SYLLABUS :-

- INTODUCTION OF JAVA

MODULE-1

1.BASIC STRUCTURE OF PROGRAM
2.ARCHITECTURE OF JAVA
3.DATA TYPES
4.VARIABLES
5.KEYWORDS AND IDENTIFIERS
6.OPERATORS AND TYPES
7.CONDITIONAL STATEMENTS
8.SWITCH CASE STATEMENTS
9.LOOPING STATEMENT
10.PATTERN PROGRAMMING
GENERAL PROGRAMS BASED ON FOR AND WHILE LOOPS

MODULE-2

11.METHODS AND METHOD OVERLOADING
12.VARIABLES AND ITS TYPES
13.ACCESSING DATA MEMBER IN MEMBER FUNCTION
14.EXECUTION PROCESS
15.CONSTRUCTORS
16.CONSTRUCTOR OVERLOADING
17.THIS KEYWORD
18.CONSTRUCTOR CHAINING
19.UML DIAGRAMS

MODULE-3

-OOPS

20.INHERITANCE
21.METHOD OVERRIDING AND SUPER KEYWORD
22.ABSTRACTION
23.INTERFACE
24.ACCESS SPECIFIER
25.ENCAPSULATION
26.POLYMORPHISM AND METHOD BINDING
27.GENERALISATION AND SPECIALISATION
28.TYPE CASTING
29.OBJECT CASTING

MODULE-4

30.OBJECT CLASS
31.STRING CLASS
32.ARRAY PROGRAMMING
33.EXCEPTION HANDLING
34.WRAPPER CLASSES
35.COLLECTION FRAMEWORK

36.MAPS
37.MULTUTHREADING

INTRODUCTION

AUTHOR ---------- JAMES GOSLING
VENDOR ---------- SUN MICRO SYSYTEM(ORACLE)
PROJECT NAME ---------- GREEN TEAM
TYPE ---------- OPEN SOURCE
INITIAL NAME ---------- OAK
PRESENT NAME ---------- JAVA
EXT ---------- .java,.class,.jar
PRESENT VERSION ---------- JAVA14
OPERATING SYSTEM ---------- ANY OPERATING SYSTEM
BASIS ---------- C++
PRINCIPLE ---------- WORA(write once run anywhere)

USES :-

-Web applications
-Mobile applications
-Client Server application
-Embeded Systems
-Robotics
-SAP

PARTS OF JAVA

---

1. J2SE/JSE(JAVA 2 STD EDITION)
2.J2EE/JEE(JAVA 2 ENTERPRISE EDITION)
3.J2ME/JME(JAVA 2 MICRO EDITION)

Version History of JAVA :

JDK Alpha and Beta (1995)
JDk 1.0 (23rd Jan 1996)
JDK 1.1 (19th Feb 1997)
J2SE 1.2 (8th Dec 1998)
J2SE 1.3 (8th May 2000)
J2SE 1.4 (6th Feb 2002)
J2SE 5.0 (30th Sep 2004)
Java SE 6 (11th Dec 2006)
Java SE 7 (28th July 2011)
Java SE 8 (18th Mar 2014)
Java SE 9 (21st Sep 2017)
Java SE 10 (20th Mar 2018)
Java SE 11 (11 September, 25th 2018)
Java SE 12 (15 March, 19th 2019)
Java SE 13 (15 September, 17th 2019)
Java SE 14 (14 March, 17th 2020)
Java SE 15 (15 Expected in September 2020)

FEATURES OF JAVA :-

A list of most important features of java.

Simple :

-Simple to learn(Syntax's)

Object-Oriented :

Java is an object-oriented programming language. Everything in java is an object.
Object-oriented means we organize our software as a combination of different types of objects that incorporates.....

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenace by providing some rules.

Basic concepts of OOPs are :

Object
Class
Inheritance
Polymorphism
Abstraction
Encapuslation

Portable :

-Can be used with any other language

Platform independent :

-Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc.
-Java code is compiled by the compiler and converted into byte code.
-This byte code is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere(WORA).

Secured :

-any problem happens only JVM(JAVA VIRTUAL MACHINE) will get effected but operating system is safe.

Robust :

-Most of things are automated
  Ex: garbage collection

Multithreaded :

-Running multiple processeses(tasks) at same time.

MODULE-1

BASIC STRUCTURE OF PROGRAM

```
public class Demo.....(1)
{
   public static void main(String args[ ] ).....(2)
```

```
    {
        System.out.println("This is my first program");.....(3)
    }
}
```

Every programme has 3 main parts

1.CLASS DECLARATION

Ex: public class Demo

It consists of 3 things

---

1. Acess Modifier :
- It indicates that program is accessible to other users or not
- There are 4 access modifiers in java public,private,protected and default
- in above section class is public so it is freely accessible
- All access modifiers will be in lower case.

2. class :
- class is a keyword(reserrve word or predefine word) in java.
- Every program must start with class keyword
- all keyword must starts with smaller case so class-c is small.

3. class Name :
- Every class has some name i.e class name or program name or file name
- class name for standard should start with Capital letter
- Java File name and class name must be same for remembering purpose
- class name can only be combination of A-Z,a-z,0-9,$,_

Note:{//}--------->scope of class

2.DEFINING MAIN METHOD

Ex:
```
public static void main(String args[ ] )
    {
        / / LOGIC OF APP
    }
```

It consists of 5 parts

1.Access Modifier :

2.Non Access Modifier :

static----------> can be used without object creation

non static---------> needs to be used with object creation

3.return type :

void indicated no specific return type

4.method name :

-if anyword contains( )----> we can identified it as a method

Ex: main( ),run( ),display( ) etc

-main is name of method

5.command line arguments :

String args[ ]
String - S is capital
-This statement can be written in 3 ways
1.String args[ ]
2.String [ ]args
3.String[ ] args
Note: In syntax of main method only String-S is capital remaining all words starting letters are small.

3.PRINTING STATEMENT

System.out.println("This is my sample program");
system------> it is a pre define class[class System]
.----------> it is a dot operator,any word after it can be reference variable/object or method
out----------> it is an object (predefine)
println( )----> it is a method (predefine)

-In simple println( ) is accessed through out object but out object is present in system class.
-System is a class which contains out object and out object is referring to println( )
-whatever we gave in double quotes that message will be printed as it is.

For Mobile Execution

---

please install this app from play store:
AIDE-IDE FOR C++ AND JAVA(ANDROID) or DECODER(ANDROID)
JEDONA(IOS)

Parts of java :

1.J2SE/JSE (java 2 std edition) - Basic/core java
2.J2EE/JEE (java 2 enterprise edition) - Full stack development
3.J2ME/JME (java 2micro edition)

Void - no specific return type

Client server application - email
Web based application - any application with url

16-07-2020

---

we can change position of public and static vice-versa.

Variations in printing message :

---

println()----->print next message in new line
print()------>print next message in same line

example-1

```
public class Today
{
  public static void main(String args [ ] )
  {
     System.out.println("Hi Rohan");
     System.out.print("Your order with id:1235 ");
     System.out.println("is on the way.");
     System.out.print("Please collect it");
  }
}
```

Output

Hi Rohan

Your order with id:1235is on the way.
Please collect it

example-2

```
public class AboutMe
{
  public static void main(String args[ ])
  {
     System.out.printl("My Name is : Paul ");
     System.out.println("I am 22 years old");
     System.out.print("I have graduated from Osmania University ");
     System.out.println("My Ambition is to be a programmer");
  }
}
```

Output

My Name is : Paul I am 22 years old
I have graduated from Osmania University My Ambition is to be a programmer

Commenting any line :

----> if we put // in starting of statement it will be commented (not considred as part of program)
  Ex: //public class AboutMe

----> if we put // in ending of statement after that whatever we write it will not be considred as part of program.
  Ex: public class AboutMe//class declaration

```
class Today//class declaration
{
   public static void main(String ags[ ])//main method
   {
     //System.out.print("My Name is : Paul ");
     //System.out.println("I am 22 years old");
     System.out.print("I have graduated from Osmania University ");
     System.out.println("My Ambition is to be a programmer");
```

```
  }
}
```
Output

I have graduated from Osmania University My Ambition is to be a programmer

\n :

```
public class MyInfo
     {
          public static void man(String args[ ])
          {
            System.out.println("Name : Rohan \n  DOB : 25-07-1993 \n Age:27 ");
           }
}
```
Output

Name : Rohan
DOB : 25-07-1993
Age:27

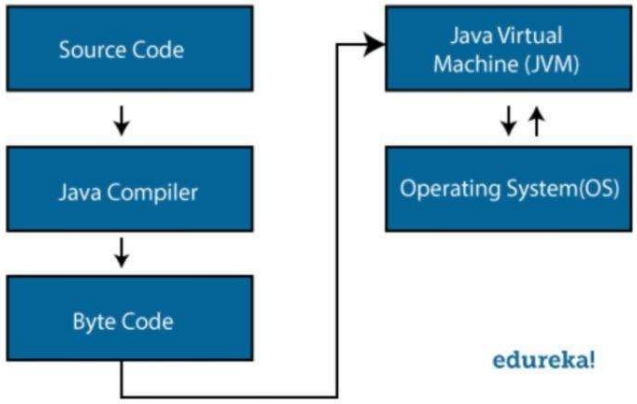
Notes: \n is used to break the line.
\n should must be in double quotes.

17-07-2020

STEPS TO DESIGN AN APPLICATION OF JAVA

STEP-1: Select n Editor (Notepad,Notepad++,Editplus or IDE-Integrated development tool)
STEP-2: Write the logic of APP
STEP-3: Save the APP
STEP-4: Compilation
STEP-5: Execution
Note: In IDE's 75% of work is automatic
Ex: Eclipse, Net Beans, etc

Architecture of Java



Step-1:

-whatever program we write it is called as source code.
-source code should always save as ext ".java"
-above program should save as Hello.java

Step-2: Compilation

-The process of converting our program into system understandable form (byte code) is purpose of compiling a program.
-to compile a program go to cmd prompt and enter command as
-javac programname.java
-Ex: javac Hello.java
-Compilation is done at once.
-During compilation, compiler will check syntax errors like [ ], ;,(),{ },:,spellings and case sensitivity.
-If anything is wrong we will get compile time error.
-If nothing is wrong there is one class file get generated (byte code file) with same as .class

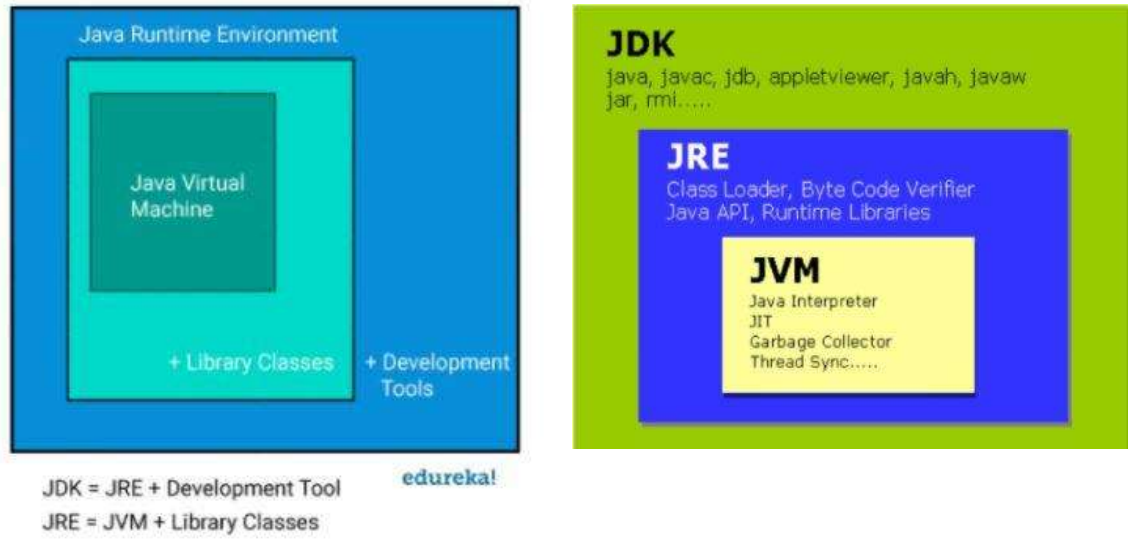
Step-3: Execution

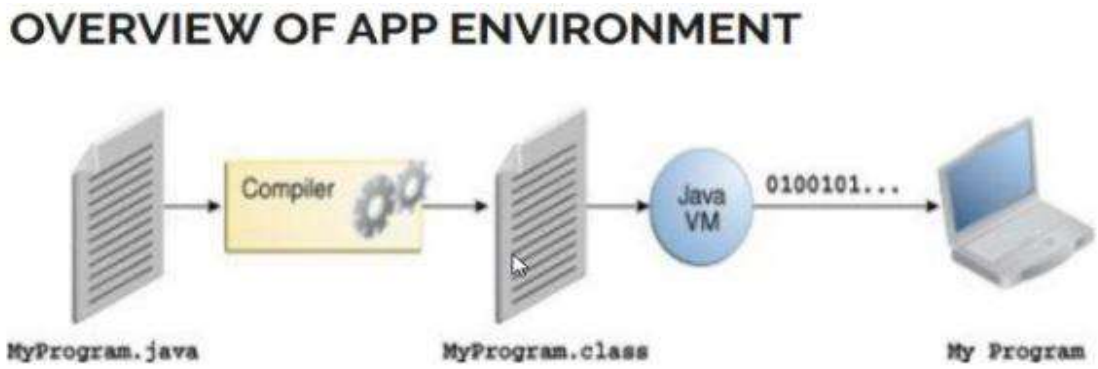
-JVM -java virtual machine is responsible for execution of every java program
-JVM's can identify by ------> Public static void main (String [ ] args)
-it is like one software or one program.

-execution will happen in line by line manner
-during execution JVM will find logical error of program
-for executing program go to command prompt and enter command as
-java programname
-Ex: java Hello
-Once we enter this command JVM will go to class file and take first line and give to operating system for execution, once OS responds that i understood that line and it sends second line and it continues till last line like this whole code of class file gets executed.

JDK-JAVA DEVELOPMENT KIT

- IF WE WANT TO DEVELOP AND EXECUTE JAVA PROGRAM IN OUR SYSYTEM WE HAVE TO INSATLL JDK(JRE,JVM,SUPPORTING TOOLS AND SUPPORTING CLASSES).





Components of JDK

-JRE - java runtime environment
-JVM - java virtual machine

JRE: JAVA RUNTIME ENVIRONMENT

-IT IS USED FOR EXECUTING JAVA PROGRAM OR RUNNING JAVA APP

JVM: JAVA VIRTUAL MACHINE

-IT IS RESPONSIBLE TO EXECUTE EVERY JAVA PROGRAM.

Note: when we install JDK with that JVM and JRE are available.

18-07-2020

- JIT : Just In Time compiler
- It helps JVM by taking code of class file line by line and give to OS.

DATA TYPES AND VARIABLES

Data types :

1.Data
2.Data Types
3.Variables

Data : Any information is called as data.
For Ex: name,age,height,marks,percentage and salary etc

Data Type : it defines type of data.

Divided into 2 types

1.Primitive(System define)
2.Non primitive(user define)

Primitive data type :

-These are system define data types
-These are fixed in there memory size
-These are 8 in numbers

Name	Size	Examples	Default values
1.byte	1 byte	10,2,5(127 is max -128 is minimum)	0
2.boolean	1 byte or no size	true or false	false
3.short	2 byte	100,220. . . .(32,767 to -32,768)	0
4.char	2 byte	A,a.	empty space
5.int	4 byte	1,2,777. . . . . .(2,147,483,647 is max)	0
6.float	4 byte	0.2,0.3,33.666.	0.0
7.long	8 byte	33333333,6565655.	0
8.double	8 byte	0.343434343,99.5555555.	0.0

Note:

-When we want 4-6 digits of accuracy we go for float else we use double
-byte,short,int,long is use for Numericals or integers
-double and float use for decimals
-when we store value greater than 2,147,438,647 we use long and we need to repesent with 'l'

Non primitive data type :

-These are not fixed in there memoy size.

1.String : it is used to represents group of char's
ex: java,manual testing. . . . . .

2.Arrays :
(10,20. . . .100)

Note:

-char can be used for single characters
-If we have more than one character we should use String
-Using String we can represent any type of data(info)

Variables :

-Variables are used to store the data for printing or using it in future.
1.Variable Declaration

Syntax : Datatype variablename;
      Ex: int a;
           float b;
           char ch;
           String s;
-variable name can be a combination of a-z,A-Z,0-9,$ and _
-variable name should not start with number
-whenever we declare a variable one memory block will get created

2.Variable Initialisation

Syntax: Variablename=value;
             a=100;
             b=0.3333f;//mandatory to write f
            ch='A';//mandatory to give ' '
             s="java";//mandatory to give " "
-we can declare and intialise a variable in single statement also
syntax: Datatype variablename=value;
           int a=22;
           float b=0.2345f;

Examples

1.int a=22,b=33;//valid
2.int a=33,b;//valid
3.float percentage=60.0;//invalid-----> f is not present
  float b=0.334;//defaultly it will be considered as doble to indicae that it is float we have to give f
4.char ch='AB';/invalid char can't be more than one character
5.float h=100f;//valid---->100.0
6.int i=0.334;//invalid---->integer can't store dcimal values
7.String s="123";//valid----->System.out.println("123"); 8.String d="3334+ghijk";//valid
9.double marks=100.3434d;//valid----->in double d is optional
10.long number=93939393939l;//valid----->in long l we need to keep if we write more than 32 bit

---
20-07-2020

- void is a return type which does not specify any particular return type
- execution will always start from main method
- componenets of JDK are JRE,JVM in additional JIT

How to print data stored in variable

Ex: int age=45;
    System.out.println(age);//45
    char grade='A';
    System.out.println(grade);//A
int---------->Data type
age---------->variable name
=---------->Operator
45---------->data/value
;---------->Termination of statement
- we write f for float because otherwise it will consider as double

Interview question

1Q) Can we change the syntax of main method?
A)
  -JVM is responsible to execute every java program.
  -JVM // the identity of jvm is public static...., ifwe change that then jvm will tell identity is not found
        {
          public static void main(command line args)
        }
  -Execution of every java program will always start from main method
  -Because JVM only knows main method
  -If main method is not there or main method syntax is changed our program will not be executed.
  -So, the answer is No we cannot change syntax of main method.

2Q) Can we keep main method as private?
A) No we cannot keep main method as private because private fields cannot be accessed from outside of class.
  -And JVM needs to access main method from outside of class.
  -If we keep main method as private, program will compile but it will not be executed.
  -We will get Error as Main Method not found in given class

3Q) Can we keep main method as non static ?
A) No we can't write main method as non static because if method is non static, we have to create an object and JVM can't create object on its own to access main method.
  -If we keep main method as non static program will compile but it will not be executed.

Note:

-Actually there is no Accessmodifier called as default, if we did not mention public,private or protected it wil be considered as default ---->which means there is no need to write like " default class Mydetails " instead we write " class Mydetails " itself considerd as there will be default accessmodifier .
-Actually there is no non acessmodifier called as non static, if we did not mention static modifier, JVM will consder it as non static.
 " public class void main (String args [ ]) " ----------> non static

---
21-07-2020

--> cd.. for coming out of any folder
--> cd give space and folder name to get in to that folder, Ex: cd ravi
--> driver: to get into any particular driver, Ex: e:
--> cls to clear the screen
--> to create a folder give command mkdir space folder name, Ex: mkdir ravi

OPERATORS :-

1.Arithemetic Operators :

+ ====> Addition
- ====> Subtraction
\ ====> Division (output:Coefficient)
* ====> Multiplication
% ====> Mode (output:Remainder)

=>Create an Application which calculates
1.Addition of two numbers
2.Subtraction of two numbers
3.Multiplication of two numbers
4.Division of two numbers
5.Mode of two numbers

```
class Problems
{
  public static void main (String args[ ])
{
   int a=10,b=5,c,d,e,f,g;
    c=a+b;
    d=a-b;
    e=a*b;
    f=a%b;
    g=a/b;
   System.out.println(c);
   System.out.println(d);
   System.out.println(e);
   System.out.println(f);
   System.out.println(g);
}
}
```

2.Operator Overloading :

- Overloading is one thing but plays multiple roles.
Ex: A person can be son,father,husband,grandfather,friend,boyfriend etc
-Java doesnot support operator overloading but there is one exception as like + operator, + is an overloaded operator because it act as addition as well as concatination.

Addition
---

1. int + int(int/float/double/long/char/short/byte)
   Ex1: 122+233,
   Ex2: 445.33+556,
   Ex3: 667+45.3f
2. char + char(int/float/double/long/char/short/byte)
   / / Java provides unicodes for every character
      A-65,B-66,C-67. . . . . .Z-90
      a-97,b-98,c-99. . . . . .z-122
   Ex: 'A' + 65----->65+65------>130
      'a'+'z'------->97+90------>187

   Ex: char ch=65;
      System.out.pintln(ch); / / A

Concatination

- If any one operand is String plus operator will always act as concatination.

```
Ex: String s="java";
     int a=123;
    System.out.prntln(s+a); / / "java"+123--->"java123"
    System.out.prntln("I am "+s+" developer"); / / "I am java"+"developer"--->I am java developer
```

- After concatination result will always be string

```
    System.out.prntln(123+" "); / /"123 "

Ex: class BankInfo
    {
      public static void main(String args[ ])
       {
          String accholdername="Rohan";
          double currbalance=450000.0;
          int minbal=3000;
          System.out.println("The account holder name is: " + accholdername);
          System.out.println("Current balance is:"+currbalance);
          System.out.println("Minimum balance needsto maintain is :"+minbal);
      }
    }
```

Assignment

=> Create ann APP whch prints below information about book using datatypes and variables
   bookname is : java
   bookprice is : 550
   bookpages is : 800
   bookrating is : A
=> Create an Application which print the details of smartphone
   Device name: Iphone
   Device Price: 132000
   ram : 64gb
   Screen Size: 6.0'
=> Create an Application which print the details of Employee
   Company Name: Infosys
   Designaion: QA
   Salary: 3.5LPA
   Experience: 2yrs

23-07-2020

- 22,july 2020 class is suspended due to technical problems

3.Assignment operator(=) :

- it is use to assign or store the value into a variable.
  Ex: int a = 10,b=30;
      int c = a+b;

4.Comparison operator(==) :

- it compares values and give output as boolean value.

5.Relational Operators :

- checks relationship between two values and result will be boolean
  > ----------> greater
  < ----------> smaller
  >= ----------> greater than equals
  <= ----------> lesser than equals
  != ----------> not equals

Result will be either true or false

6.Logical Operrators :

AND :

- it compares two inputs and if both inputs are true then output is true or else false.
- it is represented as && (in below ex 0 indicate false, 1 indicate true)

a	b	a&&b
0	0	0
0	1	0
1	0	0
1	1	1

OR :

- it compares two inputs and if any one input is true then output is true or else false.
- it is represented as || (in below ex 0 indicate false, 1 indicate true)

a	b	a||b
0	0	0
0	1	1
1	0	1
1	1	1

NOT (!) :
---

input output
0 1
1 0

Keywords :-

- These are the reserve words or predefine words which have some reserve meaning.
- Various keywords in java are,

1.accessible keywords

---

public, private, protected, static, final, abstract and return. // System(not keyword) is predefined class, it should start with capital letter.

2.conditional keywords

---

if, else, else if, switch, case, break, continue, goto, const and default.

3.iterative keywords

---

for, while, do while

4.class level

---

class, package, import, extends, implements, interface, new

5.Exception level

---

try, catch, throw, throws, finally

6.Others

---

volatile, transient, synchronised, native etc

Note :

---

1. In java there is no word called as default but meaning is there.
   Ex: class A ---------->here access modifier is deafult
2. There is no word as non static but meaning is there.
   Ex: public void run( ) ------>here non access modifier is non static
3. All keywords must starts with smaller case

Identifiers :-

- These are the names given by programmer as per convention.
  Ex: class name, variable name, method name and package name.
  rule ----------> should be followed
  convention ----> if we don't follow also no problem

Rules for defining identifiers :

1.An identifier can be a combination of A-Z,a-z,0-9,$ and _ but standard is,
class name : starts with capital(convention)
variable name: starts with small(convention)
method name: starts with small(convention)
package name: starts with small(convention)

2. *If an identifier contains more than one word spaces are not allowed.
   class My program----->Invalid

int my age---------->Invalid
public static void display details( )----->Invalid
class Myprogram----->valid
int mypercentage------>valid

3.*An identifier cannot starts with digit.
class 1A------->Invalid
int 10a---------->Invalid
class A1--------->valid
int a10-------->valid

4.class name contains more than one word for all words first letter should be capital
Ex: class MyFirstProgram(Convention)

5.If a method name and variable name contains more than one word from second word firstletter should be capital(Convention).
Ex: int myAge;
Ex: public static void displayDetails( )

6.*A variable name and class name cannot be keyword.
Ex: class new----->Invalid
int static------->Invalid

24-07-2020

7.Unary Operators :
---
a) Increment operator :

- It increases the value by one.
 for ex: a=10; increment a add +1 to a
- It is denoted as ++
- It is of two types
    1.pre-increment
    2.post-increment

b) Decrement operator :

- It decreases the value by one.
 for ex: a=10; decrement a subtract -1 to a
- It is denoted as --
- It is of two types
    1.pre-decrement
    2.post-decrement

Types :
---
preIncrement :

- The rule is first increment value then print or assign it or store it in variable.
- it is denoted as ++variablename
    For Ex: int a=10; //11
            int b=++a://pre increment---->10+1
            SOP(b);//11
        Ex2: int a=250;
            int b=++a://a is incresed by 1 and then 251 is stored in b
        Ex3: int a=50://51
            ++a: //50+1
            SOP(a);//51

postIncrement

- the rule is first print or assign it or store it in variable then increment value.
- it is denoted as variablename ++
  For ex: int a=10;//11
          int b=a++;//post increment---->1.b=10,2.10+1
          SOP(b);//10

preDecrement :

- the rule is first decrement value then print or assign it or store it in variable.
- it is denoted as --variablename
  For ex1: int a=10;
           int b=--a;//pre decrement
      ex2: int a=250:

```
        int b=--a;//a is decreased by 1 and then 249 is stored in b
    ex3: int a=50;//49
         --a; //50-1
        SOP(a);//49
```

postDecrement :

- the rule is first print or assign it or store it in variable then decrement value.
- it is denoted as variablename--

For ex:
```
int a=10;
int b=a--;//post decrement
SOP(b);//10
SOP(a);//9
```

Exercise:

1.
```
int a =100;
System.out.println(++a   +   a++);
                    |        |
                   101   +   101---->202
```

2.
```
int a=100
System.out.println(++a   +   a++    +   ++a);
                    |        |           |
                   101   +   101     +   103---->305
```

3.
```
int a=50;
System.out.println(a++   +   ++a    +   a--     +    --a)
                    |        |          |          |
                   50    +    52    +   52    +    50------>204
```

```
int a=100,b;
```

Expression	intial_value of a	final_value of b	final_value of a
b=++a;	100	101	101
b=a++;	100	100	101
b=--a;	100	99	99
b=a--;	100	100	99

7.Combinational or Compound Operators :
---
1. += (compound addition assignment operator)

For Ex: a=a+b; can also be written as
```
a+=b;//by using combinational or compound operator
```
Ex: c=a+b//we cannot apply combinational operator

2. -= (compound subtraction assignment operator)
3. *= (compound multiplication assignment operator)
4. /= (compound division assignment operator)
5. %= (compound modulo assignment operator)

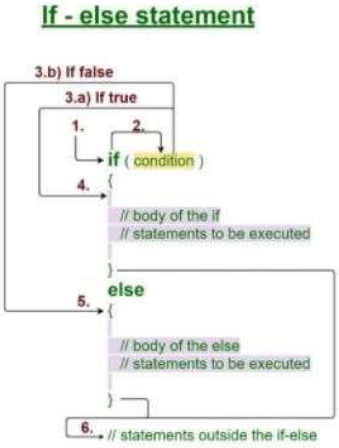
CONDITIONAL STATEMENTS

TYPE 1

- Depending on conditions it switches control flow of execution from one

Syntax:

```
 if(condition)
{
//Set of statements//
}
else
{
//set of statements
}
```



Note :

- writing else is not mandatory,we can add it as per our requirement.
- condition type must be boolean
- if we have only one statement inside if { } then we can skip/omit flower brackets
- when if is true else will not be executed.

1.WAP to check whether 10 and 20 are equal or not
2.WAP to check whether 10 is positive or negative
3.WAP to check whether pooja is eligible to cast a vote or not, pooja's age 21
4.WAP to check whether 22 is divisible in 4 or not
5.WAP to check whether 32 is even number or not

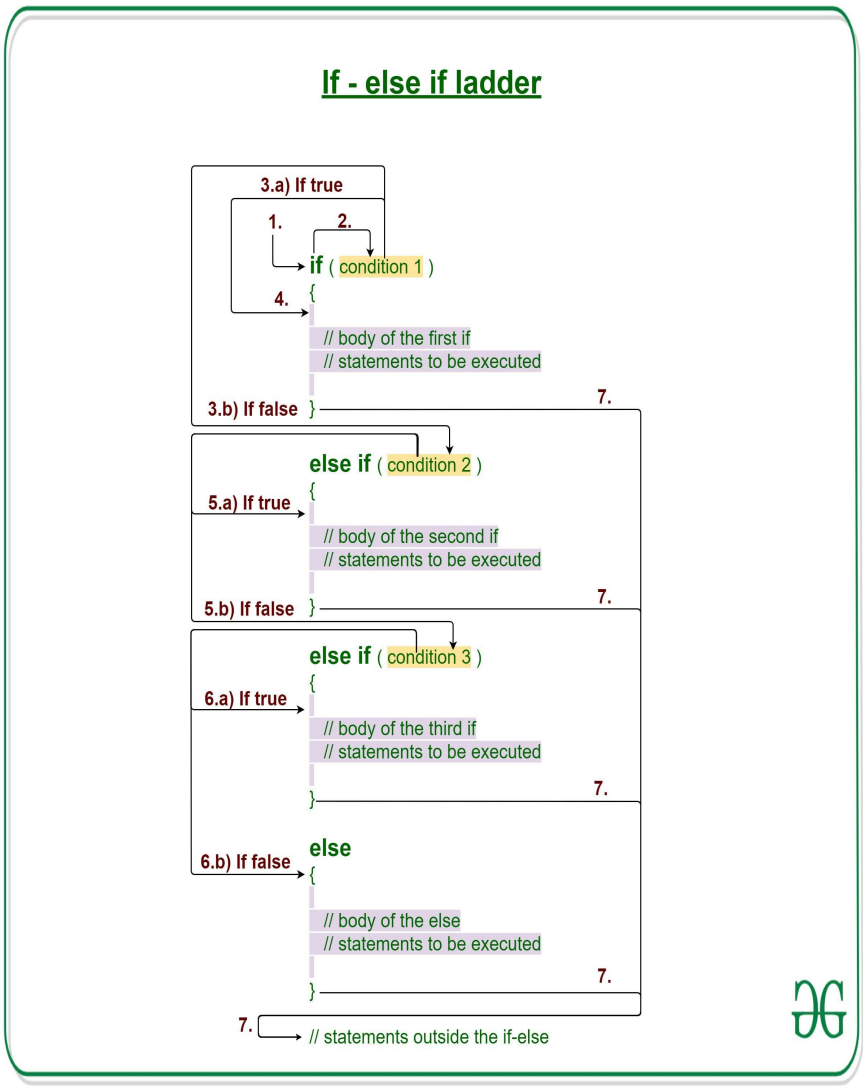
25-07-2020

Type-2

```
if(condition1)
{
 // if body
}
else if(condition2)
{
  //else if body
}
else if(condition3)
{
 //elseif body
}
else
{
  //else body
}
```

Type-3

```
if (condition1 LogicalOperator condition2)
{
  // body of if
}
else
{
  // body of else
}
```



- WAP to find greatest of 3 numbers a=10,b=27,c=6.
- WAP to check whether couples are eligible for mariage or not.
- WAP to check whether 10 is divisible in both 2 and 4.
- WAP to check whether 8 is divisible in either 3 or 4.

*Eq - Notepad

File Edit Format View Help

```
class Eq
{
  public static void main(String args[])
  {
    int num=8;
    if(num%2==0)
       System.out.println(num +"is divisible in 2 tables");
       System.out.println(num +"is also even number");
    else

      System.out.println(num+"is not divisible in 2");

 }}

Conclusion:we cannot omit flower braces for conditional stmt
           if there is more than one statement under if
```

Command Prompt

```
C:\Docs\programs\Kccm5>javac Eq.ja
va
Eq.java:9: error: 'else' without '
if'
    else
    ^
1 error

C:\Docs\programs\Kccm5>
```

Differences between C and JAVA : -

C	JAVA
C is a Procedural Programming Language.	Java is an Object-Oriented Language.
C was developed by Dennis M.Ritchie in 1972.	Java language was developed by James Gosling in 1995.
In the C declaration variable are declared at the beginning of the block.	In Java, you can declare a variable anywhere.
C does not support multithreading.	Java has a feature of threading.
C support pointers.	Java does not support pointers.
Memory allocation can be done by malloc.	Memory allocation can be done by new keyword.
Garbage collector needs to manage manually.	Garbage collector is managed automatically.
C does not have a feature of overloading functionality.	Java supports method overloading.
C is platform dependent language.	Java is platform independent language.
C is library Oriented language. #include<Stdio.h>	Java is package and class based language. package java.util;
Operating system is responsible to call main( ).	JVM is responsible to call main( ).

27-07-2020

Looping statements

- Loop is defined as repeated execution.
- If a part of code is repeatedly executing in our program rather than writing it multiple times, we can define it only once inside loop and run it as many times as we want.
For example- print java 10 times or print 1 to 10 etc
- They are basically of 4 types
1. for
2. while
3. do while
4. for each(Enhanced or Advance for loop)

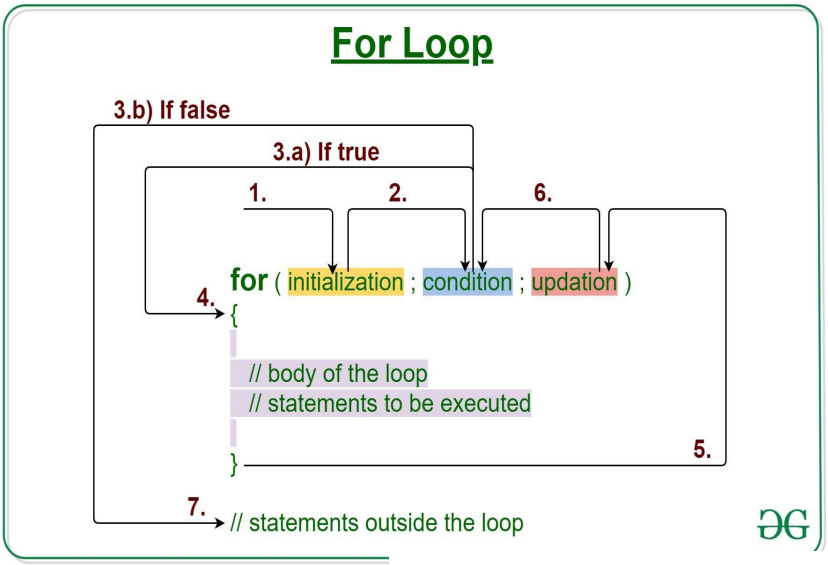
Without looping statements

1.WAP to print java 10 times.

```
                //Without using loop//
class A
{
public static void main(String args[ ])
{
System.out.println("java");
System.out.println("java");
System.out.println("java");
System.out.println("java");
System.out.println("java");
System.out.println("java");
System.out.println("java");
System.out.println("java");
System.out.println("java");
System.out.println("java");
}
}
```

//With using for loop//

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



2.WAP to print numbers from 1 to 100

```
class Loop2
```

```
{
  public static void main(String args [ ])
    {
       int i;
       for(i=1;i<=100;i++)
       {
            System.out.println(i);
       }
     }
}
```

3.WAP to print numbers from 65 to 35

```
class PrintingNumbers
{
  public static void main(String args [ ])
    {
       int i;
       for(i=65;i>=35;i--)
       {
            System.out.println(i);
       }
     }
}
```

4.WAP to print all even numbers from 1 to 30

```
class LoopEven
{
  public static void main(String args [ ])
    {
       int i;
       for(i=1;i<=30;i++)
       {
                        if(i%2==0)
                    {
            System.out.println(i);
                }
              }
    }
}
```

5.WAP to print sum of all even numbers from 1 to 15

```
class LoopEvenSum
{
  public static void main(String args [ ])
    {
       int i,sum=0;
       for(i=1;i<=15;i++)
       {
                        if(i%2==0)
                    {
            sum=sum+i;
                }
              }
      System.out.println("Final sum value is: "+sum);
```

```
    }
}
```

6.WAP to print product of all even numbers from 1 to 10

```
class LoopEvenProduct
{
  public static void main(String args [ ])
    {
       int i,product=1;
       for(i=1;i<=10;i++)
       {
                        if(i%2==0)
                      {
            product=product*i;
                }
              }
       System.out.println("Final product value is: "+product);
    }
}
```

7.WAP to print value of 5!

```
class Factorial
{
  public static void main(String args [ ])
    {
       int i,fact=1;
       for(i=5;i>=5;i--)
       {

                        fact=fact*i;

              }
       System.out.println("Factorial value is: "+fact);
    }
}
```

8.WAP to print multiplication table

```
  class MultiplicationTable
  {
   public static void main(String args[ ])
  {
     int num=3;
    for(int i=1;i<=10;i++)
   {
     System.out.println(num+"*"+i+"="+(num*i));
  }
  }
  }
```

28-07-2020

9.WAP to print
a.Sum of first 10 natural numbers
b.product of first 10 natural numbers

```
class SumPdt
{
 public static void main (String args [ ] )
{
  int sum=0,pdt=1;
  for(int i=1;i<=10;i++)
  {
    sum=sum+i;
   }
  System.out.println("Sum value is : " +sum);
  for(int j=1;j<=10;j++)
   {
      pdt=pdt*j;
   }
  System.out.println("Product value is : " + pdt);
   }
}
```

10. WAP to count all numbers present between 1 to 30 which are divisible by 5

```
class Count
{
    public static void main(String args[ ])
 {
   int count=0,num=5;
  for(int i=1;i<=30;i++)
  {
     if(i%num==0)
     {
       count=count+1; //count++
      }
}
     System.out.println("Final count is : "+count );
}
}
```

11.WAP to print Fibnocci series
0 1 1 2 3 5 8 13

```
class Fibnocci
{
    public static void main(String args[ ])
 {
   int f=0,f1=1,f2;
          System.out.print(f+" ");
          System.out.print(f1+" ");
          for(int i=1;i<=6;i++)
             {
                 f2=f+f1;
```

```
                System.out.print(f2+" ");
                f=f1;
                f1=f2;
            }
}}
```

12.WAP to swap two numbers
a.using third variable
b.without using third variable

```
class SwapNum1
{
    public static void main(String args[ ])
 {
   int a=10,b=20,c;
    c=a;       //c=10
    a=b;       //a=20
    b=c;       //b=10
    System.out.println("After swapping : " +a+ " " +b);
}}

class SwapNum2 //without using third variable
{
    public static void main(String args[ ])
 {
   int a=10,b=20;
    a=a+b;       //a=10+20---->a=30
    b=a-b;       //b=30-20----->b=10
    a=a-b;       //a=30-10------>a=20
    System.out.println("After swapping : " +a+ " " +b);
}}
```

13.WAP to check whether number 13 is prime number or not

```
class PrimeNumber
{
    public static void main(String args[ ])
 {
    int num = 13;
    boolean flag = true;
    for(int i = 2; i < num; ++i)
     {
          // condition for nonprme number
           if(num % i == 0)
            {
                flag = false;
                break;
            }
      }
       if (flag==true)
            System.out.println(num + " is a prime number. ");
       else
            System.out.println(num + " is not a prime number. ");
   }
}
```

29-07-2020

14.WAP to find the square of 4

```
class PowerOf
{
public static void main(String args[ ])
{
    int num=4,n=2,a=1,pow;
    for(int i=1;i<=n;i++)
{
     pow=a*num ;
     a=pow;
     if(i==n)
     System.out.println(num +" to the power of "+ n + " is : " +pow);
     System.out.println(Math.pow(num,2));        // actual way to represent in java
}
}}
```

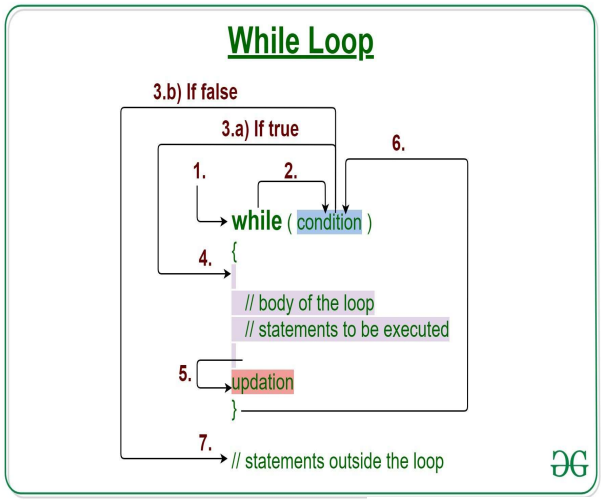
// Note: Math is predefine class
pow(num,square) is a method
import statement is for using classes of one package into our own class or own package.

//In for loop if we have not given condition JVM will consider it as true and run loop for infinite times.

```
class InfiniteLoop
{
      public static void main(String args[ ])
      {
          // for( ; ; ) //for(default;true;)
            for(int i=1; ;i++)
            {
              System.out.println("java");
             }
      }
}
```

//Command for stopping infinite loop in command prompt is ctrl+c

While Loop



1.WAP to print 1 to 30 using while loop

```
class WLoop1
{
    public static void main(String args[ ])
     {
         int i=1;
         while(i<=30)
          {
            System.out.println(i);
             i++;
          }
      }
```

```
}
```

Note:

-while( ) is an illegal statement in java
-while(boolean) is a valid statement
-while(true)
{

}
-Above loop is going to run infinite times

2.WAP to print 30 to 5 using while loop

```
class WLoop2
{
   public static void main(String args[ ])
    {
        int i=30;
        while(i>=5)
         {
           System.out.println(i);
            i--;
         }
    }
}
```

3.
```
class WLoop3
{
   public static void main(String args[ ])
    {
        int i=30;
        while(i>=5)
         {
           i--;
         }
         System.out.println(i);
    }
}
```

Output: 4

//Note :- In the above program once condition is false JVM come out of loop and print i value only once.
And i value is no more 30 it changes to 4 means whatever we did in while loop with i is going to updated

4.WAP to print
a. A to Z
b. Z to A
c. alternate charcters

```
class PrintCharacters
{
   public static void main(String args[ ])
    {
      char ch1='A',ch2='Z';
      while(ch1<='Z')
```

```
    {
        System.out.print(ch1);
        ch1++;
    }
    System.out.println(" ");
    while(ch2>='A')
    {
       System.out.print(ch2);
       ch2--;
    }
    System.out.println(" ");
    char ch3='A';
    while(ch3<='Z')
    {
        System.out.print(ch3);
        ch3+=2;//ch3=ch3+2
    }
   }
}
```

5.WAP to find reverse of a number

```
class ReverseNumber
{
  public static void main(String args [ ])
    {
       int num=123,rev=0,rem;
       while(num>0)
       {
         rem=num%10;
         num=num/10;
         rev=rev*10+rem;
       }
            System.out.println("Reverse of num is : "+rev);
    }
}
```

//working

while(123>0)	while(12>0)	while(1>0)	while(0>0)
rem=123%10---rem=3	rem=12%10---rem=2	rem=1%10---rem=1	
num=123/10---num=12	num=12/10---num=1	num=1/10---num=0	
rev=(0*10)+3---rev=3	rev=(3*10)+2---rev=32	rev=(32*10)+1---rev=321	

30-07-2020

Note: we use while loop when we don't know how many iteraions are going to execute.

