JAVA

Programming Language: - The Language which is used to write some instructions to get some work done by the System is Known as Programming Language.

Generally, there are two types of Language
1: High Level Language
2: Low Level Language

1: High Level Language: A Language which is easily readable, Understandable and intractable by the Programmer is Known as High Level Language.
E g:-C,C++,Java Ruby, Python etc.

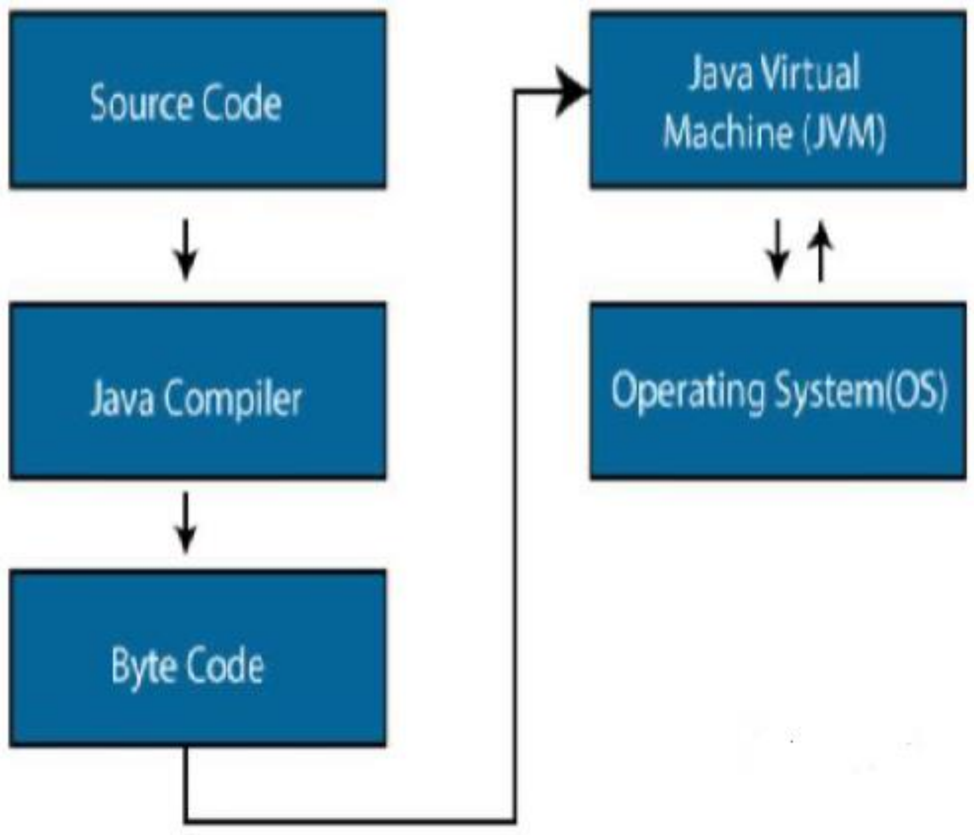
2: Low Level Language: A Language which is easily Understandable and Executed by the machine is Known as Low Level Language.

Java:- Java is a High Level Programming Language which is Developed by James Gosling and his team under Sun Microsystems. The first version of Java Was Developed on Jan 23,1996.

Features of Java
Simple: Java has made life easier by removing all the complexities such as pointers, operator overloading as you see in C++ or any other programming language.
Portable: Java is platform independent which means that any application written on one platform can be easily ported to another platform.
Object-oriented: Everything is considered to be an “object” which possess some state, behaviour and all the operations are performed using these objects.
Dynamic: It has the ability to adapt to an evolving environment which supports dynamic memory allocation due to which memory wastage is reduced and performance of the application is increased.
Robust: Java has a strong memory management system. It helps in eliminating error as it checks the code during compile and runtime.
Java
is a Platform independent i.e., The Application or a Program developed using Java
can work on different Platform.

STEPS TO DESIGN AN APPLICATION OF JAVA

STEP-1: Select n Editor (Notepad, Notepad++,Edit plus 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

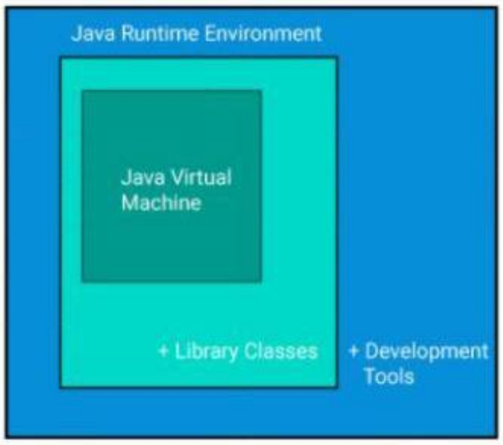
-The process of converting our program into system understandable form (byte code) is purpose of compiling a program.
-to compile a program go to cmd prompt and enter command as
-javac programname.java
-Ex: javac Hello.java
-Compilation is done at once.
-During compilation, compiler will check syntax 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

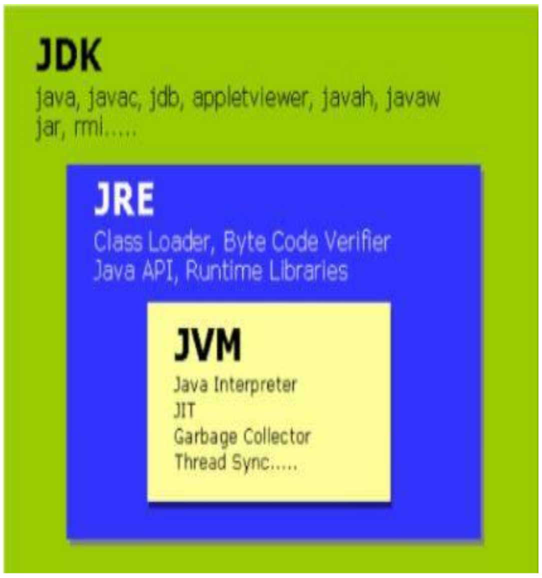
-JVM -java virtual machine is responsible for execution of every java program
-JVM's can identify by ------> Public static void main (String [ ] args)
-it is like one software or one program.

-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 program name
-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

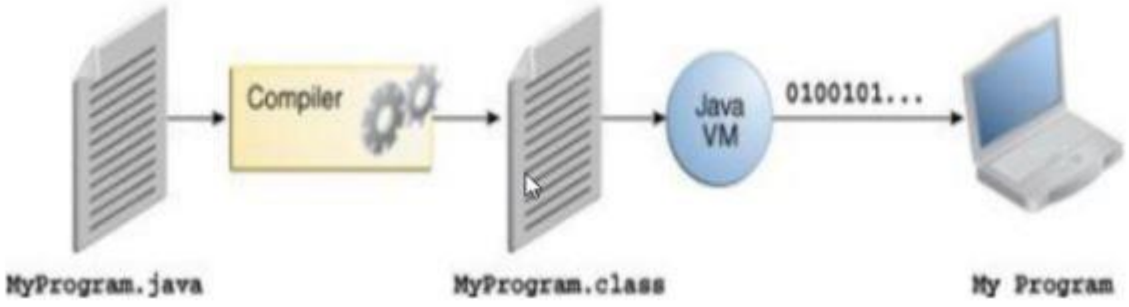
- IF WE WANT TO DEVELOP AND EXECUTE JAVA PROGRAM IN OUR SYSYTEM WE HAVE TO INSATLL JDK(JRE,JVM,SUPPORTING TOOLS AND SUPPORTING CLASSES).



JDK = JRE + Development Tool
JRE = JVM + Library Classes


JDK
java, javac, jdb, appletviewer, javah, javaw jar, rmi.....
JRE
Class Loader, Byte Code Verifier
Java API, Runtime Libraries
JVM
Java Interpreter
JIT
Garbage Collector
Thread Sync.....

OVERVIEW OF APP ENVIRONMENT



TOKENS OF JAVA

1: Keywords
2: Identifiers
3: Literals
4: Separators

1.KEYWORDS: These are the reserve words or predefine words or Compiler Aware Words which have some reserve meaning.

Eg: class, interface , abstract , import , package , static , public , private , protected , default , super, this , extends , implements , try , catch , finally , break , continue , int , double , float , if else ,switch , etc….

2.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

3)Literals:-
The types of value that can be used in a java Program are known as Literals.

There are 4 types of Literals
1)Number: -
a)Integer:- 10,20,1,100
b)Floating:- 10.4,45.2,71.21

2)Characters
a)Characters must be enclosed with ' '→ 'a','e','F',,'d'
Note:- Length of character must be 1

String:
String must be enclosed with " "---→"hello","hi","20","hye"
Note:-String length can be anything

Boolean:
There are two Boolean value
1)True
2)False

4.Separators
Separators are the symbols that are used to separate a group of statements from one another
Eg:-
{ }
()
;
,
[ ]

TYPES OF VARIABLES

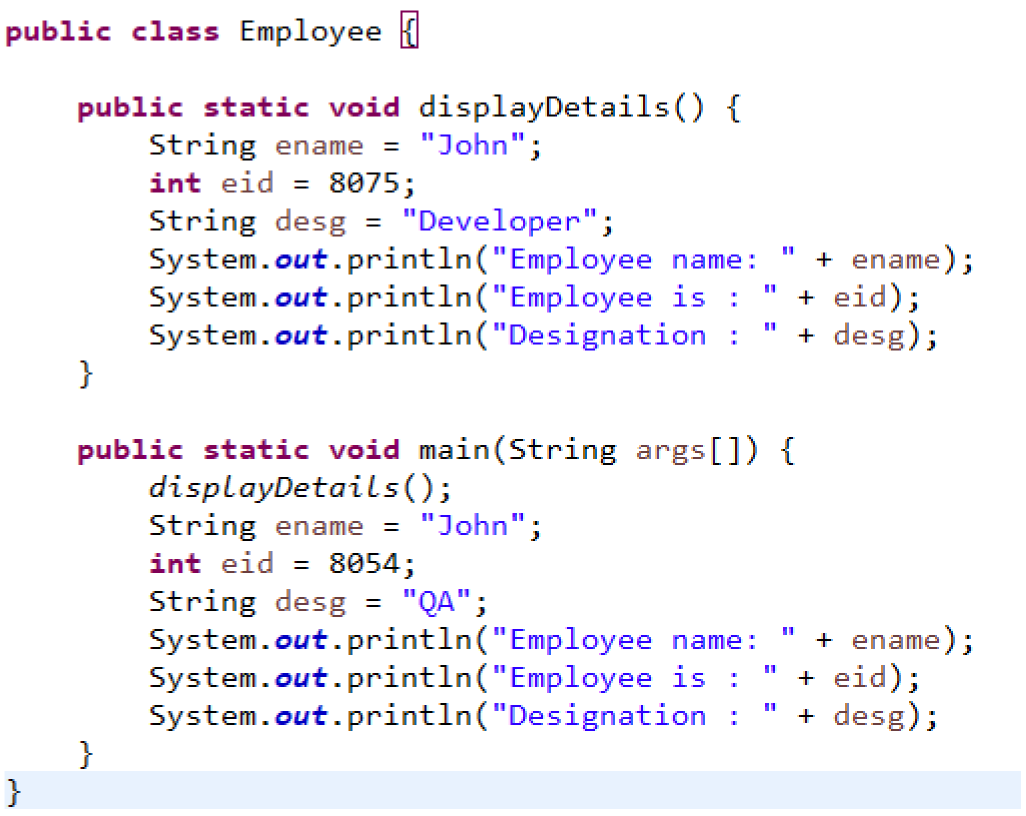
Depending on declaration variables are divided into two types.
1.Local variable
2.Global variable(Data member)

Local variables

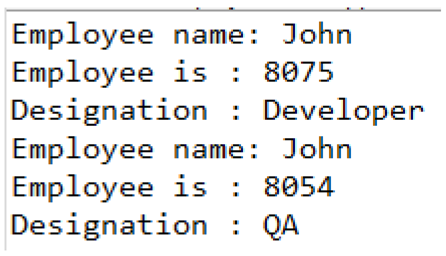
Variables which are declared inside the method(main() or user define) and within the scope of class({}). such variables
are called as Local variables.

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 saying 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);
    }
}
```



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

Is Initialisation of local variable is mandatory??

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

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 = 9587459618l;// 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;//local variable declaration is Invalid
final int i=100;// local variable declaration is valid
private int i=100;// local variable declaration is Invalid
protected int i=100;// local variable declaration is Invalid
}
```

Global variable

A variable declared directly inside class is known as global variable, Global variables can be used anywhere inside the class.

There are two types of global variable,
1)static global variable
2)non static global variable

static members

----

static is a keyword in java.
The members of java such as variable, block, methods prefixed with static keyword are known as static members.

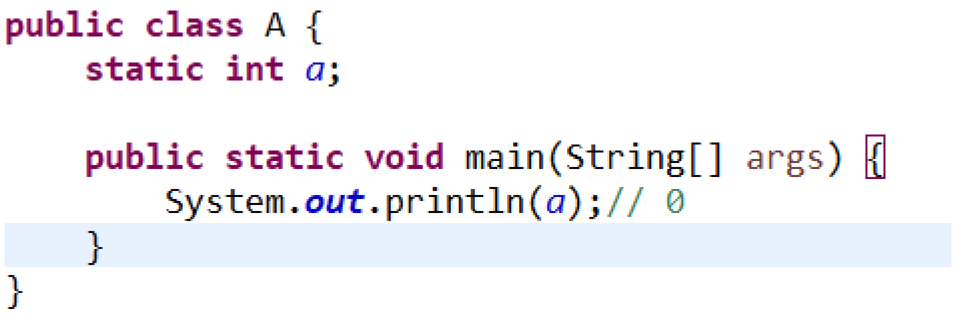
static variable

A variable declared directly inside the class with static keyword is known as static variable. Static variables are global variables.

syntax--> static datatype variable;

Note: -
----
1)The static variables gets their memory inside class area or class static area.
2)static variables will be assigned(initialized) with default values.
3)Since static variables will be initialized with default values, it can be used without initialization.

Eg:-



```java
public class A {
    static int a;

    public static void main(String[] args) {
        System.out.println(a);// 0
    }
}
```

Note:- Whenever the name of static global variable and local variable are same, the highest priority is given to local variable, In this case if we want to use global variable we should make use of Classname as reference.

Syntax--> Classname. VariableName

Eg: -

```java
public class A {
    static int a = 20;

    public static void main(String[] args) {
        int a = 10;
        System.out.println(a);
        System.out.println(A.a);
    }
}
```

The static variable can be used in two ways,
1)directly
2)using Classname
3)Through Object Reference

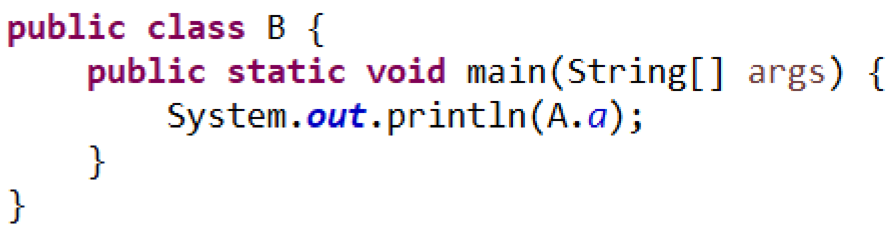
Eg: -

```java
public class A {
    static int a = 20;

    public static void main(String[] args) {
        A a1 = new A();
        System.out.println(a);// 20
        System.out.println(A.a);// 20
        System.out.println(a1.a);// 20
    }
}
```

Note: - We can use static variable of one class in another class with the help of classname.

Eg: -

```
public class A {
        static int a=20;

}
```



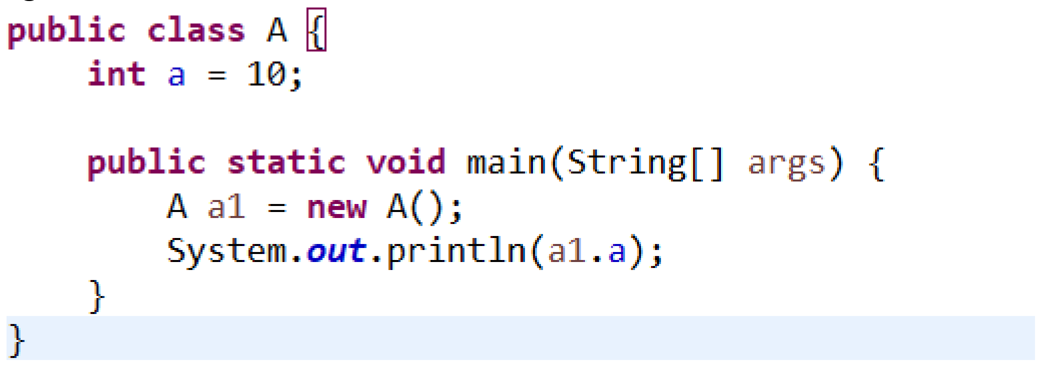
```
public class B {
    public static void main(String[] args) {
        System.out.println(A.a);
    }
}
```

Non static variable

----

1)Non static variables are global variables.
2)Global variables declared without static keyword are known as non static variables.
3)Non static variables gets memory inside the object of the class.
4)Therefore to use non static variables creating object for the class is mandatory.
5)Since they are global variables, they will have default values. Therefore initializing is not mandatory before using.
6)They can be used throughout the class.

Eg:-



```
public class A {
    int a = 10;

    public static void main(String[] args) {
        A a1 = new A();
        System.out.println(a1.a);
    }
}
```

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

Eg: -

```
class STDApp {
//static global variables//
    static String name = "Qspiders";
    static long contact = 9573267325l;
    static String email = "Qspiders@gmal.com";

    public static void enquiry() {
//local variable
        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

```
QspidersKPHB
9573267325
Qspiderskphb@gmail.com
```

Q. Is initialisation of global variable is mandatory?
A. They are not mandatory to initialise, if we did not initialise, 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 initialization//
    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

```
0000
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.

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

Its 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 their memory size
-These are 8 in numbers
1.byte -1 byte 10,2,5(127 is max -128 is minimum) 0
2.boolean- 1 byte or no size true or 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 Numerical 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 represent with 'l'

Non primitive data type:

-These are not fixed in their memory 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 initialise 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;//default it will be considered as double to indicate 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 decimal 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

Operators

Arithmetic Operators:

+ ====> Addition

- ====> Subtraction

\ ====> Division (output: Coefficient)

* ====> Multiplication

% ====> Mode (output: Remainder)

=>Create an Application which calculates

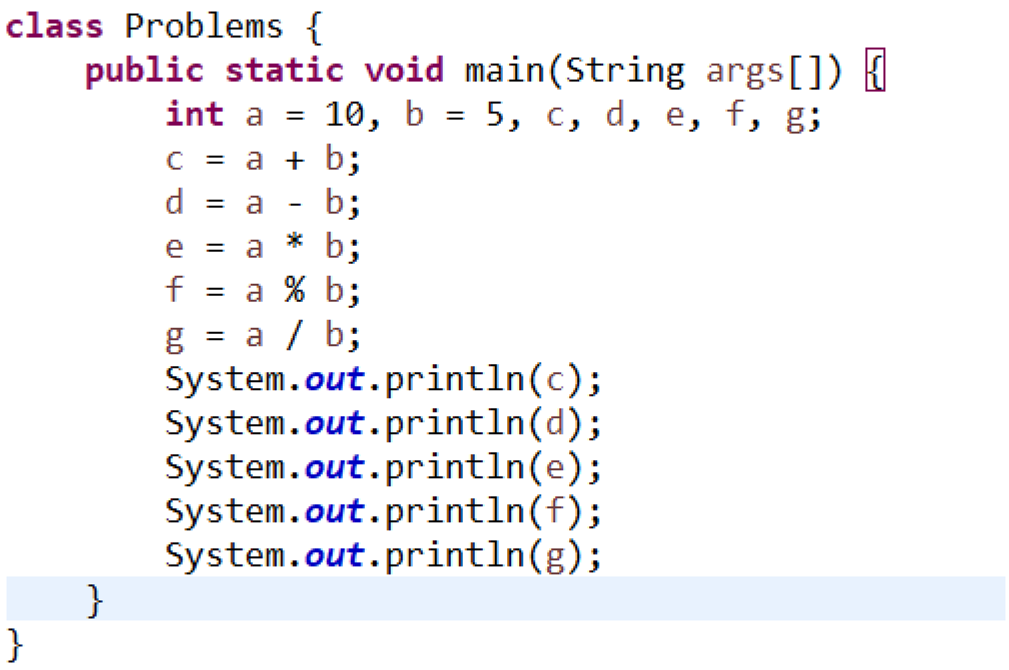
1.Addition of two numbers

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);
    }
}
```

Operators are the symbols which are used to perform some operation.
The value on which the operation is performed is known as operand.

Eg: - 1+2
    +-->operator
    1 and 2-->operands

Based on the number of operands used operators are classified into,

1)Unary operators

2)Binary operators

3)Ternary operators

1)Unary operator

An operator which accepts single operand is known as unary operator.

Eg:-Increment operator, decrement operator, cast operator.

2)Binary operator

An operator which accepts two operands is known as binary operator.

Eg: -Arithmetic operator, Relational operator, Assignment operator, Logical operator.

3)Ternary operator

An operator which accepts three operands is known as ternary operator.

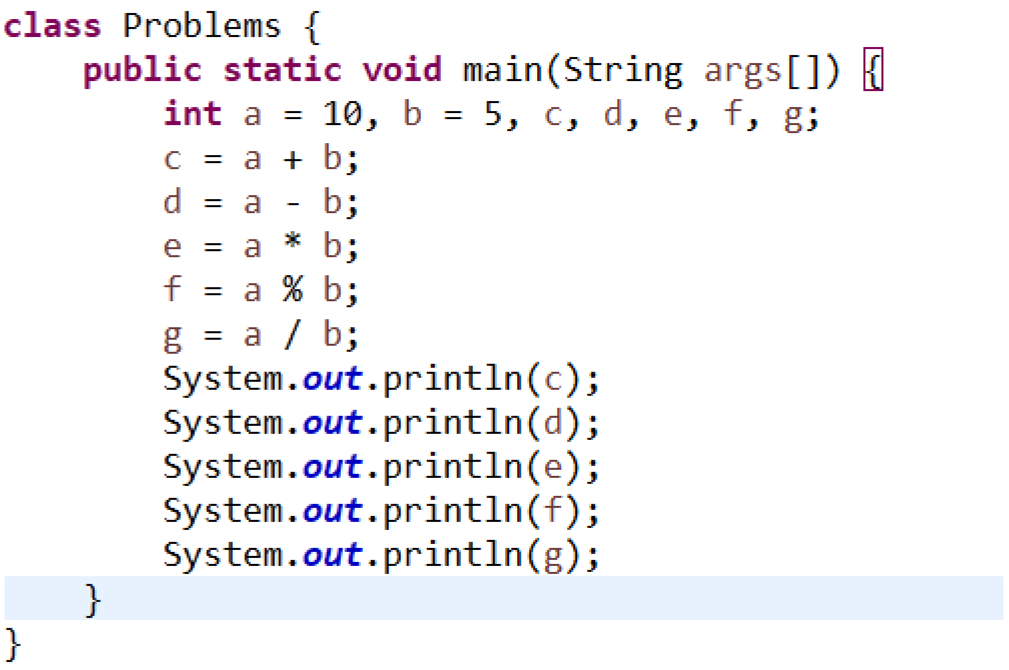
Eg:- Conditional operator.

Arithmetic operator

Arithmetic operator is a binary operator.

+
-
*
/
%

Eg: -



```
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);
    }
}
```

Increment operator (++)

Increment operator is an unary operator, which is used to increment the value of variable by 1.

It is denoted by ++

There are two types of increment operators

1)Post Increment operator

2)Pre Increment operator

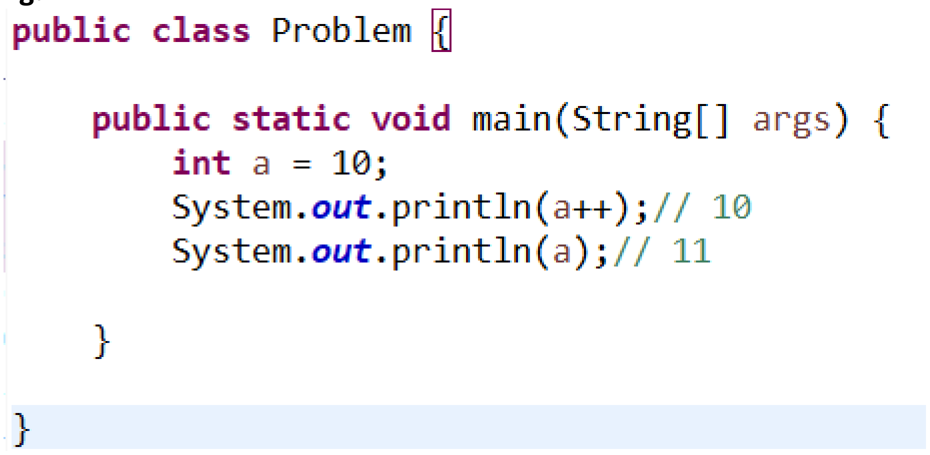
Post increment operator

If the increment operator is suffixed to a variable, it is known as post increment operator.

Eg: - a++ (a-->variable)

Note: - Post Increment operator first uses the value of the variable and then increments the value by 1 and updates the variable.

Eg: -



```
public class Problem {

    public static void main(String[] args) {
        int a = 10;
        System.out.println(a++);// 10
        System.out.println(a);// 11

    }

}
```

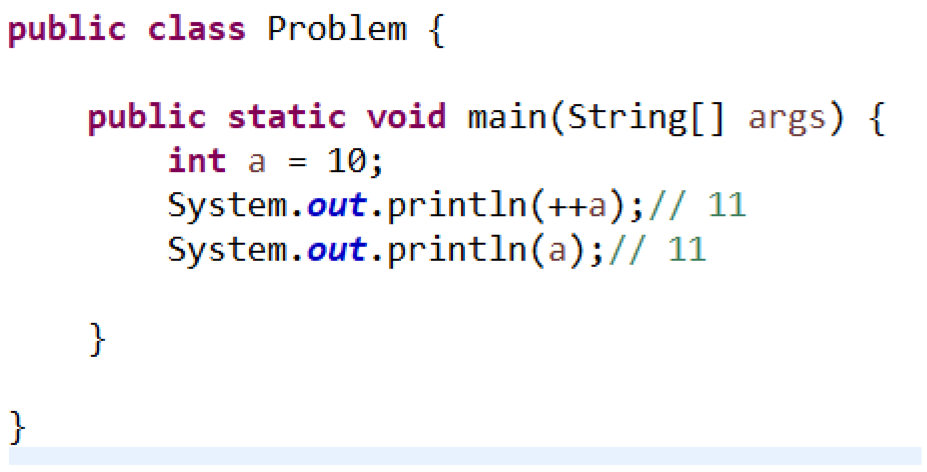
Pre increment operator

If the increment operator is prefixed to a variable, it is known as pre increment operator.

Eg: - ++a (a-->variable)

Note: - Pre Increment operator first increments the value of variable by 1 and updates the variable and then uses the value of the variable.

Eg:-



```
public class Problem {

    public static void main(String[] args) {
        int a = 10;
        System.out.println(++a);// 11
        System.out.println(a);// 11

    }

}
```

Eg: - 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 increased by 1 and then 251 is stored in b
Ex3: int a=50://51
++a: //50+1
SOP(a);//51

<u>**Post Increment Operator: -**</u> If the increment operator is suffixed to a variable, it is known as pre increment operator.

Eg:- a++ (a-->variable)

- 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

Pre-Decrement:

- 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
```

Decrement operator(--)

Decrement operator is an unary operator, which is used to decrement the value of variable by 1.
It is denoted by --

There are two types of decrement operators
1)Post decrement operator
2)Pre decrement operator

Post decrement operator

If the decrement operator is suffixed to a variable, it is known as post decrement operator.

Eg: - a-- (a-->variable)

Note: -Post decrement operator first uses the value of the variable and then decrements the value by 1 and updates the variable.
- 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
```

Pre decrement operator

If the decrement operator is prefixed to a variable, it is known as pre decrement operator.

Eg: - --a (a-->variable)

Note: -Pre decrement operator first decrements the value of variable by 1 and updates the variable and then uses the value of the variable.

Eg: - 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

Relational operator

Relational operator is a binary operator.

Note: -Relational operator returns a boolean value(either true or false).

```
==   -->2==2-->true
!=   -->2!=2-->false
>    -->10>20-->false
<    -->10<9--> false
>=   -->10>=10-->true
<=   -->9<=10 -->true
```

Assignment operator

Assignment operator is a binary operator.

Assignment operator is used to assign value present on RHS to variable present on LHS.

Eg: - int a=10;

Note: -Assignment operator has right to left associativity.

```
Eg: -       int a;
            int b;
            a=b=10;
            System.out.println(a); //10
            System.out.println(b); //10
```

```
            +=
            -=
            *=
            /=
            %=
Eg: -   int a=10;
        a+=2; ------>a=a+2; -->a=10+2--->a=12

        int b=2;
        b*=b+2;   --->b*=2+2---->b*=4--->b=b*4--->b=2*4--->b=8
```

Logical AND operator (&&)

AND operator is a binary operator.

Syntax--> Expression1 && Expression2

Note :- In the above syntax both expression1 and expression2 must return boolean value(i.e true or false).

Expression1	Expression2	Result
F	F	F
F	T	F
T	F	F
T	T	T

Logical OR operator (||)

OR operator is a binary operator.

Syntax--> Expression1 || Expression2

Note:- In the above syntax both expression1 and expression2 must return boolean value(i.e true or false).

Expression1	Expression2	Result
T	T	T
T	F	T
F	T	T
F	F	F

Note: - If expression1 returns true, expression2 will not be checked.

Conditional operator

Conditional operator is a ternary operator.

Syntax---> Expression? Statement1: Statement2 ;

Note: - 1)In the above syntax the expression must return a boolean value.(i.e true or false).
2)If expression returns true, Statement1 gets executed.
3)If expression returns false, Statement2 gets executed.

Decision making statements

Decision making statements are the statements which are used to check a condition before executing an instruction or set of instruction.

following are the decision making statements in java,

1)if
2)if-else
3)if-else ladder
4)nested decision making statements
5)switch

1)if

```
syntax--> if(Condition)    --->single statement
              statement;

            or

          if(condition)    --->multiple statements
         {
           statements;
         }
```

2)if-else

```
syntax--> if(condition)   --->single statement
              statement;
              else
              statement;

            or

          if(condition)  --->multiple statement
         {
            statements;
           }
         else
        {
          statements;
        }
```

3)if-else ladder

```
syntax--> if(condition1)
        {
         statements;
        }
        else if(condition2)
        {
         statements;
        }
        else if(condition3)
        {
         statements;
        }
            -
            -
            -
           [ else
             {
              Statements;
             }
                      ]
```

Note: -1) In if-else ladder if one condition is satisfied, remaining conditions will not be checked.
2)In if-else ladder else block is not mandatory.

1)WAP to find smallest of three numbers using if else ladder.

2)WAP to print
hi-->if the number is even and divisible by 5
hello-->if the number is just even.
bye-->if the number is odd and divisibe by 3 as well as 5
good bye-->if none of the above conditions are satisfied.

3)WAP to print
A-->if the character is uppercase alphabet
B-->if the character is lowercase alphabet
C-->if the character is a digit
D-->if the character is a space
Z-->for all remaining special characters.

----

take a character and WAP

```
1)Is it alphabet?
        a)Is it lowercase?
                a)Is it vowel?-->print "It is a lowercase vowel"
                b)else print "It is a lowercase consonant".
        a)Is it uppercase?
                a)Is it vowel?-->print "It is a uppercase vowel"
                b)else print "It is a uppercase consonant".
2)Is it digit?-->print "It is a digit".
3)Is it space?-->print "It is a space".
4)print "It is a special character" if none of the above condition is satisfied.
```

switch

----

switch is a special decision making statement, It checks for the equality.

Note: -1)For switch we can pass a value/variable/Expression.
2)switch accepts any type of value other than float, double and boolean.

