyString();  
        getCharacterBasedOnIndex();  
        lowercase();  
        uppercase();  
        compareString1();  
        compareString2();  
        existnaceOfString();  
        convertToCharArray();  
        splitString();  
        subStringDemo();  
        positionStr();  
        convertToStringDataType();  
    }
```

```
    private static void stringLength ()
```

```
    {  
        String s=new String("Welcome");  
        int v1=s.length();  
        System.out.println("# of chars :"+v1);  
        System.out.println("*****");  
    }
```

```

private static void emptyString ()
{
    String s=new String ();
    boolean v1=s.isEmpty();
    System.out.println("Is the given empty?:"+v1);
    System.out.println("*****");
}

private static void getCharacterBasedOnIndex ()
{
    String s="WELCOME";
    char ch=s.charAt(5);
    System.out.println("character :"+ch);
    System.out.println("*****");
}

private static void lowercase ()
{
    String s=new String("WELCOME");
    String res=s.toLowerCase();
    System.out.println("Lower case value:"+res);
    System.out.println("*****");
}

private static void uppercase ()
{
    String s=new String("programming");
    String v1=s.toUpperCase();
    System.out.println("Upper case value :"+v1);
    System.out.println("*****");
}

private static void compareString1()
{
    String s1=new String("welcome");
    String s2=new String("WELCOME");
    boolean v1=s1.equals(s2);
    System.out.println("s1 and s2 are equal ?:"+v1);
    boolean v2=s1.equalsIgnoreCase(s2);
    System.out.println("s1 and s2 are equal by ignoring case ?:"+v2);
    System.out.println("*****");
}

private static void compareString2()
{
    String s1=new String("welcome");
    String s2=new String("WELCOME");
    int v1=s1.compareTo(s2);
    System.out.println("s1 and s2 are equal ?:"+v1);
    int v2=s1.compareToIgnoreCase(s2);
    System.out.println("s1 and s2 are equal by ignoring case ?:"+v2);
    System.out.println("*****");
}

private static void existnaceOfString ()
{
    String s1=new String("It is a big tree at the top of the hill");
    boolean v1=s1.startsWith("It");

```

```

        System.out.println("Starts With 'It' :"+v1);
        boolean v2=s1.endsWith("hill");
        System.out.println("Ends With 'hill' :"+v2);
        boolean v3=s1.contains("tree");
        System.out.println("contains 'tree' :"+v3);
        System.out.println("*****");
    }

    private static void convertToCharArray ()
    {
        String s=new String("Welcome");
        char ch[]=s.toCharArray();
        for(char ch1 :ch)
        {
            System.out.println(ch1);
        }
        System.out.println("*****");
    }

    private static void splitString()
    {
        String s="Apple, Mango, Orange, Banana";
        String str[]=s.split(",");
        for(String kk:str)
        {
            System.out.println(kk);
        }
        System.out.println("*****");
    }

    private static void subStringDemo ()
    {
        String s="Programming";
        String s1=s.substring(3);
        System.out.println(s1);
        String s2=s.substring(3, 7);
        System.out.println(s2);
        System.out.println("*****");
    }

    private static void positionStr ()
    {
        String s="xxaxxxaXXAXXA";
        int pos1=s.indexOf('A', 0);
        System.out.println(pos1);
        int pos2=s.lastIndexOf('A');
        System.out.println(pos2);
        System.out.println("*****");
    }

    private static void convertToStringDataType ()
    {
        int a=125;
        String s=String.valueOf(a);
        System.out.println(s);
        double d=10.75;
        String s1=String.valueOf(d);
        System.out.println(s1);
    }

```

```

        System.out.println("*****");
    }
}

```

### Output:

```

# of chars :7
*****

Is the given empty?:true
*****

character :M
*****

Lower case value:welcome
*****

Upper case value :PROGRAMMING
*****

s1 and s2 are equal ?:false
s1 and s2 are equal by ignoring case ?:true
*****

s1 and s2 are equal ?:32
s1 and s2 are equal by ignoring case ?:0
*****

Starts With 'It' :true
Ends With 'hill' :true
contains 'tree' :true
*****

W
e
l
c
o
m
e
*****

Apple
Mango
Orange
Banana
*****

gramming
gram
*****

11
14
*****

125
10.75
*****

```

### String representation of an object:

-----

If any class overrides “toString” method from object superclass it provides a functionality of String representation of an object

### Example:

```

package com.sgtesting.stringdemo;

class Student

```

```

{
    String firstName;
    String CourseName;

    Student (String fn, String courseName)
    {
        this.firstName=fn;
        this.CourseName=courseName;
    }

    @Override
    public String toString()
    {
        return "Student Name "+firstName+" and his course is "+courseName;
    }
}

public class StringDemo2
{
    public static void main (String [] args)
    {
        Student obj = new Student ("Ranav", "Mechanical");
        System.out.println(obj);
    }
}

```

**Output:**

Student Name Ranav and his course is Mechanical

**StringBuffer:**

-----

The object is StringBuffer is mutable it means it accepts modifications

**When to go for StringBuffer Concept?**

**Ans:** if the content is not fixed and keep on changing but thread safety is required and at a time one thread is allowed to operate then we should go for StringBuffer concept

**Example:** Commenting section (we can edit after some days)

**Example:**

```

package com.sgtesting.stringdemo;

public class StringBufferDemo
{
    public static void main (String [] args)
    {
        StringBuffer s = new StringBuffer("Java");
        System.out.println(s);
        s.append(" Programming");
        System.out.println(s);
        s.append(" Language");
        System.out.println(s);
    }
}

```

**Output:**

Java

**Note:** append means → concatenation/combining;

### How to create an object for StringBuffer?

**Ans:**

- 1) `StringBuffer s = new StringBuffer ();`  
-----
- 2) `StringBuffer s = new StringBuffer ("Welcome");`  
-----
- 3) `StringBuffer s = new StringBuffer ("Welcome");`  
`StringBuffer s1 = new StringBuffer (s);`  
-----
- 4) `String s = new String ("Welcome");`  
`StringBuffer s1 = new StringBuffer (s);`  
-----
- 5) `StringBuilder s = new StringBuilder ("Welcome");`  
`StringBuffer s1 = new StringBuffer (s);`

**Note:** at any point StringBuffer never support literal approach of object creation

### StringBuffer Operations:

-----

```
package com.sgtesting.stringdemo;

public class StringBufferOperations
{
    public static void main (String [] args)
    {
        appendDemo();
        insertDemo();
        deleteDemo();
        reverseDemo();
    }

    private static void appendDemo()
    {
        StringBuffer s=new StringBuffer("Java");
        s.append(" Programming");
        System.out.println(s);
        System.out.println("*****");
    }

    private static void insertDemo()
    {
        StringBuffer s=new StringBuffer("It is a Book");
        StringBuffer s1=s.insert(8, "New ");
        System.out.println("Inserted Result: "+s1);
        System.out.println("*****");
    }

    private static void deleteDemo()
    {
        StringBuffer s=new StringBuffer("It is a New Book");
        StringBuffer s1=s.delete(8, 12);
    }
}
```



```

        System.out.println("Deleted Result: "+s1);
        System.out.println("*****");
    }

    private static void reverseDemo()
    {
        StringBuffer s=new StringBuffer("Welcome");
        StringBuffer s1=s.reverse();
        System.out.println("Reversed Result: "+s1);
        System.out.println("*****");
    }
}

```

#### Output:

```

Java Programming
*****
Inserted Result: It is a New Book
*****
Deleted Result: It is a Book
*****
Reversed Result: emocleW
*****

```

**Note:** StringBuffer operations has some String operations

#### Some Other important methods of StringBuffer:

- 1) public int length (); → how many characters present in StringBuffer
- 2) public int capacity (); → how many characters can StringBuffer accommodate
- 3) public char charAt (int index);
- 4) public void setCharAt (int index, Char newchar); → replace/set new char @ specified index
- 5) public StringBuffer deleteCharAt (int index);
- 6) public void setLength (int length); → it considers only specified length extra character will removes automatically
- 7) public void ensureCapacity (int capacity); → to increase capacity on fly based on our requirement
- 8) public void trimToSize (); → it will deallocate extra pre memory allocated to the object

#### StringBuilder:

The StringBuilder is mutable it accepts modifications

#### When to go for StringBuilder concept?

**Ans:** if the content is not fixed keep on changing but I don't want thread safety and multiple threads are allowed to operate then we should go for StringBuilder concept

#### Example:

```

package com.sgtesting.stringdemo;

public class StringBufferDemo
{
    public static void main (String [] args)
    {
        StringBuffer s = new StringBuffer("Java");
        System.out.println(s);
        s.append(" Programming");
    }
}

```

```

        System.out.println(s);
        s.append(" Language");
        System.out.println(s);
    }
}

```

**Output:**

```

Java
Java Programming
Java Programming Language

```

**Note:**

- default capacity of the empty StringBuffer object is 16 characters

**example:**

```

StringBuffer sb = new StringBuffer("Java");
System.out.println(sb.capacity());

```

Output: 16

- If 16 characters are completed if we trying to add new character or reaches its max capacity then another bigger StringBuffer object will create internally and its capacity will be (new capacity = (old capacity+1) \*2) it repeats every new object once old capacity is fully filled, and old capacity is eligible for garbage collection
- If we want to create bigger StringBuffer object at the starting time we have to mention its capacity like  
StringBuffer sb=new StringBuffer (int intialcapacity);
- If we create an equivalent StringBuffer for the given String (StringBuffer sb=new StringBuffer (String s);) then it's capacity will be (capacity = s.length+ empty StringBuffer capacity)

**How to create an Object for StringBuilder?**

**Ans:**

- 1) `StringBuilder s = new StringBuilder ();`  
-----
- 2) `StringBuilder s = new StringBuilder ("Welcome");`  
-----
- 3) `StringBuffer s = new StringBuffer ("Welcome");`  
`StringBuilder s1 = new StringBuilder (s);`  
-----
- 4) `String s = new String ("Welcome");`  
`StringBuilder s1 = new StringBuilder (s);`  
-----
- 5) `StringBuilder s = new StringBuilder ("Welcome");`  
`StringBuilder s1 = new StringBuilder (s);`

**Note:** at any point StringBuilder never support literal approach of object creation

**StringBuilder Operations:**

-----

```

package com.sgtesting.stringdemo;

public class StringBuilderDemo
{
    public static void main (String [] args)
    {
        appendDemo();
        insertDemo();
        deleteDemo();
        reverseDemo();
    }
}

```

```

private static void appendDemo()
{
    StringBuilder s=new StringBuilder("Java");
    s.append(" Programming");
    System.out.println(s);
    System.out.println("*****");
}

// multiple append methods are available like s.append(int i), s.append(long l) and so on

private static void insertDemo()
{
    StringBuilder s=new StringBuilder("It is a Book");
    StringBuilder s1=s.insert(8, "New ");
    System.out.println("Inserted Result: "+s1);
    System.out.println("*****");
}

private static void deleteDemo()
{
    StringBuilder s=new StringBuilder("It is a New Book");
    StringBuilder s1=s.delete(8, 12);
    System.out.println("Deleted Result: "+s1);
    System.out.println("*****");
}

private static void reverseDemo()
{
    StringBuilder s=new StringBuilder("Welcome");
    StringBuilder s1=s.reverse();
    System.out.println("Reversed Result: "+s1);
    System.out.println("*****");
}
}

```

#### Output:

```

Java Programming
*****
Inserted Result: It is a New Book
*****
Deleted Result: It is a Book
*****
Reversed Result: emocleW
*****

```

#### Assignment:

- 1) Without using reverse method reverse given String by applying charAt method
- 2) Without using reverse method reverse the given String by applying toCharArray method
- 3) Without using reverse method reverse the given string substring method
- 4) Without using length method write a program to find no of characters in a given String
- 5) WAP to verify the given String is palindrome or not
- 6) The given String is “Bangalore is capital city of Karnataka” WAP to find the number of words in the given String
- 7) The given String is “Bangalore and Mysore” WAP to display as “Mysore and Bangalore”

**8) The given String is "Program"**

- |         |                  |
|---------|------------------|
| a) P    | b) P R O G R A M |
| PR      | PROGRA           |
| PRO     | PROGR            |
| PROG    | PROG             |
| PROGR   | PRO              |
| PROGRA  | PR               |
| PROGRAM | P                |

**How to add Eclipse files to Git\_Repository?**

**Ans:**

- Create a folder for Git\_Repository in any drive in that again create two folders one for Demo\_Workspace and another one is for current\_Workspace
- Now copy the stable projects from eclipse workspace and paste it in Git\_Repository of Demo\_Workspace
- The open the Eclipse and choose the workspace as Git\_Repository then go to file and import the projects using following steps
  - Go file
  - In file choose import
  - In that go to general and choose Existing Projects into workspace
  - Then browse the select root directory then select the projects and then finish

**How to create master branch?**

**Ans:**

- After that assign the folders to GitHub using git command prompt
- Go to the Git\_Repository in Demo\_Workspace right click and choose git bash here
- In git bash here command prompt config the Repository as follows
  - \$ git config --global user.name "<user name>"
  - \$ git config --global user.email "<user email>"
  - \$ git init
  - \$ git add \*
  - \$ git commit -m "Initial Draft"
  - \$ git remote add origin <https://github.com/Username/ExampleOctober3rd2023Repository.git>  
(Note: here paste URL of the repository of GitHub)
  - \$ git push origin master  
(It will show result like this  
Enumerating objects: 879, done.  
Counting objects: 100% (879/879), done.  
Delta compression using up to 8 threads  
Compressing objects: 100% (869/869), done.  
Writing objects: 100% (879/879), 359.40 KiB | 570.00 KiB/s, done.  
Total 879 (delta 262), reused 0 (delta 0), pack-reused 0  
remote: Resolving deltas: 100% (262/262), done.  
To https://github.com/RanavRakshi/ExampleOctober3rd2023Repository.git  
\* [new branch] master -> master)

**How to clone the master branch?**

**Ans:**

After creating master branch go to Git\_Repository in that choose Current\_worksapce in that right click and select git bash here and open

- Int that clone the master branch to this workspace by typing (\$ git clone (copy and paste: -GitHub→ Repository→ code→ select HTTPS URL) and enter
- Then go to eclipse and browse the Current\_Workspace and import then you see the java project as master

### How to create a new QA branch?

**Ans:**

- In Current\_Workspace right click and open git bash here and Display all available branches (to display all available branches in git bash here command prompt) type –  
\$ git branch
- Create local branch (to create local branch in git bash here command prompt) type –  
\$ git branch qabranh
- To point or switch into a new branch in git bash here command prompt type-  
\$ git checkout qabranh

### How to write a new program and how to push into the master branch?

**Ans:**

- In Eclipse choose Git\_Repository's Current\_Workspace as an Eclipse Workspace it contains cloned data of master branch as qabranh and we see the java projects now as qabranh
  - Then create a new package once after creating a new class project
- Step1: \$ git status  
"To make sure any new changes are available in your current branch"
- Step2: \$ git add <new changes>  
(Copy the new Changes from git status and paste it here)
- Step3: \$ git commit -m "Sample Program"  
"To commit the added new changes)
- Step4: \$ git push  
(Once after executing \$ git push command, it allows to create a pull request (PR))
- Step5:
- In GitHub browser Repository click on compare and pull request button
  - Then we have to add the reviewers
  - We have to provide appropriate pull request description and click on create pull request button
  - The pull request assign to reviewers once they approved then only merge pull request button will be enabled
  - Click on merge pull request button it provides one more button for confirm merge and click on this button it merges new changes into master branch

### File System (Java.IO) or File Handling:

-----

#### File:

The object of file supports creation of file or directories, rename of files or directories or deleting of file or directories

**Note:** at any point the object of file doesn't support read and write content into a file because the file object doesn't have stream concept

### How to create a object for file class?

**Ans:** In a basic approach we can create object in bellow approaches

- File f1 = new File ("D:\\Demo\\Test\\Welcome.txt");  
or
- File f1 = new File("D:/Demo/Test/Welcome.txt");

#### Note:

- **Back slash '\\':** it is a reserved character in core java it is mainly used to lose the special meaning of any reserved character  
So, in order to use it as a file separator we should use two back slash as '\\'
- **Forward slash '/':** it is a file separator in all operating system

**Note:** the file class constructor has overloaded so we can create the object class as follows

- File f1 = new File ("D:\\Demo\\Test\\Welcome.txt");  
\*\*\*\*\*
- File f1 = new File ("D:\\Demo\\Test\\Welcome.txt");