---> Pallindrome Number : if we reverse number we should get original number. For ex: 121,111,1221

6.WAP to check whether your number is pallindrome or not

```
class PallindromeNumber
  {
    public static void main(String args [ ])
      {
        int num=121,rev=0,rem,temp=num;
        while(num>0)
        {
          rem=num%10;
          num=num/10;
          rev=(rev*10)+rem;
        }
          System.out.println("Reverse of num is : "+rev);
          if(temp==rev)
           {
             System.out.println("your number is Pallindrome number");
           }
            else
             {
               System.out.println("your number is not Pallindrome number");
             }
    }
  }
```

---> Armstrong Number : Any number we take individually add and cube it, we should get same number.
153--->1^3+5^3+3^3----->1+125+27=153

7.WAP to check whether your number is Armstrong or not.

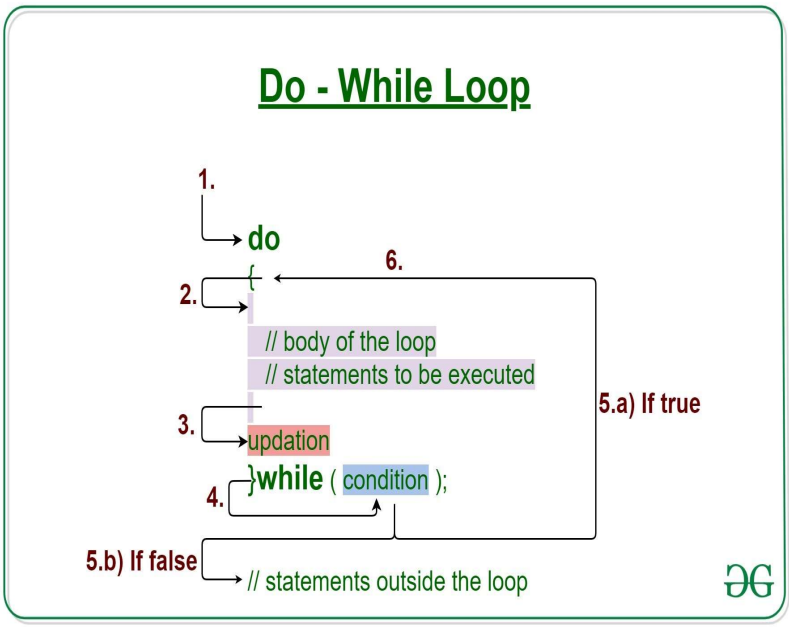
```
class ArmstrongNumber
  {
    public static void main(String args [ ])
      {
        int num=153,rev=0,rem,temp=num;
        while(num>0)
        {
          rem=num%10;
          num=num/10;
          rev=rev+(rem*rem*rem);
        }
          System.out.println("Reverse of num is : "+rev);
          if(temp==rev)
           {
             System.out.println("your number is Armstrong number");
           }
            else
```

```
        {
          System.out.println("your number is not Armstrong number");
        }
    }
}
  //working
```

while(153>0)	while(15>0)	while(1>0)	while(0>0)
rem=153%10---rem=3	rem=15%10---rem=5	rem=1%10---rem=1	
num=153/10---num=15	num=15/10---num=1	num=1/10---num=0	
rev=0+(3*3*3)---rev=27	rev=27+(5*5*5)---rev=152	rev=152+(1*1*1)---rev=153	

Do - While Loop

```
---------------------
class Demo
{
public static void main(String args[ ])
    {
       int a=10;//11//12//13
       do
       {
         System.out.println(a);//10//11//12//13
         a++;
       }while(a<=13);
}}
```



Differences between loops :-

```
---------------------------------
for loop :
------------
we go for, for-loop if we know nmber of iteration priorly(already).
syntax: for(initialisation;condition;inc/dec)
        {
          }
while loop :
--------------
we go for while loop if we don't know number of iterations.
syntax: initialisation;
        while(condition)
        {
          inc/dec;
        }
do while loop :
------------------
when we want our loop should run atleast one time then we go for do whle loop
syntax:   do
          {
            inc/dec;
          }while(condition);
```

31-07-2020

-WAP to check whether 2020 is leap year or not.

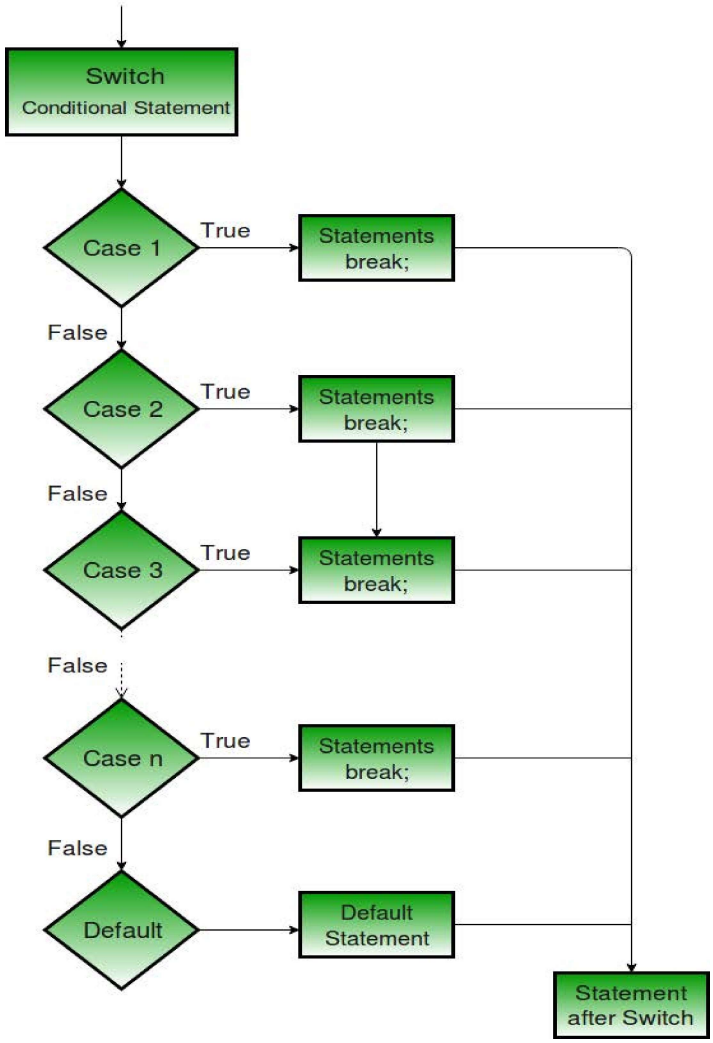
```
class LeapYear
{
  public static void main(String args [ ])
       {
         int year=2020;
         if(year%4==0)
        {
                System.out.println(year+" is Leap Year.");
        }
        else
        {
        System.out.println(year+" is not Leap Year.");
        }
       }
}
```

Switch case statements

---

syntax:

```
switch(expression)
{
  case value1:
                  break;
  case value2:
                  break;
  case value3:
                  break;
. . . . . .  . . . . . .so on
  default:
                  break;
 }
```



-WAP to take any number and print corresponding day by refering below table.

```
class SwitchDay
{
  public static void main(String args[ ])
   {
    int daynum=4;
    switch(daynum)
      {
        case 1: System.out.println("DayName : Monday");
                  break;
        case 2: System.out.println("DayName : Tuesday");
                  break;
        case 3: System.out.println("DayName : Wednesday");
                  break;
        case 4: System.out.println("DayName : Thursday");
                  break;
```

```
        case 5: System.out.println("DayName : Friday");
                break;
        case 6: System.out.println("DayName : Saturday");
                break;
        case 7: System.out.println("DayName : Sunday");
                break;
        default: System.out.println("Daynumber given is invalid as per calender");
                break;
      }
   }
}
```

-WAP to take any number and print by refering below table

```
class SwitchMonth
{
 public static void main(String args[ ])
 {
  int Monthnum=7;
  switch(Monthnum)
    {
        case 1: System.out.println("MonthName : January");
                break;
        case 2: System.out.println("MonthName : February");
                break;
        case 3: System.out.println("MonthName : March");
                break;
        case 4: System.out.println("MonthName : April");
                break;
        case 5: System.out.println("MonthName : May");
                break;
        case 6: System.out.println("MonthName : June");
                break;
        case 7: System.out.println("MonthName : July");
                break;
        case 8: System.out.println("MonthName : August");
                break;
        case 9: System.out.println("MonthName : September");
                break;
        case 10: System.out.println("MonthName : October");
                break;
        case 11: System.out.println("MonthName : November");
                break;
        case 12: System.out.println("MonthName : December");
                break;
        default: System.out.println("Monthnumber given is invalid as per calender");
                break;
      }
   }
}
```

break

-break keyword is used to break the control flow.

-We can use break keyword in switch case statements and in iterative statements also(Loops).

Q. What if we skip break keyword in case?
A. - If we skip break keyword, we wil not get compile error.
If there is no break keyword in case statement JVM will execute all the cases irrespective of condition until it finds next break statement.

Q. what if we skip break keyword in default ?
A. It does not matter because after default nothing is there to execute. So, writing of break keyword is not mandatory in default statement.

-WAP to print datatype depends on daynum and datatype.

daynum	datatype
1,2,3,4,5	weekdays
6	first weekend
7	second weekend
>7 or <1	invalid

```
class BreakDay
{
 public static void main(String args[ ])
 {
  int number=2;
  switch(number)
    {
       case 1:
       case 2:
       case 3:
       case 4:
       case 5: System.out.println("weekday");
                 break;
       case 6: System.out.println("First weekend");
                 break;
       case 7: System.out.println("Second weekend");
                 break;
       default: System.out.println("Abnormal Day");
                 break;
     }
  }
}
```

Note :

- Duplicate case values are not allowed, if we write we will get compile time error.

```
class DuplicateDay
{
 public static void main(String args[ ])
 {
  int number=2;
  switch(number)
    {
       case 1:
       case 2:
       case 3:
```

```
        case 4:
        case 5: System.out.println("weekday");
                break;
        case 5: System.out.println("First weekend");
                break;
        case 7: System.out.println("Second weekend");
                break;
    } } }
```

```
E:\Qspiders Java\programs>javac DuplicateDay.java
DuplicateDay.java:14: error: duplicate case label
        case 5: System.out.println("First weekend");
        ^
1 error
```

- Allowed data types for case values are
  byte,short,int and char
  long,double,float are not allowed
  String is allowed from 1.7 version of java

Programs

-WAP to check all the prime numbers from 1 to 15

-WAP to find sum of even numbers from given number 1234

2+4--->6

03-08-2020

Pattern programming

1.
```
*****
*****
*****
```

```
class Star
{
  public static void main(String args[ ])
   {
     for(int i=1;i<=3;i++) //outer loop for no of rows
      {
        for(int j=1;j<=5;j++) //inner loop for no of columns
        {
          System.out.print("*");
        }
          System.out.println(" ");
}}}
```

2. ****
****
*
****

```
class Star2
{
  public static void main(String args[ ])
   {
     for(int i=1;i<=4;i++)
      {
        for(int j=1;j<=4;j++)
        {
          if(i==3 && j>=2)
          {
          System.out.print(" ");
          }
          else
          {
            System.out.print("*");
          }
        }
          System.out.println(" ");
}}}
```

3. ****
####
****
####

```
class Star3
{
  public static void main(String args[ ])
   {
     for(int i=1;i<=4;i++)
      {
        for(int j=1;j<=4;j++)
        {
          if(i%2==0) //(i==2||i==4)
          {
          System.out.print("#");
          }
          else
          {
            System.out.print("*");
          }
        }
          System.out.println(" ");
}}}
```

4. *#*#
##
##
##

```
class Star4
```

```
{
  public static void main(String args[ ])
   {
     for(int i=1;i<=4;i++)
      {
        for(int j=1;j<=4;j++)
        {
          if(j==2||j==4)//(j%2==0)
          {
          System.out.print("#");
          }
          else
          {
            System.out.print("*");
          }
        }
          System.out.println(" ");
}}}
```

5.
```
*
*
*
*
class Star5
{
  public static void main(String args[ ])
   {
     for(int i=1;i<=4;i++)
      {
        for(int j=1;j<=4;j++)
         {
            if(i==j)
            {
              System.out.print("*");
            }
            else
             {
               System.out.print(" ");
             }
          }
            System.out.println( );
       }
   }
}
```

```
6.111
  222
  333
```

```
7.123
  123
  123
```

```
8.123
  456
  789
```

```
9.AAA
  BBB
  CCC
```

```
10.ABC
   ABC
   ABC
```

```
11. ABC
    DEF
    GHI
```

04-08-2020

6.
```
111
222
333
```

```
class Pattern6
{
  public static void main(String args[ ])
   {
     for(int i=1;i<=3;i++)
      {
        for(int j=1;j<=3;j++)
        {
           System.out.print(i);
        }
         System.out.println();
}}}
```

7.
```
123
123
123
```

```
class Pattern7
{
  public static void main(String args[ ])
   {
     for(int i=1;i<=3;i++)
      {
        for(int j=1;j<=3;j++)
        {
           System.out.print(j);
        }
          System.out.println();
}}}
```

8.
```
123
456
789
```

```
class Pattern8
{
  public static void main(String args[ ])
   {
     int a=1;
     for(int i=1;i<=3;i++)
      {
        for(int j=1;j<=3;j++)
        {
          System.out.print(a);
          a++;
        }
      System.out.println();
}}}
```

9. AAA
BBB
CCC

```
class Pattern9
{
  public static void main(String args[ ])
   {
      char ch='A';
      for(int i=1;i<=3;i++)
       {
         for(int j=1;j<=3;j++)
         {
            System.out.print(ch);
         }
          ch++;
         System.out.println();
}}}
```

10. ABC
ABC
ABC

```
class Pattern10
{
  public static void main(String args[ ])
   {
      for(int i=1;i<=3;i++)
       {
         char ch='A';
         for(int j=1;j<=3;j++)
         {
           System.out.print(ch);
           ch++;
         }
       System.out.println();
}}}
```

11. ABC
DEF
GHI

```
class Pattern11
{
  public static void main(String args[ ])
   {
      char ch='A';
      for(int i=1;i<=3;i++)
       {
         for(int j=1;j<=3;j++)
         {
           System.out.print(ch);
           ch++;
         }
       System.out.println();
}}}
```

12.


```
class Pattern12
{
  public static void main(String args[ ])
   {
     for(int i=1;i<=4;i++)
      {
        for(int j=1;j<=4;j++)
        {
         if(i>=j)
         {
         System.out.print("*");
         }
          else {
          System.out.print(" ");
        }}
          System.out.println();
}}}
```

13.


```
class Pattern13
{
  public static void main(String args[ ])
   {
     for(int i=4;i>=1;i--)
      {
        for(int j=1;j<=4;j++)
        {
         if(i>=j)
         {
         System.out.print("*");
         }
          else {
          System.out.print(" ");
        }}
          System.out.println();
}}}
```

14.


14 and 16 are similar, there is space after *

```
class Pattern14
{
  public static void main(String args[ ])
   {
     for(int i=1;i<=4;i++)
```

```
        {
          for(int j=4;j>=1;j--)
          {
           if(i>=j)
           {
           System.out.print("*");
           }
           else {
           System.out.print(" ");
          }}
           System.out.println();
}}}
```

15.

15 and 17 are similar, there is space after *

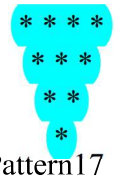
```
class Pattern15
{
 public static void main(String args[ ])
  {
    for(int i=1;i<=4;i++)
     {
       for(int j=1;j<=4;j++)
       {
        if(i<=j)
        {
        System.out.print("*");
        }
        else {
        System.out.print(" ");
       }}
        System.out.println();
}}}
```

16.

```
class Pattern16
{
  public static void main(String args[ ])
   {
     for(int i=1;i<=4;i++)
      {
        for(int j=4;j>=1;j--)
        {
         if(i>=j)
         {
         System.out.print("* ");
         }
         else {
         System.out.print(" ");
        }}
```

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

17.



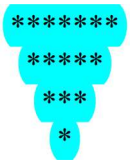
```
class Pattern17
{
  public static void main(String args[ ])
   {
      for(int i=1;i<=4;i++)
       {
         for(int j=1;j<=4;j++)
         {
          if(i<=j)
          {
          System.out.print("* ");
          }
          else {
          System.out.print(" ");
         }}
          System.out.println();
}}}
```

05-08-2020

18.


```
class Pattern18
{
 public static void main(String args[])
  {
   int star=1,space=3;//(spaces before star)
   for(int i=1;i<=4;i++)
   {
    for(int j=1;j<=space;j++)
    {
       System.out.print(" ");
    }
    for(int k=1;k<=star;k++)
    {
       System.out.print("*");
    }
    star=star+2;
    space=space-1;
    System.out.println(" ");
   }
}}
```

19.

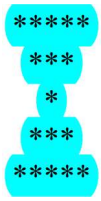



```
class Pattern19
{
 public static void main(String args[])
  {
   int star=7,space=0;//(spaces before star)
   for(int i=1;i<=4;i++)
   {
    for(int j=1;j<=space;j++)
    {
       System.out.print(" ");
    }
    for(int k=1;k<=star;k++)
    {
       System.out.print("*");
    }
    star=star-2;
    space=space+1;
    System.out.println(" ");
   }
}}
```

20.

```
class Pattern20
{
 public static void main(String args[])
 {
  int star=1,space=3;
  for(int i=1;i<=7;i++)
  {
   for(int j=1;j<=space;j++)
   {
     System.out.print(" ");
   }
   for(int k=1;k<=star;k++)
   {
     System.out.print("*");
   }
    if(i<=3 )
     {
      star=star+2;
      space=space-1;
     }
     else {
         star=star-2;
       space=space+1;
     }
   System.out.println(" ");
  }
}}
```

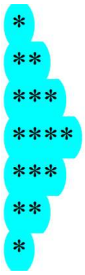
21.



```
class Pattern21
{
 public static void main(String args[])
 {
  int star=5,space=0;
  for(int i=1;i<=5;i++)
  {
   for(int j=1;j<=space;j++)
   {
     System.out.print(" ");
   }
   for(int k=1;k<=star;k++)
   {
```

```
        System.out.print("*");
      }
       if(i<=2 )
        {
         star=star-2;
         space=space+1;
        }
       else {
            star=star+2;
          space=space-1;
        }
      System.out.println(" ");
    }
}}
```

22.



```
class Pattern22
{
 public static void main(String args[])
  {
   int star=1,space=0;
   for(int i=1;i<=7;i++)
   {

    for(int k=1;k<=star;k++)
    {
       System.out.print("*");
    }
     if(i<=3 )
      {
       star=star+1;
      }
       else {
            star=star-1;
            }
    System.out.println(" ");
   }
}}
```

23.

```
class Pattern23
{
 public static void main(String args[])
 {
  int star=1,space=3;
  for(int i=1;i<=7;i++)
  {
   for(int j=1;j<=space;j++)
   {
     System.out.print(" ");
   }
   for(int k=1;k<=star;k++)
   {
     System.out.print("*");
   }
    if(i<=3 )
     {
      star=star+1;
      space=space-1;
     }
    else {
         star=star-1;
         space=space+1;
     }
   System.out.println(" ");
  }
}}
```

```
//prime number from 1 to 15

class Pn
{
 public static void main(String args[])
 {
  for(int i=1;i<=15;i++)
  {
   int count=0;
   for(int j=1;j<=i;j++)
   {
    if(i%j==0)
    {
     count++;//count=count+1
    }
   }
    if(count==2)
    {
     System.out.println(i+" Number is prime number");
    }
  }
}}
```

06-08-2020

- Every Program consists of 2 things
1. Data ------> Variables
2. Operations ------> Methods

Methods (MODULE-2)

-A set of java statements which is enclosed within curly braces having a declaration and which gets executed whenever we call it is called as method.

Syntax: AccessModifier NonAccessModifier returntype Methodname(Arguments)

```
{
    //set of statements//
}
```

AccessModifier NonAccessModifier returntype--------->Method Header

Methodname(Arguments)--------->Method signature

```
{
//set of statements//-------->Method implementation or method body
 }
```

Use of methods

- It is not recommended to write a business logic directly in class.
- So java says for such situation make use of methods.

For Ex: (Not Recommended)

```
class ATM
{
  System.out.println("withdrawl money");
   }
```

(Recommended)

```
class ATM
{
 public static void Withdrawl()
 {
   System.out.println("enter amount to withdrawl");
 }
      }
```

Access Modifier

-In java we have 4 access modifiers ----------> public,private,protected and default

```
public static void run()
private static void run()
protected static void run()
static void run() //default
```

Non Access Modifiers

-In java we have 2 non access modifiers and they are static and non static

```
public static void run() //static
public void run() //non static
```

Important Points

---

1) Methods are used to perform some specific task.
2) Method will be consisting of method header and method signature.
3) Method should be declared within scope of class.
4) Withn one class any number of methods we can write.
5) A method will be executed only when we call it.
6) we can call a method with help of method signature.
7) A method can be called any number of times depending on our requirement.
8) One method cannot be written in another method, it can only be called.

Ex1:
```
{public static void fly()
        public static void air()
        {
         }
    }
```
-Above example is invalid

Ex2:
```
public static void fly()
    {  air();
    }
    public static void air()
    {
    }
```
-Above is valid example

- The main advantage of using method is reusability,i.e once we created method, how many times we want we can execute it.

```
public class Bank
{
  public static void main(String args[])
  {
    System.out.println("Here we will show Customer's data");
    custDetails(); //call to custDetails()(go and execute custDetails method)
    System.out.println("---Showdetails again----");
    custDetails();
  }
//user define method
 public static void custDetails()
{
      System.out.println("Acc Name: Pooja");
      System.out.println("Acc No: 1234567");
      System.out.println("IFSC code: IOS345");
      System.out.println("Bank: IOB");
      System.out.println("Branch: Secunderabad");
      System.out.println("Email: poojaitsme@gmail.com");
      System.out.println("DOB: 25-07-1993");
      System.out.println("Aadhar: 23456789993");
      System.out.println("Pan: ANBG45K9");
      System.out.println("Contact: 9865472134");
```

```
        System.out.println("Address: 341/D,sector-3,ukkunagaram,visakhapatnam");
        System.out.println("Pincode: 530032");
}
}
```

program for practising

1.WAP for Book application to demonstrate usage of methods.

```
public class Book
{
  public static void main(String args[])
  {
    System.out.println("Here we will show Book details");
    bookDetails();
  }
 public static void bookDetails()
{
        System.out.println("Book Name: Machine Design");
        System.out.println("Book Pages: 957");
        System.out.println("Book Author: Ravikiran");
        System.out.println("Book Publisher: SS.Ratan");
        System.out.println("Book Edition: 7th edition");
}
}
```

2.WAP for car application to demonstrate usage of methods.

```
public class Car
{
  public static void main(String args[])
  {
    System.out.println("Here we will show Car's data");
    carDetails();
  }
 public static void carDetails()
{
        System.out.println("Company: Hyundai");
        System.out.println("Model : i10");
        System.out.println("Color: Red");
        System.out.println("Year of launch: 2015");
        System.out.println("Fuel Type: Diesel");
        System.out.println("Capacity: 5 persons");
        System.out.println("Engine Displacement: 1197 cc");
        System.out.println("Power: 81 bhp @ 6000 RPM");
        System.out.println("Torque: 114 Nm @ 4000 RPM");
        System.out.println("Transmission type: Manual");
        System.out.println("Mileage: 20 Km");
        System.out.println("Top Speed: 165 Kmph");
}
}
```

3.WAP for Account details of facebook user demonstrate usage of methods.

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

```
        System.out.println("Here we will show Customer's data");
        acntDetails();
        }
       public static void acntDetails()
       {
        System.out.println("Acc Name: Ravi");

        System.out.println("Phn No: 9573267321");
        System.out.println("Email: raviitsme@gmail.com");
        System.out.println("DOB: 25-07-1993");

        System.out.println("Location: Visakhapatnam");
        }
        }
```

07-08-2020

METHODS WITH SOME INPUT

-A method can have any number of inputs in the form of arguments.

-Ex: public static void add()//method with zero argument or 0 input
    public static void add(int i)//method with integer argument

-When we call any method with argument we have to pass that particular type of values.

-While passing arguments we have to make sure it will be as per sequence define in method declaration.

Example

```
public class Login
{
  public static void main(String args[])
   {
     registration("John","john123@gmail.com",9845915746l);
   }
//static method without specific returntype and with arguments
   public static void registration(String name,String email,long contact)
   {
     System.out.println("Name :"+name);
     System.out.println("Email address:"+email);
     System.out.println("Contact details: "+contact);
    }
 }
```

practise

1.WAP to create a facebooklogin app with args as username,password

```
public class FacebookLogin
{
  public static void main(String args[])
  {
    login("Ravi","r@Vi3421");
  }
  public static void login(String username,String password)
  {
    System.out.println("username : "+username);
    System.out.println("password : "+password);
  }
}
```

2.WAP to create a scholarship login app with args as halticket,dateofbirth,semester should print above info for 3 people.

```
public class ScholarshipLogin
{
  public static void main(String args[])
  {
    login("Ravi","10-08-1993",7);
    login("Kiran","25-07-1994",5);
    login("Prasanna","16-05-1995",3);
  }
  public static void login(String hallticket,String DOB,int semester)
  {
    System.out.println("Hallticket :"+hallticket);
    System.out.println("D.O.B :"+DOB);
    System.out.println("Semester: "+semester);
    System.out.println("--------------------");
  }
}
```

METHOD WITH RETURN TYPE

-If we want we can define a method with any specific return type.

for ex:

```
public static int add()
  {
    //perform operation whatever we want//
    return integertypevale;
  }
```

-Whenever we define a method with return type, we are telling that my method is going to return that particular type of data(value).

-It is mandatory to write return statement if we have a method with specific return type.

return keyword

-It is used inside a method whenever we define a method with specific return type

-return keywod is used to take data and exit from method

-return keyword must be last stattement in a method

-return keyword will not print the data.

-we cannot write more than one retun statement in a method because after one return statement it exits from the method, so that the next return statement cannot be executed.

Q. what is void ??
A. no specific returntype or method is not writng any specific type of value.

//method with returntype

```
public class IntRegistration
{
   public static long phnofield()
   {
      return 9874561337l;
    }
   public static char genderfield()
   {
      return 'M';
    }
   public static void main(String args[])
    {
      System.out.println(phnofield());
      System.out.println(genderfield());
     }
}
```

08-08-2020

1.Create an APP which includes the method as

```
public class QspidersLogin
{
   public static boolean login(String username,int pwd)
   {
     if(username=="Qspiders" && pwd==5454)
      {
     return true;
    }
    else
    {
     return false;
    }}

   public static void main(String args[])
    {
     System.out.println(login("Qspiders",5454));
     System.out.println(login("Qspiders",5354));
    }
}
```

note : execution will always start from main method.

2.WAP for area of tiangle ,rectangle and square

```
class Areas
{
public static float areaOfTri(int breadth,int height)
{
        float area1 = 0.5f * breadth * height;
        return area1;
}
public static float areaOfCircle(int radius)
{
        float pie=3.141f;
        float area2 = pie * radius * radius;
        return area2;
}
public static int areaOfRect(int breadth,int length)
{
        int area3 = length * breadth;
        return area3;
}
public static int areaOfSquare(int side)
{
        int area4 = side * side;
        return area4;
}
public static void main(String args[])
{
  System.out.println("Areaof Triangle is : "+ areaOfTri(5,10));
```

```
  System.out.println("Areaof Circlee is : "+areaOfCircle(5));
  System.out.println("Areaof Rectangle is : "+areaOfRect(3,8));
  System.out.println("Areaof Square is : "+areaOfSquare(4));
}
}
```

// WAP to print sum of even numbers 1234 (this logic can be used for any digit number)

```
  public class SumE
  {
   public static void main(String args[])
   {
     int num=1234,sum=0,rem;
     while(num>0)
     {
      rem=num%10;
      num=num/10;
      if(rem%2==0)
      {
        sum=sum+rem;
       }
     }
     System.out.println("Sum of even numbers is : "+sum);
   }
  }
```

// leap year

main condition: The year should be dvisible by 4
another conditon: The leapyear shouldn't be a century year or should be divisible by 400.

```
public class LeapYear
{
 public static void main(String args[])
 {
   int year=2020;
   if(year%4==0 && (year%100!=0 || year%400==0))
   {
     System.out.println("Leap Year");
   }
   else
   {
     System.out.println("Not Leap Year");
   }
}}
```

TYPES OF VARIABLES

Depending on declaration variables are divided into two types.
1.Local variable
2.Global variable(Data member)

Local variables

Variables which are declared inside the method(main() or userdefine) and within the scope of class({}). such variables are called as Local variables.

Are Local variables accessible everywhere ??

Local variables are accessible only within the method in which they are declared, they are not accessible outside of that method.
-if we try to access them we will get compile time error sayng that "Cannot find Symbol".

//Local variable//

```
public class Employee
{
   public static void displaydetails()
    {
     String ename="John";
     int eid=8075;
     String desg="Developer";
     System.out.println("Employee name: "+ename);
     System.out.println("Employee is : "+eid);
     System.out.println("Designation : "+desg);
    }
    public static void main(String args[])
    {

     displaydetails();
     String ename="John";
     int eid=8054;
     String desg="QA";
     System.out.println("Employee name: "+ename);
     System.out.println("Employee is : "+eid);
     System.out.println("Designation : "+desg);

    }
}
```

output

```
Employee name: John
Employee is : 8075
Designation : Developer
Employee name: John
Employee is : 8054
Designation : QA
```

Is Intialisation of local variable is mandatory ??

Yes,it is mandatory to intialise local variables, if we did not intialise we will get compile time error saying that "Variable might not intialised".

Examples

```
public class Student
{
  public static void main(String args[])
  {
     String sclname;//localvar
     String branch="HYD";
     System.out.println(sclname);//CTE(Compile time error) bcoz intialisation of local is mandatory
     displayDetails();
  }
  public static void displayDetails()
  {
     String sclname="S.T.H.C.S";//localvar
     String sname="Pooja";//lv
     int age=22;//lv
     long contact=95874596181;//lv
     String email="poojamine@gmail.com";//lv
     System.out.println(sname+" "+age+" "+contact+" "+email);
     System.out.println(sclname);
     System.out.println(branch);//CTE bcoz branch is local to main()
   }
}
```

- Local variables can only be default or final i.e we cannot declare a local variable as private,public or protected

Ex: public static void add()

```
    {
      int i=100;//local variable declaration is valid
      public int i=100;//loc var dec is Invalid
      final int i=100;//loc var dec is valid
      private int i=100;//loc var dec is Invalid
      protected int i=100;//loc var dec is Invalid
    }
```

10-08-2020

GLOBAL VARIABLES(Data members)

-Variables which are declared outside the method(main() or userdefine) and within the scope of class({}). such variables are called as global variables.

-global variables are divided into 2 types
1. static variables(also called as class variable)
   Ex: static int i=100;
2. Non static variable(also called as Instance variable)
   Ex: int i=100;

-Global variables are accessible everywhere with in the scope of a class.

Example

```
//Global variables//
class Student1
{
  //static global variables
  static String college="Qspiders";//global var
  static String stdname="Rohan";//global var
  static int passingmarks=60;//global var
  static int maxbacklogs=12;//global var
  public static void main(String args[])
  {
    System.out.println("Details from main method are:");
    System.out.println(college);
    System.out.println(stdname);
    System.out.println(passingmarks);
    System.out.println(maxbacklogs);
    displayDetails();
}
public static void displayDetails()
{
System.out.println("Details from display method are:");
System.out.println("Name : "+stdname);
System.out.println("College : "+college);
System.out.println("Marks : "+passingmarks);
System.out.println("AffordableBacks : "+maxbacklogs);
}}
```
output

```
E:\Qspiders Java\programs>java Student1
Details from main method are:
Qspiders
Rohan
60
12
Details from display method are:
```

Name : Rohan
College : Qspiders
Marks : 60
AffordableBacks : 12

```
public class Bank1
{
  static String custname="Pooja";
  static String bname="IndianOverseas";
  static long accno=24242421431;
  static String IFSC="IOBA127500";
  static String brname="Hyderabad";
  static String acctype="Savings";
  public static void displayDetails()
  {
       System.out.println("Cust Name : "+custname);
       System.out.println("Accno : "+accno);
       System.out.println("IFSC : "+IFSC);
       System.out.println("Branch name : "+brname);
       System.out.println("Account type : "+acctype);
   }
   public static void main(String args[])
   {
       System.out.println("---Details of Customer From---"+bname);
       displayDetails();
   }
}
```

output

```
--------
E:\Qspiders Java\programs>java Bank1
---Details of Customer From---IndianOverseas
Cust Name : Pooja
Accno : 2424242143
IFSC : IOBA127500
Branch name : Hyderabad
Account type : Savings
```

```
public class Bank2
{
  static String custname="Pooja";
  static String bname="IndianOverseas";
  static long accno=24242421431;
  static String IFSC="IOBA127500";
  static String brname="Hyderabad";
  static String acctype="Savings";
  public static void displayDetails()
  {
       String custname="Rohan";//local var
       long accno=24656565531;//local var
       System.out.println("Cust Name : "+custname);//Rohan
       System.out.println("Accno : "+accno);//2465656553
       System.out.println("IFSC : "+IFSC);//IOBA127500
```

```
        System.out.println("Branch name : "+brname);//Hyderabad
        System.out.println("Account type : "+acctype);//Savings
    }
    public static void main(String args[])
    {
        String bname="ICICI";//local var
        System.out.println("---Details of Customer From---"+bname);//ICICI
        displayDetails();
    }
}
```

output

```
E:\Qspiders Java\programs>java Bank2
---Details of Customer From---ICICI
Cust Name : Rohan
Accno : 2465656553
IFSC : IOBA127500
Branch name : Hyderabad
Account type : Savings
```

Q. What if local variable and global variable names are same?

A. If local var and global var names are same first priority is always given to local variables because they are inside method and nearer for execution to JVM.

```
class STDApp
{
//static global variables//
static String name="Qspiders";
static long contact=9573267325l;
static String email="Qspiders@gmal.com";
public static void enquiry()
{
  //local vars
  String name="QspidersKPHB";
  String email="Qspiderskphb@gmail.com";
//print statements
  System.out.println(name);
  System.out.println(contact);
  System.out.println(email);
}
public static void main(String args[])
{
   enquiry();
}}
```

Output

```
E:\Qspiders Java\programs>java STDApp
QspidersKPHB
9573267325
Qspiderskphb@gmail.com
```

Q. Is intialisation of global variable is mandatory?

A. They are not mandatory to intialise, if we did not intialise, JVM will take default values based on data types.

byte,short,int and long --------> 0
float and double ----------> 0.0
String ----------> null
boolean ----------> false
char ----------> empty space

program

```
class Demo
{
//static global 'variables without initialisation//
static int i;
static long lt;
static short s;
static byte b;
static char ch;
static boolean b1;
static float f;
static double d;
static String str;
public static void main(String args[])
{
System.out.println(i);
System.out.println(lt);
System.out.println(s);
System.out.println(b);
System.out.println(ch);
System.out.println(b1);
System.out.println(f);
System.out.println(d);
System.out.println(str);
}}
```
output

```
E:\Qspiders Java\programs>java Demo
0
0
0
0

false
0.0
0.0
null
```

-Global variables can be public, default, private, protected and final

-Global variables are also called as Data members.

DIFFERENCES BETWEEN LOCAL VARIABLE AND GLOBAL VARIABLE

Local variable	Global variable
Variables which are declared inside the method and within the scope of class.	Variables which are declared outside the method and within the scope of class.
Local variables are accessible only within the method in which they are declared, they are not accessible outside of that method.	Global variables are accessible everywhere within the scope of a class.
It is mandatory to initialize local variables, if we did not initialize we will get compile time error.	It is not mandatory to initialize, if we did not initialize, JVM will take default values based on data types.
There are no types for local variables.	Global variables are divided into 2 types i.e. static and non-static.
Local variables can be default or final.	Global variables can be public, private, protected, default or final.
Local variables are present in Stack area.	Global variables are present in Heap area.

METHOD OVERLOADING

-THE PROCESS OF DEVELOPING MULTIPLE METHODS WITH SAME NAME BUT DIFFERENT ARGUMENTS LIST IS CALLED AS METHOD OVERLOADING.

RULES FOR DEFINING ARGUMENT LIST

---

1. Number of arguments must be different.
   Ex: swiggy()
   swiggy(String name)
   swiggy(String name,int orderid)

2. Types(datatype) of arguments must be different.
   Ex: add(int i,int j)
   add(double d,double d1)
   add(String s1,String s2)

3. Sequence or position of Arguments must be different.
   Ex: login(String ul,int id)
   login(int id,String pwd)

-We go for method overloading when we want to perform one task in multiple ways.

Examples

---

a)Travelling(hyd-->bang)

-bus
-train
-aeroplane
-bike
-car.....etc

b)keeping phone password

-Nmberlock
-Drawing pattern
-Finger print
-Face print
-Face sensor

c)payment in any Ecommerce

-Credit
-Debit
-Wallets(Gpay,Phonepe,Amazonpay,Paytm)
-Cash on Delivery

d)Banking

-Online banking
-Physical baning
-ATM banking
-App banking

```
-------------
11-08-2020
-------------

class Mover
{
  public static void main(String args[])
  {
      add();
      add(100,200);
      add('A','B');//A=65,B=66
      add(100,"java");
      add("SQL",200);
   }
   public static void add()
   {
      System.out.println(10+20);
   }
   public static void add(int i,int j)
   {
      System.out.println(i+j);
   }
   public static void add(char ch1,char ch2)
   {
      System.out.println(ch1+ch2);
   }
   public static void add(int i,String s)
   {
      System.out.println(i+s);
   }
   public static void add(String s,int i)
   {
      System.out.println(s+i);
   }
}
output
-------
C:\Users\Ravi Kiran\Qspiders Java\programs>java Mover
30
300
131
100java
SQL200

-WAP to develop the APP for Payment Module using MethodOverloading.
class Payment
{
   public static void payment(String wallettype,int UID)
   {
```

```
      System.out.println("WalletType : "+wallettype);
      System.out.println("UID : "+UID);
   }
   public static void payment(String cardtype,long cardno,int cvvnumber)
   {
      System.out.println("CardType : "+cardtype);
      System.out.println("CardNo : "+cardno);
      System.out.println("CvvNumber : "+cvvnumber);
   }
   public static void payment(String type,String username,int pwd,long
Accountnumber)
   {
      System.out.println("Type : "+type);
      System.out.println("Username : "+username);
      System.out.println("Pwd : "+pwd);
      System.out.println("AccountNumber : "+Accountnumber);
   }
  public static void main(String args[])
  {
      payment("Gpay",9573);
      payment("Debitcard",4581643278941254l,522);
      payment("savings","ravikiran",6547523,354867852141);
   }
}
```

Q. Can we overload main() or not ?

A. Yes, we can overload main() beacuse JVM knows a main() which is havng arguments as String args[](command line arguments), so if we define any other methods whose name is main it will be consdered as userdefine methods.

Note:In practical or realtime it is suggested not to overload main().

```
class Mover1
{
  public static void main(String args[])
  {
      main();
      main(100,200);
      main('A','B');//A=65,B=66
      main(100,"java");
      main("SQL",200);
   }
   public static void main()
   {
      System.out.println(10+20);
   }
   public static void main(int i,int j)
   {
```

```
      System.out.println(i+j);
   }
   public static void main(char ch1,char ch2)
   {
      System.out.println(ch1+ch2);
   }
   public static void main(int i,String s)
   {
      System.out.println(i+s);
   }
   public static void main(String s,int i)
   {
      System.out.println(s+i);
   }
}
output
-------
C:\Users\Ravi Kiran\Qspiders Java\programs>java Mover1
30
300
131
100java
SQL200
```

Q. Can we overload final methods ?

A. Yes, we can overload final methods because final keyword says do not change implementation and in overloading we are not changing implementation rather we are changing arguments.

Q. Can we overload private methods or not ?

A. Yes, we can overload private methods because private methods are accessible everywhere in same class and overloading also happens within class.

Q. Can we overload non static methods or not ?

A. Yes, we can overload non static methods but to call them we have to create an Object.

Non static Methods

```
-------------------------
```

-If we did not mention static keyword in a method it will become non static.

```
 for ex: public void register()
         {
            }
```

-If we want to call non static methods from

1. static method we have to create an object
2. Non static method no need to create object we can access directly.

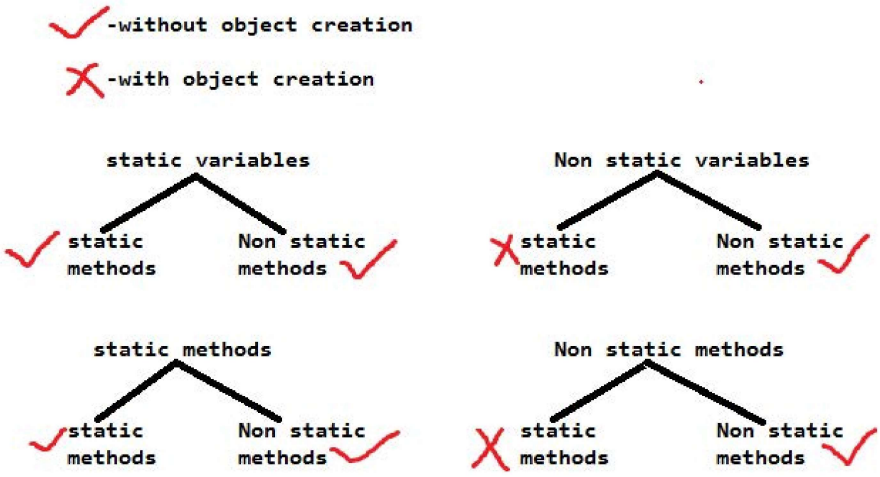
Syntax for object creation

```
-----------------------------
          classname refrencevar=new classname();
                                or
           classname referencevar=new constructor();
```

-new is a keyword which is responsible to create an object.

-with new keyword one object gets created and it loads all non static members(non static methods and non static variables) and return reference address.