```
syntax--> switch(value)
          {
                case value1: Statements;
                             break;

                case value2: Statements;
                             break;

                case value3: Statements;
                             break;
                -
                -
                -
            [   default: Statements; ]
      }
```

Note: -1) The value passed to switch will be compared with case values(checking for equality), if there is any matching case value found then the statements for that case gets executed.

2)if there is no matching case value, then default gets executed. And using default in switch is not mandatory
3)break is a control transfer statement used to transfer the control out of switch block.

Take a number and WAP to print following messages

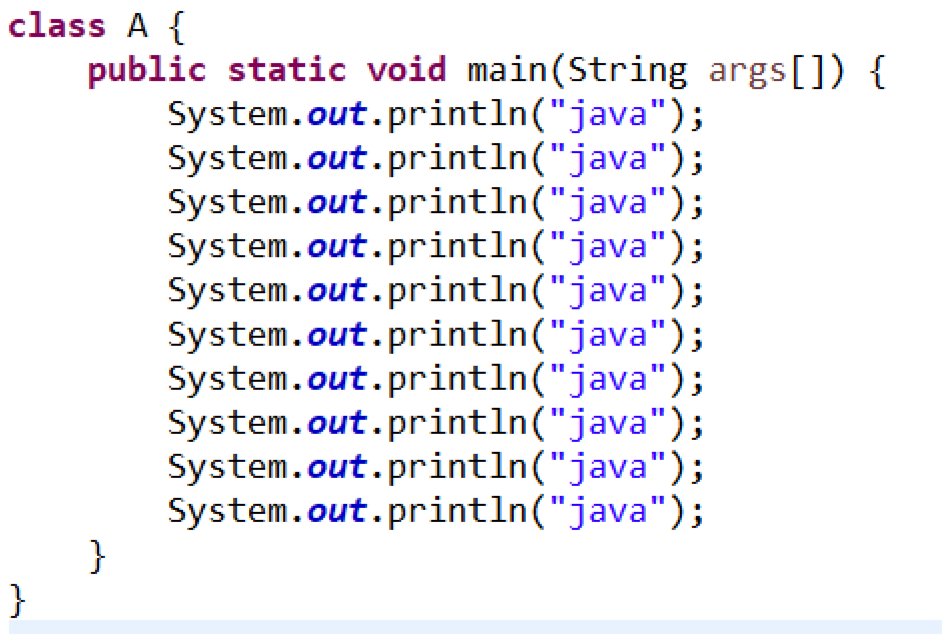
Sunday -->if number taken is 1
Monday -->if number taken is 2
Tuesday -->if number taken is 3
Wednesday -->if number taken is 4
Thursday -->if number taken is 5
Friday -->if number taken is 6
Saturday -->if number taken is 7
invalid-->if it is any other number.

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)

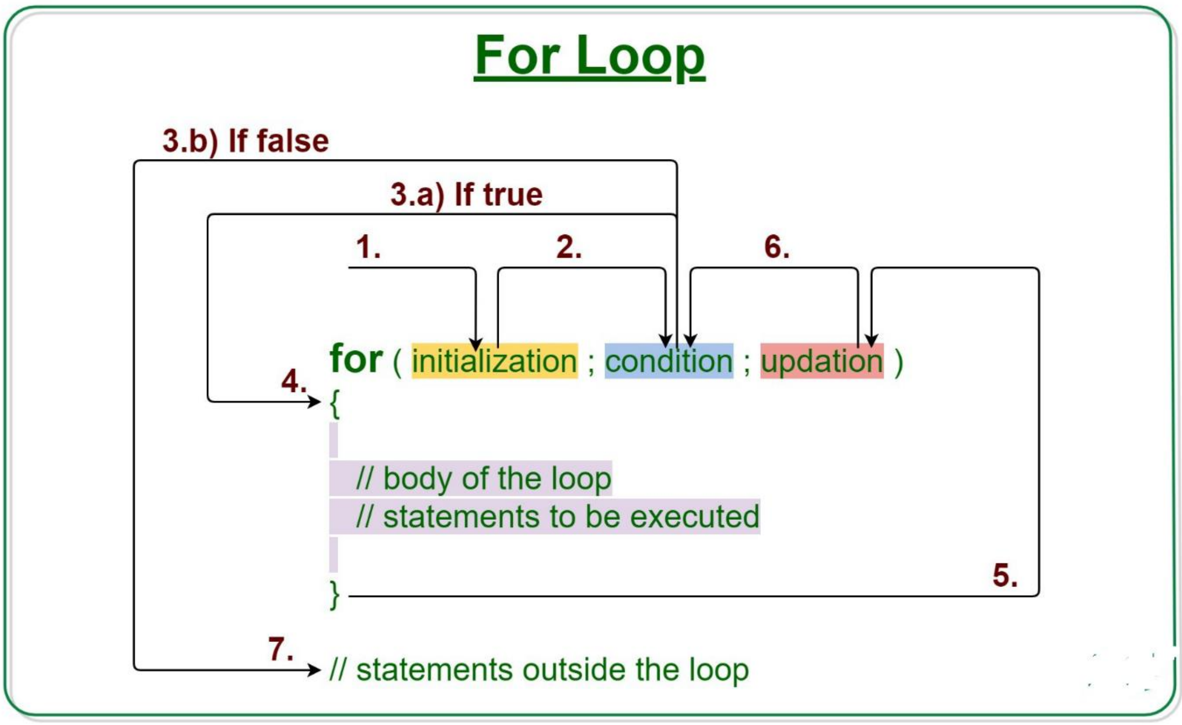
Without looping statements

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//


For Loop
3.b) If false
3.a) If true
1.
2.
6.
for (initialization ; condition ; updation)
4.
{
// body of the loop
// statements to be executed
}
5.
7.
// statements outside the loop

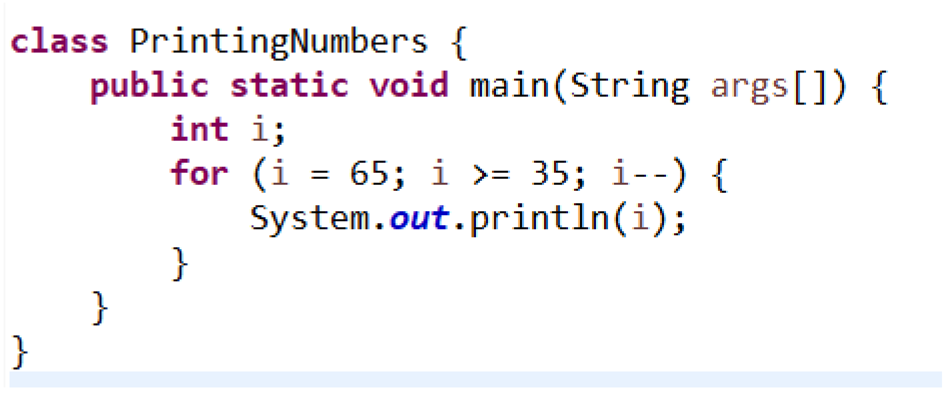
Q. Wap to Print Java 10 times

```
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

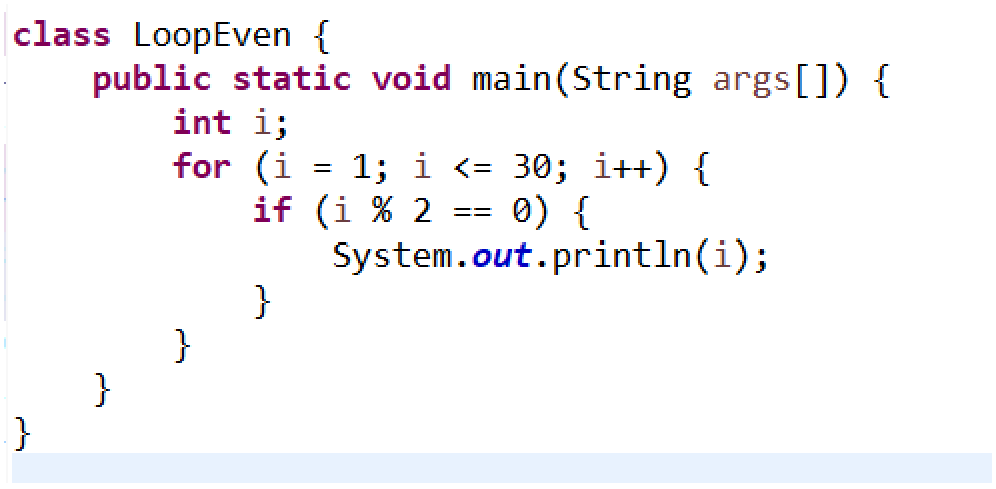
```
class Loop {

    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



```java
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



```java
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);
    }
}
```

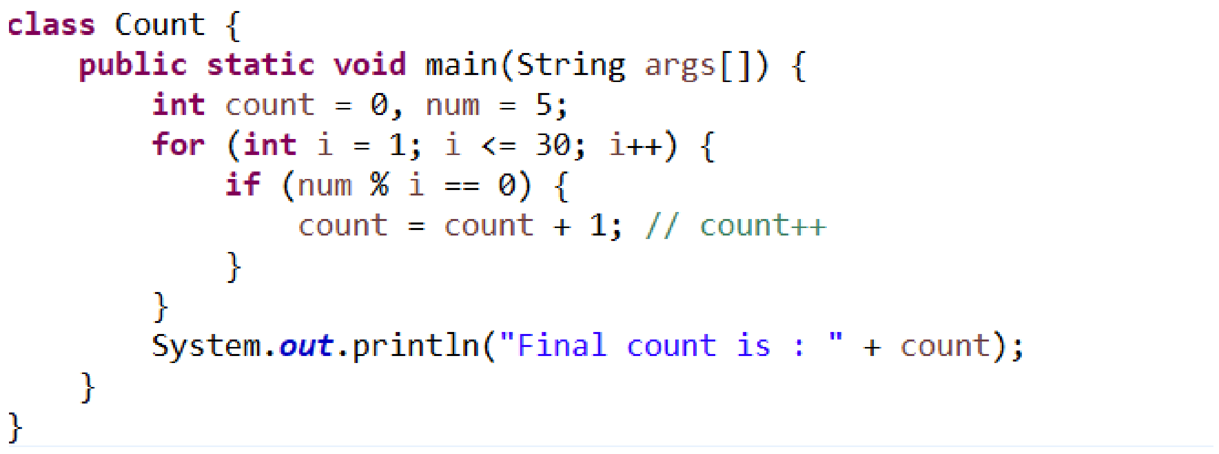
7.WAP to print value of 5!

```
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

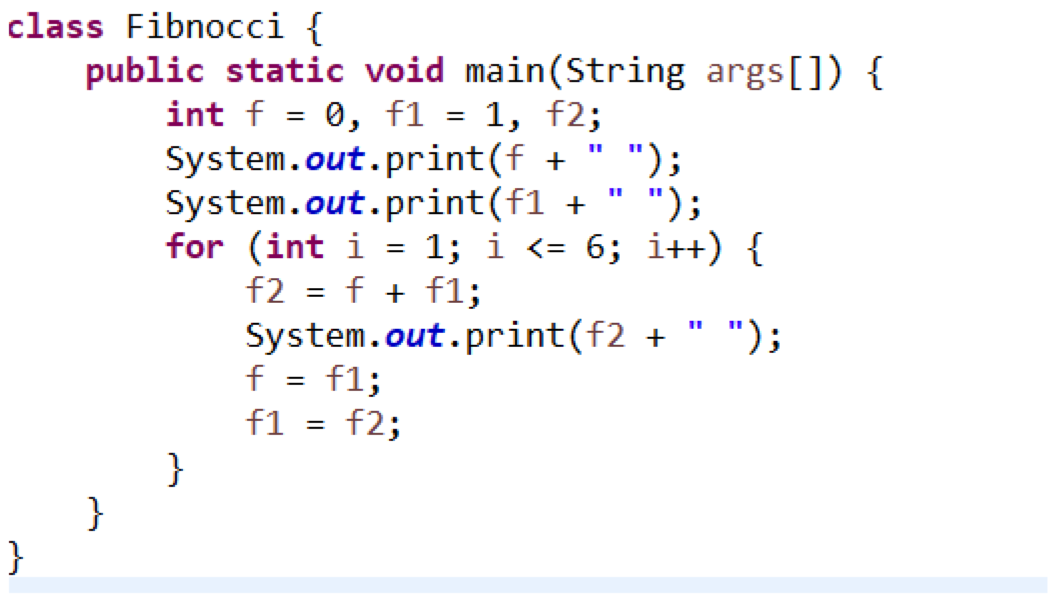
```
class MultiplicationTable {
    public static void main(String args[]) {
        int num = 3;
        for (int i = 1; i <= 10; i++) {
            System.out.println(num + "*" + i + "=" + (num * i));
        }
    }
}
```

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



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

10.WAP to print Fibonacci series
0 1 1 2 3 5 8 13



```java
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;
        }
    }
}
```

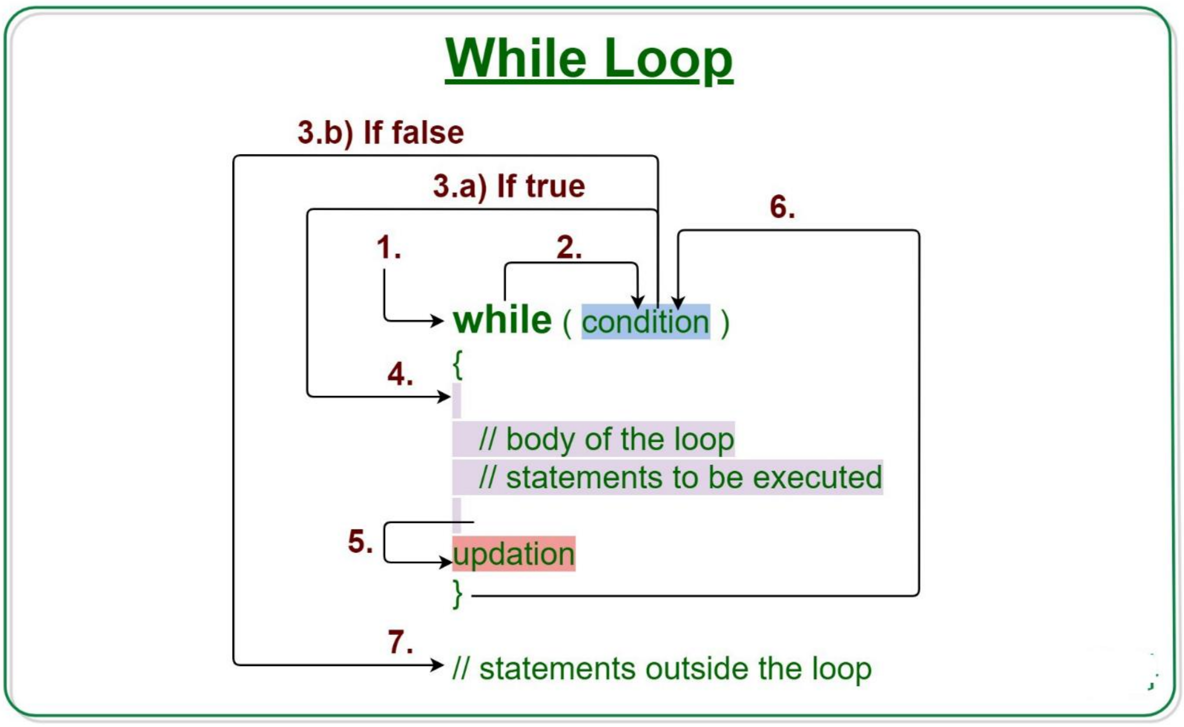
12.WAP to swap two numbers
a. using third variable
b. without using third variable

a)
```java
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);
    }
}
```

b)
```java
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

```java
class PrimeNumber {
    public static void main(String args[]) {
        int num = 13;
        boolean flag = true;
        for (int i = 2; i < num; ++i) {
// condition for non-prime 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. ");
    }
}
```

While Loop



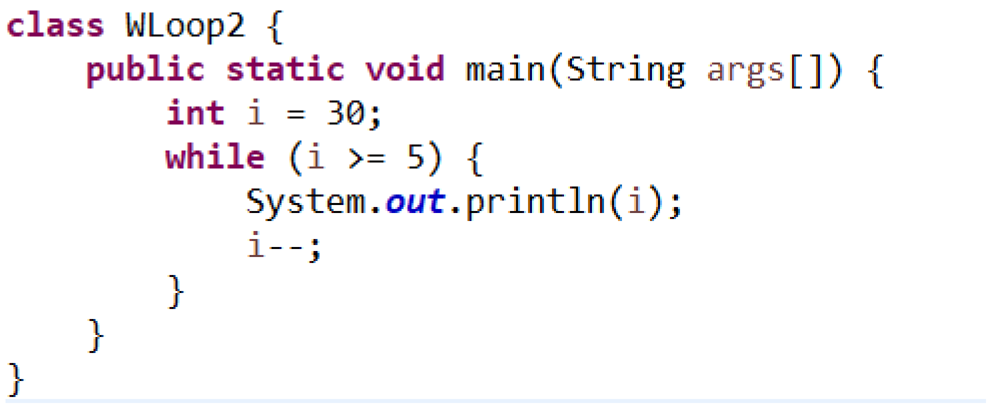
1.WAP to print 1 to 30 using while loop

```
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

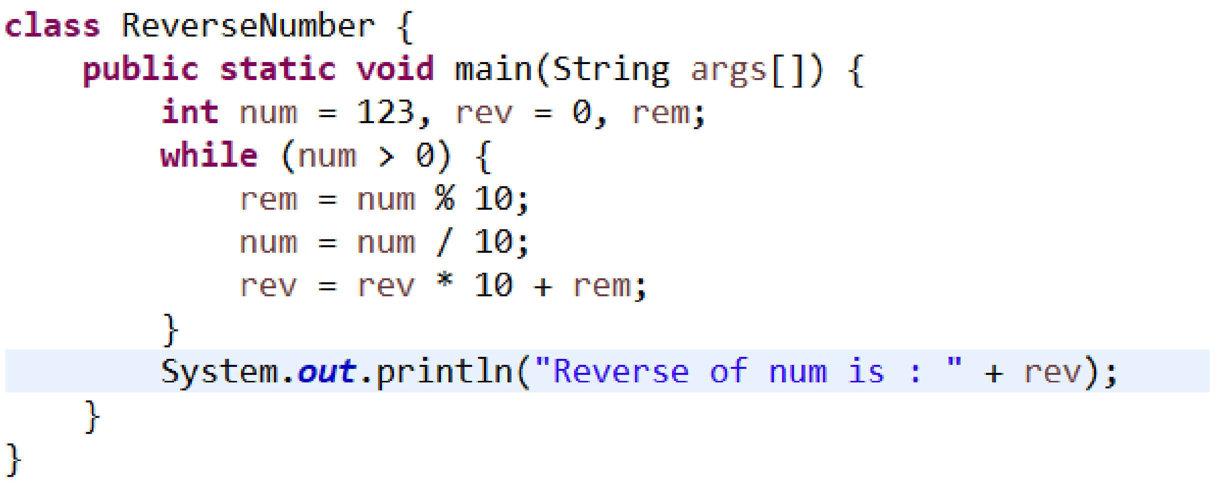
2.WAP to print 30 to 5 using while loop



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

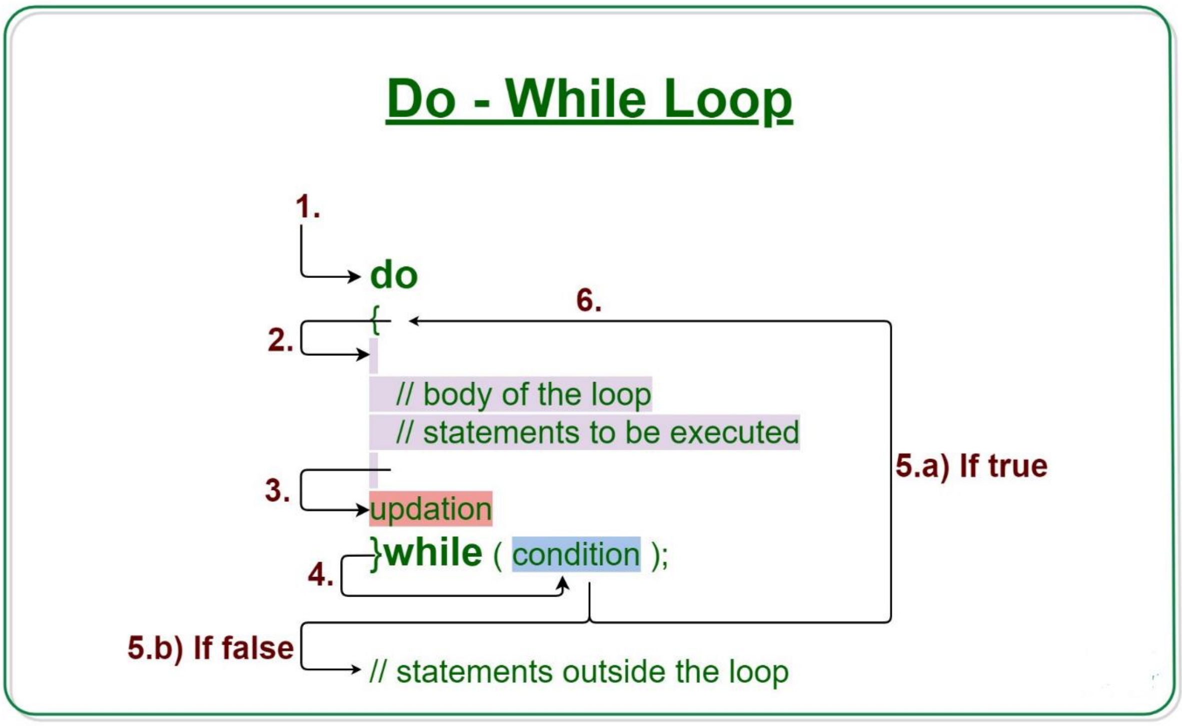
Note : - In the above program once condition is false JVM come out of loop and print i value only once.
And i value is no more 30 it changes to 4 means whatever we did in while loop with i is going to updated

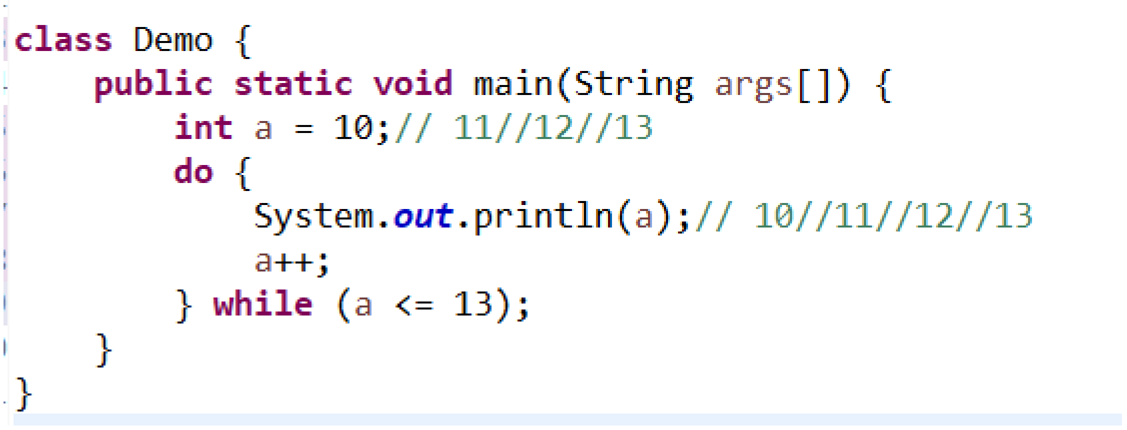
5.WAP to find reverse of a number



```
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);
    }
}
```

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 number of iteration priory(already).
syntax: for (initialisation; condition; increment / decrement)
{
}

while loop : -

----

we go for while loop if we don't know number of iterations.

syntax: initialisation;

while(condition)

{

Increment /decrement;

}

do while loop : -

----

when we want our loop should run at least one time then we go for do-while loop

syntax: do

{

Increment /decrement;

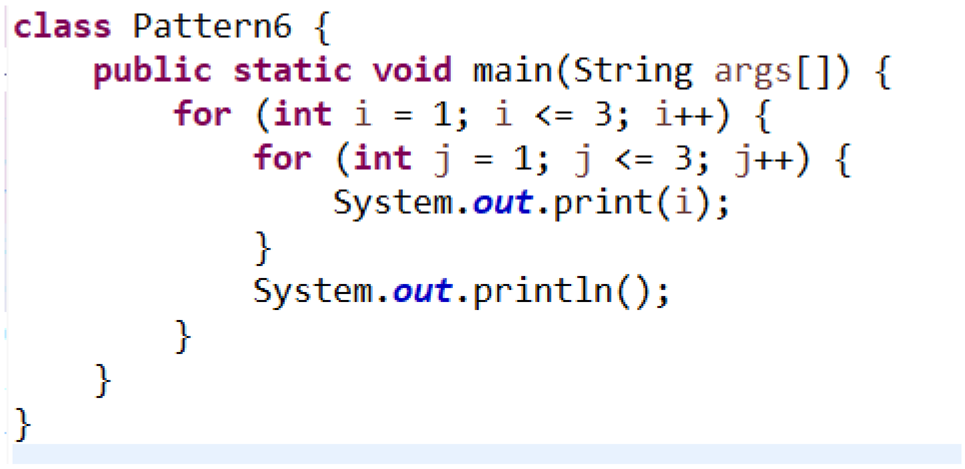
} while (condition);

Pattern programming

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

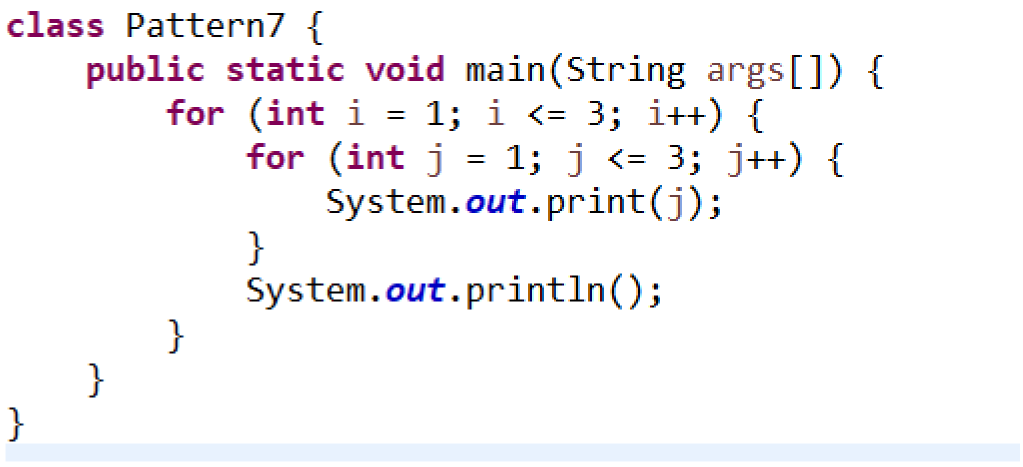
```
class Star {
    public static void main(String args[]) {
        for (int i = 1; i <= 3; i++) // outer loop for no of rows
        {
            for (int j = 1; j <= 5; j++) // inner loop for no of columns
            {
                System.out.print("*");
            }
            System.out.println("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

2.
111
222
333



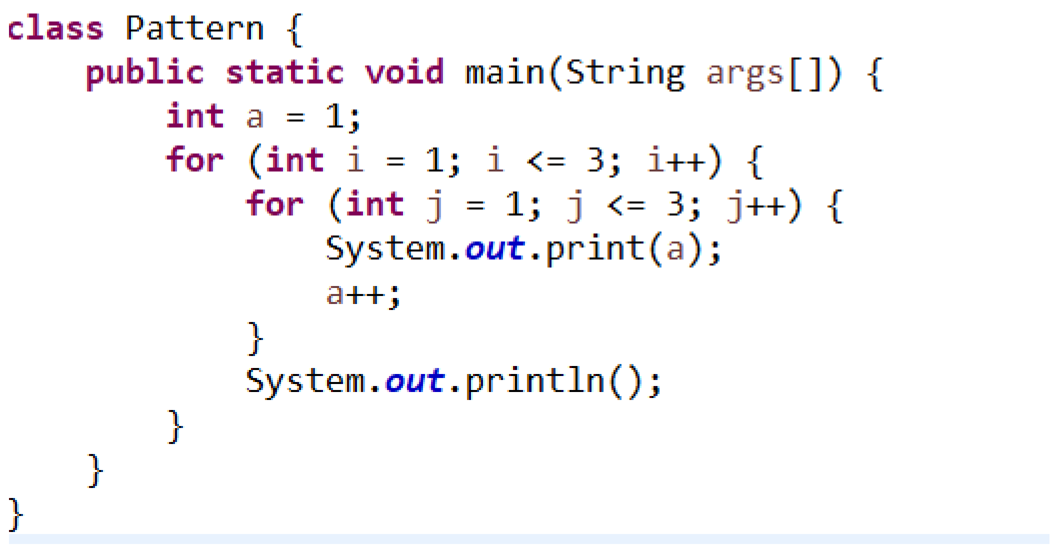
```
class Pattern6 {
    public static void main(String args[]) {
        for (int i = 1; i <= 3; i++) {
            for (int j = 1; j <= 3; j++) {
                System.out.print(i);
            }
            System.out.println();
        }
    }
}
```

3.
123
123
123



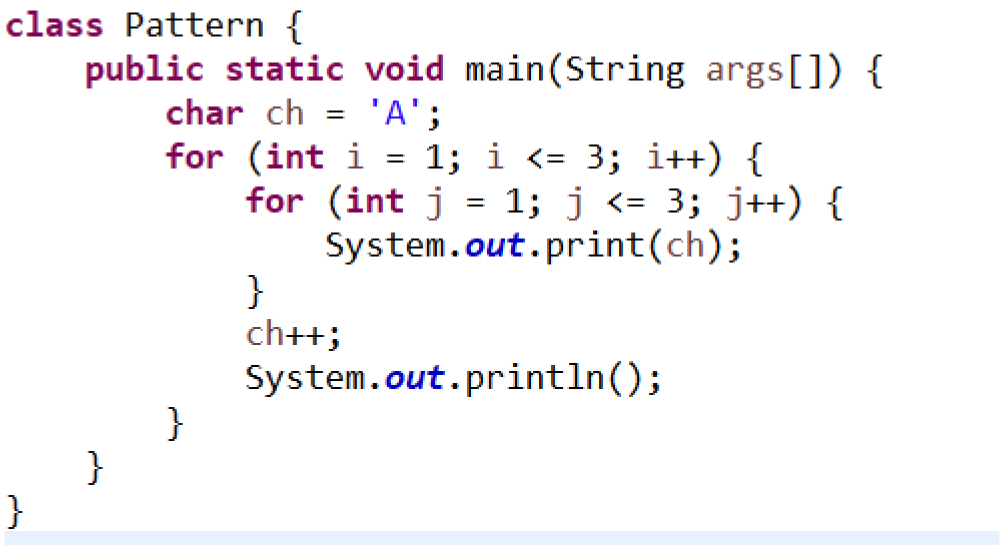
```
class Pattern7 {
    public static void main(String args[]) {
        for (int i = 1; i <= 3; i++) {
            for (int j = 1; j <= 3; j++) {
                System.out.print(j);
            }
            System.out.println();
        }
    }
}
```

4.
123
456
789



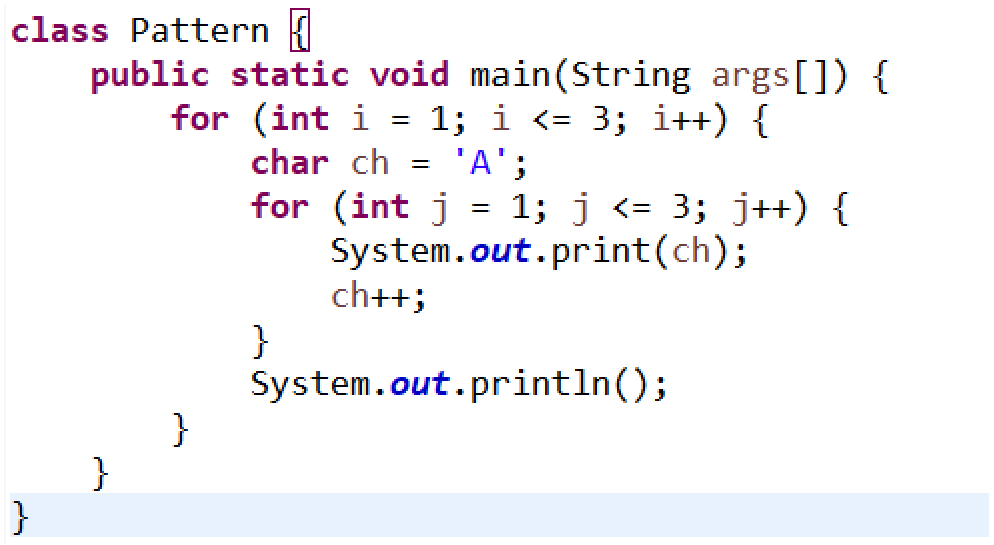
```
class Pattern {
    public static void main(String args[]) {
        int a = 1;
        for (int i = 1; i <= 3; i++) {
            for (int j = 1; j <= 3; j++) {
                System.out.print(a);
                a++;
            }
            System.out.println();
        }
    }
}
```

5.
AAA
BBB
CCC



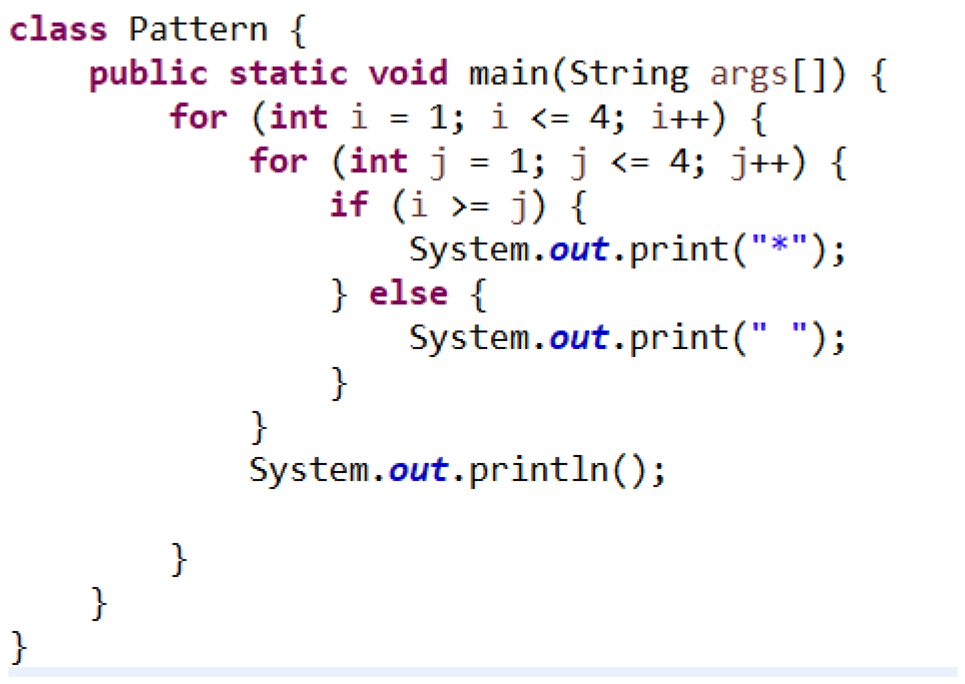
```java
class Pattern {
    public static void main(String args[]) {
        char ch = 'A';
        for (int i = 1; i <= 3; i++) {
            for (int j = 1; j <= 3; j++) {
                System.out.print(ch);
            }
            ch++;
            System.out.println();
        }
    }
}
```

6.
ABC
ABC
ABC



```java
class Pattern {
    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();
        }
    }
}
```

Q)
```
*
**
***
****
```



```java
class Pattern {
    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();

        }
    }
}
```

TYPE CASTING

The process of converting one data type into another data type is known as type casting

Type casting is classified into: -
1.Primitive type casting
a. widening(implicit)
b. Narrowing(Explicit)

2.Non primitive type casting
a. Upcasting (Implicit)
b. Down-casting(Explicit)

Primitive Type casting: - The Process of converting one primitive datatype into another data type is known as Primitive type casting

Primitive type casting is classified into,
1.Widening
2.Narrowing

Widening

The process of converting narrow(smaller) datatype into wider(larger) datatype is known as widening.

```
char-->int-->long-->float-->double
----------------------------------------->
            WIDENING
```

Note: -
1)There is no possibility of data loss in widening, therefore compiler performs widening automatically.
2)Widening is also known as Auto-widening or implicit type casting.