```

File f2 = new File (f1);
*****
- File f1 = new File ("D:\\Demo\\Test", "Welcome.txt");
*****
- File f1 = new File ("D:\\Demo\\Test");
File f2 = new File (f1, "Welcome.txt");

```

## File Details:

### ----- Example:

```

package com.sgtesting.iodeemo;

import java.io.File;

public class FileDetailsDemo
{
    public static void main (String [] args)
    {
        File f1=new File ("D:\\Demo\\Example\\Test.txt");

        // display name of the file

        String filename=f1.getName();
        System.out.println("File Name: "+filename);
        System.out.println("*****");

        // display the path of the file

        String path1=f1.getAbsolutePath();
        System.out.println("File Path: "+path1);
        System.out.println("*****");

        // display the parent folder name

        String Parentfolder=f1.getParent();
        System.out.println("Parent Folder: "+Parentfolder);
        System.out.println("*****");

        // display whether it is a file

        boolean v1=f1.isFile();
        System.out.println("It is a File: "+v1);
        System.out.println("*****");

        // display whether it is a folder

        boolean v2=f1.isDirectory();
        System.out.println("It is a Folder: "+v2);
        System.out.println("*****");

        // whether it is readable

        boolean v3=f1.canRead();
        System.out.println("It is readable: "+v3);
        System.out.println("*****");

        // whether it is Writable

```

```

        boolean v4=f1.canWrite();
        System.out.println("It is Writable: "+v4);
        System.out.println("*****");

        // whether it is executable

        boolean v5=f1.canExecute();
        System.out.println("It is Executable: "+v5);
        System.out.println("*****");
    }
}

```

#### Output:

```

File Name: Test.txt
*****
File Path: D:\Demo\Example\Test.txt
*****
Parent Folder: D:\Demo\Example
*****
It is a File: false
*****
It is a Folder: false
*****
It is readable: false
*****
It is Writable: false
*****
It is Executable: false
*****

```

#### File operations:

-----  
 The file operations include creating a file, rename the file, delete the file, displaying the file collection

#### Example:

```

package com.sgtesting.iodemo;

import java.io.File;

public class FileOperationsDemo
{
    public static void main (String [] args)
    {
        //createFile();
        //renameFile();
        //deleteFile();
        fileCollections();
    }

    private static void createFile()
    {
        try
        {
            File f1=new File ("D:\\Demo\\Example\\Sample\\Sample.txt");

```

```

        f1.createNewFile();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

private static void renameFile()
{
    File f1=new File ("D:\\Demo\\Example\\Sample.txt");
    File f2=new File ("D:\\Demo\\Example\\SampleNew.txt");

    boolean v1=f1.renameTo(f2);
    System.out.println("Whether File has renamed: "+v1);
}

private static void deleteFile()
{
    File f2=new File ("D:\\Demo\\Example\\SampleNew.txt");

    boolean v1=f2.delete();
    System.out.println("Whether file has Deleted: "+v1);
}

private static void fileCollections()
{
    File f1=new File ("D:\\Demo\\Example");
    File []f2=f1.listFiles();
    for(int i=0; i<f2.length;i++)
    {
        if(f2[i].isFile()==true)
        {
            System.out.println(f2[i].getPath());
        }
    }
}
}

```

### Folder (Directory) Operations:

---

By using folder operation, we can perform create, rename, delete of folders, folder collections

#### Example:

```

package com.sgtesting.iodemo;

import java.io.File;

public class FolderOperationsDemo
{
    public static void main(String[] args)
    {
        createFolder();
        nestedFolder();
        renameFolder();
        deleteFolder();
        folderCollections();
    }
}

```



```

    }

    private static void createFolder()
    {
        File f1=new File ("D:\\Demo\\Example\\Test");
        f1.mkdir();
    }

    private static void nestedFolder()
    {
        File f1=new File("D:\\Demo\\Example\\D1\\D2\\D3\\D4");
        f1.mkdirs();
    }

    private static void renameFolder()
    {
        File f1=new File("D:\\Demo\\Example\\Test");
        File f2=new File("D:\\Demo\\Example\\TestAssigned");

        boolean v1=f1.renameTo(f2);
        System.out.println("whether the folder has renamed > "+v1);
    }

    private static void deleteFolder()
    {
        File f2=new File("D:\\Demo\\Example\\TestAssigned");

        boolean v1=f2.delete();
        System.out.println("Whether folder has delete > "+v1);
    }

    private static void folderCollections()
    {
        File f1=new File("D:\\Demo\\Example");
        File [] f2=f1.listFiles();
        for(int i=0; i<f2.length;i++)
        {
            if(f2[i].isDirectory()==true);
            {
                System.out.println(f2[i].getAbsolutePath());
            }
        }
    }
}

```

#### Assignment:

- 1) WAP to create 10 sub folders inside the demo folder
- 2) There is a folder called weekday programmatically seven weekday name sub folders here
- 3) There is a folder month programmatically create 12-month names of sub folders
- 4) There is a folder it has five .txt files, five .xlsx files, five .png files
  - a) WAP to display only the .png files
  - b) WAP to display only the .xlsx files
  - c) WAP to display only the .txt files
  - d) WAP to rename only the .png files
  - e) WAP to delete only the .png files
- 5) WAP to display recursive directories

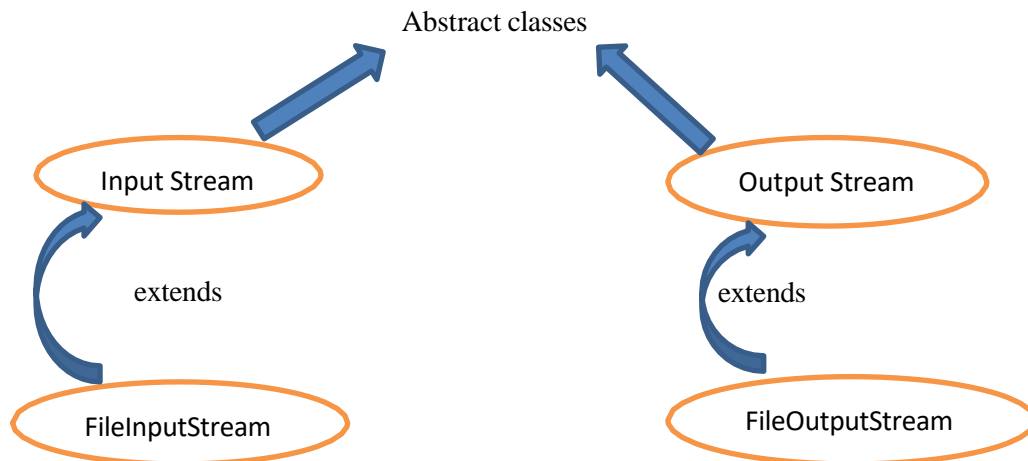
## Stream Classes:

Core java supports the following stream classes:

- Byte Stream
- Character Stream
- Buffered Stream

## Byte stream:

It is a fundamental stream concept to read or write content from the file



There are 2 abstract classes are available in java.io.package, these abstract classes are extended by FileInputStream and FileOutputStream respectively

## FileInputStream:

The object of FileInputStream is used for reading the content from Test file

- It reads the content either integer representation or byte array representation

## How to create an object for FileInputStream?

Ans:

```
FileInputStream f1=new FileInputStream("E:\\Test.txt");
```

```
File f1=new File("E:\\Test.txt");
```

```
FileInputStream f1=new FileInputStream(f1);
```

## Example:

```
package com.sgtesting.iodeemo;  
  
import java.io.FileInputStream;  
  
public class FileInputStreamDemo  
{  
    public static void main(String[] args)  
    {  
        readContent();  
    }  
}
```

```

private static void readContent()
{
    FileInputStream fin=null;
    int n=0;
    try
    {
        fin=new FileInputStream("D:\\Demo\\Example\\Test.txt");
        while(true)
        {
            n=fin.read();
            if(n==-1)
            {
                break;
            }
            char ch=(char) n;
            System.out.print(ch);
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        try
        {
            fin.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
}

```

### **FileOutputStream:**

-----

The object of FileOutputStream is used for writing the content into a Test file

### **How to create an object for FileOutputStream?**

**Ans:**

```

FileOutputStream f1=new FileOutputStream("E:\\Test.txt");
-----
File f1=new File("E:\\Test.txt");
FileOutputStream fout=new FileOutputStream(f1);
-----
FileOutputStream fout=new FileOutputStream ("E:\\Test.txt", true);
-----
File f1=new File("E:\\Test.txt");
FileOutputStream fout=new FileOutputStream (f1, true);

```

### **Example:**

#### **1) If we are not using true:**

```

package com.sgtesting.iodeemo;

```

```

import java.io.FileOutputStream;

public class FileOutputStreamDemo
{
    public static void main(String[] args)
    {
        writeContent();
    }
    private static void writeContent()
    {
        FileOutputStream fout=null;
        try
        {
            fout=new FileOutputStream("D:\\Demo\\Example\\Welcome.txt");
            String strContent="Bangalore is a capital City of Karnataka";
            strContent+=" ,It is also called as Garden City of India.";

            byte b[]=strContent.getBytes();
            fout.write(b);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        finally
        {
            try
            {
                fout.close();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
}

```

## 2) If we use true

```

package com.sgtesting.iodeemo;

import java.io.FileOutputStream;

public class FileOutputStreamDemo
{
    public static void main(String[] args)
    {
        writeContent();
    }
    private static void writeContent()
    {
        FileOutputStream fout=null;
        try
        {
            fout=new FileOutputStream("D:\\Demo\\Example\\Welcome.txt",true);

```

```

String strContent="Bangalore is a capital City of Karnataka";
strContent+=" ,It is also called as Garden City of India.";

byte b[]=strContent.getBytes();
fout.write(b);
}
catch (Exception e)
{
    e.printStackTrace();
}
finally
{
    try
    {
        fout.close();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
}
}

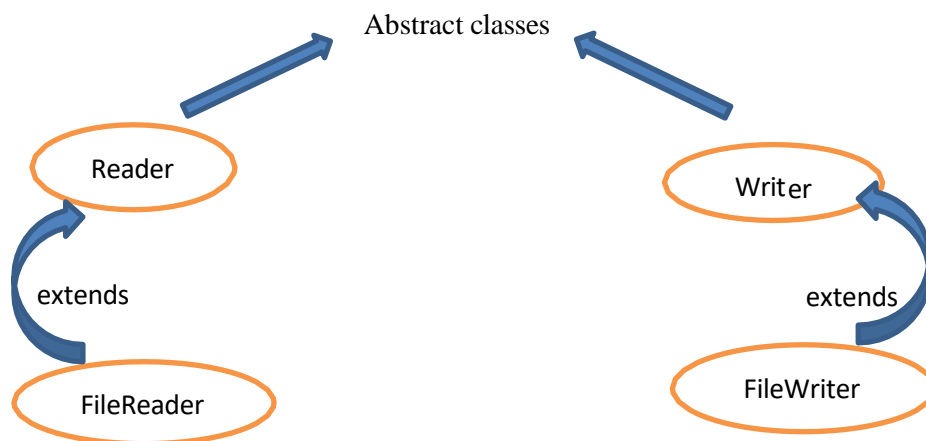
```

**Note:**

- If we use true, it writes repeatedly when we run the code respectively how much time we run that much time it will write

**Character Stream:**

-----



There are 2 abstract classes Reader and Writer these 2 abstract classes are extended by FileReader and FileWriter classes respectively

**FileReader:**

The object of FileReader is used to read the content from test file, it Reads the content either in integer or character array representation

**How to create a FileReader object?**

**Ans:**

```

FileReader fr=new FileReader ("D:\\Demo\\Test.txt");
=====
File f1=new File ("D:\\Demo\\Test.txt");
FileReader fr=new FileReader (f1);

```

**Example:**

```

package com.sgtesting.iodeemo;

import java.io.FileReader;

public class FileReaderDemo
{
    public static void main(String args [])
    {
        readContent();
    }

    private static void readConetent()
    {
        FileReader fr=null;
        Int n=0;

        try
        {
            Fr=new FileReader("D:\\Demo\\Example\\Test.txt");
            while(true)
            {
                n=fr.read();
                if(n == -1)
                {
                    break;
                }
                Char ch=(Char) n;
                System.out.println(ch);
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        finally
        {
            try
            {
                fr.close();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
}

```

**FileWriter:**

The object of FileWriter used for write content into a test file, it accepts content in the form of integer or character or String representation for writing content into a test file

**How to create an object for FileWriter?**

**Ans:**

