JAVA Notes(Main Part - 1)

Note: Strings part is covered in saparate pdf.(5,10,17 nov and half portion of 4th nov)

Language Fundamentals

1. **Identifiers**

A name in java program is called "identifier".The name can be a classname,interfacename,enumname,varaiblename,methodname and label name.

```
class Test{
    public static void main(String[] args){
        int x = 10;
    }
}
```

Identifiers: Test,main,String,args,x

Rules for Identifiers

Rule1: The allowed characters in java identifiers are a to z, A to Z,0 to 9,_,$

Rule2: If we use any other character, then the program would result in "CompileTime Error".

Rule3: Identifiers should not start with digits

eg: PWSKILLS123---> valid

123PWSKILLS---> invalid

Rule4: java identifiers are case sensitive,as such Java language only is case sensitive(JVM)

```
class Test{
    int number = 10;
    int NUMBER = 100;
    int NuMbeR = 1000;
}
```

Rule5: There is no constraint of the length of the java identifiers,but it is not recomended to take the length more than 15.

eg: int physicsWallahAlakhPandeyJavaWithMicroservices = 100;

Rule6: Reserve words or builtin words can't be used as "java identifiers". if we try to use then it would result in "CompiletimeError".

int,if,float,else,while,for,do,..... => reserve words/built in words/keywords

Inbuilt classname,interfacename,enumname can be used as a "identifier". Eventhough classname,interfacename,enumname can be used as an identifier, we don't recomend.

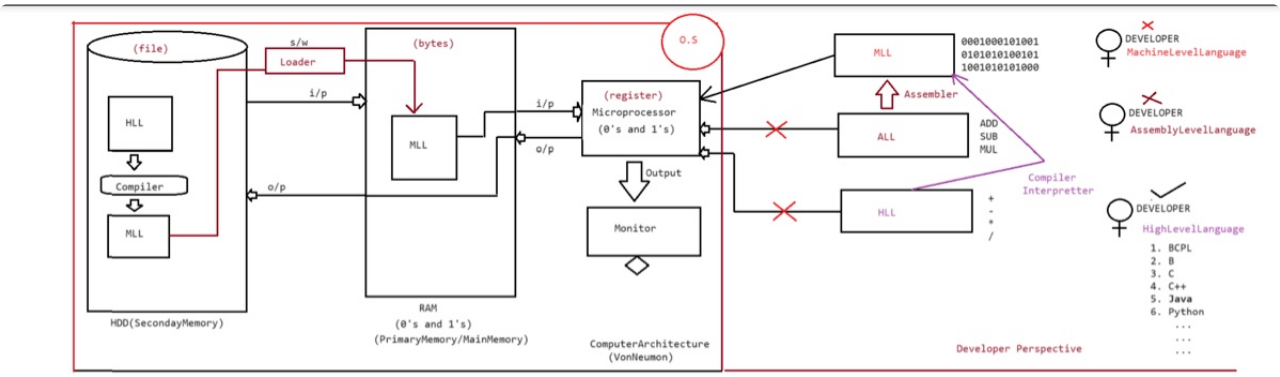
eg::

```
class Test{
    public static void main(String[] args){
        int int=10; //CE
        int if =10; //CE
        int while=100;//CE
    }
}
```

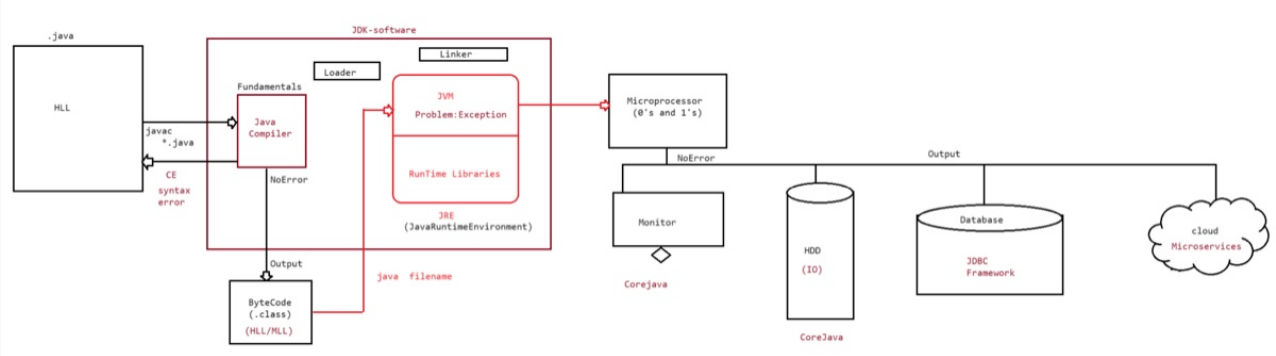
```
class Test{
    public static void main(String[] args){
        int String =10;
    }
}
```

Identify which are valid and invalid identifiers

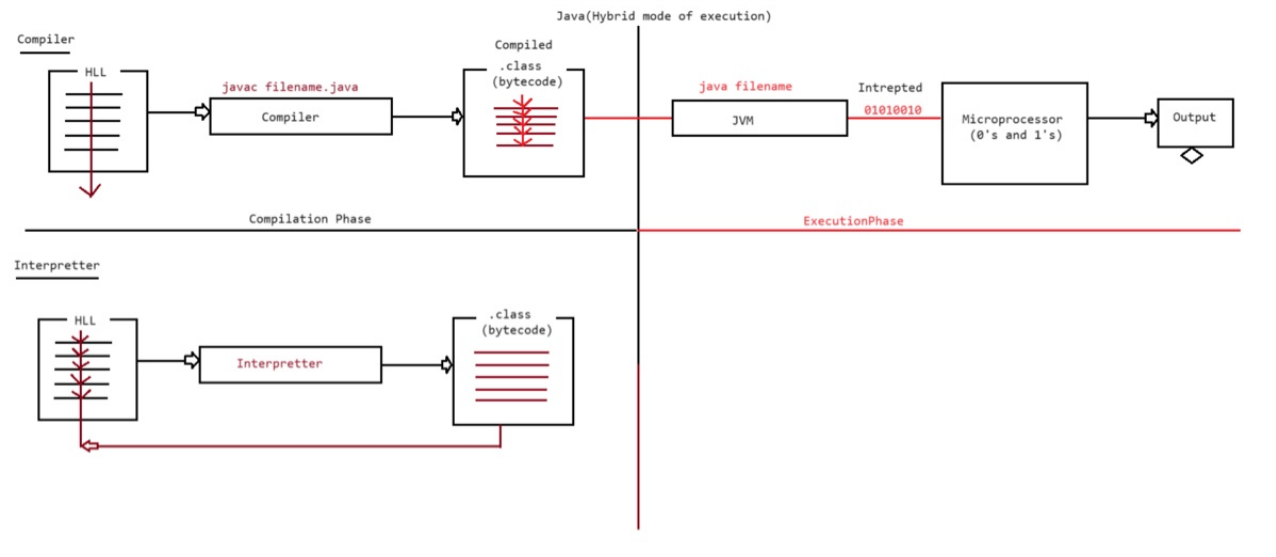
1. $ ===> valid
2. ca$h ==> valid
3. java2share=> valid
4. all@hands=> invalid
5. 123abc => invalid
6. Total# => invalid
7. Int => valid(not recomended)
8. Integer=> valid(not recomended)
9. int => invalid(reserve word)
10. tot123=> valid



Above image describes the difference between MLL,ALL and HLL and why HLL is best for developers + Computer Architecture + Compiler and Assembler



Above image describers how the execution of java program goes on and after successful execution it is stored in the form on 0's and 1's in HDD,JDBC and Cloud based on the application requirements .With the help of JDBC we can store data in database. With IO operations we can store it in HDD and in microservices we can store it in cloud.



Above example shows Hybrid mode of execution in java + Difference between Interpreter and compiler. In interpreter one line at a time is converted and executed while in compiler whole file is converted at a time and give for execution.

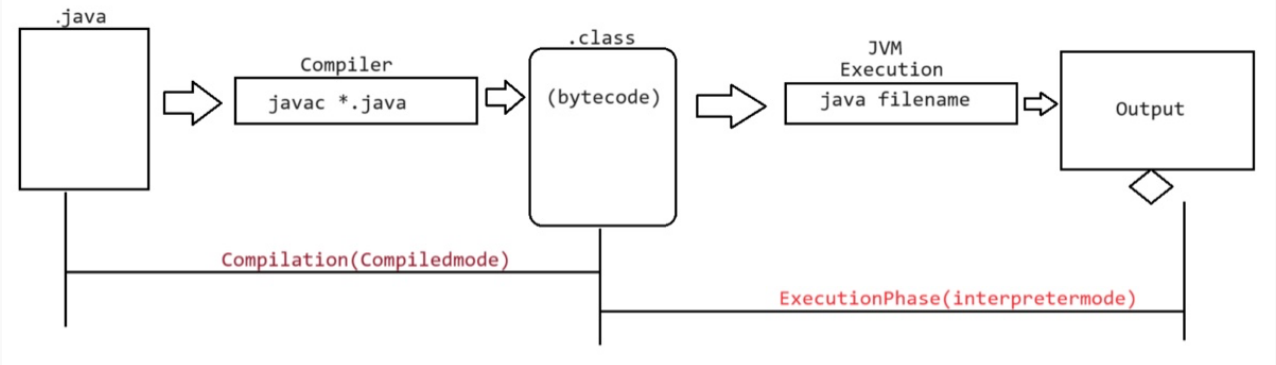
Compiler is a software which takes the instructions from source code(code writtenby developer), scans the code only once an d generates the instructions in MLL.

Interpetter is a software which takes the instructions from source code(code written by developers),scans the code line by line and generate the instructions in MLL.

How does java program executes?

Compilation(compiled mode) Execution(interpetted mode)

java ---------compiler ------> .class file(byte code)-------->JVM---------->Output



Reserve words

In java some identifiers are reserved to associate some functionality or meaning such type of reserverd indentifiers are called as "reserve words".

Reserve words for Datatypes(8)

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

Reserve words for flow control(11)

if,else,switch,case,default,for,while,do,break,continue,return

Reserve words for identifiers(11)

private,public,protected,final,abstract,native,static,strictfp,synchronized,transie t,volatile

Reserve words for ExceptionHandling(6)

try,catch,finally,throw,throws,assert(1.4 version)

Reserve words for Class types(7)

class,package,import,extends,implements,interface,enum

Reserve words for Object types(4)

new,isntanceof,super,this

Reserve word for returntype of methods(1)

void

Conculsions

1. All reserve words in java contains only lower case alphabets.
2. In java to create an object for a class we have "new" keyword,but we dont' have delete keyword to destroy the object where as destroying the object is taken care by a program(thread) called "GarbageCollector".

Which of the following list contains only java reserve words

a. final,finally,finalize(invalid)

b. throw,throws, thorwn(invalid)

c. break,continue,return,exit(invalid)

d. goto,constant(invalid)

e. byte,short,Integer(invalid),long

f. extends,implements,imports(invalid)

g. finalized(invalid),synchronized

h. instanceof,sizeOf(invalid)

i. new,delete(invalid)

j. instanceof,instanceOf(invalid)

k. public,static,void,main(invalid),String(invalid),args(invalid)

identifer :: name of class,methodname,variablename,labelname.

reservewords :: special meaning given for few identifiers(known to Compiler and JVM)

Software requriements to write java program

1. Download editplus from the following link ::

https://www.editplus.com/download.html

2. Download jdk sofware from the following link ::

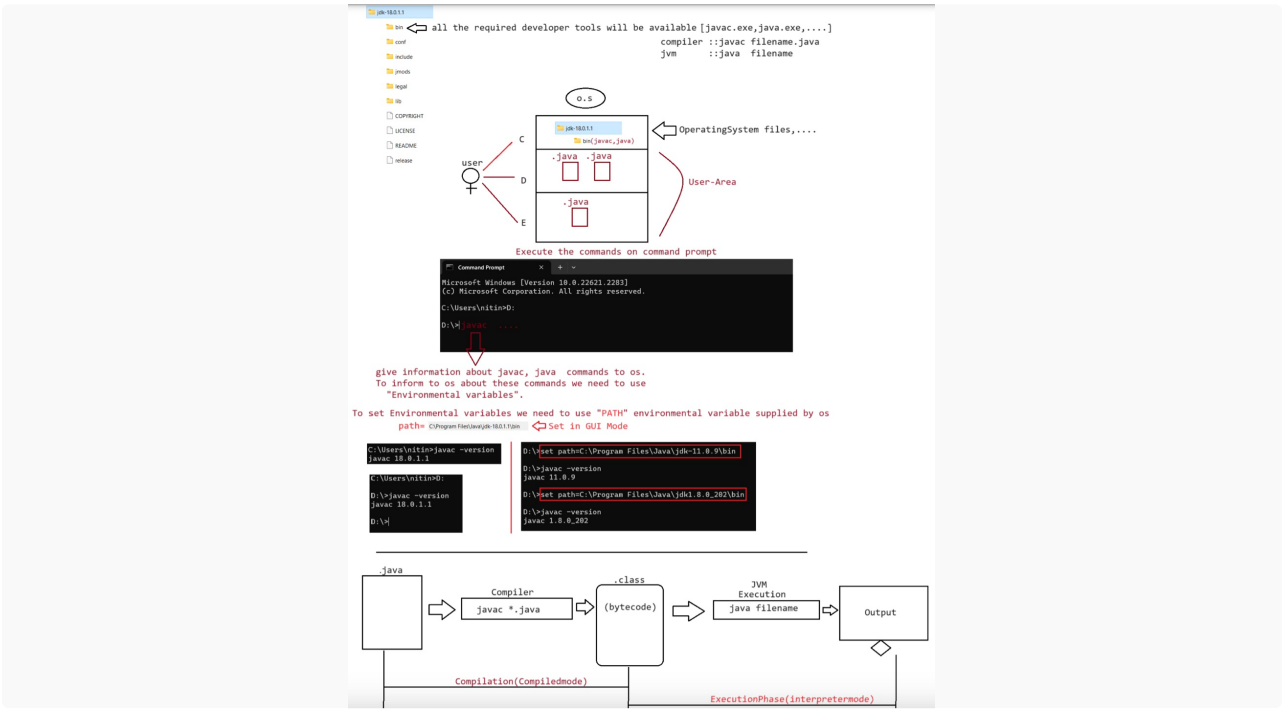
https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html

Steps to install jdk software

1. Double click on the downloaded setup file
2. click on next,next,next and finish the process.
3. Upon installation, the jdk software will be installed in the following location C:\Program Files\Java\jdk-18.0.1.1
4. set environmental variable called "PATH" to jdk/bin folder
5. Now our machine is ready to run java programs.

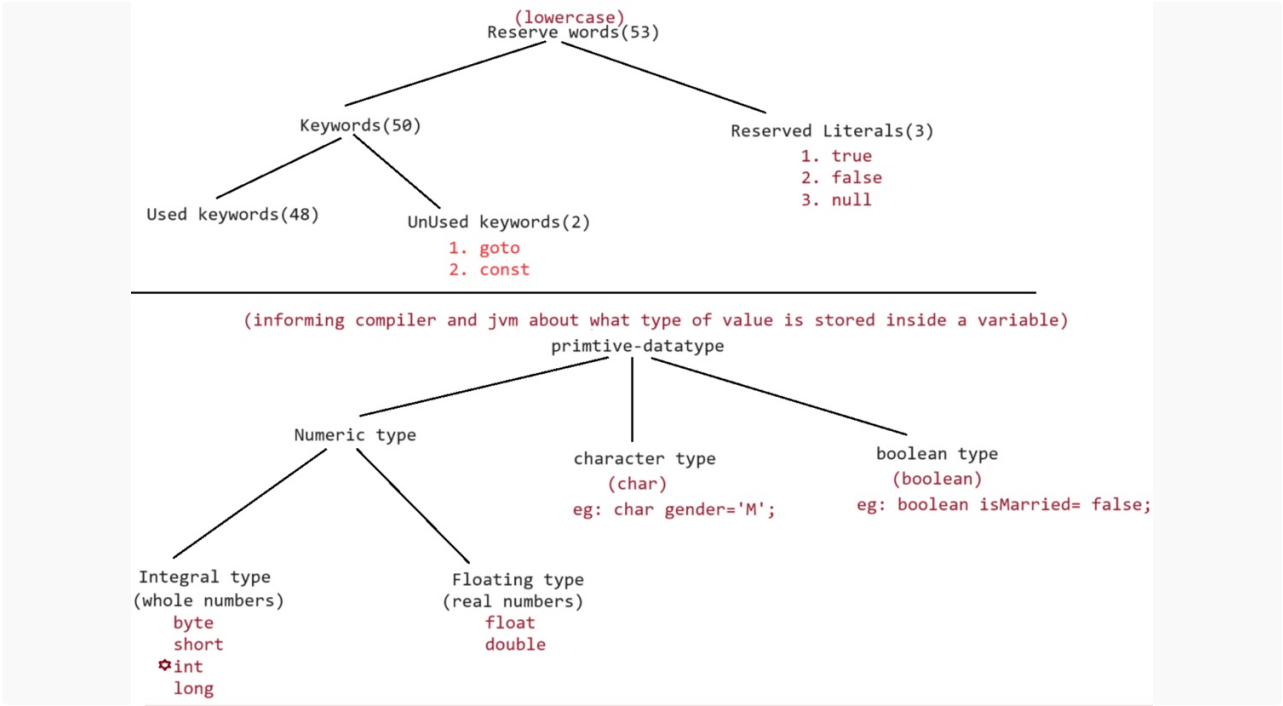
Steps to install editplus

1. Just doble click on editplus and install the software
2. Create one folder in any drive of your choice(D:\octbatchmicroservices)
3. Open editplus with this folder



Datatypes

Every variable has a type,every expression has a type and all types are Strictly defined moreover **every assignment should be checked by the compiler for the type compatibilty**,so we say java is **"Strictly typed programming langauge".**



byte

byte(size:1bytes(8bits))

minrange:-128(MIN_VALUE)

maxrange:+127(MAX_VALUE)

Corresponding wrapper class is java.lang.Byte

When to use byte datatype?

=> byte datatype is suitable only when we work with handling data in terms of streams either from the file or from the network.

Note::

If the no is positive number then first bit will be zero.

If the no is negative number then first bit will be one.

Negative number will be stored in 2's compliment.

eg

byte b=127;

byte b=130; //CE::loss of precision

byte b=10.5; //CE::Incompatible types

byte b=true; //CE::Incompatible types

byte b="ABC";//CE::Incompatible types

short

It is the rarely used data type in javalanguage.

Corresponding Wrapper class is java.lang.Short.

size:: 2 bytes

minrange::-32768(MIN_VALUE)

maxrange::+32767(MAX_VALUE)

short data type is mostly suitable only for 8086mp,since it is outdated we dont use

short data type in java language

eg:: short s=32456;

short s=32768//CE

short s=true;//CE

int

It is the most commonly used datatype to store numbers in java language

Corresponding wrapper class is java.lang.Integer

size:: 4bytes

minrange::-2147483648(MIN_VALUE)

maxrange::2147483647(MAX_VALUE)

eg:: int i=130;

int i=10.5//CE:incompatible types

int i=true;//CE::incompatible types

long

Whenever int is not enough to hold the value then we opt for long datatype

Corresponding wrapper class is java.lang.Long

size:: 8bytes

minrange::-9223372036854775808(MIN_VALUE)

maxrange::9223372036854775807(MAX_VALUE)

When to use long datatype?

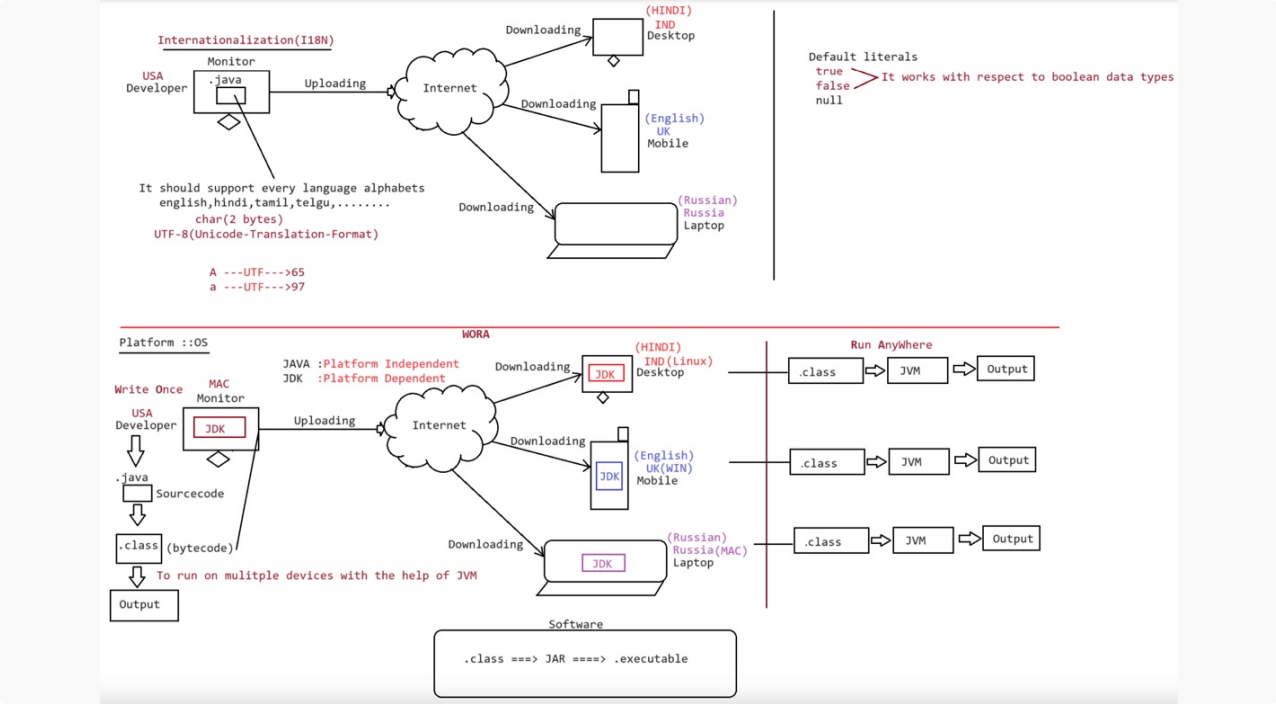
=> It is preferd when we work with file whose size is in terms of GB.

eg:: long l=35;

long l=35L;

int x =10L;

To specify the long type we can prefix it with 'L' or 'l'.



1)- Char in java is of 2 bytes due to Internationalization(in order to cover the characters of different languages.)

2)- WORA --> Write once run everywhere(Java --> Platform independent while JDK is Platform dependent). We can upload an application in the internet and users can use that application on their devices by using JDK(Platform dependent).

floating point data types

To work with real numbers we use the following datatypes

a. float

b. double

All the above datatypes are used to store whole number, if we want to store real number then we need to prefer using floating data type.

float

If we want to store the data in terms of accuracy upto 5 decimal places then we use float datatype.

Corresponding wrapper class is java.lang.Float

size 4bytes

minrange 1.4E-45(MIN_VALUE)

maxrange 3.4028235E38(MAX_VALUE)

eg float f=35.5;(double)

float f=35.5f;(float)

float f=35.5F;(float)

To specify the float type we can prefix it with 'f' or 'F'.

double

If we want to store the data in terms of accuracy upto 15 decimal places then we use double datatype.

Corresponding wrapper class is java.lang.Double

size 8bytes

minrange 4.9E-324(MIN_VALUE)

maxrange 1.7976931348623157E308(MAX_VALUE)

eg double d=35.5;(valid)

dobule d=35.5D;(valid)

dobule d=35.5d;(valid)

To specify the double type we can prefix it with 'd' or 'D'.

char types

It is represented by a single character with in a single quotes.

eg char c='a';//valid

char d=97;//valid

char c='ab';//invalid

char c= "a";//invalid

Corresponding wrapper class is java.lang.Character

size 2bytes(Since it supports Internationalization)

boolean types

The only allowed values for boolean type is true,false(case is also important)

eg:: boolean b=false;

boolean b=true;

boolean b ="true";//CE:Incompatible types

boolean b= 0;//CE:Incompatible types

Note: Reserve words for datatypes(byte,short,int,long,float,double,char,boolean)

byte(1)
short(2)
int(4)
long(8)
float(4)
double(8)
char(2)
boolean

Primitive Type casting

a. The process of converting data from one type to another type is called TypeCasting

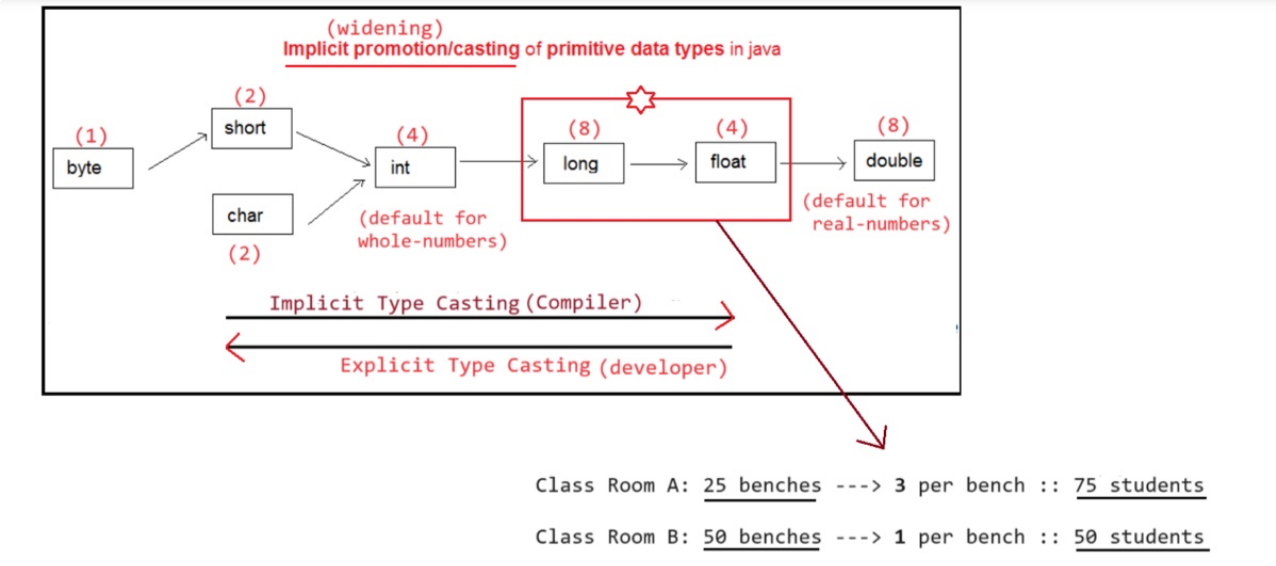
There are 2 types

1. Implicit type casting
2. Explicit type casting

Implicit TypeCasting

=> The process of converting data from lower data type to higher datatype is called "Implicit type casting".

1 2 4 8 4 8



byte => short => int => long => float => double

char

eg1:: byte b= 10;

int i=b;

System.out.println(b+" "+i);// 10 10

eg2:: int i=10;

byte b= i;//CE:: loss of precession

System.out.println(b+" "+i);

eg3:: byte b= 65;

char c= b;//CE: loss of precession

System.out.println(b+" "+c);

eg4:: char c ='A';

short s=c;//CE: loss of precession

System.out.println(c+" "+s);

Note:: byte and short internal representation is not compatible to convert into char.

eg5:: char c='A;

int i=c;

System.out.println(c+" "+i);// A 65

Note:: char internal representation is compatible with int type.

eg6:: byte b= 128;

System.out.println(b); //CE: possible of loss or precession

if the value assigned to a variable,if it reaches the max limit then compiler will automatically type promote the data to next higher data type. In case of byte and short the next higher data type is "int".

eg7:: byte b1=4;

byte b2=5;

byte b3= b1 + b2;//CE:: byte + byte = int

System.out.println(b3);

byte b1=60;

byte b2=70;

byte b3= b1 + b2;//CE:: byte + byte = int

System.out.println(b3);

Formulae:: In arithmetic operations the result will be always be Z= X + Y

if X and Y belongs to {byte,short,int} then Z should be int.
if either X or Y or both X and Y belongs to {long,float,double} then Z is max(X,Y)
eg8 long l= 10;
float f=l;
System.out.println(l + " " + f);//10 10.0
eg9 float f=10.5f;
long l= f;//CE:possible loss of precession
System.out.println(l + " " + f);
Note: refer diagram to understand how long value can sit in float and float value can't sit in long.

Explicit TypeCasting

=> The process of converting data from higher data type to lower data type is called "Explicit Type casting".

=> JVM will do Explicit Type casting only on the instructions given by the programmer.

=> In case of Explicit Type casting there would be loss of data.

Syntax:: P a = (Q) b;

P and Q should be primitive type and from Q to P there should be implicit relationship.

eg::

int i = 10;

byte b=(byte)i;

System.out.println(i);//10

System.out.println(b);//10

eg2:: int i=10;

short s= (byte)i;

System.out.println(i);//10

System.out.println(s);//10

eg3:: byte b= 65;

char c= (char)b;

System.out.println(b);//65

System.out.println(c);//A

eg4:: char c= 'A';

short s= (short)c;

System.out.println(c);//A

System.out.println(s);//65

eg5:: short s= 65;

char c = (byte) s;//CE

System.out.println(c);

System.out.println(s);

eg6:: byte b1=10;

byte b2=30;

byte b3=(byte) b1 + b2;//CE

System.out.println(b3);

eg7:: byte b1=10;

byte b2=30;

byte b3=(byte) (b1 + b2);

System.out.println(b3);//40

eg8:: double d= 22.222;

byte b= (byte)(long)(int)(short)d;

System.out.println(d);//22.222

System.out.println(b);//22

eg9:: int i = 130;

byte b= (byte)i;

System.out.println(b);//-126

solution:: minrange + (result -maxrange-1)

UTF-8

'A' => 65

'a' => 97

0 => 48

Note: Compiler -> Will performing typechecking(check value can be stored based on the range of values)

JVM -> Will allocate memory based on the datat type and performs the necessary operation(type casting)

Snippets

For the code below, what should be the name of java file?

```
public class HelloWorld {
    public static void main(String [] args) {
        System.out.println("Hello World!");
    }
}
```

A. Hello.java

B. World.java

C. HelloWorld.java[Answer]

D. helloworld.java

Convention in java-> Which ever class contains main(),check that classname and save the file with classname.

Does below code compile successfully?

```
public class Test {
    public static void main(String [] args) {
        System.out.println("Hello");;;;;;;;;
    }
}
```

A. yes[Answer]

B. no

What is the signature of special main method?

A. public static void main(String args)

B. public static void main(String[] a)[Answer]

C. public static void main()

D. private static void main(String[] args)

4. **What will be the result of compiling and executing Test class?**

```
public class Test {
    public static void main(String[] args){
        byte b1 = ( byte ) ( 127 + 21 );
        System.out.println(b1);//-108
    }
}
```

A. 148

B. Compilation Error

C. -108[Answer]

D. -128

Output: minRange + (result -maxRange-1)

-128 + (148 -127-1)

-128 + 20

-108

Consider below code:

```
public class Test {
    public static void main(String[] args) {
        char c = 'Z';
        long l = 100_00l;
        int i = 9_2;
        float f = 2.02f;
        double d = 10_0.35d;
        l = c + i; // char + int = int --> long
        f = c * l * i * f; // char * long * int * float --> float
        f = l + i + c; // long + int + char -----> long ---> float
        i = (int)d; //possible
        f = (long)d;//long -> float
    }
}
```

Does above code compile successfully?

