N
S.6

Introduction

Date - 3/09/12
R.S. - 190/-

Core Java:

Q. What is Java?

→ Java is a very simple, high-level, secured, concurrent, platform independent, object oriented programming language.

→ Java is a technology which provides —

a. Language
b. Platform.

Q) What is a platform?

→ Platform is a hardware & software ~~environment~~ which provides runtime environment for applications or programs.

→ This runtime environment consists of —

1. Memory management
2. Process management
3. I/O management
4. Device management.

→ Platform which acts as mediator or interface between software & hardware.

Q) What is the abbreviation of Java?

→ There is no abbreviation of Java. The Development Team of Java just choosen this name.

→ The name Java specifically doesn't have any meaning rather it refers to the hot, aromatic drink COFFEE. This is the reason Java programming language icon is coffee cup.

Q) What is meant by platform dependent & platform independent application?

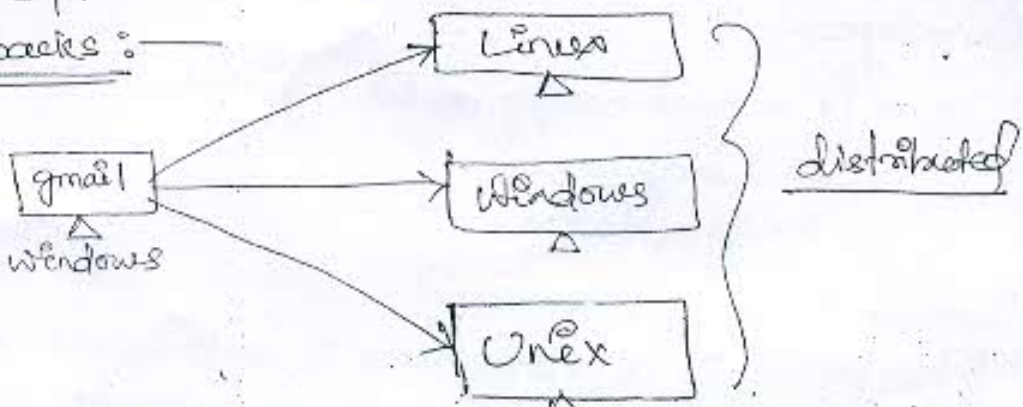
Platform dependent :—

→ An application that is compiled on one OS ~~is~~ is not runnable to run in another OS, then that application is called Platform dependent.

Pavitra Patnaik

→ A program which is prepared (compile) on one OS can be execute on same OS but cannot be executed on other OS, becoz once we compiled this it generates machine code which is specific for a machine one which it was prepared.

Drawbacks :—



→ After compilation of the program, it generates machine code which is platform dependent code.

4/9/12

Platform Independent :—

→ If the application's compiled code is able to run in another Operating System then that application is called platform idependent appl^n.

→ The program which is compiled on ~~one OS~~, it should not generate machine code.

→ When this program is compiled it generates a intermediate language code (byte code) which doesnot have any instruction related to real machine OS.

→ Bytecode generates becoz to archive platform independency.

→ Platform independency is requireed for distributed application.

→ It is requireed a software which is responsible to convert the intermediate language into machine code.

Q What is Bytecode?

→ A compiled code of java source program is called bytecode.

→ It is an intermediate language code, which is a portable code means portable across multiple OS.

→ Bytecode is a collection of mnemonics. (mnemonics → LOAD, MOV .....).

→ The name is given as bytecode becoz everey byte code in computer memory occupy "one byte".

Q. Diff. bet" bytecode & machine code?

bytecode	machine code
(1) It is a portable code.	(1) It is a non-portable code.
(2) These codes are collection of mnemonices.	(2) These codes are collections of 1's & 0's.
(3) This code is platform independent code.	(3) This code is platform dependent code.

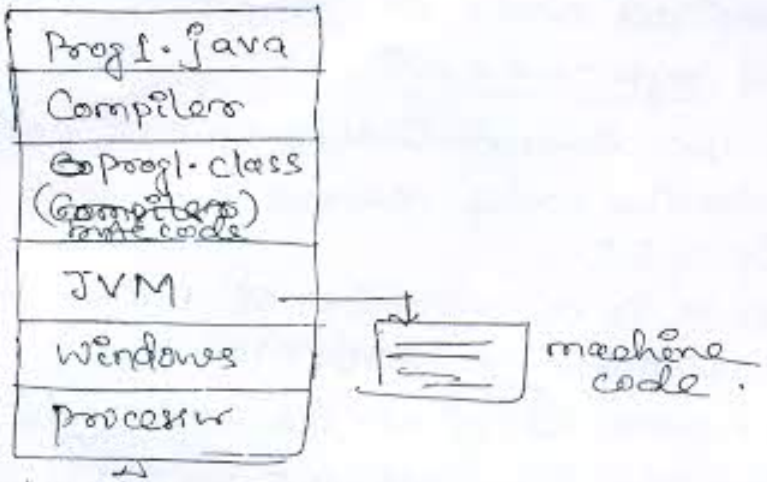
Q. What is the java file's extension?

1. Source code → "•java" → developer written code
2. Compiled code → "•class" → the bytecode generated by compiler

→ Generally ".exe" contain machine code which is understand by machine contain JVM (java virtual machine).

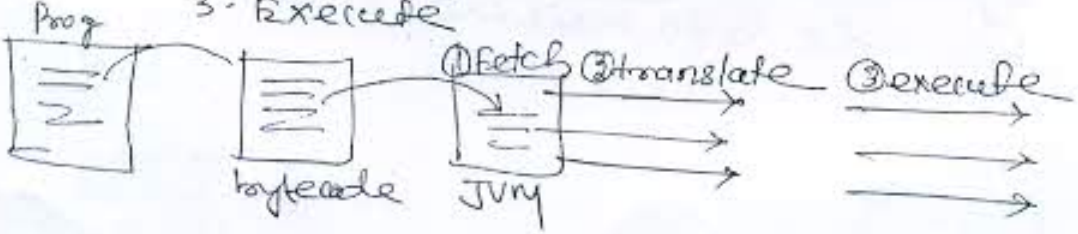
→ Compiler is responsible for converting source code into bytecode and JVM is responsible for converting bytecode into machine code.

Java Virtual Machine (JVM) 5/09/12



→ JVM is developed by SUN separetly for every OS. not by OS vendor.
→ JVM is available for all OS separately. But JVM is not install automatically, we must install JVM in our computer for executing Java program bytecodes.
→ Java is platform independent, Java Software is platform depedent, Java Software is known as JVM which is provided by java.
→ JVM provides runtime environment for java applications/programs.
→ JVM is a software in C, C++, becoz of this C & C++ are platform dependent.
→ JVM provides 2 translators for converting bytecode into machine code or executable code.
1. Interpreter
2. Hotspot compiler / JIT (just in time compiler)

Interpreter:
→ Interpretor translates byte code into machine code line by line.
→ It performs 3 operations
1. fetch (Reading)
2. translate
3. Execute



JIT :- (Just in time compiler / HOT-SPOT)

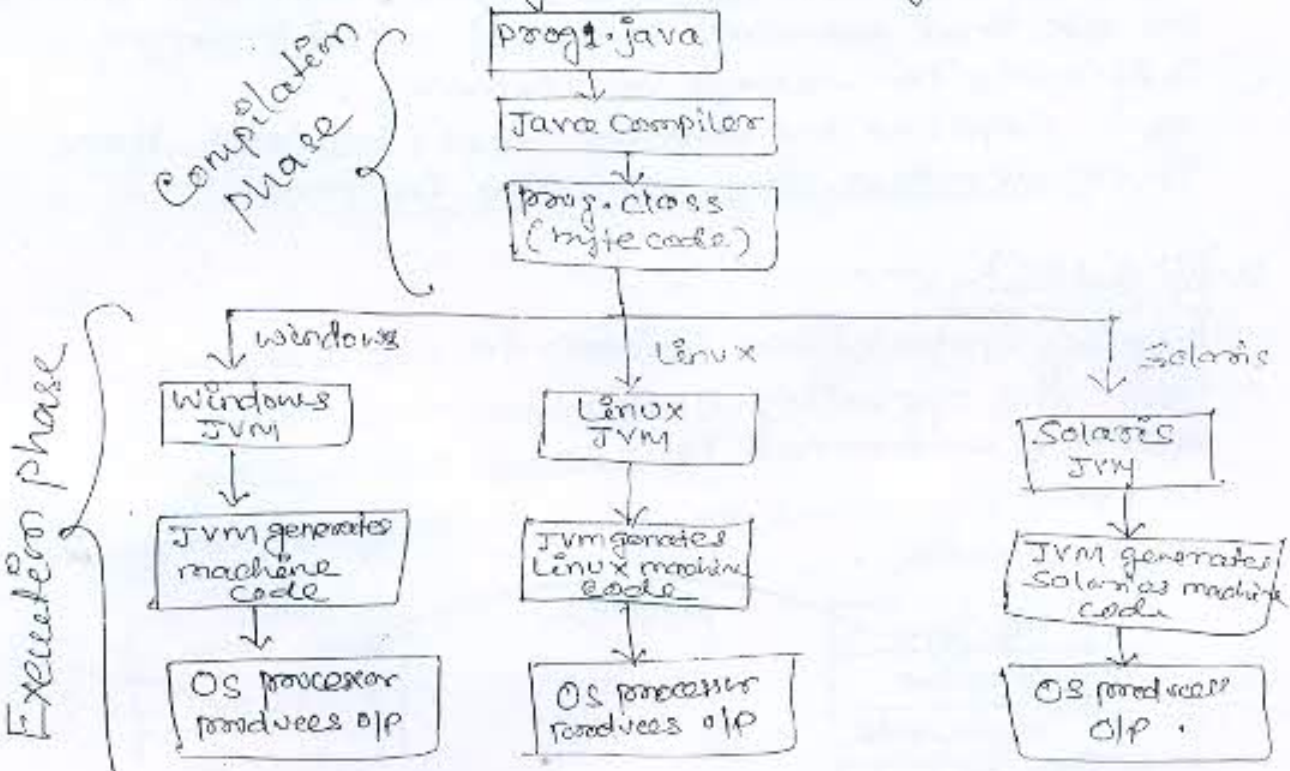
→ It is provided by JVM, which translate bytecode into machine code.

→ It performs 4 operations -
1. fetch.
2. Translate
3. Store
4. execute.

Q. Java is a platform dependent, Justify ?

→ The compiled code of java is platform (os) independent, but platform (JVM) provided by java is dependent.

→ Java software is platform dependent and Java program becomes platform independent because it generates bytecode which is run in every machine (os) as JVM is separately install in every machine.



Java Features :-

1. Simple
2. Platform independent
3. portable
4. object oriented
5. Architecture
6. Dynamic
7. Robust
8. Secured
9. Multithreading
10. Distributed.

1. Simple:—

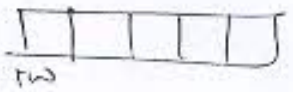
→ Java simplifies the programers job by avoiding explict memorey management.

→ Java provides automatic memorey management (garbage collector), which remove memorey which is not in use.

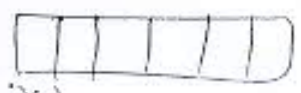
ex:-

```
void main()
{
  fun1();
  fun2();
}
```

```
void fun1()
{ int *p;
  p = malloc(1w);
}
```




```
void fun2()
{ int *q;
  q = malloc(2w);
}
```



Q) What is memorey leak?

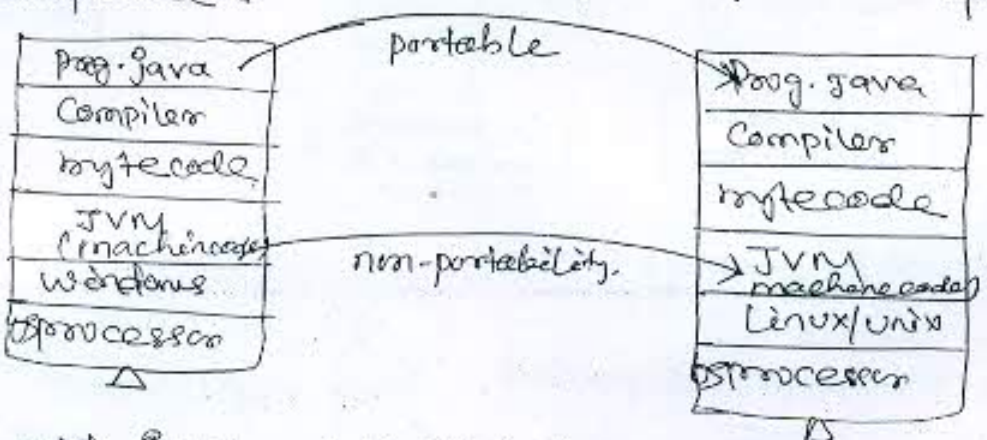
→ The memorey reserved by program unable to unreserve or use that memorey is called leaked from program. This leads to wastage of memorey.

→ Java simplifies by avoiding (rest) pointers, there is no dereferencing operator in java.

2. Portability:—

→ Moving instructions written in one language from one operating environment to another operating environment is called portability.

→ portability allowes to develop program irrespective of hardwares.



C, C++, Java → portable language.

sourcecode, bytecode of java ⟶ portable

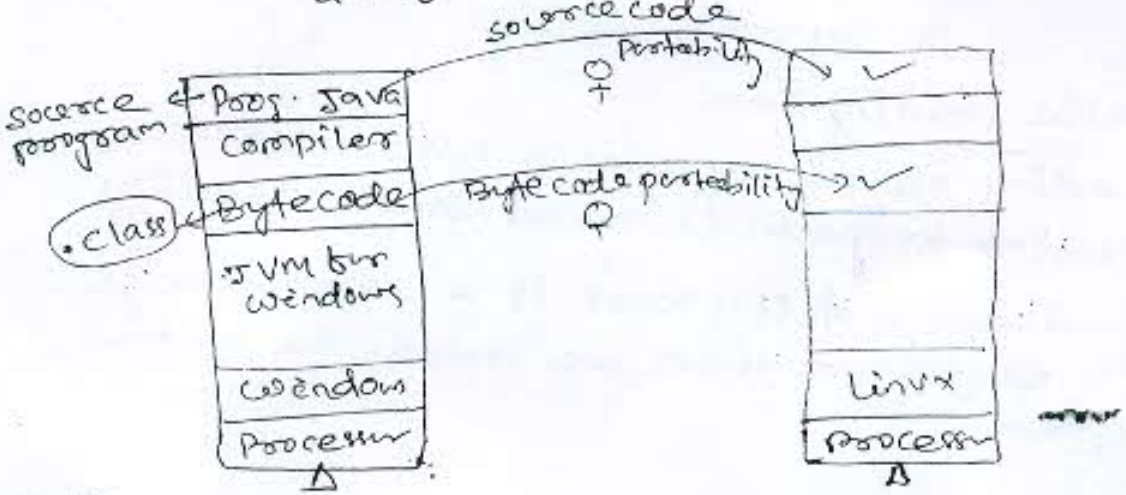
machinecode of java ⟶ non-portable.

Date - 6/09/2012

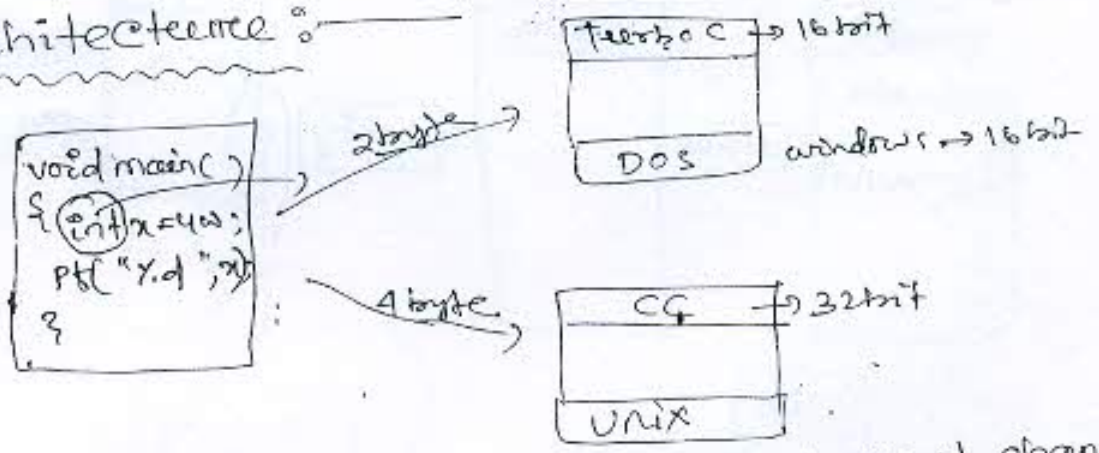
Portability :—

Java provides 2 type of portability —

1. Source code portability
2. Bytecode portability.

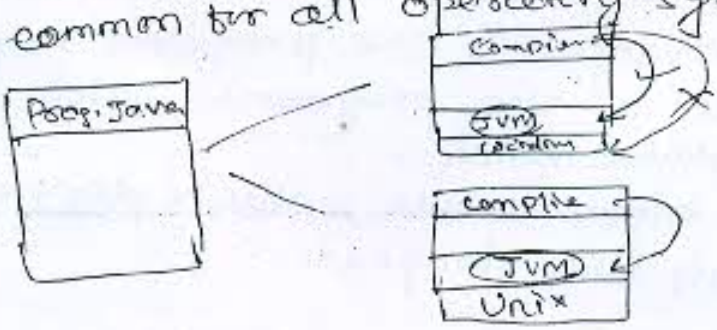


Architecture :—



→ The behaviour of java program doesn't changes from one system to another system.