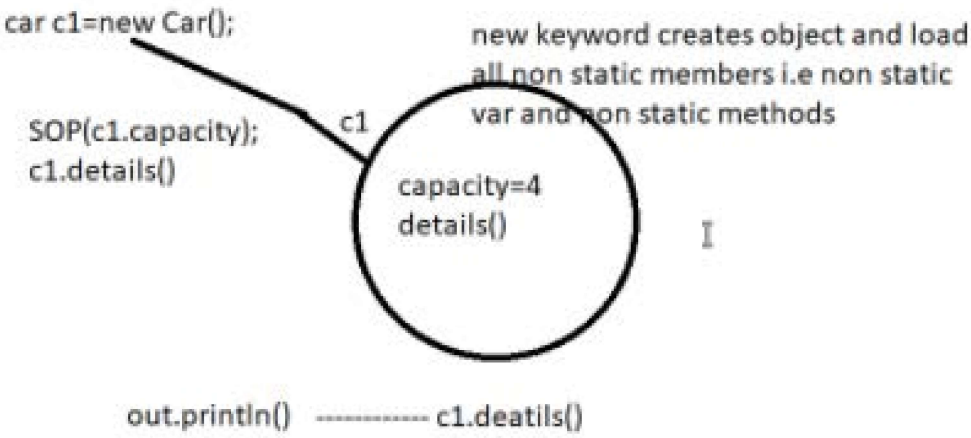
-Now using reference address and dot operator, we can access all non static methods and variables present inside object.



Conclusion points

```
-------------------
```

1. static variables can be accessed directly in static methods
2. static variables can be accessed directly in non static methods
3. Non static variables cannot be accessed directly in static methods
4. Non static variables can be accessed directly in non static methods
5. static method can be called directly from static methods
6. static method can be called directly from non static methods
7. Non static method cannot be called directly from static methods
8. Non static method can be called directly from non static methods




```
public class CarA
{
  static String brname="Audi";//static var
  int capacity=4;//non static var
   //non static
  public void details()
  {
    String color="Black";//local var
    long price=5500000l;//local var
    System.out.println("Car name : "+brname);
    System.out.println("Car capacity : "+capacity);
    System.out.println("Color is : "+color);
    System.out.println("Starting Price : "+price);
   }

   //static method
  public static void features()
  {
     //classname referencevar=new classname();
    CarA c1=new CarA();
    c1.details();
    System.out.println(c1.capacity);
   }
  public static void main(String args[])
  {
     features();
   }
}
output
--------
C:\Users\Ravi Kiran\Qspiders Java\programs>java CarA
Car name : Audi
Car capacity : 4
Color is : Black
Starting Price : 5500000
4
```

```
-----------
12-08-2020
-----------
Practise
---------
public class Book1
{
   static String bname="java";
   int bpages=578;
   String bpublisher="PoojaPrintingPress";
   static String bauthor="John.RJ";
   static String bcolor="Black";

   public static void bookDetails()
   {
     System.out.println("Name : "+bname);
     Book1 b1=new Book1();//here b1 is local
     System.out.println("Pages : "+b1.bpages);
     System.out.println("Published by : "+b1.bpublisher);
     System.out.println("Author is : "+bauthor);
     System.out.println("Color is : "+bcolor);
    }
    public void shopkeeper()
    {
     String sname="Ramu Anna";//local var
     String saddress="kingKoti beside busstop opposite to Bahubali tea
stall";//local var
     System.out.println("I have Purchased from : "+sname);
     System.out.println("Address : "+saddress);
     bookDetails();
    }
    public static void main(String args[])
    {
     //b1.shopkeeper();//CTE because b1 is local to bookDetails()
       System.out.println("---------Love Story of Book------");
       Book1 b2=new Book1();//here we can again create object as b1 also,
bcoz every time we create an object it will be local to its scope
       b2.shopkeeper();
     }
}
output
-------
C:\Users\Ravi Kiran\Qspiders Java\programs>java  Book1
---------Love Story of Book------
I have Purchased from : Ramu Anna
Address : kingKoti beside busstop opposite to Bahubali tea stall
Name : java
Pages : 578
Published by : PoojaPrintingPress
Author is : John.RJ
Color is : Black
```

EXECUTION PROCESS

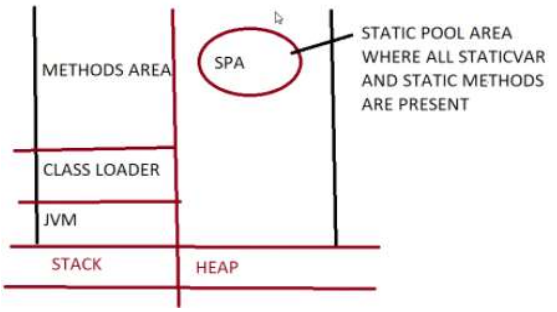
-Whenever we execute any program there will be two memory blocks that gets created.

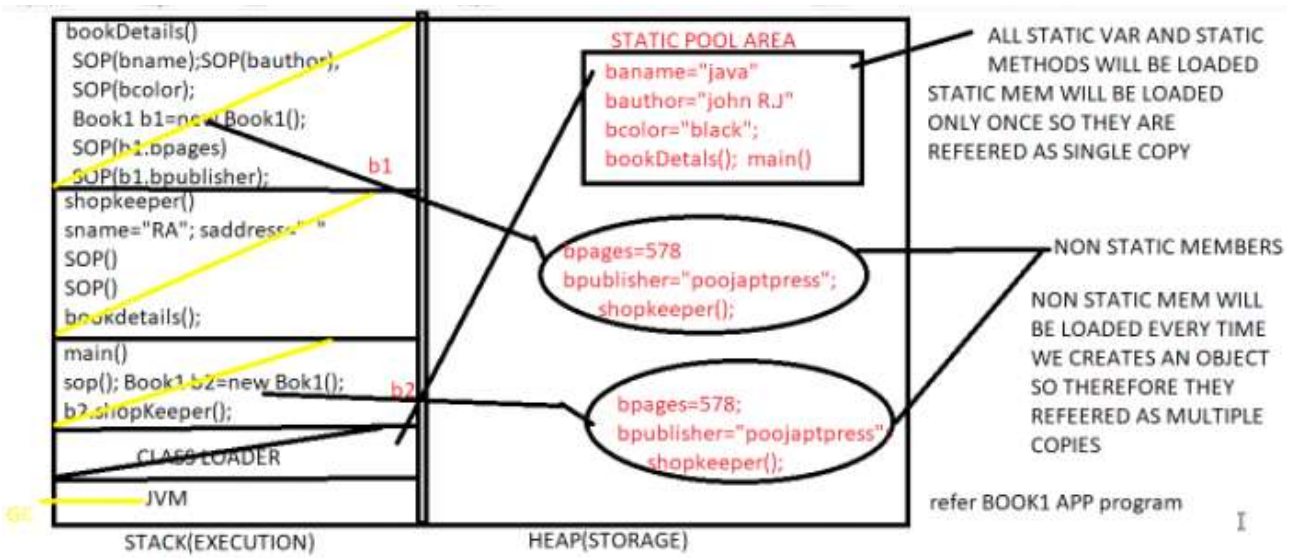
1. stack area also called as Execution Area
2. Heap area also called as Storage Area

Important points

---

1. First JVM will enter into stack area by making a call to class Loader.
2. class loader is like a function(program) whose job is to load all static members into static pool area(SPA).
3. SPA is a part of heap area.
4. Once class loader loads all static members into SPA its job is done, so it will come out of stack area.
5. Next JVM will make a call to main method.
6. Upon call to any method, method will get loaded in stack area under method area.
7. Next Execution of main method will starts.
8. Once entire execution is completed main() also come out of stack area.
9. Next JVM before coming out of stack area it will make a call to garbage collector whose job is to cleanup entire memory for next execution.





Conclusion
---

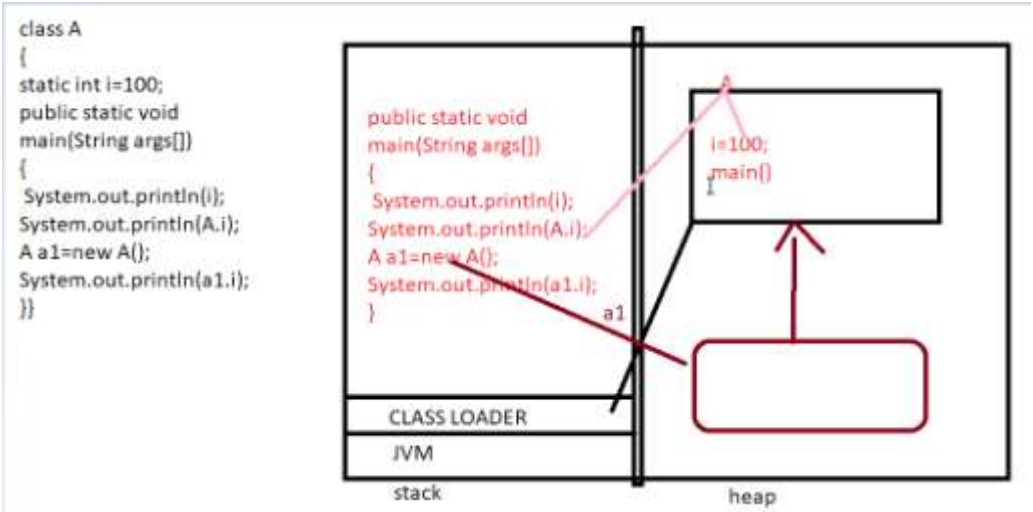
1. static methods and static variables are available (SPA) before execution starts, thats why they are accessible without object creation.
2. Non static members are not available before execution so to access them we have to create an object. With new keyword object will get created and loads all non static members inside it.
3. Non static variables and non static methods, they are loaded together in object so thats why since they belongs to same area non static variable can be accessed directly in non static method.
4. static members will get loaded only once in SPA thats why they are referred as single copy.
5. Since Non static members will get loaded whenever we created an object thats why they referred as multiple copies.
6. LOCAL variables are present in stack area and GLOBAL variables are present in heap area.
7. static variables are present in SPA(Heap).
8. Non static variables are present in object(Heap).
9. All reference variables are local variables.
10. Since once execution completes, method will come out of stack area that is the reason local variable scope is within method beacuse they are present inside method.

13-08-2020

Multiple ways to access static members

-Static members can be accessed in 3 ways

1.Directly---->because they will get loaded only once in SPA.
2.Through class name------>because class name and static pool area names are same.
3.Through Object------>because there is one way connection between Object and static pool area.

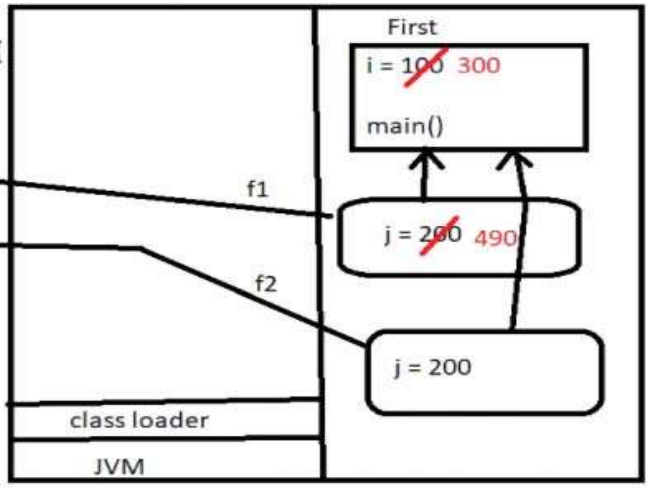


```
class A
{
static int i=100;
public static void main(String args [])
{
System.out.println(i);
System.out.println(A.i);
A a1=new A();
System.out.println(a1.i);
}}
output
------
E:\Qspiders Java\programs>java A
100
100
100

Conclusion
------------
1.We go for first approach if we access static members of same class.
2.We go for second approach if we access static members of different
class.
3.Third approach is not recommended.

public class First
{
      static int i=100;
      int j=200;
  public static void main(String[] args)
   {
   First f1=new First();
   f1.i=300;
   f1.j=490;
   First f2=new First();
   System.out.println(i);
   System.out.println(f1.i);
   System.out.println(f1.j);
   System.out.println(i);
   System.out.println(f2.j);
}}
output
--------
E:\Qspiders Java\programs>java First
300
300
490
300
200
```

```
static int i=100; class First {
  int j=200;
     public static void main(String[] args) {
          // TODO Auto-generated
method stub
     First f1=new First();
     f1.i=300;
     f1.j=490;
     First f2=new First();
     System.out.println(i);//100 300
     System.out.println(f1.i);//300 //300
     System.out.println(f1.j);//490 //490
     System.out.println(i);// 300 //300
     System.out.println(f2.j);// 200 //200
  }
  }
```



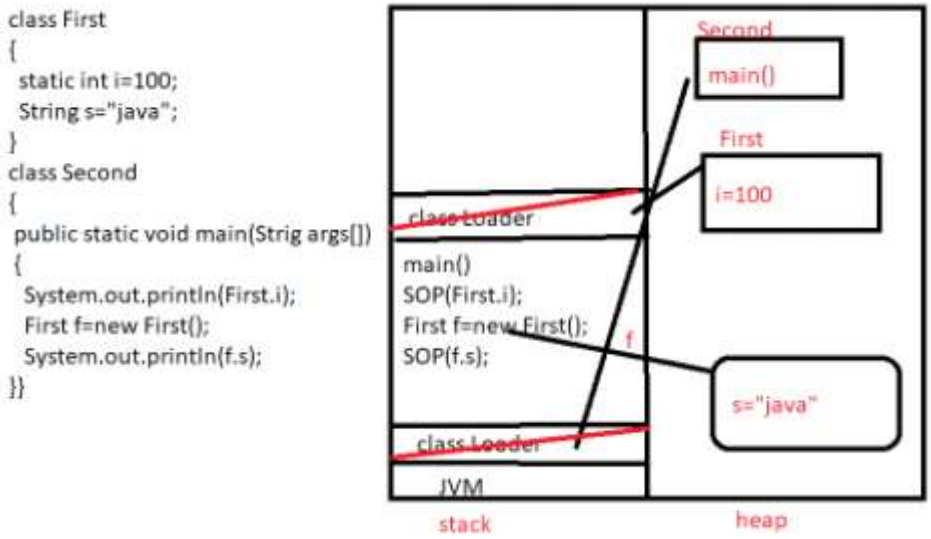
- static variables have been allocated memory only once in SPA because they will be loaded only once in SPA.
- So if i change its value it is going to change permanently(throughout program)
- Therefore static variables are called as class variables.
- Non static variables will have seperate memory for every object because they will be loaded always when we create object.
- So if we change j value it will be change for that particular object not for whole class(not permanently).
- Therefore non static variables are also called as Instance Variables.

Static	Non static
1.static means single copy.	1.Non static means multiple copies.
2.static members are present in static pool area.	2.Non static members are present in object.
3.static members can be accessed in 3 ways 1.Directly 2.Through class name 3.Through object reference	3.Non static members can be accessed through object.
4.static members will get loaded only once in SPA.	4.Non static members will get loaded whenever we created an object.
5.class loader is responsible to load static members.	5.new keyword is responsible to load all non static members.
6.Any changes made to static variables will effect entire class that is why static variables are also called as class variables.	6.Any changes made to non static variable will effect only a particular object that why they are called as instance variables.
7.We go for static, if the data is fixed. Static String clg="Qspiders";	7.We go for non static, if the data is changing. String courses="CSE";
8.static methods cannot be inherited.	8.Non static methods can be inherited.
9.static methods cannot be overridden.	9.Non static methods can be overridden.

14-08-2020

Acessing members of one class in another class

- We can access both static and non static members of one class in another class.
- If it is static, we can access through class name.
- If it is non static, we can access through object.
- If multiple class are there save file name with classname which have main method.




```
class First
{
   static int i=100;
   String s="java";
}
class Second
{
 public static void main(String args[])
  {
    System.out.println(First.i);
    First f=new First();
    System.out.println(f.s);
}}
output
-------
C:\Users\Ravi Kiran\Qspiders Java\programs>java Second
100
java
```

- Firstly class loader will load static members of second class in SPA whose name is Second.
- When SOP(First.i) is executed again, class loader will load First class static members in SPA whose name is First.

SCANNER CLASS

- Scanner is a predefine class(already defined) which is used to take an input from user during runtime or execution time.
- This Scanner class is used when we want to take inputs from user not from programmer.

Steps to use Scanner class

1. Write first statement in program as

```
import java.util.Scanner;
          OR
import java.util.*;
```

Reason: Since Scanner is a class which is present in java.util package.

Package: Package is like a folder where we can keep common classes together at one place.

```
Ex:Gallery
 |_family(folder)
 |_friends(folder)
 |_gilfriend(folder)
 |_something(folder)
```

- So in java, if we are using any class of any package we must have to write import statement.

import: it is a keyword which indicates that we are using any other class code in our own class.

2. Create an Object of Scanner class

```
Scanner s=new Scanner(System.in);
```

-here System.in indicates that we are reading input from system.

3. Using Scanner class reference call Scanner class methods

methods of Scanner class are:

```
nextByte() -----------> For taking byte value as input
nextBoolean() --------> For taking boolean value as input
nextShort() ----------> For taking Short value as input
nextInt()   ----------> For taking integer value as input
nextFloat() ----------> For taking float value as input
nextDouble() ---------> For taking double value as input
nextLong() -----------> For taking long value as input
next()    ------------> For taking String value as input without space
nextLine() -----------> For taking String value as input with space
```

Program

```
//Scanner class
//step-1
import java.util.Scanner;
class StdDetails
{
  public static void main(String args[])
  {
   //step-2
   Scanner s=new Scanner(System.in);
   //step-3
   System.out.println("Enter name ");
   String name=s.nextLine();
   System.out.println("Enter age ");
   int age=s.nextInt();
   System.out.println("Enter percentage ");
   float percentage=s.nextFloat();
   System.out.println("Enter result status ");
   boolean result=s.nextBoolean();
   System.out.println("Enter contact number ");
   long contact=s.nextLong();
   char gender;
   System.out.println("Enter Gender ");
   gender=s.next().charAt(0);
   System.out.println("Student Details Entered by you are");
   System.out.println("Name: "+name);
   System.out.println("Age : "+age);
   System.out.println("Gender : "+gender);
   System.out.println("Contact : "+contact);
   System.out.println("Score : "+percentage);
   System.out.println("Result: "+result);
}}
```
output

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java StdDetails
Enter name
Ravi Kiran
Enter age
27
Enter percentage
86.80
Enter result status
true
Enter contact number
9573267521
Enter Gender
Male
Student Details Entered by you are
Name: Ravi Kiran
Age : 27
```

```
Gender : M
Contact : 9573267521
Score : 86.8
Result: true
```

Note-1:

-If we did not give the value properly while asking during runtime, we will get "InputMismatchException"

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java StdDetails.java
Enter name
Ravi kiran
Enter age
44.5
Exception in thread "main" java.util.InputMismatchException
```

-Here age should be of integer type but we gave float type.

Note-2:

-We don't have any method called as nextChar() to take character as input rather we have a method like,

next().charAt(Index)------>To take character input i.e.

-First we will enter string value from that it takes only first character

Ex:Entered value is---->Abcd

```
charAt(0)-A
charAt(1)-b
charAt(2)-c
charAt(3)-d
charAt(>3)-Exception
```

17-08-2020

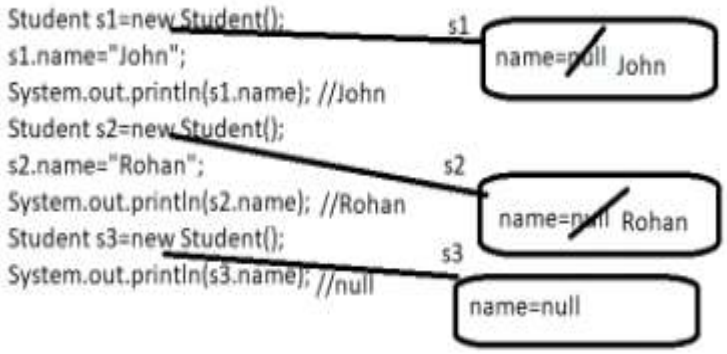
CONSTRUCTORS

Need of Consructor

Non static variables plays a special role in every program i.e. non static refers to muliple copies which means that for every object there is a seperate memory allocated for non static variable, which means programmer can intialise everytime new value to non static variable.

For Example;

```
 class Student
 {
  String name;
  public static void main(String args[])
   {
      Student s1=new Student();
      s1.name="John";
      System.out.println(s1.name);
      Student s2=new Student();
      s2.name="Rohan";
      System.out.println(s2.name);
      Student s3=new Student();
      System.out.println(s3.name);
  }}
```



Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Student
John
Rohan
null
```

- In above program name value is not initialised so it is name=null
- If we want to intialise it we have to do manuallyintialise value for every object which is a lengthy approach and chances are there that programmer may forget to intialise.
- Therefore to initialise a non static variable, a special characteristics concept is given as constructor.

Definition

- Constructor is a special type of method which gets executed whenever we created an object
- The main purpose of constructor is to initialise a non static variable.

Syntax: AccessModifier consructoname(arg/no args)

```
{
   //body of constructor
 }
```

Rules for defining constructor
---

1. Constructor can be public, private, protected or default.
2. Constructor cannot be static, non static, final or abstract.
3. Constructor name must be same as that of classname.
4. Constructor does not have any return type not even void.
5. Constructor can be with arguments or without arguments.

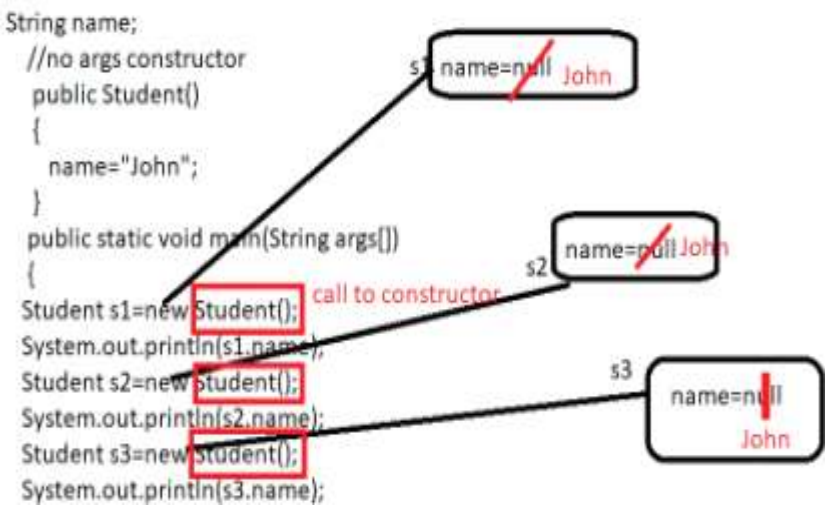
Types of Constructors:-

They are of 2 types-
1. No argument Constructor
2. Parameterized Constructor

No argument Constructor

-If a constructor does not have any argument it is called as No argument constructor.

```
class Student  //Ex-1
{
 String name;
 //define constuctor
 public Student()
 {
    name="John";
  }
 public static void main(String args[])
  {
    Student s1=new Student();
    System.out.println(s1.name);
    Student s2=new Student();
    System.out.println(s2.name);
    Student s3=new Student();
    System.out.println(s3.name);
 }}
```



Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Student
John
John
John
```

```
class Dog    //Ex-2
{
 String name;
 public Dog()
 {
  name="Tommy";
 }
   public static void main(String args[])
  {
      Dog f=new Dog();
      System.out.println(f.name);
      Dog f1=new Dog();
      System.out.println(f1.name);
  }}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Dog
Tommy
Tommy
```

```
class Book     //Ex-3
 {
  String bookname;
  int bookprice;
  public Book()
  {
    bookname="Heat Transfer";
    bookprice=600;
  }
    public static void main(String args[])
    {
      Book b=new Book();
```

```
      System.out.println(b.bookname);
      System.out.println(b.bookprice);
  }}
```
Output:
```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Book
Heat Transfer
600
```

```
class Car     //Ex-4
{
String carname;
int price;
String color;
int capacity;
public Car()
{
  carname="Hyundai i20";
  price=700000;
  color="Red";
  capacity=4;
}
    public static void main(String args[])
    {
      Car c=new Car();
      System.out.println(c.carname);
      System.out.println(c.price);
      System.out.println(c.color);
      System.out.println(c.capacity);
  }}
```
Output:
```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Car
Hyundai i20
700000
Red
4
```

18-08-2020

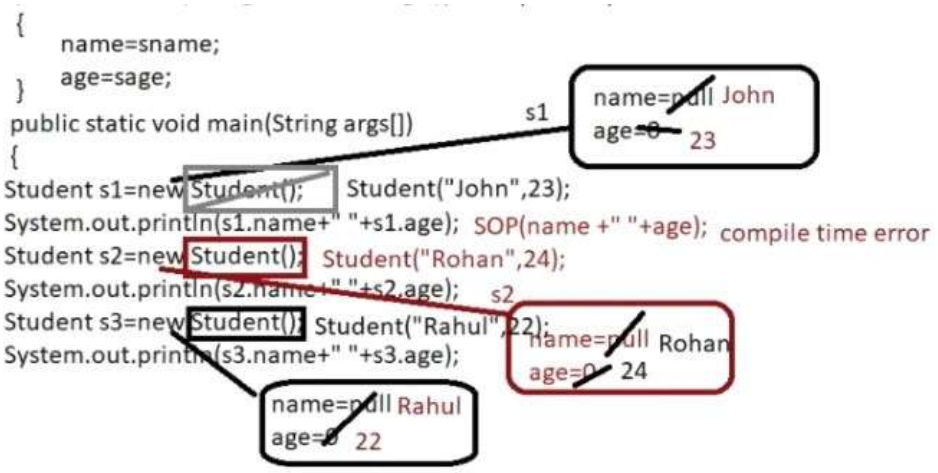
- One of the drawbacks of no argument constructor is, it provides same value for every object.
- Therefore to overcome this drawback, we go for parameterized constructor.

Parameterised Constructor

- If a constructor contains any arguments, it is called as parameterised constructor.
- While creating Object, we have to pass particular argument as constructor.

Examples:

```
class Student  //Ex-1
{
  String name;
  int age;
  //parameterized constructor
  public Student(String sname,int sage)
  {
      name=sname;
      age=sage;
  }
  public static void main(String args[])
   {
      Student s1=new Student("John",22);
      System.out.println(s1.name+" "+s1.age);
      Student s2=new Student("Rohan",24);
      System.out.println(s2.name+" "+s2.age);
      Student s3=new Student("Rahul",24);
      System.out.println(s3.name+" "+s3.age);
  }}
```



Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Student
John 22
Rohan 24
Rahul 24
```

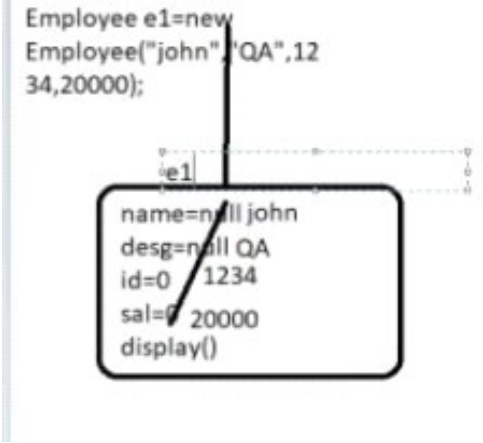
```
class Dog   //Ex-2
{
  String name;
  String color;
  public Dog(String dname,String dcolor)
  {
    name=dname;
    color=dcolor;
  }
public static void main (String args[])
{
 Dog f=new Dog("Tommy","Black");
 System.out.println("First dog name is :"+f.name+"Color is :"+f.color);
 Dog f1=new Dog("Softy","White");
 System.out.println("First dog name is :"+f1.name+"Color is :"+f1.color);
 Dog f2=new Dog("fluffy","Ash");
 System.out.println("First dog name is :"+f2.name+"Color is :"+f2.color);
}}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Dog
First dog name is :TommyColor is :Black
First dog name is :SoftyColor is :White
First dog name is :fluffyColor is :Ash
```

```
class Emp   //Ex-3
{
  String name,desg;
  int id,sal;
  //constructor
  public Emp(String ename,String edesg,int eid,int esal)
   {
        name=ename;
     desg=edesg;
     id=eid;
     sal=esal;
    }
    //defining display()
    public void display()
    {
     System.out.println("Name :"+name);
     System.out.println("Designation :"+desg);
     System.out.println("Their ID :"+id);
     System.out.println("Sal per month is :"+sal);
    }
public static void main(String args[])
```

```
{
 Emp e1=new Emp("John","QA",1234,25000);
 e1.display();
 Emp e2=new Emp("Rahul","QA",1235,27000);
 e2.display();
}}
```



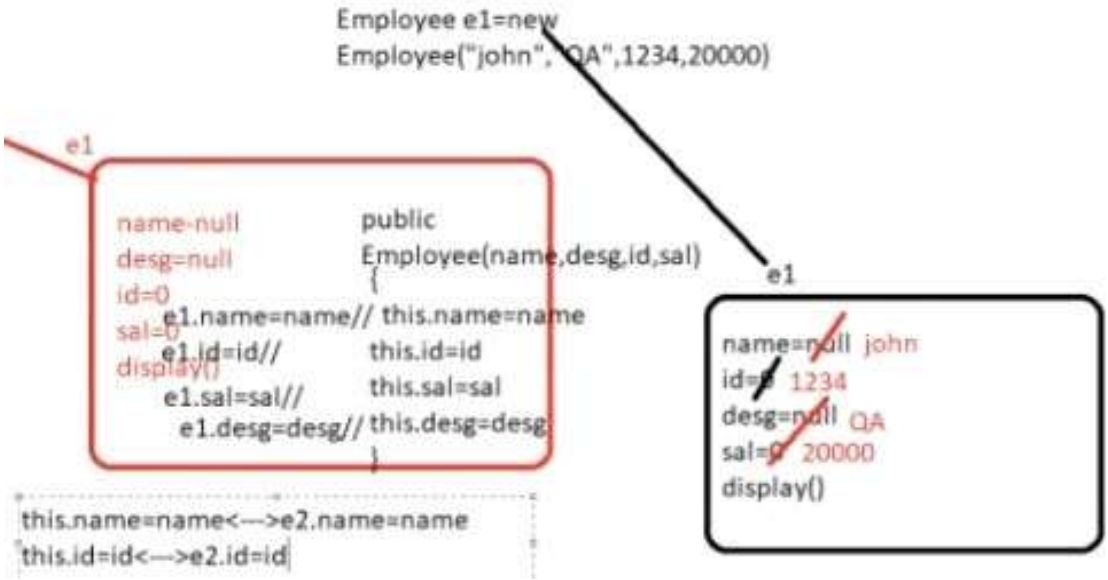
Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Emp
Name :John
Designation :QA
Their ID :1234
Sal per month is :25000
Name :Rahul
Designation :QA
Their ID :1235
Sal per month is :27000
```

this keyword

```
-------------
class Empp
{
  String name,desg;
  int id,sal;
  public Empp(String name,String desg,int id,int sal)
   {    //intialise
        this.name=name;
     this.desg=desg;
     this.id=id;
     this.sal=sal;
    }
    //defining display()
    public void display()
    {
     System.out.println("Name :"+name);
     System.out.println("Designation :"+desg);
     System.out.println("Their ID :"+id);
     System.out.println("Sal per month is :"+sal);
    }
```

```
public static void main(String args[])
{
 Empp e1=new Empp("John","QA",1234,25000);
 e1.display();
 Empp e2=new Empp("Rahul","QA",1235,27000);
 e2.display();
 Empp e3=new Empp("Ramesh","QA",1236,26000);
 e3.display();
 Empp e4=new Empp("Mahesh","QA",1237,24000);
 e4.display();
}}
```



Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Empp
Name :John
Designation :QA
Their ID :1234
Sal per month is :25000
Name :Rahul
Designation :QA
Their ID :1235
Sal per month is :27000
Name :Ramesh
Designation :QA
Their ID :1236
Sal per month is :26000
Name :Mahesh
Designation :QA
Their ID :1237
Sal per month is :24000
```

19-08-2020

Constructor Overloading

- The process of developing multiple CONSTRUCTOR with same constructor name but different argument list is called as constructor Overloading.

Rules For Arguments List:-
---
1. Between the constructors number of arguments should be different.
2. Between the constructors type(Datatype) of arguments should be different.
3. Between the constructors position or sequence of arguments should be different.

Programs

```
class Car   //Ex-1
{
String name,color,model;
double price;
int capacity;
public Car(String name,String color)
{
this.name=name;
this.color=color;
}
public Car(String name,String color,double price)
{
this.name=name;
this.color=color;
this.price=price;
}
public Car(String name,String color,double price,int capacity,String
model)
{
this.name=name;
this.color=color;
this.price=price;
this.capacity=capacity;
this.model=model;
}
public static void main (String args[])
{
Car c1=new Car("Audi","Black");
System.out.println("Car :"+c1.name+" Color :"+c1.color);
Car c2=new Car("Benz","Black",6000000.25);
System.out.println("Car :"+c2.name+" Color :"+c2.color+"
Price :"+c2.price);
```

```
Car c3=new Car("Jeep","Red",7800000.25,2,"A-class");
System.out.println("Car :"+c3.name+" Color :"+c3.color+"
Price :"+c3.price+" Capacity :"+c3.capacity+" Model :"+c3.model);
}}
```
Output:
```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Car
Car :Audi Color :Black
Car :Benz Color :Black Price :6000000.25
Car :Jeep Color :Red Price :7800000.25 Capacity :2 Model :A-class
```

```
class Book  //Ex-2
{
String name,author;
int price,pages;
public Book(String name,int price)
{
this.name=name;
this.price=price;
}
public Book(String name,String author,int price)
{
this.name=name;
this.author=author;
this.price=price;
}
public Book(String name,String author,int price,int pages)
{
this.name=name;
this.author=author;
this.price=price;
this.pages=pages;
}
public void display1()
{
System.out.println("Name :"+name+" Price :"+price);
}
public void display2()
{
System.out.println("Name :"+name+" Author :"+author+" Price :"+price);
}
public void display3()
{
System.out.println("Name :"+name+" Author :"+author+" Price :"+price+"
Pages :"+pages);
}
public static void main(String args[])
{
Book b1=new Book("Machine Design",475);
b1.display1();
Book b2=new Book("TOM","SS.Ratan",550);
b2.display2();
```

```
Book b3=new Book("Fluid Mechanics","R.K.Bansal",625,1025);
b3.display3();
}}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Book
Name :Machine Design Price :475
Name :TOM Author :SS.Ratan Price :550
Name :Fluid Mechanics Author :R.K.Bansal Price :625 Pages :1025
```

- The main purpose of constructor overloading is to create an object in multiple ways.

For ex1: Manufacturing of shirt
-half sleeves
-full sleeves
-without sleeves

For ex2: Lock
-Keylock
-passwordlock
-sensorlock

Constructor Chaining

- The process of calling one Constructor from another Constructor is called as Constructor chaining.

- Constructor chaining can be achieved in two ways-
  1. Call to this---------->represented as this()
  2. Call to Super--------->represented as super()

Call to this - this()

- The process of calling one constructor from another constructor of same class is called as call to this.
- The main Rule for this() is, Call to this must be the First statement of a constructor.

Programs

```
class Add           //Ex-1
{
     public Add()
     {
     this(100,200);//call another const of same class which has 2
integer args
      System.out.println("Default add constructor");
       }
     public Add(int i,int j)
     {
       this(223.5,10);//call another const of same class which has 2
args 1st is double 2nd is int
```

```
        System.out.println("Result of int+int is : "+(i+j));
      }
      public Add(double d,int i)
      {
        System.out.println("Result of double+int is : "+(d+i));
      }
      public static void main(String args[])
      {
       Add a1=new Add();
      }
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java  Add
Result of double+int is : 233.5
Result of int+int is : 300
Default add constructor
```

```
class Add1          //Ex-2
{
      public Add1()
      {
       this(122.3,200);//call another const of same class which has 2
integer args
       System.out.println("Default add constructor");
        }
      public Add1(int i,int j)
      {
        this();//call another const of same class which has 2 args 1st
is double 2nd is int
        System.out.println("Result of int+int is : "+(i+j));
      }
      public Add1(double d,int i)
      {
        System.out.println("Result of double+int is : "+(d+i));
      }
      public static void main(String args[])
      {
       Add1 a1=new Add1(100,200);
      }
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Add1
Result of double+int is : 322.3
Default add constructor
Result of int+int is : 300
```

20-08-2020

Q) Can i call more than one constructor from constructor ?

A. No, we cannot call more than one constructor from constructor because if we call we have to write call to this in 2nd statement which is violation of rule.

```
    public Add()
    {
     this(100,200);//call another const of same class which has 2 integer           args
     this(223.4,50);//CTE because call to this must be first statement
     System.out.println("Default add constructor");
    }
```

Q) Can i call same construcor from that particular constructor ?

A. No, we can't if we do, we will get CTE as "recursive constructor call"

```
   public Add()
   {
     this();
     System.out.println("Default add constructor");//1st exec stmt
   }
```

Call to super - super() :- we will study this after Inhertitance topic.

Differences between METHOD AND CONSTRUCTOR :

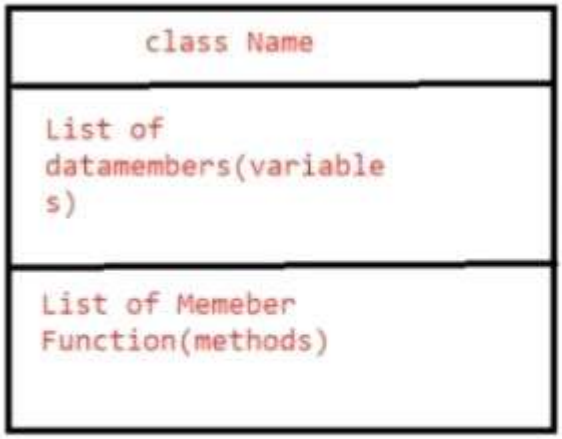
METHODS	CONSTRUCTORS
1. Since, we can't write business logic directly in class, we use methods.	1. Constructors are mainly used for intialising Non static variables.

2. Syntax:
AccessModifier NonAccessModifier reurntype methodname(args/no args)
{
}
 	
	2. Syntax:
	AccessModifier Constrctorname(args/no args)
	{
	}

3. Method name can be anything(as per user).	3. Constructor name must be same as classname.
4. To execute a method, we have to call it static------->Directly Non static--->Object	4. Constructor will get called automatically, whenever we created an Object.
5. We can call one method from another method directly or through object.	5. We can call one constructor from another constructor through consructor chaining.
6. Only Non static methods can be inherited.	6. Constructors cannot be inherited.
7. Only Non static methods can be Overridden.	7. Constructors cannot be Overridden.

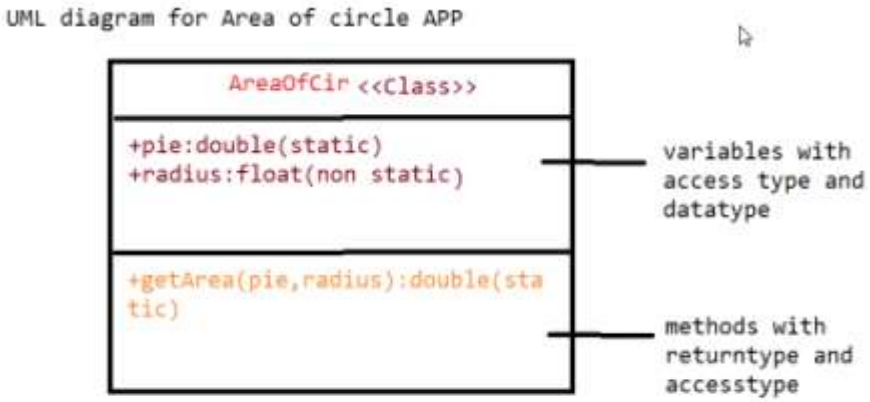
UML Diagrams or Class Diagrams

- It is pictorial representation of an Application.
- In Realtime a developer will always document the project by using UML diagram only.



Structure of UML diagram

Ex: UML diagram for Area of Circle APP



UnifiedModeling Lang(UML) of AREAOFCIR APP

21-08-2020

OOPS (Module-3)

- class and Object
- Inheritance(IS-A)
- Has-A
- call to super
- method overriding
- super keyword
- Abstraction
- Interface
- Encapsulation
- Polymorphism
- Type and Object Casting

Object Oriented Programming System: It is a programming methodology, which fulfils the project requirement.

- It consists of 4 pillars-
    1. Inheritance
    2. Abstraction
    3. Encapsulation
    4. Polymorphism

class and Object :-

A class is a logic Entity (Because it represents logic of APP).
An Object is physical Entity (Because it contains memory).
A class is a blue print or template using which we can develop code of APP.
An Object is an Entity which gets created using class and it represents the state and behaviours.
Where state indicates variables(datamembers) and behaviourindicates methods(member function).

```
Ex: class Student
    {
       //develop code/logic for Student APP
    }
       Student-Object
```

states	Behaviours
names	studying()
age	writingexams()
height	playing()
weight	etc
marks	
contact	
etc	

Differences between class and Object :

class	Object
A class is logical entity.	An object is physical entity.
class represents logic.	Object represents state and behaviour.
class can be created using class keyword.	Object can be created using new keyword.
When class is created, memory will not be allocated.	When object gets created memory will be allocated(Heap).
For each module, we can create one class.	For one class, we can create multiple Objects.

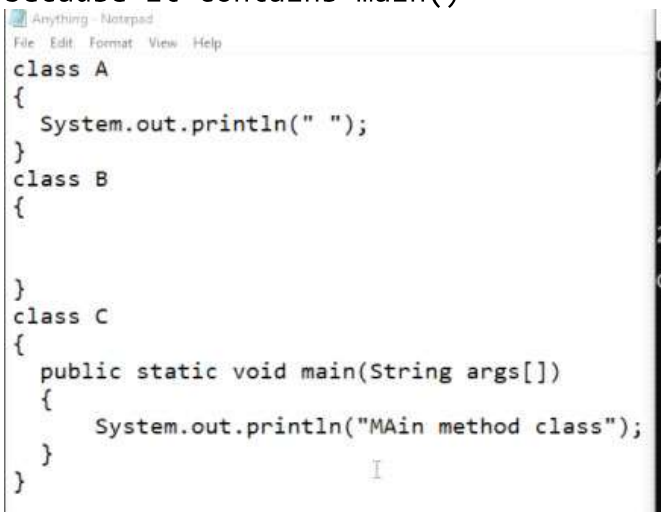
Points to remember

- In a program, we can have multiple classes. When we compile, a program depends on no of classes we define those many .class files will be generated.