A. Yes[Answer]

B. No

Consider below code of Test.java file:

```
public class Test {
    public static void main(String[] args) {
        char c1 = 'a'; //UNICODE VALUE 'a' is 97
        int i1 = c1; //Line n1
        System.out.println(i1); //Line n2
    }
}
```

What is the result of compiling and executing Test class?

A. a

B. 97[Answer]

C. Line n1 causes compilation failure

D. Line n2 causes runtime error.

In java to create an object for a class we have "new" keyword, but we don't have delete keyword to destroy the object where as destroying the object is taken care by a program(thread) called "GarbageCollector".

Every assignment should be checked by the compiler for the type compatibility, so we say java is "Strictly typed programming language".

eg:: float f=35.5;(double)

float f=35.5f;(float)

float f=35.5F;(float).

Implicit Type casting important points

1)- byte and short internal representation is not compatible to convert into char.

2)-char internal representation is compatible with int type.

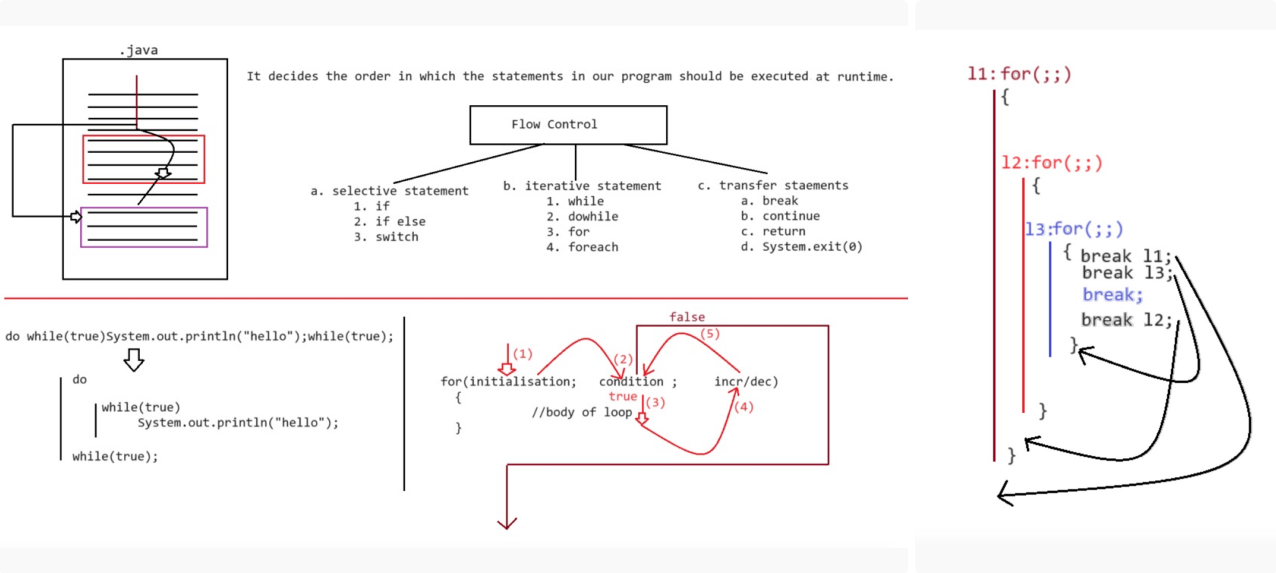
3)-If the value assigned to a variable, if it reaches the max limit then compiler will automatically type promote the data to next higher data type. In case of byte and short the next higher data type is "int".

4)-Formulae:: In arithmetic operations the result will be always be Z= X + Y if X and Y belongs to {byte, short, int} then Z should be int. if either X or Y or both X and Y belongs to {long, float, double} then Z is max(X,Y)

How many keywords are there in java?

Ans. 53

Conditional Statements Important Points



for detailed explanation of control statements and operators refer pdf from course videos → 15,20,21,22 oct +diagrams as well

1)-

int x=0;

if(x){

    System.out.println(x);

}

output:Compile time error(x should be a boolean(error: incompatible types: int cannot be converted to boolean))

2)- invalid cases

a. if(true) int x =10;

b. if(true){ int x =10;} System.out.println(x);

3)- In —>

```
switch(x){
    ...
}
```

x could be —> byte , short , char, int, Byte , Short , Character , Integer , enum , String

4)-

```
int x= 10;
switch (x){
    System.out.println("hiee");
}
```

Output: CE(If we are writing any statement inside switch then those statements should be a part of case or default.)

5)-

```
int x= 10;
int y= 20;
switch (x){
    case 10: System.out.println(10);
    case y: System.out.println(20); //CE: constant required
}
```

6)-

final -> This keyword makes the variable compile time constant, so compiler will be knowing the value of the variable.

```
int x= 10;
final int y= 20;
switch (x){     //byte,short,int,char
    case 10: System.out.println(10);
    case y: System.out.println(20);
}
```

Output: Code will be compiled

7)-Note: In java language, the memory for the variable will be given at the runtime , so we say java language as "Dynamic programming language".

8)-switch argument and case label can have expression, but case label should be constant expression.

```
int x= 10;
```

```
switch (x+1){
    case 10:
    case 10+20:
    case 10+20+30:
}
```

Output: No ouput

9)- case label values should lie in the range of switch argument type, otherwise it results in "Compiletime Error".

```
byte b= 10;
switch (b){ //byte = -128 + 127
    case 10 : System.out.println(10);
    case 100 : System.out.println(100);
    case 1000 : System.out.println(1000);//CE: not with in the range of byte
}
```

10)-. VERY VERY IMPORTANT

```
byte b= 10;
switch (b+1) { //byte + int => int
    case 10 : System.out.println(10);
    case 100 : System.out.println(100);
    case 1000 : System.out.println(1000);
}
```

Output: No Output

11)-

```
int x = 10;
switch (x){ //byte---int, short---int, char('a')---int(97), int
    case 97:System.out.println("97");
    case 99:System.out.println("99");
    case 'a':System.out.println("100");
}
```

Ouput: CE: duplicate case labels are not allowed.(`a` —> 97)

12)- Conclusion

a. Case label should be compile time constant.

b. Case label can have expression, but it should be a compile time constant expression.

c. Case label value should be within the range of the switch argument type.

d. Duplicate case label are not allowed.

13)-

```
int x = 3;
switch (x){
    default: System.out.println("default");
    case 0: System.out.println("0");
    case 1: System.out.println("1");
    break;
    case 2: System.out.println("2");
}
```

Output

x=3

default

0

1

(VERY VERY IMP —> As no break was present both 0 and 1 are also printed —> Fall Through in Switch (weird as case values are different))

14)-

```
int x = 3;
switch (x){
    default: System.out.println("default");
    default: System.out.println("default");
}
```

Output: CE: duplicate default label.

15)-Note: The argument to a while statement should be of "boolean type".if we are using anyother type it results in "CE".

A)-

```
while (1)
{
    System.out.printl("hello");   //CE
}
```

B)-

```
while (true)
    int x= 10; //CE: delcaration not allowed
```

C)-

```
while (true)
;
```

Output: no output(valid)

D)-

```
while (true)
{
    int x =10; //valid
}
```

E)-

```
while (true)
{
    int x =10;
}
System.out.println(x);    //CE: can't find symbol.
```

16)- Unreachable code —> very very imp —>

Note::

=> Every final variable will be replaced with corresponding value by the compiler

=> if any operation involves only constants then compiler is responsible to perform operation.

=> if any operation involves at least one variable, then compiler won't perform any operation, jvm will perform that operation.

eg#1.

```
while (true)
{
    System.out.println("hello");
}
System.out.println("hiee");  //CE: unreachable code
```

eg#2

```
while (false)
{
    System.out.println("hello");   //CE: unreachable code
}
System.out.println("hiee");
```

eg#3.

```
int a= 10;
int b= 20;
while (a<b).  //JVM :: 10<20 -> true
```

```
{
    System.out.println("hello");
}
System.out.println("hiee");
```

Output: hello(infinite times)

eg#4.

```
final int a= 10;
final int b= 20;
while (a<b).  //JVM :: 10<20 -> true
{
    System.out.println("hello");
}
System.out.println("hiee");  //unreachable code
```

Output: CE

eg#5.

```
final int a= 10;
while (a<20)
{
    System.out.println("hello");
}
System.out.println("hiee");  //unreachable code
```

Output: CE

eg#6.

```
int a= 10;
while (a<20)
{
    System.out.println("hello");
}
System.out.println("hiee");
```

Output: hello(infinite times)

17)- do-while

eg::

```
do{
    System.out.println("hello");
}while(true);
```

output:: hello infinite times

eg::

```
do;while(true);
```

output:: compiles succesfull

eg::

```
do
    int x=10;
while(true);
```

output:: compile time error

eg::

```
do{
    int x=10;
}while(true);
```

output:: compilation succesfull

eg::

```
dowhile(true);
```

output:: compilation error.

eg::

```
do{
    System.out.println("sachin");
}while(true);
System.out.println("dhoni");  //CE: unreachable code
```

eg::

```
int a=10,b=20;
do{
    System.out.println("sachin");
}while(a<b);
System.out.println("dhoni");
```

Output: sachin(infinite times)

eg::

```
final int a=10,b=20;
do{
    System.out.println("sachin");
}while(a<b);
```

```
System.out.println("dhoni");
```

output:Ce: unreachable(dhoni)

18)- for loop —>

eg::

```
int i=0;
for(System.out.println("hello");i<3;System.out.println("hi")){
    i++; // i = 1,2,3
}
```

Output:: hello hi hi hi

eg:: //by default condition is kept as true by compiler

```
for(;;){
    System.out.println("hello");
}
```

Output:: hello(infinite times)

eg::

```
for(;;)
    int x=10;
```

output:: CE

eg::

```
for(;;){
    int x=10;
}
```

output:: no output

eg::

```
for(int i=0;i<true;i++){
    System.out.println("sachin");
}
System.out.println("dhoni");
```

output::CE(I is integer and second operand after < is of type boolean)

eg::

```
for(int i=0;;i++){
    System.out.println("sachin");
}
System.out.println("dhoni");//unreachable
```

Output::CE

eg::

```
int a=10,b=20;
for(int i=0;a<b;i++){
    System.out.println("sachin");
}
System.out.println("dhoni");
```

output:: sachin(infinite)

eg::

```
final int a=10
final int b=20;
for(int i=0;a<b;i++){
    System.out.println("sachin");
}
System.out.println("dhoni");  //Unreachable code
```

output:: CE

Snippets —>

1)-

```
for (int i = 0; i <= 10; i++) {
    if (i > 6) break;
}
System.out.println(i);
```

Compilation fails.[Answer: i not accesible outside for loop]

2)-

```
public class Test {
    public static void main(String[] args) {
        boolean b1 = 0;
        boolean b2 = 1;
        System.out.println(b1 + b2);
    }
}
```

compilation error[Answer: boolean means only true,false]

3)-

Given:

```
public static void main(String[] args) {
    boolean i = true; //line-12
    switch(i) { //line-13
        case true: System.out.println("true"); break;
        case false: System.out.println("false"); break;
        default: System.out.println("other"); break;//line-15
    }
}
```

Compilation fails because of an error on line 13.[Answer: switch arg types can be : byte,short,int,char]

19)- Break Statement —>

a. Inside switch to avoid fallthrough

b. Inside loops to break the loop based on some condition

c. Inside label block to break label block execution based on some condition.

1)- break can be used along with blocks also.

```
int x= 10;
l1:{
    System.out.println("begin");
    if (x==10)
        break l1;
    System.out.println("end");
}
System.out.println("hello");
```

Output

begin

hello

2)- Note: break can be used inside "switch or loop", any other places if we try to use it results in "CompileTime-Error." (same for continue —> can only be used in loops)

```
int x= 10;
System.out.println("hello");
if (x==10)
break;
System.out.println("hiee");
```

20)-Note: Compiler won't check for unreachablity in case of "if-else" statement, whereas it checks for unreachability in case of loops.

eg#1.

```
while(true)
    System.out.println("hello");
System.out.println("hiee");//CE: unreachable statement
```

eg#2.

```
if (true)
    System.out.println("hello");//hello
else
    System.out.println("hiee");
```

21)- Operators —>

1)- Increment and Decrement —>

a)- int y = 10++; **//CE(increment or decrement can be applied only on variables, but not on values directly.)**

b)- we can't perform nesting of increment or decrement operator, it would result in compile time error.

```
int x= 10;
int y= ++(++x); //CE
System.out.println("x = " + x);
System.out.println("y = " + y);
```

c)- For a final variable, increment or decrement operation can't be done.

```
final int x= 4;
x++; //CE
System.out.println(x);
```

d)- we can't apply increment or decrement operator on boolean type, where as it can be applied on other primitive types.

```
int x= 10;
x++;
System.out.println(x);//11
char ch='a';
ch++;
System.out.println(ch);//b
double d = 10.5;
d++;
System.out.println(d);//11.5
boolean b= true;
b++; //CE
```

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

e)- What is the difference b/w b = b+1 and b++?

```
byte b= 10;
b = b+1; //CE (byte+int :: int)
System.out.println(b);
byte b = 10;
b++; // b = (byte)(b+1);
System.out.println(b);//11
```

Arithmetic operator —>

operators : +,-,*,/,%

i)- Note: if we apply arithmetic operators b/w 2 variables then the result type is always max(int,typeof a, typeof b)

eg:

byte + byte = int

byte + short = int

int + double = double

char + char = int

char + double = double

eg#1.

```
System.out.println('a' + 'b');  //195
System.out.println('a' + 1);  // 98
System.out.println('a' + 1.2);  //98.22. In integral arithmetic(byte , short , int , long) there is no way to represent "Infinity", so result will be "ArithmeticException".
```

ii)- In integral arithmetic(byte , short , int , long) there is no way to represent "Infinity", so result will be "ArithmeticException".

In case of double, float types, there is a possibility of storing "Infinity" so the result will be "Infinity".

```
System.out.println(10/0.0); // int/double = +double
System.out.println(-10/0.0); // -int/double = -double
System.out.println(0/0); // int/int = int
```

output

Infinity

-Infinity

ArithmeticException :/by zero

iii)-

a)- Nan(Not a Number) is a integral arithmetic(byte , short , int , long) there is no way to represent the undefined result, so it would throw an Exception called "ArithmeticException".

b)- But floating point arithmetic(double , float) there is a way to represent the undefined result, so the result would be "Nan".

```
System.out.println(0.0/0.0);   //double/double -> double
System.out.println(-0.0/0.0);  //double/double -> double
System.out.println(0/0);       //int/int -> int
```

Output

Nan

Nan

ArithmeticException : /by zero

Important Points —>

1)- 0.0/0.0 —>NAN, while 1.0/0.0 —> infinity

2)- Note: for any value of 'x' including NaN, the result will be false.

```
System.out.println(10<Float.NaN);//false
System.out.println(10<=Float.NaN);//false
System.out.println(10>Float.NaN);//false
System.out.println(10>=Float.NaN);//false
System.out.println(10==Float.NaN);//false
System.out.println(Float.NaN == Float.NaN);//false
System.out.println(10!=Float.NaN);//true
System.out.println(Float.NaN != Float.NaN);//true
```

3)- '+' operator in java is reffered as "Overloaded-Operator". '+' operator will perform addition if both the operands are of numeric type(byte , short , int , long , float , double) . '+' operator will perform concatenation, if one of the operand is of "String" type.

RelationalOperator in Java

<,<=,>,>=

=> We can apply relational operators only on primtive types except "boolean types".

=> we cannot apply relational operators on reference types(on objects)

=> Nesting of relational operator is not possible in java.

eg#1.

```
System.out.println(10>10.5);//false
System.out.println('a'>10.5);//true
System.out.println('z'>'a');//true
System.out.println(true>false);//CE
```

System.out.println("sachin">"kohli");**//CE**

System.out.println(10<20<30)**;//CE**

System.out.println(10>20>30)**;//CE**

EqualityOperator

==,!=

=> We can apply equality operators on primitive types including boolean types

eg#1.

System.out.println(10 == 20);//false

System.out.println('a' == 'b');//false

System.out.println('a' == 97.0);//true

System.out.println(false == false);//true

=> we can apply equality operators on reference type also.

eg#2.

Thread t1= new Thread();

Thread t2= new Thread();

Thread t3= t1;

System.out.println(t1==t2);//false

System.out.println(t1==t3);//true

Note::

=> To use == operator on reference type, we need to check whether there exists a relationship b/w 2 operands.

=> If relationship exists, it should be parent-child relationship, otherwise it would result in "CompileTimeError".

eg#3.

Thread t = new Thread();

Object o = new Object();

String s = new String("sachin");

StringBuffer sb =new StringBuffer("dhoni");

System.out.println(t==o);//false

System.out.println(o==s);//false

System.out.println(s==t); **//CE**

System.out.println(s==sb); **//CE**

Bitwise Operator

1)- &,|,^ => These operators can be applied on boolean and even on integral types.

2)- ~(bitwsise complement) => This operator can be applied on integral types, but not on boolean types.

3)- !(boolean complement) => This operator can be applied only on boolean types, but not on integral types.

Difference between &,| and &&,||(ShortCircuit Operator)

&,|

=> Both the arguments should be evaluated always

=> Performance is relatively slow.

=> It is applicable for both integral and boolean types.

&&,||

=> Second argument evaluation is optional.

=> Performance is relatively high

=> It is applicable only for boolean types, not for integral types.

Type casting operator

1. implicit/narrowing => compiler will automatically do(no loss of data)
2. explicit/widening => programmer should do(loss of data might happen)

=> if we work with floating point data and if we try to assign it to integral type using explicit typecasting then the data after decimal value will be lost.

eg#1.

float x = 150.1234f;

int i =(int) x;

System.out.println(i)**;//150**

double d = 130.456;

int j = (int)d;

System.out.println(j);**//130**

=> While working with integral types, storing higher value in lower type using explicit typecasting might lead to data loss.

eg#1.

int x = 150;

short s =(short) x;

System.out.println(s);//150

//minRange + (result-maxRange-1)

/*

=-128 + (150-127-1)

=-128 + 150-128

= -128+22

= -106

*/

byte b = (byte)x;

System.out.println(b);//-106

eg#2.

double d= 130.456;

int x= (int)d;

System.out.println(x);//130

//minRange + (result-maxRange-1)

/*

=-128 + (130-127-1)

=-128 + 130-128

= -128+2

= -126

*/

byte b = (byte)d;

System.out.println(b);//-126

Assignment Operator

There are 3 types of assignment operator

- **a. Simple assignment**
  - **eg:** int a= 10;
    int b= 20;
- **b. Chained assignment**
- **c. Compound assignment**

Chained assignment

eg#1

int a,b,c,d;

a= b= c= d= 10; **//valid**

eg#2.

int a=b=c=d=10; **//invalid**

System.out.println(a+" " + b + " "+c+" "+d);

eg#3.

int b,c,d;

int a=b=c=d=10; **//valid**

System.out.println(a+" " + b+" " + c +" " +d);

eg#4.

int a,b,c,d=10; // only d value is initialized

System.out.println(d);//10

Compound assignment

+= ,-=, /=, *=, %=

&=,|=,^=

Note: In case of compound assignment operator, internally type casting will be performed automatically by the JVM similar to increment or decrement operator.

eg#1.

int a= 10;

a+=20;

System.out.println(a);

eg#2.

byte b=10;

b = b+1; **//incompatible types: required : int,found : byte**

System.out.println(b);

eg#3.

byte b = 10;

b++; **// b= (byte)(b+1);**

System.out.println(b);//11

eg#4

byte b=10;

b+=1; **//b = (byte)(b+1);**

System.out.println(b);//11

eg#5.

int a,b,c,d;

a=b=c=d=20;

a+=b-=c*=d/=2;

/*

d= d/2; 20/2 = 10

c= c*d; 20*10 = 200*

b= b-c; 20-200 =-180

a= a+b; 20-180 =-160

*/

System.out.println(a+" " +b +" " +c +" " +d);

Conditional Operator

It is also called as "Ternary operator".

syntax: condition ? true : false

eg#1.

int x= 10==10 ? 100 : 500;

System.out.println(x); **//100**

eg#2.

final int a= 10,b= 20;

byte c1= (a > b) ? 30: 40**; // byte c = 40;**

eg#3.

int a1= 10,b1= 20;

byte c2= (a1 > b1) ? 30: 40; **//CE**

byte d2= (a1 < b1) ? 30: 40; **//CE**

System.out.println(c2 + " " + d2);

SNIPPET —>

1)- What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String[] args) {
        System.out.println(1 + 2 + 3 + 4 + "Hello"); // 10Hello
        System.out.println( "Hello" + 1 + 2 + 3 + 4); //Hello1234
    }
}
```

2)-

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Output is: " + 10 != 5);
    }
}
```

Ans —> compilation error

In Java, the + operator has a higher precedence than the != operator. Therefore, the expression is effectively treated as:

("Output is: " + 10) != 5

This leads to a compilation error because you are trying to compare a String with an int, which is not allowed.

To fix this issue, you need to use parentheses to explicitly define the order of operations:

"Output is: " + (10 != 5)

How many keywords are there in java?

Ans. 53

Reserve words meant for datatypes

a. byte,short,int,long
b. float,double
c. char
d. boolean

Reserve words for access modifiers(11)

a. public
b. private
c. protected
d. static
e. strictfp
f. synchornized
g. abstract
h. native
i. transient
j. volatile
k. final

What is Object?

It is an instance of a class.

What is class?

It is a blueprint as to how object should look upon creation.

Keywords related to object

a. new

How many ways are available to write a method in java program?

Ans. 4 ways

1. Method with no input and no output[void m1(){}]
2. Method with input and no output [void m1(XXXXX a, XXXX b){}]
3. method with input and output[XXXX m1(XXXX a, XXXX y){ return XXXX;}]
4. method without input and with output [XXXX m1(){ return XXXX;}]

What is the difference b/w Arguments and Parameter?

Ans. Arguments => When we are calling a method, if we pass data , those data we call as "Arguments".

Parameters => When we write a method body, the varaibles used to collect the value is called "Parameters".

What is the skeletal structure of a java program?

```
class .....
{
    public static void main(String[] args)
    {
        //method body
    }
}
```

Execution of Java Program(behind the scenes)

```
class Student
{
    //properties/fields/attributes
    String name;
    int age;
    //methods/behaviours
    public void dispStdDetails()
    {
        System.out.println(name);
        System.out.println(age);
    }
}
class Test
{
    public static void main(String[] args)
    {
        Student s = new Student();
        s.dispStdDetails();
```

```
    }
}
```

D:\OctBatchMicroservices>javac Test.java

Upon compilation of Test.java file 2 .class files will be loaded

a. Test.class[main()]

b. Student.class

D:\OctBatchMicroservices>java Test

JVM will load Test.class file by searching in the CWD[D:\OctBatchMicroservices>]

JVM found Test.class file which contains main() so JVM started the execution

JVM will split JRE into 3 regions

a. MethodArea[To keep .class file details]

b. StackArea[To keep the method body for execution]

c. HeapArea[To keep object data with default values for properties based on datatype]

Upon loading the main(), JVM started the execution

a. loaded main() body to StackArea.

b. Student s = new Student();

|=> JVM will go to heap area, create an object.

|=> Load Student.class file by searching in CWD[D:\ OctBatchMicroservices>] to MethodArea

|=> Depending upon the datatypes of properties, supply the default values by giving a memory inside heaparea.

|=> Address of the object will be returned to the user.

c. s.dispStdDetails()

|=> JVM will call the method

|=> Body of dispStdDetails() will be loaded into stack area

|=> Execution of statements present inside dispStdDetails will happen S.o.p(name); S.o.p(age);

|=> Print the details of name and age into console[monitor]

d. Remove stackarea of dispStdDetails()

e. Remove stackarea of main()

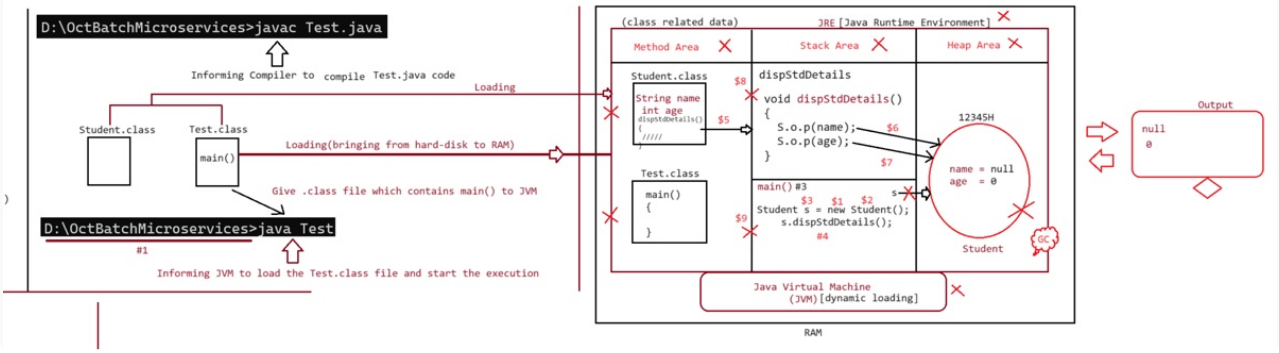
d. Since Student Object doesn't have refernce, it becomes garbage object[Call GarbageCollector[internally done by JVM]]

f. Unload the .class files from MethodArea[Test.class,Student.class]

g. Remove JRE /JVM from RAM.

Note: JVM by default will always search for main() with the following signature

public static void main(String[] args)



Arrays

1. Introduction to Arrays
2. Array Declaration
3. Array Construction
4. Array Initalization
5. Array declaration,construction,initalization in single line
5. length vs length() method
6. Annomynous array
7. Array element assignments
8. Array variable assignments

Need of Arrays : To resolve the problem of remembering multiple variable names.

Arrays

=> It refers to index collection of fixed no of homogenous[all elements should belong to same datatype] data elements.

=> Single variable holding mulitple values which imporves readability of the program.

Disadvantages

1. Once we create the size cannot be increased/decreased.
2. It stores only homogenous data elements.

Array declarations

Single Dimension Array
Declaration of array

int[] a;//recomended to use as variable is seperated from type.
int a[];
int []a;
int[6] a; // compile time error. we cannot specify the size.

Array Construction

Every array in java is an object hence we create using new operator.

example

int[] a;

a=new int[5];

or

int[] a =new int[5];

eg#1.

```
class Test
{
    public static void main(String[] args)
    {
        //Array declaration
        int[] a = null;
        System.out.println(a);//null
        //Array Construction
        a=new int[5];
        System.out.println(a);//[I
        System.out.println("Before initalization");
        for (int i =0;i<5 ;i++ )
        {
            System.out.println(a[i]);
        }
        //Array initalization
        a[0]=10;
        a[1]=20;
        a[2]=30;
        a[3]=40;
        a[4]=50;
        System.out.println("After initalization");
        for (int i =0;i<5 ;i++ )
        {
            System.out.println(a[i]);
        }
    }
}
```

Output

D:\OctBatchMicroservices>javac Test.java

D:\OctBatchMicroservices>java Test

null

[I@76ed5528

Before initalization

0

0

0

0

0

After initalization

10

20

30

40

50

Note::

For every type corresponding classes are available but these classes are part of java language but not applicable at the programmer level. **These classes are also called as "proxyClasses".**

int[] [I

float[] [F

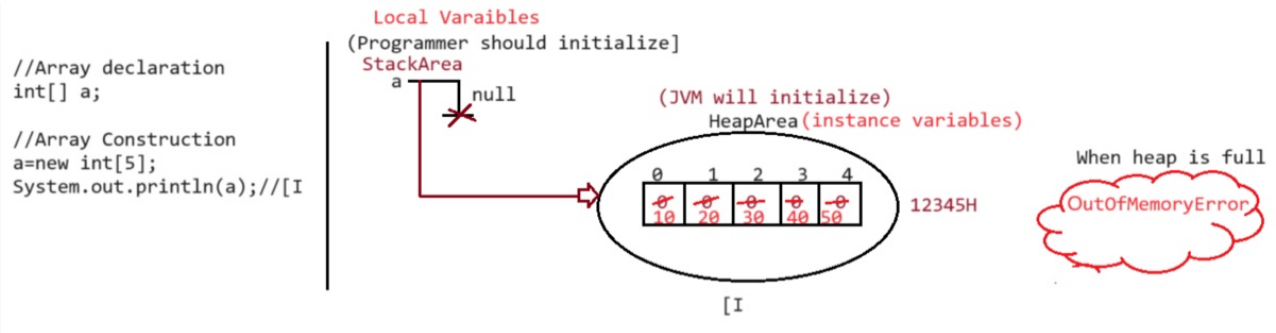
double[] [D

byte[] [B

boolean[] [Z]

short [S

String [java.lang.String...



Rule1::

At the time of Array construction compulsorily we should specify the size.

example::

```
int[] a=new int[5];
int[] a =new int[];//CE array dimension is missing.
```

Rule2

It is legal to have an array with size zero.

example

```
int[] a =new int[0];
System.out.println(a.length);
```

Rule3

If we declare an array with negative size it would result in Negative Array size exception.

example

```
int[] a=new int[-5]; //NegativeArraySizeException.
```

Rule4

The allowed atatypes to specify the size are byte,short,int,char.

example

```
int[] a =new int[5];
byte b=10;
int[] a =new int[b];//valid
short s=25;
int[] a =new int[s];//valid
char c='A';
int[] a=new int[c];//valid
int[] a=new int[10L];//CE
int[] a=new int[3.5f];//CE
```

Rule5

The maximum allowed array size in java is maximum value of int size.

```
int[] a=new int[2147483647]; //but valid:: OutOfMemoryError
int[] a=new int[2147483648]; //CE
```

Note: During the execution, if JVM is not able to create the memory for the objects due to insufficient memory space we call it as "OutofMemoryError".

During the execution, if JVM is not able to create a memory because of invalid size, we say such problems as "Exception".

ArrayInitialisation

Since arrays are treated as objects,internally based on the type of data we keep inside array JVM will keep default values.

eg::

int[] a =new int[5];

System.out.println(a);//[I@....

System.out.println(a[0]);//0

eg2::

int[] a=new int[4];

a[0]=10; a[1]=20; a[2]=30;

System.out.println(a[3]); //0

System.out.println(a[4]); //ArrayIndexOutOfBoundsException.

System.out.println(a[-4]);//ArrayIndexOutOfBoundsException.

Snippets

hint: Don't create object of Feline, directly call foo() because it is static.

```
class Feline {
    public static void main(String[] args) {
        Long x = 42L;
        Long y = 44L;
        System.out.print(" " + 7 + 2 + " "); // 72 foo425 86foo
        System.out.print(foo() + x + 5 + " ");
        System.out.println(x + y + foo());
    }
    static String foo() {
        return "foo";
    }
}
```

What is the result?