→ JVM is a specification or set of rules which are common for all operating systems.



Dynamic :—

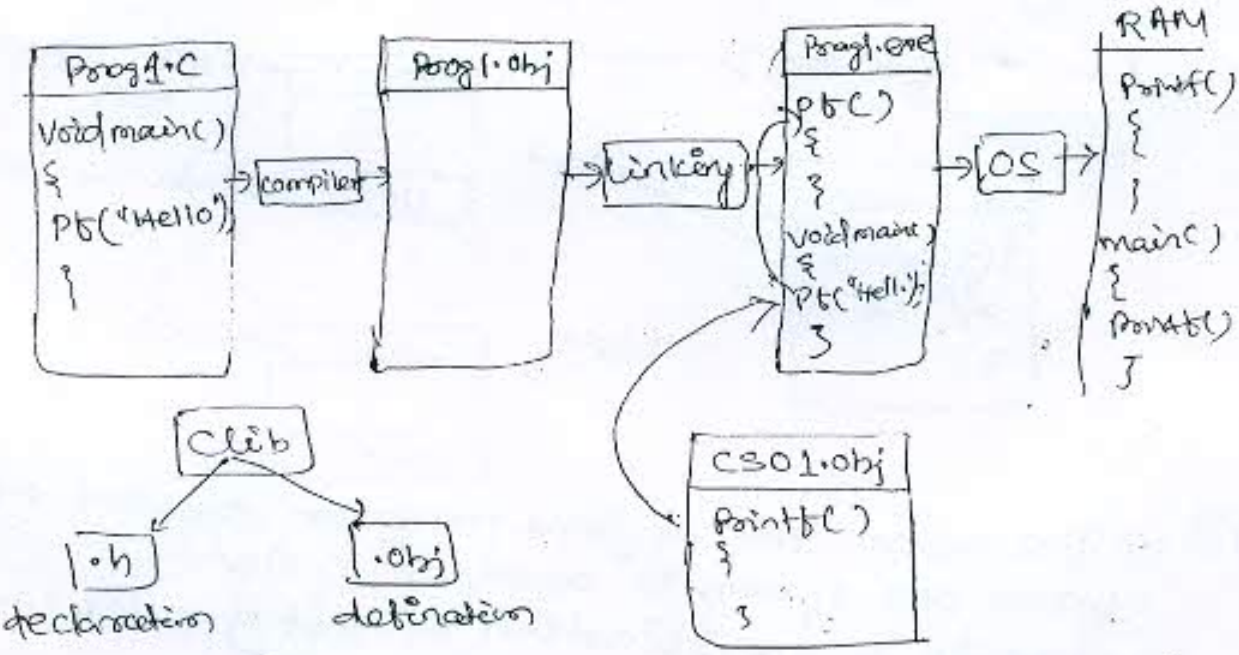
→ Loading or Linking of programs are 2 types.

1. Static loading
2. Dynamic loading

Static loading :—

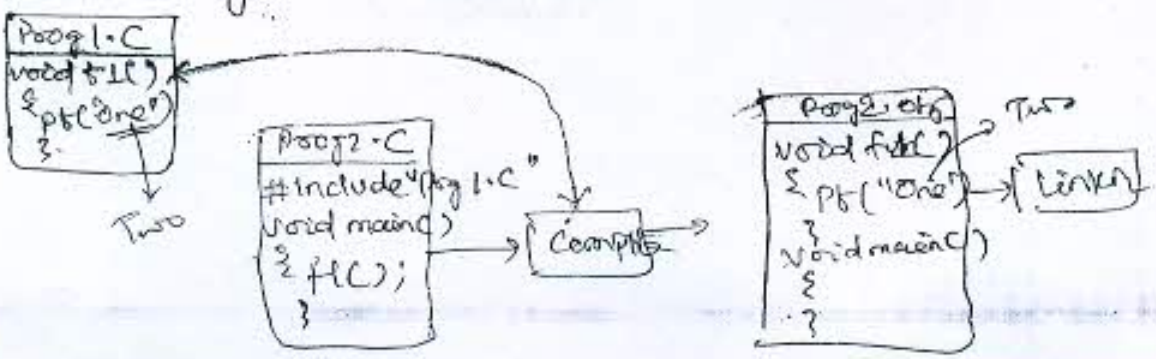
→ Loading of all executable blocks before executing program is called static loading.

→ Disadvantage of it, that if a small change in a program then we complie the entire program.



Drawback :—

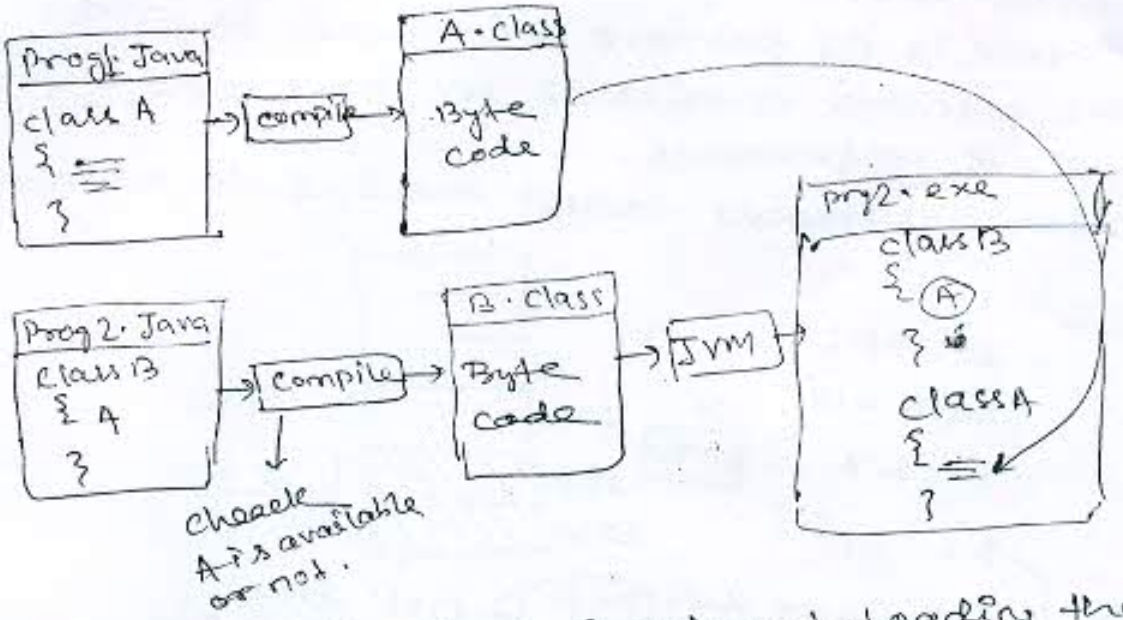
1. → Any small change in one program required to recompile all the programs, which uses that executable block.
2. Required more space, which decrease efficiency of the program.



Dynamic loading :—

→ Loading & Linking of executable blocks during execution of program is called dynamic loading.

Ex :-



→ The main advantage of dynamic loading that any small change in one program doesn't required to recompile all the programs.

Robust :—

→ Java is a streongly typed language.
→ It is having strict type checking during compile time & runtime.
→ The errors which are recognized by compiler are called syntax errors.
→ The errors which are recognized by run-time (JVM) are called logical errors.

Ex:

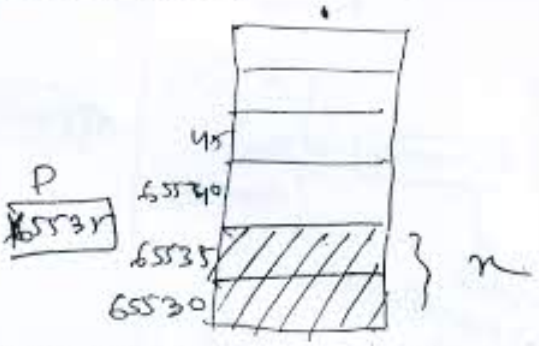
```
void main()
{ int x
  y
  z
  z=x+y;   → logical error
  printf("%d", z);
}
```

Security:—

→ Pointers are not secured bcoz it manipulate address.
→ Java restrite pointers operations.
→ There is no pointers arithmetic in java.
→ The pointers available in java are called references.
→ These references can't manipulate addresses.

Ex:

```
int *p;
int x = 10;
p = &x;
p = p - 1;
```



; in java this (p=p-1) not done.

Date - 27/09/2012

Multithreading:—

web appln → multithreaded.

→ Multithreading allows to develop a multi-tasking applications.
→ A process is an instance of a program and simultaneous execution multiple processes is called process based multitasking.
→ A thread is a instance of a process or a process under execution simultaneous execution is called thread based multi-tasking.

Distributed :—

→ It allows to develop networking application.
→ It provides set of pre-defined protocols which allows to develop distributed applications.

Java Editions :—

1. JSE → Java standard Edition. (web-supportive appln)
2. JME → Java micro edition. (device appl")
3. JEE → Java Enterprises Edition. (web appl")

J2SE → 2nd platform. JSE 5.0
J2ME JSE 6.0
J2EE JSE 7.0 → Latest version.

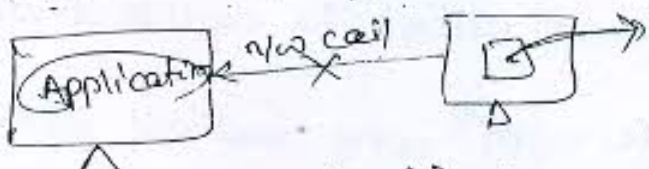
JSE :—

→ It allows us 3 types of applications —
ⓐ Standalone application.
ⓑ Client - server application.
ⓒ web-supportive application.

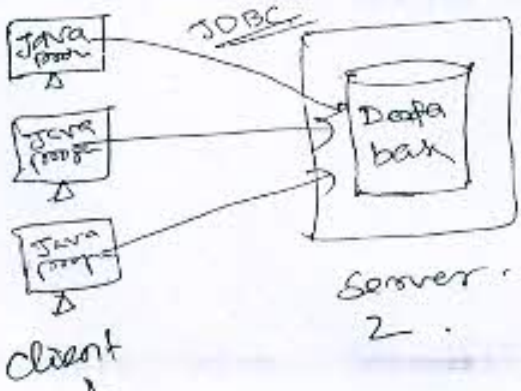
Standalone Application :—

→ An application whose resources cannot shared by more than one user is called "Standalone application".

→ These applications can be desktop applications or console based applications.



Client - server application :—



→ A client is a java program which communicate with server (database).

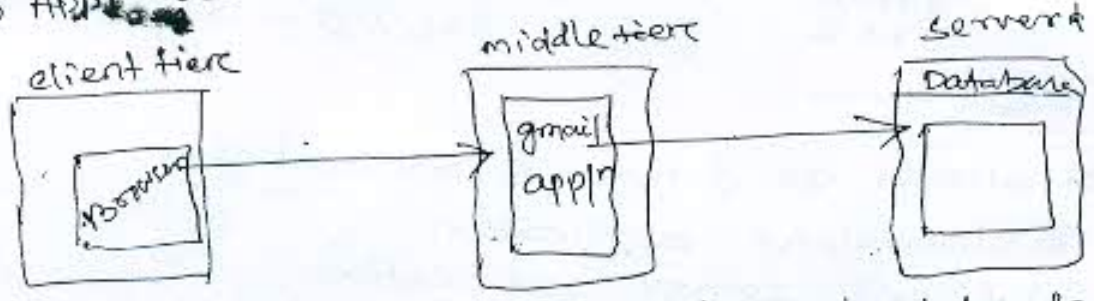
→ Applet is an embeded program which is embeded within HTML page (webpage).

→ It is downloaded along with webpage & executed in client browser.

JEE :—

→ API are "

→ Appl^n resources are shared by 'n' no. of clients.



web server → protocol dependent
appl^n server → protocol independent

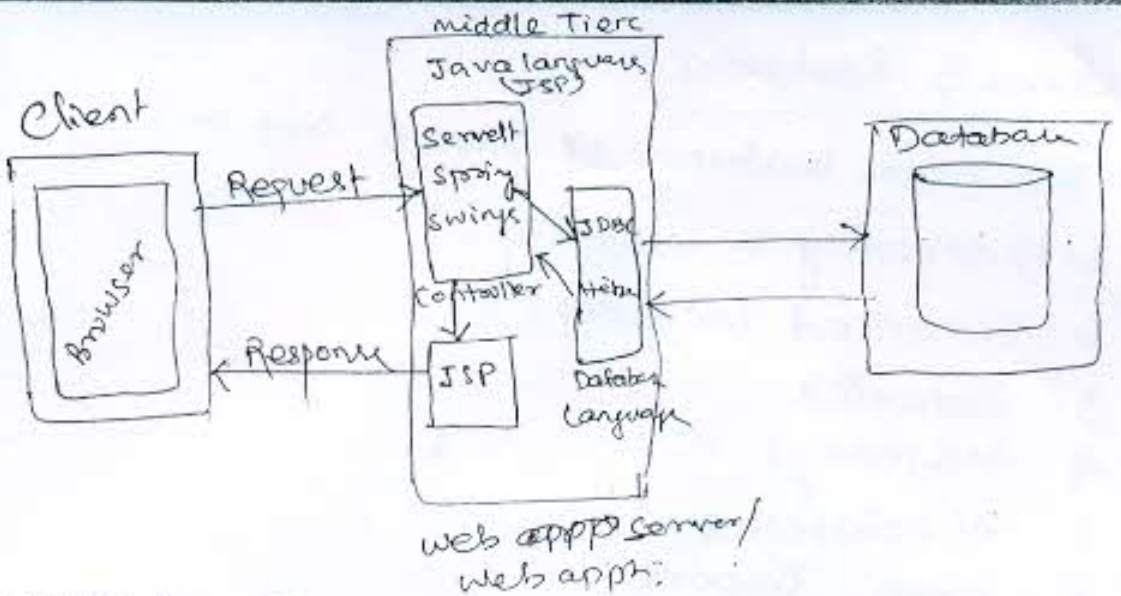
(3 tier architecture).

→ JEE consists of JDBC, JSP, servelet, JSP, EJB, spring, hibernet, struts, XML, webservices all are called API (application programming interface).

→ An application whose resources shared by more than one client is called enterprise application.

→ All internate appl^n are called enterprises applications.

→ Every internate appl^n is called web-application.



JME :-

→ This is used for developing device appln.

→ The applications which are written using C & C++ can be developed with JME.

Java Versions :-

→ A small change within the edition is called versions.

		Code Name
Java 1.0	1995	Oak
Java 1.1	1997	Oak
Java 1.2	1999	playground
also called as Java 2 platform.		
Java 1.3	2000	Kestrel
Java 1.4	2002	Merlin
Java 5	2004	Tigere
Java 6	2006	Mustang
Java 7	2009	Dolphin

J2SE 1.2, J2SE 1.3, J2SE 1.4 → no change in Language (collection of keywords) i.e change Library.

JSE 5.0, JSE 6.0, JSE 7.0 → There is a change in language.

Java 5 Features :-

→ The extra features of java5 are -

1. Autoboxing & unboxing.
2. Enhanced for loop.
3. Generics.
4. Varargs
5. Annotations
6. static import.
7. Enum

Date - 8/09/2012

Java History :-

→ Java was created in 1991 by "James Gosling" of Sun micro system. Initially called Oak. Its name was changed to Java bcoz there was already a language called Oak.

Note :-
Now java is a product of ORACLE corporation.

→ The original motivation for java is they need for a platform independent language that could be embedded in various consumer electronic products like toasters and refrigerators.

Java Software :-

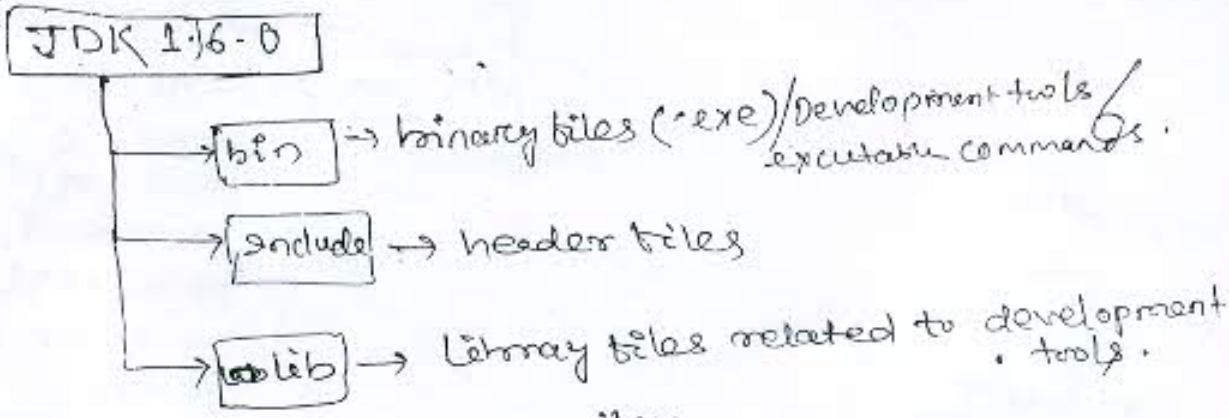
→ Java software is a platform dependent.
→ In order to develop java applications, a system should have a software called JSDK (Java Software Development Kit)

Java Software

- JDK (Java development Kit) — person want to develop a java program (developer)
- JRE (Java runtime environment) — (developers or end users) — person wants to execute the program.