Eg:- int a=10;
Double b=a;

Byte a=10;
Int b=a;

Narrowing

The process of converting wider(larger) datatype into narrow(smaller) datatype is known as narrowing.

```
char<--int<--long<--float<--double
<---------------------------------
            NARROWING
```

Note: -
1)There is a possibility of data loss in narrowing, therefore compiler does not perform narrowing automatically.
2)Narrowing is also known as explicit type casting.

Note: - Narrowing must be performed explicitly by the programmer with the help of cast operator.

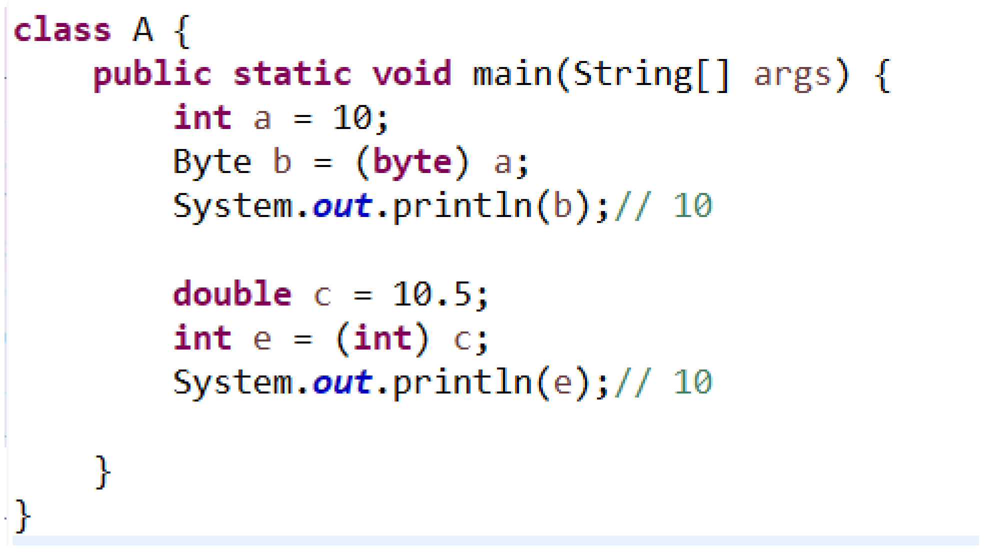
Cast operator

Cast operator is a unary operator.

Syntax --> (datatype)operand;

Note: - operand can be a value, variable or expression.

Eg:-



```
class A {
    public static void main(String[] args) {
        int a = 10;
        Byte b = (byte) a;
        System.out.println(b);// 10

        double c = 10.5;
        int e = (int) c;
        System.out.println(e);// 10

    }
}
```

Method

Method is a block of instructions which is used to perform specific task.

syntax--> Acessmodifier returntype methodname ([datatype variable1,datatype variable2,...])

```
     {
         Statements;
       }
```

Method header includes,
1)Modifier
2)returntype
3)methodName
4)formal arguments

method signature

Combination of method name and formal arguments is known as method signature.

Method body/Method implementation

The block of the method which includes the instructions to be executed, is known as method body or method implementation.

returntype

It specifies the type of value returned by the method, It can be
1)Primitive datatype
2)Non primitive datatype

3)void

based on the type of value returned by the method.
If method returns primitive value, then returntype must be suitable primitive datatype.
If method returns non primitive value, then returntype must be suitable non primitive datatype.
If method doesn't return any value, then returntype must be void.

void

-----

void is a special returntype which specifies no value.

methodName

----

methodName is an identifier, therefore we must follow all the 4 rules for identifiers while writing methodName and also we should follow following conventions.

Conventions for methodName

----

1)If methodName is a single word, then all the letters should be in lowercase.
eg:-test(), print() etc.
2)If it is multiword ,from the second word onwards each new word should start with uppercase.
eg: - toPrint() , toPrintEvenNumbers() etc.

Formal argument

The variables declared in the method header are known as formal arguments and these variables acts as local variables.

Note: -A method gets executed only when it is called.

Note: -main method is automatically called by JVM.

Note: -We can call a method for execution using method call statement.

Method call statement

It is used to call a method for execution. It transfers the control from calling method to called method.

syntax--> methodName ([actual arguments])

Eg: -

```java
public class Problem {
    public static void test() {// called method
        System.out.println("Test started");
        System.out.println("test ended");
    }

    public static void main(String[] args) {// calling method
        System.out.println("main() started");
        test();// method call statement
        System.out.println("main() ended");
    }
}
```

Output

```
main() started
Test started
test ended
main() ended
```

Actual argument

These are the inputs given to formal arguments.

calling method

A method which calls another method for execution is known as calling method.

called method

A method which is being called is known as called method.

Advantages of creating method

----

1)Code reusability
2)Number of lines of code reduces
3)Readability of the code will increase.

Note: -While calling methods with formal arguments,

1)The number of formal argument and actual arguments must be same.
2)The type of actual argument and formal argument must be same or we can have an actual argument which can be widened to formal argument type.

Eg: -

```
public class Problem {
    public static void add(int a, int b) {
        int sum = a + b;
        System.out.println(sum);
    }

    public static void main(String[] args) {
        add(1, 2);
        add(10);// CTE
        add(10, 20, 30);// CTE
        add(10, 'a');// widening
        add(10, 12.6);// CTE
    }
}
```

Return statement

It transfers the control from called method to calling method.

Note: -Whenever the returntype of the method is any type other than void, then that method must have return statement.

Eg:-

```
public class Problem {

    public static int add(int a, int b) {
        int sum = a + b;
        return sum;
    }

    public static void main(String[] args) {
        int sum = add(1, 2);
        System.out.println(sum);
    }
}
```

<u>OUTPUT</u>

```
3
```

Note: -

We can have the return statement inside a method which has void as its returntype, But that method should not return any value.

Eg: -

```
public class Problem {
    public static void add(int a, int b) {
        int sum = a + b;
        return;
    }

    public static void main(String[] args) {
        add(1, 2);
    }
}
```

Note: -return statement must be the last statement of the block, If there is any statement after return inside the same block, Then compiler generates CTE.

Eg: -

return 10;	return 10,20;→CTE
return a;	return a,b;→CTE
return a+b;	return(a+b),(a-b)→CTE

Static method

A method which is prefixed with static keyword is known as static method.

The static method body/implementation will be stored in method area and its reference will be stored in class static area.

The static methods of the class can be used in two ways inside the same class.

1)directly

2)using ClassName

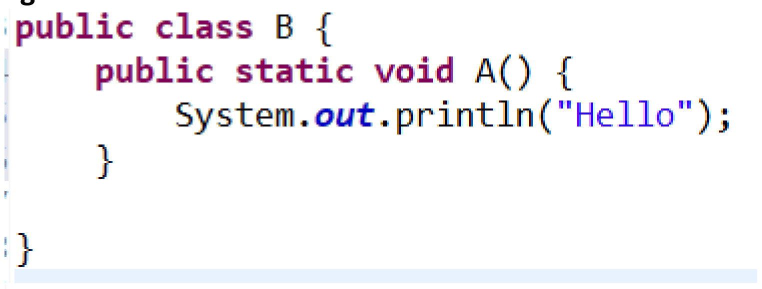
Eg: -

```
public class Problem {
    public static void A() {
        System.out.println("Hello");
    }

    public static void main(String[] args) {
        A();
        Problem.A();
    }
}
```

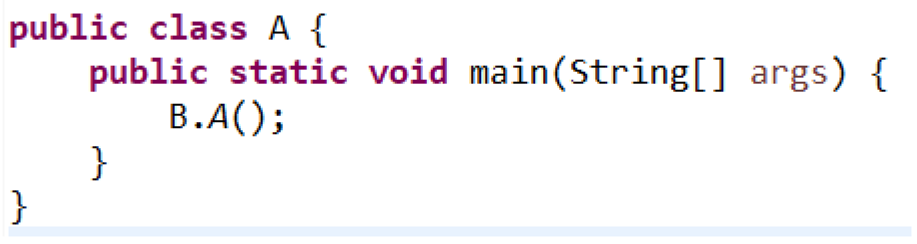
OUTPUT
Hello
Hello

The static method of one class can be used in another class with the help of ClassName.

Eg: -



```java
public class B {
    public static void A() {
        System.out.println("Hello");
    }

}
```



```java
public class A {
    public static void main(String[] args) {
        B.A();
    }
}
```

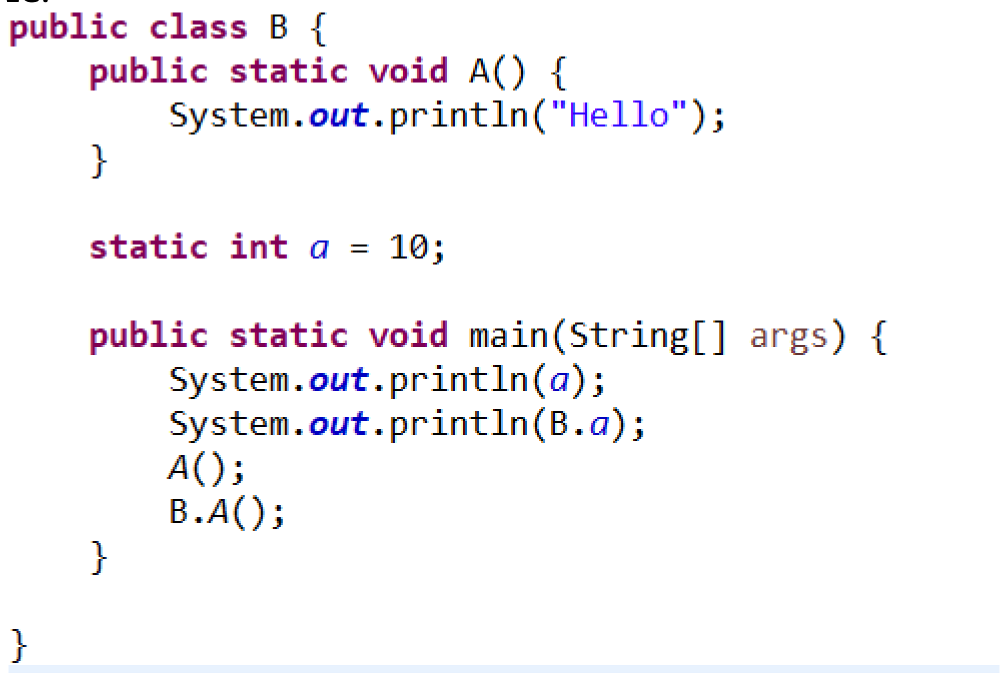
Class Loading Process

----
1)All the static variables and methods of the class gets loaded in the class static area.
2)static variables will be first assigned/initialized with default values.
3)All the static initializers gets executed in top to bottom order.

Conclusion

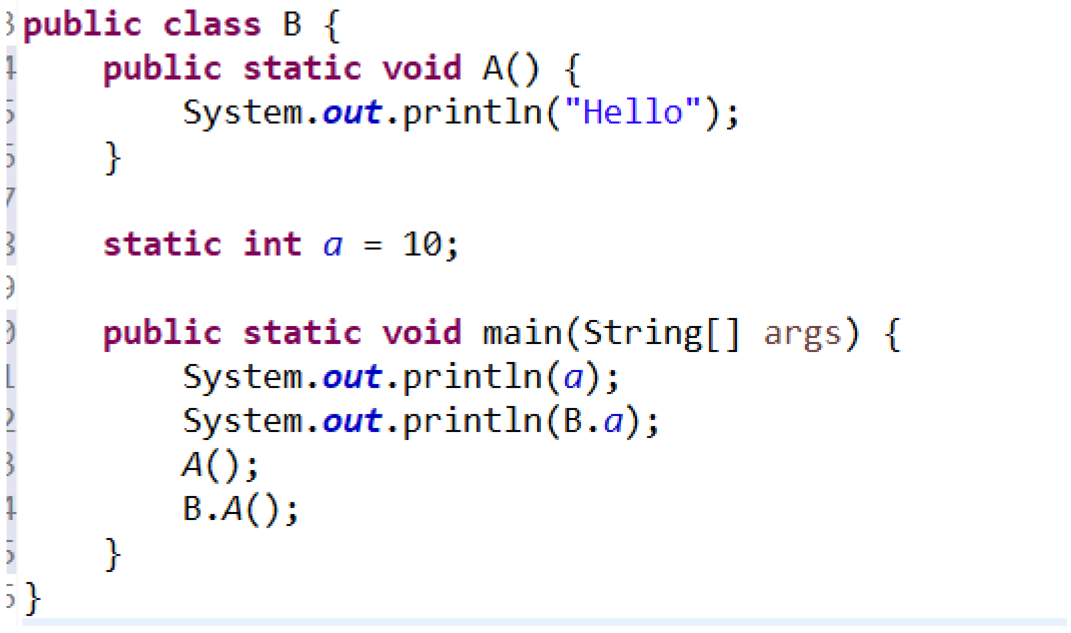
----
1)The static variable and methods of a class can be used inside the same class either directly or with the help of Classname.
2)The static variable and methods of a class can be used outside the class with the help of Classname.

EG: -

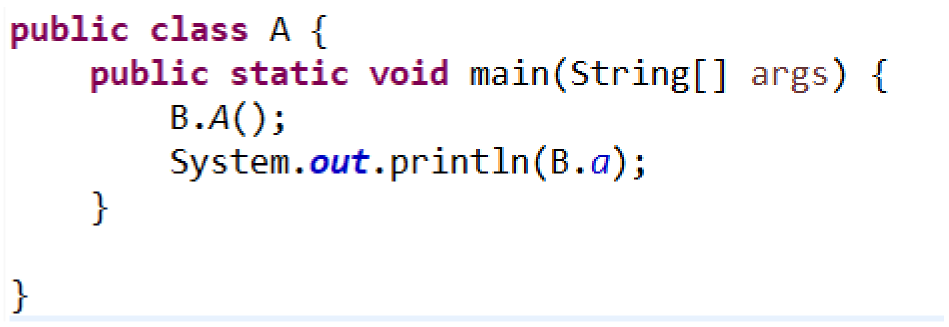


```java
public class B {
	public static void A() {
		System.out.println("Hello");
	}

	static int a = 10;

	public static void main(String[] args) {
		System.out.println(a);
		System.out.println(B.a);
		A();
		B.A();
	}

}
```

Eg: -



```java
public class B {
	public static void A() {
		System.out.println("Hello");
	}

	static int a = 10;

	public static void main(String[] args) {
		System.out.println(a);
		System.out.println(B.a);
		A();
		B.A();
	}
}
```



```java
public class A {
	public static void main(String[] args) {
		B.A();
		System.out.println(B.a);
	}

}
```

Non static method

A method without static keyword is known as non-Static method.

```
public class A {
    public void A1() {
        System.out.println("Hello");
    }

    public static void main(String[] args) {
        A a1 = new A();
        a1.A1();
    }
}
```

__OUTPUT__

Hello

Q.WAP for area of triangle, 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));
    }
}
```

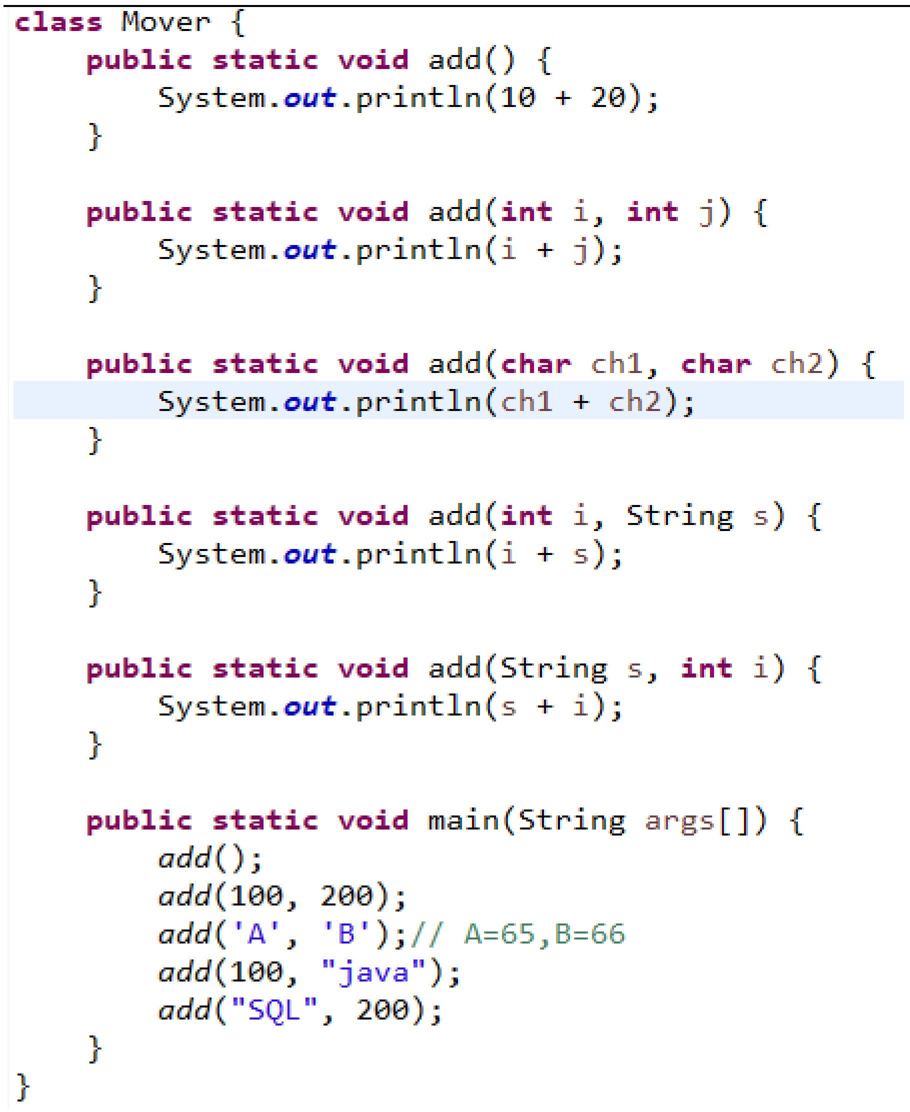
Method overloading

A class having more than one method with same name but different formal arguments is known as method overloading.

In method overloading name of the methods are same but the list of formal argument differs either in length, type or order.

test (), test () --->not method overloading
test (), test (int a) -->method overloading
test (int a), test (int b) --> not method overloading
test (int a), test (int a, int b)-->method overloading
test (int a), test (double b) --->method overloading
test (int a, double b), test (double a, int b) --->method overloading

Eg:-



```java
class Mover {
    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);
    }

    public static void main(String args[]) {
        add();
        add(100, 200);
        add('A', 'B');// A=65,B=66
        add(100, "java");
        add("SQL", 200);
    }
}
```

Note:-For method overloading return type of the method is not considered.

Note: In method overloading the link between methodcall statement and method implementation happens during compilation and it is done based on the arguments passed in the method call statement.

Array

Array is a continuous block of memory which is used to store multiple values of same type. OR

Array is a continuous block of memory which is used to store homogeneous values.

How to create an array reference variable?

Array reference variable is used to store address or reference of the array created.
Syntax
----
datatype [] variable ; OR datatype variable[];

Eg: - int [] a;
double [] b;
boolean [] c;
In the above example a, b, c are array reference variables.

An array reference variable can store 2 types of values,
1)reference of the array created
2)null

Note:-If the array reference variable is stored with null, that means it is not referring to any array. (i.e. Array is not created).

Int [] a=null;

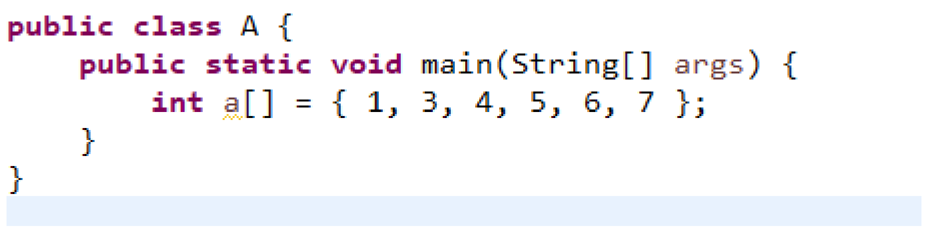
Note: -null is a keyword in java which specifies nothing.

How to create an Array?

We can create array in 2 ways,

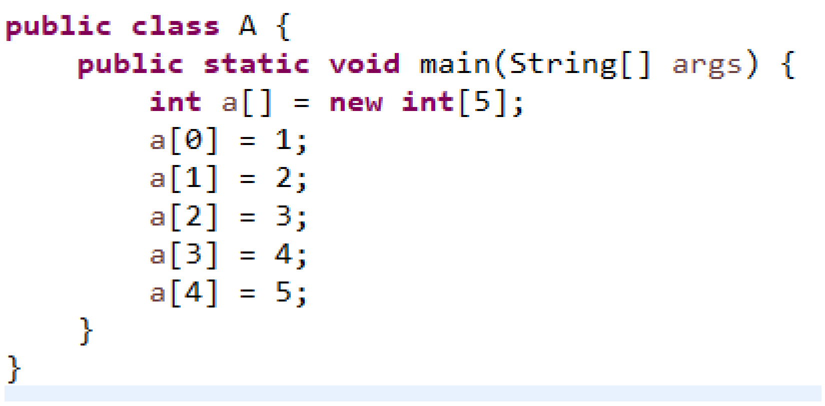
1)Using declaration and initialization statement
2)Using new keyword.

1)Using declaration and initialization statement



```
public class A {
    public static void main(String[] args) {
        int a[] = { 1, 3, 4, 5, 6, 7 };
    }
}
```

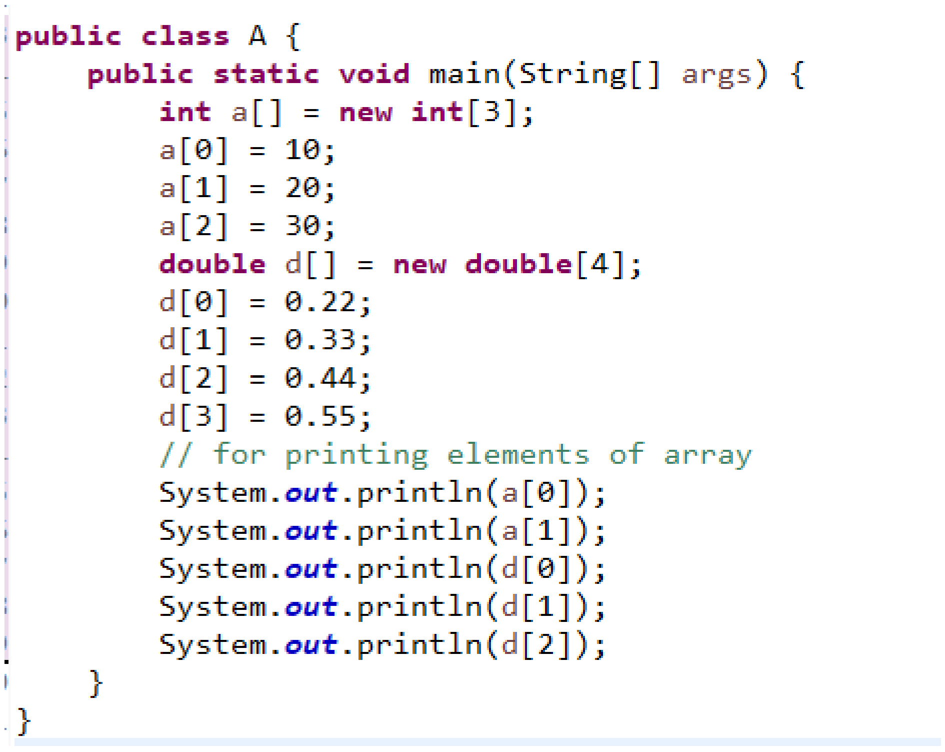
2)Using new keyword.



```
public class A {
    public static void main(String[] args) {
        int a[] = new int[5];
        a[0] = 1;
        a[1] = 2;
        a[2] = 3;
        a[3] = 4;
        a[4] = 5;
    }
}
```

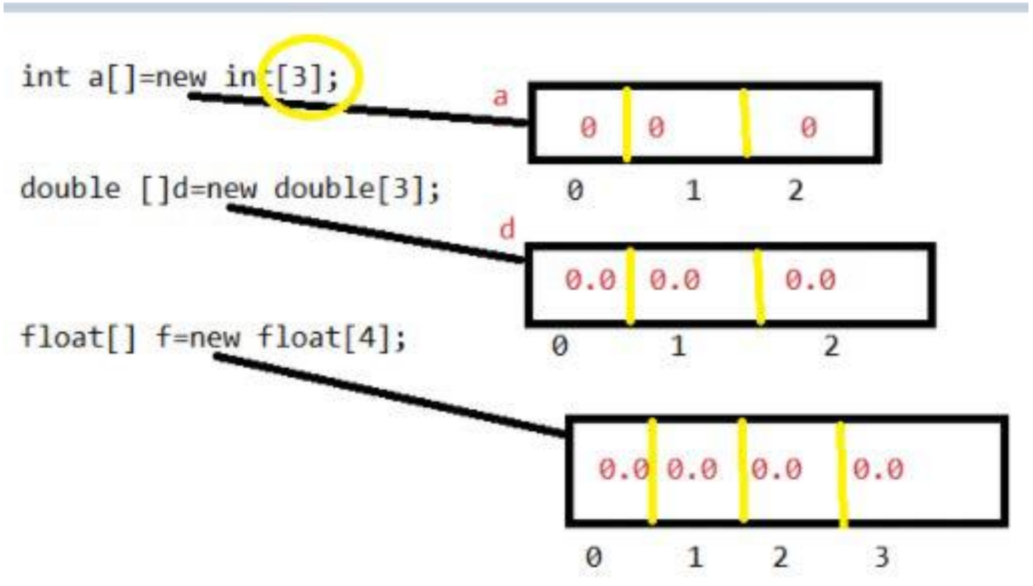
Syntax:
arraytype arrayname [] =new arraytype[size];
arraytype [] arrayname =new arraytype[size];
arraytype [] arrayname =new arraytype[size];
Ex: int a [] =new int [3];
double [] d =new double[3];
float [] f=new float [4];

- 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 array default values will be present.
- storing elements in an array by using index values

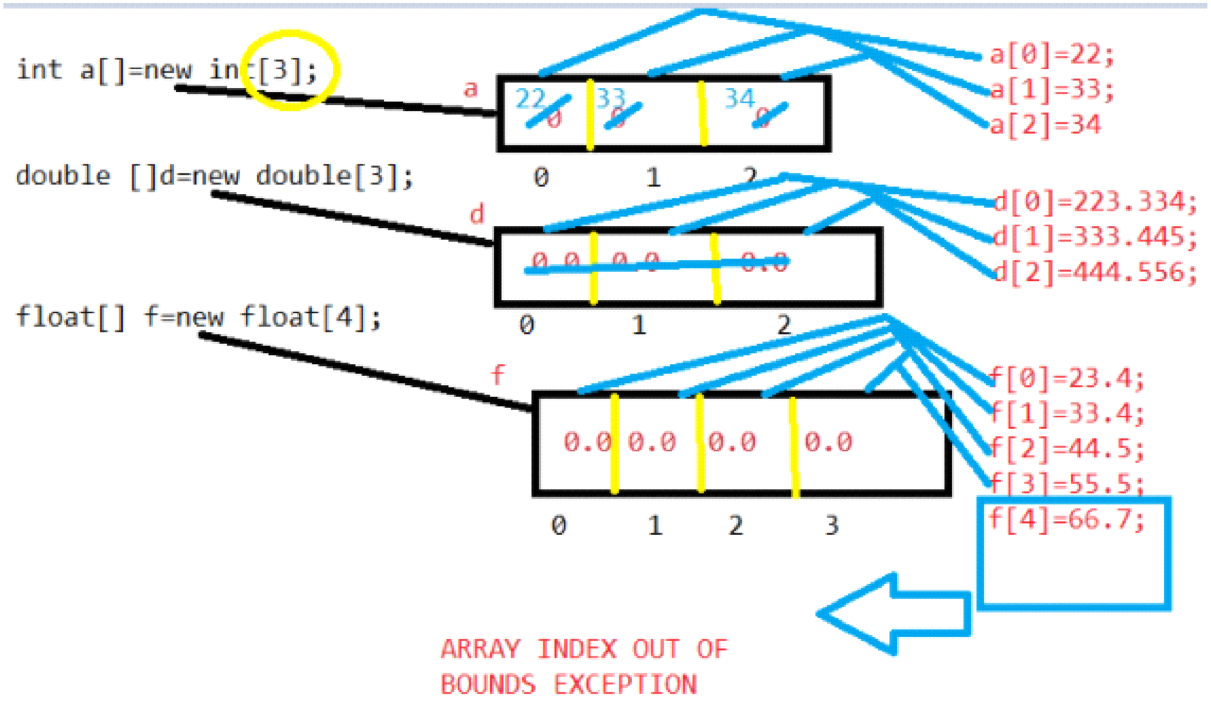


```
public class A {
    public static void main(String[] args) {
        int a[] = new int[3];
        a[0] = 10;
        a[1] = 20;
        a[2] = 30;
        double d[] = new double[4];
        d[0] = 0.22;
        d[1] = 0.33;
        d[2] = 0.44;
        d[3] = 0.55;
        // for printing elements 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//



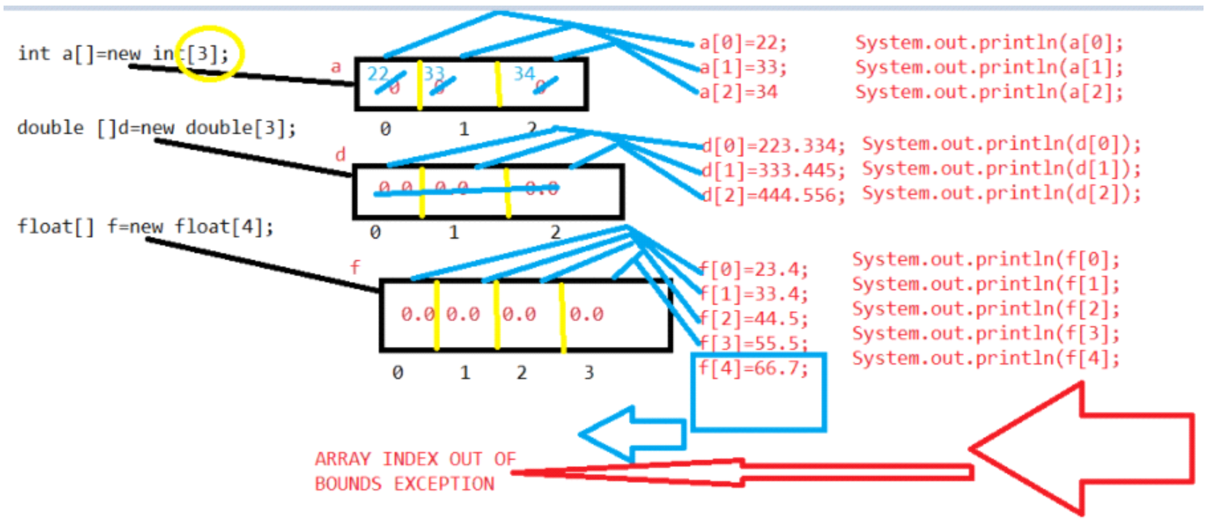
Length variable:

• It provides length of array and length will always calculated from 1.