```
FileWriter fw=new FileWriter ("D:\\Demo\\Test.txt");
```

```

-----
File f1=new File("D:\\Demo\\Test.txt");
FileWriter fw=new FileWriter (f1);
-----
FileWriter fw=new FileWriter ("D:\\Demo\\Test.txt", true);
-----
File f1=new File("D:\\Demo\\Test.txt");
FileWriter fw=new FileWriter (f1, true);

```

### Example:

```

package com.sgtesting.iodeemo;

import java.io.FileWriter;

public class FileWriterDemo
{
    public static void main(String[] args)
    {
        writeContent();
    }
    private static void writeContent()
    {
        FileWriter fw=null;
        try
        {
            fw=new FileWriter("D:\\Demo\\Example\\Welcome11.txt",true);
            String str="Java is a programming language.";
            str+="Java is mainly used for developing Applications.";

            char ch[]=str.toCharArray();
            fw.write(ch);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        finally
        {
            try
            {
                fw.close();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
}

```

### Buffered Stream (Buffered classes):

There are two classes are available here BufferedReader and BufferedWriter

#### BufferedReader:

The object of BufferedReader is used to read the content line by line

## How to create object for BufferedReader?

**Ans:**

```
FileReader fr=new FileReader ("D:\\Demo\\Test.txt");
BufferedReader br=new BufferedReader(fr);
-----
BufferedReader br=new BufferedReader(new FileReader ("D:\\Demo\\Test.txt"));
```

**Example:**

```
package com.sgtesting.iodeemo;

import java.io.BufferedReader;
import java.io.FileReader;

public class BufferedReaderDemo
{
    public static void main(String[] args)
    {
        readContent();
    }

    private static void readContent()
    {
        FileReader fr=null;
        BufferedReader br=null;
        try
        {
            fr=new FileReader("D:\\Demo\\Example\\Test.txt");
            br=new BufferedReader(fr);

            String sLine=null;
            while((sLine=br.readLine())!=null)
            {
                System.out.println(sLine);
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        finally
        {
            try
            {
                fr.close();
                br.close();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

## BufferedWriter:

The object of BufferedWriter is used to write the content line by line



## How to create object for BufferedWriter?

Ans:

```
FileWriter fw=new FileWriter ("D:\\Demo\\Test.txt");
BufferedWriter bw=new BufferedWriter (fw);
-----
BufferedWriter bw=new BufferedWriter (new FileWriter ("D:\\Demo\\Test.txt");
-----
FileWriter fw=new FileWriter ("D:\\Demo\\Test.txt", true);
BufferedWriter bw=new BufferedWriter (fw);
-----
BufferedWriter bw=new BufferedWriter (new FileWriter ("D:\\Demo\\Test.txt", true));
```

## Example:

```
package com.sgtesting.iodeemo;

import java.io.BufferedWriter;
import java.io.FileWriter;

public class BufferedWriterDemo
{
    public static void main(String[] args)
    {
        writeContent();
    }
    private static void writeContent()
    {
        BufferedWriter bw=null;
        try
        {
            bw=new BufferedWriter(new
FileWriter("D:\\Demo\\Example\\Welcome22.txt",true));
            bw.write("Bangalore is a capital city of Karnataka.");
            bw.newLine();
            bw.write("It is also called as Garden City of Karnataka.");
            bw.newLine();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        finally
        {
            try
            {
                bw.flush();
                bw.close();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

## POI API:

-----

It stands for “Poor obfuscation Implementation”

### How to create a object for Poi/ How to get the EXCEL sheet details in Eclipse?

Ans:

Step1:

FileInputStream fin=new FileInputStream (“EXCEL sheet path”); → ex: “D:\\EXCEL\\Test.xlsx”

Or

FileOutputStream fout=new FileOutputStream(“EXCEL sheet path”); → ex: “D:\\EXCEL\\Test.xlsx”

Step2:

Workbook wb=null;

wb=new XSSFWorkbook(fin);

Step3:

Sheet sh=null;

sh=wb.getSheet(“Sheet1”); // if sheet exists

Or

sh=wb.createSheet(“Sheet1”); // if sheet doesn’t exists

Step4:

Row row=null;

row=sh.getRow(rownumber); // if row exists

Or

row=sh.createRow(rownumber); // if row doesn’t exists

Step5:

Cell cell=null;

cell=row.getCell(cellnumber); // if cell exists

or

cell=row.createCell(cellnumber); // if cell doesn’t exists

Step6:

Read/Write

### How to download the POI JAR files?

Ans:

- In google search <https://poi.apache.org/>
- Click on first link “Download Release Artifacts - Apache POI”
- In the home page left hand side click on download
- Int that go to Binary artifacts and click on archies of all prior releases
- In that click on Binary artifacts and select version – 5.2.3 (poi -bin -5.2.3 -20230909.zip)
- And extract here and using winrar

### How to create poi folder in Eclipse?

Ans:

- Go to Eclipse create a new java project called ExcelAutomation
- Then in that File→ new → Folder and give a name as “Library”
- Then right click on Library and create another folder called “poi”

### How to add available JAR files into build path?

Ans:

- In Eclipse select ExcelAutomation project expand Library and poi in poi right click → Buildpath→ configure Buildpath
- Click on Libraries tab

- Select classpath in that click on add JARs
- The respective project got highlighted in popup window
- Expand the particular project and expand Library folder and expand poi
- Select the poi JAR files click on ok button and apply and apply&close  
(to select all JAR files click on first file and shift button and scroll down and click on last file)

### How to read the existing EXCEL file using Eclipse?

Ans;

```
package com.sgtesting.excel;

import java.io.FileInputStream;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class ReadExcelContentDemo
{
    public static void main (String [] args)
    {
        readContent();
    }
    public static void readContent()
    {
        FileInputStream fin=null;
        Workbook wb=null;
        Sheet sh=null;
        Row row=null;
        Cell cell=null;

        try
        {
            fin=new FileInputStream("G:\\EXCEL\\Test.xlsx");
            wb=new XSSFWorkbook(fin);
            sh=wb.getSheet("Sheet1");
            int rc=sh.getPhysicalNumberOfRows();
            for(int r=0;r<rc;r++)
            {
                row=sh.getRow(r);
                int cc=row.getPhysicalNumberOfCells();
                for(int c=0;c<cc;c++)
                {
                    cell=row.getCell(c);
                    String data=cell.getStringCellValue();
                    System.out.print(data+" ");
                }
                System.out.println();
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        finally
        {
        }
    }
}
```

```

        {
            try
            {
                fin.close();
                wb.close();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
}

```

#### Output:

ERROR StatusLogger Log4j2 could not find a logging implementation. Please add log4j-core to the classpath. Using SimpleLogger to log to the console...

Ranav is going to round the world in a car  
Ranav is going to round the world in a car  
Ranav is going to round the world in a car  
Ranav is going to round the world in a car  
Ranav is going to round the world in a car  
Ranav is going to round the world in a car  
Ranav is going to round the world in a car  
Ranav is going to round the world in a car

**Note:** here neglect the ERROR line

#### Case2: How to write the content into a new Excel file?

Ans:

```

package com.sgtesting.excel;

import java.io.FileOutputStream;

import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class WriteExcelContentDemo
{
    public static void main(String[] args)
    {
        writeContent();
    }
    private static void writeContent()
    {
        Workbook wb=null;
        Sheet sh=null;
        Row row=null;
        Cell cell=null;
        FileOutputStream fout=null;

        try
        {
            wb=new XSSFWorkbook();

```

```

        sh=wb.createSheet("Credentials");
        // 1st row object
        row=sh.createRow(0);
        cell=row.createCell(0);
        cell.setCellValue("UserName");
        cell=row.createCell(1);
        cell.setCellValue("Password");

        // 2nd row object
        row=sh.createRow(1);
        cell=row.createCell(0);
        cell.setCellValue("Ranav");
        cell=row.createCell(1);
        cell.setCellValue("password");

        fout=new FileOutputStream("G:\\EXCEL\\Creadentials.xlsx");
        wb.write(fout);
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        try
        {
            fout.close();
            wb.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
}

```

**Case3: In a same Excel file read the content from 1<sup>st</sup> sheet and write the same content into a second sheet**

**Ans:**

```

package com.sgtesting.excel;

import java.io.FileInputStream;
import java.io.FileOutputStream;

import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class ReadAndWriteExcelContentDemo
{
    public static void main (String [] args)
    {
        readAndWriteContent();
    }
}

```

```

private static void readAndWriteContent()
{
    FileInputStream fin=null;
    Workbook wb=null;
    Sheet sh1=null;
    Sheet sh2=null;
    Row rowsh1=null;
    Row rowsh2=null;
    Cell cellsh1=null;
    Cell cellsh2=null;
    FileOutputStream fout=null;

    try
    {
        fin=new FileInputStream("G:\\EXCEL\\Test.xlsx");
        wb=new XSSFWorkbook(fin);
        sh1=wb.getSheet("Sheet1");

        sh2=wb.getSheet("Sheet2");
        if(sh2==null)
        {
            sh2=wb.createSheet("Sheet2");
        }
        int rc=sh1.getPhysicalNumberOfRows();
        for (int r=0; r<rc; r++)
        {
            rowsh1=sh1.getRow(r);
            rowsh2=sh2.getRow(r);
            if(rowsh2==null)
            {
                rowsh2=sh2.createRow(r);
            }
            int cc=rowsh1.getPhysicalNumberOfCells();
            for (int c=0; c<cc; c++)
            {
                cellsh1=rowsh1.getCell(c);
                cellsh2=rowsh2.getCell(c);
                if(cellsh2==null)
                {
                    cellsh2=rowsh2.createCell(c);
                }
                String data=cellsh1.getStringCellValue();
                cellsh2.setCellValue(data);
            }
        }
        fout=new FileOutputStream("G:\\EXCEL\\Test.xlsx");
        wb.write(fout);
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        try
        {
            fin.close();
            fout.close();
        }
    }
}

```

```

        wb.close();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
}
}

```

#### Assignments:

Perform each action programmatically

- 1) Programmatically write 20 flower names in first sheet 1<sup>st</sup> column of an Excel file.
- 2) Programmatically write 20 fruit names in sheet1 10<sup>th</sup> row.
- 3) Programmatically write 12-month names in a 1<sup>st</sup> sheet diagonally.
- 4) Programmatically write 20 country names in 1<sup>st</sup> sheet 5<sup>th</sup> column of an Excel file.
- 5) Programmatically write 20 flower names and 20 color names in 1<sup>st</sup> sheet , 1<sup>st</sup> and 2<sup>nd</sup> column of an Excel file.

Perform below actions in an existing Excel file

- 1) There is an Excel file in sheet1, 1<sup>st</sup> column it has 20 state names, read the content and write it into a 2<sup>nd</sup> sheet 10<sup>th</sup> column.
- 2) There is an Excel file it has 1<sup>st</sup> sheet 1<sup>st</sup> column it has 20 fruit names, read this content and write it into a sheet2 of 5<sup>th</sup> row.
- 3) There is an Excel file it has 20 rows of student details that includes FName, Lname, CourseName, BranchName, Cityname, address, statename read this content and write it into a new Excel file

### Part3:

#### Wrapper Classes:

-----

For each primitive datatype the corresponding wrapper classes available in core java

- **AutoBoxing:** it is a process of converting primitive datatype into wrapper classes object

#### Example:

```
package com.sgtesting.utildemo;

public class WrapperClassAutoBoxingDemo
{
    public static void main(String[] args)
    {
        autoBoxing();
    }

    private static void autoBoxing()
    {
        int a=45;
        System.out.println("int a: "+a);

        // convert it into wrapper class object
        Integer b=Integer.valueOf(a);
        System.out.println("Integer b: "+b);

        // AutoBoxing
        Integer c=a;
        System.out.println("Integer c: "+c);
    }
}
```

#### Output:

```
int a: 45
Integer b: 45
Integer c: 45
```

#### UnBoxing:

-----

It is a process of converting wrapper class object into primitive datatype value

#### Example:

```
package com.sgtesting.utildemo;

public class WrapperClassUnBoxingDemo
{
    public static void main(String[] args)
    {
        unBoxingDemo();
    }

    private static void unBoxingDemo()
    {
        Integer a=Integer.valueOf(45);
        System.out.println("Integer a: "+a);
    }
}
```



```

        // convert to primitive value
        int b=a.intValue();
        System.out.println("int b: "+b);

        // UnBoxing
        int c=a;
        System.out.println("int c: "+c);
    }
}

```

#### Output:

```

Integer a: 45
int b: 45
int c: 45

```

#### Generics:

-----

- By using Generics we can make type safety for the object
- By using Generics we can eliminate the casting and we can eliminate unnecessary raised compilation errors

#### Case1: with an example describe the Generics Concept (@ class level)

```

package com.sgtesting.utildemo;

class MyClass<T>
{
    T obj;
    public void add(T obj)
    {
        this.obj=obj;
    }
    public T get()
    {
        return obj;
    }
}

public class GenericsDemo
{
    public static void main(String[] args)
    {
        // with Generic
        MyClass<Integer> o1=new MyClass<Integer>();
        o1.add(100);
        int a=o1.get();
        System.out.println(a);

        o1.add(200);
        int b=o1.get();
        System.out.println(b);
        System.out.println("*****");

        // without Generic
        MyClass o2=new MyClass();
        o2.add("Mango");
        String s1=(String)o2.get();
        System.out.println(s1);
    }
}

```

```

        o2.add(true);
        boolean b1=(boolean)o2.get();
        System.out.println(b1);
    }
}

```

**Output:**

```

100
200
*****
Mango
true

```

**Note:** classname<T> → here <T> refers Generic type @ class level

**Case2: Generic for element level**

```

package com.sgtesting.utildemo;

class Sample
{
    public static <E> void readElements(E[] elements)
    {
        for(E element: elements)
        {
            System.out.println(element);
        }
    }
}

public class GenericElementLevelDemo
{
    public static void main(String[] args)
    {
        Integer a[]= {10,20,30,40,50};
        Sample.readElements(a);
        System.out.println("*****");

        String s[]= {"Mango","Orange","Banana","Apple"};
        Sample.readElements(s);
        System.out.println("*****");

        Double d[]= {1.1,2.2,3.3,4.4,5.5};
        Sample.readElements(d);
        System.out.println("*****");
    }
}

```

**Output:**