JDK :—

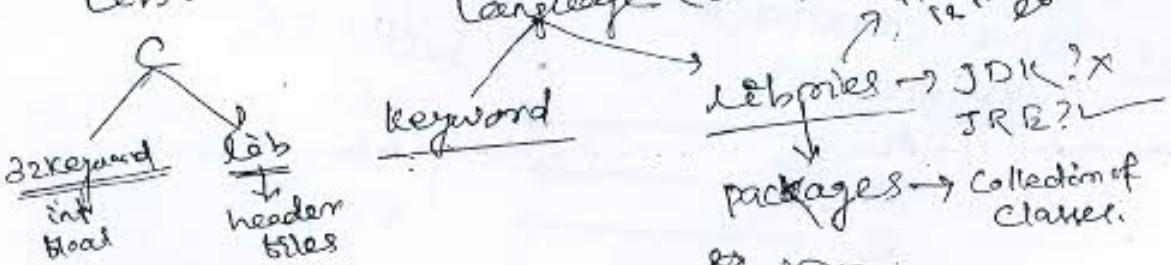
→ JDK is the java Development Kit used by java developer/ programmer in order to develop java projects/ applications.



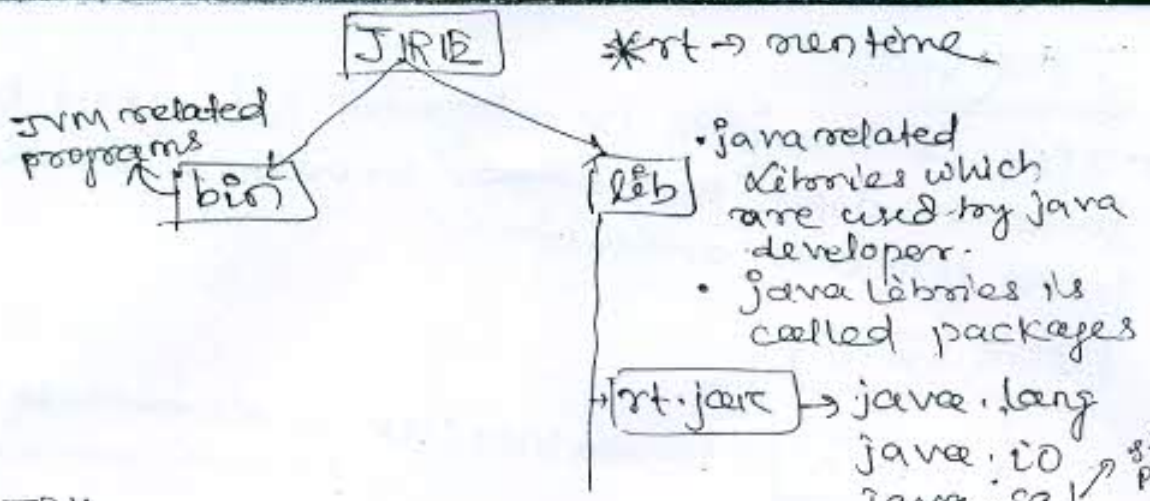
javac.exe → java compiler
javap.exe → java prototiler
java.exe → java interpreter
~~javaxhere~~ →
javah.exe → java header file generator.
jdb.exe → java debuger.
jar.exe → java archive utility.
javadoc.exe → java document generator.

JRE :—

→ JRE steands for Java Runtime environment, which consists of JVM and runtime class Libraries.



→



JDK
↓
bin
↓
(All development tools are present.)
eg: javac.exe

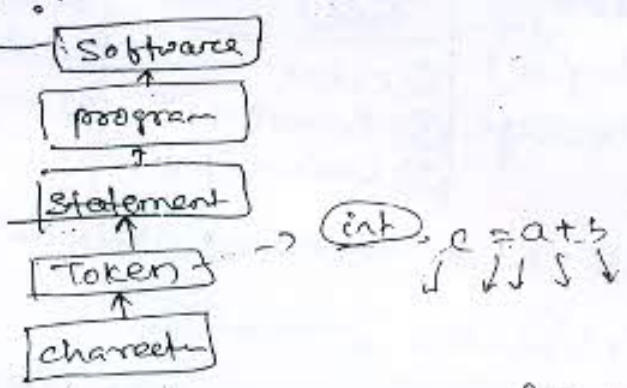
Date - 10/9/2021

Character Set of java:

→ Java is a universal language.
→ It not only support characters available in english. It also supports characters available in other languages.
→ Java uses a UNICODE character set.
→ Accordingly to java there is 65536 characters available in java.
*→ In java character takes 2 bytes.

characters	UNICODE
A	65
B	66
C	67
⋮	
a	97
b	98
c	99
⋮	
अ	300
आ	301

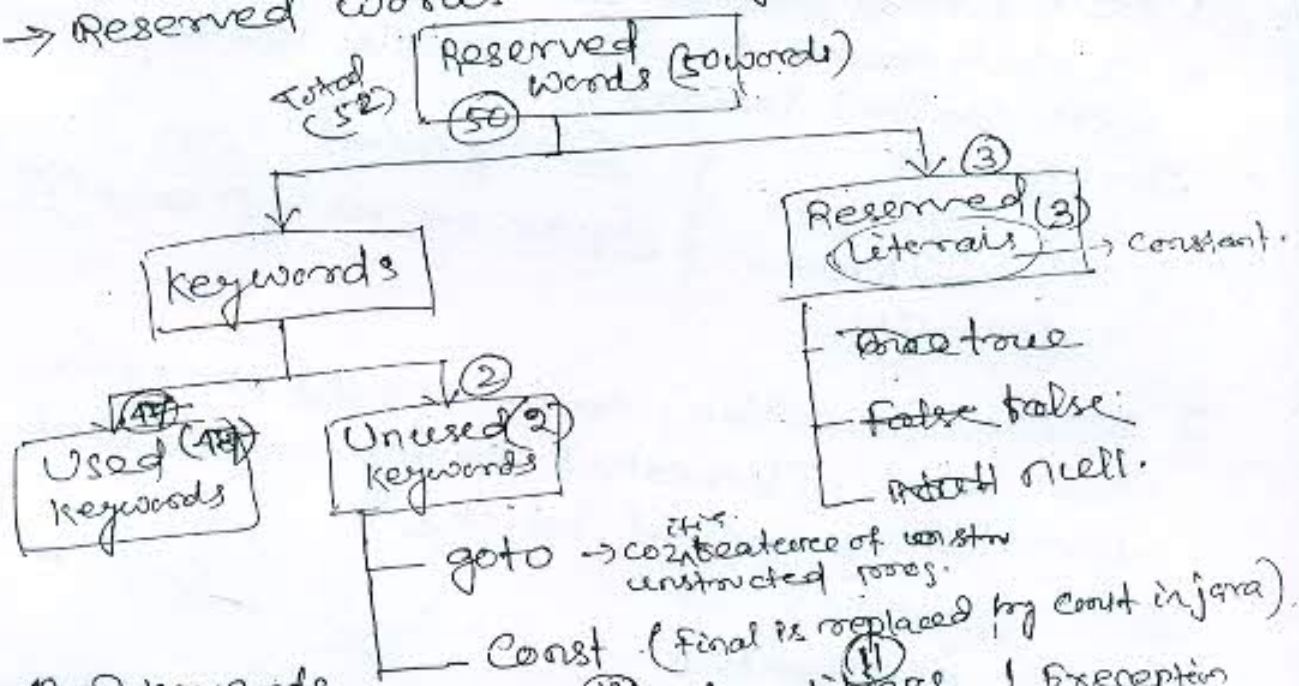
Reserved Words :—



→ Java language related words are called "reserved words".

→ These words are recognized by java compiler & java runtime (JVM).

→ Reserved words are 2 types —



Used Keywords

Datatypes (8)
1. int
2. short
3. long
4. float
5. char
6. Double
7. Boolean
8. void
9. Byte

Control Statements (12)
1. if
2. else
3. if else
4. switch
5. case
6. Default
7. for
8. while
9. DO
10. break
11. Continue
12. Return

modifiers (11)
1. Private
2. Public
3. Protected
4. abstract
5. final
6. static
7. transient
8. Synchronized
9. volatile
10. Strict FP
11. native

Exception
1. Try
2. catch
3. Finally
4. Throw
5. Throws
6. assert (1.4)

Packages	class related type/	Inheritance	References
① import	① class	① extends	① This
② Package	② interface	② implements	② super
	③ enum (1.5)		③ instance of

* Null is a reserved literals.

Identifiers :-

→ Identifier is a user-defined word.
→ Identifier is a collection of alphabets, digits, & allows 2 special characters (i.e $, _).
→ The Identifiers contain certain rules called "Naming convention".

1. class name should start with Capital letter.
* If class name having multiple words separate with capital letters.

ex:- Account
SavingAccount
CurrentAccount
Deposite

```
String name
System.out.println("Hello");
```

2. Method (function) name should start with small letter, if method having multiple words separated by capital letters.

ex:- print()
printReport()

3. Variable name should start with small letter. If variable name having multiple words separated with Capitals.

ex: print, accno, CustomerName.

4. Constant name should be given in Capital letters. If constant name having multiple words separated by underscore (_).

ex:- PI, MAX_RANGE;
MAX_VALUE

Q. Identify the following identifiers ?

Student → variable
Rollno → class
getRollno() → function/method
MAX_RNO → Constant.

Q Identify valid identifiers ?

goto → invalid
transient → invalid
true → invalid
TRUE → valid (coz java is case sensitive so find the difference.)

Q. Identify reserved literals ?

this → no
instanceof → no
true → Yes
null → yes

Datatypes in Java :

→ Java datatypes are classified into 2 categories.
1. Primitive datatypes. (value type)
2. Refrence datatypes. (address type)
→ The main purpose of Datatypes is that what kind of value we can stored in to the variable.

Primitive datatypes :

→ A variable of type primitive datatype hold values.
→ The primitive datatypes are — by default "signed".
→ Every datatype in java is by default "signed".

① int → 4bytes
② short → 2bytes
③ long → 8byte
④ byte → 1byte
⑤ double → 8byte
⑥ float → 4byte

⑦ Boolean → 1 bit
⑧ void → empty datatype
⑨ Chare → 2byte

Reference Datatypes :—

→ A variable of type hold address

→ Reference variable is restricted pointer variable.

→ The reference datatypes are —

1. class
2. Interface
3. Array
4. enum

Date - 11/09/2012

Q. Why char in java is 2 byte ?

The character in java is 2 byte because in java UNICODE is used so to allocate memory for 65536 we required 2 byte memory.

* In C → char is 1 byte coz it uses ASCII code.

Structure of java Program :—

→ Java Program is divided into 4 section.

1) Document Section.
2) Package statement.
3) import statement.
4) class defination section.

Document Section :—

→ In document Section content information about the program.

→ This section is ignored by compiler.

→ Documentation is provided using comments.

→ Java supports 3 types of comments —

1) Singleline comment (//)
2) Multiline comment (/* */)
3) Documented comment (/** */)

ex
```
/**
 company :-
 Title :-
 projects :-
 Date :-
 */
```
(Advantage: we can separate it from others.)

→ Documented comments can be separated from source program to build document file (.html).
→ This building of document file is done using utility called "javadoc"
→ Documented comments are optionals.

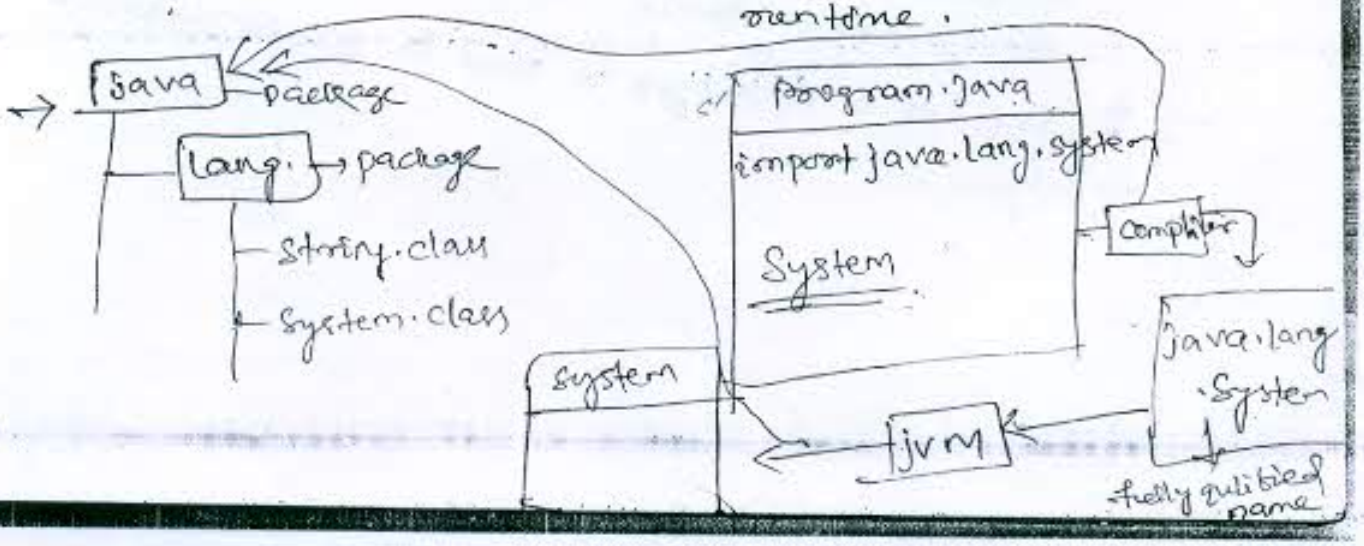
Package statement :—

→ Package is a collection of types. These types ^ (Contains) classes, interfaces or enum.
→ Package is a directory which contains "class" files.
→ Package provides ⓐ namespace management & ⓑ access protection (i.e Security).
→ One source program allows only one package statement.
→ for a beginers package may or may not compulsory but for a developer it is compulsory.

import statement :—

#include → includes the program

import
- navigates the program, to check either class is present or not & stores the reference, that will help in runtime.



→ Import is used for using the contents of existing package.

→ Import does not include class, but add package reference. It adds name of the package to it belongs.

→ A java program can have multiple import statements.

→ The default package import for any java program is "java.lang".

ex: import package_name.* // more than one //.
import package_name.class_name; // for a single class //.

Q) Can we use content of package without using Package statement?

Ans Yes, using fully qualified name. (i.e class name along with the package name).

Class definition Section:

- As the term in java encapsulated inside the class so it is called "method".

→ Every java executable program must have a class which contain "main()" method.

→ class have 2 type.
① Executable
② non-executable.

→ Class is a data type & that is user defined datatype.

First program in Java :—

Steps :—

1. Open any text editor (notepad).
2. Write java program.
3. Save with an extension ".java".
4. Compiling java program.

   Steps for it —

   a. open command prompt

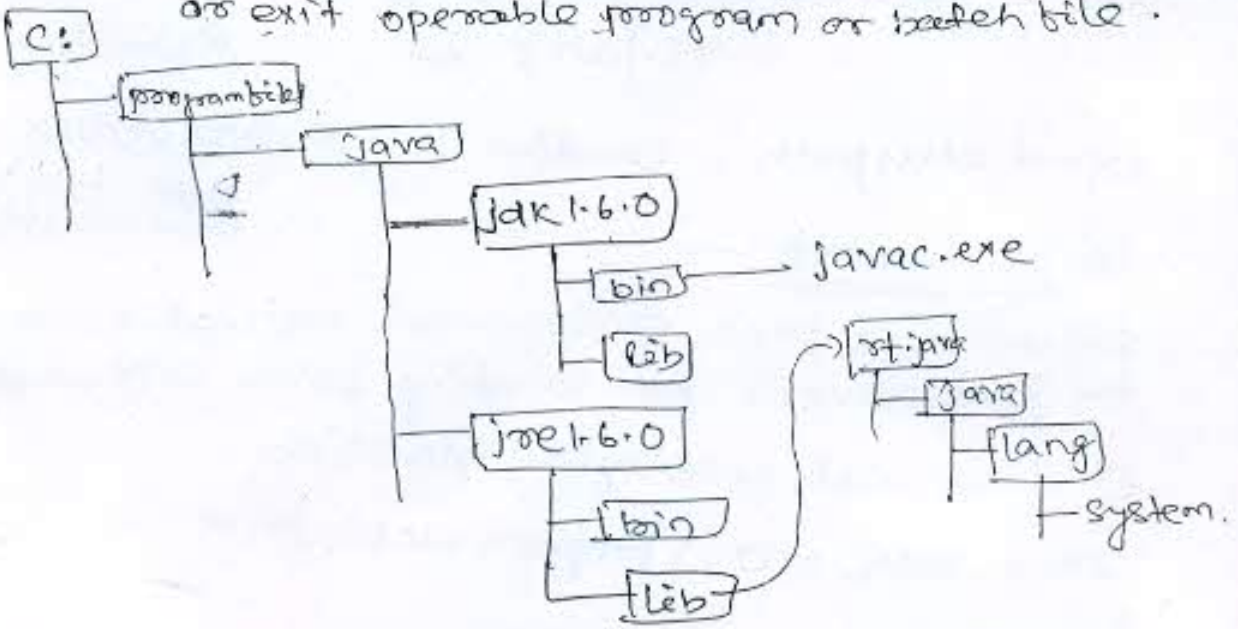
      start → run → cmd → OK

      cd \ ↵

      cd batch9am ↵

      C:\batch9am > javac prog1.java ↵

      error.. javac is not recognized as an internal or exit operable program or batch file.