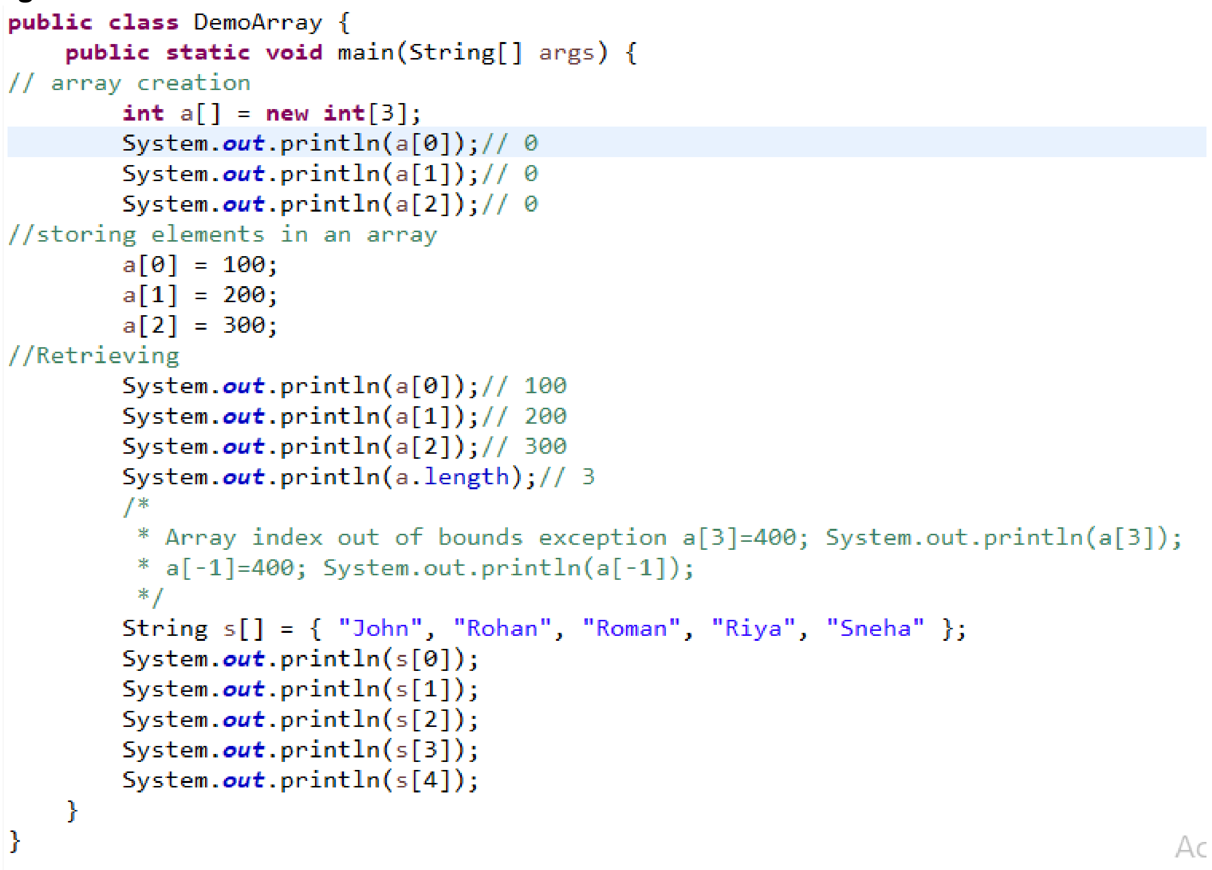
ex: int a[]=new int[300];
System.out.println(a.length);
Output is :300

• 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



Eg:-

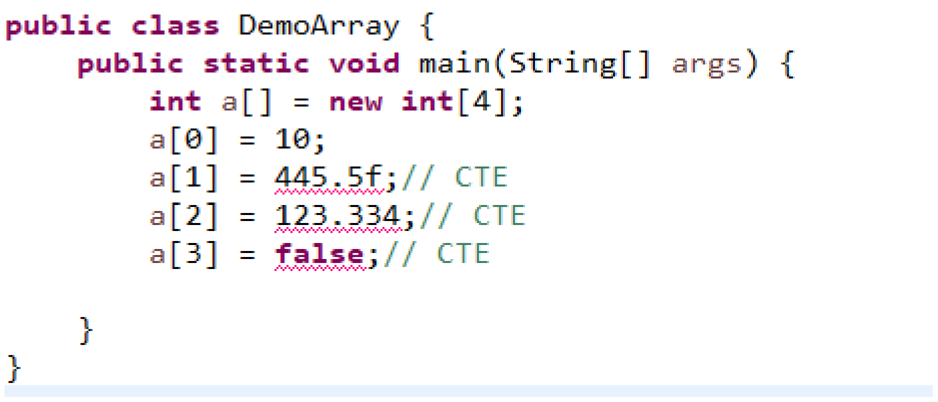


```
public class DemoArray {
    public static void main(String[] args) {
// array creation
        int a[] = new int[3];
        System.out.println(a[0]);// 0
        System.out.println(a[1]);// 0
        System.out.println(a[2]);// 0
//storing elements in an array
        a[0] = 100;
        a[1] = 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:

```
000
100
200
300
3
John
Rohan
Roman
Riya
Sneha
```

• Array stores only homogenous data.



```
public class DemoArray {
    public static void main(String[] args) {
        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 will get compile time error.

NullPointerException

Whenever array Reference Variable is initialized with null and if we try to use the variable for accessing/storing the value from the array we will get NullPointerException.

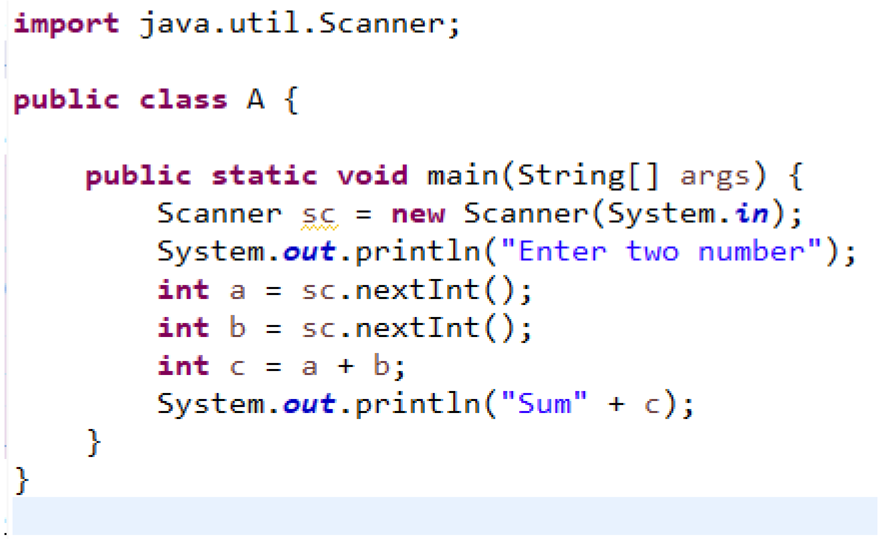
```
class H
{
        public static void main(String[] args)
        {
                int[]a=null;
                System.out.println(a[0]); //NullPointerException
        }
}
```

Scanner Class

Scanner class is used to take the input from the users

Data type	**Scanner Class**
byte	**nextByte()**
short	**nextInt()**
int	**nextInt()**
long	**nextLong()**
double	**nextDouble()**
char	**next().charAt(0);**
String	**next() or nextLine()**

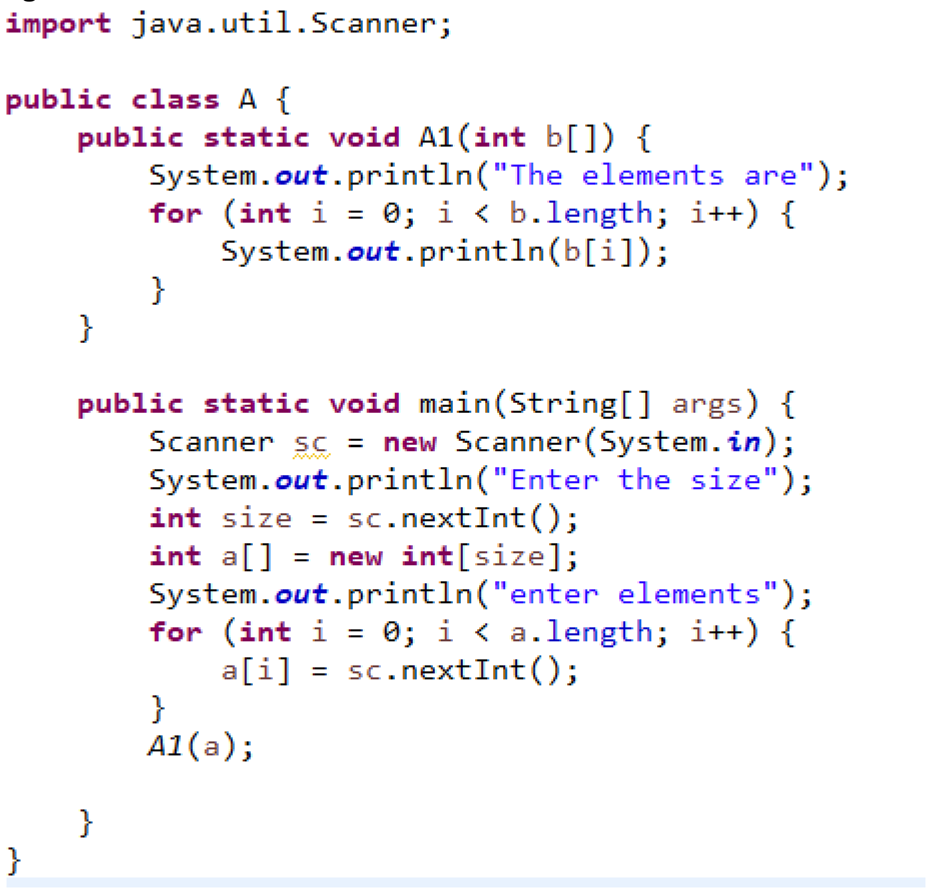
Eg:



```java
import java.util.Scanner;

public class A {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter two number");
        int a = sc.nextInt();
        int b = sc.nextInt();
        int c = a + b;
        System.out.println("Sum" + c);
    }
}
```

Passing an array to a method

Eg: -



```java
import java.util.Scanner;

public class A {
    public static void A1(int b[]) {
        System.out.println("The elements are");
        for (int i = 0; i < b.length; i++) {
            System.out.println(b[i]);
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size");
        int size = sc.nextInt();
        int a[] = new int[size];
        System.out.println("enter elements");
        for (int i = 0; i < a.length; i++) {
            a[i] = sc.nextInt();
        }
        A1(a);

    }
}
```

Array gets created inside the Heap.
An array created inside one method can be used in another method, with the help of array reference.
We can pass array from one method to another method, with the help of array reference.

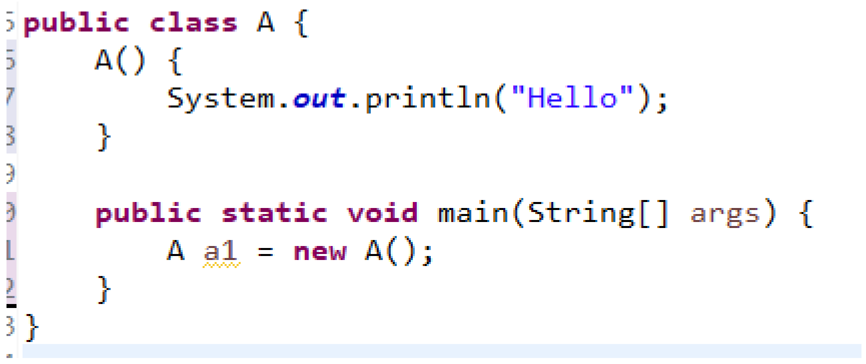
Note: -The method which accepts an array, its formal argument should be an array reference variable.

Constructor

A special non static member used for object creation.

```
Syntax--> AccessSpecifier Class Name([Formal arg])
           {
             Statements;
           }
```

Eg:-



```java
public class A {
    A() {
        System.out.println("Hello");
    }

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

<u>OUTPUT</u>

Hello

Note: If programmer fails to add a constructor inside the class then compiler by default adds a no argument constructor which is called as default constructor.

Eg:-

<u>Before</u> Compilation

```java
public class A {
public static void main(String[] args) {
         A a1=new A();
}
}
```

<u>After</u> Compilation

```java
public class A {
         A(){

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

Types of Constructor
----
1)No argument constructor
2)Parameterized constructor

No arguments constructor-->Constructor without any arguments is known as no argument constructor.
Eg:-
```
public class A {
	A(){
		System.out.println("hye");
	}
public static void main(String[] args) {
	A a1=new A();
}
}
```

Non parameterized Constructor

Parameterized constructor-->Constructor with arguments is known as parameterized constructor.
Eg:-
```
public class A {
	A(int a){
		System.out.println("hye");
	}
public static void main(String[] args) {
	A a1=new A(10);
}
} Parameterized constructor
```

Note:-Default constructor will be added only when there is no constructor added by the programmer.
(if there is atleast one constructor added by the programmer in the class then default constructor is not added by the compiler).

```
Eg:- public class A {
	A(int a){
		System.out.println("hye");
	}
public static void main(String[] args) {
	A a1=new A(10);
	A a2=new A();//CTE
}
}
```

In this case default constructor is not added by the compile

Constructor Overloading

A class having more than one constructor with different parameters is known as constructor overloading.

Eg:-

```
public class A {
	A(int a){
		System.out.println("hye");
	}
	A(int a,int b){
		System.out.println("Hello");
	}
public static void main(String[] args) {
	A a1=new A(10);
	A a2=new A(10,20);
}
}
```

OUTPUT

```
Hye
Hello
```

this

----

this is a keyword in java, it is a predefined local object reference variable which is present in every non static context.

It stores the reference of current object.

Eg:-

```
 public class A {
int a=10;
A(){
	int a=20;
	System.out.println(a);//20
	System.out.println(this.a);//10
}

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

Note:-Inside the non-static context, if there is any local variable having same name as non static variable, In this case in order to use non static variable we must make use of "this".

Note:-We cannot use "this" from static context(i.e static block or static method), It generates CTE.

```
Eg:- public class A {
int a=10;
A(){
        int a=20;
        System.out.println(a);
        System.out.println(this.a);//CTE
}

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

Purpose of using parameterized constructor

The parameterized constructor can be used for initializing the non-static variables of a class at the time of object creation.

Eg:-

```
public class A {
 String name;
 A(String name){
this.name=name;
}

public static void main(String[] args) {
        A a1=new A("Hari");
        System.out.println(a1.name);\\Haru
}
}
```

Block

A block is a set of code enclosed within curly braces {} within any class.

static block

The blocks prefixed with static keyword are known as static blocks. The static blocks are also known
as static intializer block.(SIB)

syntax-->

```
static
{
Statements;
}
```

Note:-
----
1)Static blocks doesn't have name, and they cannot be called explicitly.
2)static bocks doesn't have return type therefore they cannot return a value.
3)static blocks gets executed during loading process of the class.
4)A class can have more than one static block in a class and they gets executed in top to bottom order.

Eg:-
```
 public class B {
	static {
		System.out.println("Hello");
	}
public static void main(String[] args) {
	System.out.println("Hye");
}
}
```
OUTPUT
```
Hello
hye
```

Eg:-
```
public class B {
	static {
		System.out.println("Hello");
	}
	static {
		System.out.println("hye");
	}
public static void main(String[] args) {
	System.out.println("Bye");
}
}
```

OUTPUT
Hello
Hye
Bye

Non static Block(IIB)

A block without static keyword is known as non-static block. It is also a type of non-static initializer. Therefore it is also called as instance initializer block(IIB) .

Syntax---> {
               Statements;
             }

1)Non static blocks does not have name, therefore it cannot be explicitly called by programmer.
2)Non-Static block does not have returntype, therefore it cannot return value.
3)Non-Static block gets executed during loading process of an object.
4)We can have more than one non static block in a class, they gets executed in top to bottom order.

Eg:-

```
 public class A {
        {
               System.out.println("Sib-1");
        }

public static void main(String[] args) {
        System.out.println("Hello");
        A a1=new A();
}
}
```

OUTPUT
Hello
Sib-1

Eg:-
```
public class A {
	{
		System.out.println("Sib-1");
	}
	{
		System.out.println("Sib-2");
	}
public static void main(String[] args) {
	A a1=new A();
	System.out.println("Hello");
	A a2=new A();
}
}
```
OUTPUT
```
Sib-1
Sib-2
Hello
Sib-1
Sib-2
```

Note-The non-static blocks get memory inside Heap area when object is created

Note- The non-static blocks of the class gets inserted inside the constructor, before user defined/programmer written instructions by the compiler at the time of compilation.

Object Oriented Programming(OOP)

Object oriented programming is a theory/concept/technique which helps the programmer to represent real world scenarios thorough programming.

Important terminologies under OOP

Class
Object
Encapsulation
Inheritance
Polymorphism
Abstraction

Object

Object is a real world entity which has properties and behaviours.

Properties are also called as attributes/Fields
Behaviours are also called as Actions/tasks.

In java, the properties of an object is represented as variables(non static).
And behaviours are represented as methods(non static).

Since java supports object oriented programming it is called as Object Oriented Programming language.

Class

Class is a blueprint of an object which represents properties and behaviours of the object as variables and methods respectively.

OR

Class is a template of an object which represents properties as variables and behaviours as methods.

Pillers Of Oject Oriented Programming
--
Encapsulation-->Process of binding properties and behaviours of an object together

Inheritance-->Process of acquiring properties and behaviours of parent to child.

Polymorphism-->Ability of an object to exhibit more than one form.
Abstraction-->Process of hiding the implementation details

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.

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 :

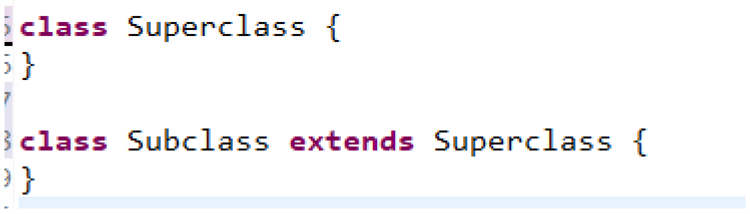
• extends is a keyword which indicates that we are creating a new class from an existing class.

```
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 class or derive class.



```
class Superclass {
}

class Subclass extends Superclass {
}
```

• Super class contains its own properties.

• 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.

Constructor chaining

The process of one constructor calling another constructor is known as constructor chaining.

We can achieve constructor chaining in two ways,
1)using super() statement
2)using this() statement

super() statement

----

super() statement is used to call constructor of superclass.

Rules for using super() statement

----

1)super() statement can be used only inside the constructor.
2)super() statement must be the first statement of the constructor.
3)If programmer fails to add super() staement inside the constructor, compiler by default adds a no argument super() statement.

Q)When the superclass has more than one constructor, which constructor of superclass is called by super() statement?

Ans) It depends on the arguments passed in the super() statement.

Eg:-super()--> calls no argument constructor of superclass
 super(10)-->calls a construcor of superclass which accepts an integer number.

Eg:-

```
 public class V {
V(){
        super();
        System.out.println("1st constructor");
}
}
public class V1 extends V {
V1(){
        super();
        System.out.println("2nd constructor");
}
public static void main(String[] args) {
        V1 v=new V1();
}
}
```

OUTPUT:-

```
1st constructor
2nd constructor
```

Eg 2:-

```
public class V {
V(){
        super();
        System.out.println("1st constructor");
}
V(int a){
        super();
        System.out.println("2nd constructor");
}
}
public class V1 extends V {
V1(){
        super();
        System.out.println("3rd constructor");
}
V1(int a){
        super(10);
        System.out.println("4th constructor");
}
public static void main(String[] args) {
        V1 v=new V1();
        V1 v1=new V1(10);
}
}
```

OUTPUT:-

```
1st constructor
3rd constructor
2nd constructor
4th constructor
```

Note:-For a class if extends keyword is not used, it by default extends a predefined class called Object class.

Object class is the supermost class for all the classes in java.

Uses of constructor chaining using super() statement

----

1)Constructor chaining using super() statement is essential to load all the non-static members of superclass to subclass object.
2)Constructor chaining using super() helps us to use programmer written instructions of superclass constructor.
3)Constructor chaining using super() helps us to send values from subclass to superclass.

this() statement

----

this() statement is used to call constructor of same class.

Rules for using this() statement

----

1)this() call statement can be used only inside the constructor.
2)this() statement must be the first statement of the constructor.
(i.e a construcor can have either super() or this() as its first statement.)
3)for a constructor which has this() statement, compiler doesn't add any instruction.
4)we cannot call constructor recursively using this() statement.It generates compile time error.

Eg:-

```
public class B {
        B(){
                this(10);
                System.out.println("1st constructor");
        }
        B(int a){
                this(10,20.5);
                System.out.println("2nd constructor");
        }
        B(int a,double b){
                System.out.println("3rd constructor");
        }
public static void main(String[] args) {
        B b1=new B();
}
}
```

OUTPUT:-

```
3rd constructor
2nd constructor
1st constructor
```

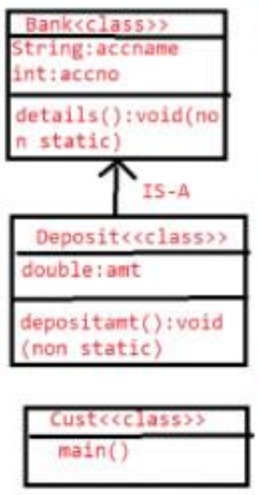
Types Of Inheritance

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

1. Single Level Inheritance

----

definition :- One super class and One sub class



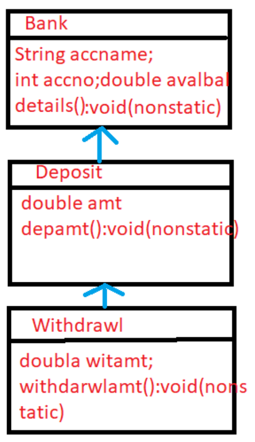
```
class Bank
{String Accname="Rohan";
   int acno=89898989;
   public void details()
   {
      System.out.println("Acc name:"+Accname+"\n Accno
:"+acno);
   }}
class Deposit extends Bank
{double bal=100;
   public void depamt()
   {
      System.out.println("Dposit amt"+bal);
   }}
public class Cust
{public static void main(String args[])
  {Deposit d1=new Deposit();
    d1.details();
    d1.depamt();
```

Eg; -

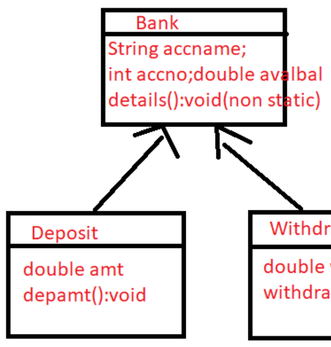
```
class Bank {
    String accname = "John";
    int accno = 245678;
    double avalbal = 100;

    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:

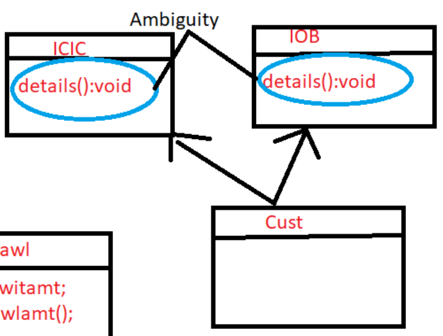
```
Account Holder : John accno:245678
Total balance after depositing Amt :4650.5
```



Multilevel



Hierarchical



Multiple

Multiple is not possible due to ambiguity probleum and Constructor chaining probleum

2. Multi-level Inheritance

----

definition :- Two super classes and Two sub classes

eg: -

```
class Bank
{
    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:

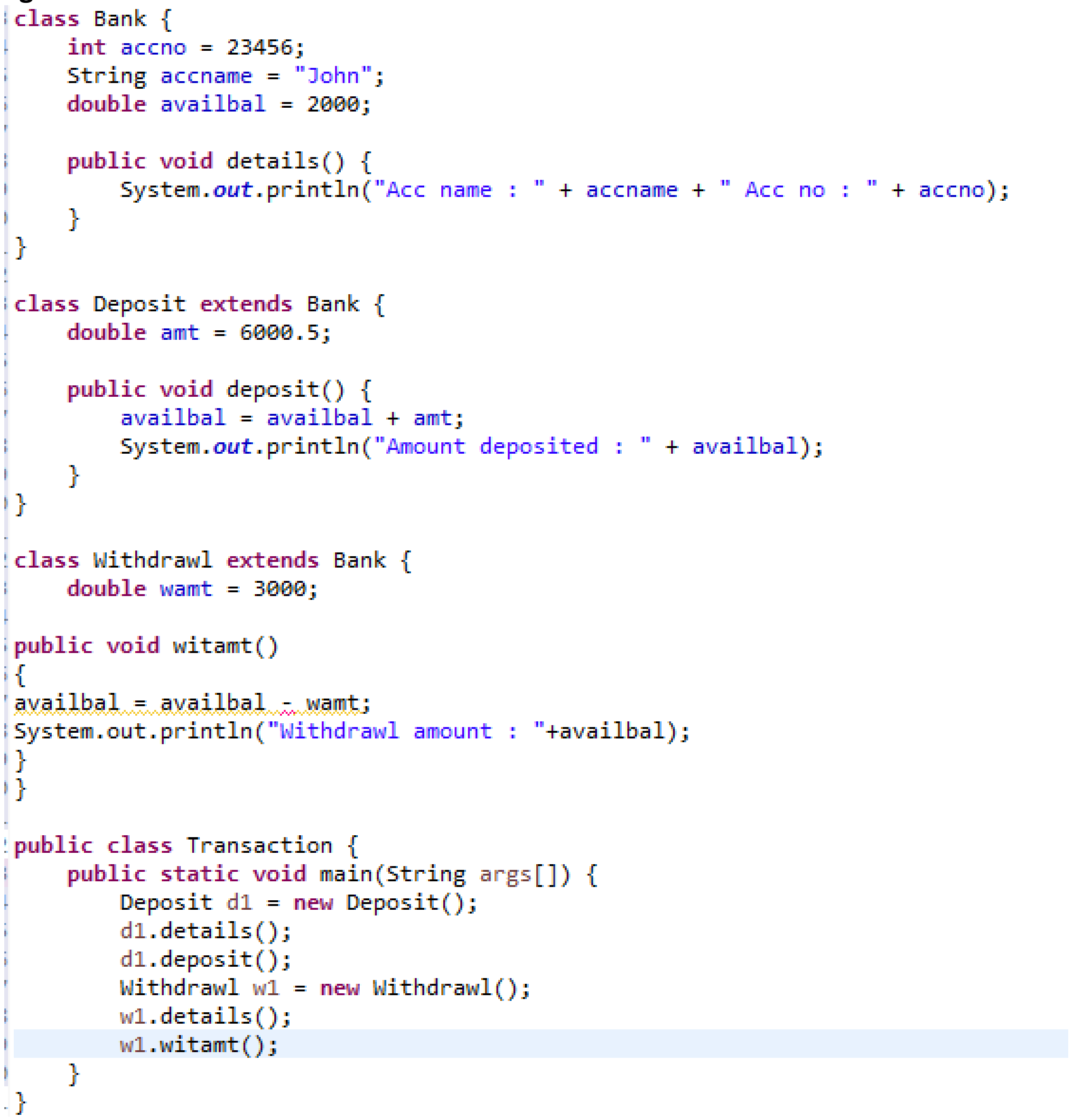
```
Acc name : John Acc no : 23456
Amount deposited : 8000.5
Withdrawl amount : 5000.5
```

3. Hierarchical Inheritance

----

definition :- One super class and Two sub classes

Eg: -



```
class Bank {
    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:

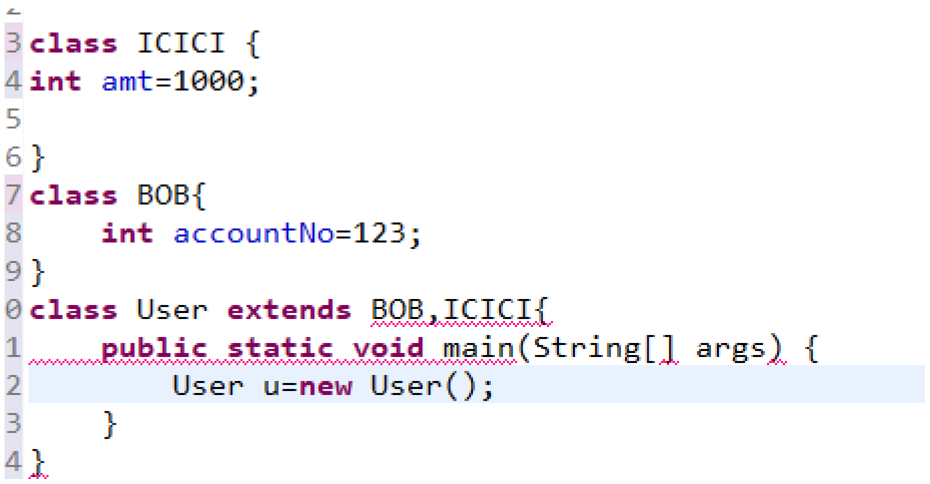
```
Acc name : John Acc no : 23456
Amount deposited : 8000.5
Acc name : John Acc no : 23456
Withdrawl amount : -1000.0 // here now availbal=2000,wamt=3000
```

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**.