```
ex:  class A
     class B
     class User
     {  main()  }
```

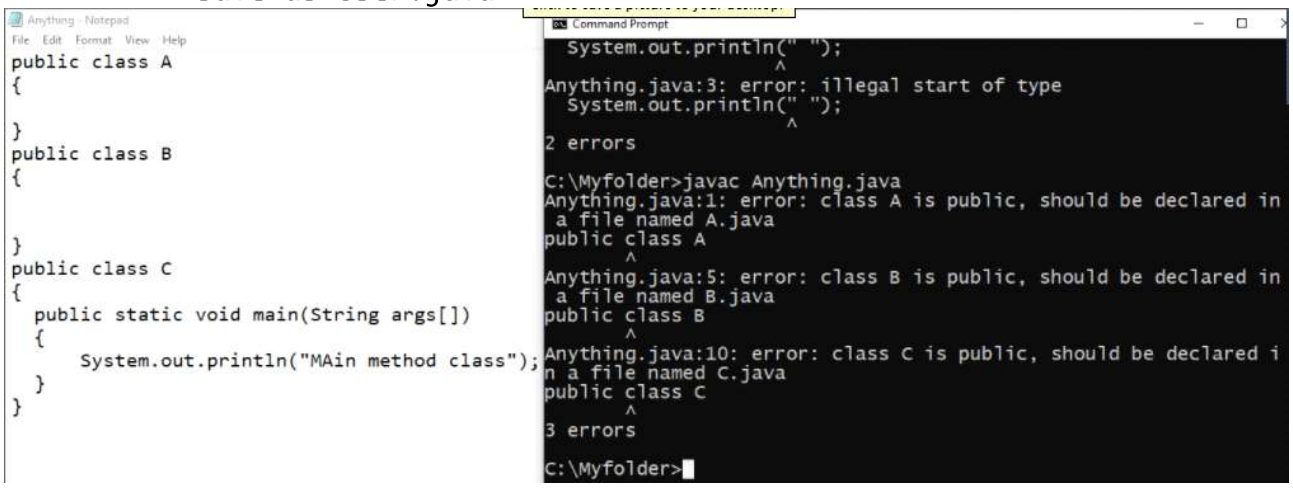
- After compilation: A.class
  B.class
  User.class

- As per convention, above program we should save as User.java because it contains main()



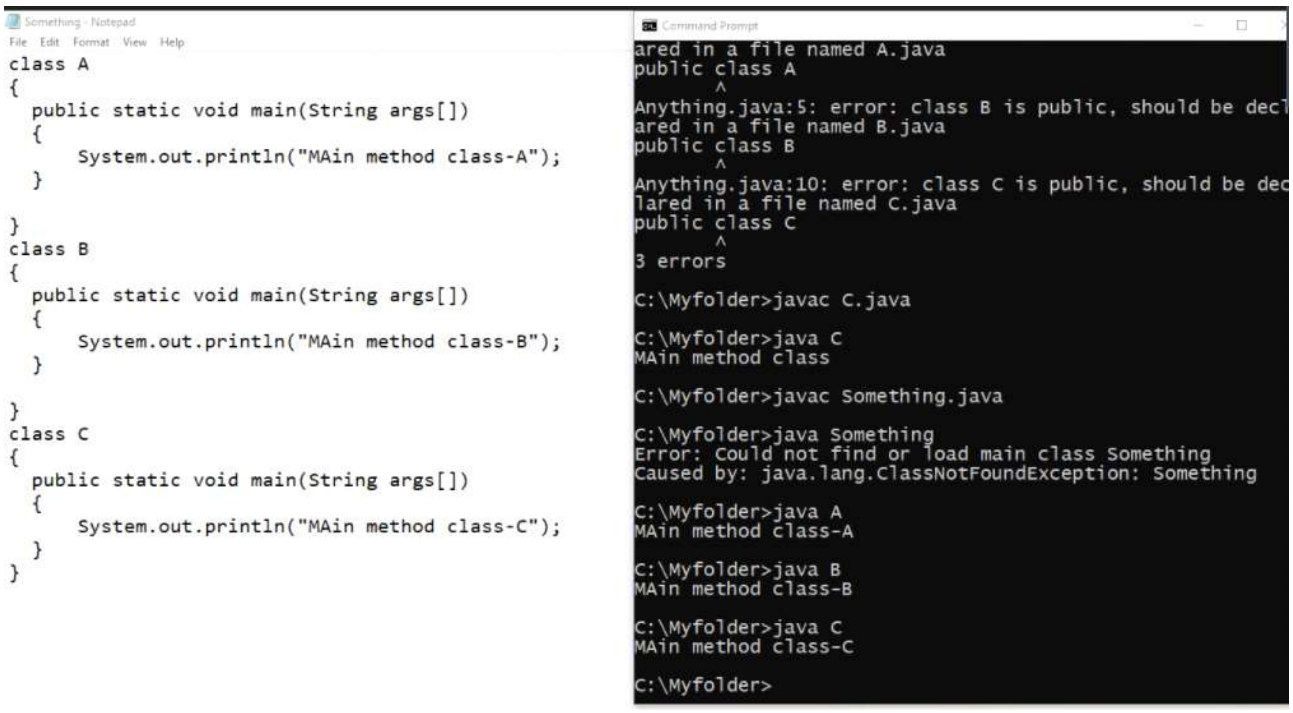
- When we have multiple class in single program. we can keep only one class as public and we hould keep main method class as public and save filename with that particular class name.

```
  class A
  class B
public class User
   {
     main()
   }
save as User.java
```



- When we have multiple class in single program and all of them contains main methods and we should keep only one class as public and save filename with that particular class name and execute individually.

```
class A
 {
   main()
 }
class B
 {
   main()
 }
public class C
 {
   main()
 }
save as C.java
execute as java A
           java B
           java C
```



```
Something - Notepad
File  Edit  Format  View  Help
class A
{
  public static void main(String args[])
  {
      System.out.println("MAin method class-A");
  }

}
class B
{
  public static void main(String args[])
  {
      System.out.println("MAin method class-B");
  }

}
class C
{
  public static void main(String args[])
  {
      System.out.println("MAin method class-C");
  }
}
```

```
Command Prompt
ared in a file named A.java
public class A
       ^
Anything.java:5: error: class B is public, should be decl
ared in a file named B.java
public class B
       ^
Anything.java:10: error: class C is public, should be dec
lared in a file named C.java
public class C
       ^
3 errors

C:\Myfolder>javac C.java

C:\Myfolder>java C
MAin method class

C:\Myfolder>javac Something.java

C:\Myfolder>java Something
Error: Could not find or load main class Something
Caused by: java.lang.ClassNotFoundException: Something

C:\Myfolder>java A
MAin method class-A

C:\Myfolder>java B
MAin method class-B

C:\Myfolder>java C
MAin method class-C

C:\Myfolder>
```

- Business logic class ----> which does not have main()
- user logic class ----> which contains main()

Inheritance (or) IS-A relationship

- It is the First Pillar of OOPS.
- Inherit means acquire, posses, access, take, etc.
- One class acquiring the properties of another class is called as Inheritance.

  (OR)
- One class is accessing the properties of another class is called as Inheritance.
- Here properties is defined as methods and variables.

extends keyword :

- extends is a keyword which indicates that we are creating a new class from an existing class.

```
class A   //superclass or parentclass or base class
{}
class B extends A  //B is called as subclass or childclass
 {}
```

Super class and Sub class :

- The class whose properties are acquired is called as super class or parent class or Base class.

- The class who is acquiring the properties is called as child class or Sub clas or derive class.

```
class Superclass
{

      }
class Subclass extends Superclass
{

   }
```

- Super class contains its own prperties.
- Sub class contains its own properties as well as properties of super class.

Without Inheritance	With Inheritance

Ex: class Father
{
gold()
land()
money()
car()
}
class Son
{
gold()
land()
money()
car()
girlfriend()
}
	Ex: class Father
	{
	gold()
	land()
	money()
	car()
	}
	class Son extends Father
	{
	girlfriend()
	}

Note:

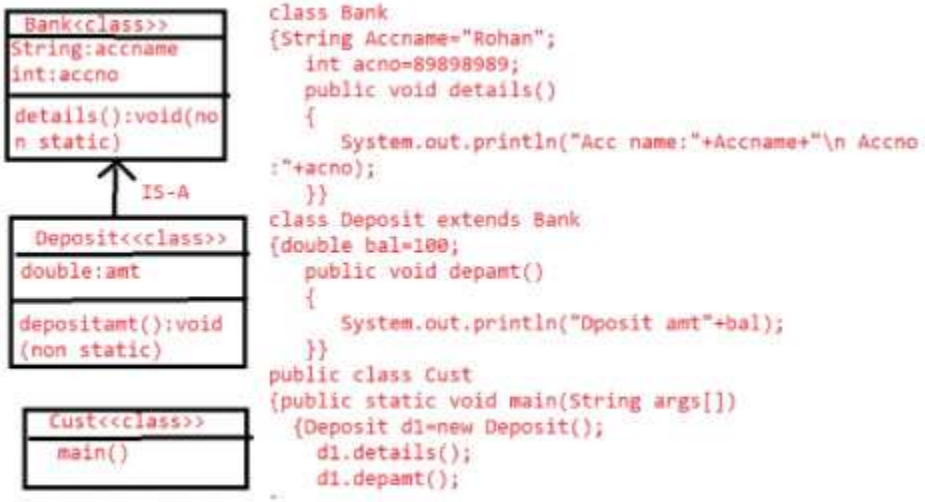
- ❖ Only Non static methods can be inherited, static methods cannot be inherited because they will get loaded only once in SAP.
- ❖ Constructors cannot be inherited because they are not a member of class,they are just used to initialise a Non static variable.
- ❖ If we create Object of super class, we can access only super class methods.
- ❖ If we create Object of sub class, we can access both super and sub class properties.

22-08-2020

Types Of Inheritance

1.Single Level
2.Multi Level
3.Hierarchical
4.Multiple
5.Hybrid

1. Single Level Inheritance
---
definition :- One super class and One sub class



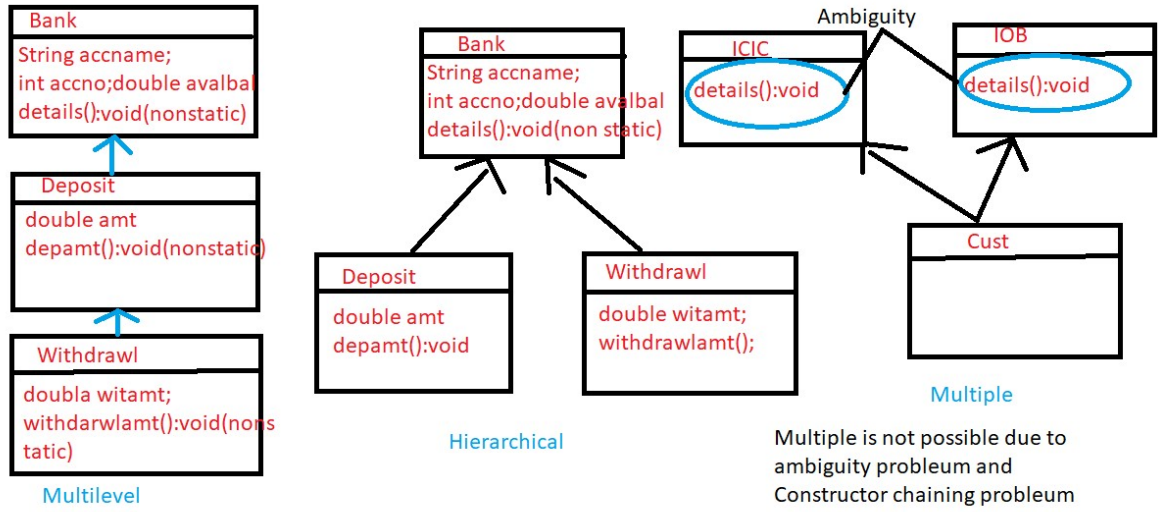

```
class Bank  //Ex-1
{
 String accname="John";
 int accno=245678;
 double avalbal=100;
public void details()
{
   System.out.println("Account Holder : "+accname+" accno:"+accno);
}}
class Deposit extends Bank
{
 double amt=4550.5;
public void depositamt()
{
  avalbal=avalbal+amt;
  System.out.println("Total balance after depositing Amt :"+avalbal);
}}
public class Cust
{
  public static void main(String args[])
  {
    Deposit d1=new Deposit();
```

```
    d1.details();
    d1.depositamt();
  }}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>javac Cust.java
C:\Users\Ravi Kiran\Qspiders Java\programs>java Cust
Account Holder : John accno:245678
Total balance after depositing Amt :4650.5
```

25-08-2020



2. Multi-level Inheritance

---

definition :- Two super classes and Two sub classes

```
class Bank                    //Ex-1
{
  int accno=23456;
  String accname="John";
  double availbal=2000;
  public void details()
  {
     System.out.println("Acc name : "+accname+" Acc no : "+accno);
   }
}
class Deposit extends Bank
{
    double amt=6000.5;
    public void deposit()
    {
      availbal = availbal + amt;
       System.out.println("Amount deposited : "+availbal);
```

```
    }
}
class Withdrawl extends Deposit
{
  double wamt=3000;
  public void witamt()
  {
      availbal = availbal - wamt;
      System.out.println("Withdrawl amount : "+availbal);
   }
}
public class Transaction
{
   public static void main(String args[])
   {
      Withdrawl w1 = new Withdrawl();
      w1.details();
      w1.deposit();
      w1.witamt();
   }
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>javac Transaction.java
C:\Users\Ravi Kiran\Qspiders Java\programs>java Transaction
Acc name : John Acc no : 23456
Amount deposited : 8000.5
Withdrawl amount : 5000.5
```

```
class Bank                    //Ex-2
{
   String Accname="Rohan";
   int acno=89898989;
   public void details()
   {
      System.out.println("Acc name:"+Accname+"\n Accno :"+acno);
   }
}
class Deposit extends Bank
{
   double bal=100;
   public void depamt()
   {
      System.out.println("Deposit amt : "+bal);
   }
}
class Withdrawl extends Deposit
{
   double amt=200;
   public void withamt()
  {
```

```
    if(amt>bal)
    {
       System.out.println("You cannot withdrawl-balance exceeds");
  }
  else{
    System.out.println("Collect Amount");
}
}}
public class Cust
{
  public static void main(String args[])
  {
    Withdrawl d1=new Withdrawl();
    d1.details();
    d1.depamt();
    d1.withamt();
  }
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>javac Cust.java
C:\Users\Ravi Kiran\Qspiders Java\programs>java Cust
Acc name:Rohan
 Accno :89898989
Deposit amt : 100.0
You cannot withdrawl-balance exceeds
```

3. Hierarchical Inheritance

---

definition :- One super class and Two sub classes

```
class Bank            //Ex-1
{
  int accno=23456;
  String accname="John";
  double availbal=2000;
  public void details()
  {
     System.out.println("Acc name : "+accname+" Acc no : "+accno);
   }
}
class Deposit extends Bank
{
    double amt=6000.5;
    public void deposit()
    {
      availbal = availbal + amt;
       System.out.println("Amount deposited : "+availbal);
    }
}
```

```
class Withdrawl extends Bank
{
  double wamt=3000;
  public void witamt()
  {
     availbal = availbal - wamt;
     System.out.println("Withdrawl amount : "+availbal);
   }
}
public class Transaction
{
   public static void main(String args[])
   {
     Deposit d1 = new Deposit();
     d1.details();
     d1.deposit();
     Withdrawl w1 = new Withdrawl();
     w1.details();
     w1.witamt();
   }
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>javac Transaction.java
C:\Users\Ravi Kiran\Qspiders Java\programs>java Transaction
Acc name : John Acc no : 23456
Amount deposited : 8000.5
Acc name : John Acc no : 23456
Withdrawl amount : -1000.0     // here now availbal=2000,wamt=3000
```

```
class Bank                    //Ex-2
{
   String Accname="Rohan";
   int acno=89898989;
   double avabal=100;
   public void details()
   {
      System.out.println("Acc name:"+Accname+"\n Accno :"+acno);
   }
}
class Deposit extends Bank
{
   double depamt=100;
   public void depamt()
   {
      System.out.println("Deposit amt : "+depamt);
      avabal=avabal+depamt;
      System.out.println("Total Ava bal is:"+avabal);
   }
}
class Withdrawl extends Bank
```

```
{
   double amt=100;
   public void withamt()
  {
    if(amt>avabal)//avabal=100
    {
       System.out.println("You cannot withdrawl-balance exceeds");
  }
  else{
    System.out.println("Collect Amount");
}
}}
public class Cust
{
  public static void main(String args[])
  {
    System.out.println("Deposit process");
    Deposit s1=new Deposit();
    s1.details();
    s1.depamt();
    System.out.println("Withdrawl Process");
    Withdrawl d1=new Withdrawl();
    d1.details();
    d1.withamt();
}
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>javac Cust.java
C:\Users\Ravi Kiran\Qspiders Java\programs>java Cust
Deposit process
Acc name:Rohan
 Accno :89898989
Deposit amt : 100.0
Total Ava bal is:200.0
Withdrawl Process
Acc name:Rohan
 Accno :89898989
Collect Amount
```

4. Multiple Inheritance

- Multiple inheritance is One class is inheriting two immediate super classes at the same time
- But in java, a class can extends only one class at a time
- So **Multiple inheritance is not possible through classes because** of -
  1. **Ambiguity problem**
  2. **Diamond problem -** since the structure/shape of class diagram is in diamond form it is also referred as Diamond problem.
  3. **Constructor chaining problem.**

- If one class extends two classes and in case, if both classes contains same method then while calling a method, JVM will get confuse which class method to call this problem is known as **Ambiguity problem**.

```
class ICIC    //Ex
{
public void details()
 {
     System.out.println("John"+"Acc 23456"+"Branch Hyd");
 }
}
class IOB
{
public void details()
 {
     System.out.println("John"+"Acc 34567"+"Branch Hyd");
 }
}
class Cust extends ICIC,IOB // CTE bcoz one class can't extends two
{                           classes at a time


 }
public class Details
{
    public static void main(String args[])
    {
      Cust c1 = new Cust();
      c1.details(); //JVM gets confused between details nethod in ICIC
                      and IOB classes, this problem is called as ambiguity
    }                 problem.
}
```

Construction chaining problem

Q> Can we inherit Constructors or not?

A.

- No we cannot inherit constructors because they are not member of a class(members of classs are methods and variables) and constructors are mainly used for intialistion of Non-static variable.
- Constructors cannot be inherited but they can be invoked by using call to super.

Call to super

- The process of calling one contructor from another constructor of different class is called as call to super.
- Call to super must be the first statement in constructor.

```
class A          //Ex-1: call to super explicit
{
   public A(int i)
   {
    System.out.println("A class default constructor");
   }
}

class B extends A
{
   public B()
   {
    super(100);//Explictly we added
    System.out.println("B class Default Constructor");
    }
}
public class Check
{
  public static void main(String args[])
  {
    B b1=new B();
  }
}
```

Output :

```
C:\Users\Ravi Kiran\Qspiders Java\programs>javac Check.java
C:\Users\Ravi Kiran\Qspiders Java\programs>java Check
A class default constructor
B class Default Constructor
```

```
class ICIC              //Ex-2
{
   String accname;
   public ICIC(String accname)
    {
      this.accname=accname;
         System.out.println(accname);
    }
}
class Bank extends ICIC
{
    public Bank()
    {
      super("John");
     }
}
public class Details
{
    public static void main(String args[])
    { Bank b1 = new Bank();
      }}
```

Output:

John

26-08-2020

- Call to super is Optional, if we don't have any arguments in super class constructor

```
class A          //Ex - call to super implicit
{
 public A()
 {
   System.out.println("A-class Constructor");
 }
}
class B extends A
{
 public B()
 {
   System.out.println("B-class Constructor");
 }
}
public class C
{
  public static void main(String args[])
  {
   B b1 = new B();
  }
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java C
A-class Constructor
B-class Constructor
```

Call to super with 3 classes

```
-----------------------------
class A
{
  public A(int i)
  {
    System.out.println("A class default constructor");
   }
}
class B extends A
{
  public B(int i)
  {
    super(100);
    System.out.println("B class default constructor");
   }
}
class C extends B
{
  public C()
  {
    super(55);
    System.out.println("C class constructor");
   }
}
public class Three
{
 public static void main(String args[])
 {
  C c1 = new C();
 }
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Three
A class default constructor
B class default constructor
C class constructor
```

Combination of call to this and call to super

```
-----------------------------------------------
class A
{
  public A()
  {
   this(100);
   System.out.println("A class default constructor");
   }
  public A(int i)
  {
   System.out.println("A class integer constructor");
  }
}
class B extends A
{
  public B()
  {
   this(99);
   System.out.println("B class default constructor");
   }
  public B(int i)
  {
   //super();//default call to super
   System.out.println("B class integer constructor");
  }
}
public class Combination
{
 public static void main(String args[])
 {
  B b1 = new B();
 }
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Combination
A class integer constructor
A class default constructor
B class integer constructor
B class default constructor
```

Constructor chaining problem

```
-----------------------------
class A
{
  public A()
  {

   }
}
class B
{
  public B()
  {

   }
}
class C extends A,B
{
  public C()
  {
    super();//Constructor chaining problem
   }
}
```

- In the above program, when call to super executed, JVM will get confuse which super class constructor to call because there are two super class. Hence, it gets confused this problem is known as Constructor chaining problem

- Therefore due to ambiguity problem and constructor chaining problem. Multiple inheritance is not possible through class but possible through Interface.

Hybrid Inheritance

```
---------------------
```

- It is a combination of Multiple and Hierarchical Inheritance since Multiple is not possible through classes. Hybrid is also not possible through classes.

Advantages of Inheritance :-

1. Reusability of code
2. Avoid duplicacy of code

Note :

To every class there is a default super class present, whether we write it or do not write it. i.e, to all predefine and userdefine classes of java there is a default super class called as Object class.

What we can see	What it is actually

class A
{
 	
}
	class A extends Object
	{
	
	}

- Object class contains 9 methods and these methods are used in multiple classes of java. So, instead of defining it seperately in all classes they have defined it in object class and make it as super class. So that without defining it again and again other class can simply use it.

Note:

```
class A extends Object
{

}
class B extends A
{

}

//Here B need not to extends Object becasue Object class is grandparent class to B.
```

Note:
---
1. We can inherit public,protected and default methods.
2. We cannot inherit private method because they are accessible only within class.
3. We can inherit non static, final and abstract methods.
4. We cannot inherit static methds because they will be loaded only once in SPA.
5. We cannot inherit constructors because they are mainly used for initialisation and they are not member of class.

HAS-A relation ship

- One class containing the reference of another class is called as HAS-A relationship.

```
class Engine     //Ex-1
{
  //properties of engine
}
class Car
{
  Engine e = new Engine();//Car has-a reference of Engine
}

class Ram     //Ex-2
{
  //properties of ram
}
class Mobile
{
  Ram r = new Ram();//Mobile has-a reference of Ram
}
```

Real Time example

For Bank APP

```
class Loans
{
   //50 methods
}
class HomeLoan
{
   //50 methods
}
class CarLoan
{
   //50 methods
}
class GoldLoan
{
   //50 methods
}
```

```
class MortLoan
{
   //50 methods
}
```

Using Inheritance

```
class Loan
{
   //All common methods
}
class TypesOfLoans extends Loan
{

}
```

Interview question: Tell me in ur Project where you used inheritance ??

Develop the Bank App (project)

- It should contains all details -
  - Account holder name
  - Account number
  - Branch name
  - IFSC code
  - Available balance
- We should able to deposit amount, after depositing available balnce is updated.
- We should able to withdrawl amount, after withdrawing amount, available balnce should be updated.
- We shold be able to change pin whenver we want like validate the old pin and provide new pin for ATM.
- We should able to generate a small mini statement for Available balnce and Last transaction.

27-08-2020

Method Overriding

During Inheritance subclass has complete previlege to change the (method)implementation of super class, this process is known as Method Overriding.

```
class Parents          //Ex-1
{
  public void car()//Overridden method
  {
    System.out.println("Blue color");
  }
  public void carname()
  {
    System.out.println("Audi");
  }
}
class Son extends Parents
{
  public void car()//Overriding method
  {
    System.out.println("Black color");
  }
}
class Daughter extends Parents
{
  public void car()
  {
    System.out.println("Pink color");
  }
  public void carname()
  {
    System.out.println("Nano");
  }
}
public class Driver
{
  public static void main(String args[])
  {
      Parents p1 = new Parents();
      p1.car();
      p1.carname();
```

```
      Son s1 = new Son();
      s1.car();
      s1.carname();
      Daughter d1 = new Daughter();
      d1.car();
      d1.carname();
  }
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Driver
Blue color
Audi
Black color
Audi
Pink color
Nano
```

```
class Parents          //Ex-2
{
 public void gold()
 {
  System.out.println("5000kg");
 }
 public void marriage()//Overridden method
 {
  System.out.println("Rani");
 }
}
class Son extends Parents
{
 public void marriage()//Overriding method
 {
  System.out.println("Fruity");
 }
}
public class Society
{
 public static void main(String args[])
 {
  Parents p = new Parents();
  p.marriage();
  p.gold();
  Son s = new Son();
  s.marriage();
  s.gold();
  }
```

```
}
```
Output:
```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Society
Rani
5000kg
Fruity
5000kg
```

```
class India          //Ex-3
{
  public void lockdown()//Overridden method
  {
    System.out.println("27-august-2020");//Method Implementation
  }
}
class Telangana extends India
{
  public void lockdown()//Overriding method
  {
    System.out.println("31-august-2020");
  }
}
class People
{
  public static void main(String args[])
  {
    India a = new India();
    a.lockdown();//call lockdown() of India class---->27-august-2020
    Telangana t = new Telangana();
    t.lockdown();//call lockdown() of India class---->31-august-2020
   }
}
```
Output:
```
C:\Users\Ravi Kiran\Qspiders Java\programs>java People
27-august-2020
31-august-2020
```

Q. Why do we go for Overriding ??

A. When subclass want the properties of super class but does not want the implementation of it. In such case, sub class can change implementation by means of Overriding.

Ex: Caller's tune

Normaldays---->tone()---->NormalRingtone

DuringMarch--->tone()---->CoronaRingtone

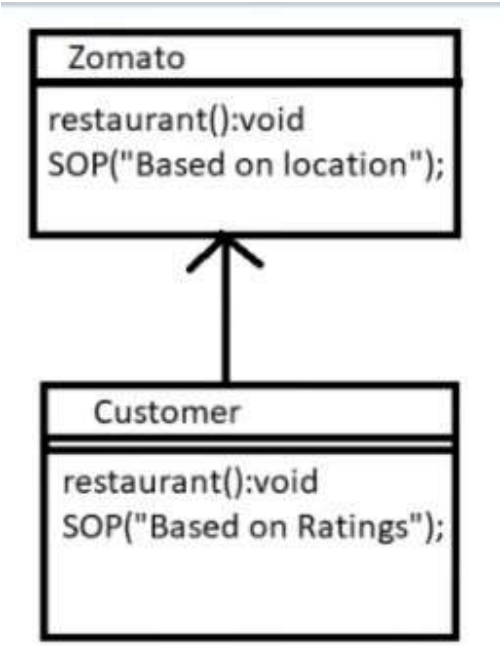
Rules For Overriding

---

1. Inheritance is compulsory.
2. method signature must be same (methodname and arguments)as super class in sub class.
3. method header must be same (accessmodifier, nonaccessmodifier, returntype)
4. Overridden method should not be final.

Assignment

```
class Zomato          //Q-1
{
 public void restaurant()
 {
  System.out.println("Based on location");
 }
}
class Customer extends Zomato
{
 public void restaurant()
 {
  System.out.println("Based on Ratings");
 }
}
public class Food
{
 public static void main(String args[])
 {
  Zomato z = new Zomato();
  z.restaurant();
  Customer c = new Customer();
  c.restaurant();
  }
}
```

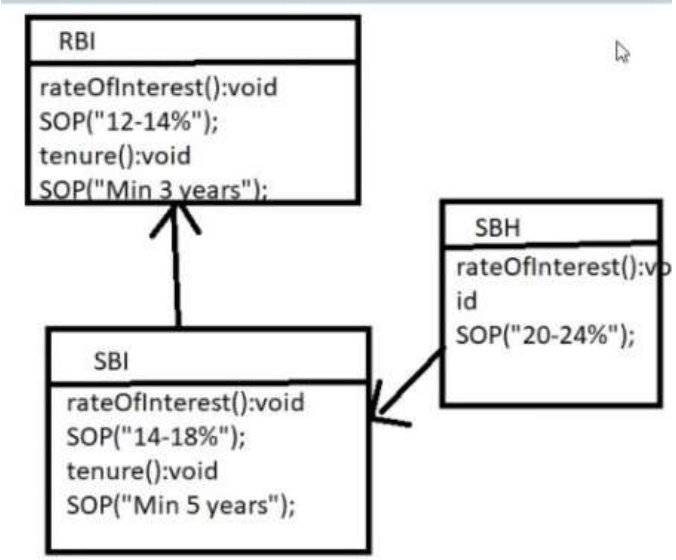


Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Food
Based on location
Based on Ratings
```

```
class Rbi       //Q-2
{
 public void rateOfInterest()
 {
  System.out.println("12.14 %");
 }
```

```
 public void tenure()
 {
  System.out.println("Minimum 3 years");
 }
}
class Sbi extends Rbi
{
 public void rateOfInterest()
 {
  System.out.println("14.18 %");
 }
 public void tenure()
 {
  System.out.println("Minimum 5 years");
 }
}
class Sbh extends Sbi
{
 public void rateOfInterest()
 {
  System.out.println("20.24 %");
 }
}
public class Bank
{
 public static void main(String args[])
 {
  Rbi r = new Rbi();
  r.rateOfInterest();
  r.tenure();
  Sbi s1 = new Sbi();
  s1.rateOfInterest();
  s1.tenure();
  Sbh s2 = new Sbh();
  s2.rateOfInterest();
  s2.tenure();
  }
}
```



Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Bank
12.14 %
Minimum 3 years
14.18 %
Minimum 5 years
20.24 %
Minimum 5 years
```

28-08-2020

Polymorphism

- it is a greek word.
- poly means many and morphism means forms.

One thing showing multiple behaviour is called as polymorphism.
(or)
One entity showing different behaviour is called as polymorphism.

Polytechnique where poly means many, techniques means technologies.

Polyclininc where poly means many, clinic means treatments.

Types Of Polymorphism :-

---

1. Compile time polymorphism :

During compile time one thing showing multiple behaviour is called as compile time polymorphism.
ex: method overloading, where during compilation, compiler will decide which behaviour to be implemented so here methodname is same but depending on type of args it shows different behaviour.

```
run()
run(int i)
run(char ch)
run(String s,int i)
```

Technical definition :

a. The process of resolving call to overloaded method during compile time depending on type of arguments is called compile time polymorphism.
b. It is also called as static polymorphism or compile time binding.

2. Run time polymorphism :

During Run/Execution time one thing showing multiple behaviour is called as Run time polymorphism.
ex: method overriding , where there are multiple methods with same name, during execution time only JVM will come to know which method to be executed depending on type of object.

```
class A
{
public void run(){SOP("A class Method");}
}
class B extends A
{
public void run(){SOP("B class Method");}
}
class Main
{
 public static void main(String args[])
  {
   A a1 = new A();a1.run();//it calls class A run()
   B b1 = new B();b1.run();//it calls class B run()
  }}
```

Technical definition :

a. The process of resolving call to overridden method during run time depending on type of object we create is called run time polymorphism.
b. It is also called dynamic binding or dynamic polymorphism.

Method Binding

The process of connecting method signature with method implementation is called as Method Binding.

```
                        {
Ex:Public static void run()------------System.out.println("In run");
                        }
```

Method Binding is divided into two types :-

---

1. Compile time binding :

- The process of connecting method signature with method implementation during compile time is called as compile time binding.
- Since, binding is happening before execution it is called as Early binding.
- All static and final methods will get binded during compile time i.e,during compilation only method will come to know what is its implementation.

```
                        {
Ex:Public final void run()-------System.out.println("In run method");
                        }
```

2. Run time binding :

- The process of connecting method signature with method implementation during execution time is called as run time binding.
- Since binding is happening during execution it is called as Late binding.
- All Non static and abstract methods will get binded during run time i.e, until program will reason Non static and abstarct method can be overridden.

```
                    {
Ex:Public void run()-----------System.out.println("In run method");
                    }
```

Note :

- private methods can be accessible anywhere within class.

29-08-2020

Encapsulation

- The process of wrapping the data members(Global variables) and member function(methods) into a single unit(class) is called as Encapsulation.
- The main purpose of encapsulation is to achieve security of data.

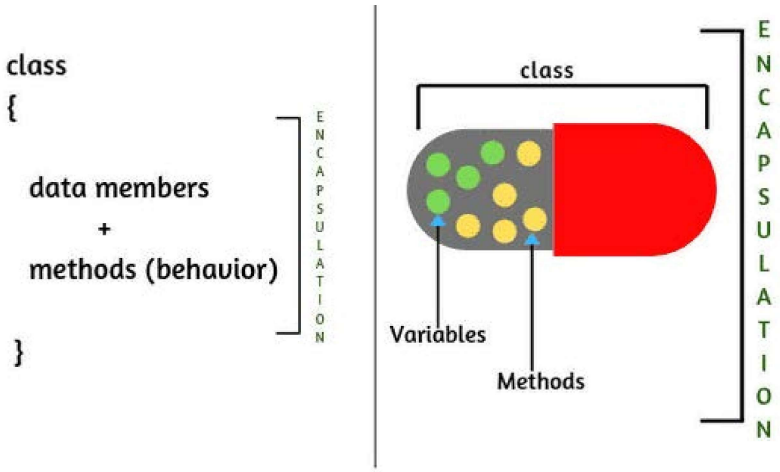


Fig: Encapsulation

What is the need of Encapsulation ?

```
public class demo
{
  public static void main(String args[])
  {
   App a1 = new App();
   a1.monthnum = 22;
   System.out.println("Month number is "+a1.monthnum);
  }
}
//Unencapsulated class
class App
{
   int monthnum;//datamember is not protected by developer or programmer
}
```

- We go for encapsulation to protect our data members from invalid user.
- For ex : if monthnum is not protected(Encapsulated) user may misuse the datamember by assigning invalid values(like >12 or <1)
- If data member is not protected(public), it is under user control i.e, user will decide what values to assign but those values can be valid or invalid.
- If data member is protected(private), it is under developers control to decide which values to be allowed.

Purpose

- Protecting the data members by keeping them private and accessing through some special methods is nothing but Encapsulation.

Rules for Encapsulation :

---

1. Decalre all the data members as private (if a datamember is private, it is not accessible outside of class).
2. define seperate setter and getter methods (it's not mandatory to define only setter and getter, we can define any user define methods).

```
public class UserCal   //ex
{
 public static void main(String args[])
  {
    Calendar c1 = new Calendar();
    c1.monthnum=22;
    System.out.println("Month number is "+c1.monthnum);
  }
}
//Encapsulated class
class Calendar
{
 private int monthnum;//data member is protected by developer or programmer
}
Output:
C:\Users\Ravi Kiran\Qspiders Java\programs>javac UserCal.java
UserCal.java:6: error: monthnum has private access in Calendar
    c1.monthnum=22;
      ^
UserCal.java:7: error: monthnum has private access in Calendar
    System.out.println("Month number is "+c1.monthnum);
                                            ^
2 errors
```

```
class Login                  //ex-using Encapsulation
{
 private String username;
 private String pwd;
 public void setusername(String username)
 {
  this.username=username;
 }
 public String getusername()
 {
  if(username=="John")
  {
   return "Username is correct, Please Enter Password" ;
  }
  else
   {
    return "Username is Incorrect" ;
   }
 }
```

```
 public void setpwd(String pwd)
 {
  this.pwd=pwd;
 }
 public String getpwd()
 {
  if(pwd=="John@14141")
  {
   return "Please go ahead" ;
  }
  else
   {
     return"Entered password is invalid" ;
   }
 }
}
public class PageUser
{
 public static void main(String args[])
 {
  Login l1 = new Login();
  l1.setusername("John");
  System.out.println(l1.getusername());
  l1.setpwd("Something@143");
  System.out.println(l1.getpwd());
 }
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java PageUser
Username is correct, Please Enter Password
Entered password is invalid
```

01-09-2020

JAVA BEAN CLASS :

A class is called as java Bean class, if all data members are private and there is seperate setter and getter method for each data member.

setter method
---
1. it is public in nature.
2. setter method does not have specific return type.
3. setter method contains arguments same as data member.
4. name of method is set followed by data member name.

getter method
---
1. it is public in nature.
2. return type must be same as that of data member.
3. name of method is get followed by data member name.
4. getter method does not have arguments.

Note :
- For every data member we have to define seperate setter and getter method.

```
import java.util.Scanner;   //EX
class EPF
{
   private long eidPF;
   private String pwd;

  //setter method---eidPF
  public void seteidPF(long eidPF)
  {
    this.eidPF=eidPF;
  }
 //getter method---eidPF
 public long geteidPF()
 {
   if(eidPF==101123458l)
   {
     System.out.println("Processing your EID please wait.............");
     return eidPF;
   }
   else
```

```
   {
     System.out.println("Incorrect EID");
     return 0;
   }
 }

  //setter method---pwd
  public void setpwd(String pwd)
  {
    this.pwd=pwd;
  }
 //getter method---pwd
 public String getpwd()
 {
   if(pwd=="Ravi123")
   {
     System.out.println("Processing your password please
wait.............");
     return "Password is Correct";
   }
   else
   {
     System.out.println("Incorrect Password");
     return "Please check your Passsword again";
   }
 }
}
public class EmployeeEP
{
  public static void main(String args[])
  {
   Scanner sc = new Scanner(System.in);
   EPF e1 = new EPF();
   System.out.println("Please Enter your EmployeePF-EID");
   long eid = sc.nextLong();
   e1.seteidPF(eid);
   System.out.println(e1.geteidPF());
   System.out.println("Please Enter your Password");
   String pwd = sc.nextLine();
   e1.setpwd(pwd);
   System.out.println(e1.getpwd());
  }
}
```

Output:

Note :

Q. System and Scanner are predefined classes but why we write import statement for Scanner class only ??
A. classes present in java.util package can only be used by writing import statement, whereas classes present in java.lang package they can be used directly.

- Scanner class is present in java.util package ---> Scanner class, Collection class, etc.
- System class is present in java.lang package ----> System class,String class,Exception class,Correct class,Object class, etc.

Examples of Encapsulation:
---
1. protecting bank details from an invalid user to access is an example of encapsulation.
2. keeping a password for mobile to protect data is an example of encapsulation.
3. protecting an online documents by keeping pasword is an example of encapsulation.
4. securing house by keeping lock is an example of encapsulation.

Fully Encapsulated class

- if all data members of a class are private and there is separate methods to access those data, such class is called as Fully Encapsulated class.

```
class A
{
  private int i;
  private float f;
  private String s;
  //methods to access them//
}
```

Partially Encapsulated class

- if any data members of a class is non private, such class is called as Partially Encapsulated class.

```
class A
{
```

```
  private String name;
  public int age;
}
```

No Encapsulated class

- if none of data members of a class is private, such class is called as No Encapsulated class.

```
class A
{
  public String name;
  public int age;
}
```

Advantages Of Encapsulated class :

- Securing the data
- We can make our class as write only by writing only setter method.
- We can make our class as read only by defining only getter method.

Dis-advantages Of Encapsulated class :

- length of the code increases.

Object Casting

- Converting one type of object into another type of object is called as Object Casting.
  They are of 2 types :-
  1.Upcastng
  2.Downcasting

Upcasting

- Converting sub class object type into super class object type is called as upcasting.

(or)

- create an object of sub class and store it into a reference of super class is called as upcasting.
- During up-casting only super class behaviour is visible, sub class behavior is hidden.

```
class Student          //Ex-1
{
 public void view()
 {
  System.out.println("Student details are :");
  System.out.println("Name:John P");
  System.out.println("MJCET");
  System.out.println("BTECH-Computer Science");
  System.out.println("Degree-58.3%\nINT-75%\nSSC-66%");
 }
}
class Admin extends Student
{
 public void edit()
 {
  System.out.println("Edit details of Student");
  System.out.println("Name:John P");
  System.out.println("MJCET");
  System.out.println("BTECH-Computer Science");
  System.out.println("Degree-60.9%\nINT-75%\nSSC-66%");
 }
}
public class App
{
 public static void main(String args[])
 {
  //upcasted object
   Student s1 = new Admin();     //here Student(left side)--->Compilation

   s1.view();                    //     Admin(right side)--->Execution
   //s1.edit();//CTE
 }
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java App
Student details are :
Name:John P
MJCET
BTECH-Computer Science
Degree-58.3%
INT-75%
SSC-66%
```

```
class Student          //Ex-2
{
 public void view()
 {
  System.out.println("Student details are :");
  System.out.println("Name:John P");
  System.out.println("MJCET");
  System.out.println("BTECH-Computer Science");
  System.out.println("Degree-58.3%\nINT-75%\nSSC-66%");
 }}
class Admin extends Student
{
 public void view()
 {
  System.out.println("Student details are :");
  System.out.println("Name:Ravi G");
  System.out.println("GITAM");
  System.out.println("BTECH-Mechanical");
  System.out.println("Degree-80.1%\nINT-94.7%\nSSC-86.8%");
 }
 public void edit()
 {
  System.out.println("Edit details of Student");
  System.out.println("Name:John P");
  System.out.println("MJCET");
  System.out.println("BTECH-Computer Science");
  System.out.println("Degree-60.9%\nINT-75%\nSSC-66%");
 }}
public class App
{
 public static void main(String args[])
 {
  //upcasted object
    Student s1 = new Admin();
    s1.view();
    //s1.edit();//CTE
 }}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java App
Student details are :
Name:Ravi G
GITAM
BTECH-Mechanical
Degree-80.1%
INT-94.7%
SSC-86.8%
```

Note:

- When Overriding and Upcasting happen, always overriding method will be executed because in upcasting, object type is sub class type.

```
class Police          //Ex-3
{
 public void lockdown()
 {
   System.out.println("stay at home");
 }
}
class Boyfriend extends Police
{
  public void meetingGirlfriend()
  {
   System.out.println("Want to see her only once");
  }
}
public class User
{
 public static void main(String args[])
 {
  Police p1 = new Boyfriend();//upcasted object
  p1.lockdown();//compiles successfully
  //p1.meetingGirlfriend();//CTE beacuse during upcasting subclass behavour is not visible.
 }
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java User
stay at home
```

- Since sub class contains the properties of super class, upcasting is possible directly i.e, super class can hold the reference of sub class. So, upcasting is implicit in nature.

02-09-20

```
class A          //Ex
{
  void m1()
  {
   System.out.println("m1 method of A-class");
  }
}
class B extends A
{
  void m1()
  {
   System.out.println("m1 method of B-class");
  }
  void m2()
  {
   System.out.println("m2 method of B-class");
  }
}
class U
{
 public static void main(String args[])
 {
   A a1 = new A();//super class object
   a1.m1();
   B b1 = new B();//sub class object
   b1.m1();
   A a2 = new B();//upcasting
   a2.m1();
  // a2.m2();//invalid CTE
   Object a3 = new A();//upcasted object--> class A extends Object class,
                                            Object class is default super class
  // a3.m1();//invalid
 }
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java U
m1 method of A-class
m1 method of B-class
m1 method of B-class
```

Downcasting

- Converting a super class object type into sub class object type is called as downcasting.

(or)

- Converting an upcasted object into normal form is called as downcasting.
- During downcasting both sub class and super class behaviour is visible.
- Since, super class does not contains property of sub class directly, downcasting is not possible explicitly we have to convert it.

Note: during upcasting some properties are hidden, during downcasting all properties are visible.