So we can set the path.

Java Environment :—

→ Java uses 3 environment variables.

i) path

ii) classpath

iii) JAVA_HOME

Path :—

→ Path is used for locating executable files/ binarey files (i.e bin).

→ In order to access binarey files outside java software, we need to set path.

C:\batch 9am> path = C:\programfiles\java\jdk1.6.0\bin ; ↵

> javac program1.java ↵

Classpath :— (Location).

→ Class path is java environment variable used by java compiler & JVM for locating class files.

Set classpath = C:\program files\java\jre1.6.0\lib\rt.jare ; ↵ . // windows.

export classpath = Location ; // UNIX, LINUX.

Date - 12/9/2012

JAVA_HOME :—

→ This is a java environment variable used by java servers for locating java softwares.

ex :- Tomcat, weblogic, glassfish.

JAVA_HOME = C:\program files\java.

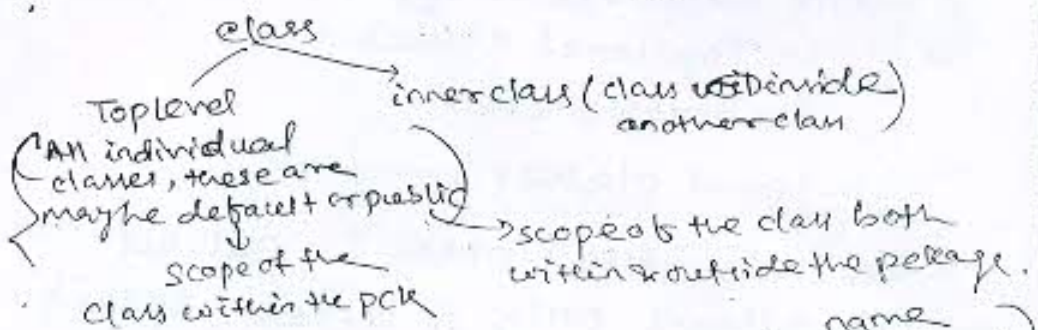
Program :—

program2.java

```
void main()
{
}
```

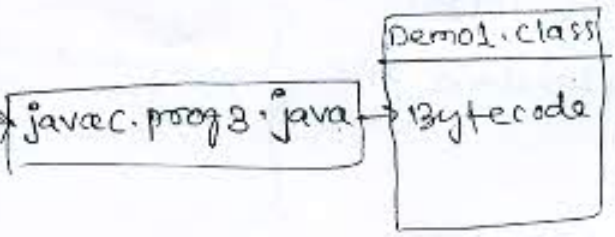
javac program2.java → class, interface, enum expected.

→ The above program display compile time error beco2 java is a pure object oriented language. Everything in java must encapsulated within the class.



Program3.java

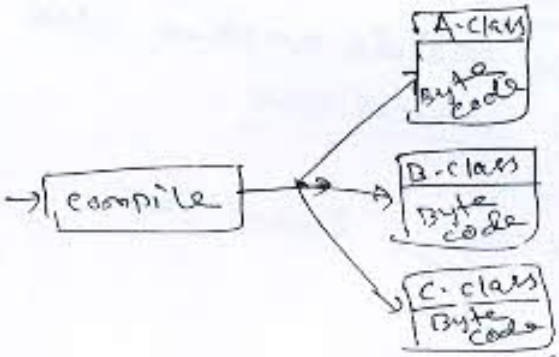
```
class Demo1
{
  void main()
  {
  }
}
```



.class files → (with class name & class is reusable)

Prog4.java

```
class A
{
}
class B{ }
class C{ }
```



→ On compiling of javac program, java compiler generates a .classfiles.

→ For everey class in java source program, compiler generates .classfile, becoz class is a reserable program or exeecutable program.

→ This class file contain bytecode

Program4.java

```
public class Demo2
{
  void main()
  {
  }
}
```

→ javac program4.java → Error: Class Demo2 is public, should be saved with a filename Demo2.java;

→ Every public class should be saved with class name.

→ Java supports 2 types of classes —
1. Top-level classes
2. Inner classes.

Top-level classes :—

→ An individual class is called top-level class.

→ It allows only 2 access specifier.
a) public
b) default.

ex: ~~private~~ class Abc { } ~~protected~~ class Prit { }

Inner classes :—

→ Class resides inside another class it is called inner class or nested class.

ex: class A { class B { } } → Inner class.

When we

C:\batch9am> java Demo1 ↵

→ Java invoke JVM which load Demo1.class from harddisk to RAM;

→ After loading it verify the bytecode is generating ~~proo~~ property or not.

→ Then look for main method.

→ On execution of Demo1 class JVM throws an exception/error, No such method Error: main.

Syntax :-

@ public static void main (String args[])
{
}

(b) public static void main (String[] args)
{
}

(c) static public void main (String args[])
{
}

(d) public static void main (String... args) (Java 5.0)
{
}

① ② ③ ④ ⑤ ⑥
Public static void main (String args[])
{
}

1 → Access specifier
2 → modifier
3 → Return type
4 → method name (user-defined class)
5 → Class name (pre-defined class & available in java.lang package). → rt.jar
6 → ~~args~~ Name of the array (identifier).

Q) Why main method is public?
→ Public is an access specifier.
→ main() is public bcoz it is access outside the environment or package.

Static :—
→ It is an access modifier.
→ This method is called without creating object of class.
→ Static methods binding with class name.

:—

⇒ it method doesn't retearen any value.

main():-

→ It is the name of the method understand by JVM.

String args[]:—

→ This method is called from OS.
→ The values which are send from operating system environment are of type String.
→ It is array beoz it receives 0 or more values.
→ It is also called command line arguments.

Programs:-

```
class Demo3
{
public static void main(string args[])
{
  System.out.println("Inside main of String");
}
public static void main(int args[])
{ System.out.println("Inside main of int");
}
public static void main(float args[])
{ System.out.println("Inside main of float");
}
}
```

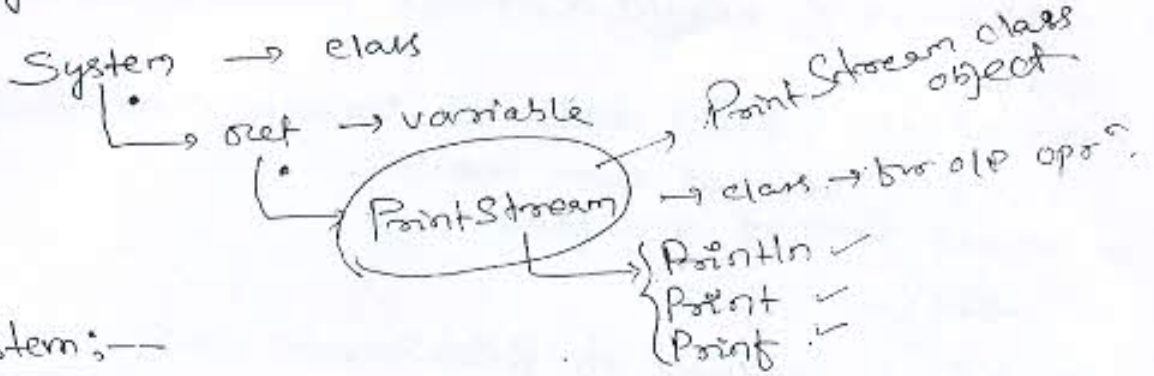
output:-

javac programs.java ↵
java Demo3 ↵
Inside main of string.

Date-13/9/12

Printing methods :—

1) System.out.println
2) System.out.print
3) System.out.printf → Java 1.5/5.0

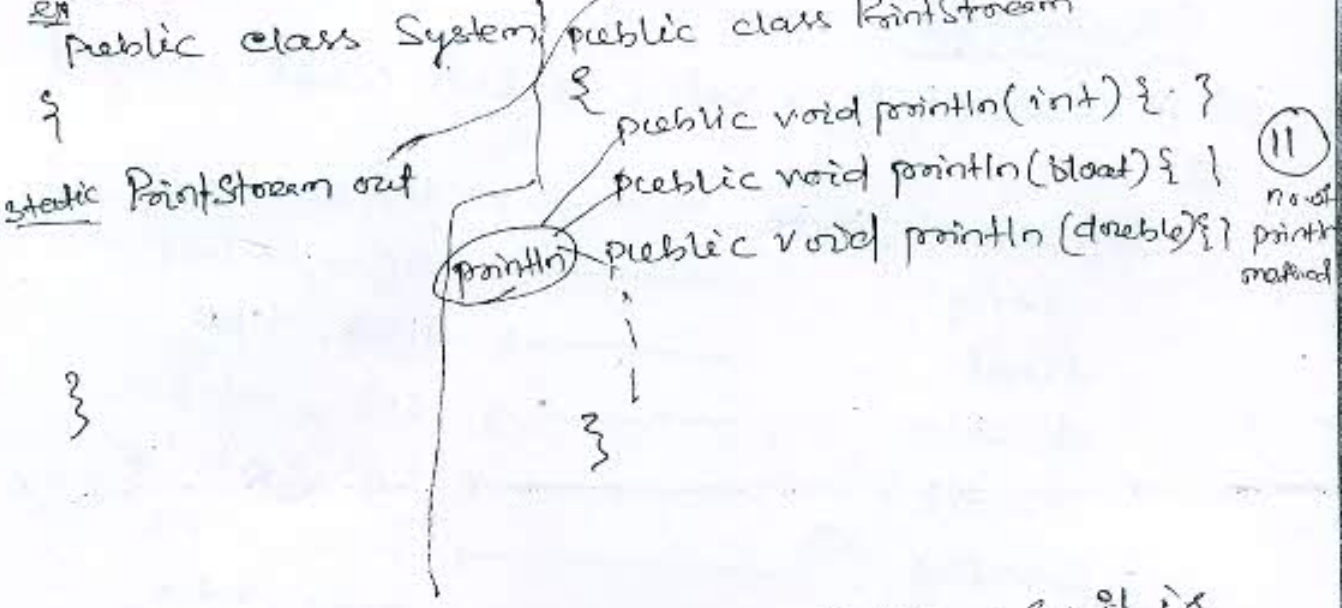


System :—

→ System is a pre-defined class.
→ This class available in java.lang package.
→ This default class provided by Java.

out :—

→ It is a variable of PrintStream class.
→ It is a reference variable which holds an address of PrintStream object.



→ It is a static reference variable. So it is binding with class name i.e System.out

println :—

→ println is a method of PrintStream class.
→ This method is overloaded inorder to print various types of values.
→ Number of println methods available are 11.
→ println, print data on console (monitor)/CUI)
*CUI → Command user interface
and insert newline.

print :——

→ It is a method of PrintStream class.
→ It print data without inserting a newline.
→ No. of print methods are 10.

printf :—

→ It is a method of PrintStream class.
→ This method print various types of values.
→ It print one or more than one value.
→ No. of printf methods are 2.

Constants :———

→ A constant is a value which never changed.

ex:

int, short, byte	→	100, 200, 0, -100, -200
long	→	100L, -100L
float	→	1.5f, 1.5F
double	→	1.5, -1.5
char	→	'A', '0', '*'
String	→	"java", "1.5", "100"
Boolean	→	true, false
Reference	→	null

Program :-

```
class Demo4
{
public static void main(String args[])
  {
    System.out.println(100);
    System.out.println(1.5);
    System.out.println(2.5f);
    System.out.println(200L);
    System.out.println('A');
    System.out.println("Java");
    System.out.println(true);
    System.out.println(null); error
  }
}
```

→ System.out.println(10,20); // Invalid. It access only one argument

System.out.printf("%d %d", 10, 20);

System.out.printf("%s %f", "Java", 1.5f);

→ printing multiple values using printf

Output :- path = C:\progfiles\Java\JDK1.6.0\bin ↵

javac prog6.java ↵

System.out.println(null);

→ The above statement gives compile time error because, null constant cannot print directly.

→ In order to print null, it is stored inside some variable (reference type) and printed.

Local variables :-

i) → A variable declared inside method is called local variable.

ii) → It is having local scope.

iii) → A local variable must assign value before accessing it.

ex:

C

```
void main()
{
  int x, y, z;
  y = 20;
  z = x + y;
  printf("%d", z);
}
```

no error

→ Garbage collection is performed.

→ Not a strongly typed language.

→ Array may be static or dynamic

Java

```
class Demo
{
  psvm(String args[])
  {
    int x, y, z;
    y = 20;
    z = x + y;
    S.O.P(z);
  }
}
```

error.

→ No Garbage collection is performed.

→ It is a strongly typed lang. i.e Robust.

→ Array must be dynamic

examples :

```
class Demo5
{
  public static void main(String args[])
  {
    int x, y;
  }
}
```

output : javac prog7.java ↵
Java Demo5 ↵

→ The above program does not display any compile time error bcoz x, y values are not access by program.

```
class Demo6
{
public static void main (String args[])
 { int x, y;
   System.out.println (x);
   System.out.println (y);
 }
}
```

output: javac prog7.java ↵

error:

→ The above program display compiletime error becoz x & y variables not assign any variable.

→ To overcome this error we assign the values to x & y.

iv) → Local variables does not allows any modifiers except final.

example:

```
class Demo7
{
public static void main (String args[])
 { int x = 100;
   long y = 200L;
   float f = 1.5f;
   double d = 2.5;
   boolean b = true;
   String s = null;
  System.out.println (x);
  System.out.println (y);
  System.out.println (f);
  System.out.println (d);
  System.out.println (b);
  System.out.println (s);
 }
}
```

* boolean b = 1 // error

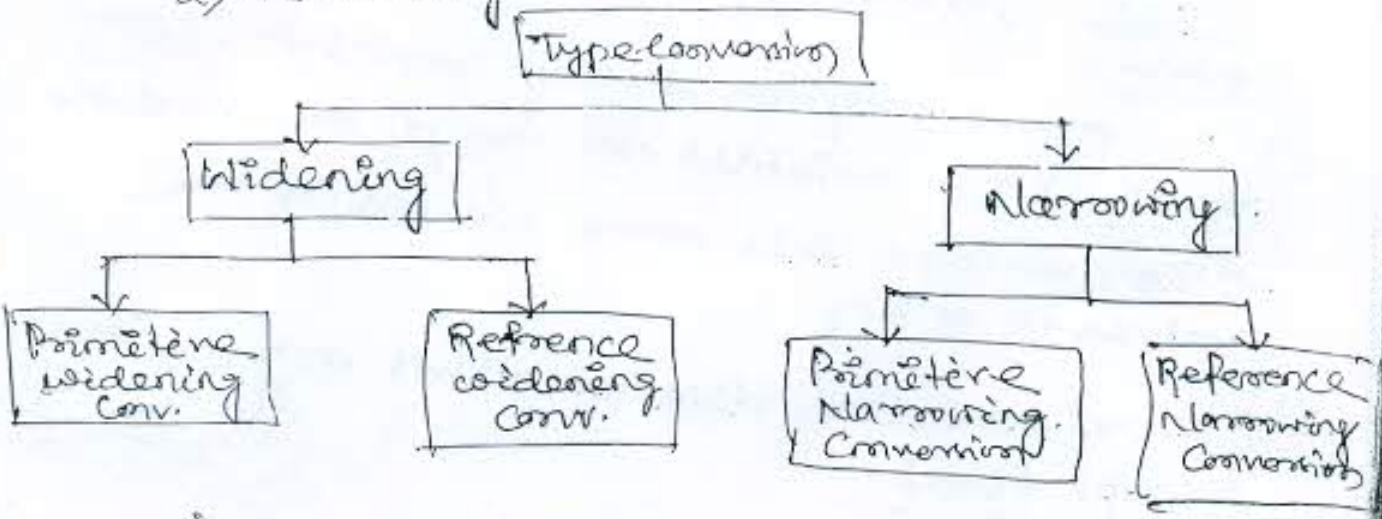
Date - 14/9/19

Type Conversions :—

→ Converting one type of value to another type is called "type conversion".

→ This type conversions are 2 types —

1) Widening conversion.
2) Narrowing conversion.