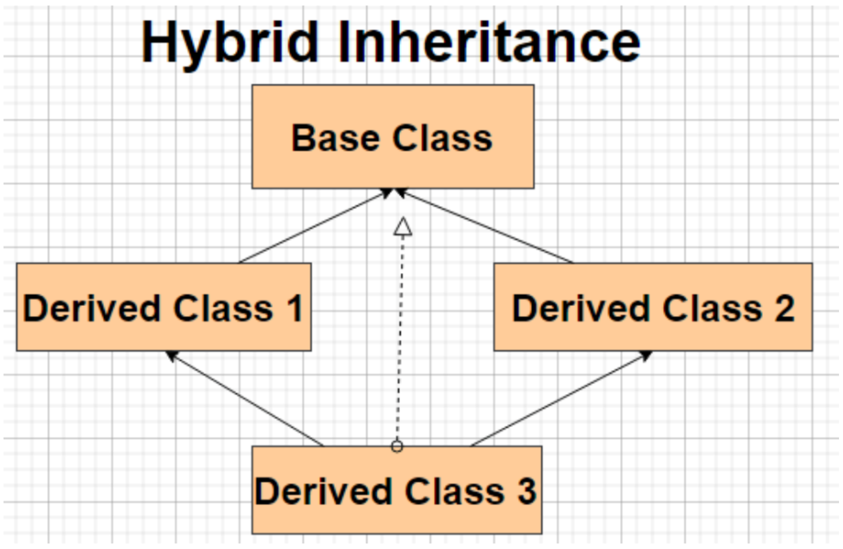


```
class ICICI {
int amt=1000;

}
class BOB{
    int accountNo=123;
}
class User extends BOB,ICICI{
    public static void main(String[] args) {
        User u=new User();
    }
}
```

class User extends ICICI,BOB // CTE because one class can't extends two classes at a time
}

5.Hybrid inheritence

Hybrid inheritance is the combination of heirarchical and multiple inheritance.



Note:- Since we cannot achieve multiple inheritance using classes we also cannot achieve hybrid inheritance.

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

Construction chaining problem

Non primitive typecasting(Derived typecasting)

--

The process of converting one type of reference into another type is known as non primitive typecasting or derived typecasting.

There are two types of Non primitive typecasting
1)UpCasting(Implicit)
2)DownCasting(Explicit)

Upcasting

The process of converting subclass reference into superclass type is known as upcasting.

Eg:-

```
class A{
int a=10;
int b=20;
}

public class B extends A{
        int c=30;
        int d=40;
}

public class Driver {
public static void main(String[] args) {
        A a1=new A();
        System.out.println(a1.a);//10
        System.out.println(a1.b);//20
        B b1=new B();
        System.out.println(b1.a);//10
        System.out.println(b1.b);//20
        System.out.println(b1.c);//30
        System.out.println(b1.d);//40
        A a2=b1;//upcasting
        System.out.println(a2.a);//10
        System.out.println(a2.b);//20
        System.out.println(a2.c);//CTE
        System.out.println(a2.d);//CTE
        }
}
```

Downcasting

The process of converting superclass reference into subclass type is known as downcasting.

EG:-

```
class A{
int a=10;
int b=20;
}

public class B extends A{
	int c=30;
	int d=40;
}

public class Driver {
public static void main(String[] args) {
	A a1=new A();
	System.out.println(a1.a);//10
	System.out.println(a1.b);//20
	B b1=new B();
	System.out.println(b1.a);//10
	System.out.println(b1.b);//20
	System.out.println(b1.c);//30
	System.out.println(b1.d);//40
	A a2=b1;//upcasting
	System.out.println(a2.a);//10
	System.out.println(a2.b);//20
	System.out.println(a2.c);//CTE
	System.out.println(a2.d);//CTE
	B b2=(B)a2;
	System.out.println(b2.a);//10
	System.out.println(b2.b);//20
	System.out.println(b2.c);//30
	System.out.println(b2.d);//40
	}
}
```

Note:- Downcasting is not automatically done by compiler. It must be done explicitly by the programmer
with the help of Cast operator.

ClassCastException

While downcasting if we don't have the instance of the class in the object created, we get a runtime exception called ClassCastException.

Generally when object is created for superclass and the reference is tried to be downcasted to subclass type, since superclass object will not have subclass instance we get ClassCastException.

Eg:-

```
class A{
int a=10;
int b=20;
}

public class B extends A{
	int c=30;
	int d=40;
}

public class Driver {
public static void main(String[] args) {
A a1=new A();
B b1=(B)a1;
System.out.println(b1.c);
}
}
```

OUTPUT:-

```
ClassCastException
```

instanceOf operator

----

instanceof operator is used to check whether instance of a given class is present in the object or not.

syntax--> object reference instanceof Classname

note:-
1)instanceof operator returns true, if the instance of given class is present in the object.
2)instanceof operator returns false, if the instance of given class is not present in the object.
3)It generates CTE, when there is no relation between the given class and the class for which object is created.

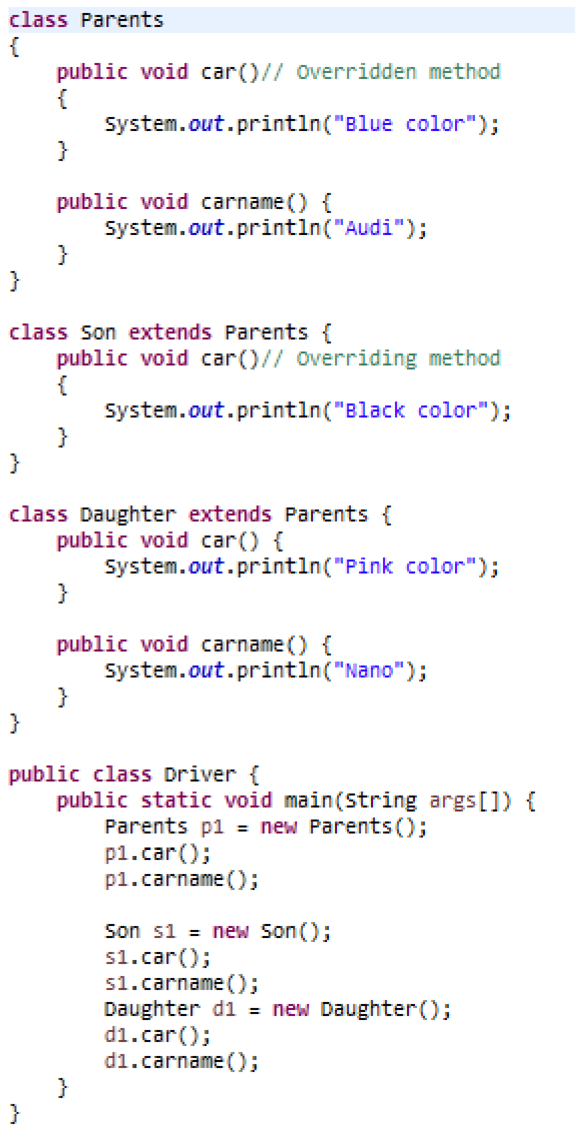
EG:-

```
class A{
int a=10;
int b=20;
}

public class B extends A{
	int c=30;
	int d=40;
}

public class C extends B {

}

public class Driver {
public static void main(String[] args) {
A a1=new A();
B b1=new B();
C c1=new C();
a1=b1;
System.out.println(a1 instanceof B);//TRUE
System.out.println(b1 instanceof A); //TRUE
System.out.println(c1 instanceof B); //TRUE
System.out.println(b1 instanceof C); //FALSE
System.out.println(b1 instanceof Driver);//CTE

	}
}
```

METHOD OVERRIDING

The Process of overriding sub class method implementation to the super class method implementation is known as Method overriding

Eg:



```
class Parents
{
    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:-

```
Blue color
Audi
Black color
Audi
Pink color
Nano
```

Eg : -

```java
class Parents {
    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:-

```
Rani
5000kg
Fruity
5000kg
```

Polymorphism

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

The ability to exhibit more than one form is known as POLYMORPHISM

Or

A statement executed multiple times generating different results is also called Polymorphism

There are two types of Polymorphism
1.Compile time Polymorphism
2.Run time Polymorphism

1.Complie time Polymorphism:- In compile time polymorphism the link between method call statement and method implementation happen during compilation hence, it is known as Compile time Polymorphism.
Eg:-Method Overloading

```
public class A4{
        void A1() {
                System.out.println("method 1");
        }
        void A1(int a) {
                System.out.println("method 2");
        }
        void A1(int a,double b) {
                System.out.println("method 3");
        }
        public static void main(String[] args) {
        A4 a1=new A4();
        a1.A1();
        a1.A1(10);
        a1.A1(10,20.5);
        }
}
```

OUTPUT

```
method 1
method 2
method 3
```

In the above example based on the arguments passed compiler links the method call statement with the method implimentation

We can achieve Run time Polymorphism

1.Is-A Relationship(Inheritance)
2.Upcasting
3.Method overriding

Eg:-
```
public class Card {
public void MakePayment() {
	System.out.println("making paymnet");
}
}

public class CreditCard extends Card{
public void MakePayment() {
	System.out.println("making payment through credit card");
}
}

public class DebitCard extends Card{
public void MakePayment() {
	System.out.println("making payment through debit card");
}
}

public class Swipe {
public void SwippingMachine(Card c) {
	c.MakePayment();
}
}

public class Driver {
public static void main(String[] args) {
Card c=new Card();
Swipe s=new Swipe();
s.SwippingMachine(c);
DebitCard d=new DebitCard();
s.SwippingMachine(d);
CreditCard c1=new CreditCard();
s.SwippingMachine(c1);
}
}
```

OUTPUT

```
making payment
making payment through debit card
making payment through credit card
```

METHOD SHADOWING

We can have same static method in same static method in super class and sub class it is known as method shadowing.

```
class A{
public static void A1() {
        System.out.println("method 1");
}
}
public class B extends A{
        public static void A1() {
                System.out.println("method 2");
        }
}
public class Driver {
public static void main(String[] args) {
 A a1=new A();
 a1.A1();
 B b1=new B();
 b1.A1();
 A a2=b1;
 a2.A1();
}
}
```

OUTPUT

```
method 1
method 2
method 1
```

VARIABLE SHADOWING

We can have same static or non-static variable in sub class and super class it is known as variable shadowing.

EG:-

```
public class A {
static int a=10;
int b=20;
}

public class B extends A{
static int a=30;
int b=40;
}

public class Driver {
	public static void main(String[] args) {
		A a1=new A();
		System.out.println(a1.a);
		System.out.println(a1.b);
		B b1=new B();
		System.out.println(b1.a);
		System.out.println(b1.b);
		A a2=b1;
		System.out.println(a1.a);
		System.out.println(a1.b);
}
}
```

OUTPUT

```
10
20
30
40
10
20
```

ABSTRACTION
Abstraction is the Process of the hiding implementation details

Abstact method

A method only with header and no body is known as abstract method.
Note:-abstract method header must be prefixed with abstract keyword.
abstract method header must end with a semicolon.

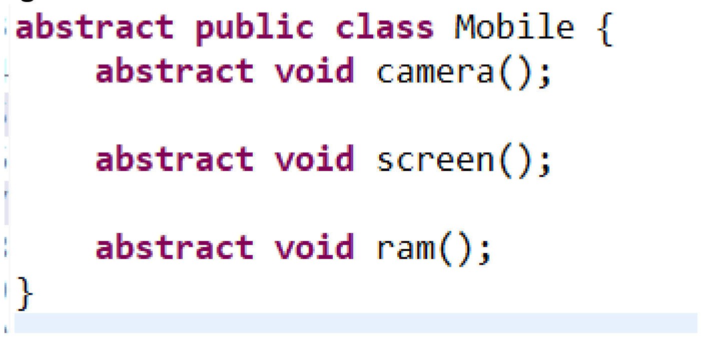
syntax-->abstract accessspecifier returntype methodname(formal arguments);

Eg:- abstract void test();
Eg 2:

```
abstract public class Mobile {
        abstract void camera();//---->This is a Abstract method because the method is
prefixing with the word called as abstract.
        abstract void screen();
        abstract void ram();
}
```
--->If we want to use a abstract method in a class then the class should also be prefixed with keyword called as abstact.

Abstract class:- A class prefixed with abstract keyword is known as abstract class.

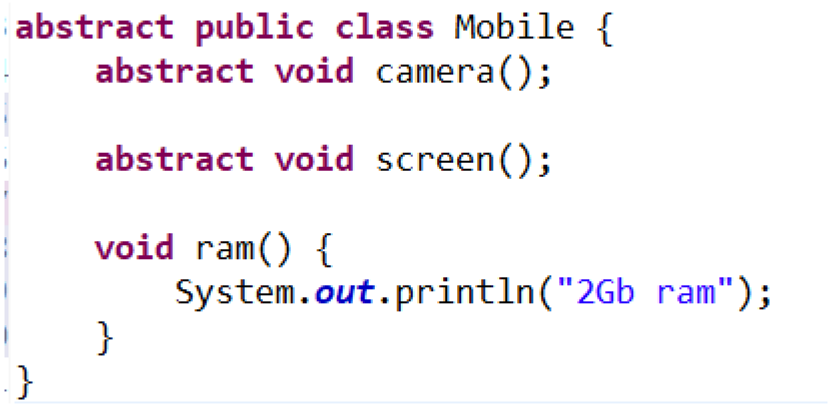
Note1:-If a class has atleast one abstract method inside it,then it is mandatory to declare that class as abstract class.

Eg:-

```
abstract public class Mobile {
    abstract void camera();

    abstract void screen();

    abstract void ram();
}
```

Note2:-An abstract class can have both abstract and concrete methods inside it.

concrete method-A method with body is known as concrete method.

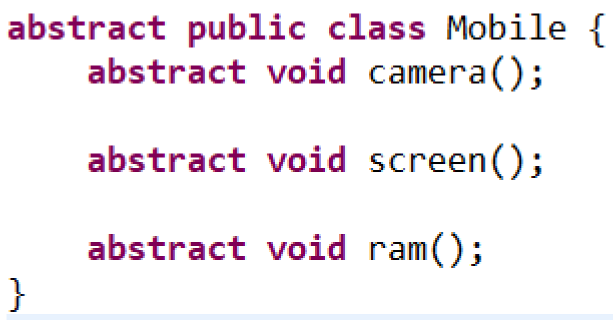
Eg:-



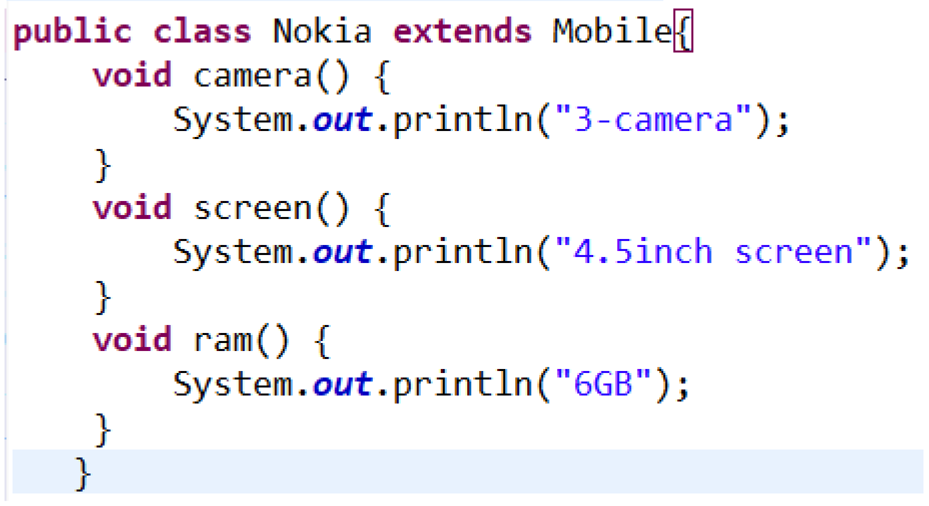
```java
abstract public class Mobile {
    abstract void camera();

    abstract void screen();

    void ram() {
        System.out.println("2Gb ram");
    }
}
```

Note3:-An abstract class can have only concrete methods inside it.

Eg:-



```java
abstract public class Mobile {
    abstract void camera();

    abstract void screen();

    abstract void ram();
}
```



```java
public class Nokia extends Mobile{
    void camera() {
        System.out.println("3-camera");
    }
    void screen() {
        System.out.println("4.5inch screen");
    }
    void ram() {
        System.out.println("6GB");
    }
  }
```

-

When exactly we should make a class abstract?

Whenever a class is having atleast one defined or inherited abstract method then it is mandatory to make that class abstract.

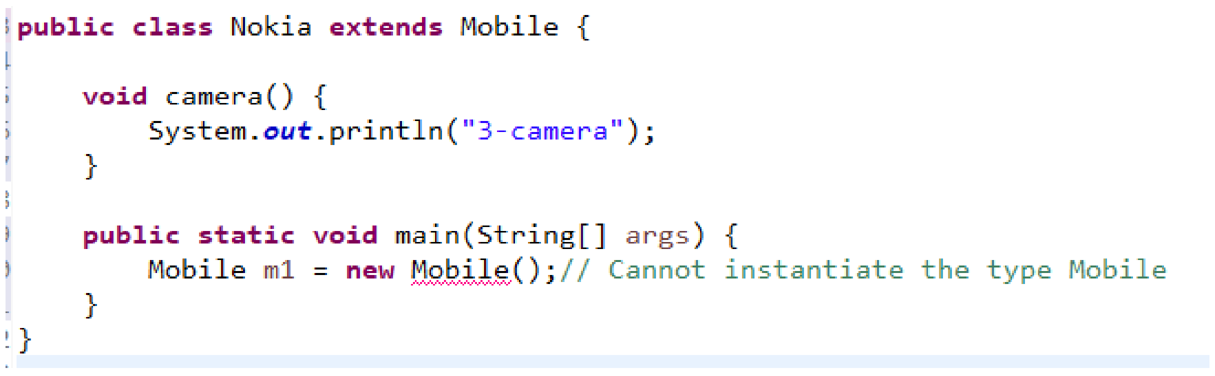
Eg:-

```java
abstract public class Mobile {
    abstract void camera();

    // This class is having only one abstract method , so it is mandatory to make
    // the class as abstract class
    void brand() {
        System.out.println("Nokia");
    }

    void screen() {
        System.out.println("6.7inch");
    }

    void ram() {
        System.out.println("4gb");
    }
}
```

Note:-An abstract class can have constrcuctor but we cannot create an object for it.

Eg:-

```java
abstract public class Mobile {
    Mobile() {
        System.out.println("A constructor can be created,but cannot be called!!!!");
    }

    abstract void camera();

    // This class is having only one abstract method , so it is mandatory to make
    // the class as abstract class
    void brand() {
        System.out.println("Nokia");
    }

    void screen() {
        System.out.println("6.7inch");
    }

    void ram() {
        System.out.println("4gb");
    }
}
```

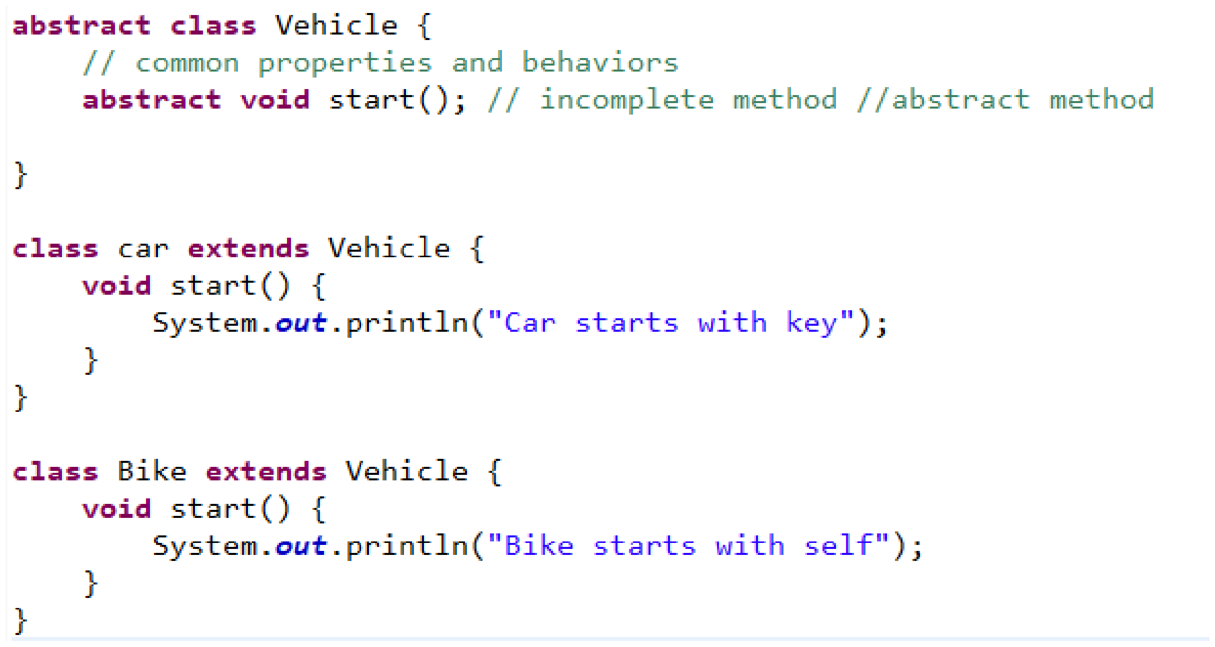


```java
public class Nokia extends Mobile {

    void camera() {
        System.out.println("3-camera");
    }

    public static void main(String[] args) {
        Mobile m1 = new Mobile();// Cannot instantiate the type Mobile
    }
}
```

Concrete class

A class is said to be concrete class when all the methods of that class i.e both defined as well as inherited are concrete.

Eg:-



```java
abstract class Vehicle {
    // common properties and behaviors
    abstract void start(); // incomplete method //abstract method

}

class car extends Vehicle {
    void start() {
        System.out.println("Car starts with key");
    }
}

class Bike extends Vehicle {
    void start() {
        System.out.println("Bike starts with self");
    }
}
```

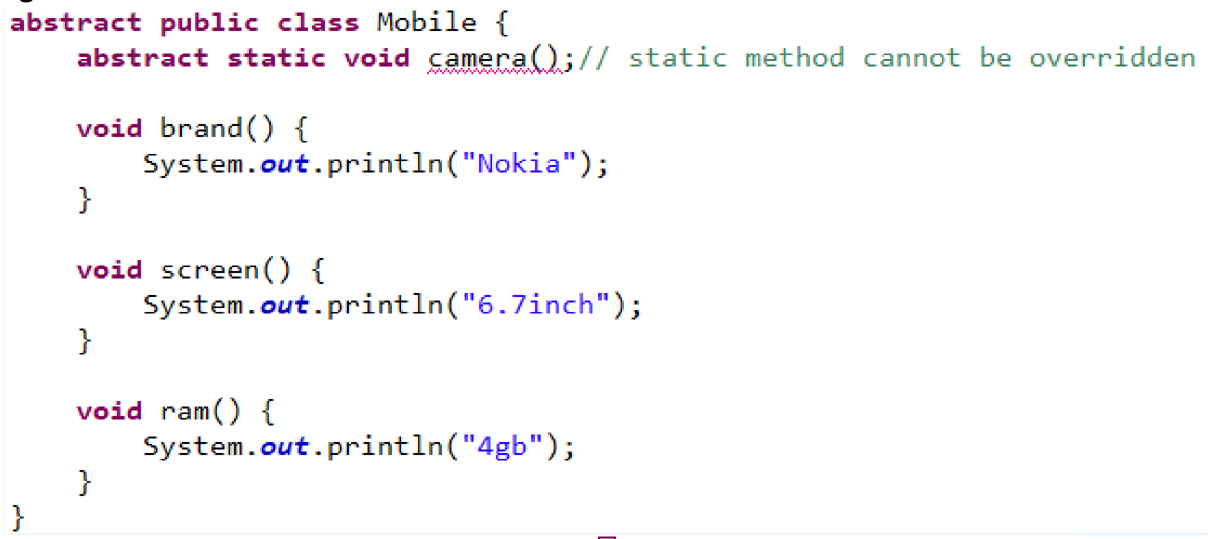
Note:-

If we want to use the non static members of an abstract class, we must create a concrete subclass for it.In the concrete subclass we must provide implementation to the abstract methods of superclass. And then by creating object of subclass we can use the non static members of superclass.

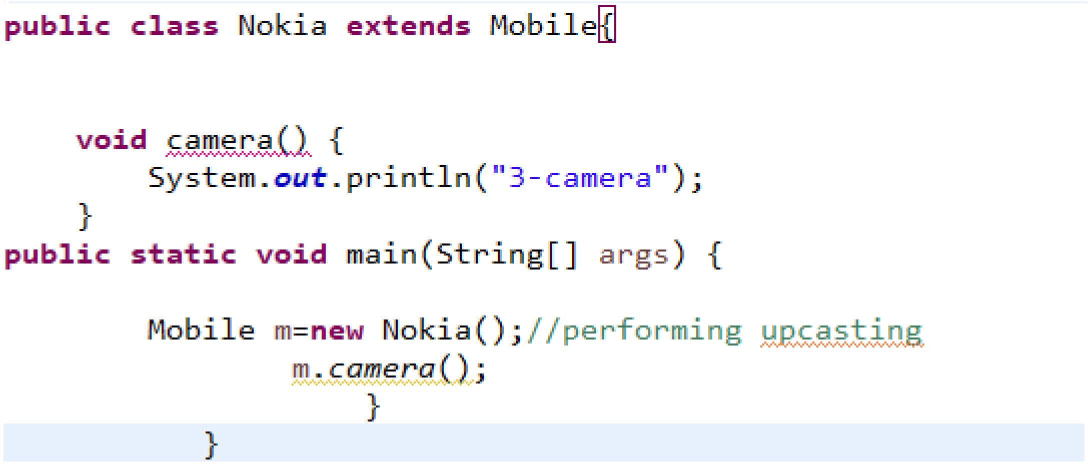
Can we make a static method abstract?

As static methods cannot be overridden, We cannot make them abstract.

Eg:-



```
abstract public class Mobile {
    abstract static void camera();// static method cannot be overridden

    void brand() {
        System.out.println("Nokia");
    }

    void screen() {
        System.out.println("6.7inch");
    }

    void ram() {
        System.out.println("4gb");
    }
}
```



```
public class Nokia extends Mobile{


    void camera() {
        System.out.println("3-camera");
    }
public static void main(String[] args) {

        Mobile m=new Nokia();//performing upcasting
                m.camera();
                    }
            }
```

Interface

Interface is a component of java which acts as blueprint of a class.

Interface can be created using the keyword interface,

Eg:- interface interfacename

```
  {

  }
```

Interface can have following members,

1)abstract non static methods

2)static concrete method (From JDK1.8)

3)final static variable. (Fom JDK1.8)

Note1:- All the non-static methods inside interface are by default considered as abstract and public. Therefore it is not mandatory to use public and abstract keywords for non static methods inside the interface.

Eg:- Before Compilation

```
------------------
interface I{
   void a2();
}
```

After Compilation

```
-----------------
interface I{
abstract public void a2();
}
```

Note2:- All the static concrete methods of an interface are by default considered as public static concrete methods.

Before Compilation

```
    -----------------------------
                 interface I{
                   static void a2()
                          {
                                 system.out.println("Method1");

                          }
                 }
```

After Compilation

```
-----------------
Interface I{
public static void a2(){
system.out.println("Method1");
}
}
```

Note3:- All the variables inside the interface are by default considered as public static and final.

Eg:- **Before Compilation**

```
---------------------------
interface I{
        int a=10;
}
```

After Compilation

```
        interface I{
                public static final int a=10;
                }
```

Therefore interface can have following members inside it,
1)public abstract non static methods
2)public static concrete methods
3)public static final variables.

Note:-Interface cannot have constructor, And we cannot create an object of interface.

Inheritance between interfaces
--

We can achieve inheritance between interfaces using extends keyword. And interface supports all types of inheritance.

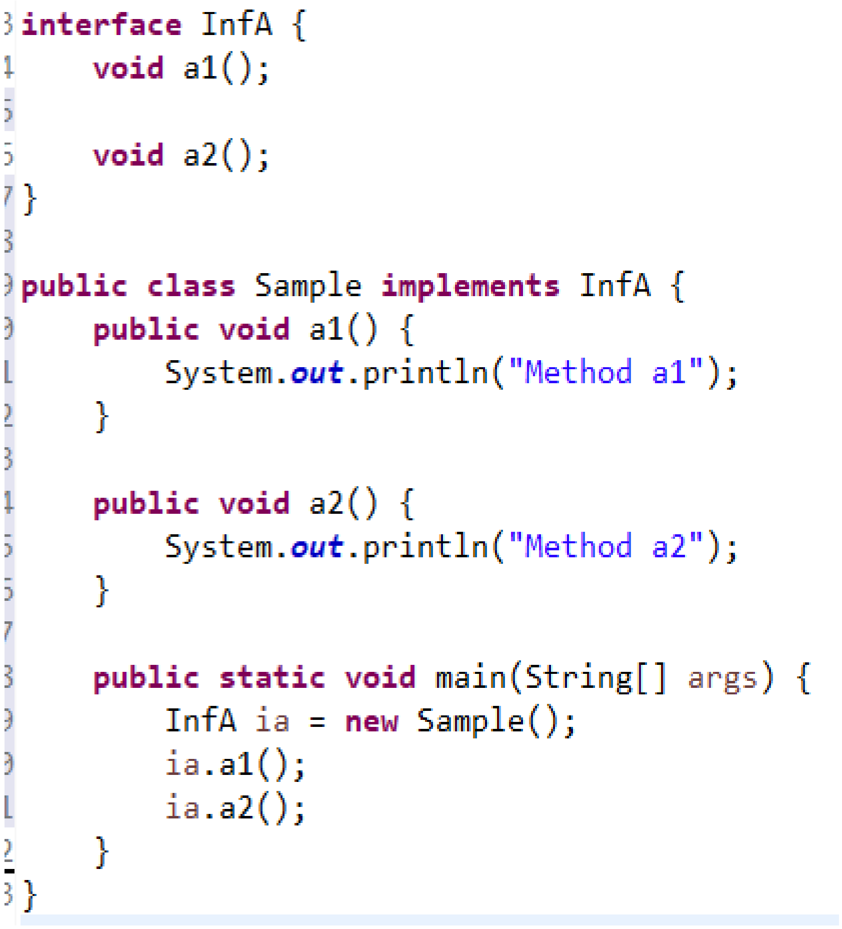
Eg:-
```
interface I1{
   }

   interface I2 extends I1 {

     }
```

inheritance between class and interface

An interface cannot inherit a class, But a class can inherit an interface.

Inheritance between class and interface is achieved with the help of implements keyword.

Eg:-



```java
interface InfA {
    void a1();

    void a2();
}

public class Sample implements InfA {
    public void a1() {
        System.out.println("Method a1");
    }

    public void a2() {
        System.out.println("Method a2");
    }

    public static void main(String[] args) {
        InfA ia = new Sample();
        ia.a1();
        ia.a2();
    }
}
```

<u>OutPut</u>

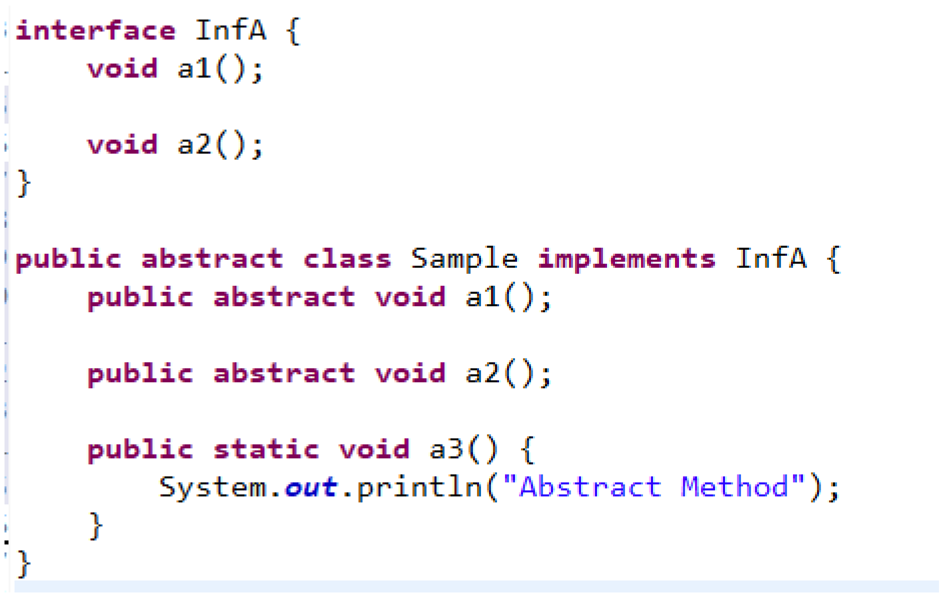
```
Method a1
Method a
```

inheritance between abstract class and interface

When we perform inheritance between interface and absract class,we cannot create object for any of the class.

Inheriatnce between interface and abstract class can be achieved by using implements keyword.
Abstract class will get all the properties of interface because of Is-a-Relationship.