```

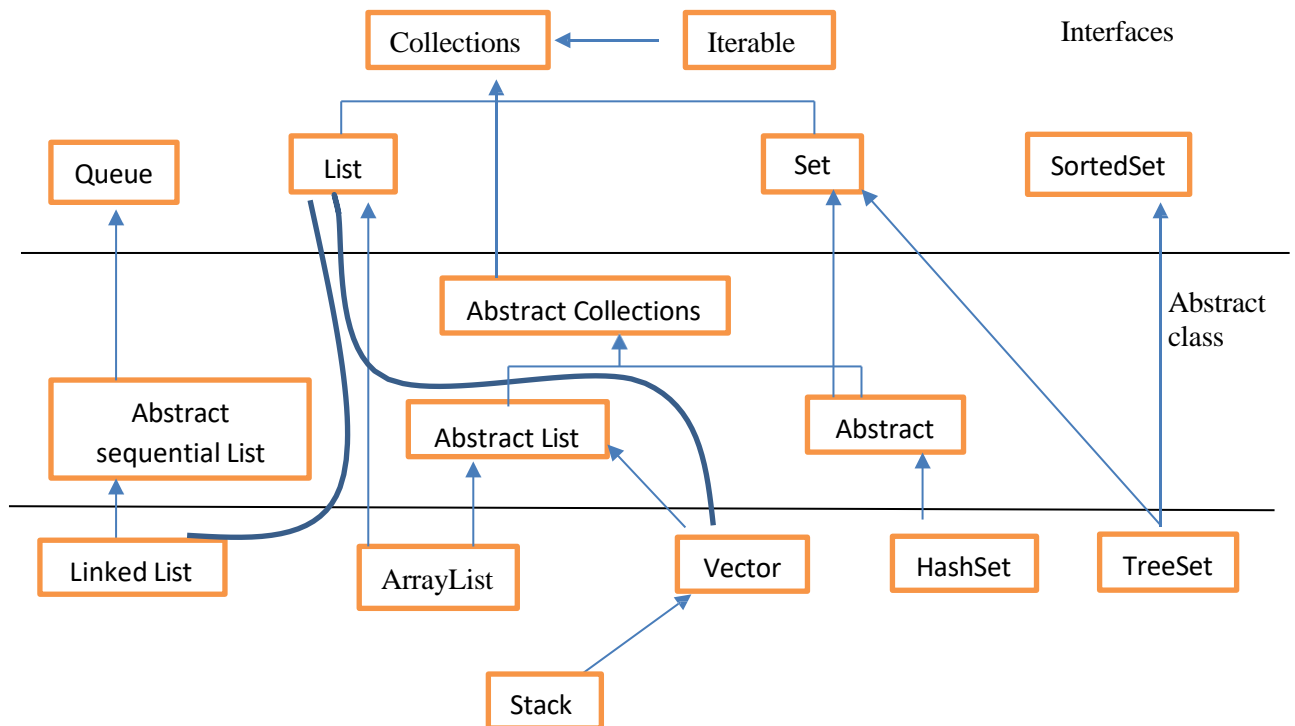
10
20
30
40
50
*****
Mango
Orange
Banana

```

Apple  
 \*\*\*\*\*  
 1.1  
 2.2  
 3.3  
 4.4  
 5.5  
 \*\*\*\*\*

**Note:** <E> → it is a element type it can accept/read irrespective of datatypes

### Collection Framework:



### List Interface:

The “List” supports duplicate elements and it stores elements based on the index, here the index always starts with zero

**The ‘List’ interface has implemented by the following classes:**

- ArrayList
- LinkedList
- Vector

### ArrayList:

The object of ArrayList supports duplicate elements and it stores elements in the memory based on the order of adding

It supports index to store the elements and index always starts with zero here

### How to create an ArrayList object?

**Ans:**

```
List<Generics> obj=new ArrayList<Generics>();
ArrayList<Generics> obj=new ArrayList<Generics>();
```

```
List obj=new ArrayList();  
ArrayList obj=new ArrayList();
```

**Example:**

```
package com.sgtesting.utildemo;  
  
import java.util.ArrayList;  
import java.util.Iterator;  
  
public class ArrayListDemo  
{  
    public static void main(String[] args)  
    {  
        addElement();  
        removeElements();  
        readElements();  
        readElements2();  
        readElementsByIterator();  
        withoutGenerics();  
    }  
  
    private static void addElement()  
    {  
        ArrayList<String> obj=new ArrayList<String>();  
        System.out.println("Elements: "+obj);  
  
        obj.add("Mango");  
        obj.add("Orange");  
        obj.add("Apple");  
        obj.add(0, "Berry");  
        obj.add("Mango");  
        obj.add("Guava");  
        System.out.println("Elements: "+obj);  
  
        ArrayList<String> obj1=new ArrayList<String>();  
        obj1.add("White");  
        obj1.add("Blue");  
        obj1.add("Green");  
        obj.addAll(obj1);  
        System.out.println("Elements: "+obj);  
    }  
  
    private static void removeElements()  
    {  
        ArrayList<String> obj=new ArrayList<String>();  
        System.out.println("Elements: "+obj);  
  
        obj.add("Mango");  
        obj.add("Orange");  
        obj.add("Apple");  
        obj.add(0, "Berry");  
        obj.add("Mango");  
        obj.add("Guava");  
        System.out.println("Elements: "+obj);  
  
        // remove based on elements  
        obj.remove("Apple");  
    }  
}
```

```

        System.out.println("Elements: "+obj);

        // remove based on index
        obj.remove(2);
        System.out.println("Elements: "+obj);

        // remove based on All
        obj.removeAll(obj);
        System.out.println("Elements: "+obj);
    }

    private static void readElements()
    {

        ArrayList<String> obj=new ArrayList<String>();
        System.out.println("Elements: "+obj);

        obj.add("Mango");
        obj.add("Orange");
        obj.add("Apple");
        obj.add(0, "Berry");
        obj.add("Mango");
        obj.add("Guava");
        System.out.println("Elements: "+obj);

        // read based on ArrayList using for-each loop
        for(String str:obj)
        {
            System.out.println("Elements: "+str);
        }
    }

    private static void readElements2()
    {

        ArrayList<String> obj=new ArrayList<String>();
        System.out.println("Elements: "+obj);

        obj.add("Mango");
        obj.add("Orange");
        obj.add("Apple");
        obj.add(0, "Berry");
        obj.add("Mango");
        obj.add("Guava");
        System.out.println("Elements: "+obj);

        // read based on ArrayList using for loop
        for(int i=0;i<obj.size();i++)
        {
            System.out.println(obj.get(i));
        }
    }

    private static void readElementsByIterator()
    {

        ArrayList<String> obj=new ArrayList<String>();
        System.out.println("Elements: "+obj);

        obj.add("Mango");

```

```

        obj.add("Orange");
        obj.add("Apple");
        obj.add(0, "Berry");
        obj.add("Mango");
        obj.add("Guava");
        System.out.println("Elements: "+obj);

        // read based on Iterator
        Iterator<String> ite=obj.iterator();
        while(ite.hasNext())
        {
            System.out.println(ite.next());
        }
    }

    private static void withoutGenerics()
    {
        ArrayList obj=new ArrayList();
        System.out.println("Elements: "+obj);

        obj.add(120);
        obj.add('Y');
        obj.add(180.122);
        obj.add(true);
        System.out.println("Elements: "+obj);
    }
}

```

### Output:

```

Elements: []
Elements: [Berry, Mango, Orange, Apple, Mango, Guava]
Elements: [Berry, Mango, Orange, Apple, Mango, Guava, White, Blue, Green]
Elements: []
Elements: [Berry, Mango, Orange, Apple, Mango, Guava]
Elements: [Berry, Mango, Orange, Mango, Guava]
Elements: [Berry, Mango, Mango, Guava]
Elements: []

Elements: []
Elements: [Berry, Mango, Orange, Apple, Mango, Guava]
Elements: Berry
Elements: Mango
Elements: Orange
Elements: Apple
Elements: Mango
Elements: Guava

Elements: []
Elements: [Berry, Mango, Orange, Apple, Mango, Guava]
Berry
Mango
Orange
Apple
Mango
Guava

Elements: []
Elements: [Berry, Mango, Orange, Apple, Mango, Guava]

```

Berry  
Mango  
Orange  
Apple  
Mango  
Guava

Elements: []

Elements: [120, Y, 180.122, true]

## LinkedList:

-----  
The object of LinkedList supports duplicates elements and it stores elements based on the index, the index always starts with zero here, it stores elements in the memory based on the order of adding

The LinkedList implements Queue interface so it supports first in first out

## How to create an object for LinkedList?

Ans:

```
List<Generics> obj=new LinkedList<Generics>();  
LinkedList<Generics> obj= new LinkedList<Generics>();
```

### Without Generics

```
List obj=new LinkedList();  
LinkedList obj=new LinkedList();
```

## Example:

```
package com.sgtesting.utildemo;  
  
import java.util.Iterator;  
import java.util.LinkedList;  
import java.util.ListIterator;  
  
public class LinkedListDemo  
{  
    public static void main(String[] args)  
    {  
        addElements();  
        removeElements();  
        readElements();  
        readElements1();  
        readElementsByIterator();  
        readElementsByListIterator();  
        withoutGenerics();  
        QueueDemo();  
    }  
  
    private static void addElements()  
    {  
        LinkedList<String> obj=new LinkedList<String>();  
        System.out.println("Elements: "+obj);  
  
        obj.add("Mango");  
        obj.add("Apple");  
        obj.add("Orange");  
        obj.add(0,"Berry");  
        obj.add("Mango");  
    }  
}
```

```

        obj.add("Guava");
        System.out.println("Elements: "+obj);

        LinkedList<String> obj1=new LinkedList<String>();
        obj1.add("White");
        obj1.add("Blue");
        obj1.add("Green");
        obj.addAll(obj1);
        System.out.println("Elements: "+obj);
    }

    private static void removeElements()
    {
        LinkedList<String> obj=new LinkedList<String>();
        System.out.println("Elements: "+obj);

        obj.add("Mango");
        obj.add("Apple");
        obj.add("Orange");
        obj.add(0, "Berry");
        obj.add("Mango");
        obj.add("Guava");
        System.out.println("Elements: "+obj);

        // remove based on element
        obj.remove("Apple");
        System.out.println("Elements: "+obj);

        // remove based on index
        obj.remove(0);
        System.out.println("Elements: "+obj);

        // remove based on all
        obj.removeAll(obj);
        System.out.println("Elements: "+obj);
    }

    private static void readElements()
    {
        LinkedList<String> obj=new LinkedList<String>();
        System.out.println("Elements: "+obj);

        obj.add("Apple");
        obj.add("Mango");
        obj.add("Orange");
        obj.add(0, "Berry");
        obj.add("Mango");
        obj.add("Guava");
        System.out.println("Elements: "+obj);

        // read elements using for each loop
        for(String str:obj)
        {
            System.out.println("Elements: "+str);
        }
    }

    private static void readElements1()

```



```

{
    LinkedList<String> obj=new LinkedList<String>();
    System.out.println("Elements: "+obj);

    obj.add("Apple");
    obj.add("Mango");
    obj.add("Orange");
    obj.add(0, "Berry");
    obj.add("Mango");
    obj.add("Guava");
    System.out.println("Elements: "+obj);

    // read elements using for loop
    for(int i=0;i<obj.size();i++)
    {
        System.out.println(obj.get(i));
    }
}

private static void readElementsByIterator()
{
    LinkedList<String> obj=new LinkedList<String>();
    System.out.println("Elements: "+obj);

    obj.add("Apple");
    obj.add("Mango");
    obj.add("Orange");
    obj.add(0, "Berry");
    obj.add("Mango");
    obj.add("Guava");
    System.out.println("Elements: "+obj);

    // read elements by Iterator
    Iterator<String> ite=obj.iterator();
    while(ite.hasNext())
    {
        System.out.println(ite.next());
    }
}

private static void readElementsByListIterator()
{
    LinkedList<String> obj=new LinkedList<String>();
    System.out.println("Elements: "+obj);

    obj.add("Apple");
    obj.add("Mango");
    obj.add("Orange");
    obj.add(0, "Berry");
    obj.add("Mango");
    obj.add("Guava");
    System.out.println("Elements: "+obj);

    ListIterator<String> ite=obj.listIterator();
    System.out.println("Forward direction:");
    while(ite.hasNext())
    {
        System.out.println(ite.next());
    }
}

```

```

    }

    System.out.println("Backward direction:");
    while(ite.hasPrevious())
    {
        System.out.println(ite.previous());
    }
}

private static void withoutGenerics()
{
    LinkedList obj=new LinkedList();
    System.out.println("Elements: "+obj);

    obj.add(120);
    obj.add("Y");
    obj.add(180.122);
    obj.add(true);
    System.out.println("Elements: "+obj);
}

private static void QueueDemo()
{
    Queue<Integer> obj=new Queue<Integer> ();
    System.out.println("Elements: "+obj);

    obj.add(100);
    obj.add(200);
    obj.add(300);
    obj.add(400);
    obj.add(500);
    System.out.println("Elements: "+obj);

    Obj.remov();
    System.out.println("Elements: "+ obj);
}
}

```

### Output:

```

Elements: []
Elements: [Berry, Mango, Apple, Orange, Mango, Guava]
Elements: [Berry, Mango, Apple, Orange, Mango, Guava, White, Blue, Green]

Elements: []
Elements: [Berry, Mango, Apple, Orange, Mango, Guava]
Elements: [Berry, Mango, Orange, Mango, Guava]
Elements: [Mango, Orange, Mango, Guava]
Elements: []

Elements: []
Elements: [Berry, Apple, Mango, Orange, Mango, Guava]
Elements: Berry
Elements: Apple
Elements: Mango
Elements: Orange
Elements: Mango
Elements: Guava

```

Elements: []  
Elements: [Berry, Apple, Mango, Orange, Mango, Guava]  
Berry  
Apple  
Mango  
Orange  
Mango  
Guava

Elements: []  
Elements: [Berry, Apple, Mango, Orange, Mango, Guava]  
Berry  
Apple  
Mango  
Orange  
Mango  
Guava

Elements: []  
Elements: [Berry, Apple, Mango, Orange, Mango, Guava]  
Forward direction:  
Berry  
Apple  
Mango  
Orange  
Mango  
Guava

Backward direction:  
Guava  
Mango  
Orange  
Mango  
Apple  
Berry

Elements: []  
Elements: [120, Y, 180.122, true]

Elements: []  
Elements: [100, 200, 300, 400, 500]  
Elements: [200, 300, 400, 500]

### **Vector:**

-----

Vector is a legacy class it has reconstructed to extend AbstractList and implementing ListInterface has similar to an ArrayList

### **How to create an object for Vector?**

**Ans:**