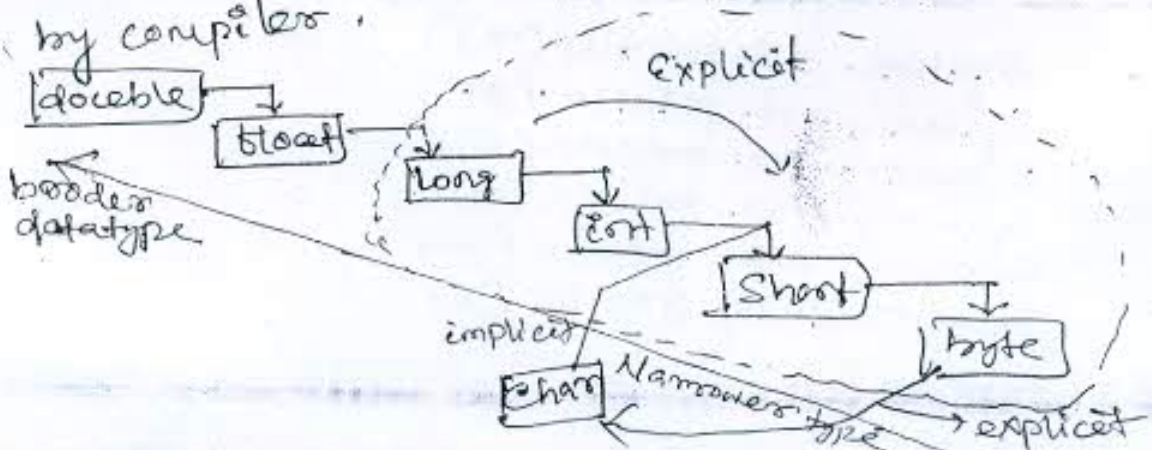
int — narrowerdatatype
float — broaderdatatype → more precession.

1) Widening Conversion :—

→ Converting of narrower datatype to broader datatype is called "widening conversion".

→ Converting narrower primitive datatype to broader primitive datatype is called widening primitive conversion.

→ This conversion is implicit and it is done by compiler.



Byte :-

→ byte is 1 byte integer datatype.
→ It accept value which range from -128 to +127.

example:

```
class CDemo1
{
public static void main(String args[])
{
    byte b1 = 100;
    byte b2 = 200;
}
}
```

output :-

→ The above program display compile time error.
becoz '200' is not ~~beto~~ within byte range.
→ 200 is of integer value.

example:-

```
class CDemo2
{
public static void main(String args[])
{
    byte b1 = 100;
    byte b2 = 20;
    byte b3 = b1/b2;        // any arithmetic opr^n (+, -, *).
    System.out.println(b3);
}
}
```

output :-

→ The above program display compile time error
becoz any arithmetic operation on byte
result it of type integer.
→ The conversion between integer and byte
is not implicit.

Note:

```
byte b3 = 10+20;    // valid, no error.
```

10 + 20 → {int value / integer constant}

30 → it is the constant & in side the byte range

→ Byte performance case

```
byte b3 = 10+20;
```

→ The above program doesnot display any compiletime error. Any arithmetic opr on integer constants, result can be byte, short, int or long.

Program 3:-

```
class CDemo3
{ p.s.v.m(String args[])
  {
    byte b3 = 100*20;
    S.o.P(b3);
  } }
```

output:-

→ Loss of precision, exceed the range.

Chars:-

→ Char is character datatype and 2bytes integer datatype.

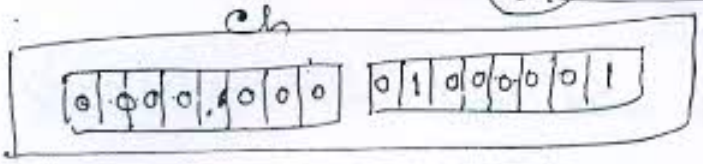
→ The range of char is 0 to 65535 "(UNICODE)

Program:-

```
classDemo4
{ public static void main(String args[])
  {
    byte b = 65;
    char c = b;
    S.O.P.(b);
    S.O.P(c);
  }
}
```

Char ch = A;

(65) UNICODE



→ error:- The above program display compile time error. because conversion between byte & character is not implicit.

→ Conversion between integer constant and characters is implicit. → integer constant.

ex: char ch1 = 65;
char ch2 = 'A';
char ch3 = 97;
sop (ch1); → A
sop (ch2); → A
sop (ch3) → a

→ So the conversion is explicit.

int :—

→ Integer datatype whose size is 4 bytes.
→ Range of integer is −2147483648 to 2147483647.

Program :— (for range)

```
Class D
{
public static void main (String args[])
{
S.O.P (Integer.MAX_VALUE);
}
}
```

Short :—

→ It is also an integer datatype, whose size is 2 byte.
→ Its range is −32768 to 32767.

Program:-

```
Class CDemo5
{
  public static void main (String args[])
  {
    byte b1 = 65;
    short s1 = 100;
    int i1 = b1;      } -> implicit conversion.
    int i2 = s1;
    S.O.P (b1);
    S.O.P (s1);
    S.O.P (i1);
    S.O.P (i2);
  }
}
```

output:

```
65
100
65
100
```

Program:

```
Class CDemo 6
{
  public static void
  {
    char ch = 'A';
    int i = ch;
    S.O.P (ch);
    S.O.P (i);
  }
}
```

output: A 65

Long :—

→ It is a integer datatype whose size is 8 bytes.

→ The range of long is

-9223372036854775808 to
+9223372036854775807

Program :—

```
class CDemo6
{ public static void main (String args[])
  { int x = 100;
    byte b = 65;
    char c = 'B';
    long l1 = x;
    long l2 = b;
    long l3 = c;
    S.O.P (l1);
    S.O.P (l2);
    S.O.P (l3);
  }
}
```

O/P 100
65
66

Program :

```
class CDemo7
{ psvm (S args[])
  { long l1 = 100;
    long l2 = 65536;
    long l3 = 2147483647;
    long l4 = 2147483649L; // exceed the integer range
    S.O.P (l1);
    S.O.P (l2);
    S.O.P (l3);
  } S.O.P (l4);
}
```

float :—

→ It is a real datatype having precisions.
→ Size of this datatype is 4 bytes.
→ Range of this data type is

3.4028235E38 E=10,
to
= 1.4E-45 i.e 10^{-45},

→ float value must be prefix with f or F.

example :-

```
class cDemo8
{
  psvm(String args[])
  {
    int x = 100;
    byte b = 65;
    char ch = 'A';
    float f1 = 1.5f;
    float fa = x;
    float f3 = b;
    float f4 = ch;
    System.out.printf("%.f %.f %.f %.f", f1, f2, f3, f4);
  }
}
```

```
class Demo9
{ psvm(string arg[])
  float b1 = 100;
  float b2 = 100.0;
  float b3 = 100.05;
}
```

double :—

→ Double is real datatype it contain precesion.
→ The size of datatype is 8 bytes.
→ Range of these is

1.79769313 4862315 7E308.
to
4.9E-324.

```
class CDemo9
{ public static void main(String args[])
  { int x = 100;
    char y = 'A';
    byte b = 65;
    float f = 1.5f;
    long l = 100;
    double d1 = 1.25;
    double d2 = x;
    double d3 = y;
    double d4 = f;
    S.O.P(d1);
    S.O.P(d2);
    S.O.P(d3);
    SO.P(d4);
  }
}
```

OP

1.25
100.0
65
1.5f.

examples:

```
double d1 = 100;  → L
double d2 = 100f; → L
double d3 = 100L; → L
```

Date-15/9/12

Narrowing Primitive Conversion:-

→ Converting bigger primitive datatype to narrower primitive datatype is called narrowing primitive conversion.

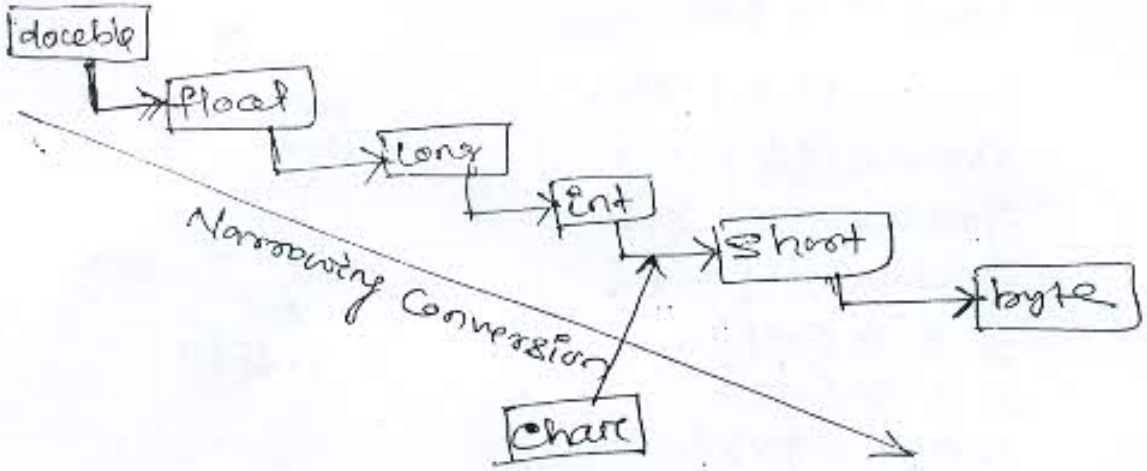
→ This conversion is explict. It has to done explicitly by programmer.

→ There is a chance of loss of value.

→ This conversion is done by using type casting operator.

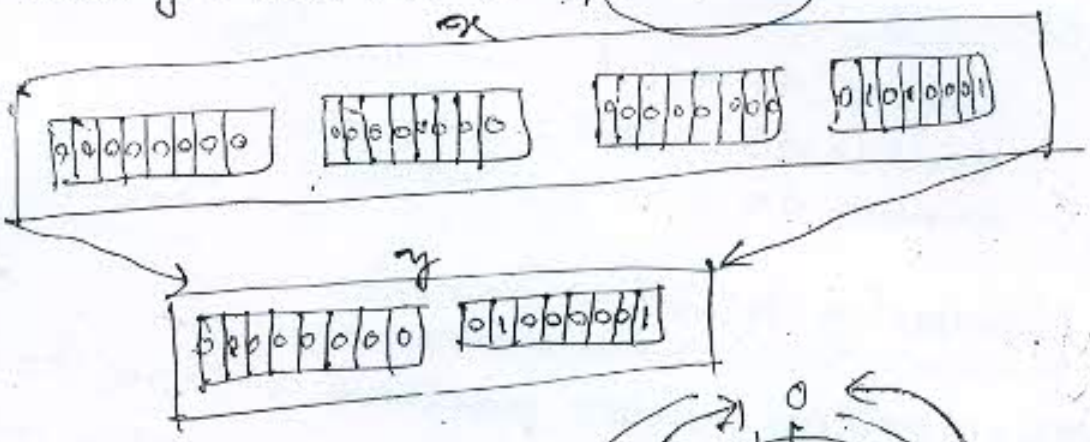
→ Syntax :-

(type) → destination type.

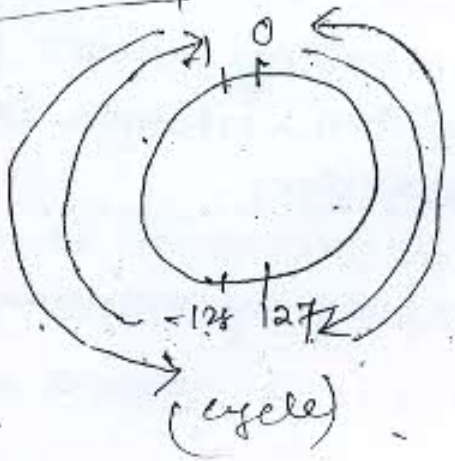



```
int x = 65;
short y = x;
```

```
int x = 65;
short y = (short) x;
```




```
short x = 200;
byte y = (byte) x;
S.O.p(x); → 200;
S.O.p(y); → -56;
```



such kind of conv. are not done in java.

```
class CDemo10
{ public static void main(String args[])
  {
    short x = 200;
    byte b = (byte) x;
    System.out.println(x);
    System.out.println(b);
  }
}
```

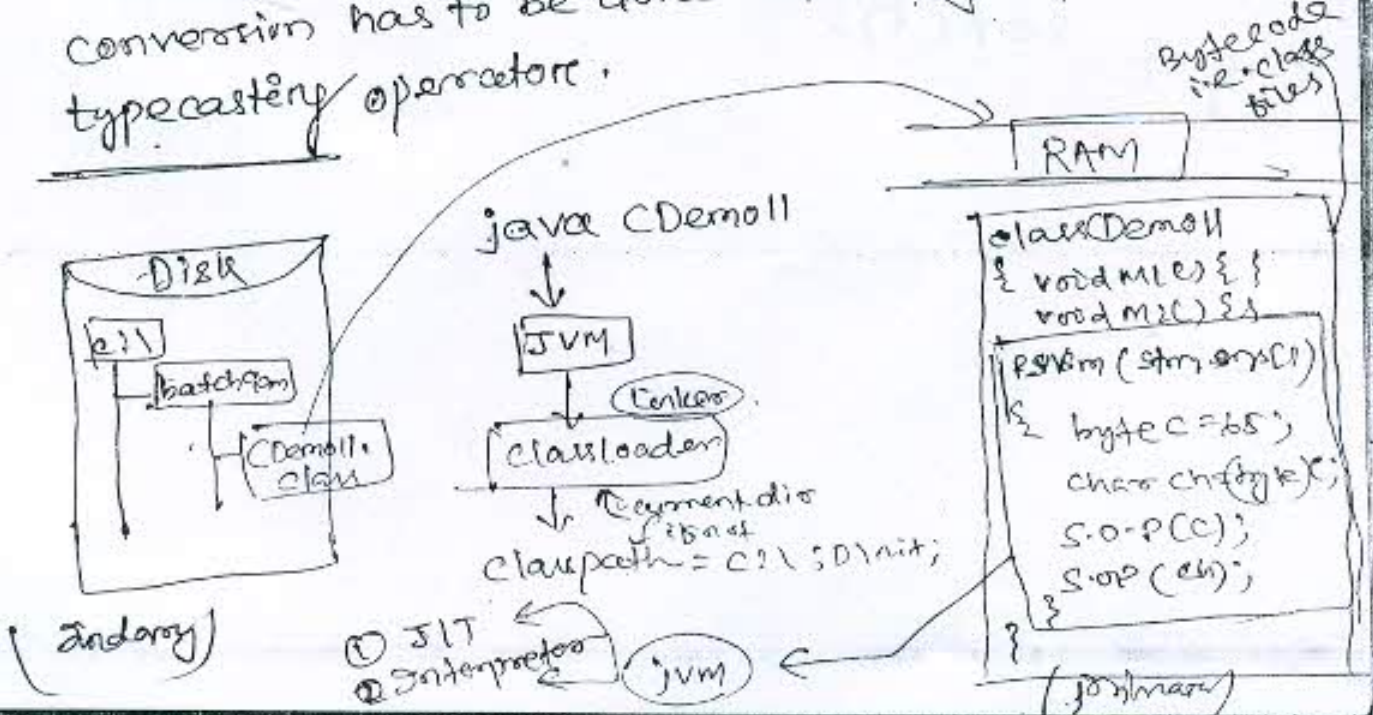
output
200
-56

Program :

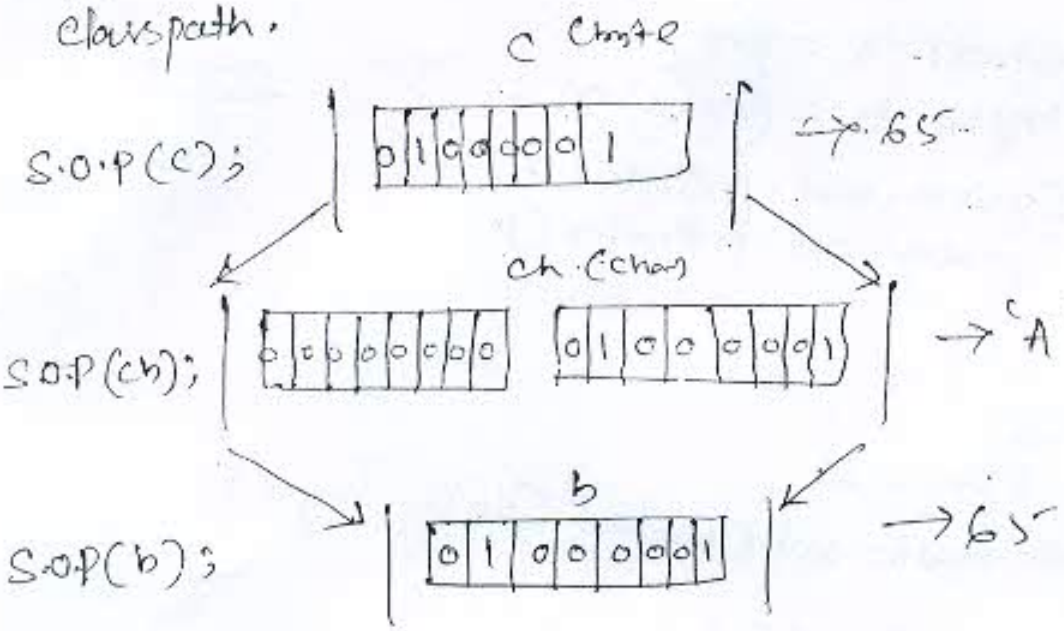
```
class CDemo11
{ public static void main(String args[])
  {
    byte c = 65;
    char ch = (char) c;
    byte b = (byte) ch;
    S.O.P(c);
    S.O.P(ch);
    S.O.P(b);
  }
}
```

o/p
65
A
65

→ conversion between byte to character and character to byte is not implicit. This conversion has to be done explicitly using typecasting operator.



→ whereever an error comes class is not found or class is not defined then set the classpath.



example:

```
class CDemo12
{ psvm (String args[])
  { double d = 1.5;
    float f = (float) d;
    int i = (int) d;  // Loss of precision.
    S.O.P(d);
    S.O.P(f);
    S.O.P(i);
} }
```

Arrays :-

→ Array is a collection of similar type of data elements.