A. 9 foo47 86foo

B. 9 foo47 4244foo

C. 9 foo425 86foo

D. 9 foo425 4244foo

E. 72 foo47 86foo

F. 72 foo47 4244foo

G. 72 foo425 86foo//Answer

H. 72 foo425 4244foo

I. Compilation fails.

Q>

```
class Sixties {
    public static void main(String[] args) {
        int x = 5;
        int y = 7;
        System.out.print(((y * 2) % x));// 14%5 = 4
        System.out.print(" " + (y % x));// 7%5 = 2
    }
}
```

What is the result?

A. 1 1

B. 1 2

C. 2 1

D. 2 2

E. 4 1

F. 4 2 //Answer

G. Compilation fails.

H. An exception is thrown at runtime.

Q>

Given

int a= 8,b=15,c=4;

System.out.println(2 * ((a%5) * (4+(b-3)/(c+2))));

What is the output?

A. 30

B. 36//Answer

C. 32

D. 35

// 2 * ((8%5) * (4 + (15-3)/(4+2))))

2 * ((3) * (4 + (12)/(6))))

2 * ((3) * (4 + 2))

2 * ((3) * 6)

2 * (18)

36

Q>

Given

```
class TestOR {
2.  public static void main(String[] args) {
3.  if ((isItSmall(3)) || (isItSmall(7))) {
4.  System.out.println("Result is true");
5.  }
6.  if ((isItSmall(6)) || (isItSmall(9))) {
7.  System.out.println("Result is true");
8.  }
9.  }
11. public static boolean isItSmall(int i) {
12. if (i < 5) {
13. System.out.println("i < 5");
14. return true;
15. } else {
16. System.out.println("i >= 5");
17. return false;
18. }
19. }
20. }
```

What is the result?

A. Compilation Error at line 3

B. Compilation Error at line 6

C. i<5

Result is true

D. i<5

Result is true

i>=5

Result is true

E. i<5

Result is true

i>=5

i>=5

Result is true

F. i<5

Result is true

i>=5

i>=5

Objects —>

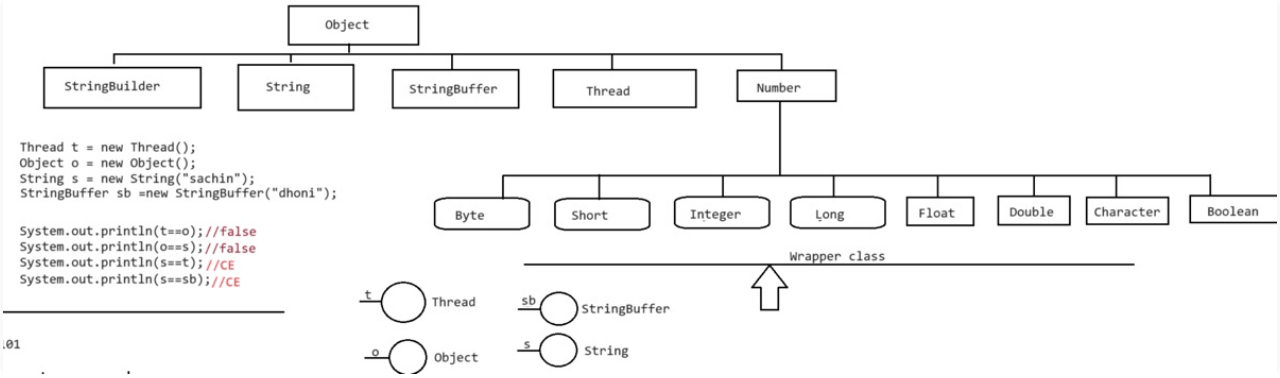
1)- new is an operator which is used to create an object/instance for a class in java.

2)- we don't have delete operator in java to destroy the objects, where destroying the objects is taken care by **"JVM(GarbageCollector)".**

++

22 —> snippets are not done as it includes & , |

++



Why we need Arrays?

To store mulitple values of same type in single varaible we need Arrays.

Some Important points —>

1. Arrays in java are treated as "Objects"
2. In java memory for objects is given in "HeapArea".
3. If memory is given in "HeapArea", jvm will give default value for the variables based on the datatype.
4. default value for boolean variable is "false".
5. **If memory is given in "StackArea", programmer should initialise the value before using the variable, otherwise it would result in "CompileTimeError".**
6. While working with Arrays in java, we should follow 3 steps
   a. Array Declaration
   b. Array Construction(new)
   c. Array Intialization
7. Any problem occured at runtime we categorise into 2 types
   a. Exception => Problem occurred and it can be handled.
   b. Error => Problem occured,but it can't be handled.

Example of error at runtime —>

The OutOfMemoryError is not thrown by the compiler but by the Java Virtual Machine (JVM) during runtime. It is a runtime exception that occurs when the Java application attempts to allocate more memory than is available in the Java heap.

Shortcut of way declartion,construction,initialisation in single line

int[]a = {10,20,30,40};

char[] a= {'a','e','i','o','u'};

String[] a= {"sachin","ramesh","tendulkar","IND"};

Array Element Assignements

case 1:: In case of primitive array as an array element any type is allowed which can be promoted to declared type.

eg#1.

int[] a=new int[10];

a[0]=97;

a[1]='a';

byte b= 10;

a[2]=b;

short s=25;

a[3]=s;

a[4]=10L; **//CE: possible loss of precession**

case 2:: In case of Object type array as an array elements we can provide either declared type object or its child class objects.

eg#1.

Object[] obj=new Object[5];

obj[0] =new Object();//valid

obj[1] =new Integer(10);//valid

obj[2] =new String("sachin");//valid

eg#2.

Number[] num=new Number[10];

num[0]=new Integer(10);//valid

num[1]=new Double(10.0);//valid

num[2]=new String("sachin")**;//invalid**

case3:: In case of interface type array as an array element we can provide its implementation class Object.

Runnable[] r=new Runnable[5];

r[0]= new Thread("sachin");

r[1]= new String("dhoni"); **//CE**

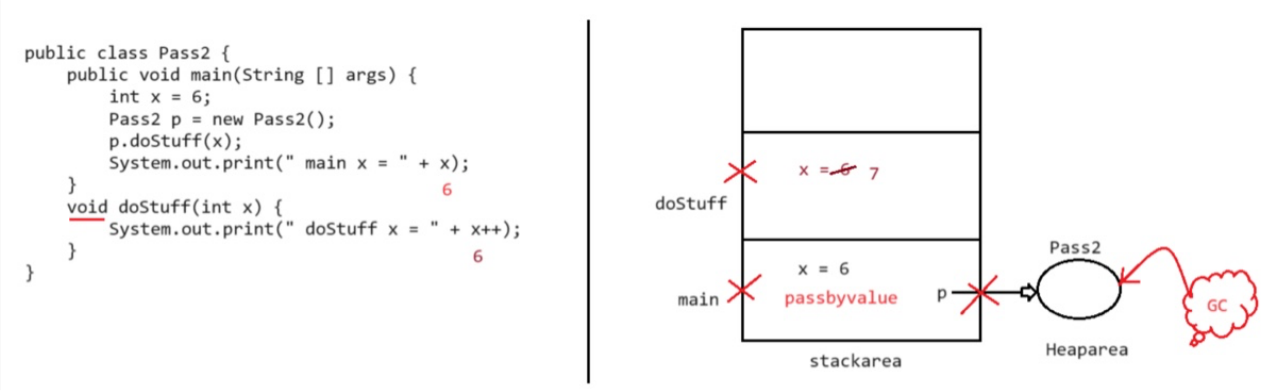
case4:: In case of abstract class type array as an array element we can provide its child class object.

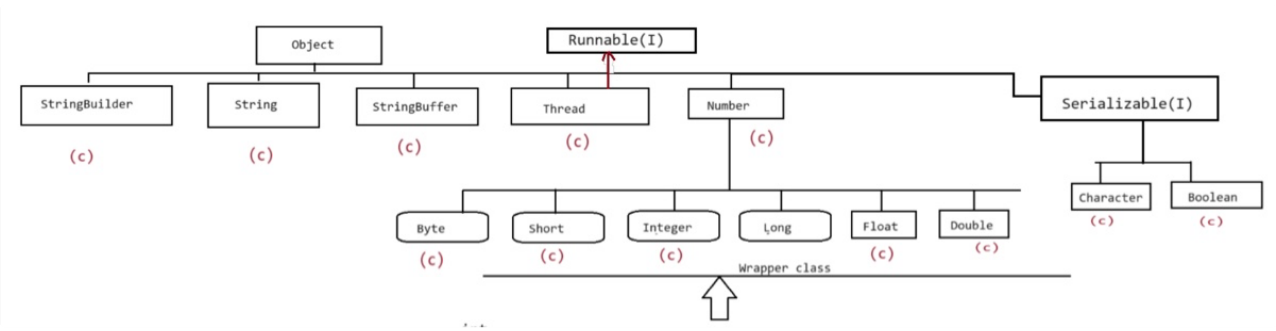
Key Differences between creating array with new keyword and without it

1)- When you use an array initializer without the new keyword, you need to provide the initial values at the time of declaration.

2)- When you use the new keyword, the array is dynamically created, and memory is allocated. The elements are initialized with default values (0 for numeric types, null for reference types).

3)- The size of an array created with the new keyword can be determined at runtime, whereas the size of an array created with an initializer is fixed at the time of declaration.





Array variable assignment

case1:: Element level type promotion is not applicable

eg:: char value can be type promoted to int, but char[] can't be type promoted to int[].

int[] a= {1,2,3};

char[] c={'a','b','c'};

int[] b = a;

int[] a = c; **//invalid**

which of the following promotions are valid?

char ----> int [valid]

char[] --> int[] [invalid]

int --> long [valid]

int[] ==> long[] [invalid]

double ==> float [valid]

double[]==> float[] [invalid]

String ==> Object [valid]

String[] => Object[] [valid]

case2:: In case of Object type array,its child type array can be assigned.

eg:: String[] names={"sachin","saurav","dhoni"};

Object[] obj=names;

case3:: Whenever we are assigning one array reference to another array reference,its just the reference which are being copied not the array elements.

While copying the reference only its type would be given importance,not its size.

eg:: int[] a= {10,20,30,40};

int[] b= {100,200};

a=b;

b=a;

case4:: Whenever we are copying the array,its reference will be copied but we should match it with the array dimension and its type,otherwise it would result in compile time error.

eg:: int[][] a= {{10},{20},{30}};

int[] b={100,200,300};

b= a; **//CE: incompatible type**

Note:: In array assignment,its type and dimension must be matched otherwise it would result in compile time error.

int[] a ={10,20,30};S.o.p(a);//[I@..

float[] f={10.0f,20.0f}; S.o.p(f); //[F@..

boolean[] b= {true,false}; S.o.p(b); //[Z@..

Integer[] i={10,20,30}; S.o.p(i);//[L@...

Float[] f = {10.0f,20.0f}; S.o.p(i); //[L@...

Working with 2D-Arrays

2D-Array = 1D-Array + 1D-Array
(ref) (data)

Declaration(All are valid)

int[][] a ;// recomended

int a[][];

int [][]a;

int[] []a;

int[] a[];

int []a[];

ArrayConstruction

int[][] a =new int[3][2];

or

int[][] a= new int[3][];

a[0]=new int[5];

a[1]=new int[3];

a[2]=new int[1];

Array Initalisation

a[0][0] = 10;

a[2][3] = 5;

Array Declaration

int[][] a= null; **//Since 'a' is a local variable so we need to assign a value to it otherwise CE while accessing it.**

Tricky Questions

a. int[] a,b; => a-1D, b-1D

b. int a[],b[]; => a-1D, b-1D

c. int a[],b; => a-1D, b-variable

d. int a,[]b; => CE

e. int []a,[]b; => CE

f. int []a,b; => a-1D,b-1D

g. int[] a,b; // a-1D, b-1D

h. int[] a[],b; // a-2D, b-1D

i. int[] []a,b; // a-2D, b-2D

j. int[] a, []b; //CE

Note: if we want to specify the dimension before the variable, that rule is applicable only for first varaible,second varaible onwards we can't apply in the same declaration.

Q>

int[][] a =new int[3][2];

a[0] = new int[3];

a[1] = new int[4];

a = new int[4][3];

Totally how many objects are created?

Ans. 11

How many objects are eligible for Garbage Collection?

Ans. 6

ShortCut to create a 2d array —>

```
int[][] b= {
    {10,20,30,40},
    {50,60}
};
```

Note: if an object does'nt have reference to access, then such objects are called as "Garbage" Object.

Collecting inputs from the User

1. We use Scanner class which is present inside "java.util" folder
2. To read integer inputs we use a method called "nextInt()".

Scanner —>

```
import java.util.Scanner;
int size=0;
System.out.print("Enter the size of the array:");
Scanner scan= new Scanner(System.in);
size = scan.nextInt();
```

Working with foreach loops

1. What is the need for foreach loop when we already have forloop?

Ans. To work with for loop as a programmer we need to do the following things

a. initalization

b. putting condition

c. performing inc/decr depending on initalisation

d. Access the element or put the element into array through index

Solution : foreach loop(just read the element, don't worry about initalization,condition and incr/dec)

syntax:

```
for(datatype variable:iterating-object)
```

```
{
    //Access the variable directly
}
int[] arr = new int[size];
for(int data : arr)
{
    System.out.print(data + "\t");
}
int[][] arr = {{10,20,30},{40,50},{60}};
for(int[] oneDArr:arr)
{
    for ( int data: oneDArr )
    {
        System.out.print(data + "\t");
    }
    System.out.println();
}
```

What is the difference b/w for loop and foreach loop?

forloop

=> meant for both read and write purpose.

=> it might lead to ArrayIndexOutOfBoundsException(AIOBE), if we dont' use it properly.

=> by placing the condition we can iterate from R to L and L to R.

=> Accessing of elements is through index only.

foreach loop

=> meant for only read purpose

=> The problem of exception won't occur.

=> iteration is possible only from L to R.

=> Accessing of elements is through "datatype" of the variable.

++

not clear

Q>

```
int[][] a =new int[3][2];
a[0] = new int[3];
a[1] = new int[4];
```

a = new int[4][3];

Totally how many objects are created?

Ans. 11

How many objects are eligible for Garbage Collection?

Ans. 6

++

Access modifiers in java

1. private
2. public
3. protected
4. static[if we mark method as static, then that method can be called without creating the object]
5. strictfp
6. synchronized
7. final
8. abstract
9. native
10. transient
11. volatile

length property vs length() method

length: It is property which belongs to Arrays.

=> public final int length;

=> This property would return the no of elements present inside the array.

eg#1.

int[] arr = {10,20,30};

System.out.println(arr);//[I@...

System.out.println(arr.length);//3

System.out.println(arr.length());//CE

eg#2.

int[][] arr = new int[6][3];

System.out.println(arr.length);//6

System.out.println(arr[0].length);//3

length() : It is available inside "String" class in java.

=> public int length();

=> This method would return the no of characters present in the given String.

eg#1.

String name = "sachin";

System.out.println(name);//sachin

System.out.println(name.length());//6

System.out.println(name.length); **//CE**

Ananomyous Array

=> An array without a name is called Ananomyous Array.

=> These type of array is created just for instance use.

=> Creation of Ananomyous Array

new int{10,20,30,40};

new String{"sachin","kohli","dhoni"};

new int[]{{10,20,30},{40,50},{60}}

eg#1.

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("The sum is :: "+sum(new int{10,20,30,40,50}));
    }
    static int sum(int[] arr)
    {
        int total = 0;
        for(int data : arr)
        {
            total+=data;
        }
        return total;
    }
}
```

CommandLineArguments

=> These are the arguments passed in the command line from the programmer to the main().

=> Any type of arguments can be passed from the command line like int,float,char,double,String.....

=> JVM will collect the arguments passed by the programmer and creates an Ananomyous array of type String.

=> JVM will call main() by passing Ananmyous array as the argument.

=> Signature of main()

public => jvm should access the main() without any authorization and authentication so make it as public.

static => jvm should not create an object of the class which contains main(), it should directly call main,so mark main() as static

void => jvm will not return anything to o.s so we need to mark the return type as void

main(String[] args) => String[] args :: it refers to command line arguments which the jvm will use to store the arguments sent by the user.

eg:: java Test 54.5 true 10

|

Test.main(new String{"54.5","true","10"})

```
public static void main(String[] args)
{
    String[] argh= {"A","B"};
    args = argh; //args = {"A","B"}
}
```

Write a java program to peform addition of 2 numbers by taking inputs from command line?

```
class Test
{
    //Pre-Defined Method[Entry point/Driving Code]
    public static void main(String[] args)
    {
        System.out.println(args);
        System.out.println("The length of command line arguments is "+args.length);
        int firstOperand = Integer.parseInt(args[0]);
        int secondOperand = Integer.parseInt(args[1]);
        int result = firstOperand + secondOperand;
        System.out.println("The sum is :: "+result);
    }
}
```

Output

D:\OctBatchMicroservices>javac Test.java

D:\OctBatchMicroservices>java Test 10 20

[Ljava.lang.String;@76ed5528

The length of command line arguments is 2

The sum is :: 30

D:\OctBatchMicroservices>java Test 100 200

[Ljava.lang.String;@76ed5528

The length of command line arguments is 2

The sum is :: 300

D:\OctBatchMicroservices>java Test 1000 2000

[Ljava.lang.String;@76ed5528

The length of command line arguments is 2

The sum is :: 3000

Types of Variables

What is variable?

=> It is an identifier or name given to memory location which holds our data.

How many type of variables are there in java language?

There are 2 types of variables

a. Based on the type of value variable holds

b. Based on the behaviour and position of its declaration

Based on the type of value variable holds

-> We have 2 types

a. primitive type

b. reference type

Primitive type

=> These are the variables which holds primitive values

eg: int x= 10; boolean isMarried = false; double avg=53.5;

Reference type

=> These are the variables which holds the address of the objects, or these varaible are used to refer the objects

eg: Student std = new Student();

Employee emp = new Employee();

Based on the behaviour and position of its declaration

-> we have 3 types

a. instance variable

b. static variable

c. local variable

a. instance variable

1)- These are varaibles which are written inside the class,but outside the method.

2)- instance variables are such variables whose value changes from object to object.

3)- instance variables are created at the time of object creation and destroyed at the time of object destruction(GC)

4)- instance variables will be stored in the heaparea of the object[seperate copy for each object].

5)- instance varaibles can be accessed directly from instance area in instance method.

6)- instatnce variables are accessed through reference from static area in static method.

eg#1.

```
class Test
{
    boolean isMarried;
    //Pre-Defined Method[Entry point/Driving Code]
    public static void main(String[] args)
    {
        System.out.println(isMarried);  //CE
        System.out.println(new Test().isMarried);//false
    }
}
```

eg#2.

```
class Test
{
    int i = 100;
    //Pre-Defined Method[Entry point/Driving Code]
    public static void main(String[] args)
    {
        System.out.println(i);//CE
        new Test().disp();
    }
    public void disp()
    {
        System.out.println(i);//100
    }
}
```

eg#3.

```
class Test
```

```
{
    //Array declaration
    int[] arr;
    //Pre-Defined Method[Entry point/Driving Code]
    public static void main(String[] args)
    {
        System.out.println(new Test().arr);//null
        System.out.println(new Test().arr[0]);  //NPE
    }
}
```

static variable

1)- These are varaibles which are written inside the class,but outside the method with an access modifer called "static".

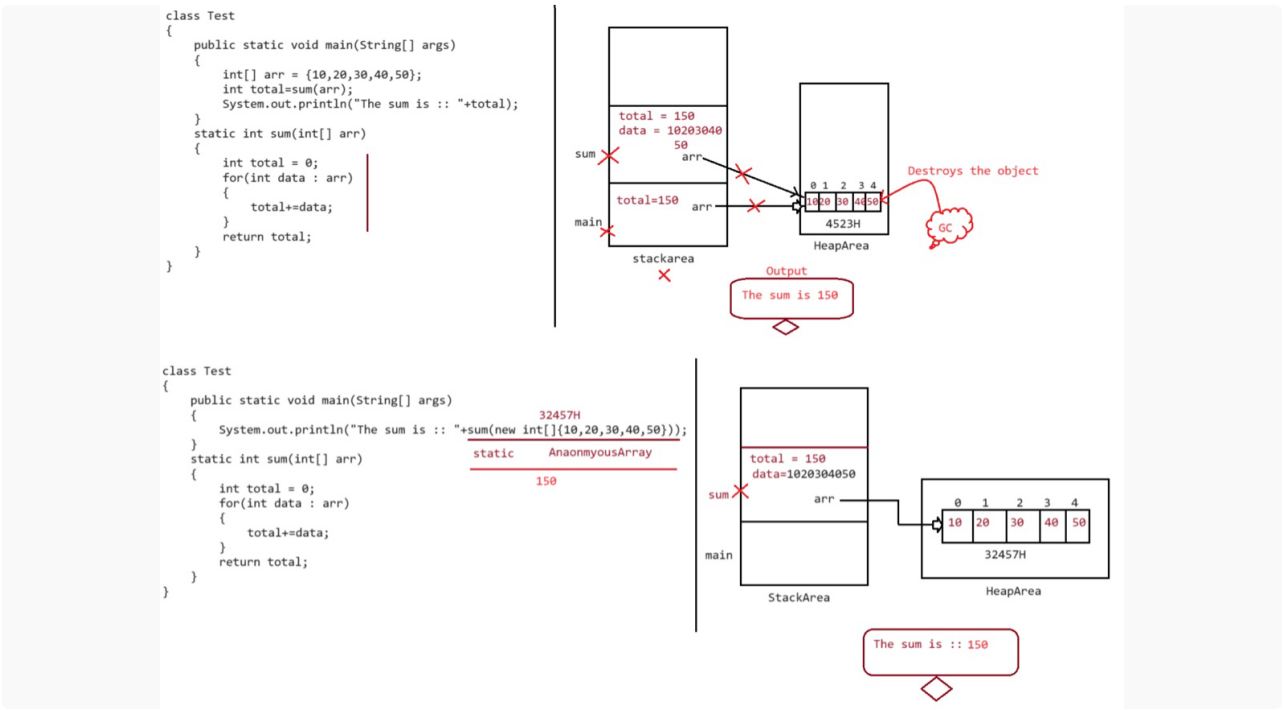
2)- static variables are such variables whose value does not changes from object to object[unique copy].

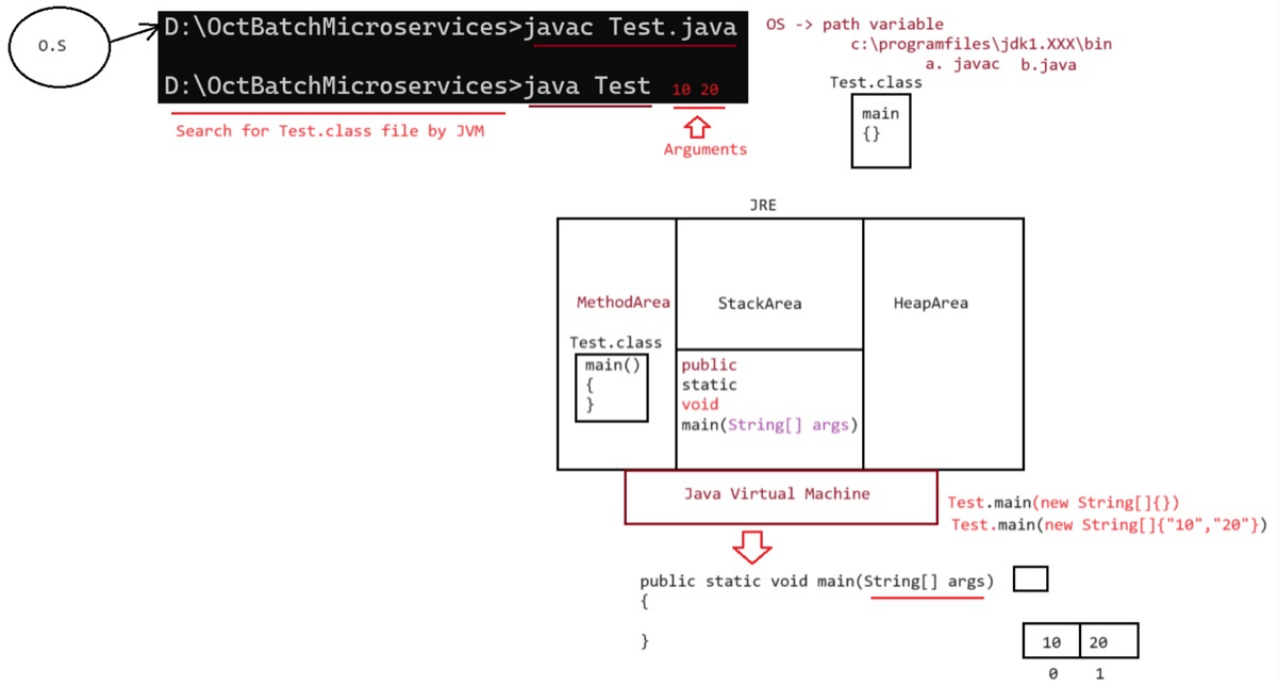
3)- static variables are created at the time of loading the .class file and destroyed at the time of unloading the .class file.

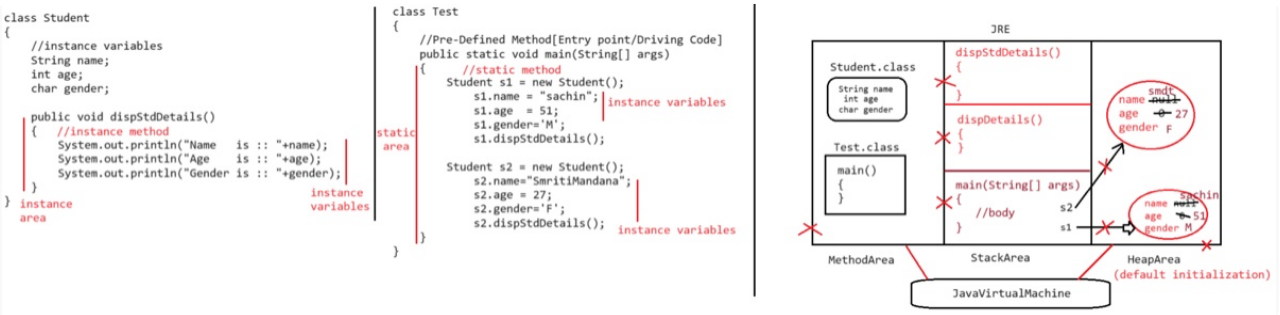
4)- static variables will be stored in the methodArea[Common copy for all the objects of the class].

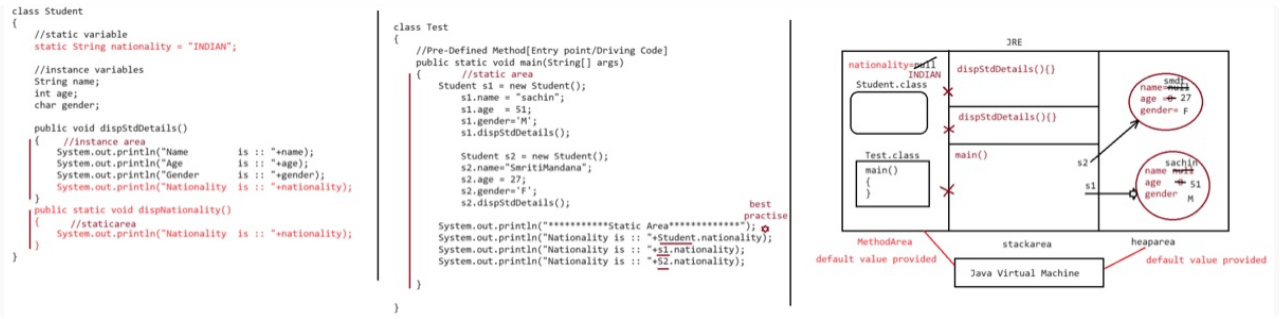
5)- static varaibles can be accessed directly inside instance or static area.

6)- static variables should be accessed using classname or object name,but good practise is through "classname".



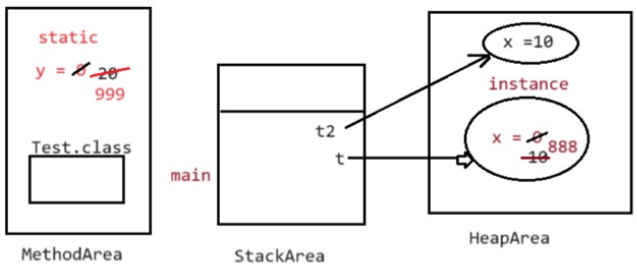




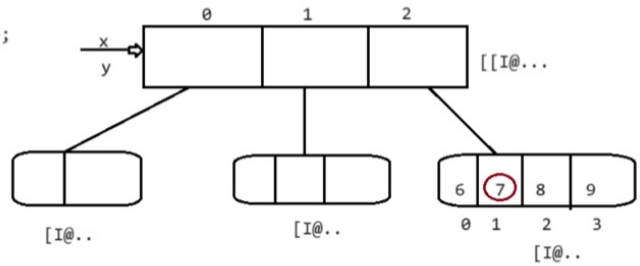


```
class Test{
    int x =10;
    static int y= 20;
    public static void main(String[] args)
    {
        Test t = new Test();
        t.x = 888;
        t.y = 999;
        Test t2 =new Test();
        System.out.println(t.x + " "  + t.y);
        System.out.println(t2.x + " " + t2.y);
    }
}
```

```
t  object -> 888 999
t2 object -> 10  999
```




```
int[] x[] = { { 1, 2 }, { 3, 4, 5 }, { 6, 7, 8, 9 } };
   int[][] y = x;
   System.out.println(y[2][1]);
```



Variables

a. Based on the type of value it holds

1. primitive
2. instance

b. Based on the behaviour and its position of declaration

1. instance variables.[jvm will give default value]
2. static variables. [jvm will give default value]
3. local variables. [jvm will not give, programmer should initialize before using]

In Java, when a class is loaded into memory, static variables are initialized with default values even if you do not explicitly assign values to them. The default values for static variables in Java depend on their data types. Here are the default values for common data types:

Numeric Types:

byte, short, int, long: **0**

float, double: **0.0**

Char Type:

char: **'\u0000' (null character)**

Boolean Type:

boolean: false

Reference Types:

Any reference type (object, array, etc.)**: null**

Based on the behaviour and its position of declaration

1. instance variables.[jvm will give default value]
2. static variables. [jvm will give default value]
3. local variables. [jvm will not give, programmer should initialise before using]

local variables

=> Sometimes to meet programming requirement the developer will declare variables inside a method or a block,such type of variables are called as "Local Variables".

=> These variables are also called as "Temporary variables/stack variables".

=> Local variables will be created during the method execution inside stack and they will be destoryed once the control comes out of method execution.

=> Scope of local varible is limited only to that method or to that block, if we try to access them outside the block or method it would result in "CompileTime Error".

Note: For local variables jvm will not give any default value, we need to initalise the variable value depending on the type before

intializing it, otherwise it would result in "CompileTimeError".

eg#3.