```
List<Generics> obj=new Vector<Generics>();  
Vector<Generics> obj=new Vector<Generics>();
```

-----  
**Without Generics**

```
List obj=new Vector();  
Vector obj=new Vector();
```

### **Example:**

```

package com.sgtesting.utildemo;

import java.util.Enumeration;
import java.util.Iterator;
import java.util.Vector;

public class VectorDemo
{
    public static void main(String[] args)
    {
        addElements();
        removeElements();
        readElements1();
        readElements2();
        readElementsByIterator();
        readElementsByEnumeration();
        withoutGenerics();
    }

    private static void addElements()
    {
        Vector<String> obj=new Vector<String>();
        System.out.println("Elements :"+obj);
        obj.add("Mango");
        obj.add("Apple");
        obj.add("Orange");
        obj.add(0,"Berry");
        obj.add("Mango");
        obj.add("Guava");
        System.out.println("Elements :"+obj);
        Vector<String> obj1=new Vector<String>();
        obj1.add("White");
        obj1.add("Blue");
        obj1.add("Green");
        obj.addAll(obj1);
        System.out.println("Elements :"+obj);
    }

    private static void removeElements()
    {
        Vector<String> obj=new Vector<String>();
        System.out.println("Elements :"+obj);
        obj.add("Mango");
        obj.add("Apple");
        obj.add("Orange");
        obj.add(0,"Berry");
        obj.add("Pineapple");
        obj.add("Guava");
        System.out.println("Elements :"+obj);
        obj.remove("Apple");
        System.out.println("Elements :"+obj);
        obj.remove(0);
        System.out.println("Elements :"+obj);
        obj.removeAll(obj);
        System.out.println("Elements :"+obj);
    }

    private static void readElements1()

```

```

{
    Vector<String> obj=new Vector<String>();
    System.out.println("Elements :"+obj);
    obj.add("Mango");
    obj.add("Apple");
    obj.add("Orange");
    obj.add(0,"Berry");
    obj.add("Pineapple");
    obj.add("Guava");
    System.out.println("Elements :"+obj);
    for(String str:obj)
    {
        System.out.println(str);
    }
}

```

```

private static void readElements2()
{
    Vector<String> obj=new Vector<String>();
    System.out.println("Elements :"+obj);
    obj.add("Mango");
    obj.add("Apple");
    obj.add("Orange");
    obj.add(0,"Berry");
    obj.add("Pineapple");
    obj.add("Guava");
    System.out.println("Elements :"+obj);
    for(int i=0;i<obj.size();i++)
    {
        System.out.println(obj.get(i));
    }
}

```

```

private static void readElementsByIterator()
{
    Vector<String> obj=new Vector<String>();
    System.out.println("Elements :"+obj);
    obj.add("Mango");
    obj.add("Apple");
    obj.add("Orange");
    obj.add(0,"Berry");
    obj.add("Pineapple");
    obj.add("Guava");
    System.out.println("Elements :"+obj);
    Iterator<String> ite=obj.iterator();
    while(ite.hasNext())
    {
        System.out.println(ite.next());
    }
}

```

```

private static void readElementsByEnumeration()
{
    Vector<String> obj=new Vector<String>();
    System.out.println("Elements :"+obj);
    obj.add("Mango");
    obj.add("Apple");
    obj.add("Orange");
}

```

```

        obj.add(0,"Berry");
        obj.add("Pineapple");
        obj.add("Guava");
        System.out.println("Elements :"+obj);
        Enumeration<String> ite=obj.elements();
        while(ite.hasMoreElements())
        {
            System.out.println(ite.nextElement());
        }
    }

    private static void withoutGenerics()
    {
        Vector obj=new Vector();
        System.out.println("Elements :"+obj);
        obj.add(120);
        obj.add("Y");
        obj.add(800.125);
        obj.add(true);
        System.out.println("Elements :"+obj);
    }
}

```

### Output:

```

Elements :[]
Elements :[Berry, Mango, Apple, Orange, Mango, Guava]
Elements :[Berry, Mango, Apple, Orange, Mango, Guava, White, Blue, Green]

```

```

Elements :[]
Elements :[Berry, Mango, Apple, Orange, Pineapple, Guava]
Elements :[Berry, Mango, Orange, Pineapple, Guava]
Elements :[Mango, Orange, Pineapple, Guava]
Elements :[]

```

```

Elements :[]
Elements :[Berry, Mango, Apple, Orange, Pineapple, Guava]
Berry
Mango
Apple
Orange
Pineapple
Guava

```

```

Elements :[]
Elements :[Berry, Mango, Apple, Orange, Pineapple, Guava]
Berry
Mango
Apple
Orange
Pineapple
Guava

```

```

Elements :[]
Elements :[Berry, Mango, Apple, Orange, Pineapple, Guava]
Berry
Mango
Apple
Orange

```

Pineapple  
Guava

Elements :[]  
Elements :[Berry, Mango, Apple, Orange, Pineapple, Guava]  
Berry  
Mango  
Apple  
Orange  
Pineapple  
Guava

Elements :[]  
Elements :[120, Y, 800.125, true]

### Stack:

-----

Stack is a sub class of 'vector', the object of stack supports all the functionalities of vector along with it supports unique functionality that is FILO (First In Last Out)

### Example:

```
package com.sgtesting.utildemo;

import java.util.Stack;
public class StackDemo
{
    public static void main(String[] args)
    {
        Stack<Integer> st=new Stack<Integer>();
        pushDemo(st, 100);
        pushDemo(st, 200);
        pushDemo(st, 300);
        pushDemo(st, 400);
        pushDemo(st, 500);
        popDemo(st);
    }

    private static void pushDemo(Stack<Integer> st,int a)
    {
        st.push(a);
        System.out.println("Element :"+a);
    }

    private static void popDemo(Stack<Integer> st)
    {
        int a=st.pop();
        System.out.println("Removed Element :"+a);
    }
}
```

### Output:

Element :100  
Element :200  
Element :300  
Element :400  
Element :500  
Removed Element :500

## Set:

----

Set is an interface, Set never accepts support duplicate records(elements) and never supports index

Set supports elements based on the hash code it is an imaginary concept in core java

### There are following classes implementing Set interfaces

- 1) HashSet
- 2) TreeSet

**HashSet:** the object of 'HashSet' never supports duplicate elements and never supports index, it stores elements randomly in the memory

### How to create an object for HashSet?

Ans:

```
Set<Generics> obj=new HashSet<Generics>();  
HashSet<Generics> obj=new HashSet<Generics>();
```

-----

#### Without Generics

```
Set obj=new HashSet();  
HashSet obj=new HashSet();
```

### Example:

```
package com.sgtesting.utildemo;  
  
import java.util.HashSet;  
import java.util.Iterator;  
  
public class HashSetDemo  
{  
    public static void main(String[] args)  
    {  
        addElements();  
        removeElements();  
        readElements();  
        readElementsByIterator();  
        convertToArray();  
        withoutGenerics();  
    }  
  
    private static void addElements()  
    {  
        HashSet<String> obj=new HashSet<String>();  
        System.out.println("Elements :"+obj);  
        obj.add("Mango");  
        obj.add("Apple");  
        obj.add("Orange");  
        obj.add("Grapes");  
        obj.add("Mango");  
        obj.add("Watermelon");  
        System.out.println("Elements :"+obj);  
        HashSet<String> obj1=new HashSet<String>();  
        obj1.add("White");  
        obj1.add("Green");  
        obj1.add("Blue");  
        obj.addAll(obj1);  
    }  
}
```



```

        System.out.println("Elements :"+obj);
    }

    private static void removeElements()
    {
        HashSet<String> obj=new HashSet<String>();
        System.out.println("Elements :"+obj);
        obj.add("Mango");
        obj.add("Apple");
        obj.add("Orange");
        obj.add("Grapes");
        obj.add("Dragon Fruit");
        obj.add("Watermelon");
        System.out.println("Elements :"+obj);
        obj.remove("Grapes");
        System.out.println("Elements :"+obj);
        obj.removeAll(obj);
        System.out.println("Elements :"+obj);
    }

    private static void readElements1()
    {
        HashSet<String> obj=new HashSet<String>();
        System.out.println("Elements :"+obj);
        obj.add("Mango");
        obj.add("Apple");
        obj.add("Orange");
        obj.add("Grapes");
        obj.add("Dragon Fruit");
        obj.add("Watermelon");
        System.out.println("Elements :"+obj);
        for(String str:obj)
        {
            System.out.println(str);
        }
    }

    private static void readElementsByIterator()
    {
        HashSet<String> obj=new HashSet<String>();
        System.out.println("Elements :"+obj);
        obj.add("Mango");
        obj.add("Apple");
        obj.add("Orange");
        obj.add("Grapes");
        obj.add("Dragon Fruit");
        obj.add("Watermelon");
        System.out.println("Elements :"+obj);
        Iterator<String> ite=obj.iterator();
        while(ite.hasNext())
        {
            System.out.println(ite.next());
        }
    }

    private static void convertToArray()
    {
        HashSet<String> obj=new HashSet<String>();

```

```

        System.out.println("Elements :"+obj);
        obj.add("Mango");
        obj.add("Apple");
        obj.add("Orange");
        obj.add("Grapes");
        obj.add("Dragon Fruit");
        obj.add("Watermelon");
        System.out.println("Elements :"+obj);
        Object a[]=obj.toArray();
        for(int i=0;i<a.length;i++)
        {
            System.out.println(a[i]);
        }
    }
}

```

```

private static void withoutGenerics()
{
    HashSet obj=new HashSet();
    System.out.println("Elements :"+obj);
    obj.add(120);
    obj.add('Y');
    obj.add(800.125);
    obj.add(true);
    System.out.println("Elements :"+obj);
}
}

```

#### Output:

```

Elements :[]
Elements :[Apple, Grapes, Watermelon, Mango, Orange]
Elements :[Apple, White, Grapes, Watermelon, Blue, Mango, Orange, Green]

```

```

Elements :[]
Elements :[Apple, Grapes, Watermelon, Mango, Dragon Fruit, Orange]
Elements :[Apple, Watermelon, Mango, Dragon Fruit, Orange]
Elements :[]

```

```

Elements :[]
Elements :[Apple, Grapes, Watermelon, Mango, Dragon Fruit, Orange]
Apple
Grapes
Watermelon
Mango
Dragon Fruit
Orange

```

```

Elements :[]
Elements :[Apple, Grapes, Watermelon, Mango, Dragon Fruit, Orange]
Apple
Grapes
Watermelon
Mango
Dragon Fruit
Orange

```

```

Elements :[]
Elements :[Apple, Grapes, Watermelon, Mango, Dragon Fruit, Orange]
Apple

```

Grapes  
Watermelon  
Mango  
Dragon Fruit  
Orange

Elements :[]  
Elements :[120, Y, 800.125, true]

**TreeSet:** the object of 'TreeSet' doesn't support duplicate elements and index, it stores elements in the memory based on stored ascending order

At any point TreeSet never support without Generic approach of adding elements

### How to create an object for TreeSet?

**Ans:**

```
Set<Generics> obj=new TreeSet<Generics>();  
TreeSet<Generics> obj=new TressSet<Generics>();
```

**Example:**

```
package com.sgtesting.utildemo;  
  
import java.util.Iterator;  
import java.util.TreeSet;  
  
public class TreeSetDemo  
{  
    public static void main(String[] args)  
    {  
        addElements();  
        removeElements();  
        readElements1();  
        readElementsByIterator();  
        convertToArray();  
    }  
  
    private static void addElements()  
    {  
        TreeSet<String> obj=new TreeSet<String>();  
        System.out.println("Elements :"+obj);  
        obj.add("Mango");  
        obj.add("Apple");  
        obj.add("Orange");  
        obj.add("Grapes");  
        obj.add("Mango");  
        obj.add("Watermelon");  
        System.out.println("Elements :"+obj);  
        TreeSet<String> obj1=new TreeSet<String>();  
        obj1.add("White");  
        obj1.add("Green");  
        obj1.add("Blue");  
        obj.addAll(obj1);  
        System.out.println("Elements :"+obj);  
    }  
  
    private static void removeElements()  
    {
```

```

TreeSet<String> obj=new TreeSet<String>();
System.out.println("Elements :"+obj);
obj.add("Mango");
obj.add("Apple");
obj.add("Orange");
obj.add("Grapes");
obj.add("Dragon Fruit");
obj.add("Watermelon");
System.out.println("Elements :"+obj);
obj.remove("Grapes");
System.out.println("Elements :"+obj);
obj.removeAll(obj);
System.out.println("Elements :"+obj);
}

```

```

private static void readElements1()
{
    TreeSet<String> obj=new TreeSet<String>();
    System.out.println("Elements :"+obj);
    obj.add("Mango");
    obj.add("Apple");
    obj.add("Orange");
    obj.add("Grapes");
    obj.add("Dragon Fruit");
    obj.add("Watermelon");
    System.out.println("Elements :"+obj);
    for(String str:obj)
    {
        System.out.println(str);
    }
}

```

```

private static void readElementsByIterator()
{
    TreeSet<String> obj=new TreeSet<String>();
    System.out.println("Elements :"+obj);
    obj.add("Mango");
    obj.add("Apple");
    obj.add("Orange");
    obj.add("Grapes");
    obj.add("Dragon Fruit");
    obj.add("Watermelon");
    System.out.println("Elements :"+obj);
    Iterator<String> ite=obj.iterator();
    while(ite.hasNext())
    {
        System.out.println(ite.next());
    }
}

```

```

private static void convertToArray()
{
    TreeSet<String> obj=new TreeSet<String>();
    System.out.println("Elements :"+obj);
    obj.add("Mango");
    obj.add("Apple");
    obj.add("Orange");
    obj.add("Grapes");
}

```

```

        obj.add("Dragon Fruit");
        obj.add("Watermelon");
        System.out.println("Elements :"+obj);
        Object[] a=obj.toArray();
        for(int i=0;i<a.length;i++)
        {
            System.out.println(a[i]);
        }
    }
}

```

### Output:

```

Elements :[]
Elements :[Apple, Grapes, Mango, Orange, Watermelon]
Elements :[Apple, Blue, Grapes, Green, Mango, Orange, Watermelon, White]

Elements :[]
Elements :[Apple, Dragon Fruit, Grapes, Mango, Orange, Watermelon]
Elements :[Apple, Dragon Fruit, Mango, Orange, Watermelon]
Elements :[]