Eg:-

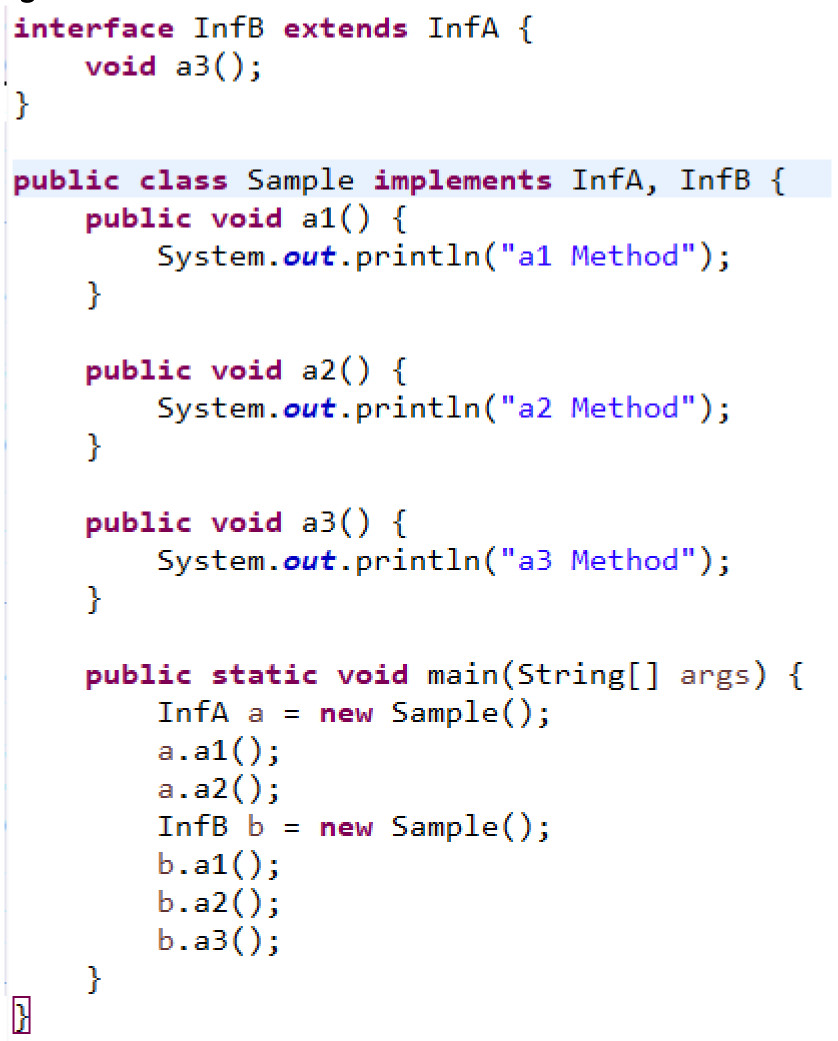


```
interface InfA {
    void a1();

    void a2();
}

public abstract class Sample implements InfA {
    public abstract void a1();

    public abstract void a2();

    public static void a3() {
        System.out.println("Abstract Method");
    }
}
```

Achieving Multiple inheritance using interface

We can achieve multiple Inheritance in interface because interface does not contains constructor.

Eg:-



```
interface InfB extends InfA {
    void a3();
}

public class Sample implements InfA, InfB {
    public void a1() {
        System.out.println("a1 Method");
    }

    public void a2() {
        System.out.println("a2 Method");
    }

    public void a3() {
        System.out.println("a3 Method");
    }

    public static void main(String[] args) {
        InfA a = new Sample();
        a.a1();
        a.a2();
        InfB b = new Sample();
        b.a1();
        b.a2();
        b.a3();
    }
}
```

OutPut

```
a1 Method
a2 Method
a1 Method
a2 Method
a3 Method
```

Encapsulation

It is the process of binding properties and behaviours of the object together.

OR
Hiding the internal states and requiring all the interactions to be performed only through non static method is known as Encapsulation.

internal state-->value of variable
interaction-->reading and modifying the value

Data Hiding

The process of hiding the properties from the outside world is known as data hiding.

We can achieve datahiding with the help of private access specifier.

Private
----
private is a keyword in java which acts as access specifier, when a member of a class is made private, it can be used only inside the same class, we cannot use it outside the class.

Note:- In order to use the private variable of a class outside the class, we should make use of
getter and setter methods.

getter method is used for reading the value of a variable.
Setter method is used for modifying the value of a variable.

If the private variable should be both readable as well modifiable outside the class we should design both getter and setter methods.

if the private variable should be only readable outside the class then we should design only getter method.

if the private variable should be only modifiable outside the class then we should design only setter method.

Syntax of getter method

```
public datatype_of_the_attribute getName_of_Attribute()
{
  return attribute;
}
```

Eg:-

```
 public String getName()
{
 return name;
}
```

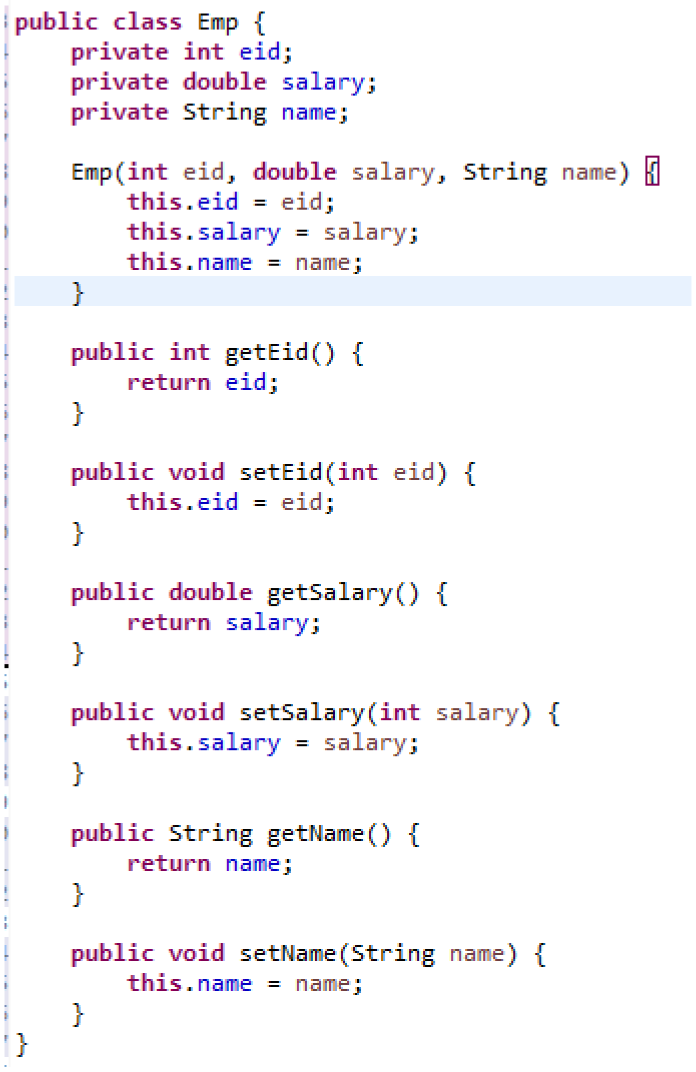
Syntax of setter method

```
public void setName_of_Attribute(datatype_of_the_attribute attribute)
 {
   this.attribute=attribute;
 }
```

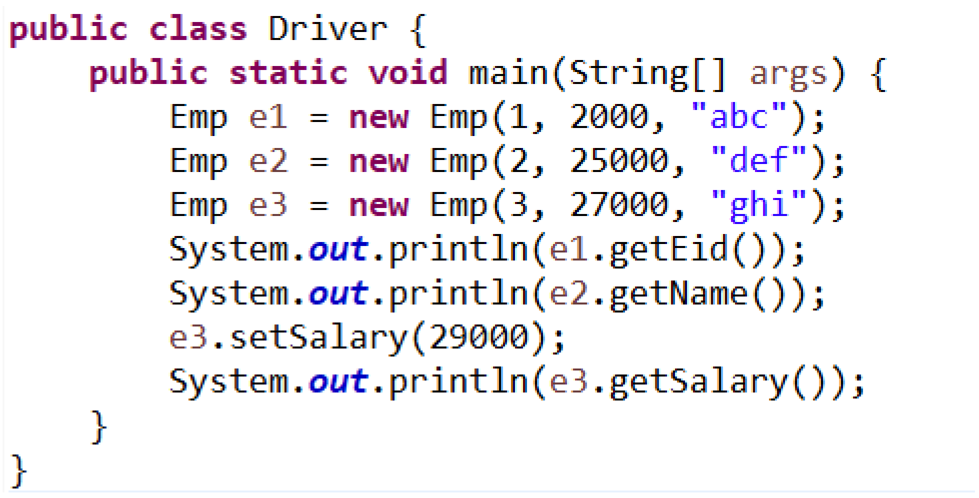
Eg:-

```
 public void setName(String name)
 {
  this.name=name;
 }
```

eg:-

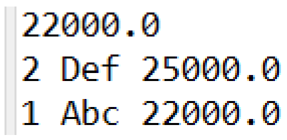


```
public class Emp {
    private int eid;
    private double salary;
    private String name;

    Emp(int eid, double salary, String name) {
        this.eid = eid;
        this.salary = salary;
        this.name = name;
    }

    public int getEid() {
        return eid;
    }

    public void setEid(int eid) {
        this.eid = eid;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(int salary) {
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```



```java
public class Driver {
    public static void main(String[] args) {
        Emp e1 = new Emp(1, 2000, "abc");
        Emp e2 = new Emp(2, 25000, "def");
        Emp e3 = new Emp(3, 27000, "ghi");
        System.out.println(e1.getEid());
        System.out.println(e2.getName());
        e3.setSalary(29000);
        System.out.println(e3.getSalary());
    }
}
```

OUTPUT



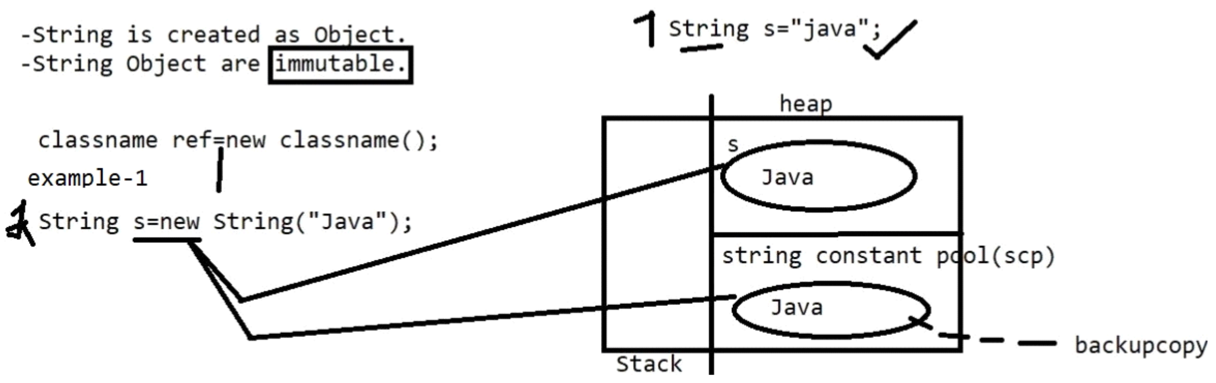
```
22000.0
2 Def 25000.0
1 Abc 22000.0
```

STRING

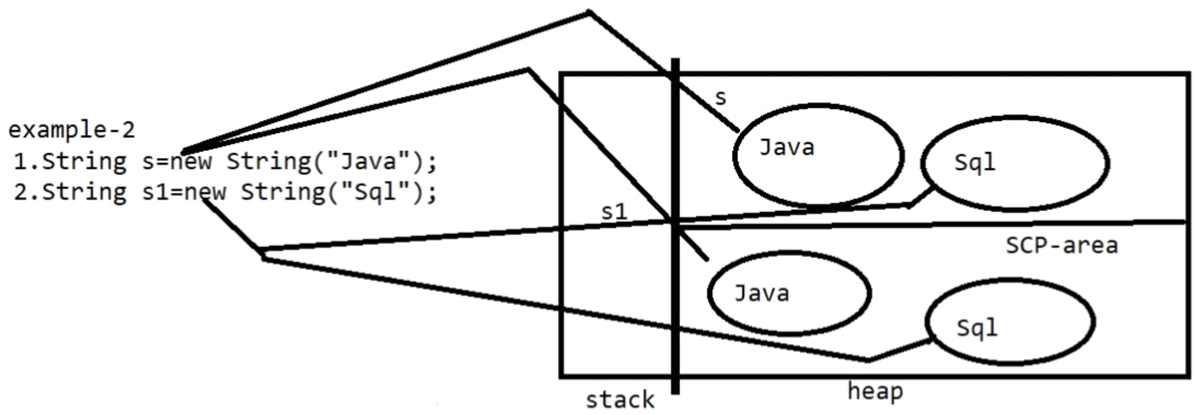
• String is predefine class which is present in java.lang package
• In java string is an -
1. object
2. datatype
3. class
4. group of characters

objects :

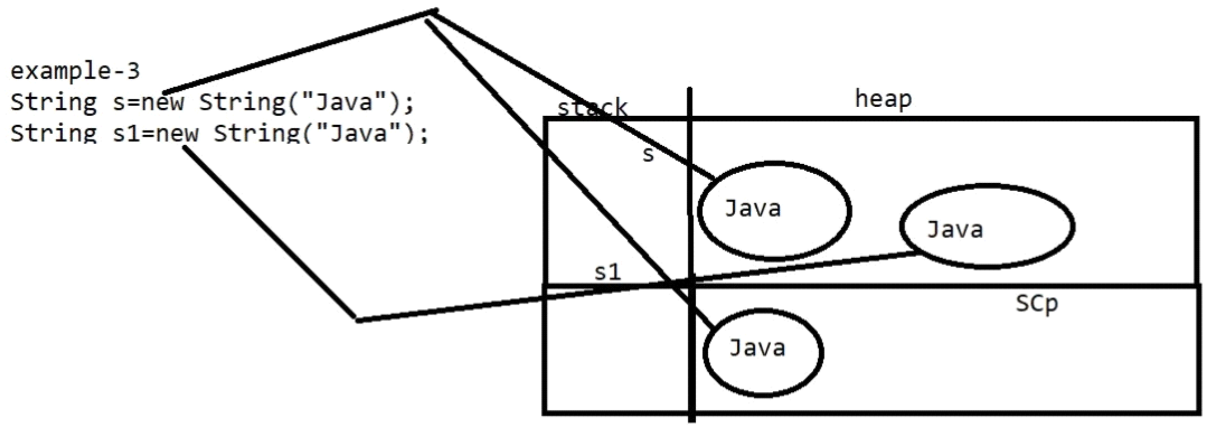
String objects can be created in two ways-
1. using new keyword - two objects will get created
1. one is under heap area - reference variable assigned to it.
2. another is under string constant pool(SCP) - it is small part of heap area no reference assigned to it, it is only for backup.



```
-String is created as Object.
-String Object are immutable.

classname ref=new classname();
example-1
1 String s=new String("Java");

1 String s="java";
```



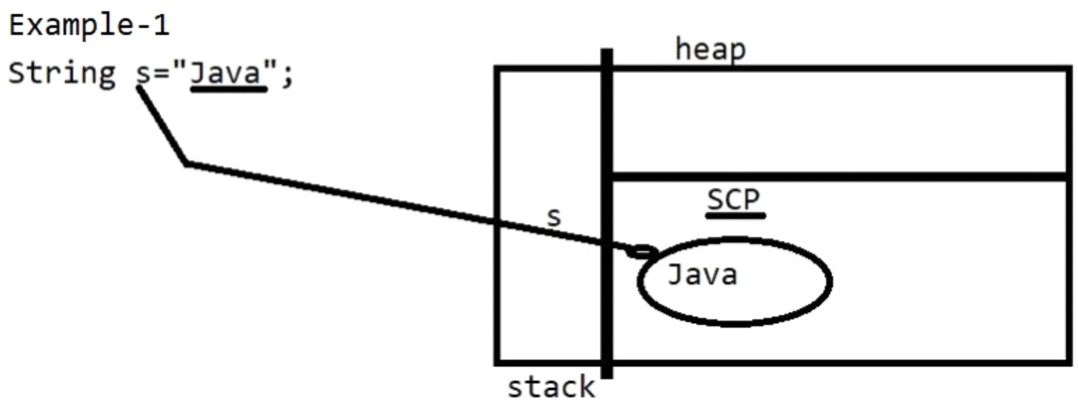
```
example-2
1.String s=new String("Java");
2.String s1=new String("Sql");
```



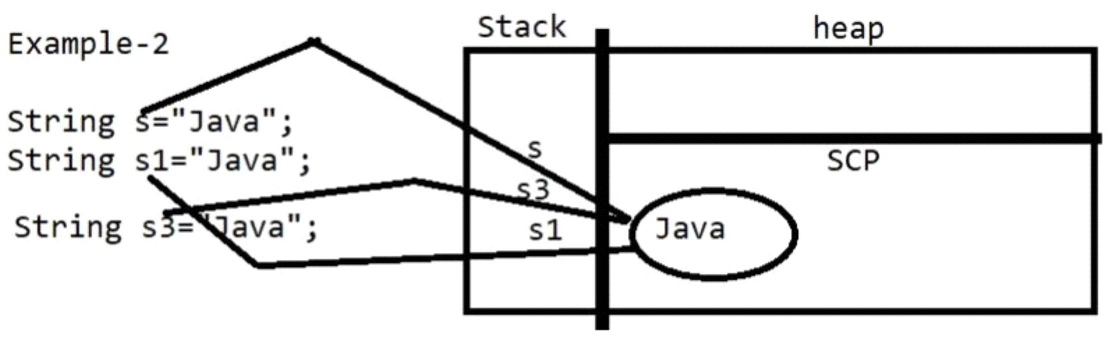
```
example-3
String s=new String("Java");
String s1=new String("Java");
```

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.



```
Example-1
String s="Java";
```

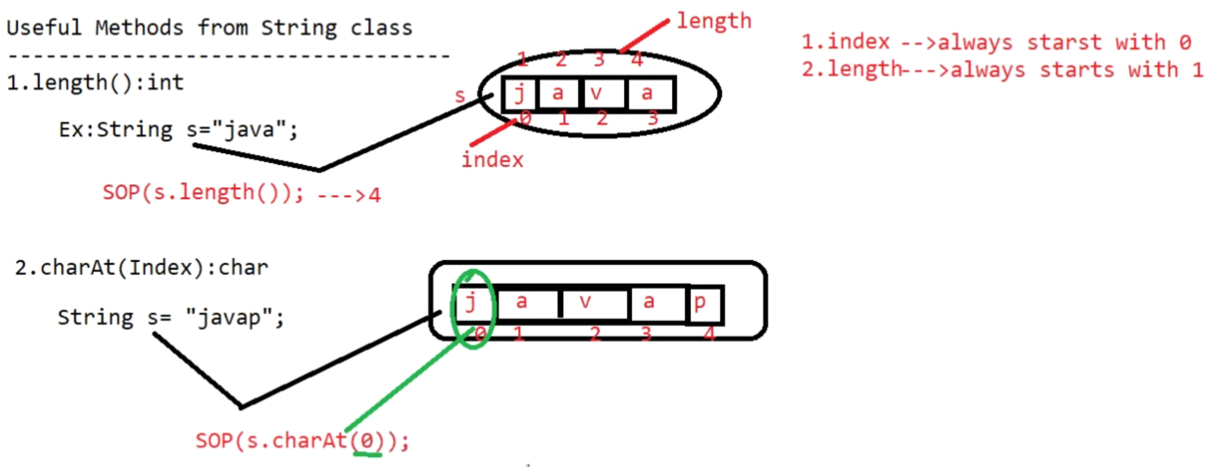


```
Example-2

String s="Java";
String s1="Java";
String s3="Java";
```

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.

methods of String class



```
Useful Methods from String class
-----------------------------------
1.length():int

    Ex:String s="java";

        SOP(s.length()); --->4

 2.charAt(Index):char

    String s= "javap";

            SOP(s.charAt(0));
```

```
1.index -->always starst with 0
2.length--->always starts with 1
```

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

EG:-1

```
public class FirstP {
public static void main(String[] args) {
String s="javap";
System.out.println(s.charAt(3));
}
}
```

Output: a

EG 2:-

```
public class FirstP {
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

Q)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 considering 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 palindrome
```

```
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 palindrome
```

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

Eg:-

```
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
```

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

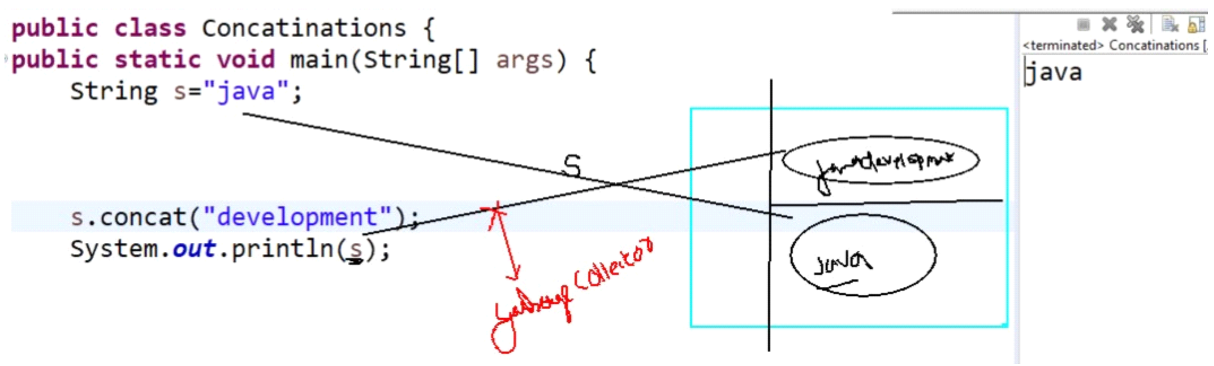
14.object.concat(String)

• 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());
}
}
```

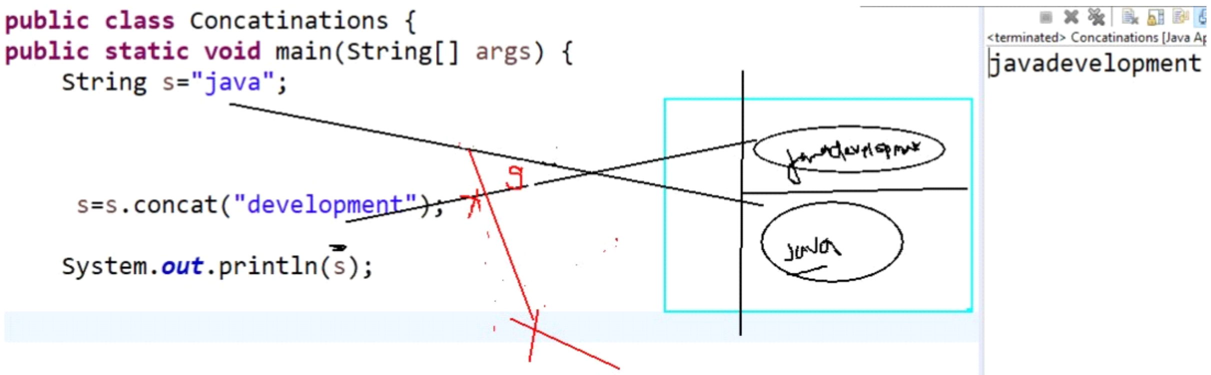
Output:

```
JAVA DEVELOPMENT
java development
false
true
true
true
```



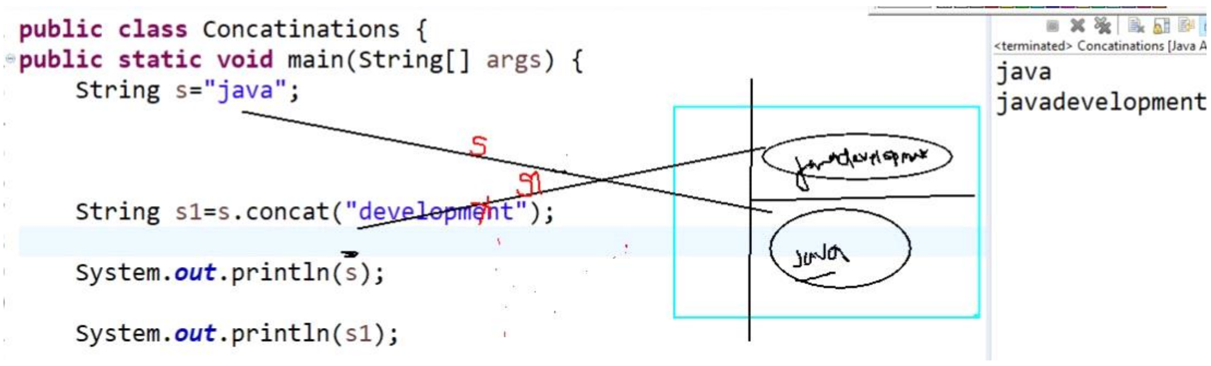
```
public class Concatinations {
public static void main(String[] args) {
    String s="java";

    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);
```

```
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);
```

```
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]

```
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:

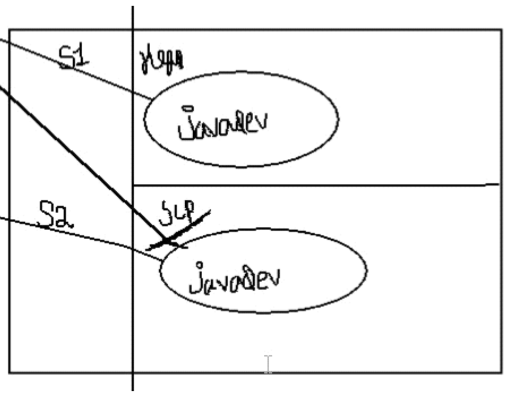
```
java davalopmant
core java development
javadevelopment
jv eeloer
AA DVP
javaDEveloper
j123vDvlpr
```

***differences between equals() and == with respect to Strings

----

• equals() ---> compares two strings based on string data
• == ---> compares two strings based on reference

```
Ex: String s1=new String("javadev");
    String s2="javadev";
```



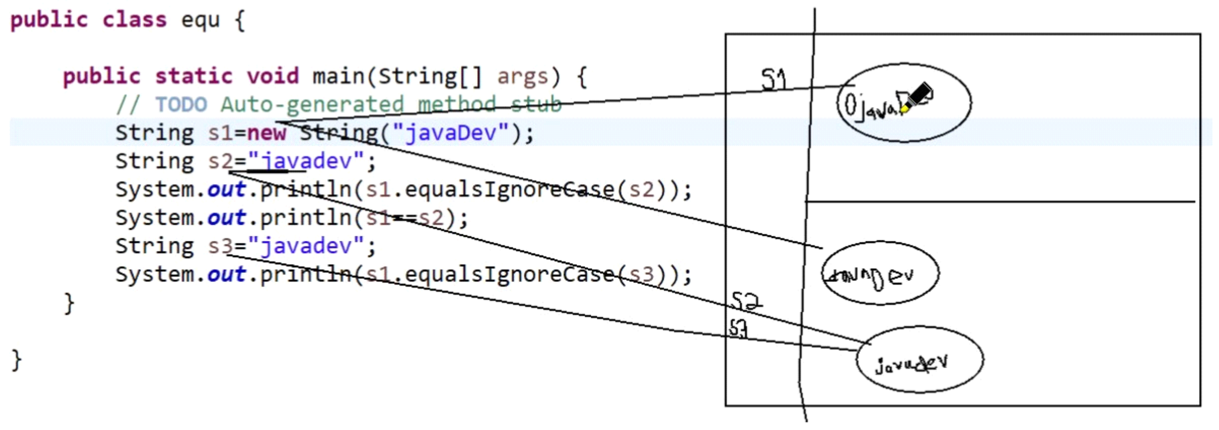
```
System.out.println(s1.equals(s2));//true
System.out.println(s1==s2);//false
```

Eg:-

```
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
```

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
```

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()

EXCEPTION HANDLING

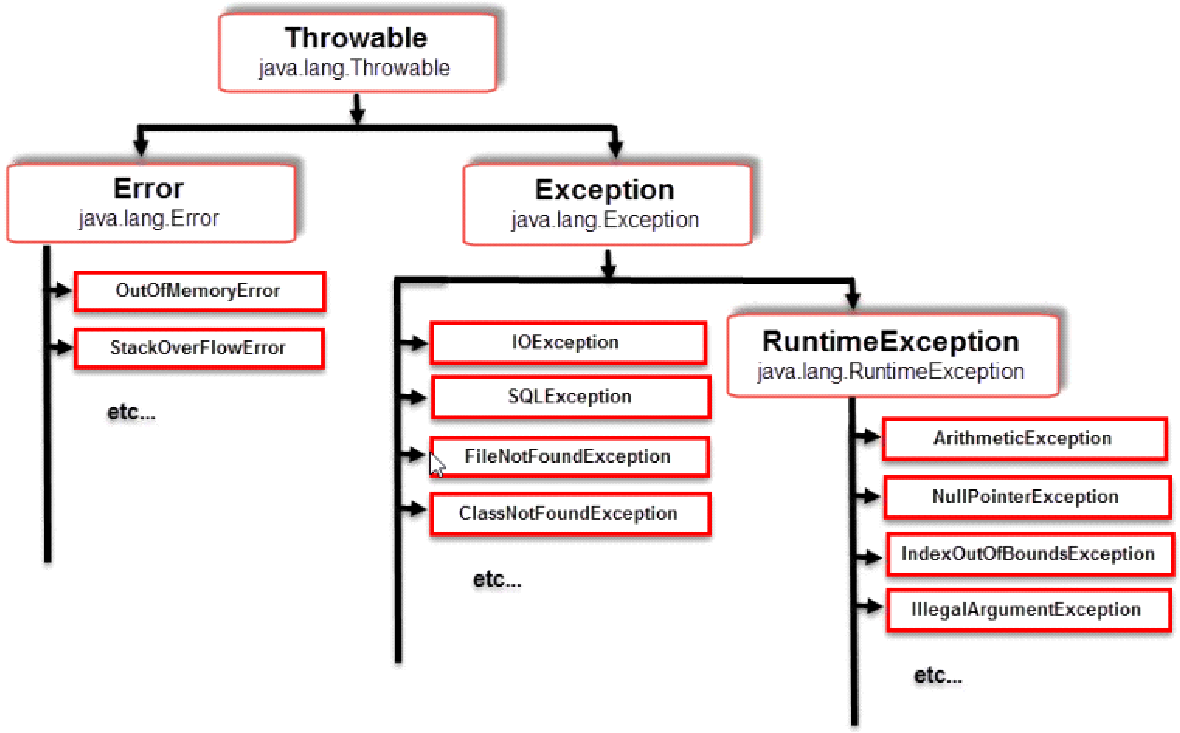
• Exception
• Hierarchy of Exception
• Types of Exception(Different classes in Excepton)
• Keywords of Exception
try, catch, throw, throws and finally
• difference between throw and throws
• **difference between final, finally, finalise
• ***user define exception or customised exception

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

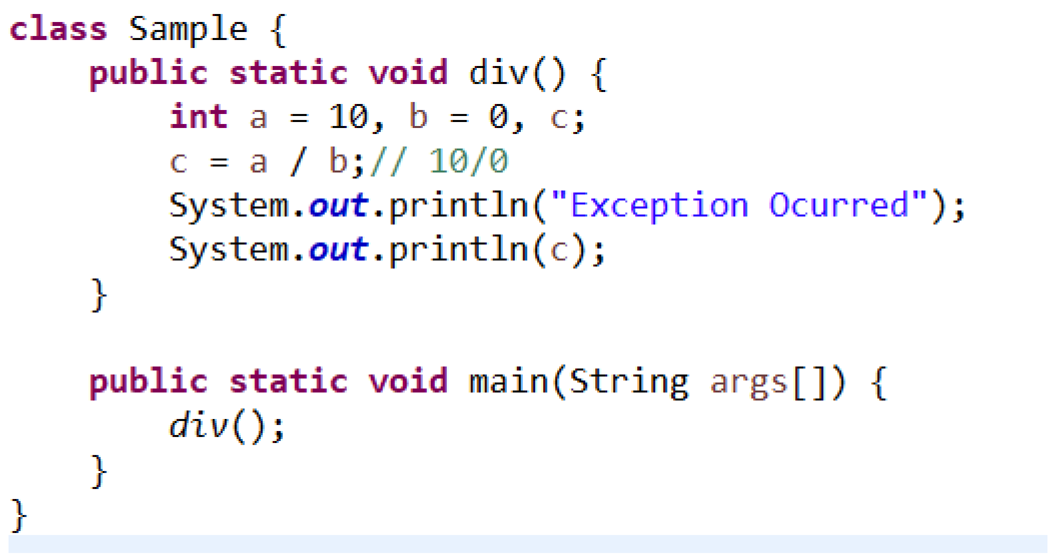
Error :

• 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

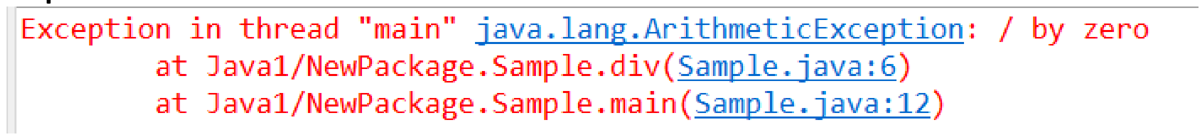
Error	**Exception**
1. An error is caused due to lack of system resources.	1. An exception is caused because of some problem in code.
2. An error is irrecoverable i.e, an error is a critical condition cannot be handled by code of program.	2. An Exception is recoverable i.e, we can have some alternate 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.lang.Exception package

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();
    }
}
```