```
class Test
{
    //Pre-Defined Method[Entry point/Driving Code]
    public static void main(String[] args)
    {
        int x;
        System.out.println("hello :: "+x);   //CE
    }
}
```

Note:

1)- It is not a convention to initialize the local variable inside the block, because there is no gaurantee that a block will be executed and it will be initialized at runtime.

2)- It is suggestible to initialize the local variable at the time of declaration itself with a default value depending upon the datatype.

eg#1.

```
class Test
{
    //Pre-Defined Method[Entry point/Driving Code]
    public static void main(String[] args)
    {
        int x ;
```

```
        if(args.length > 0 )
            x = 10;
        System.out.println(x);//CE
    }
}
```

eg#2.

```
class Test
{
    //Pre-Defined Method[Entry point/Driving Code]
    public static void main(String[] args)
    {
        int x ;
        if(args.length > 0 )
            x = 10;
        else
            x = 20;
        System.out.println(x);
    }
}
```

javac Test.java

java Test 100

x = 10

java Test

x= 20

How many access modifers are there in java?

public, private, protected

static, strictfp, synchronized

final, abstract, native

transient, volatile

Note: The only access modifier which is applicable at the local variable level is "final",if we try to use any other access modifier it would result in "CompiletimeError".

1)- class Test

```
{
    //instance variale
    int[] a =new int[3];     // a ---> [0]=0 [1]=0 [2]=0
```

```
    static int[] b =new int[3];   // a ---> [0]=0 [1]=0 [2]=0
    //Pre-Defined Method[Entry point/Driving Code]
    public static void main(String[] args)
    {
        Test t= new Test();
        System.out.println(t.a);  //[I@...
        System.out.println(t.a[0]);  //0
    }
}
```

2)- class Test

```
{
    //Pre-Defined Method[Entry point/Driving Code]
    public static void main(String[] args)
    {
        int[] a;
        System.out.println(a);   //CE
        System.out.println(a[0]);
    }
}
```

3)- class Test

```
{
    //Pre-Defined Method[Entry point/Driving Code]
    public static void main(String[] args)
    {
        int[] a =new int[3]; //(new means object will be created and it will be in heap area so default values will be given as per datatype.)
        System.out.println(a);  //[I@...
        System.out.println(a[0]);//0
    }
}
```

Note: combination of varibles can be of the following types

- **a. instance** -> primitive, reference
- **b. static** -> primitive, reference
- **c. local** -> primitive, reference

```
class Test
{
    int[] a =new int[3]; //instance-reference
```

```
    static int x= 100; //static-primitive
    //Pre-Defined Method[Entry point/Driving Code]
    public static void main(String[] args)
    {
        String name = "sachin";  //local-reference
    }
}
```

Polymorphism

1. **Overloading**

   => Two or more methods are said to overloaded methods iff they have the same methodname but change in the argument types.

   => In c language we did'nt had this Overloading concept so as a C programer for same tasks with different argument types, programmer should remembe mulitple method names like

   a. abs() -> for int type

   b. labs() -> for long type

   c. fabs() -> for float type

   ;;;

   => Remembering mulitple method names was challenging for developers.

   => In java we have concept of Overloading, where we can write one method name for same tasks with different argument types.

   => overloading concept reduces the complexity of programming in java.

   => Overloading refers to "CompileTime Polymorphism".

eg#1.

```
class Calculator
{
    public void add(int a, int b)
    {
        System.out.println(a+b);
    }
    public void add(int a,int b, int c)
    {
        System.out.println(a+b+c);
    }
    public void add(double a, double b)
    {
        System.out.println(a+b);
    }
```

```
    public void add(float a, float b)
    {
        System.out.println(a+b);
    }
}
class Test
{
    public static void main(String[] args)
    {
        Calculator c = new Calculator();
        c.add(10,20);
        c.add(10,20,30);
        c.add(20.5,30.5);
        c.add(20.5f,30.5f);
    }
}
```

Output

D:\OctBatchMicroservices>javac Test.java

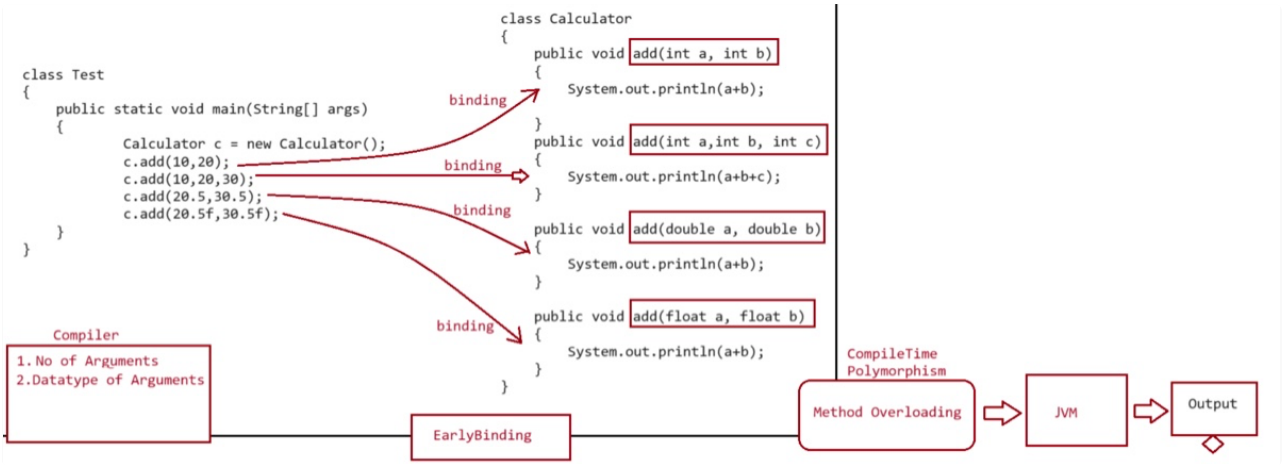
D:\OctBatchMicroservices>java Test

30

60

51.0

51.0



Conclusion: In overloading compiler is responsible for method resolution(binding) based on the reference type, so we say overloading as "CompileTime Polymorphism/Eager Binding".

eg#2.

```
class Test
{
    //overloaded method
    public void methodOne(){
        System.out.println("NO arg method");
    }
    public void methodOne(int i){
        System.out.println("Int arg method");
    }
    public void methodOne(double d){
        System.out.println("double arg method");
    }
    public static void main(String[] args)
    {
        Test t= new Test();
        t.methodOne();
        t.methodOne(10);
        t.methodOne(25.5);
    }
}
```

Output

D:\OctBatchMicroservices>javac Test.java

D:\OctBatchMicroservices>java Test

NO arg method

Int arg method

double arg method

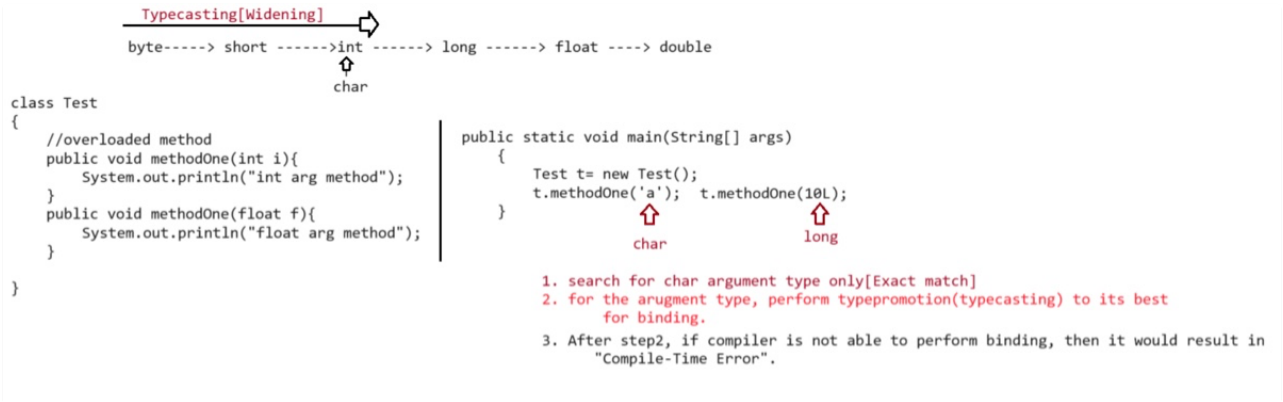
TypePromotion in Overloading

```
public class Test
{
    //overloaded method
    public void methodOne(int i){
        System.out.println("int arg method");
    }
    public void methodOne(float f){
        System.out.println("float arg method");
```

```
    }
    public static void main(String[] args)
    {
        Test t= new Test();
        t.methodOne('a');//int arg method
        t.methodOne(10L);//float arg method
        t.methodOne(19.5);//CE
    }
}
```

eg#2.

```
class Test
{
    //Overloaded method
    public void methodOne(int i)
    {
        System.out.println("int arg method");
    }
    public void methodOne(Integer i)
    {
        System.out.println("Integer version");
    }
    public void methodOne(Character c)
    {
        System.out.println("Character version");
    }
    public static void main(String[] args)
    {
        Test t= new Test();
        t.methodOne('a');
    }
}
```



// primtiive -----> Exact Match(primitive) -----> typecasting -----> bind

// primitive ------> no exact match -> no typecasting -----> Wrapper classes ---→ bind

eg#3.

```
class Test
{
    //Overloaded method
    public void methodOne(String s)
    {
        System.out.println("String version");
    }
    public void methodOne(StringBuilder sb)
    {
        System.out.println("StringBuilder version");
    }
    public void methodOne(Object o)
    {
        System.out.println("Object version");
    }
    public static void main(String[] args)
    {
        Test t= new Test();
        t.methodOne("sachin");//String version
        t.methodOne(new Object());//Object version
        t.methodOne(new StringBuilder("sachin"));//StringBuilder version
        //StringBuilder(child) -> null String(child) -> null , Object(Parent) --> null
        t.methodOne(null);
    }
}
```

Note: While resolving Overloaded methods exact match will get high priority. While resolving Overloaded methods,child class will get more priority than the parent class. While resolving Overloaded methods, if both the class are from child level only, then it would result in "Compiletime Error".

Method Overloading

=> Two or more methods is said to be overloaded method, iff both the methods have same name,but change in the argument datatype.

=> Order of binding the method calls would be based on

a. exact match

b. type promotion

=> In case of reference type, first priority is for child and then for parent.

=> In case of Overloading,binding the method call will be done by the compiler based on the reference type.

=> It is also called as "CompileTime Polymorphism/Early Binding".

eg#1.

```
class Test
{
    public void m1(int i,float f)
    {
        System.out.println("int-float arg method");
    }
    public void m1(float f,int i)
    {
        System.out.println("float-int arg method");
    }
    public static void main(String[] args)
    {
        Test t= new Test();
        t.m1(10,10.5f);//int-float
        t.m1(10.5f,10);//float-int
        t.m1(10,10); //CE: ambigous
        t.m1(10.5f,10.5f);//CE: can't find symbol
    }
}
```

Var-Args(Variable no of arguments method)

-> Untill 1.4V we can't declare a method with variable no of arguments

-> if there is a change in no of arugments compulsorily we have to define a new method

-> This approach increases the length of the code and reduces the readability.

-> From 1.5V version onwards we can declare a method with variable no of arugmentssuch type of methods are calle d as "VAR-ARGS METHODS".

-> Syntax:

```
public XXXXX methodName(XXXX... varaible)
{
}
```

eg#1.

```
class Test{
    public void m1(int... data)
    {
        System.out.println("Var-Arg method");
    }
    public static void main(String[] args) {
        Test t= new Test();
        t.m1();
        t.m1(10);
        t.m1(10,20);
        t.m1(10,20,30);
        t.m1(10,20,30,40);
    }
}
```

Note: Internally var-arg parameter is implemented by using "1-D Array",so var-arg parameter can be accessed through index.

Case1:

Which of the following var-arg declarations are valid?

methodOne(int... x) //valid [recomended]

methodOne(int ...x) //valid

methodOne(int...x) //valid

methodOne(int x...) //invalid

methodOne(int. ..x) //invalid

methodOne(int .x..) //invalid

Case2: we can mix var-args with normal parmeters also, and normal parameter can be different type and var-arg can be different type.

eg:: m1(int a, int... arr)

m1(String name, int... arr)

```
class Test{
    public void m1(String name, int... arr){
        System.out.println(name);
        System.out.println(arr);
    }
    public static void main(String[] args) {
        Test t= new Test();
        t.m1("sachin",20,30,40);
    }
}
```

Output

D:\OctBatchMicroservices>javac Test.java

D:\OctBatchMicroservices>java Test

sachin

[I@76ed5528

Case3: We can mix var-arg parameter with normal parameter,but in the parameter list the var-arg parameter should be at the last.

```
class Test{
    public void m1(int... arr, int data ){
        System.out.println(data);
        System.out.println(arr);
    }
    public static void main(String[] args) {
        Test t= new Test();
        t.m1(10,20,30,40);
    }
}
```

Output

D:\OctBatchMicroservices>javac Test.java

Test.java:2: error: varargs parameter must be the last parameter

public void m1(int... arr, int data){

Case4: In a paramter list,we can have only var-arg parameters,more than one results in "CompiletimeError".

```
class Test{
    public void m1(int... arr1, int... arr2 ){
        System.out.println(arr1);
        System.out.println(arr2);
    }
    public static void main(String[] args) {
        Test t= new Test();
        t.m1(10,20,30,40);
    }
}
```

Output

D:\OctBatchMicroservices>javac Test.java

Test.java:2: error: varargs parameter must be the last parameter

public void m1(int... arr1, int... arr2){

Case5: In general var-arg method will get least priority that is if no other methods are available to bind only then var-arg method will get a chance for binding. This is just like "default" statement in switch case.

```
class Test{
    public void m1(int... arr ){// arr-> 0,1,.... n
        System.out.println("var-arg method");
    }
    public void m1(int i){// i -> 1
        System.out.println("one-arg method");
    }
    public static void main(String[] args) {
        Test t= new Test();
        t.m1(10,20,30,40);
        t.m1(10);
        t.m1();
    }
}
```

output

D:\OctBatchMicroservices>javac Test.java

D:\OctBatchMicroservices>java Test

var-arg method

one-arg method

var-arg method

Case6: For the var-args method we can provide the corresponding type array as argument

```
class Test{
    public void m1(int... arr ){//int[] :: arr-> 0,1,.... n
        System.out.println("var-arg method");
    }
    public static void main(String[] args) {
        Test t= new Test();
        t.m1(10,20,30,40);
        t.m1(new int{100,200,300,400});
    }
}
```

output

D:\OctBatchMicroservices>javac Test.java

D:\OctBatchMicroservices>java Test

var-arg method

var-arg method

Case7:

```
class Test{
    public void m1(int... arr ){//m1(int[]) :: arr-> 0,1,.... n
        System.out.println("Var-arg method");
    }
    public void m1(int[] arr) //m1(int[])
    {
        System.out.println("Array-arg method");
    }
    public static void main(String[] args) {
        Test t= new Test();
        t.m1(10,20,30,40);
        t.m1(new int{100,200,300,400});
    }
}
```

output

D:\OctBatchMicroservices>javac Test.java

Test.java:5: error: cannot declare both m1(int[]) and m1(int...) in Test

public void m1(int[] arr)//m1(int[])

^

1 error

Case 8: wherever there is [] array,we can replace it with "..." also to provide the arguments in flexible manner(var-args,arrays)

Case 9: Wherever there is ..., if we repalce it "[]", we don't get flexibility to provide the arguments in var-args and arrays style.

eg::

```
class Test{
    public void m1(int... arr ){// arr-> 0,1,.... n
        System.out.println("Var-arg method");
    }
    public static void main(String[] args) {
        Test t= new Test();
        t.m1(10,20,30,40);
        t.m1(new int{100,200,300,400});
    }
}
```

output

D:\OctBatchMicroservices>javac Test.java

D:\OctBatchMicroservices>java Test

Array-arg method

Array-arg method

eg::

```
class Test{
    public void m1(int[] arr ){// arr-> 0,1,.... n
        System.out.println("Array-arg method");
    }
    public static void main(String[] args) {
        Test t= new Test();
        t.m1(10,20,30,40);
        t.m1(new int{100,200,300,400});
    }
}
```

D:\OctBatchMicroservices>javac Test.java

Test.java:7: error: method m1 in class Test cannot be applied to given types;

t.m1(10,20,30,40);

eg::

```
class Test{
    public static void main(String... args) {
        System.out.println("Var-Arg main method");
    }
}
```

output

D:\OctBatchMicroservices>javac Test.java

D:\OctBatchMicroservices>java Test

Var-Arg main method

Case 8:

```
class Test{
    public void m1(int[]... twoDarr){
        System.out.println(twoDarr);
        for (int[] oneDArr: twoDarr )
        {
            for (int data: oneDArr )
            {
                System.out.print(data+"\t");
            }
            System.out.println();
        }
    }
    public static void main(String... args) {
        Test t =new Test();
        int[] arr1 = {10,20,30};
        int[] arr2 = new int{100,200,300};
        t.m1(arr1,arr2);
    }
}
```

//m1(int... x) ===> arguments var-args,array ---> x []

//m1(int[]... x) ===> arguments 1D-array var-args ----> x [][]

Output

D:\OctBatchMicroservices>javac Test.java

D:\OctBatchMicroservices>java Test

[[I@76ed5528

10 20 30

100 200 300

Rule1: Binding of method call will happen based on the reference, not on the runtime object

```
class Animal{}
class Monkey extends Animal{}
class Test{
    public void talk(Monkey m){
        System.out.println("Monkey version");
    }
    public void talk(Animal a){
        System.out.println("Animal version");
    }
    public static void main(String... args) {
        Test t = new Test();
        Animal a =new Animal();
        t.talk(a);//Animal version

        Monkey m =new Monkey();
        t.talk(m);//Monkey version

        Animal a1= new Monkey();// a1 -> Animal type(compiler will bind) a1→ Monkey(runtime object::JVM)
        t.talk(a1);  //Animal version
    }
}
```

Output

D:\OctBatchMicroservices>javac Test.java

D:\OctBatchMicroservices>java Test

Animal version

Monkey version

Animal version

Try it yourself

What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String [] args) {
        int a = 3;// a = 3,4,5
        m(++a, a++); // m(4,4) //pass by value
        System.out.println(a);//5
    }
    private static void m(int i, int j) {
        i++;
        j--;
    }
}
```

A. 4

B. 5

C. 6

D. 3

Answer: B

Constructors

=> A constructor is a method whose name is same as that of the classname

=> A constructor would not have a return type.

=> Constructors gets called during the creation of an object

=> Constructors are normally used to give meaningful value to the instance variables of the class.

Note: In a class, if we don't write any constructor only then compiler will add default consturctor to our class.

=> Default constructor would not supply any meaningful values to the instance variables of the class.

=> To supply meaningful values to the instance variables, we need to write "Parameterized constructor".

Can a Constructor be Overloaded?

Ans. Yes, it is possible to Overload a constructor,but it is not a good practise to write zero argument constructor(as we will be using hard coded values for each new object) with a logic of "initialization".

Can we have normal method with the name same as classname and also constructor?

Ans. Yes, it is possible, but the constructor will be called during the creation of object where as normal method should be called by the programmer explicitly.

Eg:

```
//Parameterized constructor
Student(String name, int age, float height)
{
    System.out.println("CALLING THE CONSTRUCTOR");
    this.name = name;
    this.age = age;
    this.height = height;
}
//Normal method
void Student(String name,int age, float height)
{
    System.out.println("CALLING THE METHOD");
    this.name = name;
    this.age = age;
    this.height = height;
}
```

Can we Overload main()?

Ans. yes we can overload main(),but jvm will always call main() with the following signature

public static void main(String[] args).

eg#1.

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("Inside String[] args");
    }
    public static void main(int arg)
    {
        System.out.println("Inside int arg");
    }
    public static void main()
    {
        System.out.println("Inside zero argument");
    }
}
```

output

Inside String[] args

static

=> This access modifier is appicable at

a. class

b. variable

c. method

d. block

variable :

1)- if we mark a variable as static, then those variables are recognized as "class level variables".

2)- static variables are unique with respect to class, they are not w.r.t Object.

3)- memory for static variables will be give in "MethodArea".

4)- if we don't initialize the static variables, then memory for static variables will be taken care by "JVM".

method :

if we mark a method as static, then those methods can be called in 2 ways.

a. using ClassName(best practise)

b. using objectName

block :

1)- we can mark a block with static access modifier.

2)- This block is mainly meant for "initializing the static variables".

3)- This block will be executed only once, so we normally keep "Driving code" in static block.

eg#1. --> Important program to know the flow

```
class Student
{
    String name;
    int age;
    static String nationality = "IND";
    Student(String name,int age)
    {
        System.out.println("Constructor got called");
        this.name = name;
        this.age = age;
    }
    static
    {
```

```
        System.out.println("Static Block :: Loading of Student.class file");
    }
    public void dispStdDetails()
    {
        System.out.println("Inside instance method");
        System.out.println("Name is :: "+name);
        System.out.println("Age is :: "+age);
        System.out.println("Nationality is :: "+nationality);
    }
}
class Test
{
    static
    {
        System.out.println("Loading of Test.class file");
    }
    public static void main(String[] args)
    {
        System.out.println("Inside main()");
        Student std= new Student("sachin",49);
        std.dispStdDetails();
    }
}
```

Output

Loading of Test.class file

Inside main()

Static Block :: Loading of Student.class file

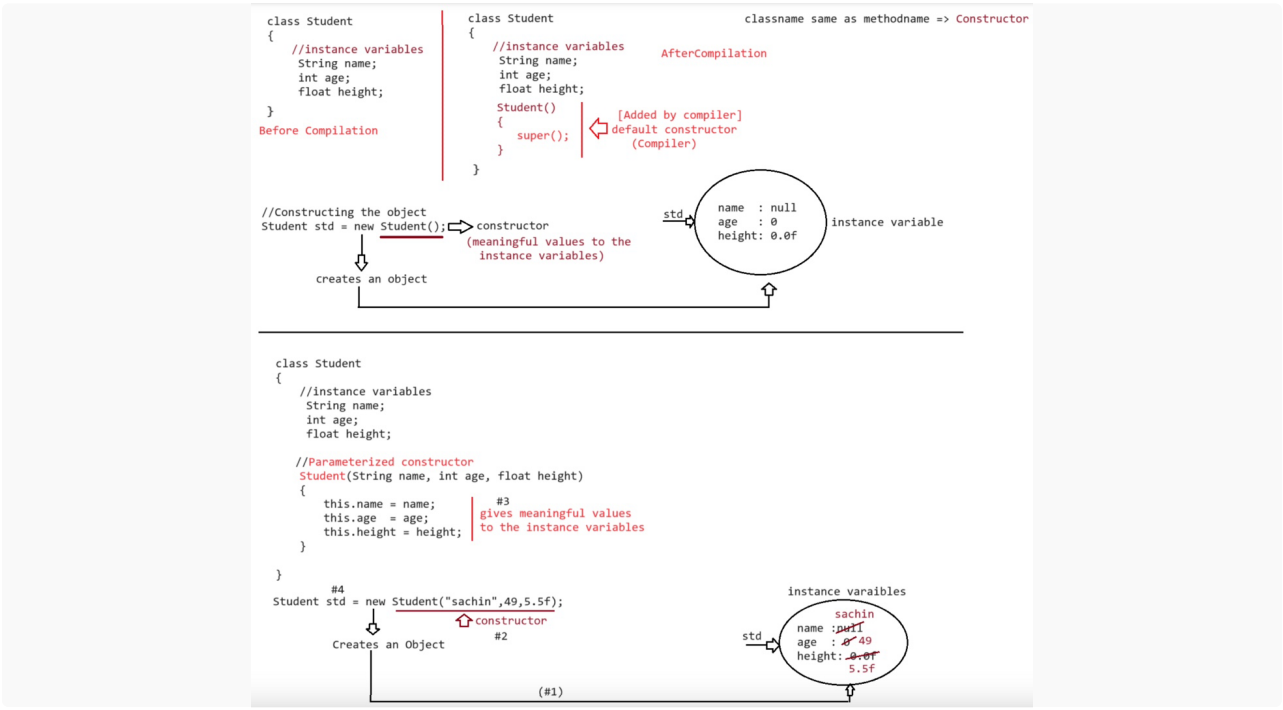
Constructor got called

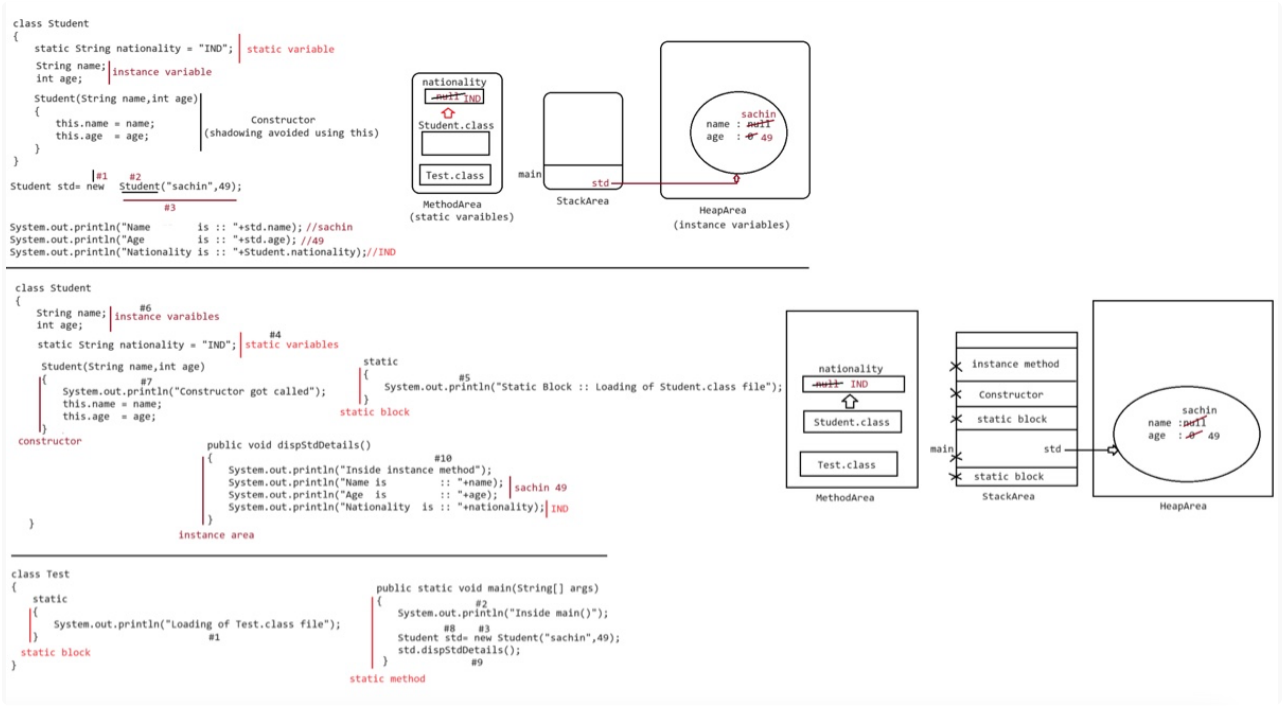
Inside instance method

Name is :: sachin

Age is :: 49

Nationality is :: IND





What type of variable we need to make as static in oops?

Ans. If we want all the objects of a particular class to get a common copy, then such variables should be made as "static",so we say static variables as class variables.

What type of methods we need to make as static in oops?

Ans. If we want some behaviour to be executed even without its object creation, then such type of methods we need to mark as "static".

Can we write multiple static blocks, if yes how will they get executed?

Ans. yes possible, it gets executed in order of placing the block in the class.

Read Indirect Write Only (RIWO)

Direct Read => Within a static block, if we are reading a variable then such type of read is called as "Direct read".

Indirect Read => If we are calling a static method,and with in the static method if we are reading a static variable then such type of read is called "Indirect Read".

1)- Indirect Read -->

A)-

```
class Test
{
    static int i = 10;
    static
    {
        methodOne();
        System.out.println("First static block");
    }
    public static void main(String[] args)
    {
        methodOne();
        System.out.println("Inside main method");
    }
    public static void methodOne()
    {
        System.out.println(j);
    }
    static
    {
        System.out.println("Second static block");
    }
    static int j = 20;
}
```

Output

0

First static block

Second static block

20

Inside main method

B)-

```
class Test
{
    static
    {
        methodOne();
    }
    public static void main(String[] args)
    {
    }
    public static void methodOne()
    {
        System.out.println(i);
    }
    static int i = 10;
}
```

Output

0

2)- Direct Read -->

If a variable is in RIWO state ,then we can't perform direct read operation, if we try to do it would result in CompileTime Error.

A)-

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

}

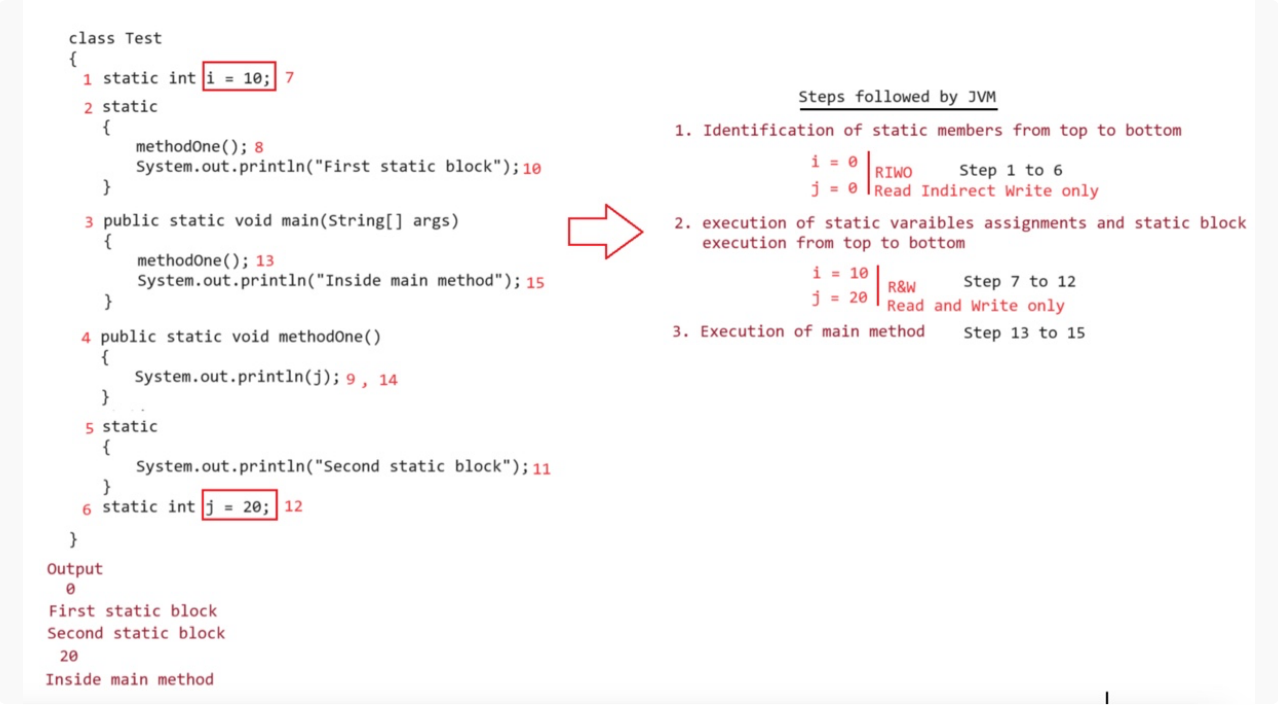
Output: CompileTimeError :illegal forward reference

Eg:

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

Output

10

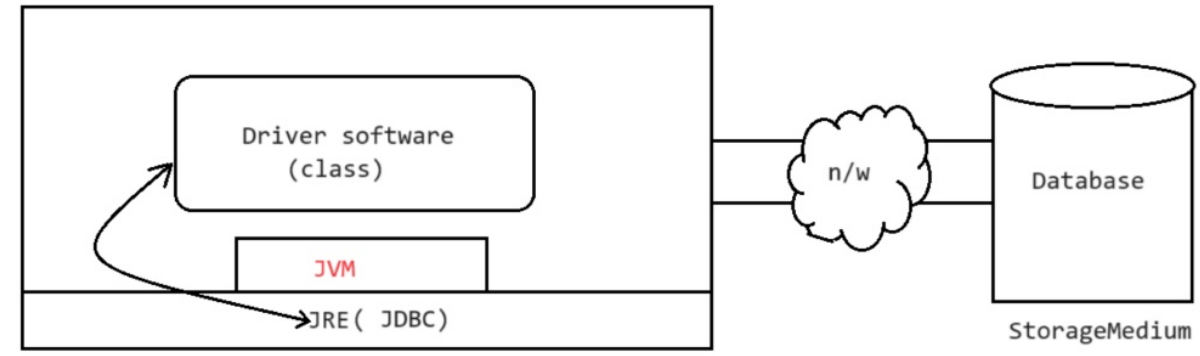


Usage of static block in realtime

static block : It is called as "One Time Execution block". **It will be executed during the loading of .class file.**

Note1:

Every driver software internally contains static block to register the driver with DriverManager, which helps the programmer to get JDBC environment in JRE, to do this we need static block.



Capable of setting environment for java program execution

```
class Driver
{
    static
    {
       //Register the driver with DriverManager(Setting JDBC environment)
    }
}
```

Note2:

Can we write any statements to the console without writing statement inside main()?

Answer : yes , by using static block.

eg#1.