```
class Student          //Ex-1
{
 public void view()
 {
  System.out.println("Student details are :");
  System.out.println("Name:John P");
  System.out.println("MJCET");
  System.out.println("BTECH-Computer Science");
  System.out.println("Degree-58.3%\nINT-75%\nSSC-66%");
 }
}
class Admin extends Student
{
 public void edit()
 {
  System.out.println("Edit details of Student");
  System.out.println("Name:John P");
  System.out.println("MJCET");
  System.out.println("BTECH-Computer Science");
  System.out.println("Degree-60.9%\nINT-75%\nSSC-66%");
 }
}
public class App
{
 public static void main(String args[])
 {
  //upcasted object
   Student s1 = new Admin();
   s1.view();
   //s1.edit();//CTE
```

```
    //downcasted object
   Admin a1 = (Admin) s1; //Admin a1 = new Admin()
a1.edit();
    a1.view();
   }
  }
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java App
Student details are :
Name:John P
MJCET
BTECH-Computer Science
Degree-58.3%
INT-75%
SSC-66%
Edit details of Student
Name:John P
MJCET
BTECH-Computer Science
Degree-60.9%
INT-75%
SSC-66%
Student details are :
Name:John P
MJCET
BTECH-Computer Science
Degree-58.3%
INT-75%
SSC-66%
```

```
class Police          //Ex-2
{
 public void lockdown()
 {
   System.out.println("stay at home");
 }
}
class Boyfriend extends Police
{
 public void meetingGirlfriend()
 {
   System.out.println("Want to see her only once");
 }
}
public class User
{
```

```
 public static void main(String args[])
 {
  Police p1 = new Boyfriend();//upcasted object
  p1.lockdown();//compiles successfully
  //p1.meetingGirlfriend();//CTE
    //downcasting
  Boyfriend b1 = (Boyfriend) p1;  // Boyfriend b1 = new Boyfriend()
  b1.lockdown();
  b1.meetingGirlfriend();
  }
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java User
stay at home
stay at home
Want to see her only once
```

Type(datatype) Casting

- Converting one type of primitive datatype into another type of primitive datatype is called as type casting.

  (Or)

- Assigning one type of value into another type is called as Type casting.
- It is divided into two types -

  1.Widening 2.Narrowing

Widening

- Converting smaller primitive datatype into bigger primitive datatype is called as widening.
- Widening is also called as Implicit Casting.
- byte-->short-->int-->long-->float-->double
- Since, we are converting a smaller type into bigger type, there is no loss of data.

```
public class Widening      //Ex
{
 public static void main(String args[])
 {
   byte b=45;
   //widening
    short s=b;
    int i=s;
    long l=i;
```

```
    float f=l;
    double d=f;
    System.out.println(s);
    System.out.println(i);
    System.out.println(l);
    System.out.println(f);
    System.out.println(d);
}}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Widening
45
45
45
45.0
45.0
```

Narrowing

- Converting bigger primitive datatype into smaller primitive datatype is called as Narrowing.
- Narrowing is also called as Explicit Casting.
- byte<--short<--int<--long<--float<--double
- Since, we are converting a bigger type into smaller type, there is loss of data.

```
public class Narrowing          //Ex
{
 public static void main(String args[])
 {
   double d=163.45;
   float f=(float) d;
   long l=(long) f;
   int i=(int) l;
   short s=(short) i;
   System.out.println(d);
   System.out.println(f);
   System.out.println(l);
   System.out.println(i);
   System.out.println(s);
 }}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Narrowing
163.45
163.45
163
163
163
```

03-09-2020

Abstraction (Abstract + ion)

- It is the second pillar of OOPS concept.
- The process of hiding the internal implementation and showing the necessary data to the end user is called as Abstraction.
- In other words, we can say that we are not going to show method implementation, we will just provide method header and method signature.

Examples:

1. When we send an email, only a send button is visible but how it is being send is hidden.
2. Remote of TV and AC, where we can operate through buttons, but how it is being operated is hidden.
3. Driver knows only to apply brakes and accelerate, but how it is working is hidden.

In JAVA abstraction can be achieved in two ways :-

1. Using Abstract class
2. Using Interface

Abstract class

- abstract is a keyword, which indicates incompleteness.
- abstract class is a class which contains atleast one abstract method.

```
abstract class Vehicle          //Ex
 {
   abstract public void noOfWheels();
 }
```

Normal/Concrete/Complete method

- If a method have method signature as well as method body such method is called Complete or Concrete method.

```
Ex: public void run()
    {
     SOP("In run");
    }
    public void fly()
    {
```

```
    SOP("fly away");
    }
```

Abstract/Incomplete method

- If a method contains only method signature but not method implementation is called as abstaract or incomplete method.

```
Ex: public void run()
    public void fly()
```

- Abstract method has to be represented with abstract keyword and we have to add ; in the end of method declaration.

```
Ex: abstract public void run();
    abstract public void fly();
```

- If we didn't add abstract keyword or semicolon, we will get compile time error.

- It is mandatory to mention the abstract keyword for abstract methods and abstract classes.

```
abstract class Fruit        //Ex-1
  {
    abstract public void taste();
  }

asbtract class Loans        //Ex-2
  {
    abstract public void type();
    abstract public void rateOfInterest();
  }
```

Q. Can we create object of abstract class ?

A. No, we cannot create an object of abstract classs because it contains abstract methods and abstract method does not have any body to execute.

```
abstract class Vehicles          //Ex
{
 abstract public void noOfWheels();
}
class A
{
  Vehicles v1=new vehicles();//instantiation is not possible
  v1.noOfWheels();
}
```

Q. Can we Inherit abstract classes or not ?

A. Yes, if any child classes are there for abstract class that child class has to undergo with any of the one rule.

1. Complete all incomplete methods of abstract class by means of overriding.
2. Declare your class also as abstract class.

abstract vehicles.java //Ex-1, Using Rule 1

```
-------------------------
public abstract class Vehicles
{
 abstract public void price();
 abstract public void noOfWheels();
}
```

User.java

```
----------
public class User
{
  public static void main(Sring[] args)
  {
    //Vehicles v = new Vehicles();//instantiation is not possible
     RoyalEnfield r = new RoyalEnfield();
     r.noOfWheels();
     r.price();
  }
}
```

Business logic

```
---------------
class RoyalEnfield extends Vehicles
{     @Override
         public void price()
          {
            System.out.println("1.6 Lakhs");
          }
      @Override
           public void noOfWheels()
            {
            System.out.println("Two wheels");
            }
}
```

```
// abstract calculator.java--->overview is there //Ex-2, Using Rule 1
abstract class Calculator
{
  abstract public void add(int i,int j);
  abstract public void sub(int i,int j);
  abstract public void div(int i,int j);
  abstract public void mult(int i,int j);

}
//Operations.java class where entire logic present
class Operations extends Calculator
{
      public void add(int i,int j) {
            System.out.println(i+j);
      }
      public void sub(int i,int j) {
            System.out.println(i-j);
      }
      public void div(int i,int j) {

            System.out.println(i/j);
      }
      public void mult(int i,int j) {
            System.out.println(i*j);
      }
}
//User logic class
public class User {

      public static void main(String[] args)
      {
            Operations c1=new Operations();
             c1.add(100, 300);
             c1.sub(500, 300);
             c1.div(100, 30);
             c1.mult(100, 300);
      }

}
```

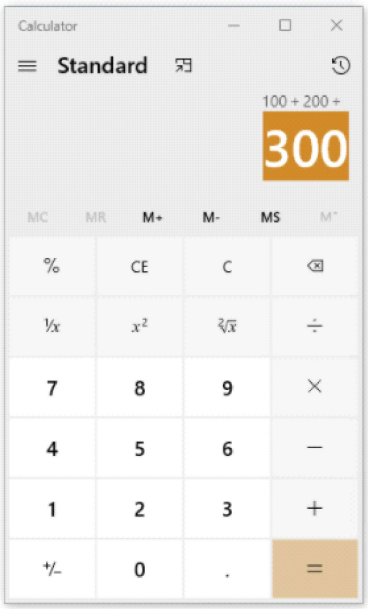
Output:

```
400
200
3
30000
```

```
abstract class Calculator
{
 abstract  public void add(int i,int j);
 abstract  public void sub(int i,int j);
 abstract  public void div(int i,int j);
 abstract  public void mult(int i,int j);

}


 public class User
{
            public static void main(String args[])
            {
    Operations c1=new Operations();
                        c1.add(100,300);
                        c1.sub(500,300);
                        c1.div(100,30);
                        c1.mult(100,300);
            }
}
```




```
File  Edit  Format  View  Help


//Programmer
class  Operations extends Calculator
{
            public void add(int i, int j) {
                        System.out.println(i+j);
            }
            public void sub(int i, int j) {
                        System.out.println(i-j);
            }
            public void div(int i, int j) {

                        System.out.println(i/j);
            }
            public void mult(int i, int j) {
                        System.out.println(i*j);
}
}
```

04-09-2020

```
// abstract calculator.java--->overview is there      //Ex-1, Using Rule 2
abstract class Calculator
{
  abstract public void add(int i,int j);
  abstract public void sub(int i,int j);
  abstract public void div(int i,int j);
  abstract public void mult(int i,int j);

}
//Operations.java class where entire logic present, since it contains abstract
method we kept class as abstract

abstract class Operations extends Calculator
{
      public void add(int i,int j) {
            System.out.println(i+j);
      }
      public void sub(int i,int j) {
            System.out.println(i-j);
      }
```

```
	public void div(int i,int j) {

		System.out.println(i/j);
	}
	public void mult(int i,int j) {
		System.out.println(i*j);
	}
	abstract public void displayResult();
}
class Result extends Operations
{
	public void displayResult()
	{
		add(100,200);
		sub(400,100);
		div(200,5);
		mult(30,4);
	}
}
//User logic class
public class User {

	public static void main(String[] args)
	{
		Result r1=new Result();
		r1.displayResult();
	}
}
```

Output:

```
300
300
40
120
```

```
//abstract Vehicles.java
public abstract class Vehicles
{
  abstract public void price();
  abstract public void noOfWheels();
  abstract public void models();
}
//RoyalEnfield is abstract because it contains abstract models()
abstract class RoyalEnfield extends Vehicles
{   //Override
	public void price(){
		System.out.println("1.6 Lakhs");
```

//Ex-2, Using Rule 2

```
            }
   //Override
         public void noOfWheels(){
                  System.out.println("Two wheels");
            }
}
class EnfieldModels extends RoyalEnfield
{     //Override
      public void models() {
            System.out.println("Classic 350");
            System.out.println("Inteceptor 600");
            System.out.println("Enfield Himalayan");
            System.out.println("Enfield classic 500");
      }
}
//User logic class
public class User {

      public static void main(String[] args)
      {
            //Vehicles v=new Vehicles();//CTE
            //RoyalEnfield r=new RoyalEnfield();//CTE
            EnfieldModels r=new EnfieldModels();
            r.noOfWheels();
            r.price();
            r.models();
      }
}
```

Output:

```
Two wheels
1.6 Lakhs
Classic 350
Inteceptor 600
Enfield Himalayan
Enfield classic 500
```

Note :

- Abstract class can have complete as well as incomplete methods.
- Abstract class can have static, non static and final variables.

Q. Can we define Constructor in abstract class ?

A. Yes, we can define a constructor in abstract class.

```
abstract class Student
{
      String sname;
```

```
	int sid;
	public Student(String sname,int sid)
	{
		this.sname=sname;
		this.sid=sid;
	}
	abstract public void details();
}
class Admin extends Student
{
	public Admin()
	{
		super("John",1234);
	}
	public void details()
	{
		System.out.println(sname+" "+sid);
	}
}
public class User1 {

	public static void main(String[] args) {
		Admin a1 = new Admin();
	  a1.details();
	}
}
```

Output:

```
John 1234
```

Q. Can we declare the class as abstract class even though it does not contains abstract methods ?

A. Yes, we can create the class as abstract even though it doesn't contains abstract method.

In 2 cases,

- Case 1: If all the members of class are static members then to access them, we do not have to create a object we can call directly or through class name.

```
	abstract class A
	{
	   static int i=100;
	   main()
	{
		SOP("In main);
		SOP(i);
		run();
	}
```

```
public static void run()
{
        SOP("In run");
}}
```

- Case 2: In general, we never create an object of super class we can declare super class as abstract class.

```
abstract class A
{
  }
class B extends A
{
  }
```

Q. Can we declare abstract class as final ?

A. No, we cannot declare a abstract class as final becasue if it is final it cannot be extended or inherited.

Q. Can we declare abstract method as final ?

A. No, we cannot declare an abstract method as final because if it is final it cannot be overriden and we must have to override abstract method to complete it.

Q. Can we declare abstract method as static ?

A. No, we cannot declare an abstract method as static because if it is static it cannot be overridde and we must have to override abstract method to complete it.

Q. Can we achieve Multiple Inheritance through abstract class ?

A. No, we cannot acheve Multiple Inheritance through abstract class because one class cannot extend more than one abstract class.

Conclusion:

- Since, Abstract class contains complete as well as incomplete methods. We can able to achieve partial abstraction i.e. 0 to 100% abstraction. Therefore, to achieve 100% abstraction, we go for interface.

07-09-2020

Interface

- An Interface is a type definition block(Block of codes) whose type convention is same as that of class.

```
interface Animal
{
  void eat();
  void makenoise();
  void color();
  void species();
}
```

- An interface has to be represented with interface keyword.
- Syntax:

```
interfacekeyword interfacename
{
  //body of interface
}
```

- By default all the methods of interface are public and abstract whether we write or don't write.

```
interface A                          interface A
{                                    {
 void run();                           public abstract void run();
 public void fly();    <-------------> public abstract void fly();
 abstract public void cry();           public abstract void cry();
}                                     }
```

- By default all the variables of interface are public, static and final whether you write or you don't write it.

```
interface A
{
 int i=9;          //public static final int i=9;
 static String s="java"; //public static final String s="java";
 final int k=88;      //public static final int k=88;
 public static final float fq=99.9f;
}
```

Q. Can we instantiate an interface ?
(or)
Can we create an object of interface ?

A. No, we cannot create an object of interface becasue all methods are by default abstract. But we can create a reference of interface.

Implements

- implements is the keyword, we will use if any class wants to form a relationship with interface.

```
syntax:     interface Car
            {
              void price();
            }
            class Audi implements Car
            {
                 }
```

- class------class------>extends
- interface------interface------>extends
- class------interface------>implements
- interface-----class----->Not possible

```
class A
{
  }
inerface B extends A-->invalid
interface B implements A-->invalid
```

- A class can extend only one class at a time
- An interface can extend multiple interfaces at a time
- A class can implement multiple interface at a time.

Example-1:

```
public interface Employer
{
    void joiningProcess();
    void pF();
    void allocateWork();
}
//User Logic
public class Emplyee {

      public static void main(String[] args) {
         //Employer e1=new Employer();
```

```
            Infosys i1=new Infosys();
            i1.joiningProcess();
            i1.pF();
            i1.allocateWork();
      }
}
//Infosys.java-Developer class
class Infosys implements Employer
{
      @Override
      public void joiningProcess() {
      System.out.println("1.Selection \n2.Document Verification\n3.Send
Offer Letter");
      }

      @Override
      public void pF() {
            System.out.println("As per the standard norm of EP");
      }

      @Override
      public void allocateWork() {
            System.out.println("As per the Vacancy Available");
      }
}
```

Output:

```
1.Selection
2.Document Verification
3.Send Offer Letter
As per the standard norm of EP
As per the Vacancy Available
```

- Whenever any class forms a relation with interface, it is the responsibility of that class to -
  Rule-1: Either complete all incomplete methods of interface.
  Rule-2: Declare the class as abstract.

Example-2:

```
//Employer.java
public interface Employer
{
    void joiningProcess();//public abstract void joiningProcess();
    void pF();
    void allocateWork();
}
```

```java
public class Emplyee {
	public static void main(String[] args) {
		HrDeptInf i1=new HrDeptInf();
		i1.joiningProcess();
		i1.allocateWork();
		i1.pF();
		i1.location();
	}
}

abstract class Infosys implements Employer
{
	@Override
	public void joiningProcess() {
	System.out.println("1.Selection \n2.Document Verification\n3.Send
Offer Letter");
}

	@Override
	public void pF() {
	System.out.println("As per the standard norm of EP");
	}

	abstract public void location();
}
class HrDeptInf extends Infosys
{
	@Override
	public void allocateWork() {
		System.out.println("Cycle-1 is for Development\nCycle-2 is for
QA");
	}

	@Override
	public void location() {
		System.out.println("For training-Bangalore
Deployment-Hyderabad");
	}
}
```

Output:

```
1.Selection
2.Document Verification
3.Send Offer Letter
Cycle-1 is for Development
Cycle-2 is for QA
As per the standard norm of EP
```

For training-Bangalore Deployment-Hyderabad

Example-3:

```
interface ATM
{
	void deposit();
	void withdrawl();
}

class Cust implements ATM
{

	@Override
	public void deposit() {
	System.out.println("Please deposit ur amount");
	}

	@Override
	public void withdrawl() {
	System.out.println("Collect ur amount");
	}
}
public class User2 {

	public static void main(String[] args) {
		Cust s1=new Cust();
		s1.deposit();
		s1.withdrawl();
	}
}
```

Output:

```
Please deposit ur amount
Collect ur amount
```

```
-----------
08-09-2020
-----------
RBI program
------------
interface RBI
{
      void deposits();
      void withdrawls();
      void aadharLink();
      void minBal();
      //default method
      default void KYC()
      {
      System.out.println("Update ur KYC as early as possible");
      }
}

abstract class SBI implements RBI
{
      public void deposits() {
      System.out.println("1.U can deposit ur money in SBH happily");
      }
      public void withdrawls() {
      System.out.println("2.U can withdrawl ur money from SBH sadly");
         }
      public void aadharLink() {
      System.out.println("3.As per rule of RBI please link ur aadhar with
account");
      }
}

class SBH extends SBI
{
      public void minBal() {
      System.out.println("4.As per rules please maintain 3k as Min
balance");
      }
}

class ICIC implements RBI
{
      public void deposits() {
      System.out.println("5.U can deposit ur money in ICIC happily");
      }
      public void withdrawls() {
```

```
	System.out.println("6.U can withdrawl ur money from ICIC happily");
	}
	public void aadharLink() {
	System.out.println("7.As per rule u need to link Aadhar");
	}
	public void minBal() {
	System.out.println("8.For All ICIC cust it is mandt to maintain min
bal of 2500Rs/-");
	}
}

class PNB implements RBI
{

	public void deposits() {
	System.out.println("9.U can deposit ur money in PNB happily");
	}
	public void withdrawls() {
	System.out.println("10.U can withdrawl ur money from PNB happily");
	}
	public void aadharLink() {
	System.out.println("11.As per rule u need to link Aadhar");
	}
	public void minBal() {
	System.out.println("12.For All PNB cust it is mandt to maintain min
bal of 2000Rs/-");
	}
}

class HDFC implements RBI
{

	public void deposits() {
	System.out.println("13.U can deposit ur money in PNB happily");
	}
	public void withdrawls() {
	System.out.println("14.U can withdrawl ur money from PNB happily");

	}
	public void aadharLink() {
	System.out.println("15.As per rule u need to link Aadhar");
	}
	public void minBal() {
	System.out.println("16.For All PNB cust it is mandt to maintain min
bal of 3500Rs/-");
	}
```

```
}
public class People {
	public static void main(String[] args)
	{
		SBH s1=new SBH();
		s1.aadharLink();
		s1.minBal();
		s1.deposits();
		s1.withdrawls();
		s1.KYC();
		ICIC i1=new ICIC();
		i1.aadharLink();
		i1.deposits();
		i1.withdrawls();
		i1.minBal();
		i1.KYC();
		PNB p1=new PNB();
		p1.aadharLink();
		p1.minBal();
		p1.deposits();
		p1.withdrawls();
		p1.KYC();
		HDFC h1=new HDFC();
		h1.aadharLink();
		h1.minBal();
		h1.deposits();
		h1.withdrawls();
		h1.KYC();
	}
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java People
3.As per rule of RBI please link ur aadhar with account
4.As per rules please maintain 3k as Min balance
1.U can deposit ur money in SBH happily
2.U can withdrawl ur money from SBH sadly
Update ur KYC as early as possible
7.As per rule u need to link Aadhar
5.U can deposit ur money in ICIC happily
6.U can withdrawl ur money from ICIC happily
8.For All ICIC cust it is mandt to maintain min bal of 2500Rs/-
Update ur KYC as early as possible
11.As per rule u need to link Aadhar
12.For All PNB cust it is mandt to maintain min bal of 2000Rs/-
9.U can deposit ur money in PNB happily
10.U can withdrawl ur money from PNB happily
```

```
Update ur KYC as early as possible
15.As per rule u need to link Aadhar
16.For All PNB cust it is mandt to maintain min bal of 3500Rs/-
13.U can deposit ur money in PNB happily
14.U can withdrawl ur money from PNB happily
Update ur KYC as early as possible
```

Multiple Inheritance through Interface

```
----------------------------------------
interface A
{
      void m1();
}
interface B
{
      void m2();
}
class C implements A,B
{
      public void m1() {
            System.out.println("In m1 method");
      }
      public void m2() {
            System.out.println("In m2 method");
      }}
public class Main
{
  public static void main(String[] args) {
      C c1=new C();
      c1.m1();
      c1.m2();
  }}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Main
In m1 method
In m2 method
```

- Through classes it is not possible because of:-
  1.Ambiguity problem
  2.Constructor chaining problem
  3.diamond problem

Q. Why Multiple inheritance is possible through interface?

A.

- A class cannot extend more than one class but it can implements multiple interfaces.
- In interafce there is no possibility of Ambiguity problem because even though multiple interfaces contains same method name but implementation will be given only once because in interface class gives implementation and it won't implement multiple methods with same name and same argument in single class

```
interface A     //Ex
{
      void m1();
}
interface B
{
      void m1();
}
class C implements A,B
{
      public void m1() {
            System.out.println("In m1 method");
}}
public class Main
{
   public static void main(String[] args) {
      C c1=new C();
      c1.m1();
  }}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java Main
In m1 method
```

- Since interface does not contain Constructors there is no possibility of Constructor chaining in interface.
- Therefore, when there is no ambiguity problem and constructor chaining problem, we can achieve multiple inheritance through interface.

Advantages :

---

1. We can achieve 100% abstraction.
2. It makes complex design into simple form.
3. Achieve multiple inheritance.

*** Difference Between Abstract class and Interface

abstract class A
{
  }
 	
	interface A
	{
	  }

class keyword must be preceeded with abstract keyword.	interfacename must be preceeded with interface keyword.
abstract class can have complete as well as incomplete methods.	Interface methods are by default abstract.
In abstract class we can hv public,private,protected,static,non static,final and non final variables.	In interface all variables are by default static, public and final.
Constructors are allowed in abstract class.	Constructors are not allowed.
Using abstract class, we can achieve 0-100 % abstraction.	Using interface, we can achieve 100% abstraction.
A class can extend only one abstract class at a time. abstract class A{} abstract class B{} class C extends (either A or B)	A class can implement multiple interfaces at a time. interface A{} interface B{} class C implements A,B
Using Abstract class, we cannot achieve multiple inheritance.	Using interface, we can achievemultiple inheritance.
We go for abstract classes, when we knows partial implementation.	We go for interface when we only knows the specification but not implementation.

Ex:
abstract class EMP
{
public void getsal()
{
SOP("CAL per Annum");
}
abstrct p v getincentives();
}
	Ex:
	interface EMP
	{
	void getsal();
	void getincentives();
	}

09-09-2020

GENERALISATION AND SPECIALISATION

METHOD DEF	run()	run(int i)	run(char ch,int i)	run(Car s)
METHOD CALL	run()	run(100)	run('A',600)	run(new Car())

Specialisation

The process of developing a method which handles only one type of object such methods are called as special methods and process is called as specialisation.

```
interface Animal            //Ex
{
      void eat();
      void makesound();
}
class Dog implements Animal
{
      public void eat(){
      System.out.println("Dog eats biscuits");
      }
      public void makesound(){
      System.out.println("Bow Bow Bow");
       }
}
class Cat implements Animal
{
      public void eat(){
      System.out.println("Cat eats mouse");
      }
      public void makesound(){
      System.out.println("meow meow meow");
       }
}
public class GenRspec{
            public static void main(String[] args){
                        dogDetails(new Dog());
                        catDetails(new Cat());
      }
            public static void dogDetails(Dog d) //Dog d=new Dog()
            {
```

```
                    d.eat();
                    d.makesound();
        }
        public static void catDetails(Cat c) //Cat c=new Cat()
        {
                    c.eat();
                    c.makesound();
        }
}
```

Output:

```
Dog eats biscuits
Bow Bow Bow
Cat eats mouse
meow meow meow
```

- The main disadvantage of specialisation is, methods will handle only one type of object.
- For an instance in above example, if we add Monkey class we need to add another method which handles monkey object.

```
        public static void monkeyDetails(Monkey m)
        {
              m.eat();
              m.makesound();
        }
```

- Therefore, to overcome this problem we go for generalisation.

Generalisation

The process of developing a method which can handles any type of object, is called as general method and this process is known as generalisation.

```
interface Animal      //Ex
{
      void eat();
      void makesound();
}
class Dog implements Animal
{
      public void eat(){
      System.out.println("Dog eats biscuits");
      }
      public void makesound(){
```

```
      System.out.println("Bow Bow Bow");
       }
}
class Cat implements Animal
{
      public void eat(){
      System.out.println("Cat eats mouse");
      }
      public void makesound(){
      System.out.println("meow meow meow");
       }
}
class Monkey implements Animal
{
      public void eat(){
      System.out.println("Monkey eat bananas");
      }
      public void makesound(){
      System.out.println("kheow kheow kheow");
       }
}
public class GenRspec{
           public static void main(String[] args){
                 animalDetails(new Dog());
                 animalDetails(new Cat());
                 animalDetails(new Monkey());
      }
      public static void animalDetails(Animal a) //Animal a=new Dog()
      {                                          //Animal a=new Cat()
                 a.eat();                        //Animal a=new Monkey()
                 a.makesound();
       }
}
```

Output:

```
Dog eats biscuits
Bow Bow Bow
Cat eats mouse
meow meow meow
Monkey eat bananas
kheow kheow kheow
```

- Using Generalisation + Upcasting + Method overriding, we can chieve RunTime Polymorphism.

```
class Animal
{
            public void eat()
            {
                  System.out.println("Animal eats");
            }
            public void makesound()
            {
            System.out.println("Different animals make different sounds");
            }
}
class Dog extends Animal
{
      public void eat(){
      System.out.println("Dog eats biscuits");
      }
      public void makesound(){
      System.out.println("Bow Bow Bow");
      }
}
public class GenRspec{
       public static void main(String[] args){
                       animalDetails(new Dog());
                       animalDetails(new Animal());
      }
        public static void animalDetails(Animal a)//Animal a=new Dog()
        {
               if(a instanceof Dog)
                {
                      Dog d1=(Dog)a;
                       d1.eat();
                       d1.makesound();
                 }
                 else
                 {
                           a.eat();
                           a.makesound();
                 }
            }
  }
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>java GenRspec
Dog eats biscuits
```

```
Bow Bow Bow
Animal eats
Different animals make different sounds
```

Conclusion

```
So RunTime Polymorphism is a combination of Method Overriding, Upcasting
and Generalisation.
```

Final Keyword

- Final is a keyword which basically indicates that there is no more changes are allowed.

```
      Example :  final match
                 final result
                 final destination
```

- final keyword is applicable with -

```
      1.methods
      2.variables(Local and Global)
      3.class
```

final with variable

- if a variable is declared as "final", we cannot reinitialise the value.

```
Ex: final int score=100;
    score=300;//Re-intialisation-->not allowed bcoz score is final
```

```
possible combination of final variable
--------------------------------------
final static int i=100;//final with static variable
final String m="fail";//final with non static variable
m(){
final char gender='M';//final with local variable
}

class A          //Ex
{
      final String maritalstatus="Married";
      public static void main(String args[])
   {
      A a1=new A();
      a1.maritalstatus="Single";//CTE bcoz final cannot be reintialised
      System.out.println(a1.maritalstatus);
```

```
    }
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>javac A.java
A.java:7: error: cannot assign a value to final variable maritalstatus
        a1.maritalstatus="Single";//CTE bcoz final cannot be reintialised
          ^
1 error
```

- It is mandatory to intialise final variable while declaraion, if we did not intialise we will get compile time error.

```
public class FinalDemo      //Ex
{
      final String maritalstatus;//CTE
      public static void main(String [] args){
                  FinalDemo a=new FinalDemo();
                  System.out.println(a.maritalstatus);
            }
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>javac FinalDemo.java
FinalDemo.java:3: error: variable maritalstatus not initialized in the
default constructor
        final String maritalstatus;//CTE
                     ^
1 error
```

** Difference between static and final

static	final
static means single copy.	final means fixed copy.
it is not mandatory to intialise static while declaration.	it is mandatory to intialise final while declaration.
static variable can be reintialised.	final variable cannot be reintialised.
A class cannot be static.	A class can be final.
static keyword is applicable with method, variables, blocks and nested class.	final is applicable with methods, variables and classes.
Ex:static String clg="Qspiders";	Ex:final float pie=3.414f;

final with class

- if a class is declared as final, we cannot inherit or extends that class. So,therefore a super class can never be final.

```
final class University      //Ex
{
      public void results()
      {
            class Student exends University  //CTE
            {
                  }
}}
```

- But a final class can extends Non final class, i.e, subclass can be final but super class cannot be final.

```
class Central  //Ex
{
      public void lockdown()
      {
            }}
      final class Andhra extends Central
      {
            }
```

- The above program is perfectly correct because here super is non final but sub class is final. So, Final class can inherit non final class.
- But again, class Andhra cannot be extended by other class because it is final.

final with methods

- if a method is declared as final we cannot override it.
- final keyword says that do not make further changes and in overriding we are changing the implementation. Therefore, final methods cannot be overridden.

```
class Flipkart        //Ex
{
      public final void logo()
      {
            System.out.println("Flipkart's Logo");
      }
}
class Myntra extends Flipkart
{
```

```
	public final void logo()
	{
		System.out.println("Myntras's Logo");
	}
}
```

Output:

```
C:\Users\Ravi Kiran\Qspiders Java\programs>javac Flipkart.java
Flipkart.java:10: error: logo() in Myntra cannot override logo() in
Flipkart
        public final void logo()
                          ^
  overridden method is final
1 error
```

Q. Will final methods get inherited ?

A. Yes, final methods will get inherited but only thing is they cannot be overridden.

Q. Can a constructor be final ?

A. No, a constructor cannot be final because final keyword is applicable only for class, methods and final not with constructors.

Conclusion

```
-----------
final variable------->cannot be changed but must be intialised during
                      declaration
final method-------->cannot be overridden but can be inherited
final class--------->cannot be inherited
```

10-09-2020
---
super keyword

- it is used to call super class instance members(non satic variables and non static methods).
- it is used in sub class non static methods.

```
class A                 //Ex-1
{
   String s="Super-class variable";//global-nonstatic
  public void m1()
  {
    System.out.println("This is m1() of A class");
  }
}
class B extends A
{
  public void m1()
  {
    System.out.println("This is m1() of B class");
  }
  public void m2()
  {
   String s="m2()-local variable";//local var
   System.out.println("This is m2() of B-class");
   super.m1();//call to m1()
   System.out.println(super.s);
  }
}
class User1
{
  public static void main(String args[])
  {
    B b1=new B();
    b1.m2();
}}
```

Output :

```
This is m2() of B-class
This is m1() of A class
Super-class variable
```

```
class A                    //Ex-2
{
      int i=100;
      public  void m1()
      {
            System.out.println("This is an m1-method of A-class");
      }
}
class B extends A
{
      public void m1()
      {
            System.out.println("This is an m1-method of B-class");
      }
      public void m2()
      {
            int i=200;
            System.out.println("This is an m2-method of B=class");
            super.m1();
        System.out.println(super.i);
      }
}
class C extends B
{
      public void m3()
      {
            super.m1();
      }
}
public class Sample {
public static void main(String[] args) {
    B b1=new B();
    b1.m2();
     C c1=new C();
     c1.m3();
      }
}
```

Output :

```
This is an m2-method of B=class
This is an m1-method of A-class
100
This is an m1-method of B-class
```

Differences between :-

this keyword	super keyword
this keyword indicates the current object under execution.	super keyword is used to call super class instance members.
this can be used inside methods and constructors.	super is used in subclass non static methods.
For using this keyword inheritance is not required.	For using super inheritance is required.

call to this	call to super
it is used to call one constructor from another constructor of same class	it is used to call super class constructor from sub class
For achieving call to this, inheritance is not required.	For achieving super(), inheritance is must.
this() is Explicit in nature.	super() is Implicit as well as Explicit in nature.

Access Specifiers :-

- It provides the accessible permission to various fields of program.
- basically they are of 4 types -
    1.public
    2.private
    3.protected
    4.default
- They are applicable to
    1.class
    2.method(static & non static)
    3.variables(static & non static)
    4.constructors
    5.Interface

package: It is like a folder where all common classes kept together at one place

- In java we have predefine packages as well as user define packages.
- Some examples of predefine packages are -
  1.java.lang package
  2.java.util package
  3.java.awt package
  4.java.swing package etc

1.public specifier :-

- it is the lowest level specifier.
- public fields are accessible within class anywhere.
- public fields are accessible within another class anywhere of same package.
- public fields are accessible within another class of another package only after import statements.

import :-

- It provides the previlege of using code of one class into another class
- syntax: import packagename.classname;
  import packagename.*;

- Accessing public field within same class and outside class of same package

```
package pac1.java;
public class d11 {
      public static int i=100;
      public static void main(String[] args)
{
        d22.fly();
        //accessible directly within class
       System.out.println(i);
      }
}
class d22
{
      public static void fly()
      {
            //it is accessible directly outside of class coz it is public
      System.out.println(d11.i);
}}
```

- Accessing public field in another class in another package but only after import statement.

```
package pac2.java;
import pac1.java.d11;
public class d22 {
     public static void main(String[] args) {
          // TODO Auto-generated method stub
   System.out.println(d11.i);
     }}
```

- If a class is public then only it can be imported other wise we can't import non public classes

```
          VALID  package java.ab;
          for ex:    public class A
                     {}
                     import java.ab.A;
           INVALID  package java.ab;
           for ex:   class A
                    {}
                    import java.ab.A;
```

2.default specifier :-

- In java there is no such word called as default i.e if we did not specify any modifier like public, private or protected. JVM will consider it as default.
- default fields can be accessible anywhere within class.
- default fields are accessible outside of class but that class should belongs to same package.
- default classes cannot be imported.
- default specifier is also called as package specifier because it is restricted within package.

3.private specifier :-

- it provides the highest level of restrictions.
- private members can only be accessed within same class.
- private fields cannot be accessed outside of declared class.
- A class can never be private.
- private methods can't be inherited and overridden.

4.protected specifier :-

- protected fields can be accessible anywhere within class.
- protected fields can be accessible within another class but that class should belongs to same package.

```
package pac1.java;

 public class d11 {
      protected static int i=100;
      public static void main(String[] args){
        d22.fly();
        //accessible directly within class
       System.out.println(i);
      }
}
class d22
{
      public static void fly()
      {
//it is accessible directly outside of class coz it is protected and belongs
to same package
      System.out.println(d11.i);
}
}
```

- protected fields can be accessible within another class of another package but only after inheritance.

```
package pac1.java;

public class A {
      protected static int i=100;
       protected int j=200;
}
package pac2.java;

import pac1.java.A;

public class B  extends A{

      public static void main(String[] args) {
            // TODO Auto-generated method stub
            System.out.println(A.i);
```

```
            B a1=new B();
            System.out.println(a1.j);
        }
}
```

- A class can never be protected.

note:

- We can write multiple import statements to use multiple classes.

For ex:
```
public class A
 {}
public class B
{}
public class C
 {}
```

```
import A
import B
import C
class D
  {
    //some code//
  }
```

11-09-2020

MODULE-4

Object class :

- it is super class to all pre define and user define classes.
- Object class is present in java.lang package.
- javap java.lang.Object

Method of object class:
---
1. getclass():class----->final
2. object1.equals(object2):boolean
3. toString():String
4. hashcode():int
5. notify():void----->final
6. notifyAll():void---->final
7. wait():void------->final
8. wait(long timeout):void---->final
9. wait(long timeout,int nanos):void---->final
10. finalise()
11. clone()

toString():

- it's a method of object class.
- When we call toString()on any object, it provides complete information of an object.
- Complete information consists of packagename.classname@objectaddress
  Ex:mypackage.Sample@zzh33453

Syntax:
```
public String toString()
{
  //return packagename.classname@objectaddress;
}
```

- Object address is unique address given to every object.
- it cannot be same for 2 objects.
- Whenever we print reference variable implicitly, it calls to toString()

```
class Object
{
	public String toString()
	{
		return packagename.classname@objectaddress;
	}
}
class Sample extends Object
{
	public String toString()
	{
		return our own implementation;
	}
}
class User
{
	public static void main(String[] args)
	{
		Sample s1=new Sample();
		System.out.println(s1);
	}
}
```

```
package java.prog;		//Ex
public class Sample
//public class Sample extends Object
{
	public static void main(String[] args)
	{
		Sample s1=new Sample();
		System.out.println(s1.toString());
		System.out.println(s1.hashCode());
		Sample s2=new Sample();
		System.out.println(s2.toString());
		System.out.println(s2.hashCode());
		Sample s3=s1;
		System.out.println(s3.toString());
		System.out.println(s3.hashCode());
		System.out.println(s1);//s1.toString()
		System.out.println(s2);//s2.toString()
		System.out.println(s3);//s3.toString()
	}
}
```

Output:

```
java.prog.Sample@2f92e0f4
798154996
```

```
java.prog.Sample@28a418fc
681842940
java.prog.Sample@2f92e0f4
798154996
java.prog.Sample@2f92e0f4
java.prog.Sample@28a418fc
java.prog.Sample@2f92e0f4
```

hashcode()

- it's a method of object class.
- Whenever we call hashcode() on any object, it prints the hashcode number for given object.
- Hashcode number is simply a 32bit integer number.
- it is a unique number allocated to every object by JVM.
- if the object address are same they will have same hashcode number.

Syntax:

```
public int hashCode()
{
      return hashcodenum;
}
```

```
package objectdemo.in;      //Ex
public class Sample{
      public static void main(String[] args)
{
      Sample s1=new Sample();
      System.out.println(s1.hashCode());
      Sample s2=new Sample();
      System.out.println(s2.hashCode());
      Sample s3=s1;
      System.out.println(s3.toString());
      System.out.println(s1.toString());
      System.out.println(s3.hashCode());
}}
```

Output:

```
798154996
681842940
objectdemo.in.Sample@2f92e0f4
objectdemo.in.Sample@2f92e0f4
798154996
```

Conclusion :

---

1. Output we got from toString() in practical there is no use of it, because if package changes and class changes output won't be same. So, it is programmer's responsibility to overide this toString() and get some meaningful output.

2. Output we are getting from hashCode() in practical there is no use of it to programmer because if object address changes, it changes hashcode number. So, it is our responsibility to override hashCode() and get meaningful output.

Overriding of toString() and hashCode()

```
package objectdemo.in;
public class Student {
	String sname;
	int sid;
	public Student(String sname,int sid) {
		this.sname=sname;
		this.sid=sid;
	}
	public String toString() {
		return sname;
	}
	public int hashCode() {
		return sid;
	}
	public static void main(String[] args)
	{
		Student s1=new Student("John",1221);
		System.out.println(s1.toString());
		System.out.println(s1.hashCode());
		Student s2=new Student("rohan",1222);
		System.out.println(s2.toString());
		System.out.println(s2.hashCode());
	}}
```

Output:

```
John
1221
rohan
1222
```

14-09-2020
---

equals()

- it's a method of object class it compares two objects based on object address.
- if object address is same, output is true else it is false.