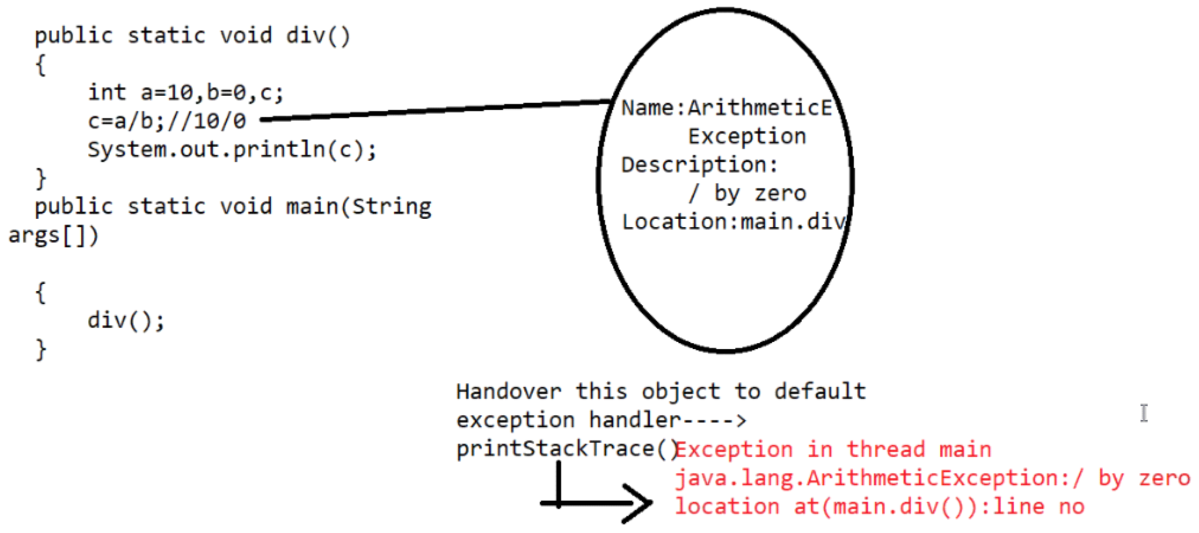
Output:



```
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at Java1/NewPackage.Sample.div(Sample.java:6)
	at Java1/NewPackage.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)

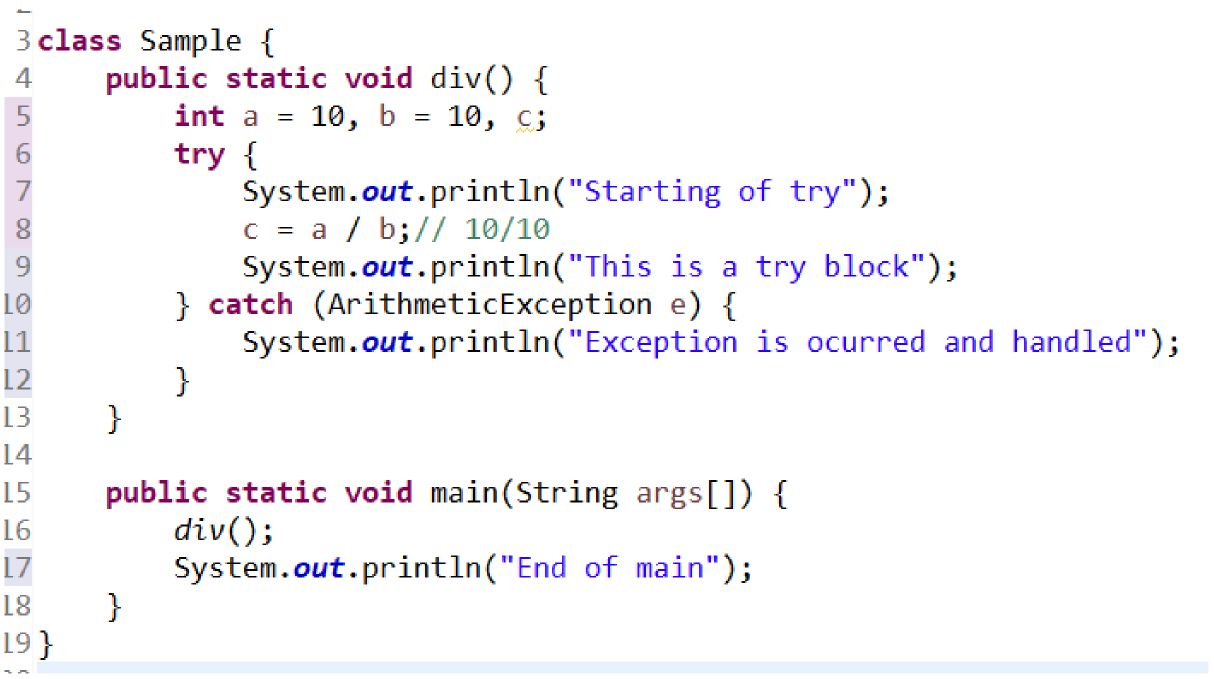
• Once exception ocurred remaining part of a program will not be executed and ending up with abnormal termination.
• So, if we want the above program to be executed and have normal termination then, we have to handle the Exception.
• Exception can be handled by using try and catch block.



```
  public static void div()
  {
      int a=10,b=0,c;
      c=a/b;//10/0
      System.out.println(c);
  }
  public static void main(String
args[])

  {
      div();
  }
```

```
Handover this object to default
exception handler---->
printStackTrace()Exception in thread main
                 java.lang.ArithmeticException:/ by zero
                 location at(main.div()):line no
```

Ex 1: Exception didn't occurred, so except catch block all normal statements will be executed

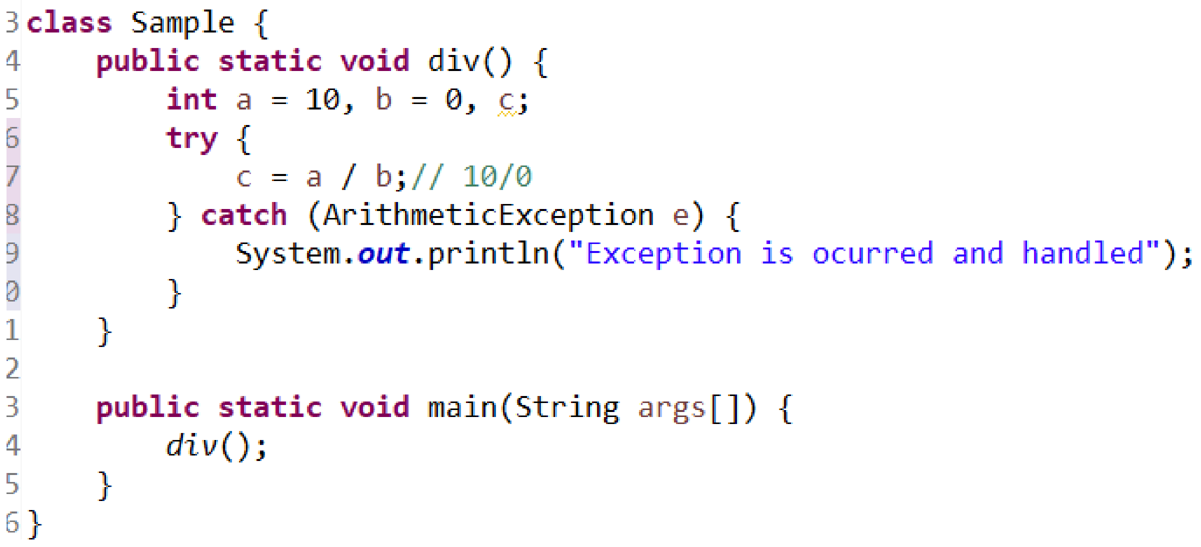


```java
class Sample {
    public static void div() {
        int a = 10, b = 10, c;
        try {
            System.out.println("Starting of try");
            c = a / b;// 10/10
            System.out.println("This is a try block");
        } catch (ArithmeticException e) {
            System.out.println("Exception is ocurred and handled");
        }
    }

    public static void main(String args[]) {
        div();
        System.out.println("End of main");
    }
}
```

Output:

```
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

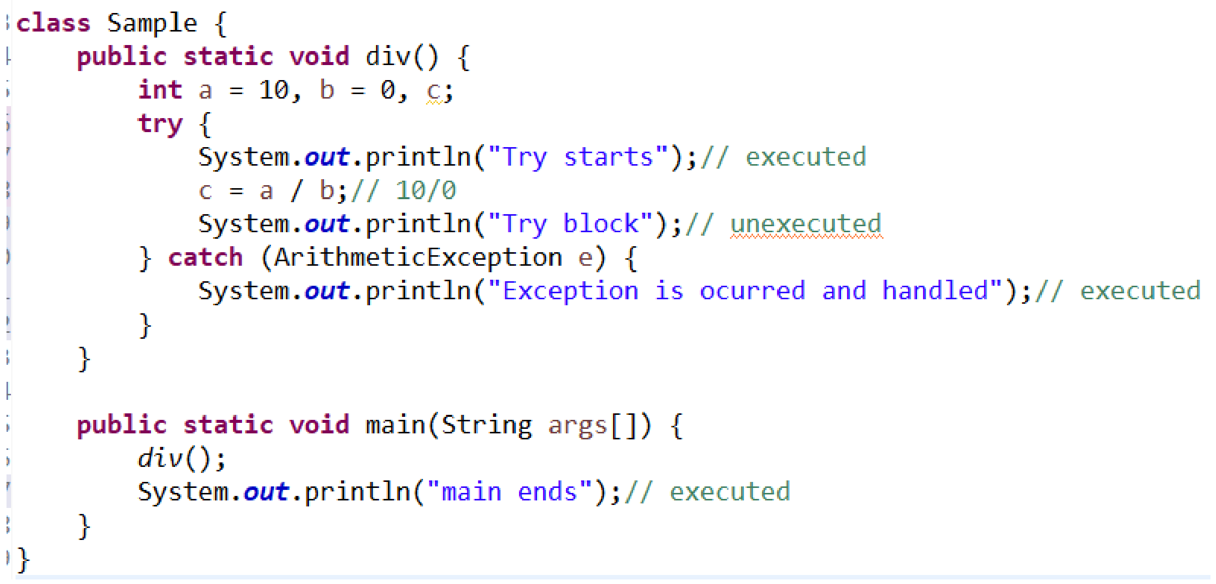


```
class Sample {
    public static void div() {
        int a = 10, b = 0, c;
        try {
            c = a / b;// 10/0
        } catch (ArithmeticException e) {
            System.out.println("Exception is ocurred and handled");
        }
    }

    public static void main(String args[]) {
        div();
    }
}
```

Output:

```
Exception is ocurred and handled
```

Ex 3: Exception occurred in try block and JVM have executed corresponding 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:

```
Try starts
Exception is ocurred and handled
main ends
```

```
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()

//Ex

```
class Sample {
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 toJVM
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.

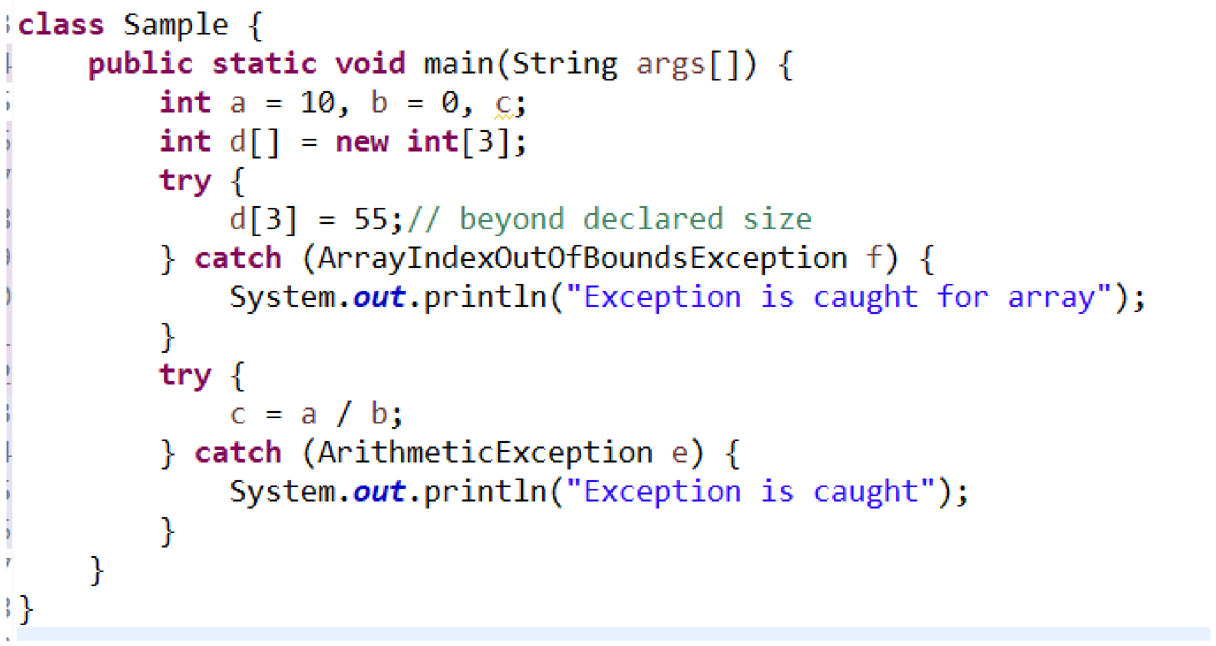
```
class Sample {
    public static void main(String args[]) {
        int a = 10, b = 0, c;
        int d[] = new int[3];
        try {
            d[3] = 55;// beyond declared size so an exception gets created
            c = a / b;// unexecuted
        } catch (ArithmeticException e) {
            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 -

Eg:-



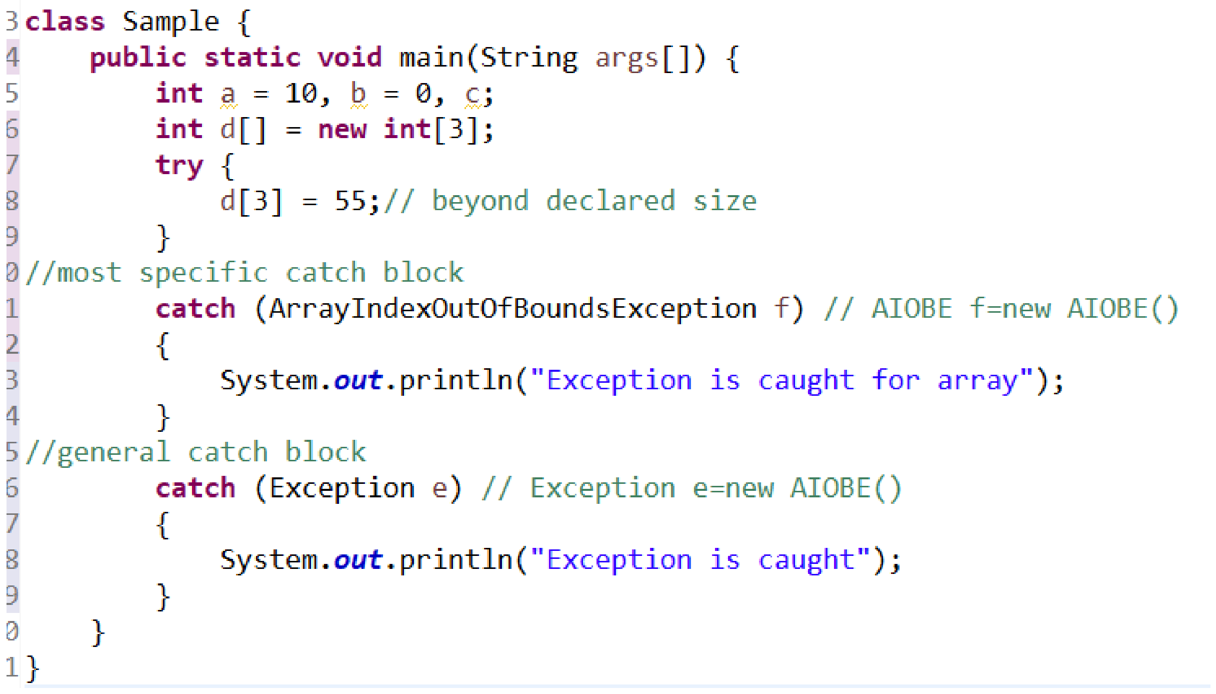
```java
class Sample {
    public static void main(String args[]) {
        int a = 10, b = 0, c;
        int d[] = new int[3];
        try {
            d[3] = 55;// beyond declared size
        } catch (ArrayIndexOutOfBoundsException f) {
            System.out.println("Exception is caught for array");
        }
        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.



```java
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");
        }
    }
}
```

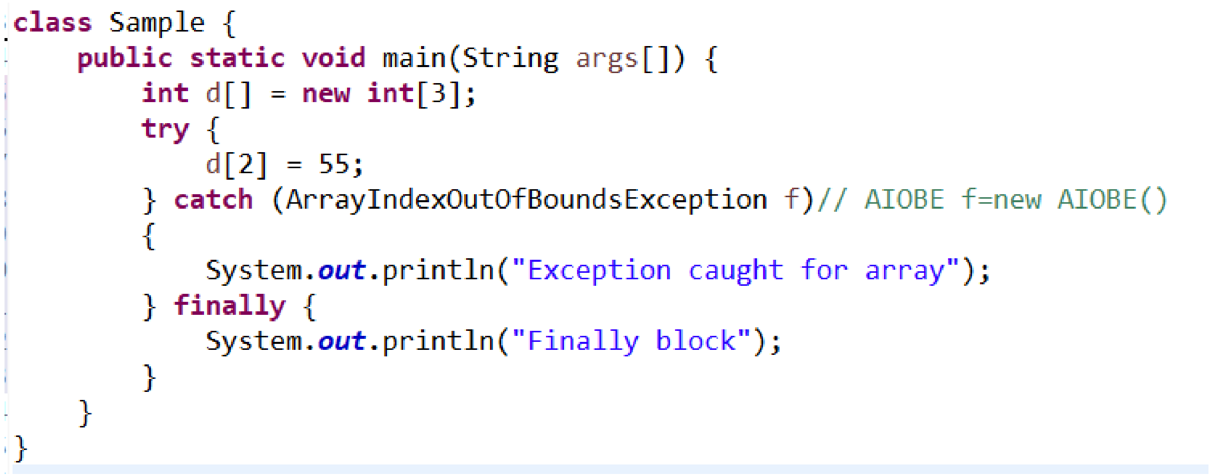
Output:

```
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
```

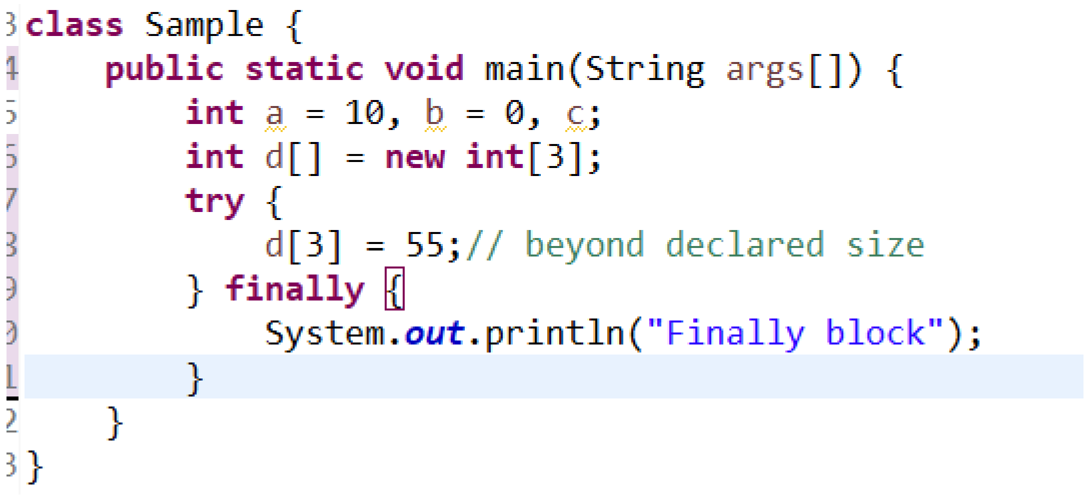
2.exception occurred and handled

```
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{}

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.

EG:-

```
class Sample {
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------
```

EG 2:-

```
class Demo
{
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.

Eg:-

```
class Demo {
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

Eg: -

```
class Demo
{
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
```

Eg:-

```
class Demo {
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:-

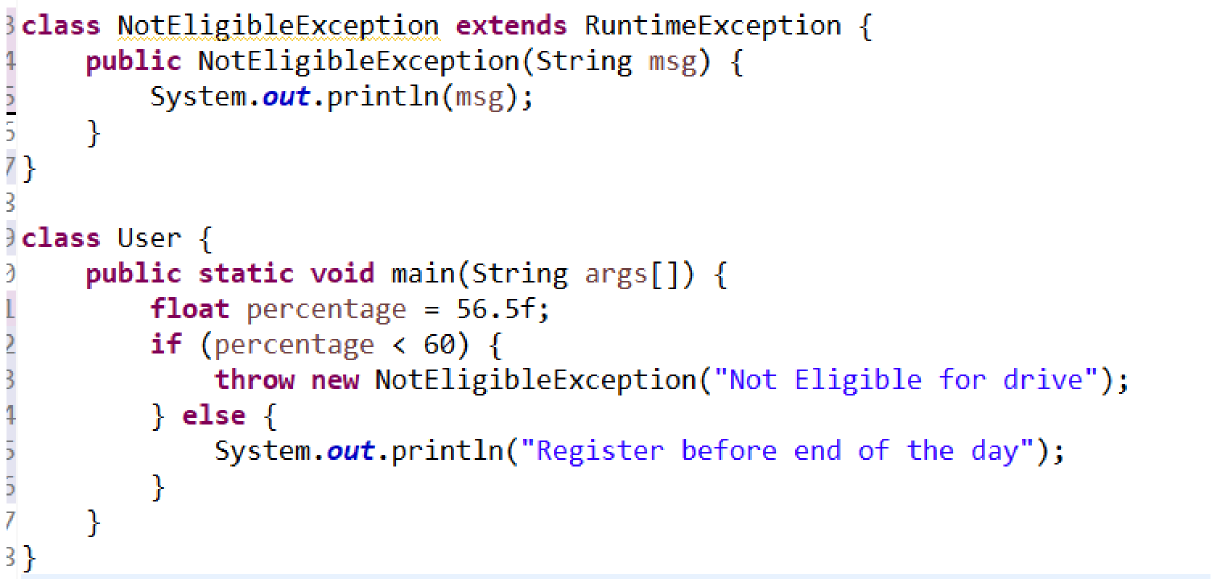
• create our Exception class and that class should be or must be extending either throwable (or) Exception (or) RuntimeException classes

Preferable to extend RuntimeException

• Define constructor whenever requires.

• Throw Exception as per our own requirement.

Eg:-



```
class NotEligibleException extends RuntimeException {
    public NotEligibleException(String msg) {
        System.out.println(msg);
    }
}

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:

```
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.

Eg:-

```java
class AgeGapException extends RuntimeException {
    public AgeGapException(String msg) {
        System.out.println(msg);
    }
}

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:

```
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

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

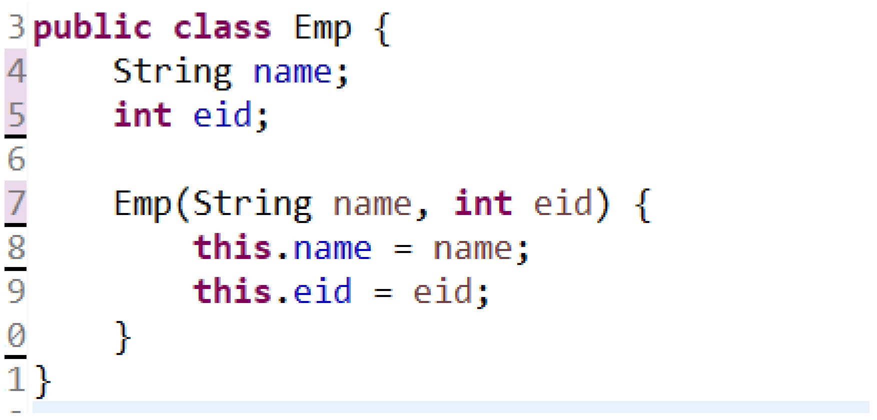
```
import java.util.Scanner;
class InsufficientBalException extends RuntimeException
{
public InsufficientBalException(String msg)
{
System.out.println(msg);
}}
public class Customer
{
public static void main(String args[]) {
System.out.println("Enter available balance in account : ");
Scanner s=new 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:

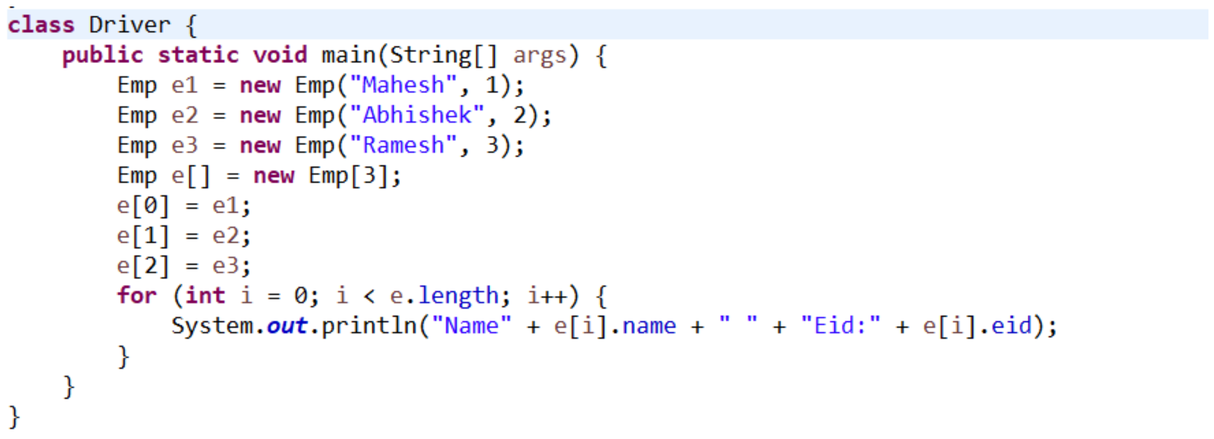
```
Enter available balance in account :
5000.87
Available balance : 5000.87
Enter the amount to withdraw :
8470.25
Exceeding the limit in the account
Exception in thread "main" InsufficientBalException
at Customer.main(Customer.java:18)
```

ARRAY OF OBJECTS

We can create array in order to store multiple objects of a class also,



```java
public class Emp {
    String name;
    int eid;

    Emp(String name, int eid) {
        this.name = name;
        this.eid = eid;
    }
}
```



```java
class Driver {
    public static void main(String[] args) {
        Emp e1 = new Emp("Mahesh", 1);
        Emp e2 = new Emp("Abhishek", 2);
        Emp e3 = new Emp("Ramesh", 3);
        Emp e[] = new Emp[3];
        e[0] = e1;
        e[1] = e2;
        e[2] = e3;
        for (int i = 0; i < e.length; i++) {
            System.out.println("Name" + e[i].name + " " + "Eid:" + e[i].eid);
        }
    }
}
```

OUTPUT:-

```
Name:Mahesh Eid:1
Name:Abhishek Eid:2
Name:Ramesh Eid:3
```

Wrapper Classes
----

byte
short
int
long
float
double
char
boolean
//primitive datatypes

Byte
Short
Integer
Long
Float
Double
Character
Boolean
//non primitive datatypes //classes //java.lang //wrapper classes

Note:-Wrapper classes are used convert primitive values into non primitive values(Object).

Why should we convert primitive values into object?

The collection framework of java can only store objects, It cannot store the primitive values.
Therefore it becomes mandatory to convert primitive values into objects.And it can be done with the help of wrapper classes.

All the wrapper classes are defined in java.lang package.
All the wrapper classes final classes.
All the wrapper classes implements comparable and serializable interfaces.

Boxing

The process of converting a primitive value into non primitive value(Object) is known as boxing.

Boxing is done with the help of a static method valueOf() present in every wrapper class.

Eg:-

Ex: Boxing

```
package collection;

public class Warpper_1 {
public static void main(String[] args) {
		int a=10;//primitive
		Integer b=Integer.valueOf(a);//no-primitive
		System.out.println(b);
		double b1=20.5;//primitive
		Double c=Double.valueOf(b1);
		System.out.println(c);//non-primitive
}
}
```

Note:-from JDK 1.5 onwards Boxing is implicitly(automatically) done by java compiler.Hence it is also known as AutoBoxing.

EX: Auto Boxing

```
package collection;

public class Warpper_2 {
public static void main(String[] args) {
		int a=10;//primitive
		Integer b=a;//Auto boxing
		System.out.println(b);//no-primitive
		double b1=20.5;//primitive
		Double c=b1;//Auto boxing
		System.out.println(c);//non-primitive
}
}
```

UnBoxing

The process of converting a non primitive value into primitive value is known as unboxing.

EX:Un-boxing

```
package collection;

public class Warpper_3{
public static void main(String[] args) {
        Integer a=10;//NON-Primitive value
        int b=a.intValue();//Unboxing
        System.out.println(b);//Primitive value
        Integer b1=new ~~Integer~~(100);//NON-Primitive value
        int c1=b1.intValue();//Unboxing
        System.out.println(c1);//Primitive value
        Double d=new ~~Double~~(20.5);//NON-Primitive value
        double d1=d.doubleValue();//Unboxing
        System.out.println(d1);//Primitive value
}
}
```

Note:-from JDK 1.5 onwards UnBoxing is implicitly(automatically) done by java compiler.Hence it is also known as AutoUnBoxing.

EX:Auto-unboxing

```
package collection;

public class Warpper_4{
public static void main(String[] args) {
        Integer a=10;//NON-Primitive value
        int b=a;//Auto-boxing
        System.out.println(b);//Primitive value
        Integer b1=new ~~Integer~~(100);//NON-Primitive value
        int c1=b1;//Auto-boxing
        System.out.println(c1);//Primitive value
        Double d=new ~~Double~~(20.5);//NON-Primitive value
        double d1=d;//Auto-boxing
        System.out.println(d1);//Primitive value
}
}
```

Converting String into respective primitive type

1.String to byte

Byte.parseByte(String)

2.String to short

Short.parseShort(String)

3.String to int

Integer.parseInt(String)

4.String to long

Long.parseLong(String)

5.String to float

Float.parseFloat(String)

6.String to double

Double.parseDouble(String)

7.String to boolean

Boolean.parseBoolean(String)

EX: String into Primitve type

```
public class String_into_primitive {
public static void main(String[] args) {
          String s="123";
          int a=Integer.parseInt(s);
          System.out.println(a);
          System.out.println(a+10);//to check
          double d=Double.parseDouble(s);
          System.out.println(d);
          System.out.println(d+10);
          System.out.println(Byte.parseByte(s)+1);//dirctly in sopln statment
}
}
```