→ An array is a reference datatype, ~~struct~~ holds the address.

→ Array is in java are dynamic.

→ JVM allocates / reserve memory for every java program, which is divided into 2 parts.

1. Static memory Area. (SMA)
2. Dynamic memory Area (DMA)

→ Static memory area is un managed memory area which cannot managed by java programmer. This memory is managed by JVM.

ex: local variable.

→ Dynamic memory area is managed memory area which is managed by java program.

→ DMA avoids the wastage of memory.

Memory management → JVM → memory manager

mem. reserved ← → mem. unreserved

→ Array is for avoiding more no. of variables ~~is~~ required for grouping.

new :—

→ This operator allocates memory within dynamic memory area (heap area).

→ new reserve memory and return address of reserved memory.

Date- 16/9/12

Types of Array :—

1) Single Dim. Array.
2) Multi Dimension Array.
3) Jagged Array.

Single dimension Array :—

→ An array which is refer with one subscript is called single dimension array.

Syntax :—

1. `datatype array_name[ ];`

Eg: int a[5] — C ✓ — Java ✗ → As in java array is dynamic.

int a[ ] ✓ → memory allocated for reference variable.

2. `datatype[ ] array_name;`

→ For declaration for array the memory is allocated for reference variable.

Syntax for Creating array :—

`new type_name [size];`

→ One creation of array memory is allocated for elements / data.

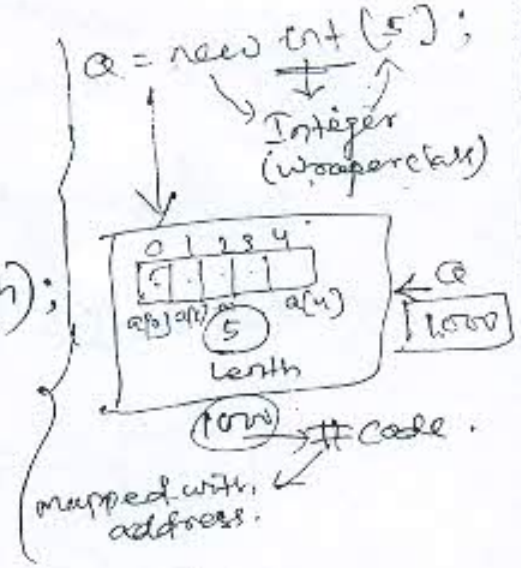
→ Arrayay is an implicit object.

Size Rules:

1) Size can be a constant or variable.
2) Size must be >=0.

example:

```
class ArrayDemo1
{
  public static void main (String args[])
  {
    int a[];
    a = new int[5];
    System.out.println(a);
    System.out.println(a.length);
  }
}
```



O/P

[I@3e25a5

5

→ [I @3e25a5

[→ Single Array
I → Integer
@3e25a5 → hashcode → address.

→ [[→ 2D array
[[[→ 3D array

example-2

```
class ArrayDemo2
{
  psvm(String args[])
  {
    int a[];
    a = new int[-2];
    S.O.P(a);
    S.O.P(a.length);
  }
}
```

*) The above program displays an error during runtime, because an array cannot be created with -ve size.

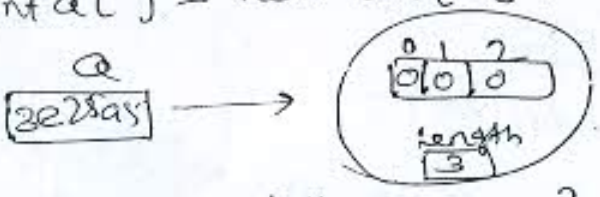
How to Initialization of Array :-

Syntax :-

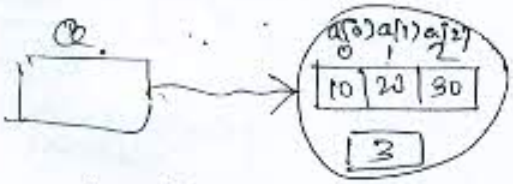
datatype array_name[] = {list of values};

example: declaration, creation

1. int a[] = new int[3]; ← declaration, Creation.



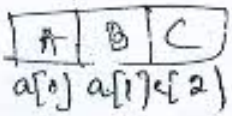
2. int a[] = {10, 20, 30} ← Initialization.



3. float b[] = {1.5f, 2.5f, 3.5f};

4. char c[] = {'A', 'B', 'C'};

char c[] = {"ABC"}; → (A group of char. created)



Note :-

5 float a[] = {10, 20, 1.5f}; ✓ (implicit conversion).

int b[10] = {10, 'A'}; ✓ (widening conv.)

double d[] = {10, 1.5, 2.5f, 'A'}; ✓ Valid.

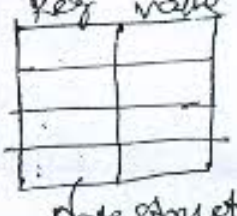
SOP(a); → [F@ ——

SOP(b); → [I@ ——

SOP(d); → [D@ ——

* Array is built in a datastr. called hashtcode. It contain 2 things

#code



(Data structure)

→ Reading and writting elements within array is done using index.

→ Java 5.0 supports enhanced for loop, which is used for reading elements from array without using index.

Syntax :- (for enhanced for loop)

for (variable : array) statements ;	for (variable : array) { statements ; } ... multiple statements.

example

```
int a[] = {10,20,30,40,50};
SOP(a[0]); → 10;
SOP(a[1]); → 20;
```

1. `for(int i=0; i<a.length; i++)` → This can be used for any purpose.
   `S.OP(a[i]);` → 10,20,30,40,50.

2. `for(int n : a)` → enhanced for loop.
   `SOP(n);` • This can only use for reading & display the elements from array.

program :- • Reading the arrays & collection → collection of different type of data

```
class ArrayDemo2
{ public static void main (String args[])
  { int a = {10,20,30,40,50};
    for(int n : a);
    { if(n%2 == 0)
        S.O.P("Even no.");
      else
        S.O.P("Odd no.");
} } }
```

Multi dim. Array :— Date - 17/9/12

2D array :—

→ An array of arrays is called 2D array, or multi dim array.

→ This array is referred using two subscripts.

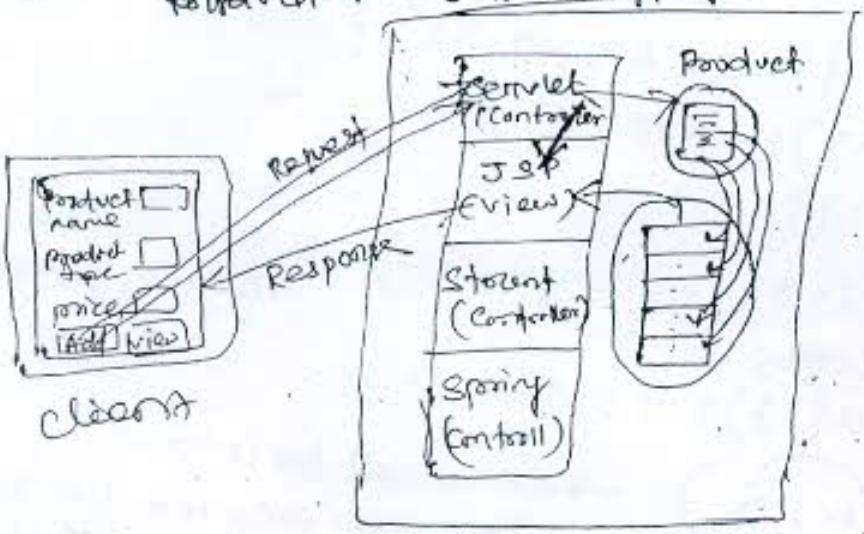
→ It is a matrix.

Syntax :-

```
datatype array_name[][];
datatype[][] arrayname;
```

ex :

Product Online shopping

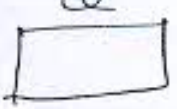


→ Array is may be primitive type or Collection of diff. types.

Syntax for Creating an Array :—

```
new datatype[rsize][csize];
new datatype[rsize][];
```

ex:- `int a[][];`

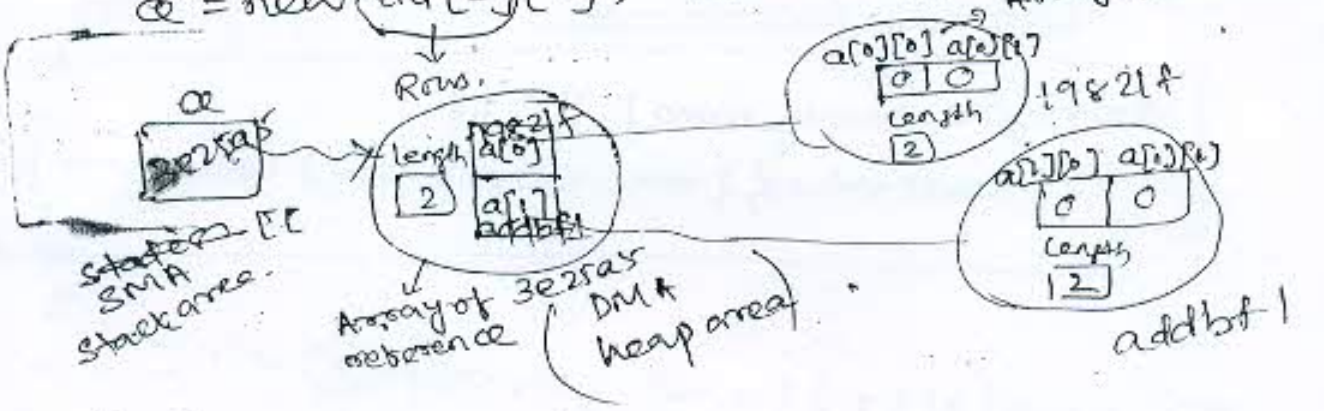


Q) What is hash code?

→ The hash code of java object is simply a number (32bit sign int) that allows an object to be managed by a hash-based data structure.

→ It contain 2 things: 1) keys
2) values.

`int a[3][3]` → X → invalid.
`int a[3][]` → invalid
`int[][] a` → valid.
`a = new int[2][2];` ← Array of arrays is called 2D. Array of values.

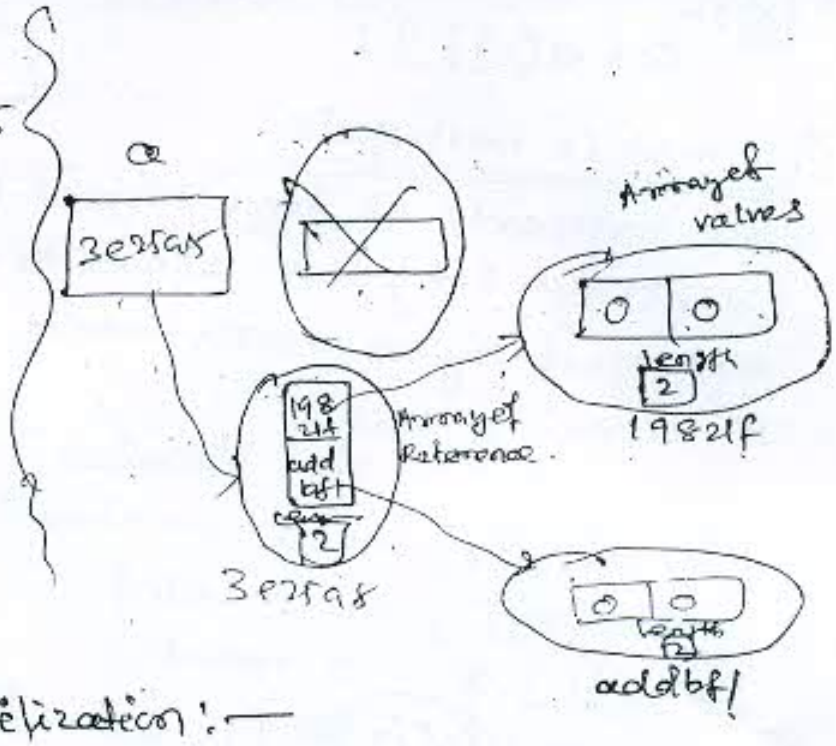


Program:-

```
class ArrayDemo3
{ public static void main (String args[])
  { int a[][];
    a = new int[2][2];
    System.out.println(a);
    System.out.println(a.length); // no. of rows.
    System.out.println(a[0].length); // no. of columns.
    System.out.println(a[1].length);
    System.out.println(a[0]);
    System.out.println(a[1]);
  }
}
```

Output :—

```
[[I@3e25a5
 2
 2
 2
[I@19821f
[I@addbf1
```



Syntax for Initialization :—

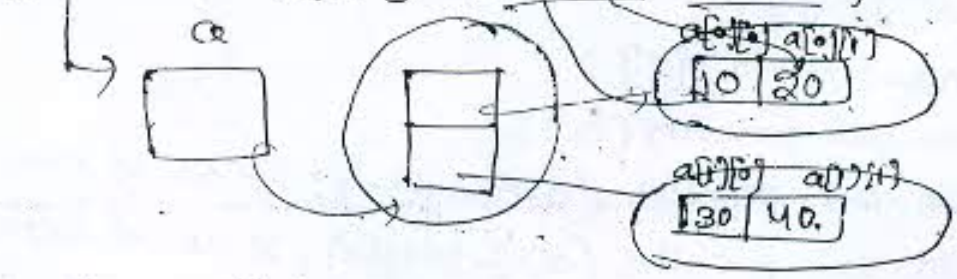
```
datatype array_name[][] = {
    {{row1 values},{row2 values},{row3 values},.....};
```

ex :- int a[][] = {{10,20},{30,40}} → (X) → in C.

↳ In C col. size is mandatory.

int a[][2] = {{10,20},{30,40}} → ✓ → in C.

int a[][2] = {{10,20},{30,40}} ✓ → In Java.



Program :—

```
class ArrayDemo4
{ public static void main (String args[])
  { int a[][2] = {{10,20},{30,40}};
  }
}
```

op: Error.

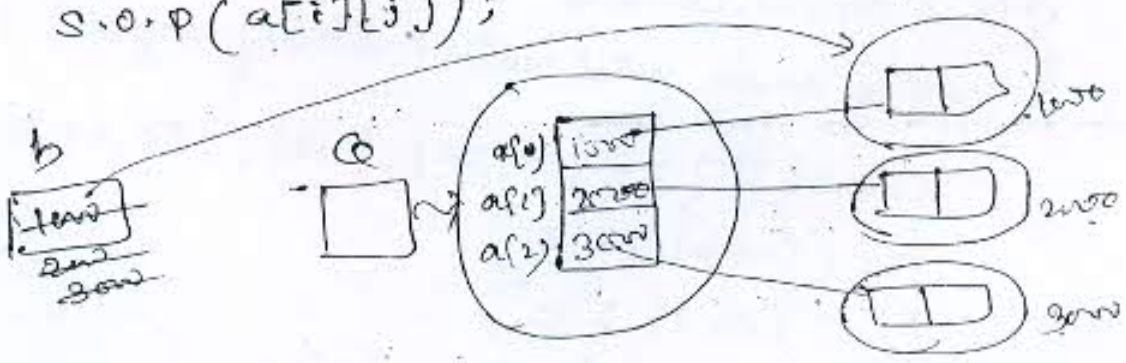
✱ The above program displaying compile time error. becoz array cannot be initialize with size.

Reading Elements from 2D Array :—

```
Class ArrayDemo5
{ public static void main (String args[])
  { int a[][] = {{10,20},{30,40},{50,60}};
    for(int b[] : a)          // row
    {
      for (int n : b)         // column
      { S.O.P (n);
      }
    }
  }
}
```

(for)

```
for (int i=0; i<a.length; i++)
for (int j=0; j<a[i].length; j++)
  S.O.P (a[i][j]);
```

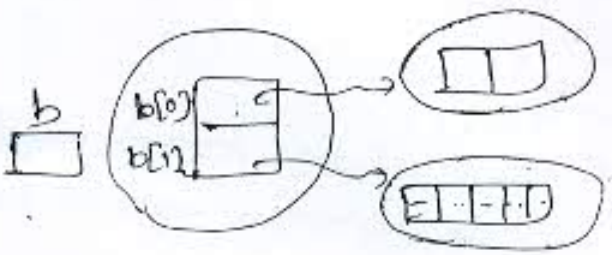


Jagged Array :-

→ Jagged Array is a multi-dimensional array having fixed no. of rows & variable length columns.

ex:- int b[][];

```
b = new int[2][];
b[0] = new int[2];
b[1] = new int[5];
```

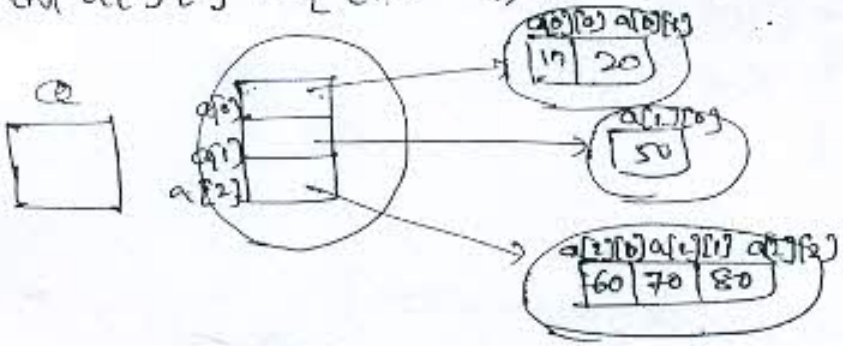


→ The above example shows the creation of jagged array.