Elements :[]
Elements :[Apple, Dragon Fruit, Grapes, Mango, Orange, Watermelon]
Apple
Dragon Fruit
Grapes
Mango
Orange
Watermelon

Elements :[]
Elements :[Apple, Dragon Fruit, Grapes, Mango, Orange, Watermelon]
Apple
Dragon Fruit
Grapes
Mango
Orange
Watermelon

Elements :[]
Elements :[Apple, Dragon Fruit, Grapes, Mango, Orange, Watermelon]
Apple
Dragon Fruit
Grapes
Mango
Orange
Watermelon

```

### Map Interface:

Map interface supports key and value pair of adding elements, here keys are always unique

The Map Interfaces has implemented by the following 2 classes:

- 1) HashMap
- 2) TreeMap

**HashMap:** The object of HashMap supports key and value pair of adding elements, here key's always must be unique, it stores keys randomly in the memory

The HashMap supports only one occurrence of null keys

### How to create the object for HashMap?

Ans:

```
Map<Generics, Generics> obj=new HashMap<Generics, Generics>();  
HashMap<Generics, Generics> obj=new HashMap<Generics, Generics>();
```

---

Without Generics

```
Map obj=new HashMap();  
HashMap obj=new HashMap();
```

### Example:

```
package com.sgtesting.utildemo;  
import java.util.HashMap;  
public class HashMapDemo  
{  
    public static void main(String[] args)  
    {  
        addElements();  
        readAndRemoveElements();  
        readElements();  
    }  
  
    private static void addElements()  
    {  
        HashMap<String,String> obj=new HashMap<String,String>();  
        System.out.println("Elements :"+obj);  
        obj.put("camel", "Camel is a ship of the desrt");  
        obj.put("lotus", "Lotus is a national flower of India");  
        obj.put("peacock", "Peacock is a national bird of India");  
        obj.put("tiger", "Tiger is a national animal of India");  
        obj.put("raichur", "Raichur is a palace city and clean city of Karnataka");  
        obj.put("mango", "Mango is a king of all fruits");  
        obj.put("bangalore", "Bangalore is a garden city of Karnataka");  
        obj.put(null, "HashMap object supports Null Keys");  
        System.out.println("Elements :"+obj);  
    }  
  
    private static void readAndRemoveElements()  
    {  
        HashMap<String,String> obj=new HashMap<String,String>();  
        System.out.println("Elements :"+obj);  
        obj.put("camel", "Camel is a ship of the desrt");  
        obj.put("lotus", "Lotus is a national flower of India");  
        obj.put("peacock", "Peacock is a national bird of India");  
        obj.put("tiger", "Tiger is a national animal of India");  
        obj.put("raichur", "Raichur is a palace city and clean city of Karnataka");  
        obj.put("mango", "Mango is a king of all fruits");  
        obj.put("bangalore", "Bangalore is a garden city of Karnataka");  
        obj.put(null, "HashMap object supports Null Keys");  
        System.out.println("Elements :"+obj);  
  
        //Read Value  
        String v1=obj.get("lotus");  
        System.out.println(v1);  
        obj.remove("lotus");  
        String v2=obj.get("lotus");
```

```

        System.out.println(v2);
    }

    private static void readElements()
    {
        HashMap<String,String> obj=new HashMap<String,String>();
        System.out.println("Elements :"+obj);
        obj.put("camel", "Camel is a ship of the desrt");
        obj.put("lotus", "Lotus is a national flower of India");
        obj.put("peacock", "Peacock is a national bird of India");
        obj.put("tiger", "Tiger is a national animal of India");
        obj.put("raichur", "Raichur is a palace city and clean city of Karnataka");
        obj.put("mango", "Mango is a king of all fruits");
        obj.put("bangalore", "Bangalore is a garden city of Karnataka");
        obj.put(null, "HashMap object supports Null Keys");
        System.out.println("Elements :"+obj);

        obj.forEach((k,v) -> System.out.println(k + " ----> "+v));
    }
}

```

### Output:

Elements :{ }

Elements :{null=HashMap object supports Null Keys, bangalore=Bangalore is a garden city of Karnataka, camel=Camel is a ship of the desrt, peacock=Peacock is a national bird of India, lotus=Lotus is a national flower of India, tiger=Tiger is a national animal of India, raichur=Raichur is a palace city and clean city of Karnataka, mango=Mango is a king of all fruits }

Elements :{ }

Elements :{null=HashMap object supports Null Keys, bangalore=Bangalore is a garden city of Karnataka, camel=Camel is a ship of the desrt, peacock=Peacock is a national bird of India, lotus=Lotus is a national flower of India, tiger=Tiger is a national animal of India, raichur=Raichur is a palace city and clean city of Karnataka, mango=Mango is a king of all fruits }

Lotus is a national flower of India

null

Elements :{ }

Elements :{null=HashMap object supports Null Keys, bangalore=Bangalore is a garden city of Karnataka, camel=Camel is a ship of the desrt, peacock=Peacock is a national bird of India, lotus=Lotus is a national flower of India, tiger=Tiger is a national animal of India, raichur=Raichur is a palace city and clean city of Karnataka, mango=Mango is a king of all fruits }

null ----> HashMap object supports Null Keys

bangalore----> Bangalore is a garden city of Karnataka

camel ----> Camel is a ship of the desrt

peacock----> Peacock is a national bird of India

lotus----> Lotus is a national flower of India

tiger ----> Tiger is a national animal of India

raichur ----> Raichur is a palace city and clean city of Karnataka

mango ----> Mango is a king of all fruits

**TreeMap:** The object of TreeMap supports key and value pair of adding elements, here keys are always must be unique, it supports keys in a memory based on the sorted ascending order

At any point TreeMap object never support null keys, TreeMap never supports creation of object without Generics

## How to create an object for TreeMap?

Ans:

```
Map<Generics, Generics> obj=new TreeMap<Generics, Generics>();  
TreeMap<Generics, Generics> obj=new TreeMap<Generics, Generics>();
```

Example:

```
package com.sgtesting.utildemo;  
  
import java.util.TreeMap;  
  
public class TreeMapDemo  
{  
    public static void main(String[] args)  
    {  
        addElements();  
        readAndRemoveElements();  
        readElements();  
    }  
  
    private static void addElements()  
    {  
        TreeMap<String,String> obj=new TreeMap<String,String>();  
        System.out.println("Elements :"+obj);  
        obj.put("camel", "Camel is a ship of the desrt");  
        obj.put("lotus", "Lotus is a national flower of India");  
        obj.put("peacock", "Peacock is a national bird of India");  
        obj.put("tiger", "Tiger is a national animal of India");  
        obj.put("raichur", "Raichur is a palace city and clean city of Karnataka");  
        obj.put("mango", "Mango is a king of all fruits");  
        obj.put("bangalore", "Bangalore is a garden city of Karnataka");  
        System.out.println("Elements :"+obj);  
    }  
  
    private static void readAndRemoveElements()  
    {  
        TreeMap<String,String> obj=new TreeMap<String,String>();  
        System.out.println("Elements :"+obj);  
        obj.put("camel", "Camel is a ship of the desrt");  
        obj.put("lotus", "Lotus is a national flower of India");  
        obj.put("peacock", "Peacock is a national bird of India");  
        obj.put("tiger", "Tiger is a national animal of India");  
        obj.put("raichur", "Raichur is a palace city and clean city of Karnataka");  
        obj.put("mango", "Mango is a king of all fruits");  
        obj.put("bangalore", "Bangalore is a garden city of Karnataka");  
        System.out.println("Elements :"+obj);  
        //Read Value  
        String v1=obj.get("lotus");  
        System.out.println(v1);  
        obj.remove("lotus");  
        String v2=obj.get("lotus");  
        System.out.println(v2);  
    }  
  
    private static void readElements()  
    {  
        TreeMap<String,String> obj=new TreeMap<String,String>();  
        System.out.println("Elements :"+obj);  
    }  
}
```



```

obj.put("camel", "Camel is a ship of the desrt");
obj.put("lotus", "Lotus is a national flower of India");
obj.put("peacock", "Peacock is a national bird of India");
obj.put("tiger", "Tiger is a national animal of India");
obj.put("raichur", "Raichur is a palace city and clean city of Karnataka");
obj.put("mango", "Mango is a king of all fruits");
obj.put("bangalore", "Bangalore is a garden city of Karnataka");
System.out.println("Elements :" +obj);

obj.forEach((k,v) -> System.out.println(k+" --> "+v));
    }
}

```

### Output:

```

Elements :{ }
Elements :{ bangalore=Bangalore is a garden city of Karnataka, camel=Camel is a ship of the desrt,
lotus=Lotus is a national flower of India, mango=Mango is a king of all fruits, peacock=Peacock is a
national bird of India, raichur=Raichur is a palace city and clean city of Karnataka, tiger=Tiger is a
national animal of India}

Elements :{ }
Elements :{ bangalore=Bangalore is a garden city of Karnataka, camel=Camel is a ship of the desrt,
lotus=Lotus is a national flower of India, mango=Mango is a king of all fruits, peacock=Peacock is a
national bird of India, raichur=Raichur is a palace city and clean city of Karnataka, tiger=Tiger is a
national animal of India}
Lotus is a national flower of India
null

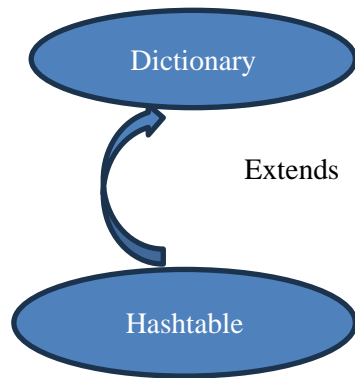
Elements :{ }
Elements :{ bangalore=Bangalore is a garden city of Karnataka, camel=Camel is a ship of the desrt,
lotus=Lotus is a national flower of India, mango=Mango is a king of all fruits, peacock=Peacock is a
national bird of India, raichur=Raichur is a palace city and clean city of Karnataka, tiger=Tiger is a
national animal of India}
bangalore --> Bangalore is a garden city of Karnataka
camel --> Camel is a ship of the desrt
lotus --> Lotus is a national flower of India
mango --> Mango is a king of all fruits
peacock --> Peacock is a national bird of India
raichur --> Raichur is a palace city and clean city of Karnataka
tiger --> Tiger is a national animal of India

```

### Legacy classes:

1. Vector
2. Stack
3. Dictionary
4. Hash table
5. Properties

Legacy classes:- these classes are introduced before collection framework



**Dictionary:** It is an abstract class it has extended by Hashtable.

**Hash Table:** The object of Hashtable supports key and value pair of adding elements, here keys are always unique and it never support null keys.  
The Hashtable stores keys randomly in the memory.

How to create an object for Hashtable :

```
Hashtable<Generics, Generics> obj=new Hashtable<Generics, Generics>();  
Dictionary<Generics, Generics> obj=new Hashtable<Generics, Generics>();
```

-----  
Without Generics

```
Hashtable obj=new Hashtable();  
Dictionary obj=new Hashtable();
```

## Program:

```
package com.sgtesting.utildemo;  
import java.util.Hashtable;  
public class HashtableDemo {  
    public static void main(String[] args) {  
        //    addElements();  
        readAndRemoveElements();  
    }  
  
    private static void addElements()  
    {  
        Hashtable<String,String> obj=new Hashtable<String,String>();  
        System.out.println("Elements :"+obj);  
        obj.put("camel", "Camel is a ship of the desrt");  
        obj.put("lotus", "Lotus is a national flower of India");  
        obj.put("peacock", "Peacock is a national bird of India");  
        obj.put("tiger", "Tiger is a national animal of India");  
        obj.put("raichur", "Raichur is a palace city of Karnataka");  
        obj.put("mango", "Mango is a king of all fruits");  
        obj.put("bangalore", "Bangalore is a garden city of Karnataka");  
        System.out.println("Elements :"+obj);  
    }  
  
    private static void readAndRemoveElements()  
    {  
        Hashtable<String,String> obj=new Hashtable<String,String>();  
        System.out.println("Elements :"+obj);  
        obj.put("camel", "Camel is a ship of the desrt");  
        obj.put("lotus", "Lotus is a national flower of India");  
        obj.put("peacock", "Peacock is a national bird of India");  
    }  
}
```

```

obj.put("tiger", "Tiger is a national animal of India");
obj.put("raichur", "Raichur is a palace city of Karnataka");
obj.put("mango", "Mango is a king of all fruits");
obj.put("bangalore", "Bangalore is a garden city of Karnataka");
System.out.println("Elements :"+obj);
//Read Value
String v1=obj.get("peacock");
System.out.println(v1);
obj.remove("tiger");
String v2=obj.get("tiger");
System.out.println(v2);

}
}
Output:
Elements :{}
Elements :{tiger=Tiger is a national animal of India, lotus=Lotus is a national
flower of India, bangalore=Bangalore is a garden city of Karnataka, camel=Camel
is a ship of the desert, peacock=Peacock is a national bird of India, mango=Mango
is a king of all fruits, raichur=Raichur is a palace city of Karnataka}
Peacock is a national bird of India
null

```

## Properties:

By using properties object we can add key and value pair into a (dot) **.properties** file

### Program:

```

package com.sgtesting.utildemo;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.Properties;

public class PropertiesDemo {
    public static void main(String[] args) {
        // writeContent();
        readContent();
    }
    /**
     * Write key and value pair into a file
     */
    private static void writeContent()
    {
        FileOutputStream fout=null;
        Properties prop=null;
        try
        {
            fout=new
FileOutputStream("C:\\Users\\Bablu\\Desktop\\property\\Test.properties");
            prop=new Properties();

            prop.setProperty("username", "admin");
            prop.setProperty("password", "manager");
            prop.setProperty("pin", "2222");

            prop.store(fout, "It is an example");
        } catch (Exception e)
        {
            e.printStackTrace();
        }
        finally
        {
            try
            {

```

```

        fout.close();
        prop.clear();
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}

//Read value from the .properties file
private static void readContent()
{
    FileInputStream fin=null;
    Properties prop=null;
    try
    {
        fin=new
FileInputStream("C:\\Users\\Bablu\\Desktop\\property\\Test.properties");
        prop=new Properties();

        prop.load(fin);

        String v1=prop.getProperty("username");
        System.out.println(v1);
        String v2=prop.getProperty("password");
        System.out.println(v2);
        String v3=prop.getProperty("pin");
        System.out.println(v3);
    } catch (Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        try
        {
            fin.close();
            prop.clear();
        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}