```
class Test
{
    static int i = methodOne();
    public static void main(String[] args)
    {
    }
    public static int methodOne()
    {
        System.out.println("Hello i can print");
        System.exit(0);
        return 10;
    }
}
```

Output

Hello i can print

eg#2.

```
class Test
{
    static Test t = new Test();
    public Test()
    {
        System.out.println("Hello i can print");
        System.exit(0);//shutdown jvm
    }
    public static void main(String[] args)
    {
    }
}
```

Output

Hello i can print

Note: It is mandatory to write main() inside every class for the execution to happen, if we don't write then class will not be loaded.

public static void main(String[] args).

System.exit(0) -> This line if jvm executes, then jvm will shutdown itself, by skipping the remaining statements in the program.

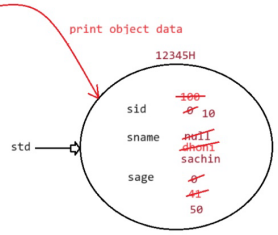
Instance control flow

instance block : This block gets executed at the time of creating an object,but before the call to constructor. This block will be executed for every object we create,but before the call to a constructor.

Note:

1. **Constructor :** initialize the object with the required values.
2. **instance block :** Apart from initalization, if we want to perform any other activities then we need to go for "instance block".
3. **Order of execution :** first instance block then constructor.

```
class Student
{
    //instance variable
    int sid;
    String sname;   3
    int sage;

    //constructor: shadowing -> resolved through "this"
    Student(int sid, String sname,int sage)
    {
        this.sid   = sid;
        this.sname = sname;                    5
        this.sage  = sage;
    }

    //instance block
    {
        sid = 100;
        sname = dhoni;    4
        sage = 41;
    }

    //instance method
    public void dispStdDetails()
    {
        System.out.println("SID   is :: "+sid);
        System.out.println("SNAME is :: "+sname);
        System.out.println("SAGE  is :: "+sage);
    }
}

class Test
{
    public static void main(String[] args) 1
    {
        Student std = new Student(10,"sachin",50); 2
                                  constructor
        std.dispStdDetails();
    }
}
```



1. new
2. load the class [only one time]
3. scan the instance variable allocate memory fill with default value.
4. scan for the instance block, if found load and execute.
5. make a call to constructor and give the values supplied by programmer
6. return the reference.

```
class Test
{
  3 int i = 10;  9
  4 {
        methodOne(); 10
        System.out.println("First instance block"); 12
    }
  5 Test()
    {
        System.out.println("Test class constructor"); 15
    }
    public static void main(String[] args)  1
    {
        Test t =new Test(); 2
        System.out.println("Inside main method"); 16
    }
  6 public void methodOne()
    {
        System.out.println(j); 11
    }
  7 {
        System.out.println("Second instance block"); 13
    }
  8 int j = 20; 14
}
```

1. Identification of instance members from top to bottom.

i = 0	RIWO
j = 0	

2. Execution of instance variable assignment and instance block execution from top to bottom

i = 10	R&W
j = 20	

3. Execution of constructor

```
Output
  0
First instance block
Second instance block
Test class constructor
Inside main method
```

Write a program to count the no of objects created for a class?

1)-

```
class Test
{
    //static variable
```

```
    static int count;
    Test(){}
    Test(int i){}
    Test(int i, int j){}
    Test(int i,int j, int k){}
    //instance block
    {
        count++;
    }
    //instance method
    public void disp()
    {
        System.out.println("No of objects created is :: "+count);
    }
    public static void main(String[] args)
    {
        Test t1 = new Test();
        Test t2 = new Test(10);
        Test t3 = new Test();
        Test t4 = new Test(100);
        Test t5 = new Test(100,200);
        Test t6 = new Test(100,200,300);
        t1.disp();
    }
}
```

Output::

No of objects created is :: 6

2)-

```
class Test
{
    //static variable
    static int count;
    Test(int... args){        //var-args
        count++;
    }
    //instance method
    public void disp()
```

```
    {
        System.out.println("No of objects created is :: "+count);
    }
    public static void main(String[] args)
    {
        Test t1 = new Test();
        Test t2 = new Test(10);
        Test t3 = new Test();
        Test t4 = new Test(100);
        Test t5 = new Test(100,200);
        Test t6 = new Test(100,200,300);
        t1.disp();
    }
}
```

Output::

No of objects created is :: 6

final variables

a. final instance variable

b. final static variable

c. final local varaible

Note: A variable is said to be final iff the value of the variable is "fixed", we can't change the value once it is initialized. If a variable is final, then compiler will get to know the value of the variable and these values will be used during the evaluation of an Expression.

eg#1.

```
class Test
{
    int i;
    public static void main(String[] args)
    {
        System.out.println(new Test().i);
    }
}
```

Output

0

If an instance variable is marked as final, then for such instance variables, jvm will not give default value, programmer should supply the value for instance variables, otherwise it would result in "CompileTimeError".

eg#2.

```
class Test
{
    final int i;
    public static void main(String[] args)
    {
        System.out.println(new Test().i);
    }
}
```

Output

CompileTimeError:: variable i not initialized in the default constructor

Places to initalize the value for final instance variable

1. **At the time of declaration**

eg#1

```
class Test
{
    final int i=100;
    public static void main(String[] args)
    {
        System.out.println(new Test().i);//100
    }
}
```

2. **Inside instance block**

eg#1

```
class Test
{
    final int i;
    {
        i = 100;
    }
    public static void main(String[] args)
    {
```

```
        System.out.println(new Test().i);//100
    }
}
```

3. **Inside constructor**

eg#1

```
class Test
{
    final int i;
    Test()
    {
        i = 100;
    }
    public static void main(String[] args)
    {
        System.out.println(new Test().i);//100
    }
}
```

Apart from these 3 places, if we try to do initalization then it would result in "CompiletimeError".

eg#1.

```
class Test
{
    final int i;
    public void m1()
    {
        i = 100;//CE: error: cannot assign a value to final variable i
    }
    public static void main(String[] args)
    {
        System.out.println(new Test());
    }
}
```

static variable

eg#1.

```
class Test
{
```

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

If an static variable is marked as final, then for such static variables, jvm will not give default value, programmer should supply the value for static variables, otherwise it would result in "CompileTimeError".

eg#2.

```
class Test
{
    final static int i;
    public static void main(String[] args)
    {
        System.out.println(i);
    }
}
```

Output : CE: variable i not initialized in the default constructor

Places to initialize final static variable

1. **At the time of declaration**

eg#1.

```
class Test
{
    final static int i=100;
    public static void main(String[] args)
    {
        System.out.println(i);//100
    }
}
```

2. **Inside static block**

eg#2.

```
class Test
{
    final static int i;
```

```
    static
    {
        i = 100;
    }
    public static void main(String[] args)
    {
        System.out.println(i);//100
    }
}
```

If a variable is marked as static and final, then for those variables initialization should be completed before class loading completes,otherwise it would result in "CompileTimeError".

eg#3.

```
class Test
{
    final static int i;
    Test()
    {
        i = 100;//CE: cannot assign a value to final variable i
    }
    public static void main(String[] args)
    {
        System.out.println(i);
    }
}
```

final local variable

eg#1.

```
class Test
{
    public static void main(String[] args)
    {
        int i;
        System.out.println(i);
    }
}
```

Output: error: variable i might not have been initialized

eg#2.

```
class Test
{
    public static void main(String[] args)
    {
        final int i;
        System.out.println(i);
    }
}
```

Output: error: variable i might not have been initialized

The only place where we can initialize the local variable is at the time of declaration.

eg#1.

```
class Test
{
    public static void main(String[] args)
    {
        final int i=100;
        System.out.println(i);//100
    }
}
```

eg#3.

```
class Test
{
    public static void main(String[] args)
    {
        methodOne(100,200);
    }
    public static void methodOne(final int i, int j)
    {
        i = 1000;
        j = 2000;
        System.out.println(i + " " + j);
    }
}
```

Output :: error: final parameter i may not be assigned i = 1000;

Note: The only access modifier applicable at local variable level is "final".

Inheritance in java

=> It refers to process of linking 2 classes.

=> In java inheritance can be promoted in 2 ways

a. IS-A relationship

b. HAS-A relationship

=> IS-A relationship is referred to "Inheritance".

What is inheritance?

It refers to process of acquiring properties and behaviours from parent to child class.

What is the benefit of inheritance?

=> Re-Usabilitiy.

IS-A relationship to promote in java we use "extends" keyword.

eg#1.

```
class Parent
{
    public void methodOne()
    {
        System.out.println("Parent method");
    }
}
class Child extends Parent
{
    //inherited method(public category from parent)
    public void methodOne()
    {
        System.out.println("Parent method");
    }
    //Specialized method
    public void methodTwo()
    {
        System.out.println("Child method");
    }
}
class Test
```

```
{
    public static void main(String[] args)
    {
        Parent p= new Parent();
        p.methodOne();
        p.methodTwo(); //CE: can't find symbol

        System.out.println();

        Child c = new Child();
        c.methodOne();
        c.methodTwo();

        System.out.println();

        Parent p1 =new Child();
        p1.methodOne();
        p1.methodTwo(); //CE: can't find symbol

        System.out.println();

        Child c1 = new Parent();  //incompatible types
    }
}
```

Output

D:\OctBatchMicroservices\Test.java:22: error: cannot find symbol

p.methodTwo();//CE: can't find symbol

^

symbol: method methodTwo()

location: variable p of type Parent

D:\OctBatchMicroservices\Test.java:34: error: cannot find symbol

p1.methodTwo();//CE: can't find symbol

^

symbol: method methodTwo()

location: variable p1 of type Parent

D:\OctBatchMicroservices\Test.java:38: error: incompatible types: Parent cannot be

converted to Child

Child c1 = new Parent();//incompatible types

Note:

1)- Whatever the parent class has under public category(variables, methods, etc.), by default will be available to child class.

2)- Whatever the child class has under public category(variables, methods, etc.), by default won't be available to the parent class.

3)- Using the parent reference,we can make a call only to the parent class methods but not the child class specialized methods.

4)- Parent class reference can be used to collect child class objects, but by using parent class reference we can call only parent class methods but not child class specialized methods.

5)- Child class reference can't be used to hold parent class objects.

eg#2.

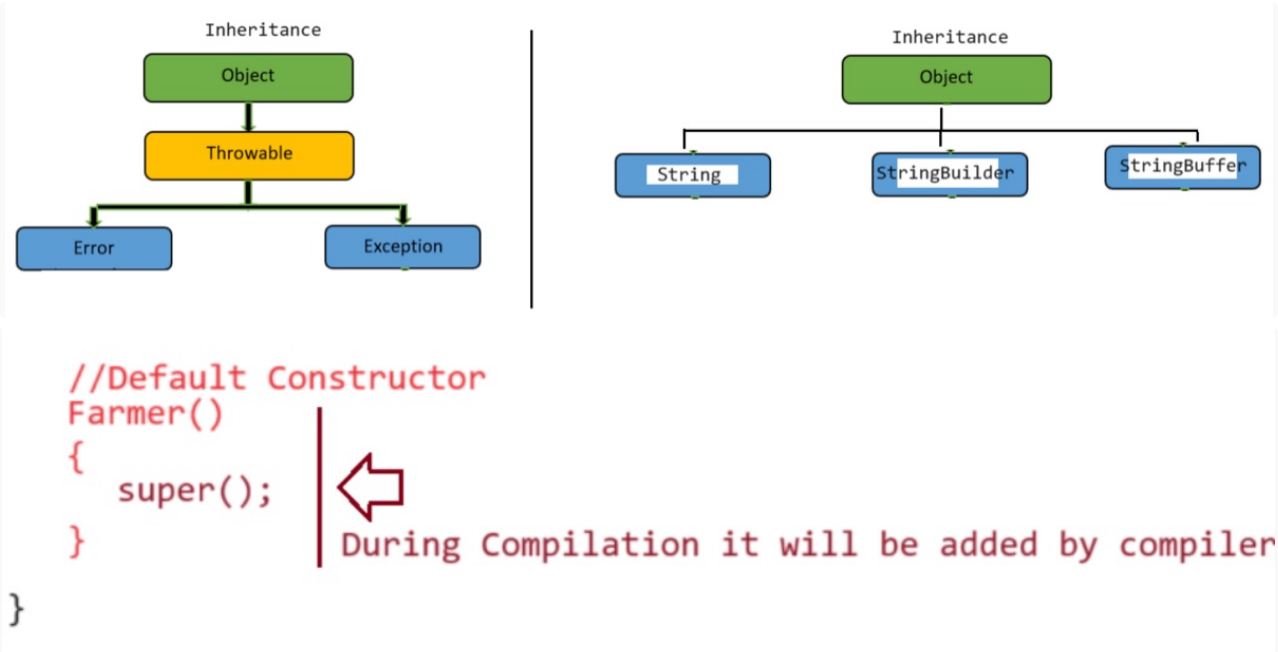
```
class Object
{
    public final native java.lang.Class<?> getClass();
    public native int hashCode();
    protected native java.lang.Object clone() throws
    java.lang.CloneNotSupportedException;

    //used while working with String,StringBuilder,StringBuffer
    public boolean equals(java.lang.Object);

    //toString() gets called automatically when we print the reference of the object[callback method/magic method]
    public java.lang.String toString();

    //Methods related to MultiThreading
    public final native void notify();
    public final native void notifyAll();
    public final void wait() throws java.lang.InterruptedException;
    public final native void wait(long) throws java.lang.InterruptedException;
    public final void wait(long, int) throws java.lang.InterruptedException;
    //Method related to GarbageCollector
    protected void finalize() throws java.lang.Throwable;
}
class String extends Object
```

```
{
    //String class specific methods
    @Override
    public String toString()
    {
        //print the content of the Strings
    }
}
class StringBuffer extends Object
{
    //StringBuffer class specific methods
    @Override
    public String toString()
    {
        //print the content of the Strings
    }
}
class StringBuilder extends Object
{
    //StringBuilder class specific methods
    @Override
    public String toString()
    {
        //print the content of the Strings
    }
}
Student std = new Student(10,"sachin");
System.out.println(std);    //std.toString():: hashCode
System.out.println();
String str = new String("sachin");
System.out.println(str);    //str.toString():: sachin
```

Note:

1. For all java classes,, the most commonly requried functionality is defined inside object class so Object class is called as "root" for all java classes.
2. For all java exceptions and errors, the most commonly required functionality is defined inside "Throwable" class, so Throwable class acts as root for "Exception Hierarchy".

3. For compiler and jvm by default all the classes available in a package/folder called " java.lang.* " will available.



Snippets

Given:

```
public class Barn {
    public static void main(String[] args) {
        new Barn().go("hi", 1);
        new Barn().go("hi", "world", 2);
    }
    public void go(String... y, int x) {
        System.out.print(y[y.length - 1] + " ");
    }
}
```

What is the result?

A. hi hi

B. hi world

C. world world

D. Compilation fails.

E. An exception is thrown at runtime.

Answer: D (varargs parameter must be the last parameter)

Q>

```
public static void test(String str) {
    int check = 4;
    if (check = str.length()) {
        System.out.print(str.charAt(check -= 1) +", ");
    } else {
        System.out.print(str.charAt(0) + ", ");
    }
}
```

and the invocation:

```
test("four");
test("tee");
test("to");
```

What is the result?

A. r, t, t,

B. r, e, o,

C. Compilation fails.

D. An exception is thrown at runtime.

Answer: C

Given:

```
public class Boxer1{
    Integer i;
    int x;
    public Boxer1(int y) {
        x = i+y;   // x = null + 4
        System.out.println(x);
    }
    public static void main(String[] args) {
        new Boxer1(new Integer(4));
    }
}
```

What is the result?

A. The value "4" is printed at the command line.

B. Compilation fails because of an error in line 5.

C. Compilation fails because of an error in line 9.

D. A NullPointerException occurs at runtime.

E. A NumberFormatException occurs at runtime.

F. An IllegalStateException occurs at runtime.

Answer: D

Given this code from Class B:

```
25. A a1 = new A();
26. A a2 = new A();
27. A a3 = new A();
28. System.out.println(A.getInstanceCount());
```

What is the result?

```
1.  public class A{
3.  private int counter = 0;
5.  public static int getInstanceCount() {
6.  return counter;
7.  }
9.  public A() {
10. counter++;
11.}
13.}
```

A. Compilation of class A fails.

B. Line 28 prints the value 3 to System.out.

C. Line 28 prints the value 1 to System.out.

D. A runtime error occurs when line 25 executes.

E. Compilation fails because of an error on line 28.

Answer: A(static methods cannot directly access instance variables without an instance of the class.)

Given

```
public class Venus {
public static void main(String[] args) {
int[] x = { 1, 2, 3 };
int y[] = { 4, 5, 6 };
new Venus().go(x, y);
}
void go(int[]... z) {
```

```
for (int[] a : z)
System.out.print(a[0]);
}
}
```

What is the result?

A. 1

B. 12

C. 14

D. 123

E. Compilation fails.

F. An exception is thrown at runtime.

Answer: C

Given

```
public class Test{
public static void main(String[] args){
String[] arr = {"L","I","V","E"};
int i=-2;
if(i++==-1) arr[-(--i)]="F";
else if (--i==-2) arr[-++i] = "O";
for(String s: arr) System.out.print(s);
}
}
```

A. compilation error

B. An exception is thrown at runtime

C. LIVE

D. LIFE

E. LIVO

F. LOVE

G. LIOE

Answer: F

Given

```
public class Test{
Boolean b[] = new Boolean[2];
public static void main(String... args){
```

Test t= new Test();

System.out.println(t.b[0] + ":" +t.b[1]);// null + ":" + null = > null:null

}

}

A. NullpointerException

B. false:false

C. true:true

D. null:null (ans)

E. RunTimeException other than NullPointerException

Given

class Alien {

String invade(short ships) { return "a few"; }

String invade(short... ships) { return "many"; }

}

class Defender {

public static void main(String [] args) {

System.out.println(new Alien().invade(7));

}

}

What is the result?

A. many

B. a few

C. Compilation fails

D. The output is not predictable

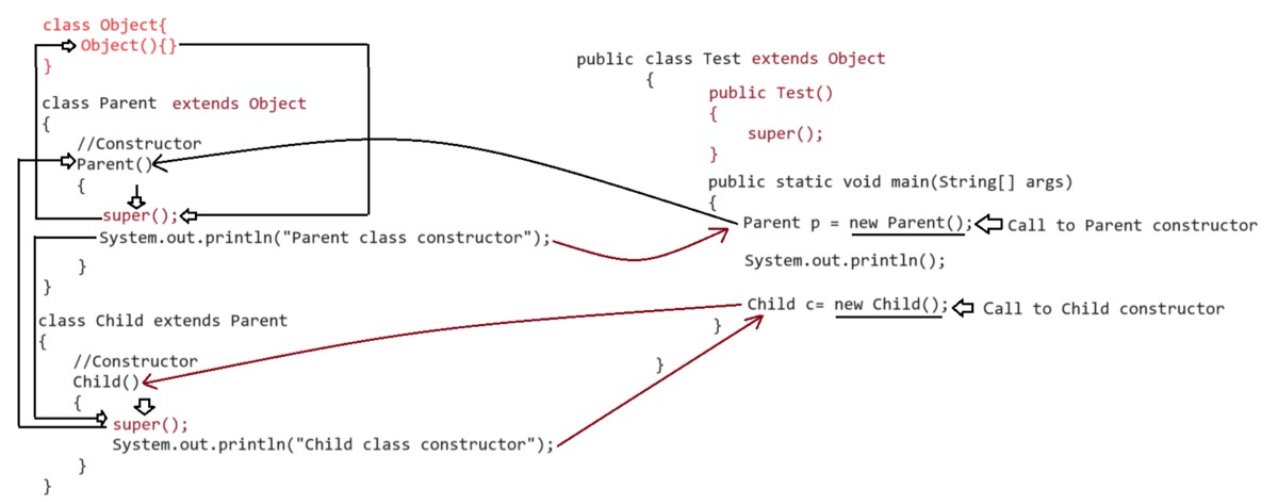
E. An exception is thrown at runtime

Answer: B

Note::

1)- When an object of parent class is created ,the parent class constructor will be called.

2)- When an object of child class is created ,both parent and child class constructor will be called because of super().




```
class Object{
   Object(){}
}
class Person extends Object
{                                   CE: call can't be resolved
    String name;
    int age;
    String gender;
    float height;

    //Parameterized constructor :: shadowing
    Person(String name,int age,String gender,float height)
    {   super();
        this.name   = name;
        this.age    = age;
        this.gender = gender;
        this.height = height;
    }
}
```

```
class Student extends Person
{
    String sid;
    int marks;

    //Parameterized constructor :shadowing
    Student(String name,int age, String gender, float height, String sid,int marks)
    {   super();
        this.sid = sid;
        this.marks = marks;
    }
    public void dispStudentDetails()
    {
        System.out.println("SID   ::   "+sid);
        System.out.println("MARKS ::   "+marks);
    }
}
```

```
public class Test extends Object
{   public Test()
    {
       super();
    }
    public static void main(String[] args)
    {
        Student std = new Student("sachin",49,"M",5.4f,"IND10",35);
        std.dispStudentDetails();
    }
}
```

```
class Object{
   Object(){}
}
class Person extends Object
{
    String name;
    int age;
    String gender;
    float height;

    //Parameterized constructor :: shadowing
    Person(String name,int age,String gender,float height)
    {   super();
        this.name   = name;
        this.age    = age;
        this.gender = gender;
        this.height = height;
    }
}
```

```
class Student extends Person
{
    String sid;
    int marks;

    //Parameterized constructor :shadowing
    Student(String name,int age, String gender, float height, String sid,int marks)
    {
      //call to parent class parameterized constructor
        super(name,age,gender,height); Call to parent class constructor
                                       explicitly made by programmer
      this.sid = sid;
      this.marks = marks;
    }
    public void dispStudentDetails()
    {
        System.out.println("SID   ::   "+sid);
        System.out.println("MARKS ::   "+marks);
    }
}
```

```
public class Test extends Object
{   public Test()
    {
       super();
    }
    public static void main(String[] args)
    {
        Student std = new Student("sachin",49,"M",5.4f,"IND10",35);
        std.dispStudentDetails();
    }
}
```

If there is a constructor created by user then compiler add super() to the constructor. And if no constructor is added then compiler will add default constructor with super() included in it.**(in above diagram red lines are done by compiler)(except in last one where it is mentioned).**

Explain the importance of super()?

In case of Parent class, if parent class has a parameterized constructor then in child class constructor compulsorily there should be a call to parent class parameterized constructor otherwise the code would result in "CompileTime Error".

super vs super()

super() -> it is used to call parent class constructor from the child class.

super -> It is used to avoid name clash of properties and behaviors b/w parent and child class.

Note:

Explain the importance of this keyword vs super keyword?

this : It is used to avoid shadowing problem[name clash b/w local variable and instance variable]

super : It is used to avoid name clash b/w properties and behaviors of parent and child class[Inheritance]

eg#1.

```
class A
{
    int i;
    A()
    {
        System.out.println("Parent class constructor");
    }
}
class B extends A
{
    int i;
    B()
    {
        System.out.println("Child class constructor");
    }
    public void setData(int x, int i)
    {
        //Giving x value to parent class i
        super.i = x;
```

```
        //Giving i value to child class i
        this.i = i;
    }
    public void disp()
    {
        System.out.println("A class i value is :: " +super.i);
        System.out.println("B class i value is :: " +i);
    }
}
public class Test
{
    public static void main(String[] args)
    {
        B b = new B();
        b.setData(10,20);
        b.disp();
    }
}
```

Output

A class i value is :: 10

B class i value is :: 20

Constructor Overloading

A class can contain more than one constructor,this way of writing a constructor we call it as "Constructor Overloading". When we have constructor overloading in our code, to make a call to constructor within a class we use "this()".

super() : it will take the control to parent class constructor.

eg#1.

```
//Constructor Overloading
class Demo
{
    Demo(int i)
    {
        super();
        System.out.println("int arg constructor");
    }
    Demo(float f)
```

```
    {
        super();
        System.out.println("float arg constructor");
    }
    Demo()
    {
        super();
        System.out.println("zero arg constructor");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Demo d1= new Demo();
        Demo d2= new Demo(10);
        Demo d3= new Demo(10.5f);
    }
}
```

Output

zero arg constructor

int arg constructor

float arg constructor

eg#2.

```
//Constructor Overloading
```

this() :: It is used to make a call to current class constructors only.

```
class Demo
{
    Demo(int i)
    {
        this(10.5f);
        System.out.println("int arg constructor");
    }
    Demo(float f)
    {
        System.out.println("float arg constructor");
```

```
    }
    Demo()
    {
        this(10);
        System.out.println("zero arg constructor");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Demo d1= new Demo();//float-arg/int-arg/zero-arg constructor
        System.out.println();
        Demo d2= new Demo(10);//float-arg/int-arg constructor
        System.out.println();
        Demo d3= new Demo(10.5f);//float-arg constructor
    }
}
```

Output

float arg constructor

int arg constructor

zero arg constructor

float arg constructor

int arg constructor

float arg constructor

Predict the output of the following code from the compiler

1)-

```
class Test
{
}
```

Compiler

```
class Test exteds Object
{
    Test()
    {
        super();
```

```
    }
}
```

2)- public class Test

```
{
}
```

Compiler

```
public class Test exteds Object
{
    public Test()
    {
        super();
    }
}
```

3)- class Test

```
{
    void test()
    {
    }
}
```

Compiler

```
class Test exteds Object
{
    Test()
    {
        super();
    }
    void test()
    {
    }
}
```

4)- class Test

```
{
    Test(int i)
    {
    }
}
```

Compiler

```
class Test extends Object
{
    Test(int i )
    {
        super();
    }
}
```

5)- class Test

```
{
    Test(int i)
    {
        this();
    }
    Test()
    {
    }
}
```

Compiler

```
class Test extends Object
{
    Test(int i)
    {
        this();
    }
    Test()
    {
        super();
    }
}
```

Conclusions of this() vs super()

Case1:

We have to take super() or this() only in the 1st line of the constructor, if we are taking anywhere else it would result in "CompileTimeError".

Case2:

We can't use either super() or this() both simultaneously

Case3:

super() or this() should always be the first statement inside the constructor but we can't use inside the method, if we try to use it would result in "CompileTime Error".

```
class Demo
{
    public void methodOne();
    {
        super();
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Demo d = new Demo();
        d.methodOne();
    }
}
```

Which of the following statemetns are true?

a. Whenever we are creating an object of child class, parent class constructor will be called ?

Answer: true

b. Whenever we are creating an object of child class object then automatically parent class object will be created?

Answer: false. only child class object will be created but not the parent class.

eg#1.

```
class Parent
{
    Parent()
    {
        System.out.println(this.hashCode()); //366712642
    }
}
class Child extends Parent
{
    Child()
```

```
    {
        System.out.println(this.hashCode()); //366712642
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Child c = new Child();
        System.out.println(c.hashCode());   //366712642
    }
}
```

Recursive call

1. Calling a function within the same function is called "recursive call".
2. In recursive call called and calling function both are same.

eg#1.

```
public static void methodOne()
{
    methodOne();
}
```

Case1: recursive method call is always "RuntimeException"(not always --> if we have wrote conditions to break the recursion then it will not lead to StackOverflowError) where as recursive constructor invocation is a "CompiletimeError".

eg#1

```
public class Test
{
    public static void methodOne()
    {
        methodTwo();
    }
    public static void methodTwo()
    {
        methodOne();
    }
    public static void main(String[] args)
```

```
    {
        methodOne();
        System.out.println("hello");
    }
}
```

Output:: java.lang.StackOverflowError

eg#2.