→ For jagged array enhanced for loop is used.

Initialization of Jagged Array :-

int a[][] = {{10,20}, {50}, {60,70,80}};



Program :-

```
class ArrayDemo6
{ public static void main (String args[])
  { int a[][] = {{10,20},{40},{50,60,70}};
    for (int b[] : a)
    for (int n : b)
      System.out.println(n);
  }
}
```

Date - 18/9/12

Commandline arguements :-

→ The values which are send from command prompt to main() are called commandline arguements.

→ These arguements are of type String.

example :-

```
class CommDemo1
{
public static void main (String args[])
{
System.out.println(args.length);
for (String s : args)
System.out.println(s);
}
}
```

o/p
2
10
20

c:\batch9am>-

```
javac CommDemo1.java↵
java CommDemo1 10 20 ↵
```
→ command arguments.

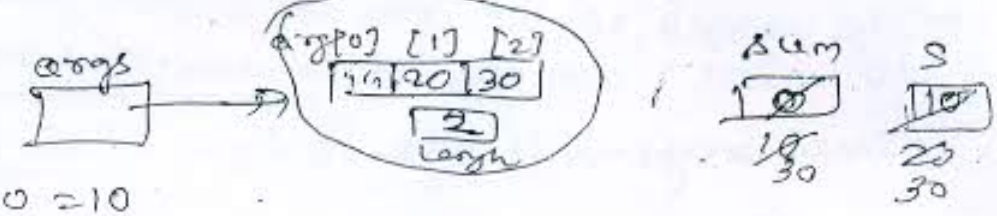
Program :-
Sum of numbers send from command prompt:

```
class CommDemo2
{
public static void main (String args[])
{
int sum=0;
for (String s : args)
{
sum = sum + Integer.parseInt(s);
}
System.out.println(sum);
}
}
```

O/P

javac CommDemo2.java ↵

java CommDemo2 ↵



0 + 10 = 10
10 + 20 = 30
30 + 30 = 60

Integer.parseInt() :—

→ Integer is a predefined class available in java.lang package.

→ parseInt is the static method of Integer class.

→ This method converts String representation of integer to int.

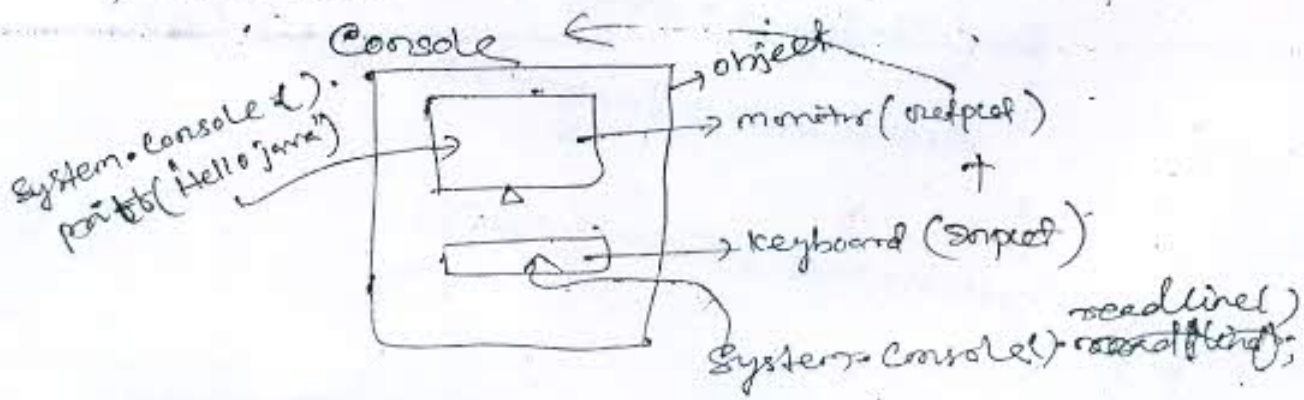
psvm(String ... args)
↳ Variable length arguments.
added in java 5.0.

System.console() :—

→ console() is a static method of System class.

→ This method returns reference of Console object.

→ This feature is added in java 6.0



Program:-

```
class consoleDemo1
{ public static void main (String args[])
  { System.console().printf("Hello Java");
  }
}
```

O/P Hello java

Program:- (instead of above)

```
class consoleDemo1.1
{ psvm(String args[])
  { Console cnt = System.console();
    cnt.printf("Hello java");
  }
}
```

//WAP to read 2 no.s from console & print sum.

```
class ConsoleDemo2
{ public static void main (String args[])
  { System.console().printf("\n Input any two numbers");
    String s1 = System.console().readLine();
    String s2 = System.console().readLine();
    int n1 = Integer.parseInt(s1);
    int n2 = Integer.parseInt(s2);
    int n3;
    n3 = n1 + n2;
    System.console().printf("\n sum is %d", n3);
  }
}
```

readLine() :-

→ It is a method of Console class which read strings.

→ If we want read the data from keyboard —

a) In C → Scanf()

b) In java → System.console().readline().

+ operators :- → Adding nos.
+ → Concatenating strings.

→ In java '+' operator perform 2 operations.

i) Adding numbers.

ii) concatenatey strings.

→ It add two numbers if both operands are of type number.

→ It perform concatenation, if any one operands are of type String.

→

Example :-

S.O.P (10+20); → 30

S.O.P ("10."+20); → "1020"

S.O.P (10+"20"); → "1020"

S.O.P ("10."+"20"); → "1020"

* S.O.P ("sum is "+10+20) → Sum is 1020

* S.O.P (10+20+"sum is") → 30 is Sum

① System.out.println()

② System.out.print()

③ System.out.printf()

④ System.console().printf()

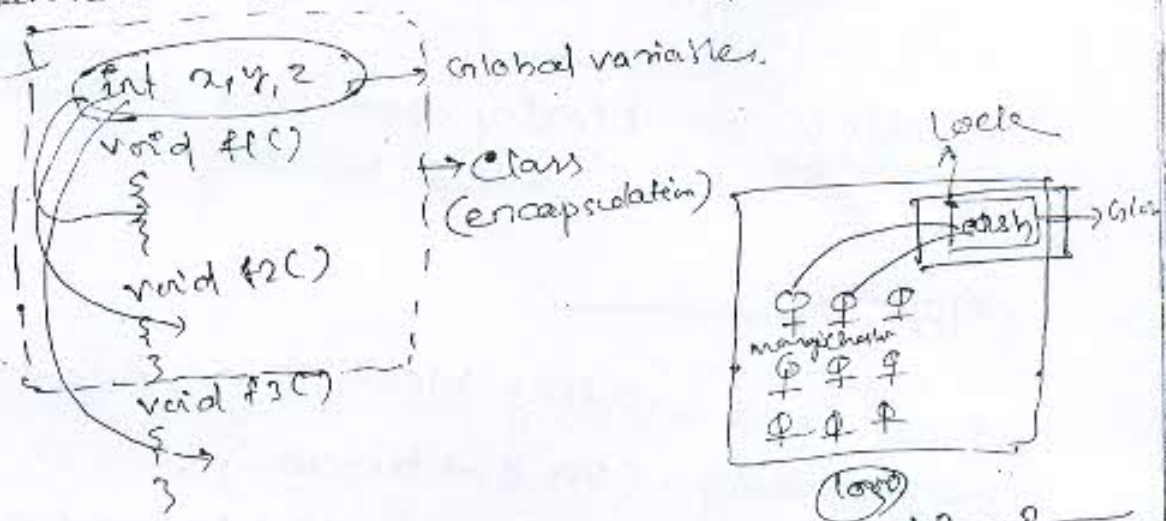
⑤ System.console().readline()

Object oriented Programming (OOP):- 20/9/12

→ Object-oriented Programming is a programming method, which define set of rules and regulation for organization of data & instruction.

→ Programming elements are consists of 2 types - ① Data. ② Instruction.

→ Organization of Data & Instruction o/c to giving problem is called programming.



→ Drawback of Structured/Procedural oriented Prog.:-

→ There is no proper organization of data & instruction.

→ Data is global, which can be access by related function and unrelated function.

→ Data is not secured.

→ Debugging application is complex. (i.e In a large program identifying which data is operated which operation (function) is complex.

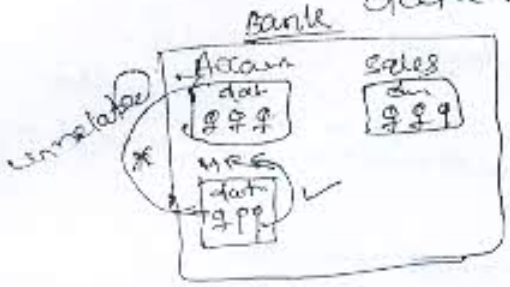
Encapsulation:-

→ It is a process of grouping data & instructions which operates on data (draft) within a single entity.

→ Wrapping of data & instructions is called encapsulation.

Adv. of encapsulation :—

1) Data hiding :- Preventing data access from unrelated operation is called data hiding.



2) Binding :— Linking operations with related data is called binding.

Class :—

→ Class is a building block of an object oriented programming. (In C → function).
→ Class is a collection of fields and methods : fields are nothing but variables.
→ Class is a blueprint of object.
→ Class defined the structure of object.
→ Class is planning before consisting object.
→ Class is datatype.
→ Class is metadata. i.e data of data.
→ Class is a reusable code component or module.

Syntax :- (optional)

```
[modifier] class class_type_name
{
    variables ; [fields]      (Java)
    functions ; [methods]
}
```

Object :—

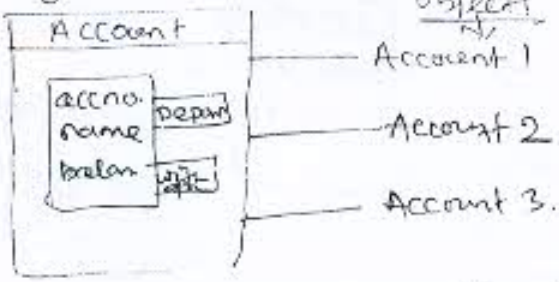
→ Object is an instance of an class.

→ In object oriented programming data is represented as objects.

Date - 21/9/12

→ An instance is nothing but allocating memorey for variables exist within class.

→ Using one class programmer can create any number of objects or instances.



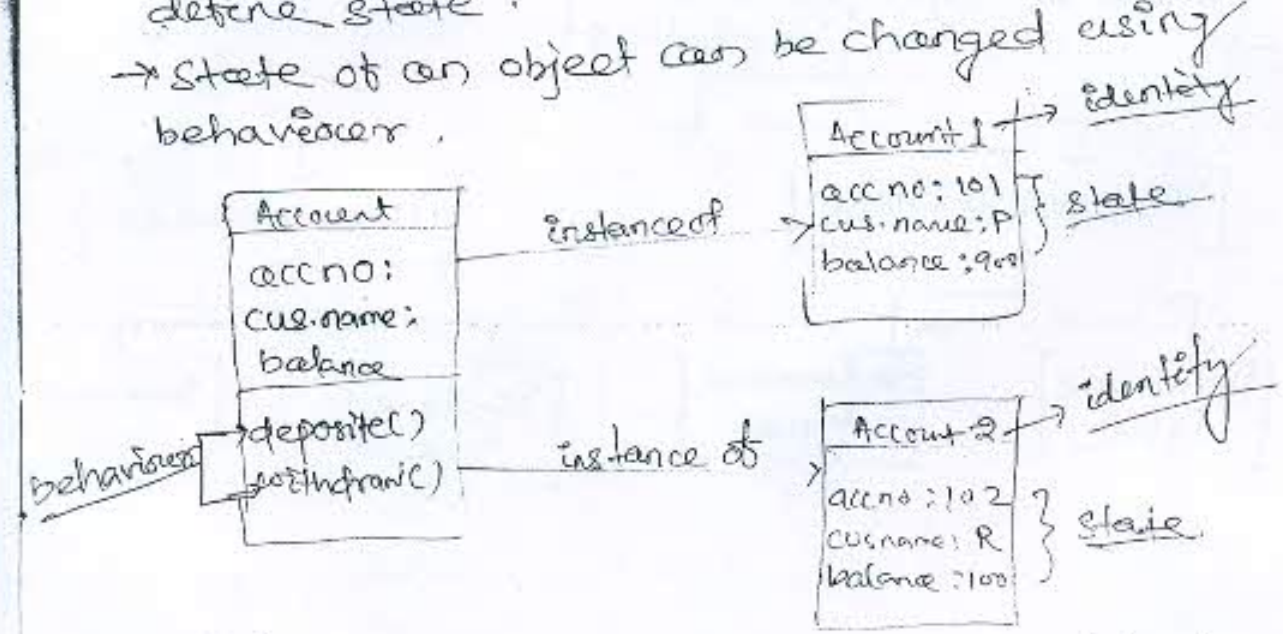
→ Every object is having 3 properties —

1). State
2) behaviour
3) identity.

State :—

→ The value given to an attribute of an object define state.

→ State of an object can be changed using behaviour.



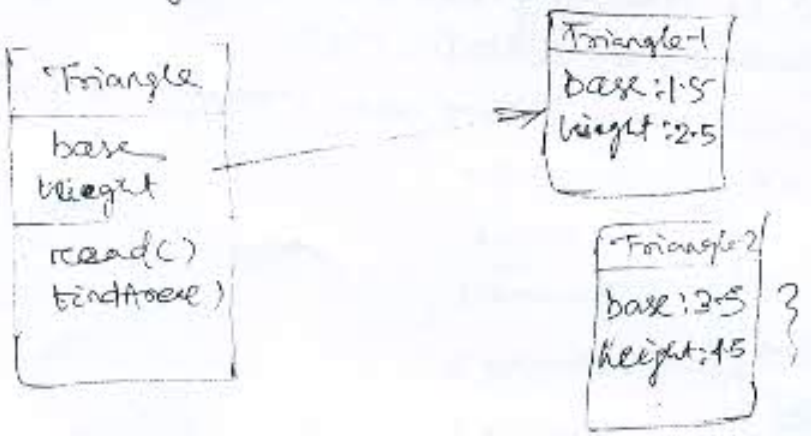
→ State is nothing but data hold by object.

Behaviour :—

It is an operation performed by ob on object is called behaviour.

Identity :—

Each object is identified by the unique name.



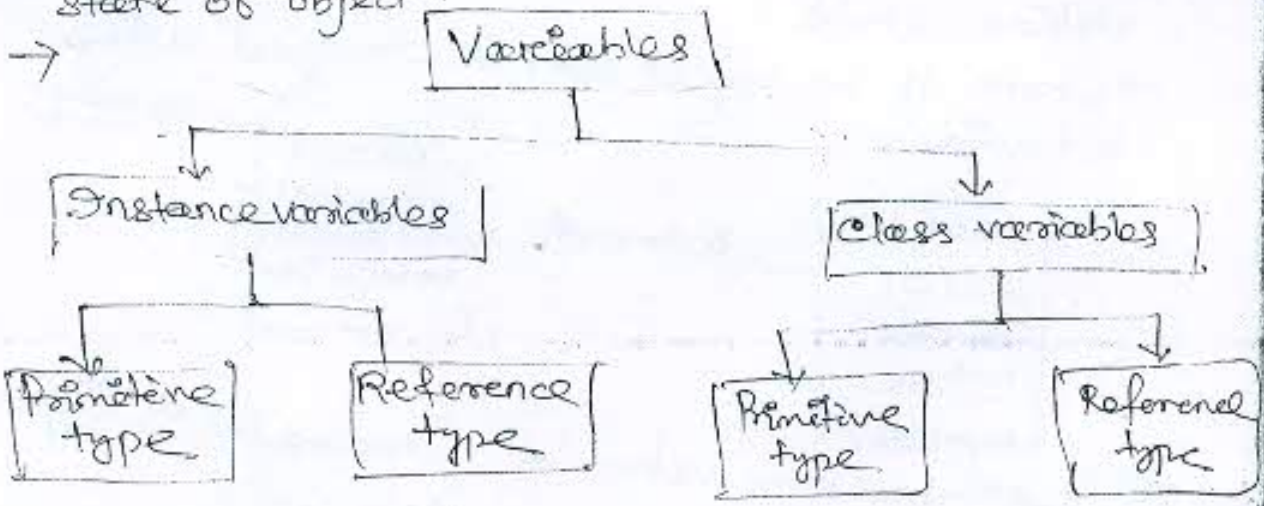
Variables :—

→ The variables declared inside class is of 2 types —

1) Instance variables.
2) Class variables

→ The variables declared within class define state of object.

→



Instance variables :—

→ A non-static variable of a class is called instance variable.
→ Instance variable memory is allocated on creation of object.
→ Every object have its own state, which is defined by using instance variable.
→ This variable bind with object name.

ex:-

```
class Book
{
    String bname;
    String author;
    float price;
}
```

- instance variable bcoz no static keyword is used.
- These are called when object is created.

How to create object ?

→ In java objects are dynamic.
→ These are created during execution of program
→ All objects are created within managed memory area called heap area.
→ Java allows to create object in 3 ways —

1) Using new operator.
2) class.forName("classname").newInstance()
3) factory method.

new operator :—

→ new is a operator which perform 3 operations —

1) Loading. (harddisk to RAM)
2) Instantiating. (allocating memory for non static variables)
3) Initialization.