```
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
```

COLLECTION FRAMEWORKS

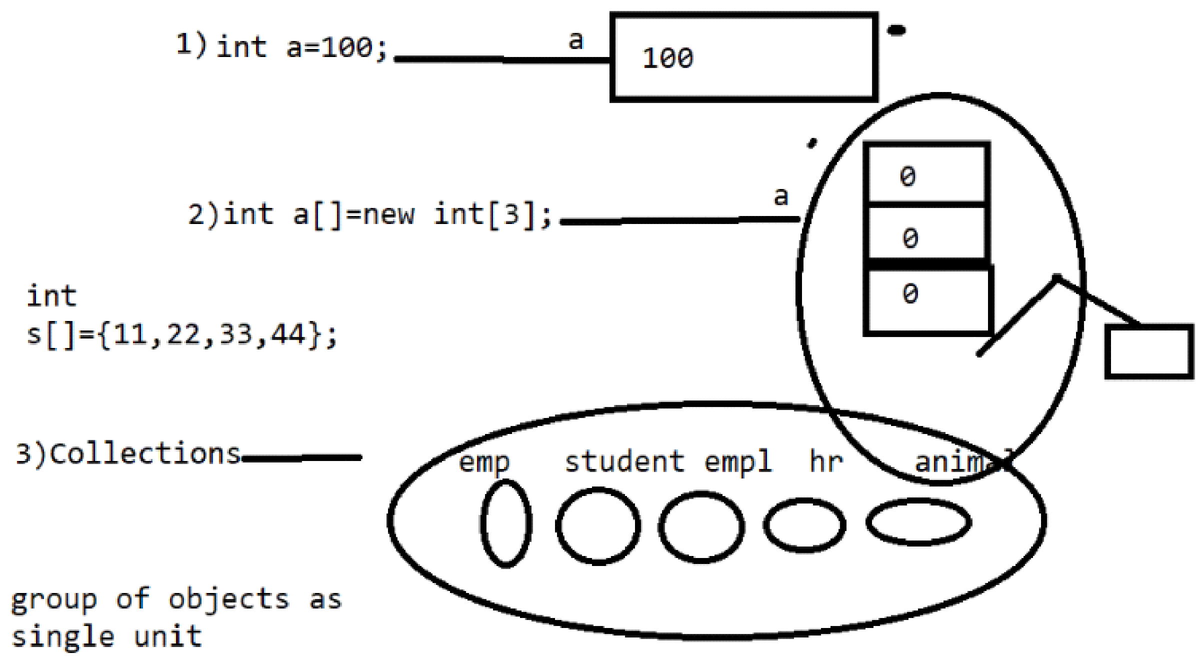
COLLECTION

• Collection is nothing but a group of objects represents as single unit.
• Its main purpose is to store huge amount of data.
• It provides multiple API(methods) to store and manipulate data.

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.

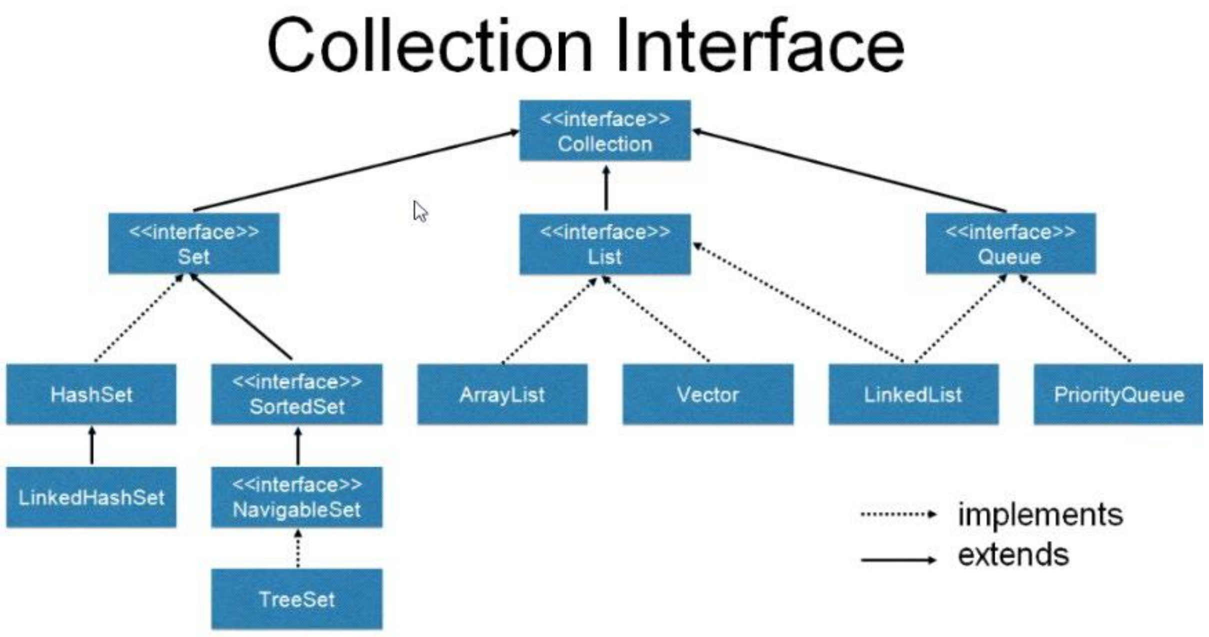


CollectionsFrameworks

• 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.

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(I)

It defines most common methods for all collection objects.

1)isEmpty()
2)add(Object ref)
3)addAll(Collection ref)
4)size()
5)remove(Object ref)
6)contains(Object ref)
7)removeAll(Collection ref)
8)retainAll(Collection ref)
9)containsAll(Collection ref)
10)clear()
11)toArray()
12)iterator()

List(I)

----

1)It is a child of Collection(I)
2)It is used to store group of objects together as a single entity where duplicates are allowed and insertion order is preserved.
3)index plays an important role here.

List interface contains all methods of Collection interface. And also contains following methods,

1)add(int index,Object ref)
2)addAll(int index,Collection ref)
3)remove(int index)
4)lastindexOf(Object ref)
5)indexOf(Object ref)
6)get(int index)
7)set(int index,Object ref)
8)listIterator()

ArrayList

It is a concrete subclass of List interface.

Characteristics of ArrayList

----
1)accepts duplicate objects
2)it maintains insertion order
3)iteration order predictable
4)dynamically growable or shrinkable
5)accepts both homogenous and heterogeneous

Steps to create ArrayList

----
1)import arraylist class from java.util package
2)create an object

Constructors of ArrayList

ArrayList()
ArrayList(Collection ref)
ArrayList(int initialcapacity)

Eg:-

```
import java.util.ArrayList;
class A {
	public static void main(String[] args) {
		ArrayList a1=new ArrayList();
		System.out.println(a1.isEmpty());//true
		a1.add(12);
		a1.add(true);
		a1.add("hello");
		System.out.println(a1.isEmpty()); //false
		System.out.println("size of arraylist :"+a1.size()); //3
		System.out.println(a1);//[12, true, hello]
		a1.add(12);
		System.out.println("size of arraylist :"+a1.size()); //4
		System.out.println(a1);		//[12, true, hello, 12]
	}
}
```

Note:-We can add elements of one arrayList into another with the help of add() or addAll() method .

add() method adds the arraylist as a single object.

addAll() adds elements of arraylist as seperate objects.

Eg:-import java.util.ArrayList;
```
class B {
	public static void main(String[] args) {
		ArrayList a1=new ArrayList();
		a1.add(10);
		a1.add(20);
		a1.add(30);

		ArrayList a2=new ArrayList();
		a2.add(40);
		a2.add(50);

		a2.add(a1);

		System.out.println("Size :"+a2.size()); //3
		System.out.println(a2); //[40, 50, [10, 20, 30]]

		a2.addAll(a1);

		System.out.println("Size :"+a2.size()); //6
		System.out.println(a2);  //[40, 50, [10, 20, 30], 10, 20, 30]
	}
}
```

Note:- We can access elements of ArrayList using get(int index) method.

```
import java.util.ArrayList;
class C {
	public static void main(String[] args) {
		ArrayList a1=new ArrayList();
		a1.add(10);
		a1.add(20);
		a1.add(30);

		System.out.println(a1); //[10, 20, 30]

	    for (int i=0;i<a1.size() ;i++ )  {
			   System.out.println(a1.get(i));
	    }
	}
}
```

for-each(enhanced for loop)

----

syntax--> for(Object ref1:Collection ref2) {
Statements;
}

```
          Statements;
            }
```

Eg:- import java.util.ArrayList;

```
class F {
	public static void main(String[] args) {
		ArrayList a1=new ArrayList();
		a1.add(10);
		a1.add(20);
		a1.add(30);
		a1.add(40);

		for(Object a:a1){
			System.out.println(a);
		}
	}
}
```

```
          for(Type ref1:Collection ref2)  ---->(for generic arraylist)   {
          Statements;
            }
```

Eg:-

```
import java.util.ArrayList;
class G {
	public static void main(String[] args) {
		ArrayList<Integer>a1=new ArrayList<>();
		a1.add(10);
		a1.add(20);
		a1.add(30);
		a1.add(40);

		for(Integer a:a1){
			System.out.println(a);
		}
	}
}
```

Iterator

Iterator is used to iterate through each and every element in the collection.

Steps to create Iterator

----

step1: Import Iterator from java.util package.

step2: create an Iterator with the help of iterator() method.

step3:Using hasNext() and next() iterate through collection as shown below.

Eg:-

```
import java.util.ArrayList;
import java.util.Iterator;
class I {
        public static void main(String[] args) {
                ArrayList<Integer> a1=new ArrayList<>();
                a1.add(100);
                a1.add(200);
                a1.add(300);
                a1.add(100);

                Iterator i=a1.iterator();

                while(i.hasNext()){
      System.out.println(i.next());
                }
        }
}
```

Comparable interface

Comparable is an interface having a non static abstract method compareTo(Object ref).

The Collections.sort() uses compareTo() internally to sort the elements of ArrayList.Therefore the objects present inside the ArrayList must be Comparable type. i.e the class for which the objects are created must implement Comparable interface and provide implementaion to compareTo method.

Note:- All wrapper classes by default implements Comparable interface and provides implementaion to compareTo() method.

Eg:-
```
class Emp implements Comparable{
	int eid;

	Emp(int eid){
		this.eid=eid;
	}

	public String toString(){
		return "Eid :"+eid;
	}

	public int compareTo(Object obj){
		if(this.eid==((Emp)obj).eid){
	return 0;
		}
		else if(this.eid>((Emp)obj).eid){
			return 1;
		}
		else{
			return -1;
		}
	}

}
```

Eg:-
```
import java.util.ArrayList;
import java.util.Collections;
class Driver1 {
	public static void main(String[] args) {
		Emp e1=new Emp(2);
		Emp e2=new Emp(3);
		Emp e3=new Emp(1);

		ArrayList<Emp>a1=new ArrayList<>();

		a1.add(e1);
		a1.add(e2);
		a1.add(e3);

		System.out.println(a1);

		Collections.sort(a1);

		System.out.println(a1);

	}
}
```

contains(Object ref)

It is used to check whether given element is present in the collection or not.

Eg:

```
 import java.util.ArrayList;
class M {
	public static void main(String[] args) {
		ArrayList a1=new ArrayList();
		a1.add(10);
		a1.add(20);
		a1.add(30);
		a1.add(40);
		System.out.println(a1.contains(10)); //true
	}
}
```

containsAll(Collection ref)

It is used to check whether the ArrayList contains all the elements of given arraylist.

```
import java.util.ArrayList;
class M {
	public static void main(String[] args) {
		ArrayList a1=new ArrayList();
		a1.add(10);
		a1.add(20);
		a1.add(30);
		a1.add(40);
		ArrayList a2=new ArrayList();
		a2.add(10);
		a2.add(20);
		System.out.println(a1.containsAll(a2)); //true
	}
}
```

clear()

It is used to remove all the elements from the arraylist.

```
import java.util.ArrayList;
class M {
	public static void main(String[] args) {
		ArrayList a1=new ArrayList();
		a1.add(10);
		a1.add(20);
		a1.add(30);
		a1.add(40);
		    a1.clear();
		System.out.println(a1.size());//0
	}
}
```

indexOf(Object ref)

It gives the index of given object(first occurence).

lastIndexOf(Object ref)

It gives the index of given object(last occurence).

```
import java.util.ArrayList;
class M
{
	public static void main(String[] args)
	{
		ArrayList a1=new ArrayList();
		a1.add(10);
		a1.add(20);
		a1.add(30);
		a1.add(40);
		a1.add(30);

		System.out.println(a1.indexOf(30)); //2
		System.out.println(a1.lastIndexOf(30)); //4
	}
}
```

set(int index,Object ref)

It is used to replace the element present at specified index with the given element.

Eg:-

```
import java.util.ArrayList;
class M {
	public static void main(String[] args) {
		ArrayList a1=new ArrayList();
		a1.add(10);
		a1.add(20);
		a1.add(30);
		a1.add(40);
		System.out.println(a1); //[10, 20, 30, 40]
		a1.set(1,100);
		System.out.println(a1); //[10, 100, 30, 40]
	}
}
```

Removing elements from ArrayList

Elements from ArrayList can be removed using following methods.

remove(Object obj)--->Removes given object

remove(int index) --->Removes object based on given index

```
/*
remove(Object)

remove(index)

*/
```

Eg:-import java.util.ArrayList;
```
class J {
	public static void main(String[] args) {
		ArrayList a1=new ArrayList();
		a1.add(1);
		a1.add(20);
		a1.add(25);
		a1.add(50);
		a1.add(2);
		a1.add(2.5);
		a1.add("hi");

		System.out.println(a1);//[1, 20, 25, 50, 2, 2.5, hi]

		a1.remove(2.5); //removes object directly
		System.out.println(a1);//[1, 20, 25, 50, 2, hi]
		a1.remove("hi");
		System.out.println(a1);//[1, 20, 25, 50, 2]
		a1.remove(4); //removes 2 based on index
		System.out.println(a1); //[1, 20, 25, 50]
		a1.remove(new Integer(50));
		System.out.println(a1); //[1, 20, 25]
	}
}
```

LinkedList

LinkedList is concrete subclass of List interface.
It belongs to java.util package.

characteristics of LinkedList
----
1)Maintains the insertion order
2)Iteration order is predictable
3)Dynamically growable or shrinkable
4)Accepts duplicate objects
5)Can store both homogeneous as well as heterogeneous objects.

Steps to create a LinkedList
----
step1:- import LinkedList class from java.util package.

step2:- Create an object using the constructors.

Constructors of LinkedList
----
1)LinkedList()

2)LinkedList(Collection ref)

LinkedList specific methods
----
1)addFirst(Object ref)
2)addLast(Object ref)
3)getFirst()
4)getLast()
5)removeFirst()
6)removeLast()

Eg1:-
```
import java.util.LinkedList;
class K {
        public static void main(String[] args) {
                LinkedList l1=new LinkedList();
                l1.add(10);
                l1.add(20);
                l1.add("hello");
                l1.add(20);

                System.out.println(l1); //[10, 20, hello, 20]
                System.out.println(l1.size()); //4

     for(Object a:l1){
                        System.out.println(a);
                }
        }
}
```

Eg2:-

```
import java.util.LinkedList;
class L {
        public static void main(String[] args) {
                LinkedList<Integer>l1=new LinkedList<>();
                l1.add(10);
                l1.add(20);
                l1.add(30);
                l1.add(40);

                System.out.println(l1); //[10, 20, 30, 40]
                l1.addFirst(100);
                l1.addLast(200);
                System.out.println(l1); //[100, 10, 20, 30, 40, 200]

                System.out.println("First Element "+l1.getFirst());
                System.out.println("Last Element "+l1.getLast());

                l1.removeFirst();
                l1.removeLast();
                System.out.println(l1); //[10, 20, 30, 40]

        }
}
```

Difference between ArrayList and LinkedList

ArrayList	**LinkedList**
1.Best choice for retrieval.	1.Best choice for inserting and deleting from middle
2.worst choice for inserting and deleting from middle.	2.worst choice for retrieval
3.Underlying datastructure is growable array.	3.Underlying datastructure is doubly linked list.

Vector

It is a concrete subclass of List interface.
It belongs to java.util package.

characteristicks of Vector
----
1)Maintains the insertion order
2)Iteration order is predictable
3)Dynamically growable or shrinkable
4)Accepts duplicate objects
5)Can store both homogeneous as well as heterogeneous objects.

Note:- Vector came before ArrayList,List and Collection(I). i.e, in java 1.0 itself.

After java 1.2 vector is the child of Collection(I) and List(I).
Therefore it has 3 methods for adding an element
1)add(Object ref) --->from Collection(I)
2)add(int index,Object ref)--->from List(I)
3)addElement(Object ref) --->Vector

for removing,
1)remove(object ref)-->from Collection(I)
2)remove(int index)-->from List(I)
3)removeElement(Object ref)-->from Vector
4)clear()-->from Collection(I)
5)removeAllElements()-->from Vector

for Accessing elements,
1)get(int index) -->from List(I)
2)elementAt(int index) -->from Vector
3)firstElement()-->from Vector
4)lastElement()-->from Vector

Eg:-

```
import java.util.Vector;
class M{
	public static void main(String[] args) {
		Vector v=new Vector();
		v.add(10);
		v.addElement(200);
		v.addElement(20);
		v.addElement(2);

		System.out.println(v);

		v.removeElement(20);

		System.out.println(v);
		System.out.println("First Element :"+v.firstElement());
		System.out.println("Last Element :"+v.lastElement());
		System.out.println(v.elementAt(2));

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

Difference between ArrayList and Vector

ArrayList	**Vector**
-----------	----------------
1.All methods are non synchronized	1.All methods are synchronized
2.performance high compared to vector.	2.performance low
3.introduced in java 1.2 (non-legacy)	3.introduced in java 1.0(legacy class).

Stack

Stack is subclass of Vector.It belongs to java.util package.
It is specially designed for First in last out.

Constructor

Stack()

Methods

----

1)push(Object obj)

It is used to insert an object to the stack.

2)pop()

It is used to remove and return top of the stack.

3)peek()

Returns the top of the stack without removal of object.

Eg:-

```
import java.util.Stack;
class N {
	public static void main(String[] args) {
		Stack s1=new Stack();

		s1.push(10);
		s1.push(20);
		s1.push(30);

		System.out.println(s1.pop()); //30
		System.out.println(s1);//[10, 20]
		System.out.println(s1.peek());
		System.out.println(s1); //[10, 20]
	}
}
```

SET

When to choose SET?

1.If we dont want store duplicate objects.

2.If insertion order is not required.

Set(I) extends Collection(I). And it belongs to java.util package.

HashSet

HashSet is the concrete subclass of Set(I).

characteristics of HashSet

1.Doesn't accept duplicate objects.

2.Doesn't maintain insertion order.

3.We cannot predict iteration order.

4.We can add and remove elements.

5.Accepts both homogeneous as well as heterogeneous objects.

Constructors

1.HashSet()

2.HashSet(Collection ref)

Eg:-

```
import java.util.HashSet;
class D
{
public static void main(String[] args)
{
HashSet h1=new HashSet();
h1.add(10);
h1.add(20);
h1.add(30);
h1.add(40);
System.out.println(h1);
System.out.println(h1.size()); //4
}
}
import java.util.HashSet;
class E{
public static void main(String[] args){
HashSet h1=new HashSet();
System.out.println(h1.add(10)); //t
System.out.println(h1.add(20)); //t
System.out.println(h1.add(30)); //t
System.out.println(h1.add(40)); //t
System.out.println(h1.add(10));//f
for (Object i:h1){
System.out.println(i);
}
}
}
```

Eg:-import java.util.HashSet;

```
class F{
public static void main(String[] args){
//Generic HashSet
HashSet<Integer>h1=new HashSet<>();
h1.add(10);
h1.add(20);
h1.add(30);
h1.add(10);
System.out.println(h1.size());//3
h1.add("hello");//cte
}
}
```

Sorting elements of HashSet

--

As HashSet doesn't maintain a perticular order of objects, We don't have any mechanism designed to sort the objects
of HashSet.
Note: If we really want to sort the objects of HashSet we can convert it into ArrayList and sort using Collections.sort
() or we can convert it into TreeSet.

Eg:-
```
import java.util.HashSet;
import java.util.ArrayList;
import java.util.Collections;
class H
{
public static void main(String[] args)
{
HashSet<Integer>h1=new HashSet<>();
h1.add(10);
h1.add(60);
h1.add(30);
h1.add(100);
System.out.println(h1);
//CONVERTING HashSet into ArrayList
ArrayList a1=new ArrayList(h1);
Collections.sort(a1);
for(Object i:a1){
System.out.println(i);
}
}
}
```

TreeSet

TreeSet is a concrete subclass of Set(I) and it belongs to java.util package.
characteristics of TreeSet

1.Doesn't accept duplicate objects.
2.TreeSet gives the objects in ascending order.
3.We can add and remove elements.
4.Accepts homogeneous objects.

Constructors

1.TreeSet()
2.TreeSet(Collection ref)
3.TreeSet(Comparator ref)
Eg:-

```
import java.util.TreeSet;
import java.util.Iterator;
class I{
public static void main(String[] args){
TreeSet t1=new TreeSet();
t1.add(14);
t1.add(78);
t1.add(60);
t1.add(160);
t1.add(6);
System.out.println(t1); //[6,14,60,78,160]
System.out.println(t1.size()); //5
}
}
```

LinkedHashSet

LinkedHashSet is an implementing class of Set interface.
It belongs to java.util package.
characteristics of LinkedHashSet

1.Doesn't accept duplicate objects.
2.maintains Insertion order.
3.We can predict iteration order.
4.We can add and remove elements.
5.Accepts homogeneous and heterogeneous objects.

Eg:-

```
import java.util.LinkedHashSet;
class K{
public static void main(String[] args){
LinkedHashSet h1=new LinkedHashSet();
h1.add(10);
h1.add(60);
h1.add(40);
h1.add(120);
h1.add(10);
System.out.println(h1); //[10,60,40,120]
System.out.println("size :"+h1.size()); //4
for(Object i:h1){
System.out.println(i);
}
}
}
```

o/p

```
----
[10,60,40,120]
size :4
10
60
40
120
```

Comparator

Comparator is an interface which belongs to java.util package. It has an abstract method i.e public int compare(Object obj1,Object obj2).

Comparator is used for customized sorting as shown in below examples.

Sorting elements of ArrayList in descending order of Integers using Comparator

----

Eg:-class MyComparator implements Comparator{

```
//implementation given to get descending order of integers
public int compare(Object obj1,Object obj2){
int i1=(Integer)obj1;
int i2=(Integer)obj2;
if(i1==i2){
return 0;
}
else if(i1>i2){
return -1;
}
else{
return 1;
}
}
}
```

Eg:-

```
import java.util.ArrayList;
import java.util.Collections;
class H
{
public static void main(String[] args)
{
ArrayList<Integer> a1=new ArrayList<>();
a1.add(14);
a1.add(144);
a1.add(7);
a1.add(77);
a1.add(17);
Collections.sort(a1,new MyComparator());
System.out.println(a1); //[144,77,17,14,7]
}
}
```

To arrange elements of TreeSet in descending order of Integers(using Comparator)

```
class MyComparator implements Comparator{
//implementation given to get descending order of integers
public int compare(Object obj1,Object obj2){
int i1=(Integer)obj1;
int i2=(Integer)obj2;
if(i1==i2){
return 0;
}
else if(i1>i2){
return -1;
}
else{
return 1;
}
}
}
```

Eg:-import java.util.TreeSet;

```
class I{
public static void main(String[] args){
TreeSet t1=new TreeSet(new MyComparator());
t1.add(12);
t1.add(121);
t1.add(22);
t1.add(222);
System.out.println(t1); //[222, 121, 22, 12]
}
}
```

Example to sort elements of ArrayList based on multiple properties of the elements store in ArrayList

----

Eg:-class Emp implements Comparable

```
{
String name;
int eid;
double salary;
Emp(String name,int eid,double salary)
{
this.name=name;
this.eid=eid;
this.salary=salary;
}
//overriding toString() to get name of the employees instead of reference
//when we try to print view of ArrayList.
public String toString()
{
return name;
}
public int compareTo(Object obj)
{
if(this.eid==((Emp)obj).eid)
{
return 0;
}
else if(this.eid>((Emp)obj).eid)
{
return 1;
}
else
{
return -1;
}
}
}
```

Eg:-import java.util.Comparator;

```
class SalaryComparator implements Comparator{
//implementation given to sort in ascending order of salary
public int compare(Object obj1,Object obj2){
Emp e1=(Emp)obj1;
Emp e2=(Emp)obj2;
if(e1.salary==e2.salary){
return 0;
}
else if(e1.salary>e2.salary){
return 1;
}
else
{
return -1;
}
}
}
```

Eg:-import java.util.ArrayList;

```
import java.util.Collections;
class Driver2{
public static void main(String[] args){
Employee e1=new Employee("Ronak",3,80000.0);
Employee e2=new Employee("Bhaiyya",1,100000.0);
Employee e3=new Employee("Rahul",2,90000.0);
ArrayList<Employee> e=new ArrayList<>();
e.add(e1);
e.add(e2);
e.add(e3);
System.out.println("Before sorting");
System.out.println(e); //view //[Ronak, Bhaiyya, Rahul]
System.out.println("After sorting based on eid");
Collections.sort(e);
System.out.println(e);//[Bhaiyya, Rahul, Ronak]
System.out.println("After sorting based on salary");
Collections.sort(e,new SalaryComparator());
System.out.println(e); //[Ronak,Rahul,Bhaiyya]
}
}
```

Queue

Queue(I) extends Collection(I). and it belongs to java.util package.
Queue is specially designed for FIRST IN FIRST OUT.
methods

1.offer(Object)

Used to add the element to the queue.

2.poll()

Used to removes and returns the element from the queue.

3.peek()

Used to fetch the obect which is ready to be removed.
LinkedList and PriorityQueue are implementing classes of Queue interface.

Using LinkedList as Queue,

Eg:-import java.util.LinkedList;

```
class L{
public static void main(String[] args){
LinkedList l1=new LinkedList();
l1.offer(14);
l1.offer(25);
l1.offer(117);
l1.offer(7);
System.out.println(l1); //[14,25,117,7]
System.out.println(l1.poll()); //14
System.out.println(l1);//[25,117,7]
System.out.println(l1.peek()); //25
}
}
```

Eg:-

```
import java.util.LinkedList;
class M{
public static void main(String[] args){
LinkedList l1=new LinkedList();
l1.offer(14);
l1.offer(25);
l1.offer(117);
l1.offer(7);
while(!(l1.isEmpty())){
System.out.println(l1.poll());
}
}
}
```

OUTPUT

```
14
25
117
7
```

PriorityQueue

PriorityQueue is an implementin class of Queue interface. It is used to process the objects based on priority.

Eg:-

```
import java.util.PriorityQueue;
class N
{
public static void main(String[] args)
{
PriorityQueue p1=new PriorityQueue();
p1.offer(14);
p1.offer(36);
p1.offer(365);
p1.offer(18);
p1.offer(6);
System.out.println(p1);
while(!(p1.isEmpty()))
{
System.out.println(p1.poll());
}
}
}
```

o/p

```
----
6
14
18
36
365
```

Here the PriorityQueue is having Integer objects, Therefore priority is ascending order of the numbers.
Hence when we try to remove objects one by one using poll(), Objects are taken out in the ascending order.

To poll elements of PriorityQueue in descending order of Integers using Comparator

----

```
class MyComparator implements Comparator{
//implementation given to get descending order of integers
```

Eg:-public int compare(Object obj1,Object obj2){

```
int i1=(Integer)obj1;
int i2=(Integer)obj2;
if(i1==i2){
return 0;
}
else if(i1>i2){
return -1;
}
else{
return 1;
}
}
}
```

Eg:-import java.util.PriorityQueue;

```
class J{
public static void main(String[] args){
PriorityQueue p1=new PriorityQueue(new MyComparator());
p1.offer(12);
p1.offer(122);
p1.offer(152);
p1.offer(14);
p1.offer(6);
while(!(p1.isEmpty())){
System.out.println(p1.poll());
}
}
}
```

o/p

```
6
12
14
122
152
```

Map

Map is an interface, which is defined in java.util package.
Map interface helps to store the data in the form of key-value pairs.
Each key value pair in a Map is called Entry.
Eg:-RollNumber-Student, Eid-Employee etc

characteristics of Map

----

1)A map can cannot contain duplicate keys.
2)Map can contain duplicate values.
3)Each key can contain atmost one value, and not more than one.
Map interface allows 3 types of views,
1)A set of keys (As keys cannot be duplicate)
2)A List of values(As values can be duplicate)
3)A set of key-value Mapping

Methods

----

1)put(k,v)
2)putAll(Map)
3)size()
4)clear()
5)isEmpty()
6)containsKey(key)
7)containsValue(value)
8)get(key)
9)equals(Object)
10)remove(key)
11)keySet()
12)values()
13)entrySet()

HashMap

HashMap is an implementing class of Map interface.
HashMap is present in java.util package.
characteristics

----

1)Does not maintain insertion order of the entries
2)We cannot predict the iteration order.

Eg:-import java.util.HashMap;

```
class A{
public static void main(String[] args){
HashMap h1=new HashMap();
h1.put(1,"naveen");
h1.put(2,"rakesh");
h1.put(3,"pramod");
h1.put(4,"ashith");
h1.put(5,"praveen");
System.out.println(h1); //view
System.out.println(h1.size()); //5
h1.put(2,"manoj");
System.out.println(h1); //view
System.out.println(h1.size()); //5
h1.put(6,"ashith");
System.out.println(h1); //view
System.out.println(h1.size()); //6
h1.clear();
System.out.println(h1); //view
System.out.println(h1.size()); //6
}
}
```

Eg:-
```
import java.util.HashMap;
class B{
public static void main(String[] args){
HashMap<Integer,String> h1=new HashMap<>();
h1.put(10,"Hello");
h1.put(2,"Hi");
h1.put(8,"Bye");
h1.put(18,"tata");
//h1.put("C u",12); //CTE
System.out.println(h1.containsKey(8)); //true
System.out.println(h1.containsValue("Hi")); //true
System.out.println(h1.get(2)); //Hi
h1.remove(8);
System.out.println(h1);
}
}
```

Extracting only keys from the HashMap

We can extract only keys from the HashMap using keySet() method. KeySet() returns a Set of keys.
Return type of keySet() is Set.
```
import java.util.HashMap;
import java.util.Set;
```

Eg:-class C{
```
public static void main(String[] args){
HashMap h1=new HashMap();
h1.put(1,"naveen");
h1.put(2,"rakesh");
h1.put(3,"pramod");
h1.put(4,"ashith");
h1.put(5,"praveen");
Set s=h1.keySet();
for(Object i:s){
System.out.println(i);
}
}
}
```
OUTPUT
```
1
2
3
4
5
```

Extracting only values from the HashMap

We can exctract only values from the HashMap using values().
Returntype of values() is Collection.

Eg:-

```
import java.util.HashMap;
import java.util.Collection;
class D{
public static void main(String[] args){
HashMap h1=new HashMap();
h1.put(1,"naveen");
h1.put(2,"rakesh");
h1.put(3,"pramod");
h1.put(4,"ashith");
h1.put(5,"praveen");
Collection v=h1.values();
for(Object i:v){
System.out.println(i);
}
}
}
```

o/p

```
naveen
rakesh
pramod
ashith
praveen
```

To access key-value pair one by one from HashMap

HashMap doen't have get(index) method And it doesn't support Iterator.
Therefore we can access key-value par(entry) individually using entrySet() (which returns set of entries)and for-eac
h loop as shown in below example.

Eg:-import java.util.HashMap;

```
import java.util.Set;
import java.util.Map;
class E
{
public static void main(String[] args)
{
HashMap<String,Integer>h1=new HashMap<>();
h1.put("Tuborg",180);
h1.put("Kf",160);
h1.put("UB",150);
h1.put("Budwiser",190);
h1.put("Carlsberg",200);
h1.put("Corona",230);
h1.put("Old monk",200);
System.out.println(h1);
for(Map.Entry<String,Integer> i:h1.entrySet()){
System.out.println(i);
}
}
}
```

Note:- In a Map each entry is of Map.Entry<k,v> type.
Map.Entry is a nested interface. (Entry is an interface present inside Map interface).
Note:- Using Map.Entry type reference we can access keys and values seperately, using getKey() and getValue() res
pectively.

```
for(Map.Entry<String,Integer> i:h1.entrySet()){
System.out.println("Key ="+i.getKey()+" Value= "+i.getValue());
}
```

Eg:-

```
import java.util.*;
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
```

TreeMap

----

1)TreeMap doesn't maintain insertion order.
2)It stores the elements as per the natural sorting order of keys.
3)We can predict the iteration order(As per the natural sorting order of keys).
4)Doesn't accept heterogeneous keys.
5)Introduced in 1.2v

Eg:-

```
import java.util.*;
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)
```

LinkedHashMap

----

1)introduced in 1.4v
2)heterogenous data allowed
3)Data structure is hashtable
4)duplicate keys are not allowed but values can be duplicate,if we add duplicate key it
5)replaces with original one.
6)As per Insertion order
7)only one null key is allowed and multiple null values are allowed.

Eg:-

```
import java.util.*;
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}
```

Author:-Prem Kumar Choudhary