```
public class Sampl {                                    //Ex
      public static void main(String[] args) {
            Sampl s1=new Sampl();
            Sampl s2=new Sampl();
            Sampl s3=s1;
            Sampl s4=s2;
            System.out.println(s1+" "+s2+" "+s3+" "+s4);
            System.out.println(s1.equals(s2));
            System.out.println(s2.equals(s3));
            System.out.println(s1.equals(s3));
            System.out.println(s4.equals(s2));
      }}
```

Output:

```
Sampl@372f7a8d Sampl@2f92e0f4 Sampl@372f7a8d Sampl@2f92e0f4
false
false
true
true
```

Conclusion

---

equals() compares the object based on there object address which is not a proper way for comparison so it is programmers's responsibility to override equals(). So that, it should compare based on features not based on address.

Note:

---

s1==s2 and s1.equals(s2) are not same.

Syntax:

```
public boolean equals(Object o)
      {
            return byComparngAddress;
      }
```

ARRAYS

- An array is collection of homogenous elements(data).
- Whenever we want to use multiple or group of elements or data at same time we go for arrays.
- for ex: if i want to use 1 to 100 at same time, its not possible to store in variables and use it because it is very lengthy process so for such kind of situation, java has given arays.
- In java an array is an Object.

Syntax:

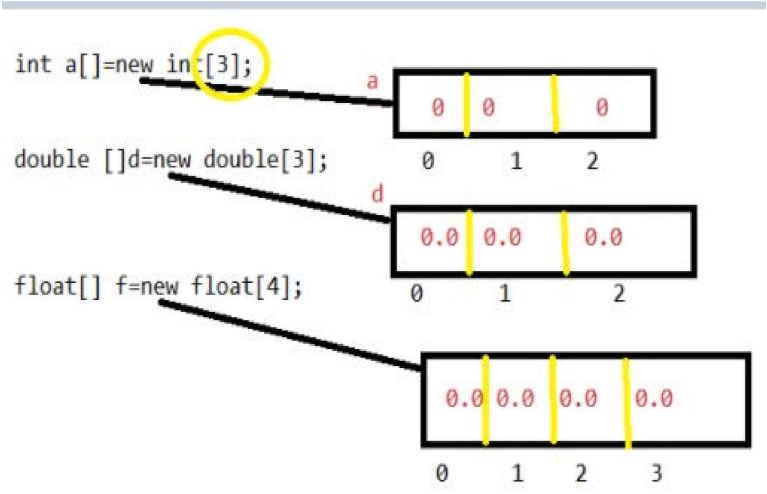
```
arraytype arrayname[]=new arraytype[size];
arraytype[] arrayname=new arraytype[size];
arraytype []arrayname=new arraytype[size];
```

Ex:

```
int a[]=new int[3];
double []d=new double[3];
float[] f=new float[4];
```

- array stores the elements in index format which always starts from 0(for above example it is 0,1,2).
- Once array gets created there will be default values stored in that array object.
- Once we created an aray default values will be present.
- storing elements in an array by using index values

```
a[0]=10;
a[1]=20;
a[2]=30;
doubled[]=new double[4];
d[0]=0.22;
d[1]=0.33;
d[2]=0.44;
d[3]=0.55;
```



- for printing elements of array

```
System.out.println(a[0]);
System.out.println(a[1]);
System.out.println(d[0]);
System.out.println(d[1]);
System.out.println(d[2]);
```

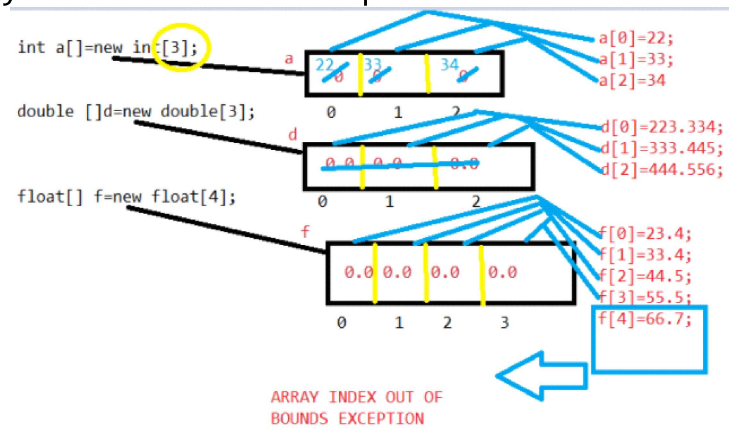
- When we already know elements of array we can also create the array as

```
arraytype arrayname[]=(elements);
int a[]={10,22,33,44,555,666,7777,88);
System.out.println(a[0]);
System.out.println(a[1]);
```

System.out.println(a[2]);
|
|
so on.....

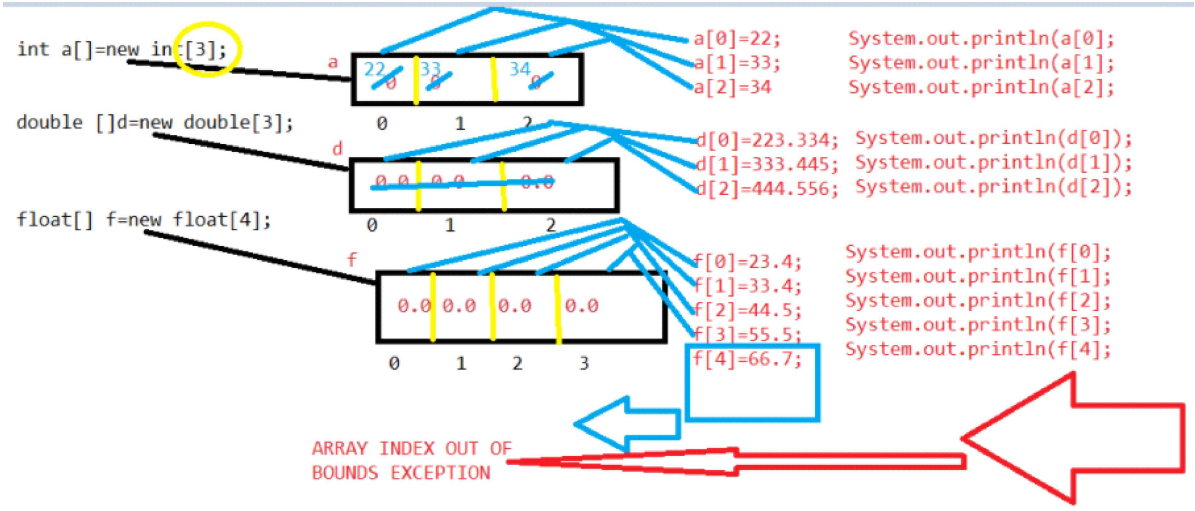
- if we store elements more than declared size we will get array index out of bounds exception.

ex: int a[]=new int[3];
a[0]=11;
a[1]=22;
a[2]=33;
a[3]=44;//array index out of bounds exception//
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException:4



Length variable:

- It provides length of array and length will always calculated from 1.
  ex: int a[]=new int[300];
  System.out.println(a.length);
  Output is :300
- Size of the array is cannot be a decimal value.
  int a[]=new int[4.0];//CTE
- it is mandatory to give size of array if we did not given size, we will get CTE
  int a[]=new int[];//CTE




```java
public class DemoArray {                                    //Ex
public static void main(String[] args) {
          // array creation
          int a[]=new int[3];
          System.out.println(a[0]);//0
          System.out.println(a[1]);//0
          System.out.println(a[2]);//0
          //storing elements in an array
          a[0]=100;
          a[1]=200;
          a[2]=300;
          //Retrieving
          System.out.println(a[0]);//100
          System.out.println(a[1]);//200
          System.out.println(a[2]);//300
          System.out.println(a.length);//3
          /* Array index out of bounds exception
          a[3]=400;
          System.out.println(a[3]);
          a[-1]=400;
          System.out.println(a[-1]); */
          String s[]= {"John","Rohan","Roman","Riya","Sneha"};
          System.out.println(s[0]);
          System.out.println(s[1]);
          System.out.println(s[2]);
          System.out.println(s[3]);
          System.out.println(s[4]);
     }}
```

```
Output:
0
0
0
100
200
300
3
John
Rohan
Roman
Riya
Sneha
```

15-09-2020

- Aray stores only homogenous data.

```
int a[]=new int[4];
a[0]=10;
a[1]=445.5f;//CTE
a[2]=123.334;//CTE
a[3]=false;//CTE
```

- Array stores only homogeneous data i.e, in integer array we can add only integer data, if we add any other type of data we wil get compile time error.

```
public class Array1 {                    //Ex
public static void main(String[] args) {
      int a[]=new int[5];
      a[0]=100;
      a[1]=200;
      a[2]=300;
      a[3]=400;
      a[4]=500;
  /*  System.out.println(a[0]);
      System.out.println(a[1]);
      System.out.println(a[2]);
      System.out.println(a[3]);
      System.out.println(a[4]);    */
      System.out.println(a.length);
      //print array elements using for loop
      for(int i=0;i<a.length;i++)
      {
            System.out.println(a[i]);
      }
      String s[]= {"java","sql","aptitude","selenium"};
      System.out.println("-----Printing using for each loop------");
      //for-each Loop or enhanced Loop
      /*    syntax: for(arraytype varname:arrayname)
            {
            System.out.println(varname);
            }                              */
      for(String data:s)
      {
            System.out.println(data);
      }
}
```

```
}
Output:
5
100
200
300
400
500
-----Printing using for each loop------
java
sql
aptitude
selenium
```

```java
import java.util.*;                    //Ex: Using scanner
public class Array2 {
public static void main(String[] args)
{
      Scanner s=new Scanner(System.in);
      int size;
      System.out.println("Enter the size of an array:");
      size=s.nextInt();//4
      int a[]=new int[size];//an array gets created of size=4
      //for accepting inputs
      for(int i=0;i<a.length;i++)
      {
            a[i]=s.nextInt();
      }
      System.out.println("Array elements are.........");
      //for retrieving values
      for(int j=0;j<a.length;j++)
      {
            System.out.println(a[j]);
      }
}
}
```

1.WAP to take an array from the user and print first and last element.

```
import java.util.*;
public class PrintElement {
	public static void main(String[] args) {
		Scanner s=new Scanner(System.in);
		int size;
		System.out.println("Enter the size of an array:");
		size=s.nextInt();
		int a[]=new int[size];
		for(int i=0;i<a.length;i++)
		{
			a[i]=s.nextInt();
		}
			System.out.println(a[0]);
			System.out.println(a[a.length-1]);
		}
	}
```

2.WAP to take an array from user and find
a.sum of an array
b.average of an array.

```
import java.util.*;
public class SumAvg {
	public static void main(String[] args) {
		Scanner s=new Scanner(System.in);
		int size;
		System.out.println("Enter the size of an array:");
		size=s.nextInt();
		int a[]=new int[size];
		for(int i=0;i<a.length;i++)
		{
			a[i]=s.nextInt();
		}
		int sum=0;
		for(int j=0;j<a.length;j++)
		{
		sum=sum+a[j];
		}
		System.out.println("Sun : "+sum);
		float average=(float)sum/a.length;
		System.out.println("Average : "+average);
	}
}
```

16-09-2020

3.WAP by taking input from user, print sm of even position numbers of an array.

```
import java.util.*;
public class SumOfEven {
	public static void main(String[] args) {
		Scanner s=new Scanner(System.in);
		int size;
		System.out.println("Enter the size of an array:");
		size=s.nextInt();
		int a[]=new int[size];
		for(int i=0;i<a.length;i++)
		{
			a[i]=s.nextInt();
		}
		int sum=0;
		for(int j=0;j<a.length;j++)
		{
			if(j%2==0)
		    sum=sum+a[j];
		}
		System.out.println("Sum Of Even positions : "+sum);
	}}
```

Output:

```
Enter the size of an array:
4
22
33
44
55
Sum Of Even positions : 66
```

4.WAP to reverse an array.

```
import java.util.Scanner;
public class RevArray {
	public static void main(String[] args) {
		Scanner s=new Scanner(System.in);
		int size;
		System.out.println("Enter the size of an array:");
		size=s.nextInt();
		int a[]=new int[size];
		for(int i=0;i<a.length;i++)
		{
```

```
                a[i]=s.nextInt();
            }
            for(int j=a.length-1;j>=0;j--)
            {
            System.out.println("a["+j+"]="+a[j]);
            }
      }}
```

Output:

```
Enter the size of an array:
4
22
33
44
55
a[3]=55
a[2]=44
a[1]=33
a[0]=22
```

5.WAP to copy an array into another array.

```
import java.util.Scanner;
public class CopyArray {
      public static void main(String[] args) {
            Scanner s=new Scanner(System.in);
            int size;
            System.out.println("Enter the size of an array:");
            size=s.nextInt();
            System.out.println("Enter the data : ");
            int a[]=new int[size];
            for(int i=0;i<a.length;i++)
            {
                  a[i]=s.nextInt();
            }
            int b[]=new int[a.length];
            //copying
            for(int j=0;j<a.length;j++)
            {
                  b[j]=a[j];
            }
            //printing
            for(int k=0;k<a.length;k++)
            {
                  System.out.println("b["+k+"]="+b[k]);
            }
      }
}
```

Output:

```
Enter the size of an array:
4
Enter the data :
11
22
33
44
b[0]=11
b[1]=22
b[2]=33
b[3]=44
```

6.WAP to take input from user and find greatest(maximum) element from an array.

```
import java.util.Scanner;
public class MaxElement {
	public static void main(String[] args) {
		Scanner s=new Scanner(System.in);
		int size;
		System.out.println("Enter the size of an array:");
		size=s.nextInt();
		System.out.println("Enter the data : ");
		int a[]=new int[size];
		for(int i=0;i<a.length;i++)
		{
			a[i]=s.nextInt();
		}
		int max=a[0];
		//checking condition of maximum element
		for(int j=0;j<a.length;j++)
		{
			if(a[j]>max)
			{
				max=a[j];
			}
		}
		System.out.println("Maximum number fro array is : "+max);
	}}
```

Output:

```
Enter the size of an array:
4
Enter the data :
25
945
326
```

```
8
Maximum number fro array is : 945
```

7.WAP to take input from user and find smallest(minimum) element fro an array.

```
import java.util.Scanner;
public class MinElement {
	public static void main(String[] args) {
		Scanner s=new Scanner(System.in);
		int size;
		System.out.println("Enter the size of an array:");
		size=s.nextInt();
		System.out.println("Enter the data : ");
		int a[]=new int[size];
		for(int i=0;i<a.length;i++)
		{
			a[i]=s.nextInt();
		}
		int min=a[0];
		//checking condition of minimum element
		for(int j=0;j<a.length;j++)
		{
			if(a[j]<min)
			{
				min=a[j];
			}
		}
		System.out.println("Minimum number fro array is : "+min);
	}}
```

Output:

```
Enter the size of an array:
4
Enter the data :
897
26
4
95
Minimum number fro array is : 4
```

8.WAP to sort(arrange) the elements of array in increasing order.
input: 22 5 6 88 9
output: 5 6 9 22 88

```
import java.util.Scanner;
public class SortArray {
	public static void main(String[] args) {
		Scanner s=new Scanner(System.in);
```

```
            int size;
            System.out.println("Enter the size of an array:");
            size=s.nextInt();
            System.out.println("Enter the data : ");
            int a[]=new int[size];
            for(int i=0;i<a.length;i++)
            {
                  a[i]=s.nextInt();
            }
            //for comparing and swapping
            int temp;
            for(int j=0;j<a.length;j++)
            {
                  for(int k=0;k<a.length-1;k++)
                  {
                        if(a[k]>a[k+1])
                        {
                              temp=a[k];
                              a[k]=a[k+1];
                              a[k+1]=temp;
                        }
                  }}
      //for  printing sorted array
            for(int m=0;m<a.length;m++)
            {
                  System.out.println("a["+m+"]="+a[m]);
                  }
}}
```

Output:

```
Enter the size of an array:
5
Enter the data :
22
5
6
88
9
a[0]=5
a[1]=6
a[2]=9
a[3]=22
a[4]=88
```

9.WAP to sort(arrange) the elements of array in decreasing order.
input: 22 5 6 88 9
output: 88 22 9 6 5

```
import java.util.Scanner;
public class SortDec {
	public static void main(String[] args) {
		Scanner s=new Scanner(System.in);
		int size;
		System.out.println("Enter the size of an array:");
		size=s.nextInt();
		System.out.println("Enter the data : ");
		int a[]=new int[size];
		for(int i=0;i<a.length;i++)
		{
			a[i]=s.nextInt();
		}
		//for comparing and swapping
		int temp;
		for(int j=0;j<a.length;j++)
		{
			for(int k=0;k<a.length-1;k++)
			{
				if(a[k]<a[k+1])
				{
					temp=a[k];
					a[k]=a[k+1];
					a[k+1]=temp;
				}
			}}
	//for  printing sorted array
		for(int m=0;m<a.length;m++)
		{
			System.out.println("a["+m+"]="+a[m]);
			}
}}
```

Output:

```
Enter the size of an array:
5
Enter the data :
22
5
6
88
9
a[0]=88
a[1]=22
a[2]=9
a[3]=6
a[4]=5
```

18-09-2020

10.WAP to find second highest element from an aray without sorting.

input: 22 5 6 88 9
output: 22

```
public class SecHigh {
public static void main(String[] args) {
	int a[]= {22,5,6,88,9};
	int max=a[0],secmax=a[0];
	for(int i=1;i<a.length;i++)
	{
		if(a[i]>max)
		{
			secmax=max;
			max=a[i];
		}
		else if(a[i]>secmax)
		{
			secmax=a[i];
		}
	}
	System.out.println("maximum value : "+max);
	System.out.println("second maximum value : "+secmax);
}}
```

Output:

```
maximum value : 88
second maximum value : 22
```

11.WAP to find second smallest element from an aray without sorting.

input: 22 5 6 88 9
output: 6

```
public class SecondSmallest {
public static void main(String[] args) {
		int a[]= {22,5,6,88,9};
		int min=a[0],secmin=a[0];
		for(int i=1;i<a.length;i++)
		{
			if(a[i]<min)
			{
				secmin=min;
				min=a[i];
			}
			else if(a[i]<secmin)
			{
```

```
                    secmin=a[i];
                }
            }
            System.out.println("minimum value : "+min);
            System.out.println("second minimum value : "+secmin);
      }}
```

Output:

```
minimum value : 5
second minimum value : 6
```

12.WAP to find all misiing numbers from 1 to 100 from an array {22,17,4,66,8,2,87}

```
public class Missing {
public static void main(String[] args) {
            boolean status=true;
            int a[]= {22,17,4,66,8,2,87};
            for(int j=1;j<=100;j++)
            {
                  for(int i=0;i<a.length;i++) {
                        if(j==a[i])
                        {
                              status=false;
                              break;
                        }
                  }
                  if(status==true)//number is not present
                  {
                        System.out.println(j);
                  }
                  status=true;//for every number status should be true
            }
      }}
```

Assignment

1.WAP to find prime numbers from an array given below a[]={13,4,56,32,99,11}

PASSING AN ARRAY AS A METHOD ARGUMENTS

```
//Passing an array as an argument to a method
public class Students {
	String name;int age;float per;
	public Students(String name,int age,float per)
	{
		this.name=name;
		this.age=age;
		this.per=per;
	}
	public static void main(String[] args) {
		Students std[]=new Students[3];//i created a students array
		std[0]=new Students("Ravi",27,81.2f);
		std[1]=new Students("Rohan",23,77.5f);
		std[2]=new Students("Riya",21,77.9f);
		details(std);//call details method and pass array as arguments
	}
	public static void details(Students sarray[])
	//Students sarray[]=new Students[3]
	{
		System.out.println("Name Age percentage");
		for(Students s1:sarray)
		{
			System.out.println(s1.name+" "+s1.age+" "+s1.per);
		}
	}
}
```

Output:

```
Name Age percentage
Ravi 27 81.2
Rohan 23 77.5
Riya 21 77.9
```

21-09-2020

EXCEPTION HANDLING

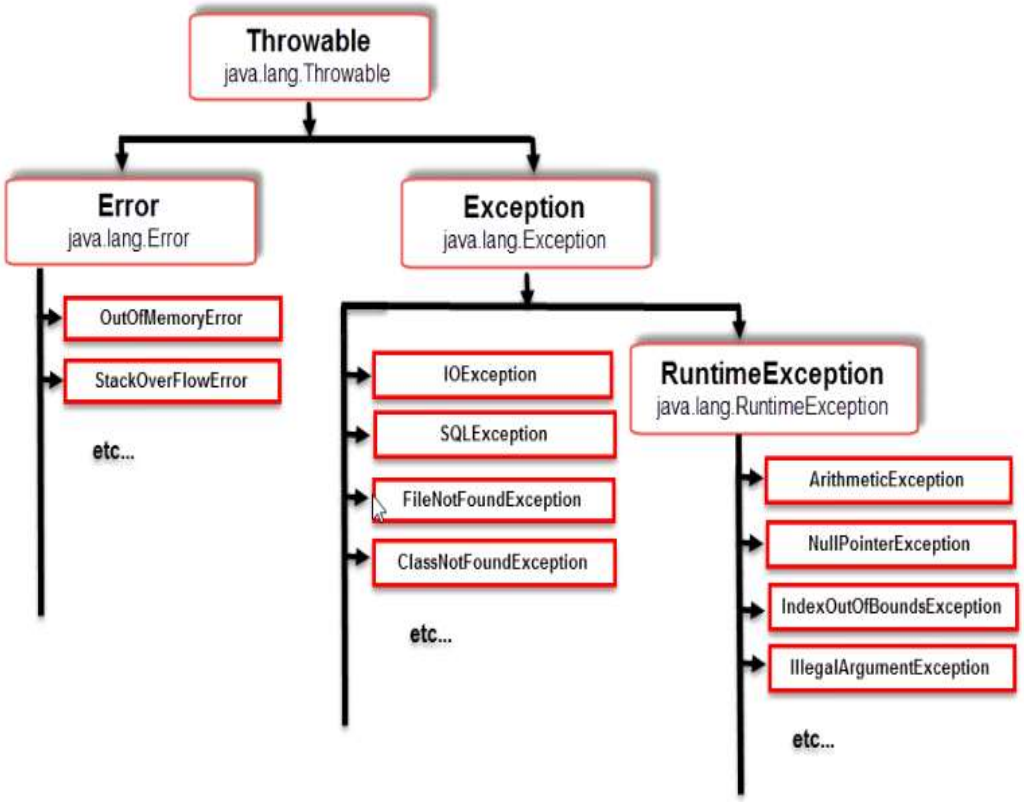
- Exception
- Hierarchy of Exception
- Types of Exception(Different classes in Excepton)
- Keywords of Exception
  try, catch, throw, throws and finally
- difference between throw and throws
- **difference between final, finally, finalise
- ***user define exception or customised exception

Exception :

- An Exception is an unwanted or unexpected condition which disturbs our normal flow of execution.
  Ex: 1.CoronaException
  2.lockDownException
- Once Exception occured remaining part of program will not be executed.
- So, it is our responsibility to handle the exception.
- Exception handling doesn't means, we are resolving an exception it is just like providing an alternate solution so that even though exception happens our program should work properly.

Exception Hierarchy:

- Object class is a super class to all the predefine and userdefine classes of java.
- Throwable class is a super class to "Exception" class and "Error" class.
- Exception class is a super class to RuntimeException class and other Exception classes.
- All the Exception classes belongs to java.lang package.



- Depending on Hierarchy, Exceptions are divided into 2 types -
  1.Checked Exception (Compile time Exceptions)
  2.Unchecked Exception (Run time Exceptions)

Checked Exception

- Exception which are checked(identified or found out) during compile time by compiler, such type of exception are called as Checked Exceptions.

  (or)

  Exception classes which are directly inheriting Exceptionclass except RuntimeExceptionclass is called as checked exception.
- Checked Exceptions are also called as Compile time Exception.
- Examples(Classes) of Checked Exceptions are :-
  -InterruptedException
  -ClassNotFoundException
  -SQLException
  -FileNotFoundException

Unchecked Exception

- Exception which are checked(identified or found out) during Runtime or execution time, such type of exception are called as Unchecked Exceptions.
- Incase of Unchecked Exception our program will atleast compiles successfully.
- Unchecked Exceptions are also called as Runtime Exceptions.
- RuntimeExceptionclass is a super class to all UncheckedException classes.
- Examples(Classes) of Unchecked Exceptions are :-
  - -ArithmeticException
  - -ArrayIndexOutOfBoundsException
  - -NullPointerException
  - -StringIndexOutofBoundsException
  - -ClassCastException
  - -NumberFormatException

Error :

- An Error is an irrecoverable Condition i.e, if error occured it is not under programmers control to get over it.
- For Ex: if we develop any program whose size is 4gb but our system's storage is 3gb so such condition is not in programmers control and such situation is referred as Error.
- Examples(Classes) of Error are :-
  - -StackoverFlowError
  - -VirtualMemoryError
  - -404pagenotfound

***Differences between Error and Exception

Error	Exception
1. An error is caused due to lack of system resources.	1. An exception is caused because of some problem in code.
2. An error is irrecoverable i.e, an error is a critical condition cannot be handled by code of program.	2. An Exception is recoverable i.e, we can have some alternate code to handle exception.
3. There is no ways to handle error.	3. We can handle exception by means of try and catch block.
4. As error is detected program is terminated abnormally.	4. As Exception is occurred it can be thrown and caught by catch block.
5. There is no classification for Error.	5. Exceptions are classified as checked and unchecked.
6. Errors are define in java.lang.Error package	6. Exceptions are define in java.l

---
22-09-2020

What happens when an Exception occured ?

```
class Sample
{
	public static void div()
	{
		int a=10,b=0,c;
		c=a/b;//10/0
		System.out.println("Exception Ocurred");
		System.out.println(c);
	}
	public static void main(String args[])
	{
		div();
	}
}
```

Output:

```
E:\Qspiders Java\programs>java Sample
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Sample.div(Sample.java:6)
        at Sample.main(Sample.java:12)
```

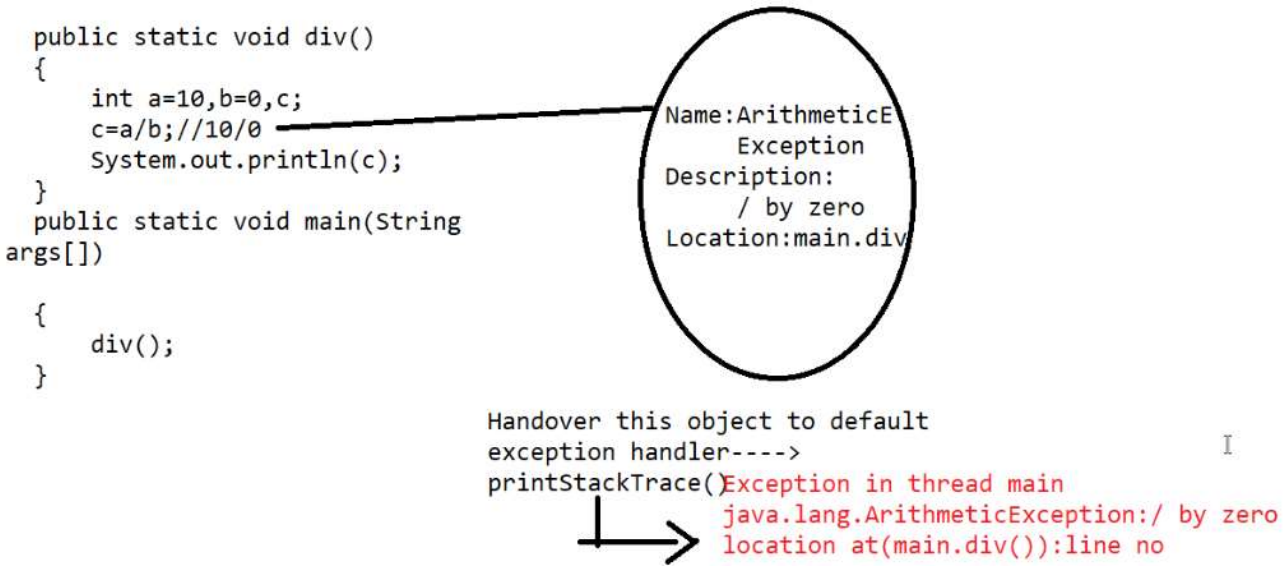
In the above program execution begins from main method and it calls to div(). In div() there is an unexpected statement i.e c=10/0 when this statement is encountered div() creates an Exception object which includes -

Name:
description:
location:

and handover it to JVM, Now JVM will checks if there is any exception handling code present in div() since there is no exception handling code present in div() it checks with caller method in above program i.e, main(). So it checks in main() if there is any exception handling code present or not. Since, there is no exception handling code present in main() also, JVM will calls to default Exception handler i.e printStackTrace()(method) and that default exception handler provides the complete inforamtion of Exception.

i.e, Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Sample.div(Sample.java:6)
        at Sample.main(Sample.java:12)

- Once exception ocurred remaining part of a program will not be executed and ending up with abnormal termination.
- So, if we want the above program to be executed and have normal termination then, we have to handle the Exception.
- Exception can be handled by using try and catch block.



Ex 1: Exception didn't occurred, so except catch block all normal statements will be executed

```
class Sample
{
	public static void div()
	{
		int a=10,b=10,c;
		try
		{
		 System.out.println("Starting of try");
		 c=a/b;//10/10
		 System.out.println("This is a try block");
		}
		catch(ArithmeticException e)
		{
		 System.out.println("Exception is ocurred and handled");
		}
	}
```

```
	public static void main(String args[])
	{
		div();
		System.out.println("End of main");
	}
}
```

Output:

```
E:\Qspiders Java\programs>java Sample
Starting of try
This is a try block
End of main
```

Ex 2: Exception occurred in try block and JVM have executed corresponding catch block after exception statement

```
class Sample
{
	public static void div()
	{
		int a=10,b=0,c;
		try
		{
		 c=a/b;//10/0
		}
		catch(ArithmeticException e)
		{
		 System.out.println("Exception is ocurred and handled");
		}
	}
	public static void main(String args[])
	{
		div();
	}
}
```

Output:

```
E:\Qspiders Java\programs>java Sample
Exception is ocurred and handled
```

Ex 3: Exception occurred in try block and JVM have executed coresponding catch block after exception statement no other statements executed.

```
class Sample
{
	public static void div()
	{
		int a=10,b=0,c;
		try
		{
		 System.out.println("Try starts");//executed
		 c=a/b;//10/0
		 System.out.println("Try block");//unexecuted
		}
		catch(ArithmeticException e)
		{
		 System.out.println("Exception is ocurred and
		                        handled");//executed
		}
	}
	public static void main(String args[])
	{
		div();
		System.out.println("main ends");//executed
	}
}
```

Output:

```
E:\Qspiders Java\programs>Java Sample
Try starts
Exception is ocurred and handled
main ends
```

23-09-2020

```
try
{
      //risky statements which causes exception//
}
catch(Exceptionname refvar)
{
      //Alternate solution
}
```

- try block is used to keep a code which causes an exception.
- Once Exception occured in try block remaining part of code from try block will not be executed. So, we should keep only statements which causes exception under try block.
- Once exception occurred in try block, JVM immediately make a search for corresponding catch block.
- Catch block is where we will caught the exception.
- Under catch block we provide some statements which works as alternate solution for Exception.

Q. Can we write any statement between try and catch block ?
A. No, we cannot write any statement between try and catch block. Immediately after try block there should be catch or finally block.

Q. Can we write only try block without catch block ?
A. No, a try block should always followed by either catch or finally block.

Q. If we don't know exception type, what type should we mention in catch block ?
A. When we do not know Exception type, we can mention it as ExceptionClass type or Throwable type. Because, Exception is a super class to all the class and during upcasting we studied that superclass can hold reference of subclass object.

Ex: Exception e=new ArithmeticException()
Ex: Throwable e1=new ArithmeticException()

```
class Sample          //Ex
{
      public static void main(String args[])
  {
            int a=10,b=0,c;
      try
      {
```

```
       //put a code which causes exception//
       c=a/b;//An exception object get created by main() and handover to
JVM
       System.out.println("Under try block");//Unexecuted statement
      }
      catch(Exception e)//e is reference which holds ArithmeticExp object
      {
       System.out.println("Exception is being noticed");
      }
   }
}
```

Output:

```
Exception is being noticed
```

Note:

- Single program can have multiple risky statements i.e, one try block can have multiple exception statement.
- In that case only, one Exception will get caught beacuse one try block can define one exception at a time.
- If we want to handle multiple exceptions, it is always recommended to define separate try and catch blocks.

```
class Sample
{
      public static void main(String args[])
  {
            int a=10,b=0,c;
            int d[]=new int[3];
      try
      {
       d[3]=55;//beyond declared size so an exception gets created
       c=a/b;//unexecuted
       }
      catch(ArithmeticException e)
      {
       System.out.println("Exception is caught");
      }
      catch(ArrayIndexOutOfBoundsException f)
      {
       System.out.println("Exception is caught for array");
      }
   }
}
```

Output:

```
Exception is caught for array
```

- The above program is always recommended to write in below way -

```
class Sample
{
      public static void main(String args[])
  {
            int a=10,b=0,c;
            int d[]=new int[3];
      try
      {
        d[3]=55;//beyond declared size
       }
      catch(ArrayIndexOutOfBoundsException f)
      {
       System.out.println("Exception is caught for array");
      }
         try
      {
       c=a/b;
       }
      catch(ArithmeticException e)
      {
       System.out.println("Exception is caught");
      }
   }
}
```

Output:

```
Exception is caught for array
Exception is caught
```

Single try with multiple catch blocks

- Yes, it is allowed to write single try with multiple catch blocks but it should be most specific to most general.

```
class Sample
{
      public static void main(String args[])
  {
            int a=10,b=0,c;
            int d[]=new int[3];
      try
      {
        d[3]=55;//beyond declared size
       }
  //most specific catch block
      catch(ArrayIndexOutOfBoundsException f)     //AIOBE f=new AIOBE()
```

```
      {
       System.out.println("Exception is caught for array");
      }
  //general catch block
      catch(Exception e)    //Exception e=new AIOBE()
      {
       System.out.println("Exception is caught");
      }
   }
}
```

Output:

```
Exception is caught for array
```

- In the above program there is ArrayIndexOutOfBoundsException occurred and if we have written multiple catch blocks we always have to write specific catch block i.e, catch block which contains AIOUBException reference first then we can write general catch block means catch block which have Exception class reference.

- For the above program, we cannot define catch blocks as most general to most specific if we did, we will get compile time error.

```
class Sample
{
      public static void main(String args[])
  {
            int a=10,b=0,c;
            int d[]=new int[3];
      try
      {
        d[3]=55;//beyond declared size
       }
  //general catch block
      catch(Exception e)    //Exception e=new AIOBE()
      {
       System.out.println("Exception is caught");
      }
  //most specific catch block
      catch(ArrayIndexOutOfBoundsException f)     //AIOBE f=new AIOBE()
      {
       System.out.println("Exception is caught for array");
      }
   }
}
```

Output:

```
Sample.java:17: error: exception ArrayIndexOutOfBoundsException has
already been caught
        catch(ArrayIndexOutOfBoundsException f) //AIOBE f=new AIOBE()
        ^
1 error
```

finally block

- finally is a block which will get executed irespective of
  1.exception occurred or not
  2.exception occurred and handled
  3.exception occurred and not handled

1.exception occurred or not :

```
class Sample
{
      public static void main(String args[])
      {
            int d[]=new int[3];
       try
       {
        d[2]=55;
       }
       catch(ArrayIndexOutOfBoundsException f)//AIOBE f=new AIOBE()
       {
         System.out.println("Exception caught for array");
        }
       finally
       {
         System.out.println("Finally block");
       }
      }
}
```

Output:

```
Finally block
```

2.exception occurred and handled

```
class Sample
{
      public static void main(String args[])
      {
            int d[]=new int[3];
       try
       {
```

```
            d[3]=55;//beyond declared size
           }
           catch(ArrayIndexOutOfBoundsException f)//AIOBE f=new AIOBE()
           {
             System.out.println("Exception caught for array");
           }
           finally
           {
             System.out.println("Finally block");
           }
          }
     }
```

Output:

```
Exception caught for array
Finally block
```

3.exception occurred and not handled

```
     class Sample
     {
          public static void main(String args[])
          {
               int a=10,b=0,c;
               int d[]=new int[3];
           try
           {
            d[3]=55;//beyond declared size
           }
           finally
           {
             System.out.println("Finally block");
           }
          }
     }
```

Output:

```
Finally block
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
Index 3 out of bounds for length 3
        at Sample.main(Sample.java:9)
```

- Basically, "finally" is used to keep an important code which should not be skipped at any condition like closing of data base connection or closing of opened file etc.

Valid Combinations

```
1.    try{}
      catch{}
      finally{}

2.    try{}
      finally{}

3.    try{}
      catch{}
      catch{}
      finally{}

4.    try{}
      catch{}
      finally{}
      try{}
      catch{}
      finally{}
```

Invalid Combinations

```
1.    try{}
      catch{}
      finally{}
      finally{}

2.    catch{}
      finally{}

3.    finally{}
      try{}
      catch{}
```

24-09-2020

Throws Keyword

Example of checked exception :

```
class Test
{
      public static void main(String args[])
      {
            System.out.println("Go to sleep");
            Thread.sleep(1000);
            System.out.println("Awake");
}}
```

- In the above program we have called sleep() of thread class means, we are making current thread i.e, main thread to go in sleepng state. Which means main() should stop execution. Once sleep() is invoked for given amount of time i.e, 1000 milliSeconds.
- But, When it is in sleeping state, there will be a chance of other threads trying to interupt main thread from sleeping. So, that is why for above program we will get the Exception as

```
      Test.java:6: error: unreported exception InterruptedException;
must be caught        or declared to be thrown
                Thread.sleep(1000);
                           ^
                      1 error
```

- i.e, there is an chance of Exception to occur which we didn't reported. So, it is our responsiblity to report that exception by menas of try, catch or throws.

Conclusion :

In case of checked Exception, user has 2 options -

1. Caught the Exception : use try and catch block
2. Declare the Exception : use throws keyword

- throws keyword is used to declare or report a checked Exception.
- throws keyword is used with method declaration.

```
      public void fly() throws ExceptionName
```

- When we use throws keyword, we are indicating that current method will not handle exception rather calling method or its caller will handle the exception.

```
class Test
{
      public static void main(String args[]) throws InterruptedException
      {
            System.out.println("Go to sleep");
            Thread.sleep(1000);
            System.out.println("Awake");
}}
```

Output:

```
Go to sleep
---wait for 1sec---
Awake
```

- In the above program, main method declared an Exception using throws keyword that means it is telling that i won't handle exception rather it is responsible of my caller to handle it.
- So, in above program the caller of main() is JVM. And, this process where current method is not handling exception and telling caller to handle, it is called as Exception Propagation.

```
class Sample                //Ex
{
      public static void cry() throws InterruptedException //exception
propagated caller
      {
          Thread.sleep(1000);//when this statement execute make current
thread to go in                         //sleeping state(stop execution)
         }
        public static void main(String args[])
      {
        System.out.println("In main");
        try
           {
          cry();
         }
        catch(InterruptedException e)
        {
            }
}}
```

Output:

```
In main
----Execution ends in one sec------
```

```
class Demo      //Ex
{
      public static void main(String args[])
      {
            System.out.println(10/0);
}}
```

- In above program, main methods create an exception object and handover it to JVM. Since, there is no or won't be exception handling code. JVM will handover that object to default exception handler and it points exception message as -

  Exception in thread "main" java.lang.ArithmeticException: / by zero at Demo.main(Demo.java:5)

- But, if we want, we can create our own exception object by using throw keyword.
- throw keyword is used to explicitly creating an exception object.
- Syntax:

  throw new ExceptionType(description of exception);
- throw keyword is mainly used for userdefine exceptions.

```
class Demo      //Ex
{
      public static void main(String args[])
      {
            throw new ArithmeticException("My Exception");
}}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: My Exception
        at Demo.main(Demo.java:5)
```

Note:

- Once we throw the exception object explicitly we should not give any printing statementfurther after that statement, if we give we will get compile time error.
- Unreachable statement

```
-----------------------
class Demo            //Ex
 {
      public static void main(String args[])
       {
            throw new ArithmeticException("My own Exception
occurred");
            System.out.println("something");//Unreachable statement
      }}
```

Output:

```
Demo.java:6: error: unreachable statement
            System.out.println("something");//Unreachable statement
            ^
      1 error
```

```
class Demo            //Ex
{
  public static void check(int age){
   if(age<18){
     throw new ArithmeticException("He is small kid - enjoying life");
    }
   else
   {
     System.out.println("cast a Vote");
 }}
  public static void main(String args[])
  {
      check(15);
 }}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: He is small kid
- enjoying life
        at Demo.check(demo.java:5)
        at Demo.main(demo.java:13)
```

User define Exception (or) Customised Exception

When pre-define exceptions does not fulfil our requirement, we will go for user-define exception i.e we can create our exceptions. Such type of exception is called as User define Exceptions (or) Customised Exceptions.

Rules for creating User define Exceptions:-

- create our Exception class and that class should be or must be extending either throwable (or) Exception (or) RuntimeException classes
  Preferable to extend RuntimeException
- Define constructor whenever requires.
- Throw Exception as per our own requirement.

```
class NotEligibleException extends RuntimeException         //Ex-1
{
 public NotEligibleException(String msg)
 {
  System.out.println(msg);
```

```
 }
}
public class User {
public static void main(String args[]) {
      float percentage=56.5f;
      if(percentage<60)
      {
        throw new NotEligibleException("Not Eligible for drive");
      }
      else
      {
         System.out.println("Register before end of the  day");
      }
}}
```

Output:

```
E:\studies\Qspiders\Qspiders Java\programs>java User
Not Eligible for drive
Exception in thread "main" NotEligibleException
        at User.main(User.java:13)
```

Explanation:

- Execution started from main method, under that i gave condition, if that condition satisfied, i am creating an explicit Exception object with details as-
  Name: NotEligibleException
  Description: NotEligible for drive
  Location: User.main
- Once Exception objects created it calls to NotEligibleException constructor and pass msg as argument
- under that constructor we print msg, so it prints that information.

```
class AgeGapException extends RuntimeException   //Ex-2
{
 public AgeGapException(String msg)
 {
  System.out.println(msg);
 }}
public class Demo
{
  public static void main(String args[]) {
   int bage=10,gage=27;
   if(bage<15 && gage>=27) {
     throw new AgeGapException("He is small kid enjoying life");
    }
   else
```

```
   {
     System.out.println("Go out with girl friend and destroy life");
   }
}}
```

Output:

```
E:\studies\Qspiders\Qspiders Java\programs>java Demo
He is small kid enjoying life
Exception in thread "main" AgeGapException
        at Demo.main(Demo.java:12
```

Explanation:

- Execution started from main method, under that i gave condition, if that condition satisfied, i am creating an explicit Exception object with details as-
  Name: AgeGapException
  Description: He is small kid enjoying life
  Location: Demo.main
- Once Exception objects created it calls to AgeGapException constructor and pass msg as argument
- under that constructor we print msg, so it prints that information.

**Differences between throw and throws

throw	throws
throw keyword is used to create Exception object explicitly	throws is used to declare the Exception
throw keyword is used inside the method	throws keyword is used with method declaration

Syntax:
throw new ExceptionName(Excp description);
Ex:
throw new ArithmeticException("MyExcept");
	Ex:
	method declaration Exceptionname
	public void fly() throws InterruptedException

throw keyword is mainly used for Userdefine exception	throws keyword is mainly used for checked exception
Using throw keyword, we can throw only one exception at a time	Using throws keyword, we can declare multiple exceptions at a time
throw new MinBalException("Zero");	public void check() throws InterruptedException,SQLException

Q. WAP to demonstrateuser defne exception
create InsufficientBalanceException class
create cust class
if(withdrawlamt > avabal)---->throw InsufficentBalException
else------------>collect the amt

```
import java.util.Scanner;
class InsufficientBalException extends RuntimeException
{
 public InsufficientBalException(String msg)
 {
  System.out.println(msg);
 }}
public class Customer
{
  public static void main(String args[]) {
   System.out.println("Enter available balance in account : ");
   Scanner s=new Scanner(System.in);
   double avlbal=s.nextDouble();
   System.out.println("Available balance : "+avlbal);
   System.out.println("Enter the amount to withdraw : ");
   double wdramt=s.nextDouble();
   if(wdramt>avlbal) {
     throw new InsufficientBalException("Exceeding the limit in the
account");
    }
   else
   {
     System.out.println("Collect the amount");
   }
}}
```

Output:

```
E:\studies\Qspiders\Qspiders Java\programs>java Customer
Enter available balance in account :
5000.87
Available balance : 5000.87
Enter the amount to withdraw :
8470.25
Exceeding the limit in the account
Exception in thread "main" InsufficientBalException
        at Customer.main(Customer.java:18)
```

25-09-2020

Multithreading

Introduction :

Multitasking

The process of performing more than one task parallelly is called as Multitasking.
It is of 2 types -
1.Process based multitasking
2.Thread based multitasking

Process based multitasking

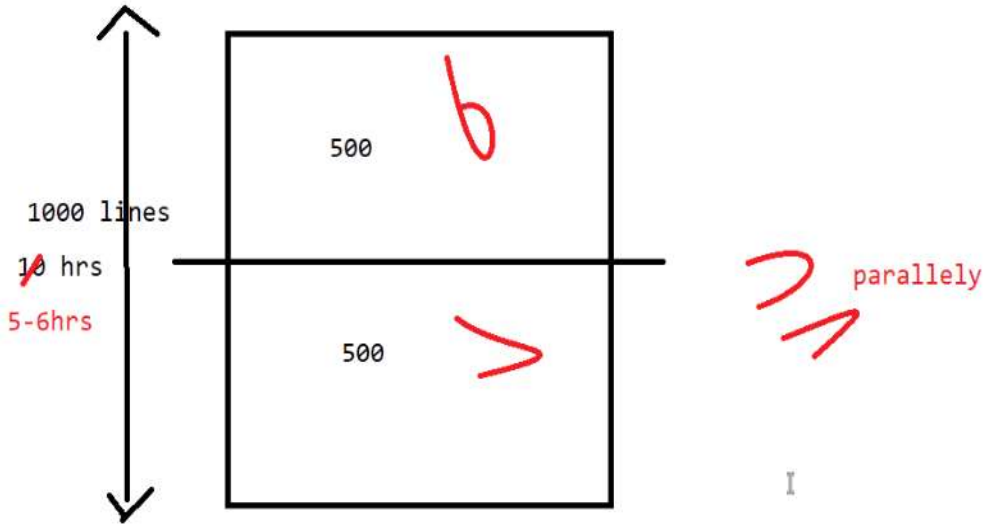
The process of executing more than one task where each task is independent of other.

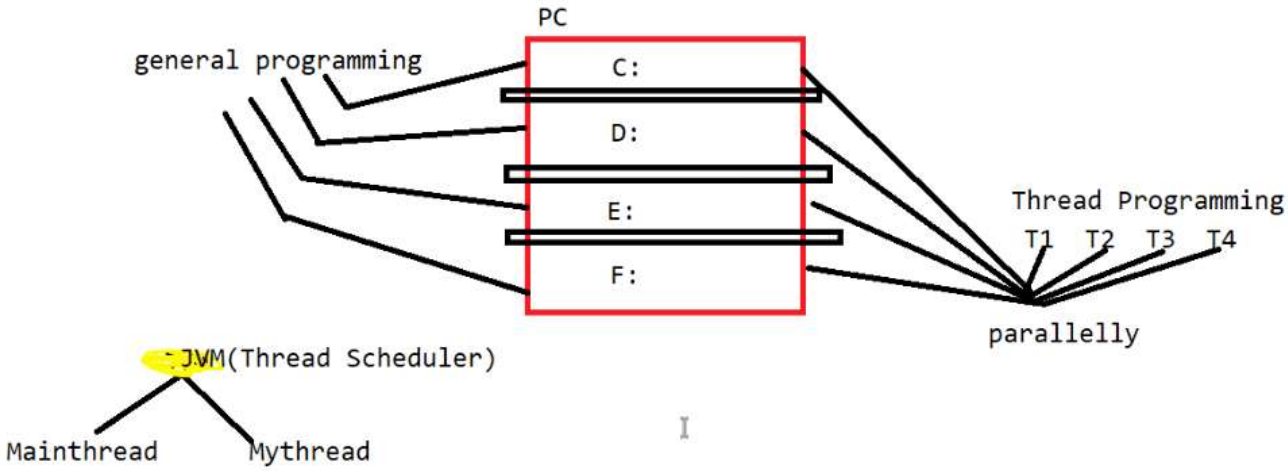
Ex: browsing on google----Task-1
watching something on youtube----Task-2
using excel----Task-3
typin something on word----Task-4

Thread based multitasking

- The process of executing more than one task parallelly where each task is an independent part of same program such type of processing is called as thread based multitasking or thread programming.

Ex: If we have a program which contains 100lines of code but what we observed is next 500 is independent of first 500, but still it has to wait until first 500 finishes its execution so due to this,
-performance is decreasing
-execution time is increasing
-CPU utilisation time is decreasing.





- Therefore, in order to overcome above situations, we will go for Thread programming.

Thread

- it is a flow of execution (or)
- it is a small part of an Application (or)
- it is a light weight process

- We will create 1-thread for first 500lines
- We will create 2-thread for second 500lines and run both the threads parallely

- Programs which contains multiple threads is called as multi threaded program and such process is called as Multithreading.

Creation of Thread

A thread can be created in two ways -

1. By extending Thread class
2. By implementing Runnable Interface

Note:

In every program always therre is one default thread
i.e main thread(main())

Creating Thread by Extending Thread class

- Create our class which extends Thread class
- For defining thread we have to override run()
  public void run();
- it is a predefine method of thread class
- Using start() we can start execution of thread
  public synchronized void start();

Note:

Synchronized is a keyword which indicates that only thread can access method at a time.