```
public class Test
{
    Test(int i)
    {
        this();
    }
    Test()
    {
        this(10);
    }
    public static void main(String[] args)
    {
        Test t = new Test();
        System.out.println("hello");
    }
}
```

CE: recursive constructor invocation

Q>

Given

```
public class Hello {
    String title;
    int value;
    public Hello() {
        title += " World";
    }
    public Hello(int value) {
        this.value = value;
        title = "Hello";
        Hello();            //we cannot call a constructor
```

```
    }
}
```

and:

```
Hello c = new Hello(5);
System.out.println(c.title);
```

What is the result?

A. Hello

B. Hello World

C. Compilation fails.

D. Hello World 5

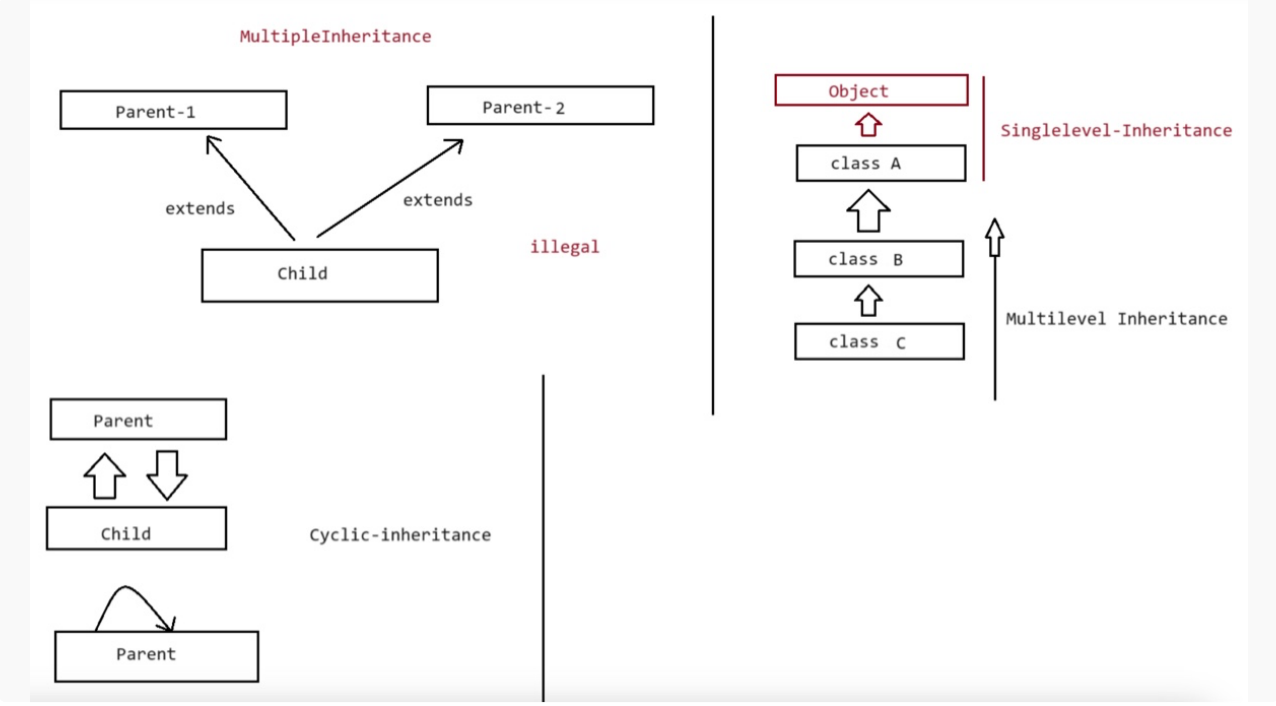
E. The code runs with no output.

F. An exception is thrown at runtime.

Answer: C

Types of inheritance supported by java

1. **Multiple inheritance :** Not supported because of ambiguity in java through "classes".
2. **Multilevel inheritance :** Getting properties from parent to child in hierarchial way is refered as "MultiLevel Inheritance".
3. **Cyclic inheritance :** Not supported in java becoz of constructor invocation in loop.



Why java won't support Multiple Inheritance?

Ans. To avoid ambiguity b/w method calls coming for mulitple inheritance, java won't support mulitple inheritance.

eg#1.

```
class Parent1
{
    public void methodOne()
    {
        System.out.println("From Parent1");
    }
}
class Parent2
{
    public void methodOne()
    {
        System.out.println("From Parent2");
    }
}
class Child extends Parent1,Parent2
{
}
public class Test
{
    public static void main(String[] args)
    {
        Child c = new Child();
        c.methodOne();
    }
}
```

Output: CE: Parent1,Parent2

eg#2.

```
class Parent extends Child
{
    Parent()
    {
        super();
```

```
    }
}
class Child extends Parent
{
    Child()
    {
        super();
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Child c = new Child();
    }
}
```

Output:CE cyclic inheritance involing the parent

program to Demonstrate the usage of inheritance

```
class Plane
{
    String engine;
    float fuel;
    int wheel;
    public void takeOff()
    {
        System.out.println("Plane tookoff...");
    }
    public void fly()
    {
        System.out.println("Plane is flying...");
    }
    public void land()
    {
        System.out.println("Plane is landing...");
    }
}
```

```
class Passenger extends Plane
{
    public void carryPassengers()
    {
        System.out.println("Carrying Passengers...");
    }
}
class Cargo extends Plane
{
    public void carryCargo()
    {
        System.out.println("Carrying Cargo...");
    }
}
class Fighter extends Plane
{
    public void carryWeapons()
    {
        System.out.println("Carrying Weapons...");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        //Creating 3 objects of Plane Type
        Cargo c = new Cargo();
        Passenger p =new Passenger();
        Fighter f = new Fighter();
        //Taking the actions for all the 3 planes
        c.takeOff();
        c.carryCargo();
        c.fly();
        c.land();

        System.out.println();
```

```
        p.takeOff();
        p.carryPassengers();
        p.fly();
        p.land();

        System.out.println();

        f.takeOff();
        f.carryWeapons();
        f.fly();
        f.land();
    }
}
```

Output

Plane tookoff...

Carrying Cargo...[Specialized method]

Plane is flying...

Plane is landing...

Plane tookoff...

Carrying Passengers...[Specialzied method]

Plane is flying...

Plane is landing...

Plane tookoff...

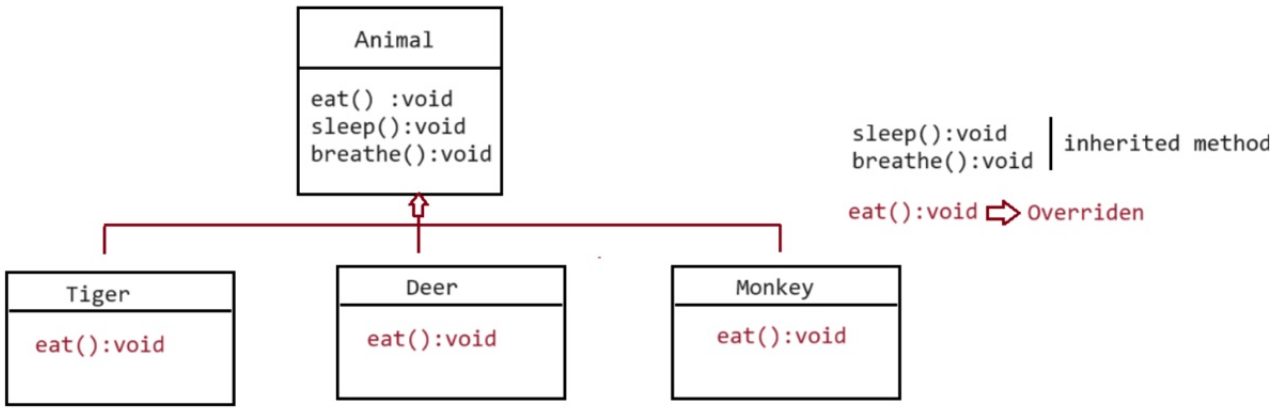
Carrying Weapons...[Specialzied method]

Plane is flying...

Plane is landing...

Overident Methods

UML(Unified Modeling Language)



Taking method from Parent,but child is not happy with the implementation so child will give the body for the method coming from parent, we say such methods as "Overriden Methods".

```
class Animal
{
    public void eat()
    {
        System.out.println("Animal is eating...");
    }
    public void sleep()
    {
        System.out.println("Animal is sleeping...");
    }
    public void breathe()
    {
        System.out.println("Animal is breathing...");
    }
}
class Tiger extends Animal
{
    //informing compiler about overidden method
    @Override
    public void eat()
    {
        System.out.println("Tiger hunts and eat...");
    }
}
```

```
class Deer extends Animal
{
    //informing compiler about overidden method
    @Override
    public void eat()
    {
        System.out.println("Deer will graze and eat...");
    }
}
class Monkey extends Animal
{
    //informing compiler about overidden method
    @Override
    public void eat()
    {
        System.out.println("Monkey steal and eat...");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        //Creating an Object of Animal Type
        Tiger t = new Tiger();
        Deer d = new Deer();
        Monkey m = new Monkey();

        //Invoking the behaviours of all 3 animals
        t.eat();
        t.sleep();
        t.breathe();

        System.out.println();

        d.eat();
        d.sleep();
        d.breathe();
```

```
        System.out.println();

        m.eat();
        m.sleep();
        m.breathe();
    }
}
```

Output

Tiger hunts and eat... [Overriden method got called]

Animal is sleeping...

Animal is breathing...

Deer will graze and eat...[Overriden method got called]

Animal is sleeping...

Animal is breathing...

Monkey steal and eat...[Overridentmethod got called]

Animal is sleeping...

Animal is breathing...

Note: In case of Overriding

1. Compiler will use reference of the type to check whether the respective method is available in the class or not.
2. JVM will use the current object and respective method of that object will be called.

Overriding

1)- Whatever the parent has by default available to the child class through inheritance,if the child is not satisfied with the parent class method implementation then child class is allowed to redefine that parent class method in the Child class in its own way this process is called as "Overriding".

In java Polymorphism is one of the pillar of oops. We have 2 types of polymorphism

a. Compiletimebinding / Earlybinding / staticbinding

eg: Overloading, method-hiding

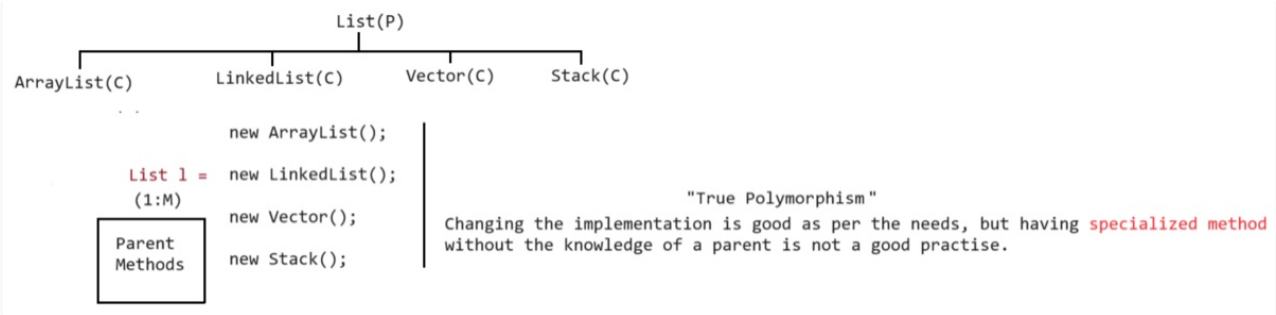
b. Runtime / dynamic / latebinding

eg: Overriding

2)- In case of Overriding JVM will play a vital role of binding the method call to method body, so we say overriding as "RuntimePolymorphism". In case of Overriding compiler will just check the reference type and see whether the method signature is present in the class or not.

Parent p1 = new Child();

This syntax is really important in terms of code readability and consistency. The main idea is that everything should be in parent class and child class should either use it or override it if child is not satisfied with it. So there should be no child specific methods(as we can't access it with p1). With p1 we can access all the methods which are in parent and the overridden methods if any will be taken from child(logic). We cant access child specific methods with p1. Overloading increases number of lines in code while with overiding we can decrease those lines.And JVM will make a decision which method to execute.



Difference b/w Method Overloading and Overriding -->

eg#1.

```
class Animal
{
    public void eat()
    {
        System.out.println("Animal is eating...");
    }
    public void sleep()
    {
        System.out.println("Animal is sleeping...");
    }
    public void breathe()
    {
        System.out.println("Animal is breathing...");
    }
}
class Tiger extends Animal
{
    //informing compiler about overidden method
    @Override
    public void eat()
    {
```

```
        System.out.println("Tiger hunts and eat...");
    }
}
class Deer extends Animal
{
    //informing compiler about overidden method
    @Override
    public void eat()
    {
        System.out.println("Deer will graze and eat...");
    }
}
class Monkey extends Animal
{
    //informing compiler about overidden method
    @Override
    public void eat()
    {
        System.out.println("Monkey steal and eat...");
    }
}
//Helper class
class Forest
{
    //Method Overloading
    public void allowAnimal(Tiger t)
    {
        t.eat();
        t.sleep();
        t.breathe();
    }
    public void allowAnimal(Deer d)
    {
        d.eat();
        d.sleep();
        d.breathe();
    }
```

```
    public void allowAnimal(Monkey m)
    {
        m.eat();
        m.sleep();
        m.breathe();
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Tiger t = new Tiger();
        Deer d = new Deer();
        Monkey m = new Monkey();
        Forest f = new Forest();
        f.allowAnimal(t);

        System.out.println();

        f.allowAnimal(d);

        System.out.println();

        f.allowAnimal(m);
    }
}
```

Output

Tiger hunts and eat...

Animal is sleeping...

Animal is breathing...

Deer will graze and eat...

Animal is sleeping...

Animal is breathing...

Monkey steal and eat...

Animal is sleeping...

Animal is breathing...

Overriding

In case of Overriding lines of code would be less, but because of JVM playing a role the actions will be peformed based on the runtime object.

```
class Animal
{
    public void eat()
    {
        System.out.println("Animal is eating...");
    }
    public void sleep()
    {
        System.out.println("Animal is sleeping...");
    }
    public void breathe()
    {
        System.out.println("Animal is breathing...");
    }
}
class Tiger extends Animal
{
    //informing compiler about overidden method
    @Override
    public void eat()
    {
        System.out.println("Tiger hunts and eat...");
    }
}
class Deer extends Animal
{
    //informing compiler about overidden method
    @Override
    public void eat()
    {
        System.out.println("Deer will graze and eat...");
    }
}
class Monkey extends Animal
```

```
{
    //informing compiler about overidden method
    @Override
    public void eat()
    {
        System.out.println("Monkey steal and eat...");
    }
}
```

//Helper class

```
class Forest
{
    /*
        RunTime Polymphism[1:M]
            = new Tiger();
        Animal ref  = new Deer();
            = new Monkey();
    */
    public void allowAnimal(Animal ref)
    {
        ref.eat();
        ref.sleep();
        ref.breathe();
        System.out.println();
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Tiger t = new Tiger();
        Deer d = new Deer();
        Monkey m = new Monkey();
        Forest f = new Forest();
        f.allowAnimal(t);
        f.allowAnimal(d);
        f.allowAnimal(m);
    }
}
```

Output

Tiger hunts and eat...

Animal is sleeping...

Animal is breathing...

Deer will graze and eat...

Animal is sleeping...

Animal is breathing...

Monkey steal and eat...

Animal is sleeping...

Animal is breathing...

In Plane class(mentioned above inside(program to Demonstrate the usage of inheritance)) program as well we can implement same thing by adding a new method in Parent class named as carry() and in children we can override the same method in different children.

Importance of @Override -->

Since private class of a parent cannot be overridden and if we try to do so by mentioning @Override, it will lead to CE. But if we remove @Override and compile than program will be compiled successfully(that method will not be overridden but that method will be treated as a specialized method of child class.)Hence it is always a good practice to mention @Override(Rule 2 in following points)

Rule1:

In case of MethodOverriding, method signature(Method Signature is the combination of a method's name and its parameter list.) should be same in child class while overriding.

It is possible to change the return type also if it is of reference type[Relationship should be "IS-A"(inheritance)].

If the return type is of primitive type, then we can't change the returntype, if we try to change it would result in "CE".

eg#1.

```
class Parent
{
    public Object methodOne(){
        return null;
    }
}
class Child extends Parent
{
```

```
    @Override
    public String methodOne(){
        return "sachin";
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Parent p = new Child();
        System.out.println(p.methodOne()); //sachin
    }
}
```

eg#2.

```
class Parent
{
    public Number methodOne(){
        return null;
    }
}
class Child extends Parent
{
    @Override
    public Integer methodOne(){
        return 10;
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Parent p = new Child();
        System.out.println(p.methodOne());//10
    }
}
```

eg#3.

```
class Parent
{
    public String methodOne(){
        return null;
    }
}
class Child extends Parent
{
    @Override
    public Object methodOne(){    //CE
        return 10;
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Parent p = new Child();
        System.out.println(p.methodOne());
    }
}
```

eg#4.

```
class Parent
{
    public int methodOne(){
        return 10;
    }
}
class Child extends Parent
{
    @Override
    public void methodOne(){
        System.out.println("sachin");
```

```
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Parent p = new Child();
        p.methodOne();
    }
}
```

Rule2:

Private methods are not visible in child class, so Overriding them in the child class is not possible.

eg#1.

```
class Parent
{
    private void methodOne(){
        System.out.println("From Parent...");
    }
}
class Child extends Parent
{
    @Override
    private void methodOne(){
        System.out.println("From Child...");
    }
}
```

Output :CE

eg2.

```
class Parent
{
    private void methodOne(){
        System.out.println("From Parent...");
    }
}
```

```
class Child extends Parent
{
    private void methodOne(){
        System.out.println("From Child...");
    }
}
```

Even though the above code would run, still the method present in Child class is not overridden method it is a specialized private method under child class.

Note: **final access modifier can be applied to**

a. class :: These classes won't participate in inheritance.(this is required because some classes are helper classes whose properties we can directly use(So we cannot inherit and use it on our classes)(eg: String is a final class(if it is not final than someone can inherit the properties and overwrite them and say he has created those methods from scratch hence final is used.)))

b. method :: These methods implementation can't be changed , but it will be inherited to child class.

c. variable :: it would be treated as compile time constants whose value should not be changed during the execution.

eg#1.

```
class Parent
{
    public final void methodOne(){
        System.out.println("From Parent...");
    }
}
class Child extends Parent
{
}
public class Test
{
    public static void main(String[] args)
    {
        Parent p = new Child();
        p.methodOne();
    }
}
```

Output

From Parent...

Rule3:

1. Parent class final methods can't be changed to non-final in child class during overriding.
2. Parent class non-final methods can be made as final in child class during overriding.

eg#1.

```
class Parent
{
    public final void methodOne(){
        System.out.println("From Parent...");
    }
}
class Child extends Parent
{
    public void methodOne(){
        System.out.println("From Child...");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Parent p = new Child();
        p.methodOne();
    }
}
```

eg#2.

```
class Parent
{
    public void methodOne(){
        System.out.println("From Parent...");
    }
}
class Child extends Parent
{
    @Override
```

```
    public final void methodOne(){
        System.out.println("From Child...");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Parent p = new Child();
        p.methodOne();
    }
}
```

Rule4: In case of Overriding, we can increase the privilege of the access modifier ,if we try to decrease it would result in "CompileTimeError".

eg#1.

```
class Parent
{
    void methodOne(){
        System.out.println("From Parent...");
    }
}
class Child extends Parent
{
    @Override
    public void methodOne(){
        System.out.println("From Child...");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Parent p = new Child();
        p.methodOne();
    }
}
```

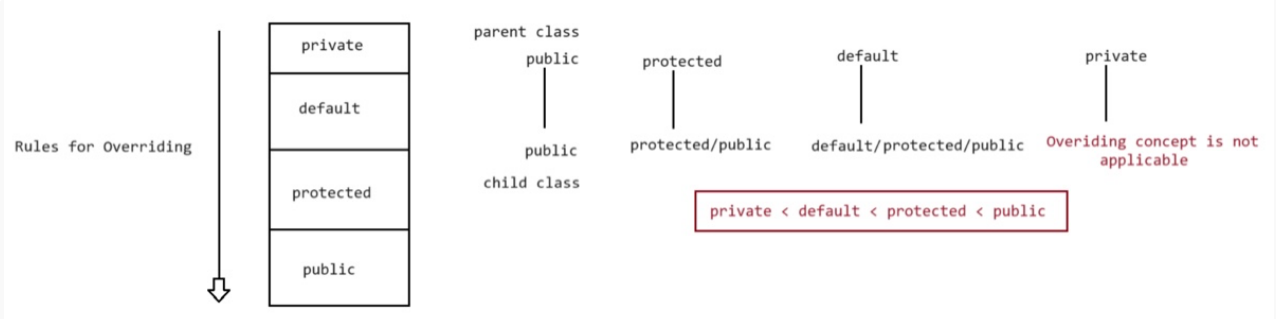
Output

From Child...

eg#2.

```
class Parent
{
    public void methodOne(){
        System.out.println("From Parent...");
    }
}
class Child extends Parent
{
    @Override
    void methodOne(){
        System.out.println("From Child...");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Parent p = new Child();
        p.methodOne();
    }
}
```

Output: CE: attempting to assign weaker access privileges; was public



Overriding w.r.t static methods

1. We can't override static method as non-static

eg#1.

```
class Parent
{
    public static void methodOne(){
        System.out.println("From Parent...");
    }
}
class Child extends Parent
{
    public void methodOne(){
        System.out.println("From Child...");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Parent p = new Child();
        p.methodOne();
    }
}
```

Output:: CE

Rule2: Non-static method can't be made static in Overriding

eg#1

```
class Parent
{
    public void methodOne(){
        System.out.println("From Parent...");
    }
}
class Child extends Parent
{
    @Override
    public static void methodOne(){
```

```
        System.out.println("From Child...");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Parent p = new Child();
        p.methodOne();
    }
}
```

Output: CE:

Rule3: static methods can't be Overriden --> (Reason from chatgpt --> The concept of method overriding is closely tied to polymorphism and the dynamic dispatch of methods during runtime. However, static methods operate at the class level rather than the instance level, and their behavior is determined at compile-time.)

eg#1.

```
class Parent
{
    public static void methodOne(){
        System.out.println("From Parent...");
    }
}
class Child extends Parent
{
    @Override
    public static void methodOne(){
        System.out.println("From Child...");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Parent p = new Child();
        p.methodOne();
```

```
    }
}
```

Output: CE

MethodHiding

In case of static methods, we assume the child class method is overridden ,but reality is it is not overridden where as it would "MethodHidden", where compiler will bind the method calls based on reference type.

eg#1.

```
class Parent
{
    public static void methodOne(){
        System.out.println("From Parent...");
    }
}
class Child extends Parent
{
    public static void methodOne(){
        System.out.println("From Child...");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Parent p = new Child();
        p.methodOne();   //From Parent...
    }
}
```

Again if I write @Overide over methodOne in Child class then it will lead to CE(as static methods cannot be overridden) otherwise it is method hiding --> Importance of mentioning @Overide

Difference b/w methodhiding and methodoverloading

MethodHiding

1. both child class and parent class methods should be static.
2. Method resoultion will be taken care by compiler based on the reference type.
3. Method hiding is considered as "static binding/early binding".

MethodOverrding

1. both child class and parent class methods should be non-static.
2. Method resoultion will be taken care by JVM based on the runtime object.
3. Method Overriding is considered as "runtime binding/late binding".

eg#1.

```
class Parent
{
    public static void methodOne(){
        System.out.println("From Parent...");
    }
}
class Child extends Parent
{
    public static void methodOne(){
        System.out.println("From Child...");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Parent p = new Parent();
        p.methodOne();//From Parent...

        Child c = new Child();
        c.methodOne();//From Child...

        Parent p1 = new Child();
        p1.methodOne();  //From Parent...   ///. ---> very very imp --> method hiding
    }
}
```

Overriding w.r.t var-args method

A var-arg method should be overriden as "var-arg" method only. if we try to override with normal method then it would become overloading but not overriding.

eg#1.

```
class Parent
{
    //var-arg method
    public void methodOne(int... i){
        System.out.println("From Parent...");
    }
}
class Child extends Parent //overloading not overriding
{
    //normal method
    public void methodOne(int i){
        System.out.println("From Child...");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Parent p = new Parent();
        p.methodOne(10);   //From Parent...

        Child c = new Child();
        c.methodOne(10);   //From Child...

        Parent p1 = new Child();
        p1.methodOne(10);   //From Parent...
    }
}
```

eg#2.

```
class Parent
{
    //var-arg method
    public void methodOne(int... i){
        System.out.println("From Parent...");
    }
}
class Child extends Parent // Overriding
```

```
{
    @Override
    public void methodOne(int... i){
        System.out.println("From Child...");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Parent p = new Parent();
        p.methodOne(10);   //From Parent...

        Child c = new Child();
        c.methodOne(10);   //From Child...

        Parent p1 = new Child();
        p1.methodOne(10);   //From Child...
    }
}
```

Overriding w.r.t variables

=> Overriding is not applicable for variables.

=> Variable resolution is always taken care by compiler based on reference type.

eg#1.

```
class Parent
{
    int x= 888;
}
class Child extends Parent
{
    int x= 999;
}
public class Test
{
    public static void main(String[] args)
```

```
    {
        Parent p = new Parent();
        System.out.println(p.x);   //888

        Child c = new Child();
        System.out.println(c.x);   //999

        Parent p1 = new Child();
        System.out.println(p1.x);   //888
    }
}
```

Question

```
class Foo {
    public int a = 3;
    public void addFive() { a += 5; System.out.print("f "); }
}
class Bar extends Foo {
    public int a = 8; //Bar :: a = 13
    public void addFive() { this.a += 5; System.out.print("b " ); }
}
```

Invoked with:

```
Foo f = new Bar();
f.addFive(); // b
System.out.println(f.a);//3
```

What is the result?

A. b 3

B. b 8

C. b 13

D. f 3

E. f 8

F. f 13

G. Compilation fails.

H. An exception is thrown at runtime.

//next 2 questions are not solved

Q>

```
class Thingy { Meter m = new Meter(); }
class Component { void go() { System.out.print("c"); } }
class Meter extends Component { void go() { System.out.print("m"); } }
class DeluxeThingy extends Thingy {
public static void main(String[] args) {
DeluxeThingy dt = new DeluxeThingy();
dt.m.go();
Thingy t = new DeluxeThingy();
t.m.go();
}
}
```

Which two are true? (Choose two.)

A. The output is mm.

B. The output is mc.

C. Component is-a Meter.

D. Component has-a Meter.

E. DeluxeThingy is-a Component.

F. DeluxeThingy has-a Component.

Q>

```
class Foo {
private int x;
public Foo( int x ){ this.x = x;}
public void setX( int x ) { this.x = x; }
public int getX(){ return x;}
}
public class Gamma {
static Foo fooBar(Foo foo) {
foo = new Foo(100);
return foo;
}
public static void main(String[] args) {
Foo foo = new Foo( 300 );
System.out.println( foo.getX() + "-");
```

```
Foo fooFoo = fooBar(foo);
System.out.println(foo.getX() + "-");
System.out.println(fooFoo.getX() + "-");
foo = fooBar( fooFoo);
System.out.println( foo.getX() + "-");
System.out.println(fooFoo.getX());
}
}
```

What is the output of the program shown in the exhibit?

A. 300-100-100-100-100

B. 300-300-100-100-100

C. 300-300-300-100-100

D. 300-300-300-300-100

In Java, when you create an object of a child class, memory is allocated for both the child class's fields and the fields inherited from its parent class. The memory for the inherited fields is part of the memory allocated for the child object. This is known as "object composition" rather than "object inheritance."

When you create an object of a child class, the constructor of both the child class and its parent class are invoked. The memory for the object includes space for the fields declared in both classes. The child class can access and modify the fields inherited from the parent class, and they are treated as part of the child object.

QUESTION What is the result?

```
public class Person
{
String name =”No name”;
public Person(String nm) { name = nm; }
}
public class Employee extends Person
{
String empID = ”0000”;
public Employee(String id) { empID = id; }  //18
}
public class EmployeeTest
{
public static void main(String[] args)
{
```

Employee e = new Employee(”4321”);

System.out.println(e.empID);

}

}

Choose the answer

A. 4321

B. 0000

C. An exception is thrown at runtime.

4. Compilation fails because of an error in line 18.

Answer: D(when child constructor is called, first parent constructor will be called and since parent constructor has parametrized constructor so we should call super manually by passing a value to it.)

QUESTION Given:

class Atom

{

Atom() { System.out.print(”atom ”); }

}

class Rock extends Atom

{

Rock(String type) { System.out.print(type); }

}

public class Mountain extends Rock

{

Mountain()

{

super(”granite”);

new Rock(”granite ”);

}

public static void main(String[] a) {

new Mountain();

}

}

What is the result?

A. Compilation fails.

B. atom granite

C. granite granite

D. atom granite granite

E. An exception is thrown at runtime.

F. atom granite

Answer: F

Given:

```
public class SuperCalc
{
protected static int multiply(int a, int b) { return a * b; }
}
public class SubCalc extends SuperCalc
{
public static int multiply(int a, int b) {
int c = super.multiply(a, b);  //22
return c;
}
}
```

and:

```
SubCalc sc = new SubCalc ();
System.out.println(sc.multiply(3,4));
System.out.println(SubCalc.multiply(2,2));
```

What is the result?

A. 12

B. The code runs with no output.

C. An exception is thrown at runtime.

D. Compilation fails because of an error in line 21.

5. Compilation fails because of an error in line 22.
6. F. Compilation fails because of an error in line 31.

Answer: E(we can't use super in static block. Super can be used in instance block)

Given:

```
public class Yikes
{
public static void go(Long n) { System.out.print(”Long”); }
public static void go(Short n) { System.out.print(”Short ”); }
```

public static void go(int n) { System.out.print(”int ”); }

public static void main(String[] args) { short y = 6; long z = 7; go(y); go(z); }

}

What is the result?

1. int Long (ans)

B. Short Long

C. Compilation fails.

D. An exception is thrown at runtime.

instance control flow in parent to child relationship

Whenever we are creating an object of child class the following sequence of events will take place

a. Identification of instance variable from Parent to Child.

b. Execution of instance variable assignments and instance block only in Parent class.

c. Execution of parent class constructor.

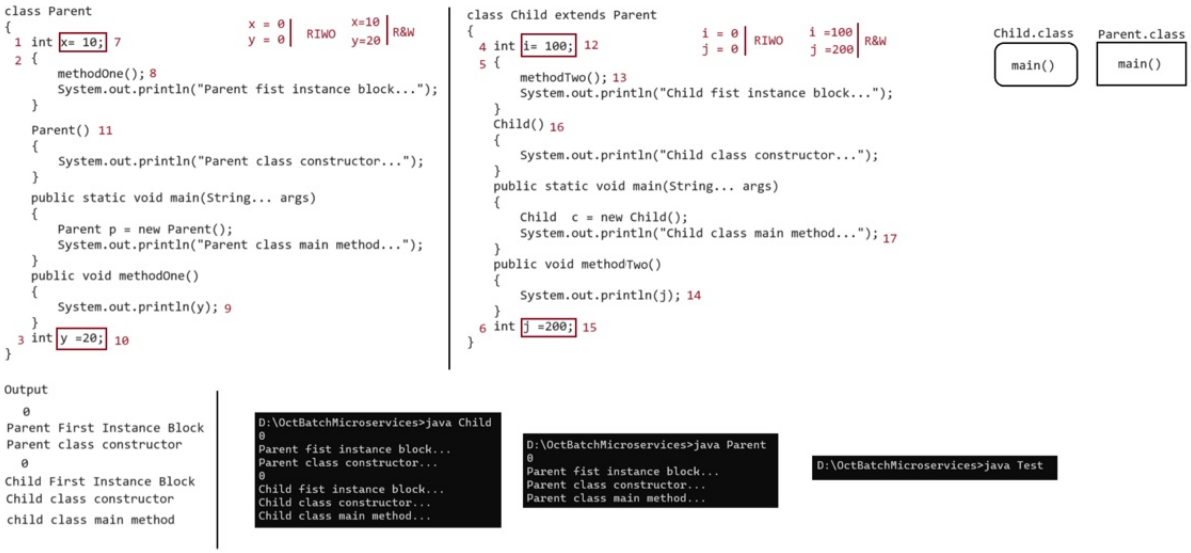
d. Execution of instance variable assignments and instance block only in child class.

e. Execution of child class constructor.

eg#1.