}
Output:-      admin
              manager
              2222

```

## **Serialization & De-Serialization:** (This is available java 8 on words)

**Serialization:** It is a process of writing object or instance into a byte stream.

Serialization can be achieved by using serializable interface, this interface usually called as marker interface.

Note: if we write any object or instance into a file the file extension should be **.ser**

**Program:****(Create employee class)**

```
package com.sgtesting.java8;

import java.io.Serializable;

public class Employee implements Serializable {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private int empNo;
    private String employeeName;
    private String jobName;
    private int salary;

    public Employee(int empno, String employeeName, String jobName, int salary) {
        super();
        this.empNo = empno;
        this.employeeName = employeeName;
        this.jobName = jobName;
        this.salary = salary;
    }

    public void showEmployeeNumber() {
        System.out.println("Employee number"+empNo);
    }

    public void showEmployeeName() {
        System.out.println("Employee name"+employeeName);
    }

    public void showJobName() {
        System.out.println("Job name"+jobName);
    }

    public void showSalary() {
        System.out.println("Salary"+salary);
    }
}
```

**(create serialization demo)**

```
package com.sgtesting.java8;

import java.io.FileOutputStream;
import java.io.ObjectOutputStream;

public class SerilalizationDemo {

    public static void main(String[] args) {
        writeObjectInToFile();
    }

    private static void writeObjectInToFile() {
        FileOutputStream fout = null;
        ObjectOutputStream out = null;
        Employee obj = null;
        try {
            fout=newFileOutputStream("C:\\Users\\Bablu\\Desktop\\serialization\\Employee.ser");
            out= new ObjectOutputStream(fout);
            obj=new Employee(10, "santosh", "manager", 1000000);
            out.writeObject(obj);
        } catch (Exception e) {
            e.printStackTrace();
        }
        finally {
            try {
                out.close();
            } catch (Exception e) {
            }
        }
    }
}
```

```

        e.printStackTrace();
    }
}
}
}

```

Output :- it's create .ser file("C:\Users\Bablu\Desktop\serialization\Employee.ser")

## De-serialization:

Reading object or instance from byte stream or files is called as de-serialization.

### Program:

(Create De-Serialization Demo)

```

package com.sgtesting.java8;

import java.io.FileInputStream;
import java.io.ObjectInputStream;

public class DeSerializationDemo {

    public static void main(String[] args) {
        readObjectFromFile();
    }

    private static void readObjectFromFile() {
        FileInputStream fin=null;
        ObjectInputStream in =null;
        Employee obj=null;
        try {
            fin=new
FileInputStream("C:\\Users\\Bablu\\Desktop\\serialization\\Employee.ser");
            in=new ObjectInputStream(fin);
            obj=(Employee)in.readObject();
            obj.showEmployeeNumber();
            obj.showEmployeeName();
            obj.showJobName();
            obj.showSalary();
        } catch (Exception e) {
            e.printStackTrace();
        }
        finally {
            try {
                in.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

}

```

Output :

```

Employee number10
Employee namesantosh
Job namemanager
Salary1000000

```

## Interface(java8)

In java8 interface they introduced default and static method

Case 1 : If interface contains default method.

```

package com.sgtesting.interfaceprogram;

interface Product {

```

```

        void showProductName(String name);

        default void showProductPrice(double price) {
            System.out.println("Price of the product :-" + price);
        }
    }

    class Lenove implements Product {

        @Override
        public void showProductName(String name) {
            System.out.println("product name :-" + name);
        }

    }

    public class InterfaceDemo {

        public static void main(String[] args) {
            Lenove laptop = new Lenove();
            laptop.showProductName("HP pavallion ");
            laptop.showProductPrice(100000.00);
        }

    }

    Output:      product name :-HP pavallion
                Price of the product :-100000.0

```

Case 2: If interface contains static method.

```

package com.sgtesting.interfaceprogram;

interface ProductDemo
{
    void showProductName(String name);
    static void showProductPrice(double price) {
        System.out.println("Price of the product: "+price);
    }
}

class Hp implements ProductDemo{

    @Override
    public void showProductName(String name) {
        System.out.println("product name : "+name);
    }

}

public class InterfaceDemo1 {

    public static void main(String[] args) {
        Hp laptop = new Hp();
        laptop.showProductName("HP pavallion");
        ProductDemo.showProductPrice(750000.00);
    }

}

Output :      product name : HP pavallion
                Price of the product: 750000.0

```

### Case 3: If multiple inheritance all interface contains same default method with signature

```
package com.sgtesting.interfaceprogram;

interface RevaUniversity
{
    default void showUniversityName(String name) {
        System.out.println("University name :"+name);
    }
}

interface JainUniversity
{
    default void showUniversityName(String name) {
        System.out.println("University name :"+name);
    }
}

class EngineeringCollege implements RevaUniversity,JainUniversity
{
    @Override
    public void showUniversityName(String name) {
        System.out.println("university name :"+ name);
    }
}

public class InterfaceDemo2 {

    public static void main(String[] args) {
        EngineeringCollege ec = new EngineeringCollege();
        ec.showUniversityName("VTU university");
    }
}

Output: university name :VTU university
```

## JDBC Connection:

(Java Data base connectivity)

By using JDBC object model we can execute any SQL Queries in java programming.

Step 1:

```
Connection conn = null;
```

Step 2:

```
Class.forName("oracle.jdbc.driver.oracleDriver");
```

Step 3:

```
Conn=DriverManager.getConnection(String connectionURL, String user,
String pwd);
```

Step 4:

```
Statement stmt = conn.createStatement();
```

Step 5:

```
String Query ="Select * from Employee";
ResultSet rs = stmt.executeQuery(query);
```

Step 6:

```
rs.next();
```

if records available it Returns true otherwise false.



## How to configure JDBC in Eclipse

→ We can download the jar files from official web site from oracle

Or

→ In local machine navigate to oracle installed path under JDBC lib folder we can get the jar files

C:\oracle\app\oracle\product\11.2.0\server\jdbc\lib

## Program:

```
package com.sgtesting.jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;

public class DatabaseDemo {
    public static void main(String[] args) {
        // readRecordsFromTable();
        insertRecords();
    }

    private static void readRecordsFromTable() {
        Connection conn = null;
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE",
            "system", "tiger");
            System.out.println("DB Connection becomes succesfully!!");
            Statement stmt = conn.createStatement();
            String query = "select * from employee";
            ResultSet rs = stmt.executeQuery(query);
            ResultSetMetaData rsdata = rs.getMetaData();
            String first = rsdata.getColumnName(1);
            String second = rsdata.getColumnName(2);
            String third = rsdata.getColumnName(3);
            String fourth = rsdata.getColumnName(4);
            String fifth = rsdata.getColumnName(5);
            System.out.printf("%-12s", first);
            System.out.printf("%-12s", second);
            System.out.printf("%-12s", third);
            System.out.printf("%-12s", fourth);
            System.out.printf("%-12s", fifth);
            System.out.printf("\n");
            while (rs.next()) {
                String empid = rs.getString("EMPID");
                String empname = rs.getString("EMPNAME");
                String empage = rs.getString("EMPAGE");
                String empmobile = rs.getString("EMPMOBILE");
                String empaddress = rs.getString("EMPADDRESS");
                System.out.printf("%-12s", empid);
                System.out.printf("%-12s", empname);
                System.out.printf("%-12s", empage);
                System.out.printf("%-12s", empmobile);
                System.out.printf("%-12s", empaddress);
                System.out.printf("\n");
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
```

```

        conn.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private static void insertRecords() {
    Connection conn = null;
    try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE",
            "system", "tiger");
        System.out.println("DB Connection becomes succesfully!!");
        Statement stmt = conn.createStatement();
String query = "insert into employee values(23,'Ram','28','9900889966','Ayodhya)";
        stmt.executeUpdate(query);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
}

```

**Output:** DB Connection becomes successfully !!

## Using Statement:

### Program:

```

package com.sgtesting.jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;

public class SQLOperationsUsingStatementDemo {
    public static void main(String[] args) {
        // readRecordsFromTable();
        insertRecords();
    }

    private static void readRecordsFromTable()
    {
        Connection conn=null;
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            conn=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE",
                "system", "tiger");
            System.out.println("DB Connection becomes succesfully!!");
            Statement stmt=conn.createStatement();
            String query="select * from employee";
            ResultSet rs=stmt.executeQuery(query);
            ResultSetMetaData rsdata=rs.getMetaData();
            String first=rsdata.getColumnName(1);
            String second=rsdata.getColumnName(2);
            String third=rsdata.getColumnName(3);
            String fourth=rsdata.getColumnName(4);

```

```

String fifth=rsdata.getColumnName(5);
System.out.printf("%-12s", first);
System.out.printf("%-12s", second);
System.out.printf("%-12s", third);
System.out.printf("%-12s", fourth);
System.out.printf("%-12s", fifth);
System.out.printf("\n");
while(rs.next())
{
    String empid=rs.getString("EMPID");
    String empname=rs.getString("EMPNAME");
    String empage=rs.getString("EMPAGE");
    String empmobile=rs.getString("EMPMOBILE");
    String empaddress=rs.getString("EMPADDRESS");
    System.out.printf("%-12s", empid);
    System.out.printf("%-12s", empname);
    System.out.printf("%-12s", empage);
    System.out.printf("%-12s", empmobile);
    System.out.printf("%-12s", empaddress);
    System.out.printf("\n");
}
} catch (Exception e)
{
    e.printStackTrace();
}
finally
{
    try
    {
        conn.close();
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}
}

private static void insertRecords()
{
    Connection conn=null;
    try
    {
        Class.forName("oracle.jdbc.driver.OracleDriver");

        conn=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE",
        "system", "tiger");
        System.out.println("DB Connection becomes succesfully!!");
        Statement stmt=conn.createStatement();
        String query="insert into employee values(16,'laxman',33,7777667777,'Ayodhya)";
        stmt.executeUpdate(query);
    } catch (Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        try
        {
            conn.close();
        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
}

```

```
}
```

Output:

```
DB Connection becomes succesfully!!
EMPID      EMPNAME      EMPAGE      EMPMOBILE      EMPADDRESS
1           Santosh       27          7676977798     Gulbarga
2           Ramya        22          9900990099     ck halli
17          kishan       25          9955443322     bagalkot
4           Rakshith     29          7766554433     Tumkur
5           Ashok        34          1122334455     Mangalore
6           Akshay       32          2233223322     Raichur
7           syed         31          9988776688     Tumkur
8           Darshan     22          8233288899     Yadgir
9           suhas        25          9900889977     Mysore
23          santoshu     28          9900889966     pune
10          Hrutik       30          9900889988     BANGALORE
3           santosh      28          9900889966     pune
23          Ram          28          9900889966     Ayodhya
16          laxman       33          7777667777     Ayodhya
27          vishwanath   30          9944998800     Raichur
16          laxman       33          7777667777     Ayodhya
```

## Using Prepared Statement:

### Program:

```
package com.sgtesting.jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;

public class SQLOperationsUsingPreparedStatementDemo {
    public static void main(String[] args) {
        readRecordsFromTable();
        //insertRecordsData();
    }

    private static void readRecordsFromTable()
    {
        Connection conn=null;
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");

            conn=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE",
            "system", "tiger");
            System.out.println("DB Connection becomes succesfully!!");
            String query="select * from employee";
            PreparedStatement stmt=conn.prepareStatement(query);
            ResultSet rs=stmt.executeQuery();
            ResultSetMetaData rsdata=rs.getMetaData();
            String first=rsdata.getColumnName(1);
            String second=rsdata.getColumnName(2);
            String third=rsdata.getColumnName(3);
            String fourth=rsdata.getColumnName(4);
            String fifth=rsdata.getColumnName(5);
            System.out.printf("%-12s", first);
            System.out.printf("%-12s", second);
            System.out.printf("%-12s", third);
            System.out.printf("%-12s", fourth);
            System.out.printf("%-12s", fifth);
```

```

        System.out.printf("\n");
        while(rs.next())
        {
            String empid=rs.getString("EMPID");
            String empname=rs.getString("EMPNAME");
            String empage=rs.getString("EMPAGE");
            String empmobile=rs.getString("EMPMOBILE");
            String empaddress=rs.getString("EMPADDRESS");
            System.out.printf("%-12s",empid);
            System.out.printf("%-12s",empname);
            System.out.printf("%-12s",empage);
            System.out.printf("%-12s",empmobile);
            System.out.printf("%-12s",empaddress);
            System.out.printf("\n");
        }
    } catch (Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        try
        {
            conn.close();
        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}

private static void insertRecordsData()
{
    Connection conn=null;
    try
    {
        Class.forName("oracle.jdbc.driver.OracleDriver");

        conn=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE",
        "system", "tiger");
        System.out.println("DB Connection becomes succesfully!!");
        String query="insert into employee values(17,'kishan',25,9955443322,'bagalkot')";
        PreparedStatement stmt=conn.prepareStatement(query);
        stmt.executeUpdate();
    } catch (Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        try
        {
            conn.close();
        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
}
}

```

## Output: -

```
DB Connection becomes succesfully!!
EMPID      EMPNAME      EMPAGE      EMPMOBILE      EMPADDRESS
1           Santosh       27          7676977798    Gulbarga
2           Ramya        22          9900990099    ck halli
17          kishan       25          9955443322    bagalkot
4           Rakshith     29          7766554433    Tumkur
5           Ashok        34          1122334455    Mangalore
6           Akshay       32          2233223322    Raichur
7           syed         31          9988776688    Tumkur
8           Darshan     22          8233288899    Yadgir
9           suhas        25          9900889977    Mysore
23          santoshu     28          9900889966    pune
10          Hrutik       30          9900889988    BANGALORE
3           santosh      28          9900889966    pune
23          Ram          28          9900889966    Ayodhya
16          laxman       33          7777667777    Ayodhya
27          vishwanath   30          9944998800    Raichur
```

## JDBC Utilities:

### Program:

```
package com.sgtesting.jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.util.ArrayList;
import java.util.List;

class DBUtility
{
    /**
     * @param query
     * @param dburl
     * @param dbuser
     * @param dbpwd
     * @return rowCount
     */
    public static int getRecordsCount(String query,String dburl,String dbuser,String
    dbpwd)
    {
        int rowCount=0;
        Connection conn=null;
        try
        {
            conn=DBUtility.getDbConnection(dburl, dbuser, dbpwd);
            PreparedStatement stmt=conn.prepareStatement(query);
            ResultSet rs=stmt.executeQuery();
            while(rs.next())
            {
                rowCount=rowCount+1;
            }
        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

```

        finally
        {
            try
            {
                conn.close();
            } catch (Exception e)
            {
                e.printStackTrace();
            }
        }
        return rowCount;
    }
}
/**
 *
 * @param query
 * @param dburl
 * @param dbuser
 * @param dbpwd
 * @return columnCount
 */
public static int getTableColumnCount(String query,String dburl,String
dbuser,String dbpwd)
{
    int columnCount=0;
    Connection conn=null;
    try
    {
        conn=DBUtility.getDbConnection(dburl, dbuser, dbpwd);
        PreparedStatement stmt=conn.prepareStatement(query);
        ResultSet rs=stmt.executeQuery();
        ResultSetMetaData rsmeta=rs.getMetaData();
        columnCount=rsmeta.getColumnCount();
    } catch (Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        try
        {
            conn.close();
        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    return columnCount;
}

public static List<String> getFirstRecord(String query,String dburl,String
dbuser,String dbpwd)
{
    List<String> obj=null;
    Connection conn=null;
    try
    {
        obj=new ArrayList<String>();
        conn=DBUtility.getDbConnection(dburl, dbuser, dbpwd);
        PreparedStatement stmt=conn.prepareStatement(query);
        ResultSet rs=stmt.executeQuery();
        ResultSetMetaData rsmeta=rs.getMetaData();
        int cc=rsmeta.getColumnCount();
        String arr[]=new String[cc];
        for(int i=0;i<arr.length;i++)
        {

```

```

        arr[i]=rsmeta.getColumnName(i+1);
    }
    if(rs.next()==true)
    {
        for(int i=0;i<arr.length;i++)
        {
            String data=rs.getString(arr[i]);
            obj.add(data);
        }
    }

} catch (Exception e)
{
    e.printStackTrace();
}
finally
{
    try
    {
        conn.close();
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}
return obj;
}

public static boolean isInsertedRecord(String query,String dburl,String
dbuser,String dbpwd)
{
    boolean insertAction=false;
    Connection conn=null;
    try
    {
        conn=DBUtility.getDbConnection(dburl, dbuser, dbpwd);
        PreparedStatement stmt=conn.prepareStatement(query);
        int statusValue=stmt.executeUpdate();
        if(statusValue==1)
        {
            insertAction=true;
        }
    } catch (Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        try
        {
            conn.close();
        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    return insertAction;
}

public static Connection getDbConnection(String dburl,String username,String
pwd)
{
    Connection conn=null;
    try
    {
        Class.forName("oracle.jdbc.driver.OracleDriver");

```



```

        conn=DriverManager.getConnection(dburl, username, pwd);
    } catch (Exception e)
    {
        e.printStackTrace();
    }
    return conn;
}
}
public class UtilityDemo {
    public static void main(String[] args) {

        //          int rc=DBUtility.getRecordsCount("select * from employee",
        "jdbc:oracle:thin:@localhost:1521:XE", "system", "tiger");
        //          System.out.println("# of Rows in a table :"+rc);
        //          int cc=DBUtility.getTableColumnCount("select * from employee",
        "jdbc:oracle:thin:@localhost:1521:XE", "system", "tiger");
        //          System.out.println("# of Column in a table :"+cc);
        //
        //          List<String> record=DBUtility.getFirstRecord("select * from
        employee", "jdbc:oracle:thin:@localhost:1521:XE", "system", "tiger");
        //          System.out.println(record);
        //
        String query="insert into employee
        values(27,'vishwanath',30,9944998800,'Raichur')";
        boolean val=DBUtility.isInsertedRecord(query,
        "jdbc:oracle:thin:@localhost:1521:XE", "system", "tiger");
        System.out.println(val);
    }
}

```

### Output:

- 1) # of Rows in a table :18
- 2) # of Column in a table :5
- 3) [1, Santosh, 27, 7676977798, Gulbarga]
- 4) true

## Thread Topics

There is an interface Runnable this interface contains an abstract method run.

If any class implementing Runnable interface that class responsibility it has to override run method.

This run method acts as a entry point for the Thread.

### Program:

```

package com.sgtesting.thread;
class MyClass1 implements Runnable{

    @Override
    public void run() {
        System.out.println("It is an entry point for the Thread !!");
    }

}

public class ThreadDemo1 {

    public static void main(String[] args) {
        MyClass1 job = new MyClass1();
        Thread t1 = new Thread(job);
        t1.start();
    }
}

```

```

    }
}

```

Output :

It is an entry point for the Thread !!

If we can create object for a class which implements Runnable interface that object acts as a job.

If we can create object based on thread class that object acts as a worker.

The thread class constructor has over loaded which creating object if you class object which implements Runnable interface as a parameter in this case the worker has known about the job

## Multi-Threading Environment:

Here the multiple threads executing the same task parallely

**Program:**

```

package com.sgtesting.thread;
class MyClass2 implements Runnable{

    @Override
    public void run() {
        displayEvenNumber();
    }

    void displayEvenNumber() {
        try {
            for(int i=20; i<=50; i++) {
                if(i%2==0) {
                    System.out.println("display even number"+i);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

public class ThreadDemo2 {

    public static void main(String[] args) {
        MyClass2 job = new MyClass2();
        Thread t1 = new Thread(job);
        Thread t2 = new Thread(job);
        t1.setName("alpha");
        t2.setName("omaga");
        t1.start();
        t2.start();
    }
}

```

Output :

```

display even number20
display even number20
display even number22
display even number22
display even number24
display even number26

```

```

display even number24
display even number26
display even number28
display even number30
display even number28
display even number32
display even number30
display even number34
display even number36
display even number32
display even number38
display even number40
display even number42
display even number44
display even number46
display even number48
display even number34
display even number50
display even number36
display even number38
display even number40
display even number42
display even number44
display even number46
display even number48
display even number50

```

### Program on Thread sleep:

#### Program:

```
package com.sgtesting.thread;
```

```
class MyClass3 implements Runnable{
```

```
    @Override
```

```
    public void run() {
        displayEvenNumber();
    }
```

```
    void displayEvenNumber() {
```

```
        try {
```

```
            for(int i=20; i<=50; i++) {
                if(i%2==0) {
```

```
                System.out.println(Thread.currentThread().getName()+"display even number"+i);
```

```
                Thread.sleep(1000);
```

```
            }
```

```
        }
```

```
    } catch (Exception e) {
        e.printStackTrace();
    }
```

```
    }
```

```
}
```

```
public class ThreadDemo3 {
```

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

```
        MyClass3 job = new MyClass3();
```

```
        Thread t1 = new Thread(job);
```

```
        Thread t2 = new Thread(job);
```

```
        t1.setName("alpha");
```

```
        t2.setName("omaga");
```

```

        t1.start();
        t2.start();

    }
}

```

#### Output:

```

omagadisplay even number20
alphadisplay even number20
alphadisplay even number22
omagadisplay even number22
omagadisplay even number24
alphadisplay even number24
alphadisplay even number26
omagadisplay even number26
omagadisplay even number28
alphadisplay even number28
omagadisplay even number30
alphadisplay even number30
omagadisplay even number32
alphadisplay even number32
alphadisplay even number34
omagadisplay even number34
alphadisplay even number36
omagadisplay even number36
omagadisplay even number38
alphadisplay even number38
omagadisplay even number40
alphadisplay even number40
omagadisplay even number42
alphadisplay even number42
alphadisplay even number44
omagadisplay even number44
omagadisplay even number46
alphadisplay even number46
omagadisplay even number48
alphadisplay even number48
alphadisplay even number50
omagadisplay even number50

```

- **How to make thread as a synchronized?**

The synchronized provides thread safe concept

#### Program:

```
package com.sgtesting.thread;
```

```
class MyClass4 implements Runnable {
```

```
    @Override
```

```
    public void run() {
        displayEvenNumber();
    }

```

```
    synchronized void displayEvenNumber() {
```

```
        try {
            for (int i = 20; i <= 50; i++) {
                if (i % 2 == 0) {
                    Thread.sleep(1000);

```

```
                System.out.println(Thread.currentThread().getName() + "displayEvenNumber" + i);
            }

```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

    }
}

public class ThreadDemo4 {

    public static void main(String[] args) {
        MyClass4 job = new MyClass4();
        Thread t1 = new Thread(job);
        Thread t2 = new Thread(job);
        t1.setName("Alpha");
        t2.setName("gamma");
        t1.start();
        t2.start();

    }
}

```

### Output:

```

gammadisplayEvenNumber20
gammadisplayEvenNumber22
gammadisplayEvenNumber24
gammadisplayEvenNumber26
gammadisplayEvenNumber28
gammadisplayEvenNumber30
gammadisplayEvenNumber32
gammadisplayEvenNumber34
gammadisplayEvenNumber36
gammadisplayEvenNumber38
gammadisplayEvenNumber40
gammadisplayEvenNumber42
gammadisplayEvenNumber44
gammadisplayEvenNumber46
gammadisplayEvenNumber48
gammadisplayEvenNumber50
AlphadisplayEvenNumber20
AlphadisplayEvenNumber22
AlphadisplayEvenNumber24
AlphadisplayEvenNumber26
AlphadisplayEvenNumber28
AlphadisplayEvenNumber30
AlphadisplayEvenNumber32
AlphadisplayEvenNumber34
AlphadisplayEvenNumber36
AlphadisplayEvenNumber38
AlphadisplayEvenNumber40
AlphadisplayEvenNumber42
AlphadisplayEvenNumber44
AlphadisplayEvenNumber46
AlphadisplayEvenNumber48
AlphadisplayEvenNumber50

```

## Regular Expression:

Regular expression provides pattern searching and pattern matching concept.

Regular expression in core-java can be achieved based on the following 2 classes

1. Pattern
2. Matcher

The above two classes are available in the java.util package.

### Program:

```
package com.sgtesting.regularexpression;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegularExpressionDemo {

    public static void main(String[] args) {
        matchesDemo();
        findDemo();
        displayMatchingWords();
        countOfMatchingWords();
        displayMatchingPosition();
        replaceMatchDemo();
        splitDemo();
    }

    private static void matchesDemo() {
        Pattern pattern = Pattern.compile("java");
        Matcher match = pattern.matcher("java");
        boolean val = match.matches();
        System.out.println(val);
    }

    private static void findDemo() {
        Pattern pattern = Pattern.compile("java");
        Matcher match = pattern.matcher("java jre java jvm java");
        boolean val = match.find();
        System.out.println(val);
    }

    private static void displayMatchingWords() {
        Pattern pattern = Pattern.compile("java");
        Matcher match = pattern.matcher("java jre java jvm java");
        while(match.find()) {
            System.out.println(match.group());
        }
    }

    private static void countOfMatchingWords() {
        Pattern pattern = Pattern.compile("java");
        Matcher match = pattern.matcher("java jre java jvm java");
        int count=0;
        while(match.find()) {
            count++;
        }
        System.out.println("# of occurrence : "+count);
    }

    private static void displayMatchingPosition() {
        Pattern pattern = Pattern.compile("java");
        Matcher match = pattern.matcher("java jre java jvm java");
        while(match.find()) {
            System.out.println(match.group()+"Start position : "+match.start()+"End position : "+match.end());
        }
    }

    private static void replaceMatchDemo() {
```

```

        Pattern pattern = Pattern.compile("java");
        Matcher match = pattern.matcher("java jre java jvm java");
        String str = match.replaceAll("python");
        System.out.println(str);
    }
    private static void splitDemo() {
        Pattern pattern = Pattern.compile("[!@#$$%^]");
        String str[]=pattern.split("orange!mango@grapes#Banana$Apple%kiwi
fruit^cherry");
        for(String kk : str) {
            System.out.println(kk);
        }
    }
}

```

#### Output:

- o True
- o True
- o java
- o java
- o java
- o # of occurrence : 3
- o javaStart position :0End position :4
- o javaStart position :9End position :13
- o javaStart position :18End position :22
- o python jre python jvm python
- o orange
- o mango
- o grapes
- o Banana
- o Apple
- o kiwi fruit
- o cherry

| Sl.no | Regular Expression Character | Description                                                                                       |
|-------|------------------------------|---------------------------------------------------------------------------------------------------|
| 1     | ^                            | It matches beginning of the String                                                                |
| 2     | \$                           | It matches End of the String                                                                      |
| 3     | *                            | It matches preceding character zero or more time                                                  |
| 4     | ?                            | It matches preceding character zero or one time                                                   |
| 5     | .                            | It matches only one character at a time                                                           |
| 6     | +                            | It matches preceding character one or more time                                                   |
| 7     | {n}                          | It matches nth occurrence                                                                         |
| 8     | x/y                          | It matches either x or y                                                                          |
| 9     | [wxyz]                       | It matches any one character in the list                                                          |
| 10    | [^wxyz]                      | It matches any one character other than character specified in the list                           |
| 11    | [a-z]                        | It matches any one lower case character in between a to z                                         |
| 12    | [^a-z]                       | It matches any one character other than the lower-case character in between a to z                |
| 13    | [A-Z]                        | It matches any one upper case character in between A to Z                                         |
| 14    | [^A-Z]                       | It matches any one character other than the upper-case character in between A to Z                |
| 15    | [a-z A-Z]                    | It matches any one character either lower-case or upper-case character in between a to z / A to Z |
| 16    | [1234]                       | It matches any one enclosed with the list                                                         |
| 17    | [^1234]                      | It matches any one digit other than the specified digit in the list                               |
| 18    | [0-9]                        | It matches any one digit in-between zero to nine (0 to 9)                                         |
| 19    | [^0-9]                       | It matches any one-digit character other than the digit in-between 0 to 9                         |

## Program:

```
package com.sgtesting.regularexpression;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegularExpressionDemo2 {

    public static void main(String[] args) {
        regExpExample1();
        regExpExample2();
    }

    private static void regExpExample1() {
        String str = "Bangalore received 125 milimeter of rain yesterday";
        Pattern pattern = Pattern.compile("[0-9]{3}");
        Matcher match = pattern.matcher(str);
        String strResult = match.replaceAll("many");
        System.out.println(strResult);
    }

    private static void regExpExample2() {
        String str = "The temple is at the top of the hill";
        Pattern pattern = Pattern.compile("[a-zA-z]");
        Matcher match = pattern.matcher(str);
        while (match.find()) {
            System.out.println(match.group());
        }
    }
}
```

## Output:

Bangalore received many milimeter of rain yesterday

T  
h  
e  
t  
e  
m  
p  
l  
e  
i  
s  
a  
t  
t  
h  
e  
t  
o  
p  
o  
f  
t  
h  
e  
h  
i  
l  
l