```
//step-1: create our own class and extends Thread class
class Mythread extends Thread
{
      //step-2: Override run() of Thread class
      public void run()
      {
            //step-3: define job of thread
            for(int i=1;i<=5;i++)
            {
             System.out.println("Mythread 1.0");
           }
}}
public class User {
public static void main(String[] args) throws InterruptedException
      {
        Mythread t1=new Mythread();
        t1.start();//making my thread ready for execution by calling run()
```

```
        Thread.sleep(5000);
        for(int i=1;i<=5;i++)
        {
            System.out.println("Mythread 2.0");
        }
}}
```

Output:

```
Mythread 1.0
Mythread 1.0
Mythread 1.0
Mythread 1.0
Mythread 1.0
----wait for 5 sec----
Mythread 2.0
Mythread 2.0
Mythread 2.0
Mythread 2.0
Mythread 2.0
```

28-09-2020

Creating thread by extending thread class

Example program

```
package Multithreding.com;
class Mythread extends Thread
{     //@override run() of thread class for defining thread
      public void run()
      {
            //JOB of thread
       for(int i=0;i<=5;i++)
       {
        System.out.println("MyThread");
       }
}}
public class User {
      //default thread of everyprogram
public static void main(String[] args) {
//@job of main() thread
      Mythread t1=new Mythread();
      t1.start();//creates a thread by calling run() and make it available
                  //for execution
                  //after this statement there are two threads under
                  //execution, main and mythread
      for(int i=0;i<=5;i++)
      {
       System.out.println("Main  Thread");
      }
}}
```

Explanation

- In above program execution starts from main thread and till t1.start() there is only one thread under execution once t1.start() is invoke it creates a thread and calls run()
- Now two threads are there for execution main thrread and mythread
- So out of two thread which thread to select first for execution is decided by THREAD SCHEDULER.
- THREAD SCHEDULER is nothing but JVM and it is upto JVM whichever algorithm it follows and select a thread for execution.
- Since we don't know on what basis thread scheduler picks a thread for execution we cannot predict the output of program.

Creating a thread by implementing runnable interface :-

Runnable Interface

```
public interface java.lang.Runnable {
public abstract void run();
}
```

Example Program

```
package Multithreding.com;
class Mythread implements Runnable
{
 //@override run() of thread class for defining thread
      public void run()
      {
        //JOB of thread
      for(int i=0;i<=5;i++)
      {
        System.out.println("MyThread");
      }
}}
public class User {
      //default thread of every program
public static void main(String[] args) {
      //@job of main() thread
      Mythread t1=new Mythread();
     //t1.start();//CTE because there is no start() in Mythread class
     Thread t2=new Thread(t1);
     t2.start();//creates a thread by calling run()
//after this stmt ther are two threads under exe main amd mythread
      for(int i=0;i<=5;i++)
      {
       System.out.println("Main  Thread");
      }
}}
```

Q. Out of two ways of creating a Thread which one is preferrable?

A. Second way of thread creation is preferrable because when we create a thread by implementing Runnble interface at same time we can extend any other base class also but if we create a thread by extending thread class sub class cannot extend any other class.

Multiple independent threads

```
------------------------------
package Multithreding.com;
public class User1 {
public static void main(String[] args) {
Mythread1 t=new Mythread1();
Mythread11 t1=new Mythread11();
Mythread12 t2=new Mythread12();
t.start();
t1.start();
t2.start();
for(int i=0;i<5;i++)
{
System.out.println("Poooja 1.0");
}
}}
class Mythread1 extends Thread
{
public void run()
{
for(int i=0;i<5;i++)
{
System.out.println("Poooja 2.0");
}
}}
class Mythread11 extends Thread
{
public void run()
{
for(int i=0;i<5;i++)
{
System.out.println("Poooja 3.0");
}
}}
class Mythread12 extends Thread
{
public void run()
{
for(int i=0;i<5;i++)
{
System.out.println("Poooja 4.0");
}
}}
```

Output:

```
Poooja 1.0
Poooja 1.0
Poooja 1.0
Poooja 1.0
Poooja 1.0
Poooja 4.0
Poooja 4.0
Poooja 4.0
Poooja 4.0
Poooja 4.0
Poooja 3.0
Poooja 3.0
Poooja 3.0
Poooja 3.0
Poooja 3.0
Poooja 2.0
Poooja 2.0
Poooja 2.0
Poooja 2.0
Poooja 2.0
```

```
Execution
-------------
main--------------->task-1

Mythread1--------->task2

Mythread11----------->task3

Mythread12----------->task4
```

Q. Can we Resart a thread?

A. No We cannot restart a thread,if we do we will get IllegalThreadStateException

```
   Ex:MyThread m1=new MyThread();
   m1.start();//Valid
   m1.start();//Exception
Exception in thread "main" java.lang.IllegalThreadStateExceptionMyThread
at java.lang.Thread.start(Unknown Source)
```

Internal Implementation

```
--------------------------
interface Runnable
{
public abstract void run();
}
public class Thread implements Runnable
{
public void run()
{
// No implementation
}
}

class MyThread extends Thread
{
public void run()
{
//define job of thread//
}
}
```

Single class multiple tasks

```
-----------------------------
class Myth implements Runnable
{
      //2.Override run() of Thread class
      public void run()
      {
            //3.define job of thread
             set();
             get();
      }
      public void get()
      {
            for(int i=1;i<=3;i++)
             {
                  System.out.println("Myth 1.0");
             }
      }
      public void set()
      {
            for(int i=1;i<=3;i++)
             {
                  System.out.println("Myth 1.1");
             }
```

```
	}
}
public class User1 {
	public static void main(String[] args) {
		Myth t1= new Myth();
	    Thread t2=new Thread(t1);
	    t2.start();//making my thread ready for execution by calling
		  //run()
	   for(int i=1;i<=3;i++)
	    {
		  System.out.println("Main Thread-2.0");
	    }
	}
}
```

Life Cycle of Thread

Depending on different phases a Thread will be in any of one phase:

1.New
2.Runnable/Ready
3.Running
4.Blocked
5.Terminated

New:-

When we create object of our thread class

```
MyThread m1=new Mythread();
```

Ready:-

When we call run() by using start(),but before thread schedule picks that thread for execution

```
m1.start();
```

Running:-

When thread scheduler(CPU) picks thread for execution(run() execution started)

Blocked:-

If the running thread goes to sleeping state.

Terminated or dead:-

When execution of run() is completed.

29-09-2020
---

wrapper classes :

- In order to represent primitive datatype as an object we use wrapper classes
- For every primitive data type, there is a corresponding wrapper class available in java.
- Since java is 100 % object oriented, we have to represent everything as object
- Therefore to represent primitive datatype as object we use wrapper classes.

primitive datatype	wrapper class
byte	Byte
boolean	Boolean
char	Character
short	Short
float	Float
int	Integer
double	Double
long	Long

- All wrapper classes are final classes.
- All wrapper classes are present in java.lang package.

Examples

```
int a=100;//primitive data type
Integer a=new Integer(100);        (or) Integer a=100;//Objects

double d=22.45;//primitive data type
Double d=new Double(222.45);      (or) Double d=445.5;//Objects
```

Constructors

```
Integer a=new Integer(int);
Integer a1=new Integer(String);
byte b1=new Byte(byte);
byte b2=new Byte(String);
and so on ..............
```

- In all Wrapper classes toString() is overridden(defaultly) and it

prints value of object not complete information of object.

```
Integer a=new Integer(100);
System.out.println(a.toString());//100
System.out.println(a);//100
```

- Using Wrapper classes we can perform boxing and Unboxing.

Boxing

- The process of converting primitive type into Wrapperclass is called as Boxing.
- Boxing can be performed by using valueOf()

```
Ex:  int a=990;
     Integer i=Integer.valueOf(a);
```

```
public class Wrap1 {
      public static void main(String[] args) {
      int a=300;                                          Output:
      Integer box=Integer.valueOf(a);                     300
      System.out.println(box.toString());
}}
```

UnBoxing

- The process of converting Wrapperclass into primitive type is called as UnBoxing.
- UnBoxing can be performed by using xxxvalue()(xxx--->primitivedatatype)

```
Ex:   Integer i=100;
      int a=i.intValue();
      Double d=22.23;
      double a2=d.doubleValue();
```

```
public class Wrap2 {
      public static void main(String[] args) {
            Integer i=new Integer(100);           Output:
            int a=i.intValue();                   100
            System.out.println(a);
}}
```

Note:

From 1.5 version of java we can perform boxing and Unboxing directly, so they are reffered as autoboxing and autounboxing respectively.

```
   Ex-1:  int i=100;
          Integer a=i;///Autoboxing

   Ex-2:  Integer i=122;
          int k=i;//AutoUnboxing

public class Wrap3 {
     public static void main(String[] args) {
     //Autoboxing
       int a=100;
       Integer i=a;                                  Output:
       System.out.println(i.toString());             100
     //AutoUnboxing                                  300
       Integer j=300;
       int k=j;
       System.out.println(k);
}}
```

parse()

- It is a method of Wrapper class, it converts the string data into int, double or float.
- But will work only, if string holds proper data other wise we will get NumberFormatException

```
  For Ex: String s="123";String s1="223.33";
          int i=Integer.parseInt(s);//Converts string into integer
          double d=Double.parseDouble(s1);//converst string to double
          System.out.println(i+d);//123+223.33

public class Wrap4 {
     public static void main(String[] args) {
           String s="123";String s1="223.33";
           int i=Integer.parseInt(s);//Converts string into integer
           double d=Double.parseDouble(s1);//converst string to double
           System.out.println(s+s1);//123223.33
           System.out.println(i+d);//123+223.33
}}
```

Output:

```
123223.33
346.33
```

Q. There are two inputs of String type we need to add them and result should be a sum ?

A. Using parse()

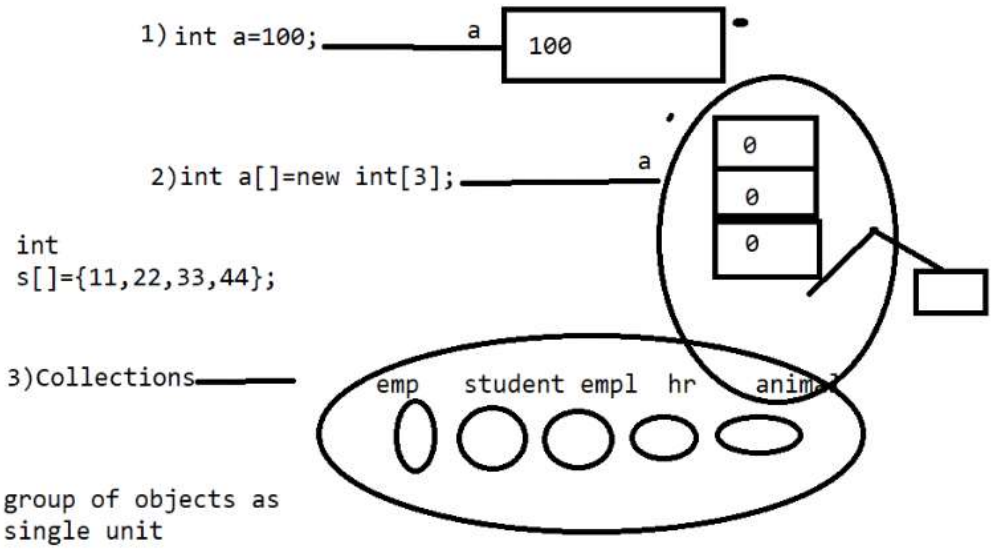
30-09-2020

COLLECTION FRAMEWORKS

Arrays	Collections
Arrays are used to store collection of homogeneous(similar) data.	Collections is used to store heterogeneous data as well as homogeneous data.
Arrays are fixed in size. int a[]=new int[3];	Collections are numerous (growable in size).
Arrays can deal with primitive as well as wrapper classes int a[]; Integer a[];	collections purely deals with wrapper classes(Objects). ArrayList<int>a1;(Invalid) ArrayList<Integer>a1;(Valid)
Arrays does not have any underlying data structure.	Every classes of Collection have data structure.
Arrays does not contains predefine methods(add, sorting, removing, replacing) which makes manipulation of data difficult.	In collection 80% support is by predefine API's(App Programming Interface).
We should use arrays, when we already knows the elements in advance.	We should go for collection, when we don't know the elemens in advance.
Memory wise array is not preferred.	Memory wise collections is prefered.
Perfomance wise arays are preferred.	Performance wise collection is not preferred.

COLLECTION

- Collection is nothing but a group of objects repesents as single unit.
- It's main purpose is to store huge amount of data.
- It provides multiple API(methods) to store and manipulate data.



CollectionsFrameworks

- It is an Architecture, which provides the group of classes and interfaces, which is used to store multiple objects as single unit.

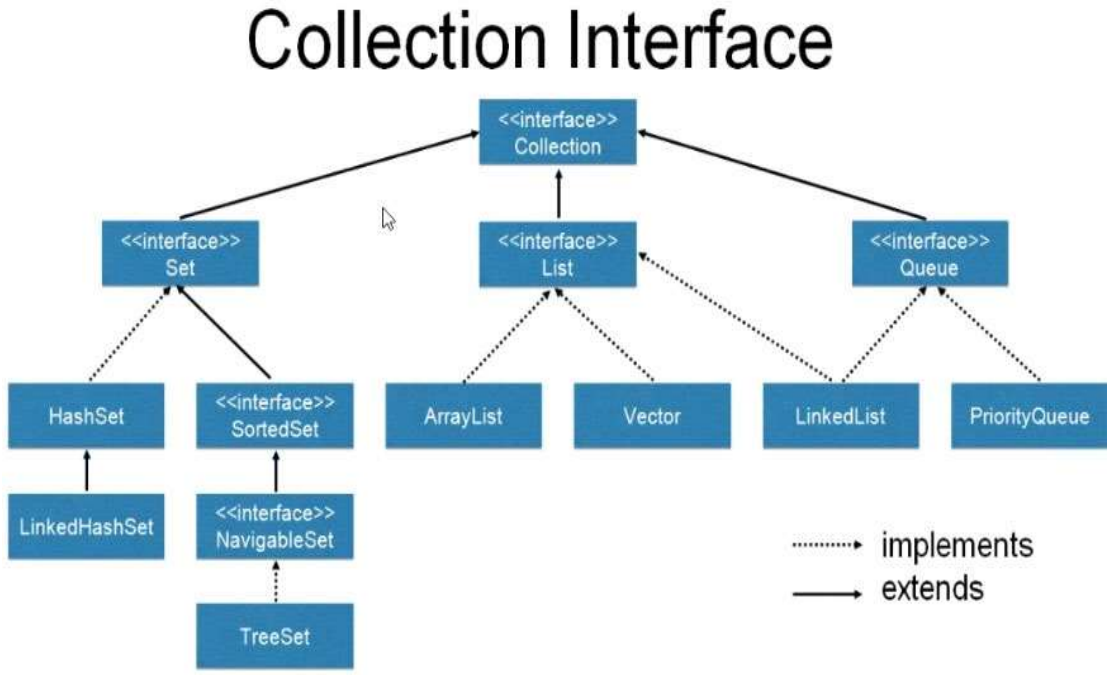
- CollectionsFramework--------->concept
- Collection------------>Root Interface
- collections------------------>class

- All the classes and interfaces of CollectionsFramework are present in java.util package.

01-10-2020

Collection Hierarchy

- Collection is the root interface of CollectionFramework.
- If you want to see methods of Collection interface enter cmd as javap java.util.Collection
- All the classes which are implementing collection interface can use these methods



- Collection interface is extended by 3 different interfaces
    1. List<Interface>
    2. Set<Interface>
    3. Queue<Interface>
- Learning about collection means learning about classes and its characteristics.
- Different classes of collection are :
    1. ArrayList
    2. LinkedList
    3. Vector
- implements List Interface
    4. PriorityQueue
    5. LinkedList

- implements Queue interface
  6.Treeset
  7.Hashset
  8.LinkedHashSet
- implements SetInterface

characteristics to study for every class -

- version
- homogeneous or hetrogeneous
- Null object is possible or not
- duplicates allowed or not
- insertion order is preserve or not
- data structure
- cursors

List Interface :

- List<Interface>1.2 version
- List is an Interface which extends Collection Interface
- List is implemented by 3 different classes
  1.ArrayList
  2.LinkedList
  3.vector

-javap java.util.List

C:\Users\Ravi Kiran>javap java.util.List
Compiled from "List.java"

List follows Index Based process.
List allows hetrogeneous Objects.
List allows Null objects.

Useful Methods :

- add(object):Used to add objects
- addAll(Collection): Used to copy one collection objects into another
- remove(Object): Used to Removes object
- remove(Index): Used to Remove the Object at given index
- set(Index,Object): it replaces the old object with new object at given index
- get(index): Retrieves the object at given index

- isEmpty(): check whether Collections empty or not
- contains(Object): check whether Object is present or not
- containsAll(Collection): checks whether content of one collection is present in another collection or not.
- removeAll(Collection): Removes whole objects of collection
- size(): Provides size of collection(always calculates from 1)
- Collections.sort(): Sort the collection in ascending order(In case char's as per unicode order)
- Collections.reverse(): Sort in reverse way

1.ArrayList :

- ArrayList is a class which implements List interface
- For checking methods
  javap java.util.Arraylist

- characteristics of ArrayList:

1.It is introduced in java 1.2-version.
2.ArrayList stores Heterogeneous data .
3.It is possible to add NULL objects in ArrayList.
4.It allows Duplicate objects.
5.In ArrayList Insertion Order is preserved i.e the way we added objects the way it will be printed.
6.It follows Data Structure as growable size array.
7.Iterator and ListIterator cursors are used.

```
import java.util.ArrayList;          //Ex
public class First {
      public static void main(String[] args) {
            ArrayList a1=new ArrayList();
            a1.add("Java");
            a1.add(111);       //a1.add(Integer.valueOf(111))
            a1.add(66.5f);     //a1.add(Float.valueOf(66.5f))
            a1.add('A');       //a1.add(Character,valueOf('A');
            System.out.println("Collection Objects are : "+a1);
      }}
```

Output:

```
Collection Objects are : [Java, 111, 66.5, A]
```

02-10-2020

```java
import java.util.ArrayList;      //Ex-1
public class First {
	public static void main(String[] args) {
		ArrayList a1=new ArrayList();
		a1.add("Java");
		a1.add(111);     //a1.add(Integer.valueOf(111))
		a1.add(66.5f);   //a1.add(Float.valueOf(66.5f))
		a1.add('A');     //a1.add(Character,valueOf('A');
		System.out.println("Collection Objects are : "+a1);
		a1.add("Java");
		Integer i=new Integer(100);
		a1.add(i);
		System.out.println("After adding duplicates : "+a1);
		a1.add(null); //null objects means programmer is just reserving
				//memory
		System.out.println("After adding null : "+a1);
		a1.remove("Java");
		System.out.println("After removing : "+a1);
		a1.remove(i);
		System.out.println("After removing numeric object : "+a1);
		System.out.println(a1.get(4));
		a1.set(2,"SQL");
		System.out.println("After setting object :"+a1);
		a1.add(1,"Full Stack JAVA");
		System.out.println("After adding at particular index :"+a1);
		System.out.println(a1.size());
		ArrayList a2=new ArrayList();
		a2.addAll(a1);
		System.out.println("After copying :"+a2);
		System.out.println(a2.contains("SQL"));
		System.out.println(a1.isEmpty());
		//printing using for loop
			for(int j=0;j<a1.size();j++) {
				System.out.println(a1.get(j));
			}
}}
```

Output:

```
Collection Objects are : [Java, 111, 66.5, A]
After adding duplicates : [Java, 111, 66.5, A, Java, 100]
After adding null : [Java, 111, 66.5, A, Java, 100, null]
After removing : [111, 66.5, A, Java, 100, null]
After removing numeric object : [111, 66.5, A, Java, null]
```

```
null
After setting object :[111, 66.5, SQL, Java, null]
After adding at particular index :[111, Full Stack JAVA, 66.5, SQL, Java,
null]
6
After copying :[111, Full Stack JAVA, 66.5, SQL, Java, null]
true
false
111
Full Stack JAVA
66.5
SQL
Java
null
```

- Arrays are type safe but collections are not type safe

```
Ex:   int a[]=new int[3];
      a[0]=100;
      a[1]="java";//CTE

Ex:   ArrayList a=new ArrayList()
      a.add("java");
      a.add(100);
      a.add('A');
```

Generics

Since Collections are not type safe i.e programmer can add any types of objects there is no restriction for that so, to overcome this drawback JAVA has given Generics concept where you can create an object of specific type, if we add any other type of object than specified we will get CTE.

```
1. ArrayList<Integer> a1=new ArrayList<>();
      a1.add("java");//CTE
      a1.add(100);
      a1.add('A');//CTE

2. ArrayList<String> a1=new ArrayList<>();
      a1.add("java");
      a1.add(100);//CTE
      a1.add('A');//CTE

3. ArrayList<Character> a1=new ArrayList<>();
      a1.add("java");//CTE
      a1.add(100);//CTE
      a1.add('A');
```

```java
import java.util.*;                    //Ex-2
public class Second {
	public static void main(String[] args) {
		ArrayList<Integer> a1=new ArrayList<>();
		a1.add(111);
		a1.add(222);
		System.out.println("Integer array list : "+a1);
		/*a1.add('A');
		a1.add("Java");*/
		ArrayList<String> a2=new ArrayList<>();
		a2.add("java");
		a2.add("sql");
		System.out.println("String array list : "+a2);
		ArrayList<Object> a3=new ArrayList<>();
		a3.addAll(a1);
		a3.addAll(a2);
		System.out.println("Object array list : ");
		//for each loop
		for(Object data:a3)
		{
			System.out.println(data);
		}
}}
```

Output:

```
Integer array list : [111, 222]
String array list : [java, sql]
Object array list :
111
222
java
sql
```

03-10-2020

Cursors

- This is the special characteristics given for collection concepts
- Using cursors we can retrieve the objects in collection one by one
- they are of 2 types -
    1.Iterator
    2.ListIterator

1.Iterator

- Iterator is an interface which is used to traverse the list in forward direction.
- Basically it provides the privilige to access the objects without using index.

```
interface Iterator
{
  public boolean hasNext();
  public object next();
  public void remove();
}
```

hasNext():

- It returns true, if there is Object available in collection.

next():

- It returns current Object and move the cursor to next Object.

2.ListIterator

- ListIterator is an interface which provides the facility to traverse the list in forward as well as backward direction.

```
interface ListIterator
{
 public boolean hasNext();
 public object next();
 public void remove();
 public boolean hasPrevious();
 public object previous();
 public void add();
```

}

hasPrevious():

- Returns true, if object is available to iterate from previous direction.

previous():

- prints the current object and move the cursor to next object in previous direction.

hasNext():

- It returns true, if there is Object available.

next():

- It returns current Object and move cursor to next Object.

```
import java.util.*;                    //Ex-1
public class Third {
	public static void main(String[] args) {
		ArrayList<String> a1=new ArrayList<>();
		a1.add("Java");
		a1.add("SQL");
		a1.add("Selenium");
		a1.add("Aptitude");
		System.out.println("Objects from forward direction");
		//toString(),foreachloop,forloop
		//(listIteraot()=new ListIterator())
		//using Listiterator interface
		ListIterator i1=a1.listIterator();
		while(i1.hasNext())
		{
			System.out.println(i1.next());
		}
		System.out.println("Object from backward direction");
		while(i1.hasPrevious())
		{
			System.out.println(i1.previous());
		}
}}
```

Output:

```
Objects from forward direction
Java
SQL
Selenium
Aptitude
```

```
Object from backward direction
Aptitude
Selenium
SQL
Java
```

```java
import java.util.*;                    //Ex-2
public class Fourth {
      public static void main(String[] args) {
            ArrayList<String> a1=new ArrayList<>();
            a1.add("Java");
            a1.add("SQL");
            a1.add("Selenium");
            a1.add("Aptitude");
            Collections.sort(a1);//it is method of collections class
            //which sort the list in unicode order ascending
            System.out.println("Increasing sorted order : "+a1);
            Collections.reverse(a1);
            System.out.println("Reverse sorted order : "+a1);
}}
```

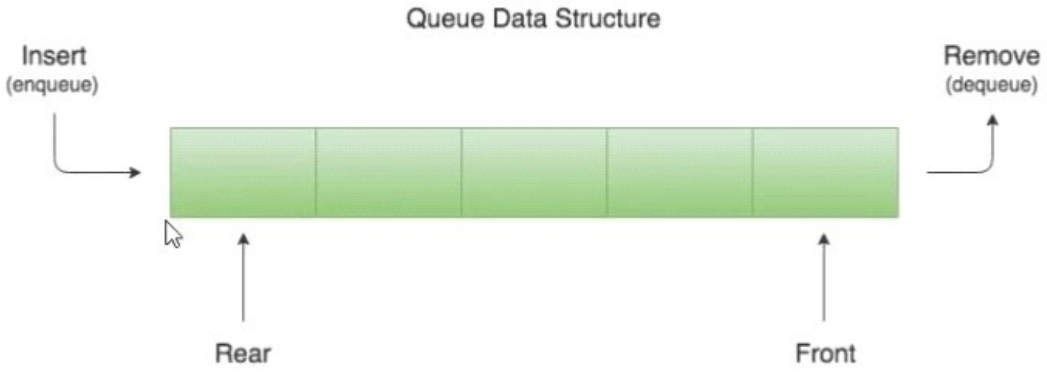
Output:

```
Increasing sorted order : [Aptitude, Java, SQL, Selenium]
Reverse sorted order : [Selenium, SQL, Java, Aptitude]
```

05-10-2020

Queue Interface

- Queue is an Interface which is implemented by 2 different classes -
  1. PriorityQueue
  2. LinkedList
- Queue is a data structure which usually follows FIFO principle(First in First out)



1.PriorityQueue

- Introduced in 1.2 version.
- Default capacity is 11.
- It allows only homogenous objects.
- Bascially data structure of Queue is FIFO but in PriorityQueue internal sorting will happen.
- Duplicate objects are allowed.
- Null objects are not allowed.
- For retrieving of data, they have given the special methods called as peek() and poll().

peek():

- It retrieves the head element(top element) without deleting it from queue.
- It returns null, if there is no head element present.

poll():

- It retrieves head element(First element) and delete it from queue.

```java
import java.util.*;                    //Ex
public class Fifth {
	public static void main(String[] args) {
		PriorityQueue<Object> q1=new PriorityQueue<>();
		q1.add("JAVA");
		q1.add("HTML");
		q1.add("CSS");
		q1.add("SOAP");
		q1.add("AWS");
		q1.add("Selenium");
		System.out.println(q1.peek());
		System.out.println(q1.poll());
		System.out.println(q1.contains("AWS"));
		while(q1.peek()!=null)
		{
			System.out.println(q1.poll());
		}
		System.out.println("After while loop : "+q1);
}}
```

Output:

```
AWS
AWS
false
CSS
HTML
JAVA
SOAP
Selenium
After while loop : []
```

2.LinkedList

- LinkedList is a class which implements List as well as Queue Interface.
- class LinkedList implements List,Queue
- Queue q1=new LinkedList();
- Here q1 will exhibits behaviour of queue.

Note:

- All features are same, only difference is LinkedList implementing Queue interface.
- Objects will not be sorted rather insertion order is maintained.

```
import java.util.*;                    //Ex
public class Sixth {
      public static void main(String[] args) {
            Queue<Object> q1=new LinkedList<>();
            q1.add("JAVA");
            q1.add("HTML");
            q1.add("CSS");
            q1.add("SOAP");
            q1.add("AWS");
            q1.add("Selenium");
            while(q1.peek()!=null)
            {
                  System.out.println(q1.poll());
            }
}}
```

Output:

```
JAVA
HTML
CSS
SOAP
AWS
Selenium
```

Set Interface

- Set is an interface which is implemented by 3 different classes -
  1. HashSet
  2. Treeset
  3. LinkedHashSet
- Set interface does not have index based process.
- Default capacity is 16.
- Duplicate objects are not allowed.
- Set is only uni directional, so it supports only Iterator interface(cursor).
- Methods of set interface -

```
javap java.util.set
```

1.HashSet

- It is a class which implements set interface.
- Introduced in 1.2 Version
- Hetrogenous objects are allowed.
- Duplicates are not allowed. In case if we add, we won't get compile time error, it will just add once.
- Only one NULL object is allowed.
- Data structure is hashtable.
- Insertion order is preserved(depends on hashcode number).

```
import java.util.*;                    //Ex
public class Seventh {
      public static void main(String[] args) {
            HashSet s1=new HashSet();
            s1.add("java");
            s1.add(1223);
            s1.add("selenium");
            s1.add(88.9f);

            Iterator i1=s1.iterator();
            while(i1.hasNext())
            {
                  System.out.println(i1.next());
            }
}}
```

Output:

```
java
selenium
1223
88.9
```

06-10-2020

** Differences between Iterator and ListIterator

Iterator	ListIterator
It is an interface used to retrieve objects in forward direction.	It is an interface used to retrieve objects in forward as well as backward direction.
It is used for all collection classes.	It is only used for classes which implements ListInterface - Arraylist, LinkedList, Vector
Iterator object, we will get by using iterator()	Listiterato object, we will get by using listIterator()
Using iterator interface methods, we can traverse only in forward direction i.e hasNext(), next()	Using ListIterator methods, we can traverse in both forward and backward direction hasNext(), next(), hasPrevious(), previous()

LinkedList

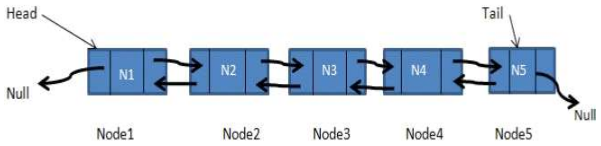
- Introduced in 1.2 version.
- Hetrogenous objects are allowed.
- Null objects are allowed.
- Insertion order is preserved.
- Duplicate objects are allowed.
- Data structure is doubly LinkedList.
- cursors - ListIterator and Iterator
- Same process to write program, only in place of arraylist object create object of linkedlist.

Vector

- Introduced in 1.0 version, so Vector is also referred as Legacy class.
- Heterogenous objects are allowed.
- Null objects are allowed.
- Insertion order is preserved.
- Duplicate objects are allowed.
- Datastructure is growable array.

- cursors - ListIterator and Iterator
- Same process to write program, only in place of arraylist object create object of Vector.

** Differences between ArrayList and LinkedList

ArrayList	LinkedList
Default size/capacity of ArrayList is 10	Deafult size of LinkedList is 0
It's Data structure is growable size array ex: ---------- |0|1|2|3|4|5|6|7|8|9| ----------	It's Data structure is doubly LinkedList ex: 
It is good, if our operation is retrieval.	It is good, if our operation is insertion and deletion.
While adding data in middle, ArrayList is slower.	LinkedList is faster, when we add object in middle.
AayList is faster, when we add objects in the end.	LinkedList is slower, when we add objects in end.

** Differences between ArrayList and Vector

ArrayList	Vector
Default capacity of ArrayList is 10	Default capacity of Vector is 10
Inroduced in 1.2 version of Java, so Non Legacy Class.	Introduced in 1.0 version of Java, so Legacy Class.
When it reaches its saturation point size increases by 3/2+1 ex: 10*3 ------ + 1 = 16 2	when it reaches its saturation point size increases by its equal size ex: 10+10=20
Performance is higher in ArrayList(Not Thread Safe)	Performance is poor in Vector(Thread Safe)

instanceof operator :

- Referencevar instanceof classname/interfacename
- We are checking whether the reference var on left is an instance/Object of specified type(class or subclass or interface type) on the right.

```
Animal a1=new Animal();
System.out.println(a1 instanceof Animal);//true
```

- An object of subclass type is also a type of parent class

```
class Animal                    //Ex
{}
class Dog extends Animal
{
  }
class C
{
  public static void main Sring(args []) {
        Dog d1=new Dog();
        System.out.println(d1 instanceof Dog);//true
        System.out.println(d1 instanceof Animal);//true
        System.out.println(d1 instanceof Student);//false
      }
}
```

```
import java.util.ArrayList;      //Ex
public class Eighth {
      public static void main(String[] args) {
            ArrayList<Object> a1=new ArrayList<>();
            a1.add(new Student("John",123));
            a1.add(new Student("Rohan",124));
            a1.add(new Animal("Tiger","Yellow"));
            a1.add(new Animal("Dog","White"));
            System.out.println(a1);
            for(Object obj:a1)
            {
                  if(obj instanceof Student)
                  {
                        Student s1= (Student) obj;
                        System.out.println(s1.sname+" "+s1.sid);
                  }
                  if(obj instanceof Animal)
                  {
                        Animal s2= (Animal) obj;
                        System.out.println(s2.name+" "+s2.color);
```

```
                }
            }
}}
class Student
{
      String sname;int sid;
      public Student(String sname,int sid) {
            this.sname=sname;
            this.sid=sid;
      }
}
class Animal
{
      String name,color;
      public Animal(String name,String color) {
            this.name=name;
            this.color=color;
      }
}
```

Output:

```
[Student@28a418fc, Student@5305068a, Animal@1f32e575, Animal@279f2327]
John 123
Rohan 124
Tiger Yellow
Dog White
```

07-10-2020

2.LinkedHashSet

It is a class which extends HashSet and implements set interface.

Features:

- Introduced in 1.2v
- Heterogenous objects are allowed
- Duplicates are not allowed. In case if we add we won't get compile time error, it will just add once
- only one NULL object is allowed
- Data structure is Linkedlist.
- Insertion order is preserved.
- Set is only uni directional so it supports only Iterator

//Same program only difference is output is as per insertion order//

```
import java.util.*;                    //Ex
public class Seventh
{
      public static void main(String[] args) {
          LinkedHashSet<Object> s1=new LinkedHashSet<>();
                  s1.add("Rohan");
                  s1.add("Riya");
                  s1.add("Pooja");
                  s1.add(445);
                  s1.add('A');
                  System.out.println("Hash set objectss are: "+s1);
                  s1.add("Riya");
                  s1.add(445);
                  System.out.println("After add duplicates: "+s1);
                  s1.add(null);
                  s1.add(null);
                  System.out.println("After adding null objects: "+s1);
                  System.out.println("-------Forward direction-----");
                  Iterator<Object> i1=s1.iterator();
                  while(i1.hasNext())
                  {
                 System.out.println(i1.next());
                  }
       }}
```

Output:

```
Hash set objectss are: [Rohan, Riya, Pooja, 445, A]
```

```
After add duplicates: [Rohan, Riya, Pooja, 445, A]
After adding null objects: [Rohan, Riya, Pooja, 445, A, null]
-------Forward direction-----
Rohan
Riya
Pooja
445
A
null
```

3.Treeset

It is a class which implements set interafce.

Features:

- Introduced in 1.2v
- Heterogenous objects are not allowed if we add we will get class cast exception
- Duplicates are not allowed .In case if we add we won't get compile time error, it will just add once
- No NULL object is allowed
- Data structure is tree.
- Output is in Sorted Order.
- Set is only uni directional so it supports only Iterator

```
import java.util.*;                    //Ex-1
public class Seventh
{
      public static void main(String[] args) {
            TreeSet<Object> s1=new TreeSet<>();
                  s1.add("Rohan");
                  s1.add("Riya");
                  s1.add("Pooja");
                  s1.add(445);
                  s1.add('A');
                  System.out.println("Hash set objectss are: "+s1);
}}
```

Output:

```
Exception in thread "main" java.lang.ClassCastException:
java.base/java.lang.String cannot be cast to java.base/java.lang.Integer
```

```
import java.util.*;                    //Ex-2
public class Treedemo {
      public static void main(String[] args) {
       TreeSet<Object> s1=new TreeSet<>();
        s1.add("Java");
        s1.add("SQL");
        s1.add("APtitude");
        s1.add("J2EE");
        System.out.println("Treeset Objects are: "+s1);
        //s1.add(122);
        //s1.add('A');
        //s1.add(new Student("John",566));
        //System.out.println("Tree set after adding heterogenous obj
:"+s1);
        //s1.add(null);
        //System.out.println("After adding null :"+s1);
        System.out.println( s1.first());
        System.out.println(s1.last());
        System.out.println(s1.pollFirst());
        System.out.println(s1.pollLast());
        System.out.println(s1.contains("APtitude"));
        System.out.println(s1.contains("SQL"));
}}
```

Output:

```
Treeset Objects are: [APtitude, J2EE, Java, SQL]
APtitude
SQL
APtitude
SQL
false
false
```

- In Treeset output will defaultly in sorted order
- It allows only homogenous objects i.e it allows only comparable type of objects, if we add heterogeneous objects it gives ClassCastExc.
- If we add null objects it gives NullPointerException.
- first()-->provides first object
- last()--->provides last object
- pollFirst()-->provides first object and delete it from tree
- pollLast()--->provides last object and delete it from tree

2-Dimensional Arrays

- It is represented as Rows and Columns in matrix format.
  Ex: 10 22
      30 44
- Syntax:
  1. Arraytype arrayname[][]=new arraytype[rowsize][colsize];

  2. Arraytype [][]arrayname=new arraytype[rowsize][colsize];

  3. Arraytype[][] arrayname=new arraytype[rowsize][colsize];

Ex: int a[][]=new int[3][3];

rowsize-3
colsize-3

a[0][0]=10
a[0][1]=20
a[0][2]=30
a[1][0]=40
a[1][1]=50
a[1][2]=60
a[2][0]=70
a[2][1]=80
a[2][2]=90

2. We can directly store the elements as
   int a[][]={{1,2,3},{2,3,4},{4,5,6}}
   1 2 3
   2 3 4
   4 5 6

   int a[]={{1,2},{3,4}}
   1 2
   3 4

   int a[][]={{2,3,4},{1,5,7}}
   2 3 4
   1 5 7

```java
import java.util.Scanner;                    //Ex
public class Array2d {
	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter Row Size :");
		int rowsize=sc.nextInt();
		System.out.println("Enter Column Size :");
		int colsize=sc.nextInt();
		int a[][]=new int[rowsize][colsize];
	//for taking values from user
	System.out.println("Enter the elements :");
	  for(int i=0;i<rowsize;i++) {
		  for(int j=0;j<colsize;j++) {
			a[i][j]=sc.nextInt();
	}}
	//for printing values
	for(int i=0;i<rowsize;i++)  {
		for(int j=0;j<colsize;j++) {
		 System.out.println("Value at a["+i+"]["+j+"]th : "+a[i][j]);
		}}
}}
```

Output:

```
Enter Row Size :
3
Enter Column Size :
3
Enter the elements :
1
2
3
4
5
6
7
8
9
Value at a[0][0]th : 1
Value at a[0][1]th : 2
Value at a[0][2]th : 3
Value at a[1][0]th : 4
Value at a[1][1]th : 5
Value at a[1][2]th : 6
Value at a[2][0]th : 7
Value at a[2][1]th : 8
Value at a[2][2]th : 9
```

08-10-2020

WAP to Add a 3*3 matrix

```
public class Add{                                   //Ex
public static void main(String args[]){
//creating two matrices
int a[][]={{1,3,4},{2,4,3},{3,4,5}};
int b[][]={{1,3,4},{2,4,3},{1,2,4}};

//creating another matrix to store the sum of two matrices
int c[][]=new int[3][3];  //3 rows and 3 columns

//adding and printing addition of 2 matrices
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
c[i][j]=a[i][j]+b[i][j];
System.out.print(c[i][j]+" ");
}
System.out.println();//new line
}
}}
```

WAP to multiply a 3*3 matrix

```
public class MatrixMultiplicationExample{   //Ex
public static void main(String args[]){
//creating two matrices
int a[][]={{1,1,1},{2,2,2},{3,3,3}};
int b[][]={{1,1,1},{2,2,2},{3,3,3}};

//creating another matrix to store the multiplication of two matrices
int c[][]=new int[3][3];  //3 rows and 3 columns

//multiplying and printing multiplication of 2 matrices
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
c[i][j]=0;
for(int k=0;k<3;k++)
{
c[i][j]+=a[i][k]*b[k][j];
}//end of k loop
```

```
System.out.print(c[i][j]+" ");  //printing matrix element
}//end of j loop
System.out.println();//new line
}
}}
```

Passing an Array to a method

```
import java.util.Scanner;                    //Ex
public class Sample3 {
public static void main(String[] args) {
          int size;
        Scanner s=new Scanner(System.in);
        System.out.println("Enter the size of array");
        size=s.nextInt();//5
       int  arr[]=new int[size];
        for(int i=0;i<arr.length;i++)//0-<5
        {
           System.out.print("Enter array element ar a["+i+"]th index  ");
           arr[i]=s.nextInt();//a[0]=10,a[1]=20,......a[4]=888
        }
        passarray(arr);//calling passarray() which is taking args as
                              array objeect
     }
     public static void passarray(int a[])
     {
          for(int i=0;i<a.length;i++)
          {
               System.out.println(a[i]);
          }
     }}
```

Passing an Array as a method

```
-----------------------------
public class Demo1 {                                    //Ex
      public static void main(String[] args) {
            String s[]= {"Amar","Rohan","John"};
            int a[]= {12,33,44};
           details(s,a);//call to details() and pass args as String array
                              and integer array
      }
      public static void details(String names[],int age[])
      {
            for(int i=0;i<names.length;i++)
            {
                  System.out.println("NAme of person is : "+names[i]+" And
                        Age is :"+age[i]);
}}
```

Output:

```
NAme of person is : Amar And Age is :12
NAme of person is : Rohan And Age is :33
NAme of person is : John And Age is :44
```

Map Interface

- Map is an interface which stores the objects as a key value pair.
- Each key,value pair is called as entry.
- So map is also referred as collection of entry objects
- Whenever we want to represents objects as key value pair we go for maps.

- Map interface is implemented by 3 different classes
  1. HASHMAP
  2. LINKEDHASHMAP
  3. TREEMAP

Useful methods are :

- put(k,V): Used to add objects as key value pair into Map
- keyset(): Returns all key's available in Map
- values(): Returns all values available in Map.
- containsKey(key) and containsValue(value): checks key and value is present or not respectively.

HashMap

- HashMap is a class which implements Map interface.

Features

- introduced in 1.2v
- heterogenous data allowed
- Data structure is hashtable
- duplicate keys are not allowed but values can be duplicate
- random order based on hashcode
- only one null key is allowed and multiple null values are allowed.