```
class Parent
{
    int x= 10;
    {
        methodOne();
        System.out.println("Parent fist instance block...");
    }
    Parent()
    {
        System.out.println("Parent class constructor...");
    }
    public static void main(String... args)
    {
        Parent p = new Parent();
        System.out.println("Parent class main method...");
    }
    public void methodOne()
```

```
    {
        System.out.println(y);
    }
    int y =20;
}
class Child extends Parent
{
    int i= 100;
    {
        methodTwo();
        System.out.println("Child fist instance block...");
    }
    Child()
    {
        System.out.println("Child class constructor...");
    }
    public static void main(String... args)
    {
        Child c = new Child();
        System.out.println("Child class main method...");
    }
    public void methodOne()
    {
        System.out.println(j);
    }
    int j =200;
}
public class Test
{
    public static void main(String[] args)
    {
    }
}
```



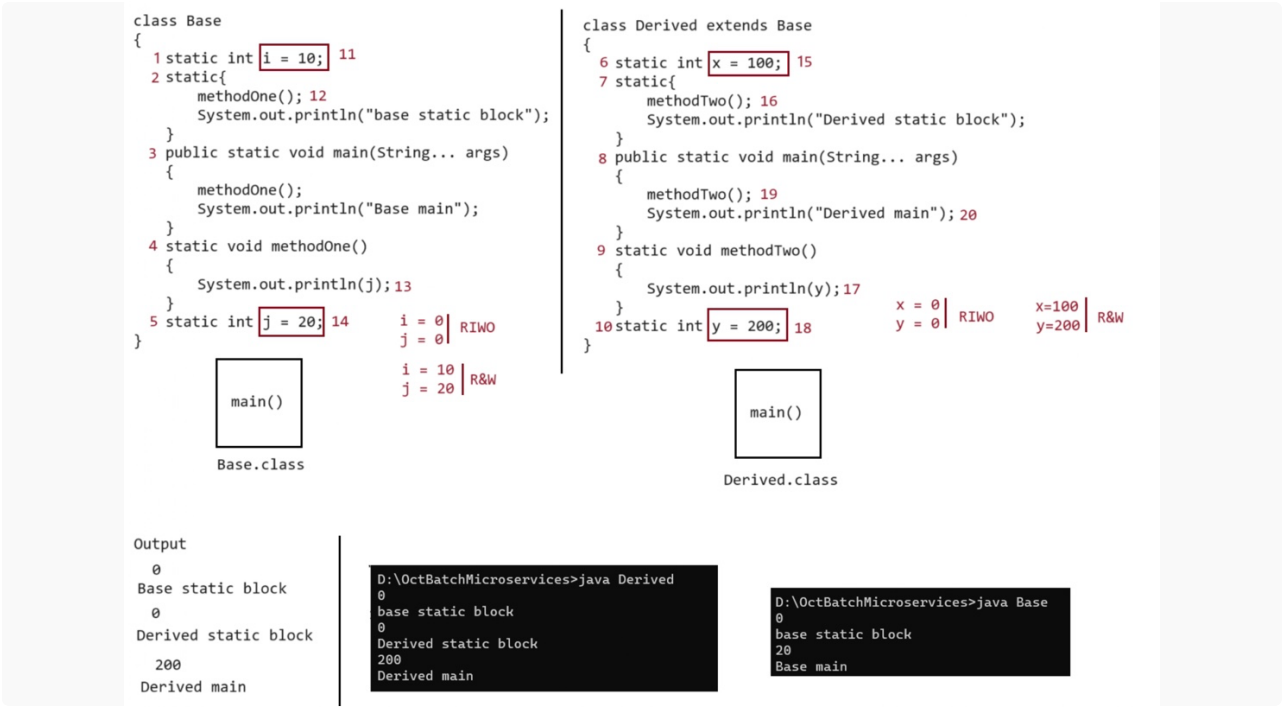
static control flow in inheritance

Whenever we are executing child class the following sequence of events takes place

a. Identification of static memebers from parent to child.

b. Execution of static variable assignments and static block execution from parent to child

c. Execution of child class main().



Creating an object everytime without its need is it a good programming practice?

Ans. No, it is not a good programming practise because to create an object

1. loading of .class file should be happen[All the activities of static control flow should happen]

b. instance control flow from parent to child should happen.

Since it is time consuming event, only when it is required we need to use "new" keyword in java.

++

added later as not included in original notes

Q>

```
public class Test {
    public static void doSum(int x, int y){
        System.out.println("int sum is:: "+(x+y));
    }
    public static void doSum(Integer x, Integer y){
        System.out.println("Integer sum is:: "+(x+y));
    }
    public static void doSum(double x, double y){
        System.out.println("double sum is:: "+(x+y));
    }
    public static void doSum(float x, float y){
        System.out.println("float sum is:: "+(x+y));
    }
public static void main(String[] args) {
doSum(10,20);
doSum(10.0,20.0);
}
}
```

What is the result?

A. int sum is :: 30

float sum is :: 30.0

B. int sum is :: 30

double sum is :: 30.0

C. Integer sum is :: 30

double sum is :: 30.0

D. Integer sum is:: 30

float sum is :: 30.0

Answer: B

Q>

Consider the following snippet and predict the output

```
public class Test{
public static void main(String... args){
String language="java";
while(language.equals("java"))
{
if(language.equals("java"))
language=language.toUpperCase();
if(language.equals("JAVA"))
language=language.toLowerCase();
}
System.out.println(language); //line n1
}
}
```

A. java

B. JAVA

C. Compile time error at line n1

D. Infintie time loop will run

E. None of the above

Answer: D

Q>

Consider below code of Test.java file:

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

What will be the result of compiling and executing Test class?

A. 112

B. 012

C. 011

D. 12

E. 01

F. CompilationError

Answer: C

Q>

Consider below code of Test.java file:

```
public class Test {
public static void main(String[] args) {
int i = 1;
int j = 5;
int k = 0;
A: while(true) {
i++;
B: while(true) {
j--;
C: while(true) {
k += i + j;
if(i == j)
break A;
else if (i > j)
continue A;
else
continue B;
}
}
}
System.out.println(k);
}
}
```

What will be the result of compiling and executing Test class?

A. Compilation Error

B. 6

C. 11

D. 15

E. Program never terminates it results in infinte loop

F. None of the above

Pillars of Object Orientation

1. Encapsulation
2. inheritance
3. Polymorphism

Datahiding => Our internal data should not go to outside world directly that is outside person can't access internal data directly.

To promote datahiding, we need to use "access modifiers"

eg: private,protected

It promotes security.

eg#1.

```
class Account
{
    private double balance;
}
```

Note: To access the balance variable data from the enduser, validation(collect username,password) will be performed by the application.

Abstraction

Hiding internal implementation,but exposing the set of services is called "Abstraction".

In java to bring abstraction, we use "abstract classes and interfaces"

eg: ATM GUI Screen

BankPeople -> Highlight the set of services they are offering.

EndUser -> Using the offered services, the end user will use the application.

Encapsulation

Binding of data and corresponding methods into single unit is called "Encapsulation".

If any java class follows datahiding and abstraction such type of class is said to be "Encapsulated class".

Encapsulation = Datahiding + abstraction.

eg#1.

```
//Encapsulated class
class TextBook
{
```

```
    //instance variable : encapuslated
    private int pages;
    //Setter method
    public void setPages(int pages)
    {
        if(pages > 0)
            this.pages = pages;
        else
            this.pages = 0;
}
//Getter method
    public int getPages()
    {
        return pages;
    }
}
class Test{
    public static void main(String[] args) {
        TextBook tb = new TextBook();
        tb.setPages(-100);
        int pageCount= tb.getPages();
        System.out.println("No of pages is :: "+pageCount);
    }
}
```

Syntax for Setter methods

1. MethodName should be prefixed with set
2. It should be public
3. return type should be void
4. Compulsorily it should take an argument

```
public void setXXXXX(XXXXX varaibleName)
{
    this.variableName = variableName;
}
```

Syntax for getter methods

1. MethodName should be prefixed with get

2. It should be public
3. Return type should not be void
4. it is alwasy no argument method.

```
public XXXXXX getXXXXX()
{
    return variableName;
}
```

Note: If the variable type is boolean, then for getter method the name as per the convention is "isVaraibleName()"

eg#1.

```
//Encapsulated class
class Doctor
{
    private String sname;
    private boolean married;
    public void setSname(String sname)
    {
        this.sname = sname;
    }
    public void setMarried(boolean married)
    {
        this.married =married;
    }
    public String getSname()
    {
        return sname;
    }
    public boolean isMarried()
    {
        return married;
    }
}
class Test{
    public static void main(String[] args) {
        Doctor d= new Doctor();
        d.setSname("karthik");
        d.setMarried(true);
```

```
        boolean status = d.isMarried();
        String name = d.getSname();
        System.out.println("Name is : " +name);
        System.out.println("Status is : " +status);
    }
}
```

Output

Name is : karthik

Status is : true

Need of "this" keyword in java

```
class Student
{
    String name;
    int age;
    float height;
    public void setData(String name, int age, float height)
    {
        name=name;
        age=age;
        height=height;
    }
    public void displayData()
    {
        System.out.println("Name is : "+name);
        System.out.println("Age is : "+age);
        System.out.println("Height is : "+height);
    }
}
class Test
{
    public static void main(String[] args)
    {
        Student std = new Student();
        std.setData("sachin",49,5.5f);
        std.displayData();
    }
```

```
}
```

Output

Name is : null

Age is : 0

Height is : 0.0

As noticed in the above code, the method setData() would set the supplied value to the variables called name,age,height.

These supplied values are not been assigned to instance variables by jvm because "jvm by default in method will always" give priority to "local varaibles" but not the instance variables. this problem is technically termed as "**Shadowing**". **To resolve this problem we use "this" keyword in java.**

Solution using this keyword

```
class Student
{
    String name;
    int age;
    float height;
    public void setData(String name, int age, float height)
    {
        //we can use "this" to refer to Object
        this.name=name;
        this.age=age;
        this.height=height;
    }
    public void displayData()
    {
        System.out.println("Name is : "+this.name);
        System.out.println("Age is : "+this.age);
        System.out.println("Height is : "+this.height);
    }
}
class Test
{
    public static void main(String[] args)
    {
        Student std = new Student();
```

```
        std.setData("sachin",49,5.5f);
        std.displayData();
    }
}
```

Output

Name is : sachin

Age is : 49

Height is : 5.5

Dependancy Injection

=> It refers to process of injecting the values to the instance variables of a class.

=> we can perform depedancy injection in 2 ways

- **a. through setter method.**
- **b. through constructor**

Dependancy injection using setter method

eg#1.

```
class Student
{
    //instance variables
    private String name;
    private int age;
    private float height;
    //setter methods
    public void setName(String name){
        this.name = name;
    }
    public void setAge(int age){
        this.age = age;
    }
    public void setHeight(float height){
        this.height = height;
    }
    //getter methods
    public String getName(){
        return name;
    }
```

```
    public int getAge(){
        return age;
    }
    public float getHeight(){
        return height;
    }
}
class Test
{
    public static void main(String[] args)
    {
        //Constructing the object
        Student std = new Student();
        //setting the values for instance variable
        std.setName("sachin");
        std.setAge(49);
        std.setHeight(5.5f);
        //getting the values from instance variables
        System.out.println("Name is :: "+std.getName());
        System.out.println("Age is :: "+std.getAge());
        System.out.println("Height is :: "+std.getHeight());
    }
}
```

Output

Name is :: sachin

Age is :: 49

Height is :: 5.5

++

Pillars of oops

a. Encapsulation => Datahiding(private, setXXXX(),getXXXX()) + abstraction[abstract,interface]

b. Inheritance => ReUsablitiy(IS-A,HAS-A)

c. Polymorphism => Code Flexiblity(Overloading,Overriding, MethodHiding)

What is TightCoupling and What is Loose Coupling?

TightCoupling(if i use Tiger t as parameter than i will not able to pass any other animal to it other than Tiger hence it is tightly coupled)

eg#1.

```
class Forest {
    //Method Over-loading :: Tight Coupling
    public void allowAnimal(Tiger t)
    {
        t.eat();
        t.sleep();
        t.breathe();
    }
    public void allowAnimal(Deer d)
    {
        d.eat();
        d.sleep();
        d.breathe();
    }
    public void allowAnimal(Monkey m)
    {
        m.eat();
        m.sleep();
        m.breathe();
    }
}
```

Loose Coupling

eg#1.

//Helper class

```
class Forest {
    /* RunTime Polymphism[1:M] = new Tiger(); Animal ref = new Deer(); = new Monkey(); */
    //Method-Overriding :: LooseCoupling
    public void allowAnimal(Animal ref)
    {
        ref.eat();
        ref.sleep();
        ref.breathe();
        System.out.println();
    }
}
```

Abstract class in java

=> If we don't want an object to be created for a particular class, then such class we need to mark as " abstract".

=> abstract access modifer is applicable at

a. class level : prevents object creation.

b. method level : prevents giving the implementation for body of a method.

c. variable level : not applicable at variable level.

Through abstract keyword we can promote "abstraction".

=> By referring to abstract class, we would get to know only the services name(methodnames),but not the internal implementation given by developers, this mechanism only we call it as " abstraction".

=> for an abstract class,"instantiation is not possible",but we can create a reference for an " abstract class".

=> If a class contains only one abstract methods also,then we need to mark the class as "abstract".

eg#1.

```
//Parent class normally should be abstract class with abstract methods.
abstract class Plane
{
    //abstract methods : Method without implementation
    public abstract void takeOff();
    public abstract void fly();
    public abstract void land();
    public abstract void carry();
}
//Child class
final class Passenger extends Plane
{
    //Specific implementation
    @Override public void takeOff()
    {
        System.out.println("Passenger plane take off...");
    }
    @Override public void fly()
    {
        System.out.println("Passenger plane is flying...");
    }
    @Override public void land()
```

```
    {
        System.out.println("Passenger plane is landing...");
    }
    @Override public void carry()
    {
        System.out.println("Passenger plane is carrying passengers...");
    }
}
//Child class
final class Cargo extends Plane
{
    //Specific implementation
    @Override public void takeOff()
    {
        System.out.println("Cargo plane is carrying...");
    }
    @Override public void fly()
    {
        System.out.println("Cargo plane is flying...");
    }
    @Override public void land()
    {
        System.out.println("Cargo plane is landing...");
    }
    @Override public void carry()
    {
        System.out.println("Cargo plane is carrying goods...");
    }
}
//Child class
final class Fighter extends Plane
{
    //Specific implementation
    @Override public void takeOff()
    {
        System.out.println("Fighter plane is taking off...");
    }
    @Override public void fly()
```

```
    {
        System.out.println("Fighter plane is flying...");
    }
    @Override public void land()
    {
        System.out.println("Fighter plane is landing...");
    }
    @Override public void carry()
    {
        System.out.println("Fighter plane is carrying weapons..");
    }
}
//Helper class
final class Airport
{
    /*MethodOverriding :True Poly-morphsim[1:M] => Loose Coupling = new Cargo(); Plane ref = new
    Passenger(); = new Fighter(); */
    public void allowPlane(Plane ref)
    {
        System.out.println("Working with object called :: "+ref.getClass().getName());
        ref.takeOff();
        ref.carry();
        ref.fly();
        ref.land();
        System.out.println();
    }
}
Public class Test
{
    public static void main(String[] args)
    {
        //Creating 3 objects of
        Plane Type Cargo c = new Cargo();
        Passenger p =new Passenger();
        Fighter f = new Fighter();
        //Taking the actions for all the 3 planes
        Airport a = new Airport();
        a.allowPlane(c);
```

```
        a.allowPlane(p);
        a.allowPlane(f);
    }
}
```

Output

Working with object called :: Cargo

Cargo plane is carrying...

Cargo plane is carrying goods...

Cargo plane is flying...

Cargo plane is landing...

Working with object called :: Passenger

Passenger plane take off...

Passenger plane is carrying passengers...

Passenger plane is flying...

Passenger plane is landing...

Working with object called :: Fighter

Fighter plane is taking off...

Fighter plane is carrying weapons..

Fighter plane is flying...

Fighter plane is landing...

In java abstraction shows incompleteness

=> Eg. If we want to override the methods of parent to child than that method is also defined in parent class which is of no use as Plane doesnt exist only its type passengers plane exist so we are creating method implemention in parent which is of no use other that overriding.

=> So what if we just declare that method in parent class and then write its implementation in child class with overriding. So it will be a incomplete method as it doesnt have method defination. So the class that consist such methods should also me incomplete as after creating the object of the parent incomplete class we cannot access its methods as they are incomplete.

=> So if a class that consist of incomplete or abstract methods are abstract class. So we need to add keyword to both methods and class.

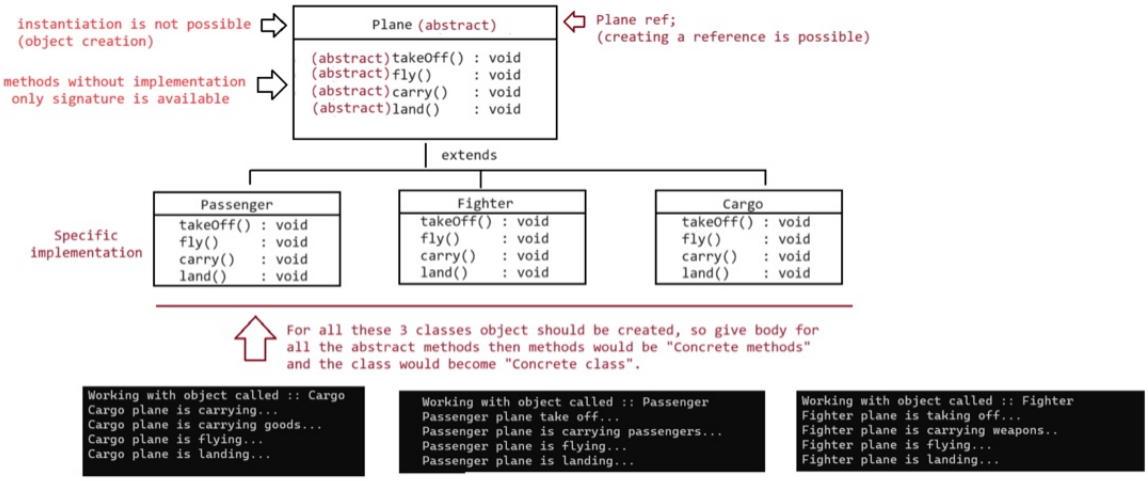
=> So rule --> we cannot make object of abstract class but we can use it as a refrence. Which makes sence as if we create object of abstract class than there is no way we can access its methods as they are also abstract or incomplete.

=> The methods with implementation are called concrete methods and the classes with concrete methods are called concrete classes(normal classes whose object can be created.)

=> Abstraction is not applicable at variable level as abstract means incompleteness and variable represents a memory location which cannot be incomplete logically

=> Now a passanger plane inherited the properties from plane abstract class. **Now if i want that no other class should inherit the properties of passanger plane than we can make it a final class.**

=> Can parent class be final --> no, then no further class can inherit its properties and define implemention for abstract methods. Then that class will be of no use.



Given:

11. abstract class Vehicle { public int speed() { return 0; }
12. class Car extends Vehicle { public int speed() { return 60; }
13. class RaceCar extends Car { public int speed() { return 150; } ...
21. RaceCar racer = new RaceCar();
22. Car car = new RaceCar();

23 Vehicle vehicle = new RaceCar();

24 System.out.println(racer.speed() + ", " + car.speed() + ", " + vehicle.speed());

What is the result?

A. 0, 0, 0

B. 150, 60, 0

C. Compilation fails.

D. 150, 150, 150

E. An exception is thrown at runtime.

Answer: D

Note:

1. An abstract class can contain "concrete methods" also along with abstract methods.
2. During inheritance, concrete methods can be "Overriden".
3. An abstract class need not have any abstract methods also.
4. For an abstract class object can't be created, only reference can be created.

eg#1.

```
abstract class Vehicle
{
    public abstract int getNoOfWheels();
    public void infoAboutVehicle()
    {
        System.out.println("Generic information...");
    }
}
class Bus extends Vehicle
{
    @Override
    public int getNoOfWheels()
    {
        return 7;
    }
    @Override
    public void infoAboutVehicle()
    {
        System.out.println("Volvo bus...");
    }
}
class Auto extends Vehicle
{
    @Override
    public int getNoOfWheels()
    {
        return 3;
    }
    @Override
    public void infoAboutVehicle()
```

```
    {
        System.out.println("Uber Auto...");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        //Vehicle v = new Vehicle(); //instantiation of Vehicle is not possible : abstract
        Vehicle v1 = new Bus();
        v1.infoAboutVehicle();
        System.out.println("No of wheels for Bus is :: "+v1.getNoOfWheels());
        Vehicle v2 = new Auto();
        v2.infoAboutVehicle();
        System.out.println("No of wheels for Auto is :: "+v2.getNoOfWheels());
    }
}
```

Output

Volvo bus...

No of wheels for Bus is :: 7

Uber Auto...

No of wheels for Auto is :: 3

eg#2.

```
import java.util.Scanner;
abstract class Shapee
{
    public float area;

    //abstract methods
    public abstract void input();
    public abstract void calcArea();

    //concrete methods
    public void disp()
    {
        System.out.println("Area is :: "+area);
```

```
    }
}
class Square extends Shapee
{
    private float length;
    @Override
    public void input()
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the length for Square object:: ");
        length = scanner.nextFloat();
    }
    @Override
    public void calcArea()
    {
        area=length*length;
    }
}
class Rectanglee extends Shapee
{
    private float length;
    private float breadth;
    @Override
    public void input()
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the length of Rectangleee object:: ");
        length = scanner.nextFloat();
        System.out.print("Enter the breadth of Rectanglee object:: ");
        breadth = scanner.nextFloat();
    }
    @Override
    public void calcArea()
    {
        area=length*breadth;
    }
}
class Circle extends Shapee
```

```
{
    private float radius;
    @Override
    public void input()
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the radius of circlee:: ");
        radius = scanner.nextFloat();
    }
    @Override
    public void calcArea()
    {
        area=3.1414f * radius *radius;
    }
}
public class Test
{
    public static void main(String[] args)
    {
        //Creating a reference variable for Shape
        Shapee s;
        //Working with Square Object
        s = new Square();
        s.input();
        s.calcArea();
        s.disp();   //generic method

        System.out.println();

        //Working with Rectangle Object
        s = new Rectanglee();
        s.input();
        s.calcArea();
        s.disp();//generic method

        System.out.println();

        //Working with Circle Object
```

```
            s = new Circle();
            s.input();
            s.calcArea();
            s.disp();  //generic method
        }
}
```

Output

Enter the length for Square object:: 3

Area is :: 9.0

Enter the length of Rectangleee object:: 5

Enter the breadth of Rectanglee object:: 6

Area is :: 30.0

Enter the radius of circlee:: 4

Area is :: 50.2624

eg#2.

```
import java.util.Scanner;
abstract class Bird
{
    public abstract void fly();
    public abstract void eat();
}
final class Sparrow extends Bird
{
    @Override
    public void fly()
    {
        System.out.println("Fly very fast...");
    }
    @Override
    public void eat()
    {
        System.out.println("Sparrows eats grains...");
    }
}
abstract class Eagle extends Bird
```

```
{
    @Override
    public void fly()
    {
        System.out.println("All Eagles Fly very very high...");
    }
    public abstract void eat();
}
final class SerpentEagle extends Eagle
{
    @Override
    public void eat(){
        System.out.println("SerpentEagles eats snakes...");
    }
}
final class GoldenEagle extends Eagle
{
    @Override
    public void eat(){
        System.out.println("GoldeEagles catches the prey over the ocean...");
    }
}
final class Crow extends Bird
{
    @Override
    public void fly()
    {
        System.out.println("Crow fly at Meidum Height...");
    }
    @Override
    public void eat()
    {
        System.out.println("Crow eat Flesh...");
    }
}

//HelperClass
```

```
abstract class Sky
{
    /*
        Polymorphism : Overriding(1:M)
        = new Sparrow();
        Bird ref = new SerpentEagle();
        = new goldenEagle();
        = new Crow();
    */
    public static void allowBird(Bird ref)
    {
        System.out.println("Working with object called::"+ref.getClass().getName());
        ref.fly();
        ref.eat();
        System.out.println();
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Sparrow sp = new Sparrow();
        Eagle e1;
        Eagle e2;
        e1 = new SerpentEagle();
        e2 = new GoldenEagle();
        Crow c = new Crow();
        Sky.allowBird(sp);
        Sky.allowBird(e1);
        Sky.allowBird(e2);
        Sky.allowBird(c);
    }
}
```

Output

Working with object called::Sparrow

Fly very fast...

Sparrows eats grains...

Working with object called::SerpentEagle

All Eagles Fly very very high...

SerpentEagles eats snakes...

Working with object called::GoldenEagle

All Eagles Fly very very high...

GoldeEagles catches the prey over the ocean...

Working with object called::Crow

Crow fly at Meidum Height...

Crow eat Flesh...

Note: Illegal combinations of access modifier at methods level

a. abstract and final -------> [Illegal]

b. abstract and static -------> [Illegal]

Question.

1. Will constructor be called at the time of Object creation?
   Ans. yes
2. Can we create an object for abstract class?
   Ans. No.
3. Does abstract class have constructor?
   Ans. yes.
4. If object can't be created for an abstract class, then why do we need a constructor?
   Ans. Even though we can't create an object for abstract class, still constructor is required, because in inheritance the child class object will be initialized by making a call to parent class constructor. constructor in abstract class is required to initialize the object completely.

```
//Concrete class
class Student extends Person
{
    int sid;
    float avg;
    Student(String name,int age,char gender,int sid, float avg)
    {
        super(name,age,gender);
        this.sid = sid;
        this.avg = avg;
```

```
    }
    public void dispDetails(){
        super.dispDetails();
        System.out.println("SID is :: "+sid);
        System.out.println("AVG is :: "+avg);
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Person p;
        p = new Student("sachin",51,'M',10,57.5f);
        p.dispDetails();
    }
}
```

Output

Name is :: sachin

Age is :: 51

Gender is :: M

SID is :: 10

AVG is :: 57.5

5. **When we create an object of child class, will object of parent class also be created?**

Ans. No, parent class constructor will be executed to make the child object complete(initialized).

eg#1.

```
//Person class
abstract class Person
{
    String name;
    int age;
    char gender;
    Person(String name,int age, char gender)
    {
        System.out.println("HashCode is :: "+this.hashCode());
```

```
        this.name =name;
        this.age = age;
        this.gender =gender;
    }
    public void dispDetails(){
        System.out.println("Name is :: "+name);
        System.out.println("Age is :: "+age);
        System.out.println("Gender is :: "+gender);
    }
}
//Concrete class
class Student extends Person
{
    int sid;
    float avg;
    Student(String name,int age,char gender,int sid, float avg)
    {
        super(name,age,gender);
        System.out.println("HashCode is :: "+this.hashCode());
        this.sid = sid;
        this.avg = avg;
    }
    public void dispDetails(){
        super.dispDetails();
        System.out.println("SID is :: "+sid);
        System.out.println("AVG is :: "+avg);
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Person p;
        p = new Student("sachin",51,'M',10,57.5f);
        System.out.println("HashCode is :: "+p.hashCode());
        p.dispDetails();
    }
}
```

Output

HashCode is :: 366712642

HashCode is :: 366712642

HashCode is :: 366712642

Name is :: sachin

Age is :: 51

Gender is :: M

SID is :: 10

AVG is :: 57.5

NOT SOLVED-->

Q>

Consider below code of Test.java file:

```
public class Test {
public static void main(String[] args) {
for(int x = 10, y = 11, z = 12; y > x && z > y; y++, z -= 2) {
System.out.println(x + y + z);
}
}
}
```

What will be the result of compiling and executing Test class?

A. 33

B. 32

C. 34

D. 33

5. 32

F. CompilationError

Ans --> 33

Interfaces in java

Definition-1

=> Any Service Requirement Specification is called "interface".

eg#1. SunMS is responsible to define JDBC API and Database vendors is responsible to provide implementation to it.

Definition-2

=> It refers to the contract b/w client and the service provider

eg#2. ATM GUI screen describes what bank people are offering,at the same time GUI SCREEN represents what customer is expecting So GUI screen acts like a bridge b/w client and the service provider.

Definition-3

=> Inside interface every method present are "public and abstract".

=> Since it is public and abstract,we say interface as "pure abstract class".

eg#1.

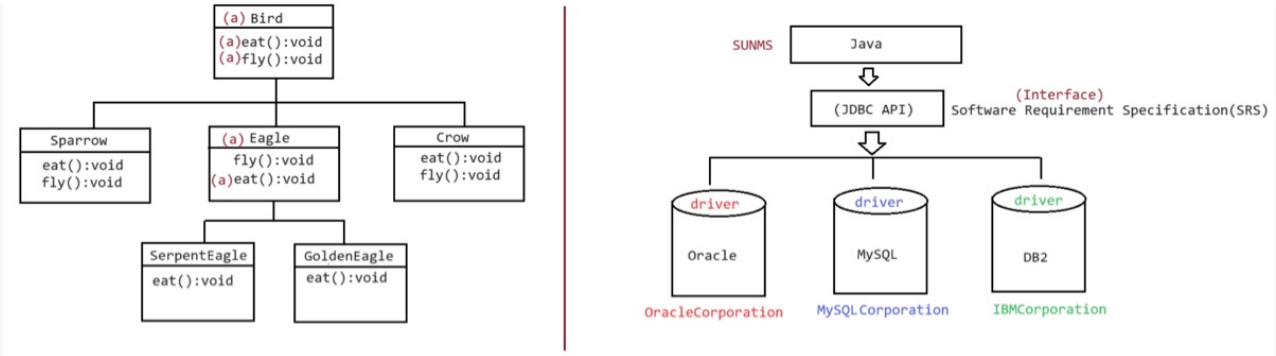
```
interface Calculator
{
    //By default all the methods are public and abstract.
    void add(int a,int b);
    void sub(int a,int b);
    void mul(int a,int b);
    void div(int a,int b);
}
```

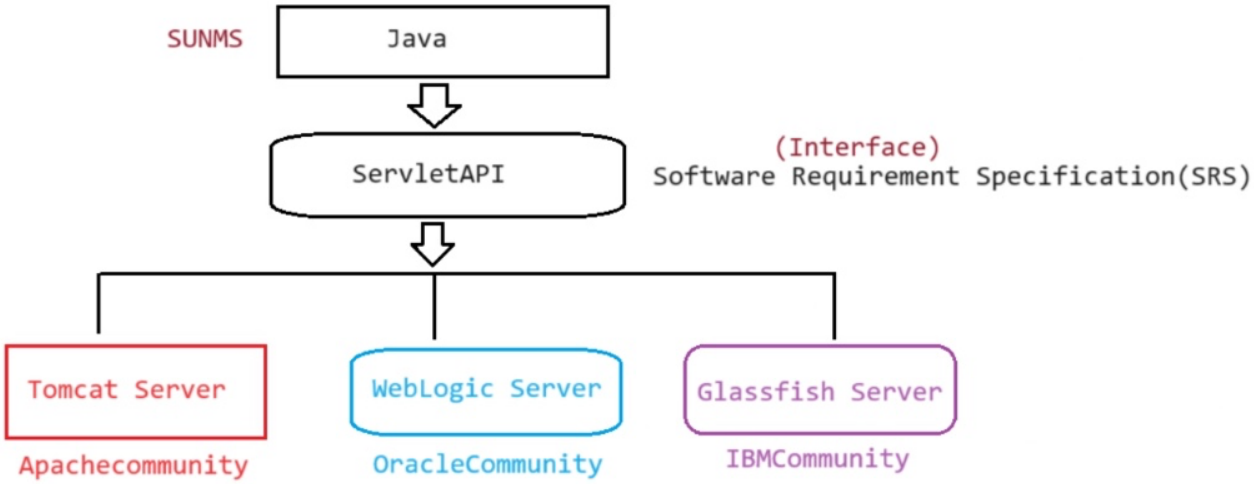
Encapsulation => Datahiding + abstraction

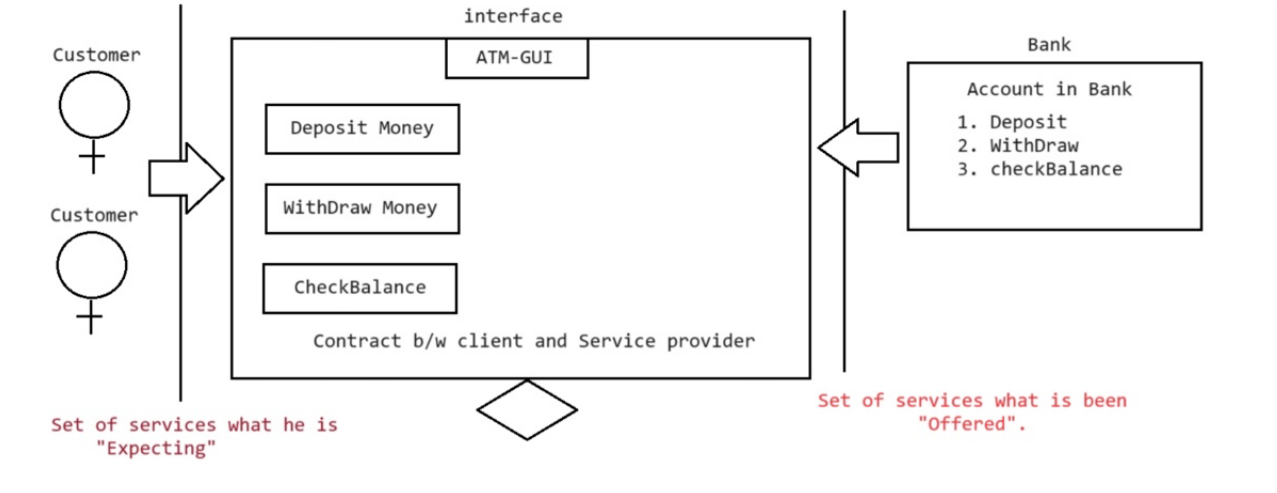
(private) (interface & abstract class)

|

setter&getter







Eg 4

9 dec--> 2:3 to 2:4

Very very imp--> helpers class-->

Helper class are the classes which are required to reduce the redundant code. So we pass the refrence to this class and than call respective methods.

The refrence is of parent type in helper's class parameter to make the code general for all types.

Now when we call this class we make an object and than call it.

Cant we make it abstract?

Ans - but than we will not be able to make its object and access it. Than no problem mark the methods of this class as static and then we can directly access it with class name. In this way we can avoid object creation as well which is a costly process--> very important for optimisation.

At method level using abstract and static(both) is illegal because we can access that method with class name and since its implementation is not there hence illegal.

Interface is called "pure abstract class" because in interface we will have only abstract methods no concrete methods are allowed while in abstract class we can have both.

Rules of Interface

case 1:

1. **Whenever we are implementing an interface, we need to give body for all the abstract methods present inside the interface.If we fail to give body for all the methods present inside the interface, then we need to mark the class as "abstract".**
2. For an interface instantiation(creation of object) is not possible.
3. For an interface, creation of reference is possible.
4. **Through interface we achieve :: TruePolymorphism(Overriding).[The key to achieving true polymorphism here is that at compile time, Java only knows that you have a reference variable of type `Shape`. However, when you call `calculateArea()`, the actual method invoked depends on the object's type at runtime (either `Square` or `Circle` object). This allows for flexible and dynamic behavior based on the actual object instance.]**

JDK1.7 Version : interface

```
//Pure-Abstract-class : SRS[Software Requirment Specification]
interface ICalculator
{
    //By Default methods are :: public abstract
    void add(int a,int b);
    void sub(int a,int b);
    void mul(int a,int b);
    void div(int a,int b);
}
//Implemented class : concrete class
class CalculatorImpl implements ICalculator
{
    @Override
    public void add(int a,int b)
    {
        int sum = a+b;
        System.out.println("The sum is :: "+sum);
    }
    @Override
```

```
    public void sub(int a,int b)
    {
        int diff = a-b;
        System.out.println("The sub is :: "+diff);
    }
    @Override
    public void mul(int a,int b)
    {
        int res = a*b;
        System.out.println("The res is :: "+res);
    }
    @Override
    public void div(int a,int b)
    {
        int quotient = a/b;
        System.out.println("The Quotient is :: "+quotient);
    }
}
public class Test
{
    public static void main(String[] args)
    {
        //Creating a reference of interface
        ICalculator calc;
        calc = new CalculatorImpl();
        //Calling the method based on runtime object
        calc.add(10,20);
        calc.sub(20,10);
        calc.mul(10,20);
        calc.div(5,2);
    }
}
```

Output

The sum is :: 30

The sub is :: 10

The res is :: 200

The Quotient is :: 2

Given

```
1.  public class KungFu {
2.  public static void main(String[] args) {
3.  Integer x = 400;
4.  Integer y = x;
5.  x++;
6.  StringBuilder sb1 = new StringBuilder("123");
7.  StringBuilder sb2 = sb1;
8.  sb1.append("5");
9.  System.out.println((x  y) + " " + (sb1  sb2));
10. }
11.}
```

What is the result?

A. true true

B. false true

C. true false

D. false false

E. Compilation fails.

F. An exception is thrown at runtime.

Answer: B

Case2: Whenever we are implementing an interface method compulsory, it should be declared as public otherwise we will get "CompileTime Error"**(rule where we cannot assign weaker access privileges during overriding).**

```
//Pure-Abstract-class : SRS
interface ICalculator
{
    //By Default methods are :: public abstract
    void add(int a,int b);
    void sub(int a,int b);
}
//Implemented class : concrete class
class CalculatorImpl implements ICalculator
{
    @Override
```

```
    void add(int a,int b)
    {
        int sum = a+b;
        System.out.println("The sum is :: "+sum);
    }
    @Override
    void sub(int a,int b)
    {
        int diff = a-b;
        System.out.println("The sub is :: "+diff);
    }
}
public class Test
{
    public static void main(String[] args)
    {
        //Creating a reference of interface
        ICalculator calc;
        calc = new CalculatorImpl();
        //Calling the method based on runtime object
        calc.add(10,20);
        calc.sub(20,10);
    }
}
```

Output

CE: attempting to assign weaker access privileges; was public

case3:

=> Relationship b/w interface to class is always "implements".

=> Relationship b/w interface to interface is always "extends".

=> If we implemented the interface which has extended from one more interface, then as a programmer the implementation class should give body for all the abstract methods present in the interface, if not we need to mark the class as "abstract", otherwise the code would result in "CompileTime Error".

eg#1.

//Pure-Abstract-class : SRS

```
interface ICalculator1
{
    //By Default methods are :: public abstract
    void add(int a,int b);
    void sub(int a,int b);
}
//Pure-Abstract-class : SRS
interface ICalculator2 extends ICalculator1
{
    //By Default methods are :: public abstract
    void mul(int a,int b);
    void div(int a,int b);
}
//Implemented class : concrete class
class CalculatorImpl implements ICalculator2
{
    @Override
    public void add(int a,int b)
    {
        int sum = a+b;
        System.out.println("The sum is :: "+sum);
    }
    @Override
    public void sub(int a,int b)
    {
        int diff = a-b;
        System.out.println("The sub is :: "+diff);
    }
    @Override
    public void mul(int a,int b){
        int res = a*b;
        System.out.println("The res is :: "+res);
    }
    @Override
    public void div(int a,int b){
        int quotient = a/b;
        System.out.println("The quotient is :: "+quotient);
    }
```

```
}
public class Test
{
    public static void main(String[] args)
    {
        //Creating a reference of interface
        ICalculator2 calc;
        calc = new CalculatorImpl();
        //Calling the method based on runtime object
        calc.add(10,20);
        calc.sub(20,10);
        calc.mul(10,20);
        calc.div(5,2);
    }
}
```

Output

The sum is :: 30

The sub is :: 10

The res is :: 200

The quotient is :: 2

case4:

At a time one class can extend from how many classes?

Answer. One because java doesn't support multiple inheritance through class to avoid "Ambiguity problem".

At a time one class can implement multiple interfaces?

Answer: Yes possible, so we can say mulitple inheritance is supported in java through "interfaces" and "Ambiguity problem " won't occur because Compiler will keep the method signature in the implementation class only if it is not available.

As noticed in the below example ICalculator1 and ICalculator2 both have void add(int a,int b) method,but compiler will keep only one method void add(int a,int b) in the implementation class through which "Ambiguity problem" will not occur in interfaces.

At a time can one class implement an interface and extends a class?

Answer: yes,but first we need to have extends and followed by implements.

eg#1.

```
//Pure-Abstract-class : SRS
interface ICalculator1
{
    //By Default methods are :: public abstract
    void add(int a,int b);
    void sub(int a,int b);
}
//Pure-Abstract-class : SRS
interface ICalculator2
{
    //By Default methods are :: public abstract
    void mul(int a,int b);
    void div(int a,int b);
    void add(int a,int b);
}
//Implemented class : concrete class
class CalculatorImpl implements ICalculator1,ICalculator2
{
    @Override
    public void add(int a,int b)
    {
        int sum = a+b;
        System.out.println("The sum is :: "+sum);
    }
    @Override
    public void sub(int a,int b)
    {
        int diff = a-b;
        System.out.println("The sub is :: "+diff);
    }
    @Override
    public void mul(int a,int b){
        int res = a*b;
        System.out.println("The res is :: "+res);
    }
    @Override
    public void div(int a,int b){
```

```
        int quotient = a/b;
        System.out.println("The quotient is :: "+quotient);
    }
}
public class Test
{
    public static void main(String[] args)
    {
        //Creating a reference of interface
        CalculatorImpl calc;
        calc = new CalculatorImpl();
        //Calling the method based on runtime object
        calc.add(10,20);
        calc.sub(20,10);
        calc.mul(10,20);
        calc.div(5,2);
    }
}
```

Output

The sum is :: 30

The sub is :: 10

The res is :: 200

The quotient is :: 2

To promote loose coupling we follow the rule of

interface ->abstract class -> class

```
//Pure-Abstract-class : SRS
interface ICalculator1
{
    //By Default methods are :: public abstract
    void add(int a,int b);
    void sub(int a,int b);
}
//Pure-Abstract-class : SRS
interface ICalculator2
{
```

```
    //By Default methods are :: public abstract
    void mul(int a,int b);
    void div(int a,int b);
    void add(int a,int b);
}
abstract class Calculator implements ICalculator1,ICalculator2
{
}
//Implemented class : concrete class
class CalculatorImpl extends Calculator
{
    @Override
    public void add(int a,int b)
    {
        int sum = a+b;
        System.out.println("The sum is :: "+sum);
    }
    @Override
    public void sub(int a,int b)
    {
        int diff = a-b;
        System.out.println("The sub is :: "+diff);
    }
    @Override
    public void mul(int a,int b){
        int res = a*b;
        System.out.println("The res is :: "+res);
    }
    @Override
    public void div(int a,int b){
        int quotient = a/b;
        System.out.println("The quotient is :: "+quotient);
    }
}
public class Test
{
    public static void main(String[] args)
```

```
    {
        //Creating a reference of interface
        Calculator calc;
        calc = new CalculatorImpl();
        //Calling the method based on runtime object
        calc.add(10,20);
        calc.sub(20,10);
        calc.mul(10,20);
        calc.div(5,2);
    }
}
```

Output

The sum is :: 30

The sub is :: 10

The res is :: 200

The quotient is :: 2

eg#2.

```
interface ICalculator
{
    public void add(int a,int b);
}
class CalculatorDemo
{
    public void sub(int a,int b)
    {
        System.out.println("The sub is :: "+(a-b));
    }
}
class CalculatorImpl extends CalculatorDemo implements ICalculator
{
    @Override
    public void add(int a,int b){
        System.out.println("The sum is :: "+(a+b));
    }
}
```

```
public class Test
{
    public static void main(String[] args)
    {
        CalculatorImpl calc;
        calc = new CalculatorImpl();
        calc.add(10,20);
        calc.sub(10,3);
    }
}
```

Output

The sum is :: 30

The sub is :: 7

eg#3.

```
interface ICalculator
{
    public void add(int a,int b);
}
class CalculatorDemo
{
    public void sub(int a,int b)
    {
        System.out.println("The sub is :: "+(a-b));
    }
}
abstract class Calculator extends CalculatorDemo implements ICalculator
{
}
class CalculatorImpl extends Calculator
{
    @Override
    public void add(int a,int b){
        System.out.println("The sum is :: "+(a+b));
    }
}
```

```
public class Test
{
    public static void main(String[] args)
    {
        Calculator calc;
        calc = new CalculatorImpl();
        calc.add(10,20);
        calc.sub(10,3);
    }
}
```

Output

The sum is :: 30

The sub is :: 7

Which of the following is true?

a. A class can extend any no of class at a time.

b. An interface can extend only one interface at at time.

c. A class can implement only one interface at at a time.

d. A class can extend a class and can implement an interface but not both simultaneously.

e. An interface can implements any no of Interfaces at a time.

f. None of the above

Answer: f

Consider the expression X extends Y which of the possibility of X and Y expression is true?

1. Both x and y should be classes.
2. Both x and y should be interfaces.
3. Both x and y can be classes or can be interfaces.
4. No restriction.

Answer: 3

Predict X,Y,Z

1)- a. X extends Y,Z?

X,Y,Z => interface

2)- b. X extends Y implements Z?

X,Y => class

Z => interface

3)- X implements Y,Z?

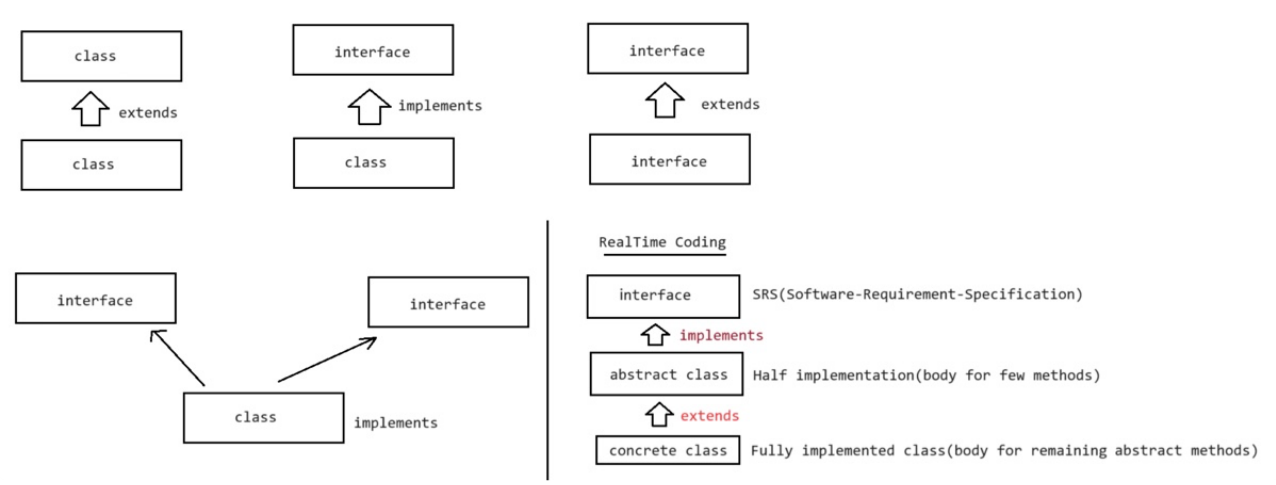
X => class

Y,Z => interface

4)- X implements Y extends Z?

invalid case.

Interface variables are specific to implementation or SRS? --> it is specific to requirement . Hence we need to access it through interface, but can we access to through interface instantiation==> hence it should be static



Interface variables

=> Inside the interface we can define variables.

=> Inside the interface variables is to define requirement level constants.

=> Every variable present inside the interface is by default public static final.

eg::

```
interface ISample
{
    int x=10;
}
```

public :: To make it available for implementation class Object.

static :: To access it without using implementation class Name.

final :: Implementation class can access the value without any modification.

variable declaration inside interface

a. int x=10;

b. public int x=10;

c. static int x=10;

d. final int x=10;

e. public static int x=10;

f. public final int x=10;

g. static final int x=10;

h. public static final int x=10;

Answer: All are valid

Note:

Since the variable defined in interface is public static final, we cannot use modifiers like private, protected, transient, volatile. since the variable is static and final, compulsorily it should be initialized at the time of declaration otherwise it would result in compile time error.

eg::

interace IRemote{ int x;}// compile time error.

Eg.1

```
interface IRemote
{
    //public static final
    int MIN_VOLUME = 0;
    int MAX_VOLUME = 100;
}
public class Test implements IRemote
{
    public static void main(String[] args)
    {
        int MIN_VOLUME = -5;
        System.out.println(MIN_VOLUME);
        System.out.println(IRemote.MIN_VOLUME);
        System.out.println(Test.MIN_VOLUME);
    }
}
```

Output

-5

0

0

eg#2.

```
interface IRemote
{
    //public static final
    int MIN_VOLUME = 0;
    int MAX_VOLUME = 100;
}
public class Test implements IRemote
{
    public static void main(String[] args)
    {
        MIN_VOLUME = -5;
        System.out.println(MIN_VOLUME);
        System.out.println(IRemote.MIN_VOLUME);
        System.out.println(Test.MIN_VOLUME);
    }
}
```

Output

CE: final variable value can't be modified.

Interface Naming Conflicts

Case 1::

If 2 interfaces contain a method with same signature and same return type in the implementation class only one method implementation is enough.

eg#1.

```
interface IRight
{
    public void methodOne();
}
interface ILeft
{
    public void methodOne();
}
public class Test implements ILeft,IRight
{
    @Override
```

```
    public void methodOne()
    {
        System.out.println("Impl for MethodOne...");
    }
    public static void main(String[] args)
    {
        Test t =new Test();
        t.methodOne();
    }
}
```

Output

Impl for MethodOne...

Case2:

If 2 interfaces contain a method with same name but different arguments in the implementation class we have to provide implementation for both methods **and these methods acts as a Overload methods.**

eg#1.

```
interface IRight
{
    public void methodOne();
}
interface ILeft
{
    public void methodOne(int i);
}
public class Test implements ILeft,IRight
{
    @Override
    public void methodOne()
    {
        System.out.println("Impl for MethodOne...");
    }
    @Override
    public void methodOne(int i)
    {
        System.out.println("Impl for MethodOne with One argument");
    }
```

```
    public static void main(String[] args)
    {
        Test t =new Test();
        t.methodOne();
        t.methodOne(10);
    }
}
```

Output

Impl for MethodOne...

Impl for MethodOne with One argument

case3:

If two interfaces contains a method with same signature but different return types then it is not possible to implement both interface simultaneously.

eg#1.

```
interface IRight
{
    public void methodOne();
}
interface ILeft
{
    public int methodOne();
}
public class Test implements ILeft,IRight
{
    @Override
    public void methodOne()
    {
        System.out.println("Impl for MethodOne...");
    }
    @Override
    public int methodOne()
    {
        System.out.println("Impl for MethodOne with One argument");
    }
    public static void main(String[] args)
    {
```

```
        Test t =new Test();
        //Overloading
        t.methodOne();
        t.methodOne();
    }
}
```

Output

CE: ambigous method call.

Can a java class implement any no of interfaces simultaneously?

Answer.yes, except if two interfaces contains a method with same signature but different return types.

Variable naming conflicts::

Two variables can contain a variable with same name and there may be a chance variable naming conflicts but we **can resolve variable naming conflicts by using interface names.**

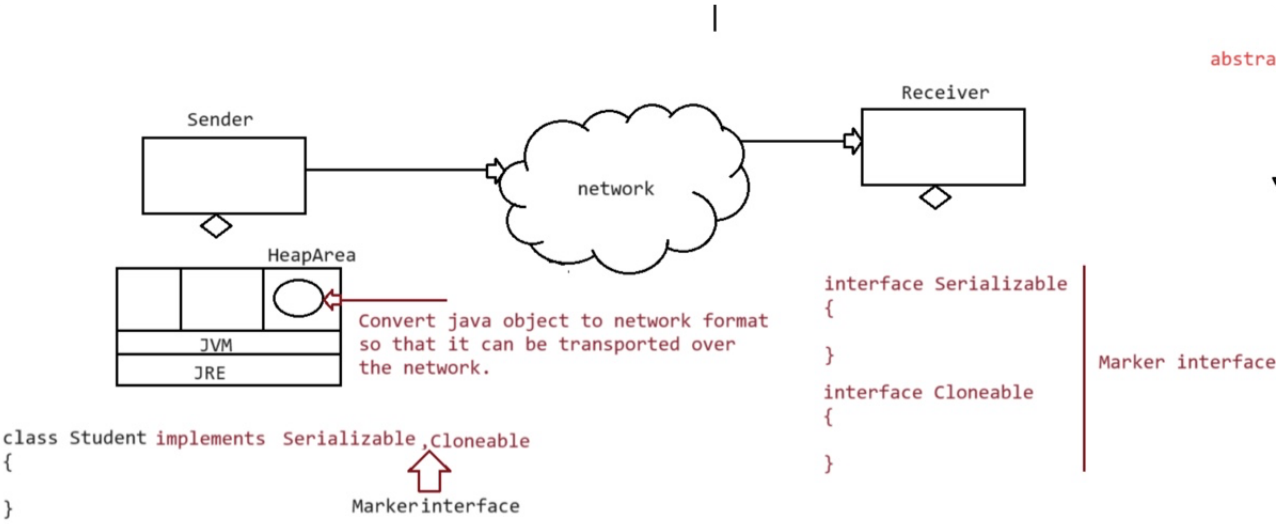
eg#1.

```
//SRS :: methods -> public abstract
//SRS :: variables -> public static final
interface IRight
{
    int x = 888;
}
interface ILeft
{
    int x = 999;
}
public class Test implements ILeft,IRight
{
    public static void main(String[] args)
    {
        //System.out.println(x);
        //System.out.println(Test.x);
        System.out.println(IRight.x);
        System.out.println(ILeft.x);
    }
}
```

Output

888

999



Explanation of the image -->

1)- If we want to send an object from sender to receiver over a network than we can't directly send it first we need to convert it into network format.

2)- To achieve we need to implement an interface where we need not to implement anything all will be done by jvm and such type of interface is known as **Marker Interface.**

3)- Serializable --> interface

4)- Since this pre-defined interface don't have I infront of it to avoid confusion we can add that.

5)- Now suppose we have sent this object but at receiver end there was some problem and he didn't receive the object than we again have to create an object and send it which is a costly process, instead we can create a copy of the object and send it. This can be achieved with Cloneable interface again it will be done by jvm.

Marker Interface(name --> as we are just marking its name with class, not providing any implementation from our side)

=> If an interface does not contain any methods and by implementing that interface if our Object will get some ability such type of interface are called "Marker Interface"/"Tag Interface"/"Ability Interface".

=> example --> Serializable, Cloneable, SingleThreadModel, RandomAccess.

Example1 -->

By implementing Serializable interface we can send that object across the network and we can save state of an object into the file.

Example2 -->

By implementing SingleThreadModel interface servlet can process only one client request at a time so that we can get "Thread Safety".

Example3 -->

By implementing Cloneable Interface our object is in a position to provide exactly duplicate cloned object.

Without having any methods in marker interface how objects will get ability?

Ans. JVM is responsible to provide required ability.

Why JVM is providing the required ability to Marker Interfaces?

Ans. To reduce the complexity of the programming.

Can we create our own marker interface?

Ans. Yes, it is possible but we need to cusomtize JVM.

Adapter class

1)- It is a simple java class that implements an interface only with empty implememtation for every method.

2)- If we implement an interface compulsorily we should give the body for all the methods whether it is required or not. This approach increases the length of the code and reduces readability.

eg::

```
interface X{
    void m1();
    void m2();
    void m3();
    void m4();
    void m5();
}
class Test implements X{
    public void m3(){
        System.out.println("I am from m3()");
    }
    public void m2(){}
    public void m3(){}
    public void m4(){}
    public void m5(){}
}
```

1)- In the above approach, even though we want only m3(), still we need to give body for all the abstract methods, which increase the length of the code, to reduce this we need to use "Adapater class".

2)- Instead of implementing the interface directly we opt for "Adapter class". Adapter class are such classes which implements the interface and gives dummy implementation for all the abstract methods of interface.

3)- So if we extends Adapter classes then we can easily give body only for those methods which are interested in giving the body.

eg::

```
interface X{
    void m1();
    void m2();
    void m3();
    void m4();
    void m5();
}
abstract class AdapaterX implements X{
    public void m1(){}
    public void m2(){}
    public void m3(){}
    public void m4(){}
    public void m5(){}
}
class TestApp extends AdapterX{
    public void m3(){
        System.out.println("I am from m3()");
    }
}
```

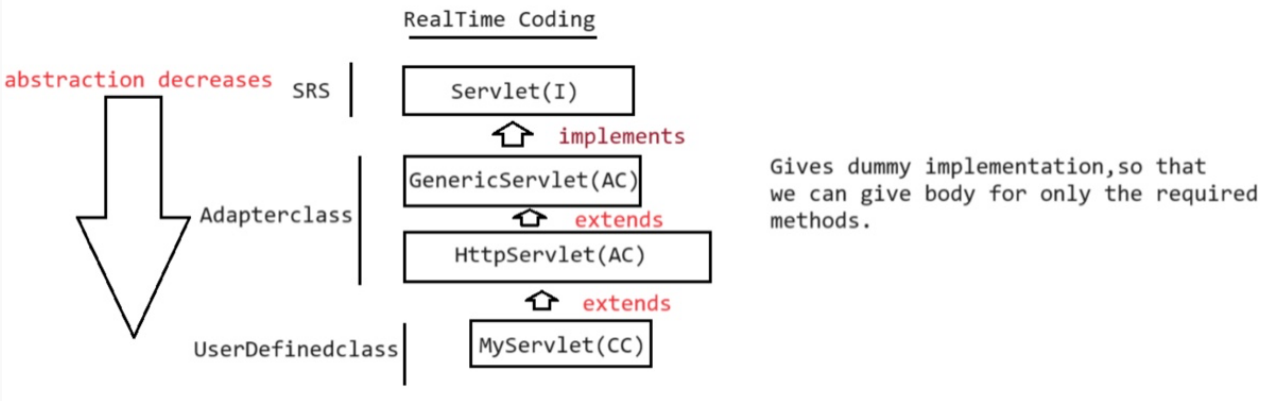
eg:: interface Servlet{....}

abstract class GenericServlet implements Servlet{}

abstract class HttpServlet extends GenericServlet{}

class MyServlet extends HttpServlet{}

Note:: Adapter class and Marker interface are big utilities (or design) to programmer to simplify programming.



Interview Questions

Q>What is the difference b/w abstract class and interface?

Q>Every method present inside the interface is abstract,but in abstract class also we can take only abstract methods also then what is the need of interface concept?

Q> Why abstract class can contains constructor and interface doesn't contains constructor?

Q> When to go for interface and when to go for abstract class?

Difference b/w Abstract class and Interface?

Interface::

1)- If we dont know anything about implementation just we have requirement specification then we should go for interface.

2)- Every method present inside the interface is always public and abstract whether we are declaring or not.

3)- We can't declare interface methods with the modifiers like private,protected,final,static,synchronized,native,strictfp.

4)- Every interface variable is always public static final whether we are declaring or not.

5)- Every interface variable is always public static final we can't declare with the following modifiers like private,protected,transient,volatile.

6)- For every interface variable compulsorily we should perform initialisation at the time of declaration, otherwise we get compile time error.

7)- Inside interface we can't write static and instance block.

8)- Inside interface we can't write constructor.

Abstract class::

1)- If we are talking about implementation but not completely then we should go for abstract class.

2)- Every method present inside abstract class need not be public and abstract.

3)- There are not restrictions on abstract class method modifiers.

4)- Every abstract class variable need not be public static final.

5)- No restriction on access modifiers

6)- Not required to perform initialisation for abstract class variables at the time of declaration.

7)- Inside abstract class we can write static and instance block.

8)- Inside abstract class we can write constructor.

Q>Every method present inside the interface is abstract,but in abstract class also we can take only abstract methods also then what is the need of interface concept?

Ans. we can replace interface with abstract class,but it is not a good programming practise.if we try to do, it would result in "missusing the role" of abstract class and it would also create peformance issue.

Using interface

```
interface ISample
{
}
class SampleImpl implements ISample
{
}
```

1. ISample sample=new SampleImpl(); **//one level chaining :: SampleImpl ---> Object[Performance is relatively high]**
2. **While Implementing ISample, we can also get the benefit from Another class[Inheritance : Reusability].**

Using Abstract class

```
abstract Sample
{
}
class SampleImp extends Sample
{
}
```

1. Sample sample=new SampleImp(); **//Multi level chaining:: SampleImp ---> Sample ---> Object [Performance is low]**
2. **While extending Sample, we can't get the benefit of other classes[Inheritance can't be used here]**

Q> Why abstract class can contains constructor and interface doesn't contains constructor?

=> Constructor :: To initialize the instance variable of an object, meaning is to provide values for instance variables.

:: In abstract class we have instance variable so we need constructor for initializing the instance variables.

:: In case of interface, we don't have instance variable we have variables which are of type public static final, these variables are initialized at the time of declaration only.

so we dont' need constructors in interface.

Q> When to go for interface and when to go for abstract class?

interface -> To promote 100 percent abstraction we need to go for "interface" or to provide Software Requirement Specification we need to go for "interface".

abstract class -> If we are taking about implementation that is partial implementation, then we need to go for "abstract class".

Which of the following are valid?

1. The purpose of the constructor is to create the object.
2. The purpose of the constructor is to initialize the object,not to create the object.
3. Once constructor completes then only object creation completes.
4. First object will be created and then constructor will be executed.
5. The purpose of the new keyword is to create object and the purpose of constructor is to initalize the object.
6. **We can't create Object for abstract class directly but indirectly we can create.**
7. Whenever we are creating child class object automatically parent class object will be created.
8. Whenever we are creating child class object automatically abstract class constructor(provided if it is parent) will be executed.
9. Whenever we are creating child class object automatically parent constructor will be executed but parent object wont be created.
10. Either directly or indirectly we can't create Object for abstract class and hence constructor concept is not applicable for abstract class.
11. Interface can contain constructor.

Valid : 2,4,5,8,9

Invalid : 1,3,6,7,10,11