```
import java.util.*;                    //Ex
public class One {
      public static void main(String[] args) {
            HashMap<Integer,String> m1=new HashMap<>();
            m1.put(101,"Samsung");
            m1.put(102,"Realme");
            m1.put(103,"Oppo");
            m1.put(104,"Vivo");
            System.out.println("Map Objects are : "+m1);

            m1.put(103, "Redmi");
            System.out.println("Duplicate key : "+m1);
            m1.put(105, "Samsung");
            System.out.println("Duplicate Value : "+m1);
            System.out.println("All keys : "+m1.keySet());
            System.out.println("All Values : "+m1.values());
            System.out.println("Particular Value : "+m1.get(102));
            System.out.println("Check key : "+m1.containsKey(1001));
         System.out.println("Check Value :
"+m1.containsValue("Iphone"));
}}
```

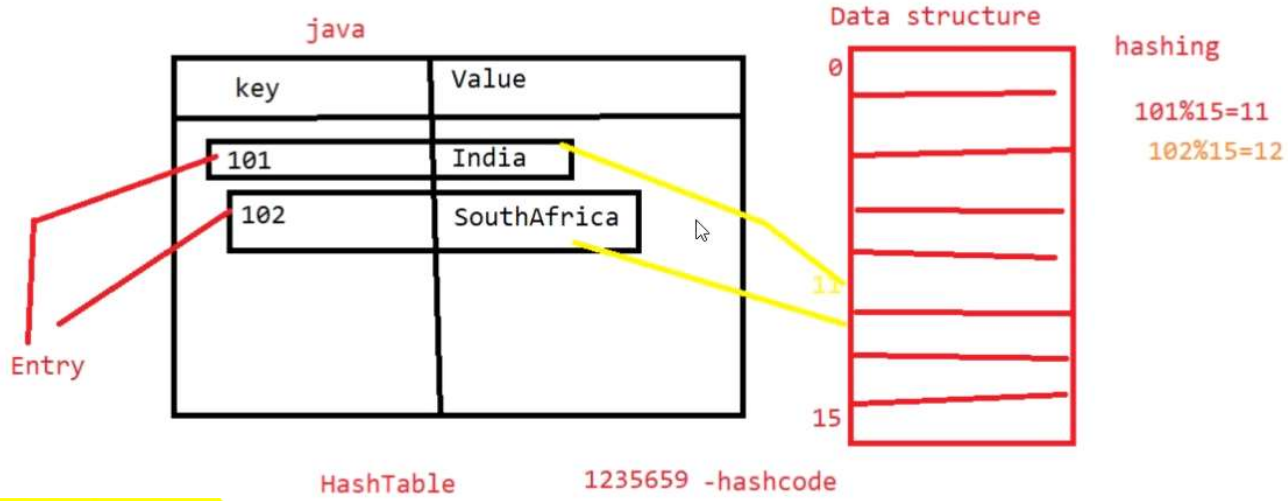
Output:

```
Map Objects are : {101=Samsung, 102=Realme, 103=Oppo, 104=Vivo}
Duplicate key : {101=Samsung, 102=Realme, 103=Redmi, 104=Vivo}
Duplicate Value : {101=Samsung, 102=Realme, 103=Redmi, 104=Vivo,
105=Samsung}
All keys : [101, 102, 103, 104, 105]
All Values : [Samsung, Realme, Redmi, Vivo, Samsung]
Particular Value : Realme
Check key : false
Check Value : false
```

09-10-2020

HashTable

- It is a datastructure(like an Array) which stores the objects as key value pair by means of Hashing.
- Default size of hashtable is 11
- once hashtable reaches the 3/4th of default size it size will be doubled i.e 22.



LinkedHashMap

- LinkedHashMap is a class which extends HashMap and implements Map interface.

Features

- introduced in 1.4v
- heterogenous data allowed
- Data structure is hashtable
- duplicate keys are not allowed but values can be duplicate,if we add duplicate key it
- replaces with original one.
- As per Insertion order
- only one null key is allowed and multiple null values are allowed.

//Only difference is output is as per insertion order//

```
import java.util.*;                    //Ex
public class Linkmapdemo {
     public static void main(String[] args) {
```

```
        LinkedHashMap<Integer,String> m1=new LinkedHashMap<>();
         m1.put(388,"Rahul");
         m1.put(1,"Riya");
         m1.put(103,"Pooja");
        // m1.put("334",445);
         System.out.println(m1);
         System.out.println(m1.keySet());
         System.out.println(m1.values());
         System.out.println( m1.get(101));
         System.out.println(m1.get(334));
         System.out.println(m1.containsKey(1002));
         System.out.println(m1.containsValue("Java"));
         m1.put(101,"Sanju");
         m1.put(103,"Pooja1.0");
         System.out.println("After adding duplicate keys"+m1);
         m1.put(null,"Pooja2.0");
         System.out.println(m1);
         m1.put(null,"Pooja3.0");
         System.out.println(m1);
}}
```

Output:

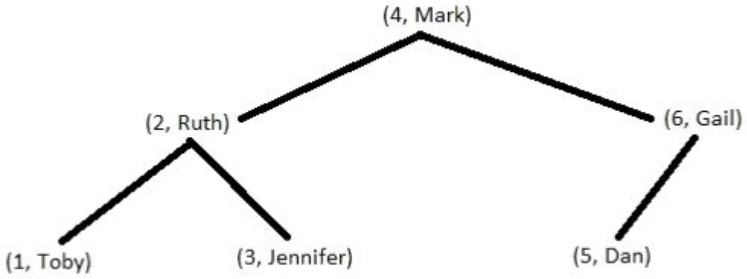
```
{388=Rahul, 1=Riya, 103=Pooja}
[388, 1, 103]
[Rahul, Riya, Pooja]
null
null
false
false
After adding duplicate keys{388=Rahul, 1=Riya, 103=Pooja1.0, 101=Sanju}
{388=Rahul, 1=Riya, 103=Pooja1.0, 101=Sanju, null=Pooja2.0}
{388=Rahul, 1=Riya, 103=Pooja1.0, 101=Sanju, null=Pooja3.0}
```

TreeMap

- TreeMap is a class which implements Map interface.

Features

- introduced in 1.2v
- only homogenous data allowed ,if we add heterogeneous objects will get class cast exception
- Data structure is Tree
- duplicate keys are not allowed but values can be duplicate
- sorting order.
- Null keys are not allowed.




```
import java.util.*;                          //Ex
public class Treemapdemo {
      public static void main(String[] args) {
         TreeMap<Integer,String> m1=new TreeMap<>();
        m1.put(101,"Rahul");
        m1.put(102,"Riya");
        m1.put(103,"Pooja");
        System.out.println(m1);
        System.out.println(m1.keySet());
        System.out.println(m1.values());
        System.out.println( m1.get(101));
        System.out.println(m1.get(334));
        System.out.println(m1.containsKey(1002));
        System.out.println(m1.containsValue("Java"));
        m1.put(101,"Sanju");
        m1.put(103,"Pooja1.0");
        System.out.println("After adding duplicate keys"+m1);
        m1.put(null,"Pooja2.0");
        System.out.println(m1);
        m1.put(null,"Pooja3.0");
        System.out.println(m1);
}}
```

Output:

```
{101=Rahul, 102=Riya, 103=Pooja}
[101, 102, 103]
[Rahul, Riya, Pooja]
Rahul
null
false
false
After adding duplicate keys{101=Sanju, 102=Riya, 103=Pooja1.0}
Exception in thread "main" java.lang.NullPointerException
      at java.base/java.util.TreeMap.put(TreeMap.java:561)
                  at Treemapdemo.main(Treemapdemo.java:18)
```

12-10-2020

STRING PROGRAMMING

- A group of characters represents a string.

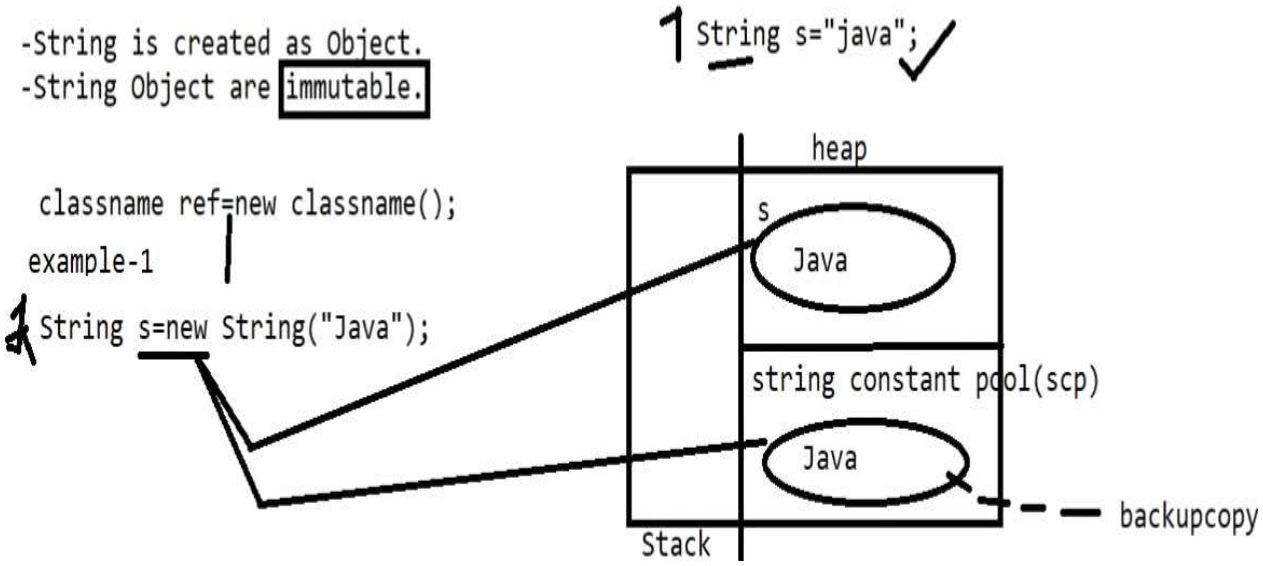
String

- String is predefine class which is present in java.lang package
- In java string is an -
    1. object
    2. datatype
    3. class
    4. group of characters

objects :

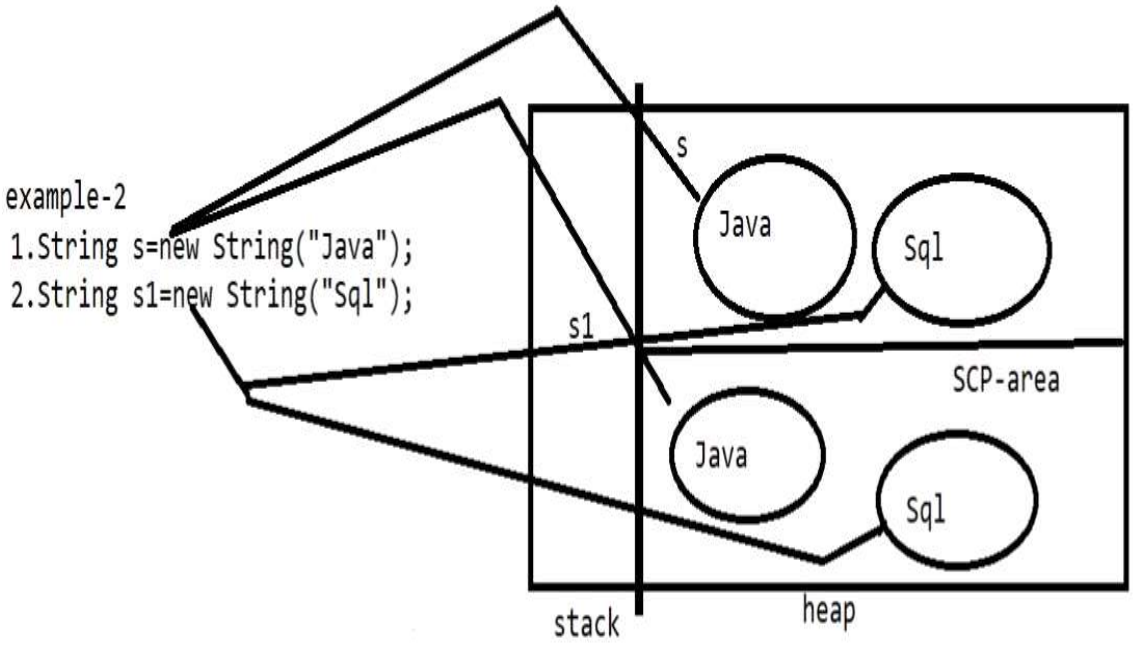
String objects can be created in two ways-

1. using new keyword - two objects will get created
    1. one is under heap area - reference variable assigned to it.
    2. another is under string constant pool(SCP) - it is small part of heap area no reference assigned to it, it is only for backup.



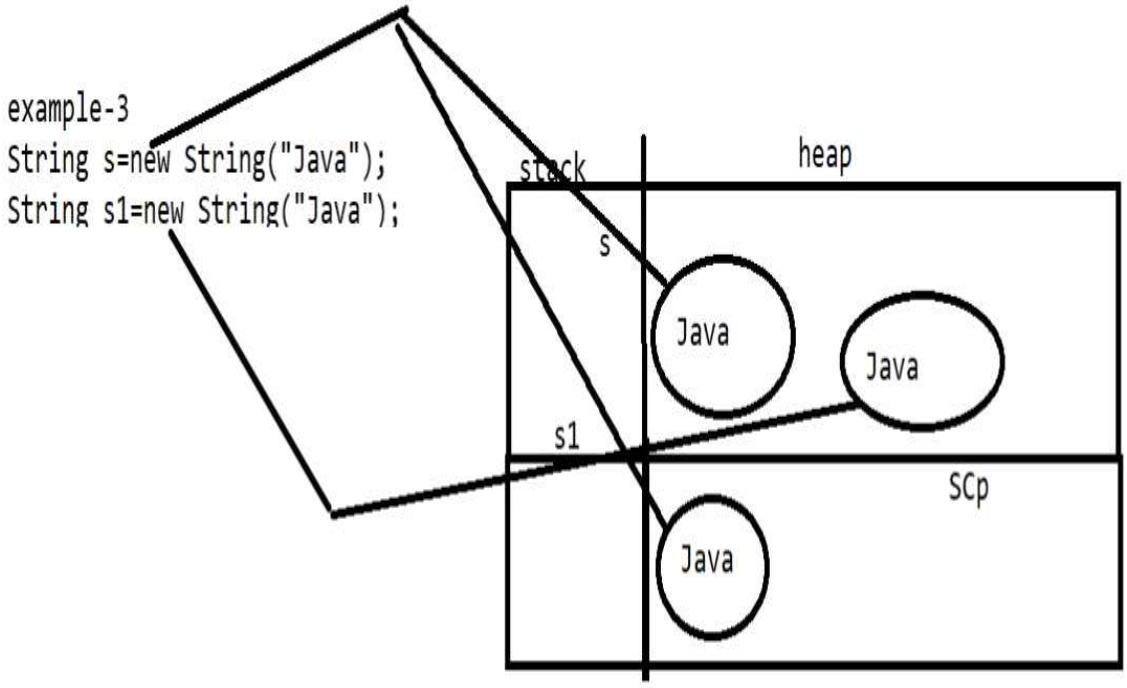
example-2

```
1.String s=new String("Java");
2.String s1=new String("Sql");
```



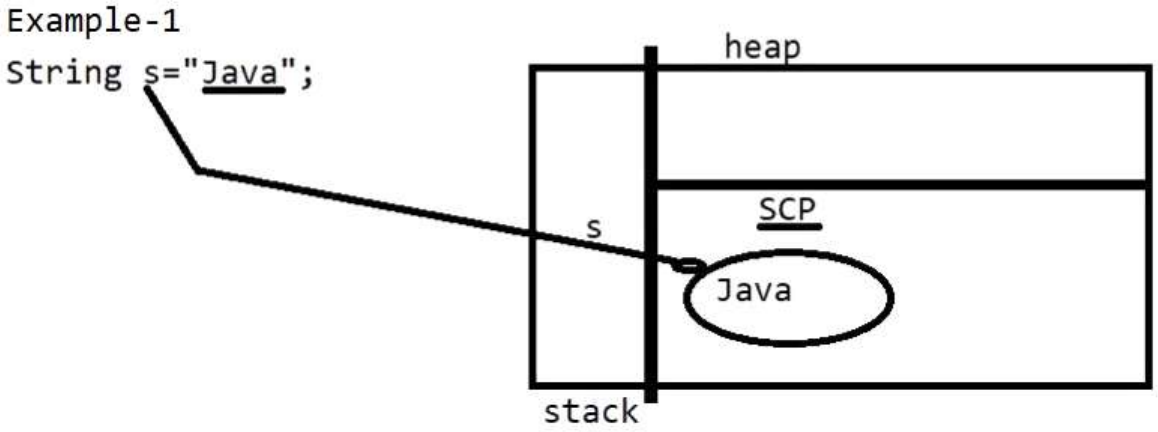
example-3

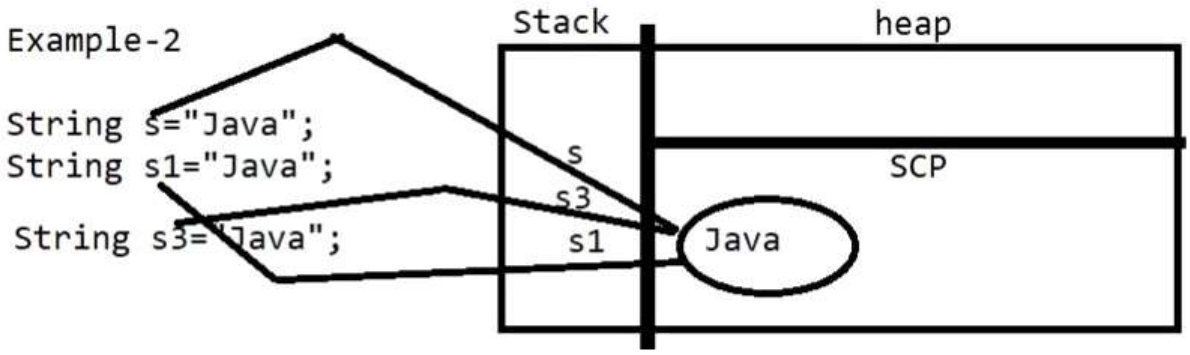
```
String s=new String("Java");
String s1=new String("Java");
```



2. using literal

- first JVM will go to SCP and check if there is any object present with string data, if it is there it will assign reference to it, if it is not there it creates new object and assign reference.



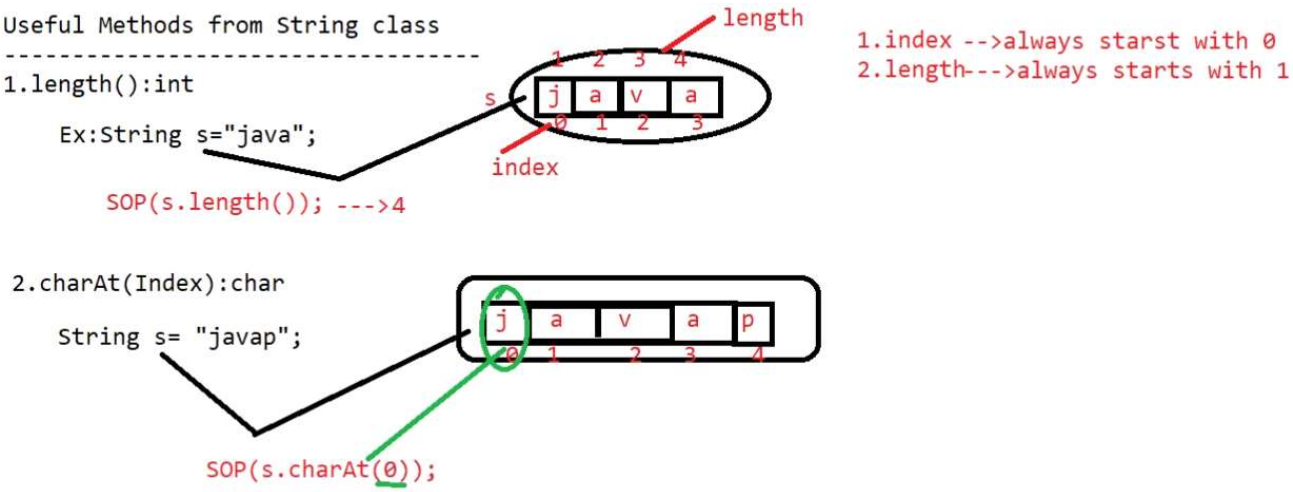


IMMUTABILITY

From above examples, we understand that for one string object there can be multiple references assigned to it, if we change data of one string object it will effect to multiple references thats why java says that string objects are immutable i.e once we created we cannot modify it.

13-10-2020
---

methods of String class



1.length():int

- it provides length of String
- length will always calculated from 1

```
public class FirstP {
	public static void main(String[] args) {
		String s="I am a java developer";
		System.out.println(s.length());
}}
```

Output: 21

2.charAt(int Index):char

- it provides character at given index
- index will always start from 0

```
public class FirstP {                                   //Ex-1
	public static void main(String[] args) {
		String s="javap";
		System.out.println(s.charAt(3));
}}
```

Output: a

```
public class FirstP {                                   //Ex-2
	public static void main(String[] args) {
		String s="javap";
```

```
            System.out.println(s.charAt(5));
}}
```
Output:
```
Exception in thread "main" java.lang.StringIndexOutOfBoundsException:
String index out of range: 5
      at java.base/java.lang.StringLatin1.charAt(StringLatin1.java:48)
      at java.base/java.lang.String.charAt(String.java:711)
      at FirstP.main(FirstP.java:5)
```

Note:
if we provide invalid indes as arguments for charAt() we will get StringIndexOutOfBoundException

1. WAP to check count of e/E character present in a String s="javaEEdeve"

```
public class FirstP {
      public static void main(String[] args) {
            String s="javaEEdeve";
            int count=0;
            for(int i=0;i<s.length();i++)
            {
                  if(s.charAt(i)=='e'||s.charAt(i)=='E')
                  {
                        count++;
                  }
            }
            System.out.println("Count of E/e is : "+count);
}}
```
Output: Count of E/e is : 4

2. WAP to find smaller case vowels fom string s="javadev"
   a.print vowels b.count vowels

```
public class FirstP {
      public static void main(String[] args) {
            String s="javadev";
            int count=0;
            for(int i=0;i<s.length();i++)
            {

      if(s.charAt(i)=='a'||s.charAt(i)=='e'||s.charAt(i)=='i'||
            s.charAt(i)=='o'||s.charAt(i)=='u')
                  {
                        System.out.println(s.charAt(i));
                        count++;
                  }
            }
```

```
      System.out.println("Count of vowels in smaller case is : "+count);
}}
```
Output:
```
a
a
e
Count of vowels in smaller case is : 3
```

3. WAP to provide reverse of a string, actual string is "javadev"and reverse string is "vedavaj"

```
public class Reverse {
      public static void main(String[] args) {
            String actual="javadev";
            String rev="";
            for(int i=actual.length()-1;i>=0;i--)
            {
                  rev=rev+actual.charAt(i);
            }
            System.out.println("reverse of a string is : "+rev);
}}
```
Output:
```
reverse of a string is : vedavaj
```

3.object1.equals(object2):boolean

- it checks whether two strings are same or not, i.e it compars two strings based on string data
- ex-1:
```
String s1="java";
String s2="java";
System.out.println(s1.equals(s2);//true
```
- ex-2:
```
String s1="java";
String s2="javadev";
System.out.println(s1.equals(s2);//false
```
- ex-1:
```
String s1="java";
String s2="JAva";
System.out.println(s1.equals(s2);//false
```
because equals method consider case sensitivity

4.object1.equalsIgnoreCase(object2):boolean

- compares two strings based on string data without cosidering case sensitivity

4. WAP to check whether the given string is pallindrome or not.
a."madam" b."Mom"

```
public class Reverse {
	public static void main(String[] args) {
		String actual="madam";
		String rev="";
		for(int i=actual.length()-1;i>=0;i--)
		{
			rev=rev+actual.charAt(i);
		}
		System.out.println("Actual String : "+actual);
		System.out.println("Reverse String : "+rev);
		if(actual.equals(rev))
		{
			System.out.println("Given String is pallindrome");
		}
}}
```

Output:

```
Actual String : madam
Reverse String : madam
Given String is pallindrome
```

```
public class Reverse {
	public static void main(String[] args) {
		String actual="Mom";
		String rev="";
		for(int i=actual.length()-1;i>=0;i--)
		{
			rev=rev+actual.charAt(i);
		}
		System.out.println("Actual String : "+actual);
		System.out.println("Reverse String : "+rev);
		if(actual.equalsIgnoreCase(rev))
		{
			System.out.println("Given String is pallindrome");
		}
		else
		{
			System.out.println("Given String is not pallindrome");
		}
}}
```

Output:

```
Actual String : Mom
Reverse String : moM
Given String is pallindrome
```

14-10-2020

5.trim():String

- it removes white spaces from starting and ending of string
- trim() will not remove spaces from middle
- Ex: s=" java dev";
  s.trim(); --->"java dev"

5. WAP to count no of words present in string

```
public class CountWords {
	public static void main(String[] args) {
		String s="  I am a java    developer    ";
		System.out.println("Before trimming:"+s);
		String s1=s.trim();//helps to remove spaces from start and end
				//of sentence
		System.out.println("After trimming:"+s1);
		int count=1;
		for(int i=0;i<s1.length();i++) {
			if(s1.charAt(i)==' ' && s1.charAt(i+1)!=' ') {
				count=count+1;
		}}
		System.out.println("No of words are : "+count);
}}
```

Output:

```
Before trimming:  I am a java    developer
After trimming:I am a java    developer
No of words are : 5
```

6.(a) subString(int arg):String

- it provides subpart of a string

(b) subString(int fromindex,int toindex):String

- subString(int fromindex(including),int toindex(excluding)):String

```
public class Substring {
	public static void main(String[] args) {
		String s="java development";
		String s1=s.substring(3);
		System.out.println(s1);
		String s2=s.substring(2,9);
		System.out.println(s2);
}}
```

Output:

```
a development
va deve
```

7.indexOf():int

indexOf(char):int
indexOf(String):int
indexOf(char,int fromindex):int
indexOf(String,int fromindex):int

- it provides index value for given string or character
- ex: String s="java"
  int i=s.index('v');//2
  int j=s.index("va");//2
  int k=s.index("a",2);//3 ---> provide index of a but start searching from 2nd index
  int l=s.index('z');//-1
- indexOf() returns -1, if character or string is not present

```
public class IndexOff {
	public static void main(String[] args) {
		String s="java development";
		int i=s.indexOf('a');//--->1
		System.out.println(i);
		int j=s.indexOf("dev");//--->5
		System.out.println(j);
		int k=s.indexOf('a',2);//--->3
		System.out.println(k);
		int l=s.indexOf("eve",4);//--->6
		System.out.println(l);
		int m=s.indexOf('Z');
		System.out.println(m);//--->-1
}}
```

* WAP to print all characters only once from string
String s="javajavajavadevdevdev" **Output:-javde**

```
public class Uniq {
	public static void main(String[] args) {
		String s="javajavajavadevdev";
		String un="";
		for(int i=0;i<s.length();i++) {
			char ch=s.charAt(i);
			if(un.indexOf(ch)==-1) {
				un=un+ch;
		}}
		System.out.println("Unique string is :"+un);
}}
```

Output:

```
Unique string is :javde
```

15-10-2020

8.toUpperCase():String
- Converts string into uppercase

9.toLowerCase():String
- Converts string into lowercase

10.startsWith(string):boolean
- check whether string is starting with given string or not

11.endsWith(string):boolean
- check whether string is ending with given string or not

12.contains(String):boolean
- checks whether string is present in given string or not

13.isEmpty():boolean
- checks whether string is empty or not

14.object.concat(String)
- concatinates string in end of called string

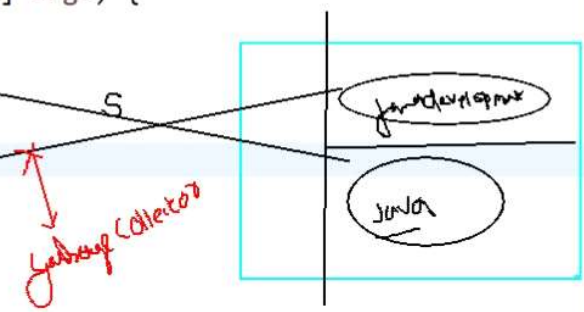
```
public class SMethods {
	public static void main(String[] args) {
		String s="jaVA DEvelopment";
		String up=s.toUpperCase();
		String lo=s.toLowerCase();
		System.out.println(up);
		System.out.println(lo);
		System.out.println(s.startsWith("sql"));
		System.out.println(s.endsWith("ent"));
		System.out.println(s.contains("vel"));
		String s1="";
		System.out.println(s1.isEmpty());
	}}
```

Output:

```
JAVA DEVELOPMENT
java development
false
true
true
true
```

```
public class Concatinations {
public static void main(String[] args) {
    String s="java";

    s.concat("development");
    System.out.println(s);
```

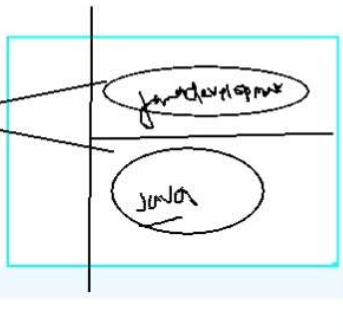



```
<terminated> Concatinations [
java
```

```
public class Concatinations {
public static void main(String[] args) {
    String s="java";

    s=s.concat("development");

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



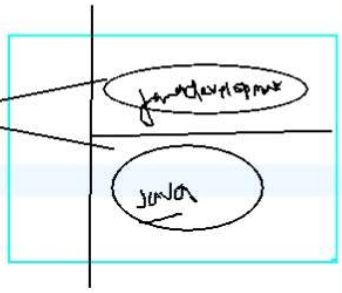

```
<terminated> Concatinations [Java Ap
javadevelopment
```

```
public class Concatinations {
public static void main(String[] args) {
    String s="java";

    String s1=s.concat("development");

    System.out.println(s);

    System.out.println(s1);
```




```
<terminated> Concatinations [Java A
java
javadevelopment
```

```
public class Concatinations {
public static void main(String[] args) {
      String s="java";
      s.concat("development");
      System.out.println(s);

      s=s.concat("development");
      System.out.println(s);

      String s1="java";
```

```
        String s2=s1.concat("development");
        System.out.println(s2);
}}
```

Output:

```
java
javadevelopment
javadevelopment
```

replace():

- Replaces all occurences given character/sequence of character/string with replacement character/string respectively

1. replace(givenchar,desirechar):String
    - used to replace characters
2. replaceAll(givenString,desireString):String
    - used to replace String
3. replaceAll(String regrex,String replacement)
    - regrex - regular expression, ex:[0-9], [a-z], [A-Z]

1. WAP to replace e with a in given string "java development"
2. WAp to replace "java" with "core java" in given "java development"
3. WAP to remove spaces from given string "java development"
4. WAP to remove all capital letters from string "jAvA DeVeloPer"
5. WAP to remove all small letters from same string
6. WAP to remove digits from string "ja123vaDEveloper"
7. WAP to remove vowels from string "ja123vaDEveloper"

```
public class ReplaceM {
        public static void main(String[] args) {
                String s="java development";
                String r1=s.replace('e','a');
                System.out.println(r1);
                String r2=s.replaceAll("java","core java");
                System.out.println(r2);
                String r3=s.replaceAll(" ","");
                System.out.println(r3);
                String s1="jAvA DeVeloPer";
                String r4=s1.replaceAll("[A-Z]","");
                System.out.println(r4);
                String r5=s1.replaceAll("[a-z]","");
                System.out.println(r5);
                String s2="ja123vaDEveloper";
```

```
            String r6=s2.replaceAll("[0-9]","");
            System.out.println(r6);
            String r7=s2.replaceAll("[aeiouAEIOU]","");
            System.out.println(r7);
}}
```

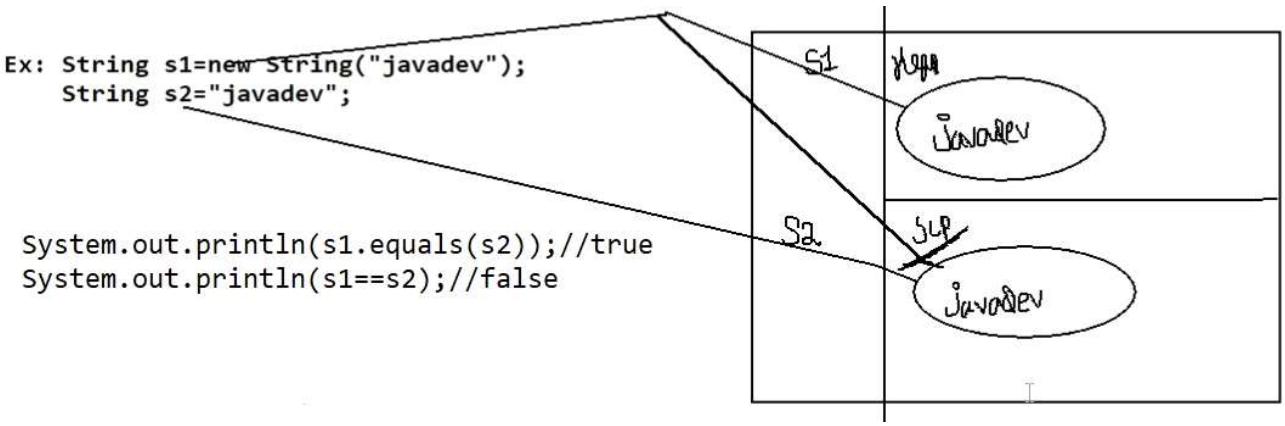
Output:

```
java davalopmant
core java development
javadevelopment
jv eeloer
AA DVP
javaDEveloper
j123vDvlpr
```

***differences between equals() and == with respect to Strings

---

- equals() ---> compares two strings based on string data
- == ---> compares two strings based on reference




```
public class equ {
      public static void main(String[] args) {
            String s1=new String("javadev");
            String s2="javadev";
            System.out.println(s1.equals(s2));
            System.out.println(s1==s2);
            String s3="javadev";
            System.out.println(s2==s3);
}}
```

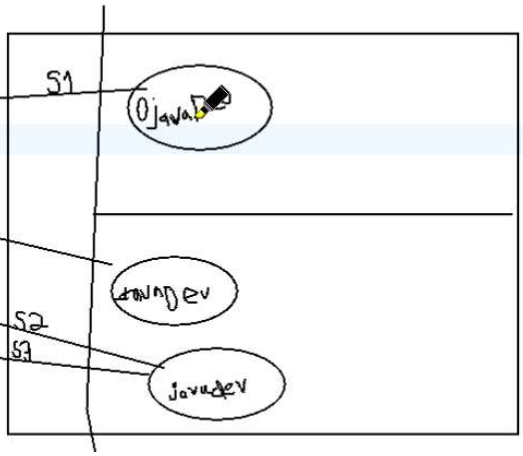
Output:

```
true
false
true
```

```
public class equ {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String s1=new String("javaDev");
        String s2="javadev";
        System.out.println(s1.equalsIgnoreCase(s2));
        System.out.println(s1==s2);
        String s3="javadev";
        System.out.println(s1.equalsIgnoreCase(s3));
    }

}
```




```
public class equ {
      public static void main(String[] args) {
            String s1=new String("javaDev");
            String s2="javadev";
            System.out.println(s1.equalsIgnoreCase(s2));
            System.out.println(s1==s2);
            String s3="javadev";
            System.out.println(s1.equalsIgnoreCase(s3));
}}
```

Output:

```
true
false
true
```

`split(args):String[]`

- It converts or breaks the string into an array depending on argument
- ex-1:
```
String s="i am a java developer";
String s1[]=s.split(" ");
Output:
      s1[0]=i
      s1[0]=am
      s1[0]=a
      s1[0]=java
      s1[0]=developer
```
- ex-2:
```
String s="java";
String ch[]=s.split("");//for each character of string break it
                        //and stored in array
Output:
      ch[0]="j";
      ch[1]="a";
      ch[2]="v";
```

```
ch[3]="a";
```

- WAP to find frequency of substring from given string

```
public class Frequency {
 public static void main(String[] args) {
      String str1="we work to live and live to be happy live";
      String word1="live";
      check(str1,word1);
 }
 public static void check(String str, String word) {
      String s[]=str.split(" ");
      int count=0;
      for(int i=0;i<s.length;i++)
      {
            if(word.equals(s[i]))
            {
                  count++;
      }}
      System.out.println(count);
}}
```

Output:- 3

- WAP to count longest word from a string

String s="I am a java developer"

```
public class LongestWord {
public static void main(String[] args) {
      String s="I am a java developer";
      String s1[]=s.split(" ");
      System.out.println("length of array : "+s1.length);
      for(int i=0;i<s1.length;i++)
      {
                  System.out.print(s1[i]+"-");
                  System.out.println(s1[i].length());
      }
      int max=0;
      for(int i=0;i<s1.length;i++)
      {
            if(s1[i].length()>max)
            {
                  max=s1[i].length();
            }
      }
      System.out.print("The longest word from the string : "+max);
}}
```

Output:

```
length of array : 5
I-1
am-2
a-1
java-4
developer-9
The longest word from the string : 9
```

toCharArray():char[]

---

it converts string into character array

```
ex:   String s-"javadev";
      char ch[]=s.toCharArray();
      Output:    ch[0]='j'
                 ch[1]='a'
                 ch[2]='v'
                 ch[3]='a'........
```

- WAP to calculate frequency of characters present in a string "javadev"

```
public class freq {
public static void main(String[] args) {
      String str="javadev";
      String s=str.toUpperCase();//s=JAVADEV
      char[] s1=s.toCharArray();//{'J','A','V','A','D','E','V'}
      for(char ch='A';ch<='Z';ch++)
      {
            int count=0;
            for(int i=0;i<s1.length;i++)
            {
                  if(ch==s1[i])
                  {
                        count++;
            }}
            if(count>0)
                  System.out.println(ch+"-"+count);
      }
}}
```

Output:

```
A-2
D-1
E-1
J-1
V-2
```

19-10-2020

String Buffer and String Builder

- These are classes which are present in java.lang package
- String buffer and String builder objects can be created only by using new keyword

  Ex: StringBuffer b1=new StringBuffer("java");
  StringBuilder b2=new StringBuilder("java");
  StringBuffer b3="java";//invalid
  StringBuilder b4="java";//invalid
- StringBuffer & StringBuilder objects are mutable i.e once we created we can modify it.
- StringBuffer & StringBuilder objects will get created in heap area only.

equals()

- In String class equal() compares based on string data.
- In StringBuffer and StringBuilder class equals() compares based on reference.
- In String classs equals() is overridden from object class to compare two string objects based on content.
- In StringBuffer and StringBuilder class equals() is not overridden so it compares based on reference (same like objectclass).
- == operator can't be applied for string builder and string buffer objects.

```
public class Str1 {
	public static void main(String[] args) {
		StringBuffer b1=new StringBuffer("java");
		StringBuilder b2=new StringBuilder("java");
		System.out.println(b1.equals(b2));
		System.out.println(b1);
		System.out.println(b2);
		b1.append("Development");
		b2.append("Full Stack");
		System.out.println(b1);
		System.out.println(b2);
}}
```

Output:

```
false
java
java
```

javaDevelopment
javaFull Stack

***Differences between STRING, STRINGBUFFER & STRINGBUILDER

S. No	STRING	SRINGBUFFER	STRINGBUILDER
1.	String objects are immutable	Stringbuffer objects are mutable	Stringbuilder objects are mutable
2.	Objects can be created in two ways 1.new 2.literal	Objects can be created only using new keyword	Objects can be created only using new keyword
3.	Objects will created in SCP or heap	Objects will ceated in heap	Objects will created in heap
4.	Thread safe i.e at a time only one thread is allowed to operate on string object	Thread safe i.e at a tie only one thread is allowed to operate on string buffer object	Not a Thread safe
5.	If context is fixed we will go for string	If context is varying we will go for string buffer	If context is varying we will go for string builder
6.	equals()-compares two String by seeing content	equals()-compares two String by seeing reference	equals()-compares two String by seeing reference
7.	performance is faster	performance is moderate	performance is faster
8.	For concatination we have concat()	For concatination we have append()	For concatination we have append()