Loading :-

→ Copying ".class" files or class from secondary to primary.

Instantiation :-

Allocating memory for non static variables of a class (creating object).

initialization :-

```
class A
{   int x;  → default value can be assign
    P.S.V.M ( S )
    {  int y; → x
    }
}
```

→ Assigning default values to object is called initialization.

Default value :-

	default value
int, short long, byte	0
float, double	0.0
char	\00000 → non printable value
boolean	false
Reference	null

ex :-

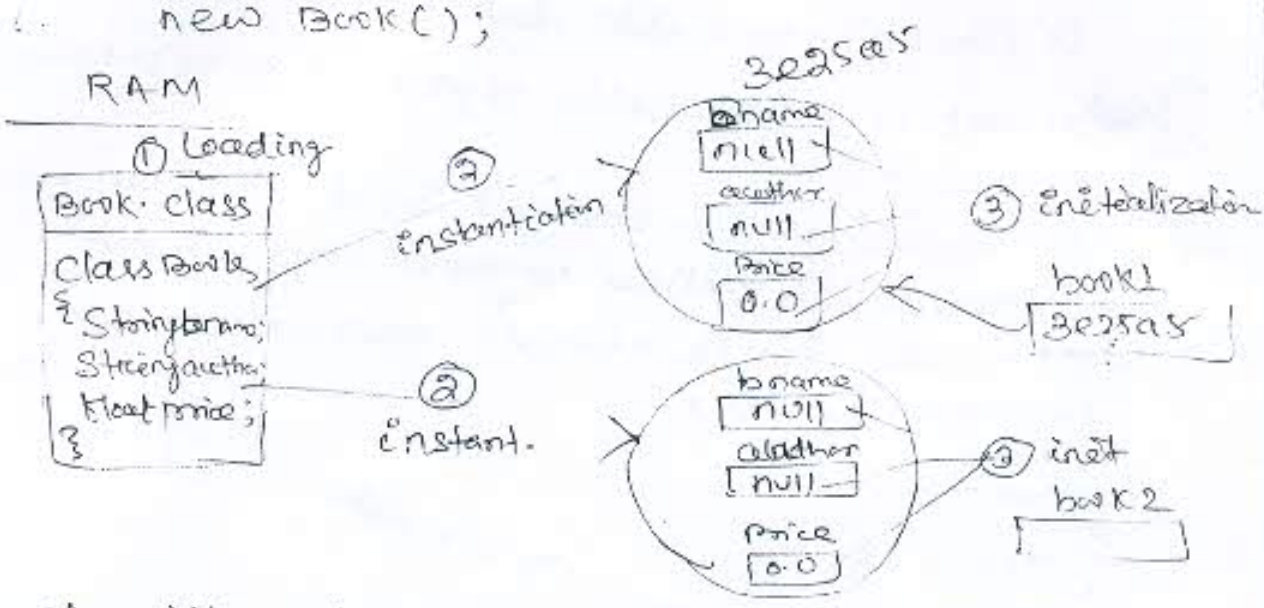
```
class Book
{   String bname;  → null
    String author;  → null
    float price;  → 0.0
}
```

default value

Syntax :-

```
new class_type_name();
```

ex:- new Book();
new Book();



→ class file only once loaded into the RAM.

Refrence Variable :-

→ A variable of type class is called reference variable.

→ Refrence variable is used to store an address/ hashcode of object.

→ A refrence variable is repeated in order to reach an object by program.

Syntax :-

```
Class_type_name variable_name;
```

ex:- Book book1, book2; → Refrence variables. not object, but it holds the reference of book object.

book1 = new Book();
book2 = new Book();
object

reachable object | unreachable object.

→ Reference variable always refer a non-static class.

② what is reachable object or referenced object?

→ An object binds with reference variable is called reachable object.

② what is unreachable object or unreferenced object?

→ An object which doesnot bind with any reference is called unreachable objects.

→ All unreachable objects automatically removed by JVM.

example:-

```
class A
{
    int x;          → non-static
    psvm(String args[])
    {
        S.O.P(x);
    }
}
```

* → The above program display compilatime error, because non-static members of class can't access by static members.

→ In order to access we need to create object.

Modifiers declared/used by the variables declared within the class:-

① private
② public
③ protected
④ default

access specifiers

⑤ static
⑥ final
⑦ transient
⑧ Volatile

→ access modifier.

Creating class & object :—

Date-22/9/12

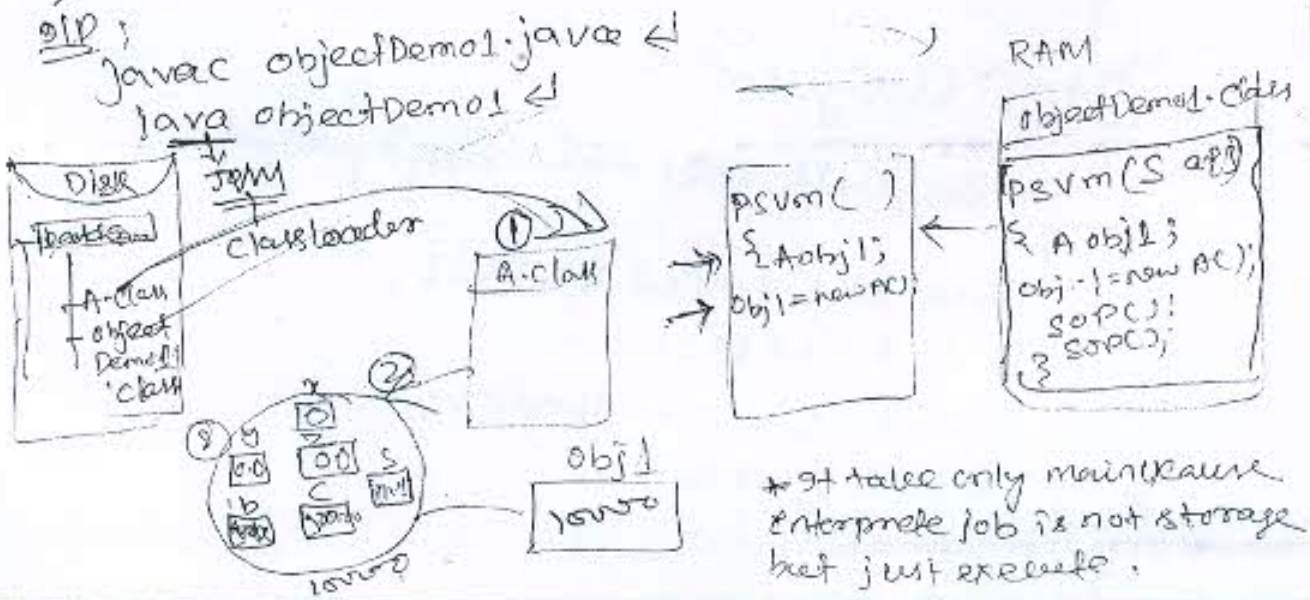
```
Class A
{ int x;
  float y;
  double z;
  boolean b;
  char c;
  String s;
}
```

Class
- executable class → Contain main() method.
- Reusable class / non-executable class → doesn't contain main().

```
class ObjectDemo1
{ public static void main (String args[])
  { A obj1;
    obj1 = new A();
    System.out.println(obj1.x);
    System.out.println(obj1.y);
    System.out.println(obj1.z);
    System.out.println(obj1.b);
    System.out.println(obj1.c);
    System.out.println(obj1.s);
  }
}
```

DIP;

```
javac objectDemo1.java ↵
java objectDemo1 ↵
```



* JVM takes only main() because interpreter job is not storage but just execute.

Access specifiers:

Package

```
public class A
{
  int x;
  private int y;
  protected int z;
  public int P;
}
```

ObjectDemo2.java

```
class ObjectDemo2
{
PSVM(String args[])
{
  A ob1;
ob1 = new A();
S.O.P(ob1.x); → X
S.O.P(ob1.y); → X
S.O.P(ob1.z); → X
S.O.P(ob.P); → ✓
```

→ Private members can't access outside the class & package.

→ Default members cannot access outside the package. but it can accessible outside the class.

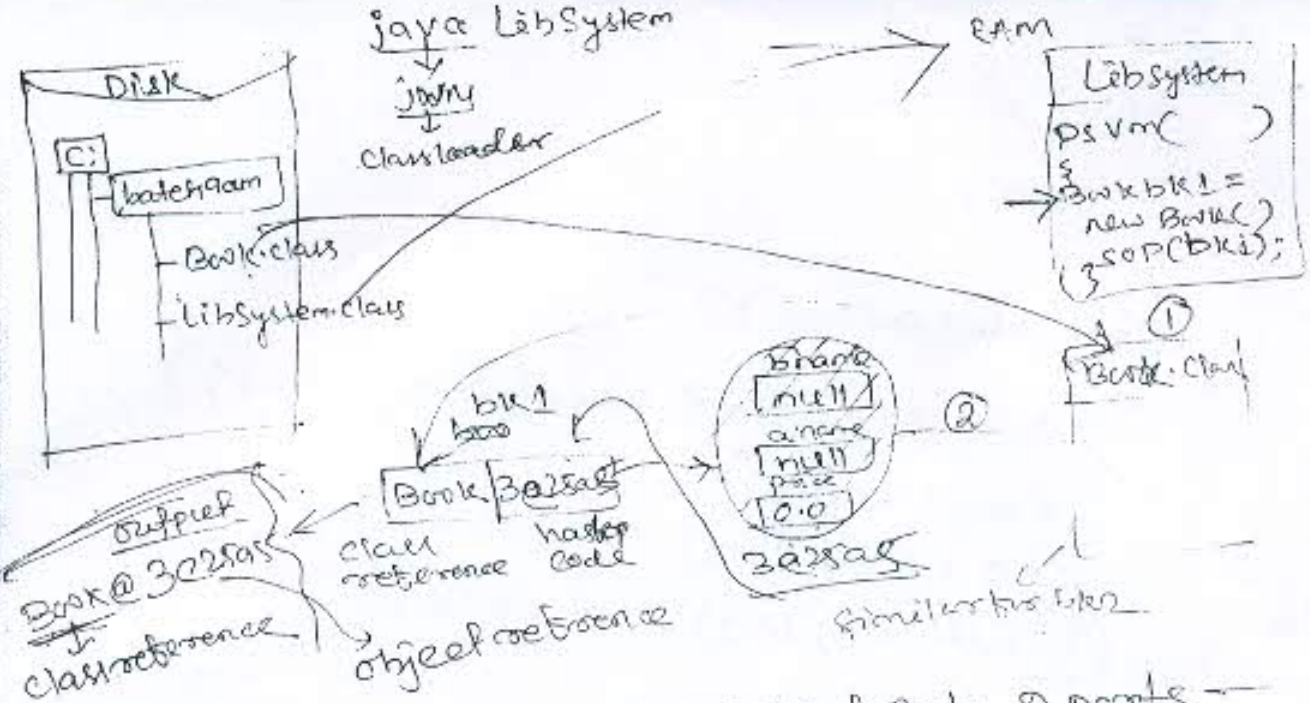
→ Public members are accessible in anywhere outside the class & package.

→ Protected members also not access by outside the package.

example:

```
class Book
{
  private String bname;
  private String aname;
  private float price;
}
class LibSystem
{
  public static void main(String args[])
  {
    Book bk1 = new Book();
    S.O.P(bk1);
    Book bk2 = new Book();
    S.O.P(bk2);
  }
}
```

* Data hiding can be achieved by using the variable as private scope.



→ Refrence variable is divided into 2 parts —

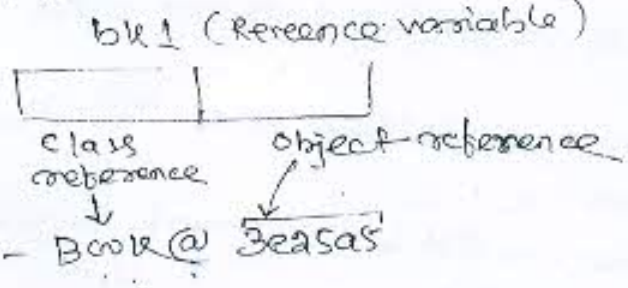
1.) object reference

2.) class reference

→ Object created with runtime is referred with hashcode.

→ A class loaded within RAM during runtime is refered with class name.

ex:-



* → Data hiding in object oriented is achieved by declaring variables within class as private.

→ Private members cannot access outside the class.

→ { int a;
a = null; // invalid as null is reference value.

→ class is loaded with class name & object is loaded with hashcode (hexadecimal no.)

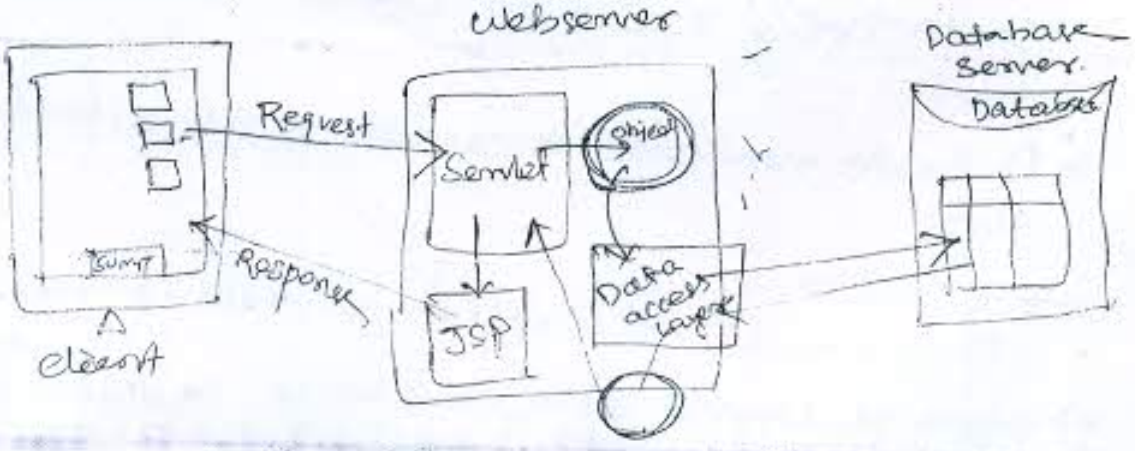
```
class A
{
  int x;
}
class ObjectDemo2
{
  public static void main(String args[])
  {
    A obj1 = null;
    S.O.P(obj1.x);
    //S.O.P(obj1.y); → error.
  }
}
```

obj1
[A | null]

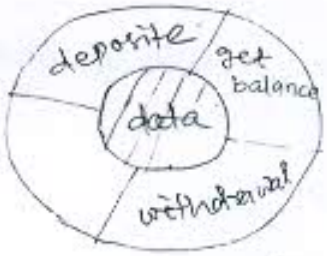
* → The above program display an error during runtime because non-static members cannot access without creating objects.

Date - 22/9/12

→ Instance variables does not bind with class name.
→ Instance variables bind with object name or reference.
→ Instance variables cannot access without creating objects.
→ Instance variables define state of an object.



methods :-



→ A method define behaveour of an object.
→ In OOP any operations performed on object is represented as method.
→ An object communicate with another object by using method.

```
class Sum
{ int x,y,z;
  void assign()
  {
    x=10;
    y=20;
  }
  void add()
  {
    z=x+y;
  }
  void Result()
  {
    S.O.P(z);
  }
}
```

executable stm
error

→ Any executable statement must be present write inside the method.
→ methods are of 2 types —
1) Instance method.
2) class method.
→ A small piece of program inside the class is called method.
→ A method is a function which bind with either class or object.

Instance method :-

→ Non-static method of class is called instance method.
→ This method is bind with object name.
→ This method cannot called without creating object.

Syntax:-

```
[modifier] return_type method-name(parameters)
{
    statements;
}
```

→ Java does not allows to write method without return type.
→ If method does not return any value it should be define with void.
→ Instance method performs object operations.

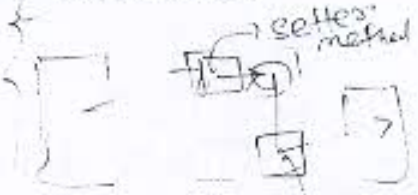
ex:-

```
class Account
{
  private int accno;
  private String name;
  private float balance;
  void setAccount()
  {
      accno = 101;
      name = "Preetam";
      balance = 5000f;
  }
  void printAccount()
  {
      S.O.P(accno);
      S.O.P(name);
      S.O.P(balance);
  }
}
class Bank
{
  public static void main(String args[])
  {
      Account acc1, acc2;
```

Class is loaded with class name & Account is a object bcoz we can create class on the view of object.

```
acc1 = new Account();
acc2 = new Account();
acc1.setAccount();
acc2.setAccount();
acc1.printAccount();
acc2.printAccount();
}
}
```

* immutable objects
→ If we don't access the value i.e private members.



method not called by object

to get (on) object.

→ A method within class performs 2 types of operations.

1) Setter operation
2) Getter operation.

Q) What is Setter method?

→ An operation which change the state (values) of object is called setter method / modifier method

Q) What is Getter method?

An operation which doesn't change the state of object is called getter method. This performs accessing operations.

Date 24/9/12

Method with parameters :-

→ method having parameter receives values and perform operation on object.

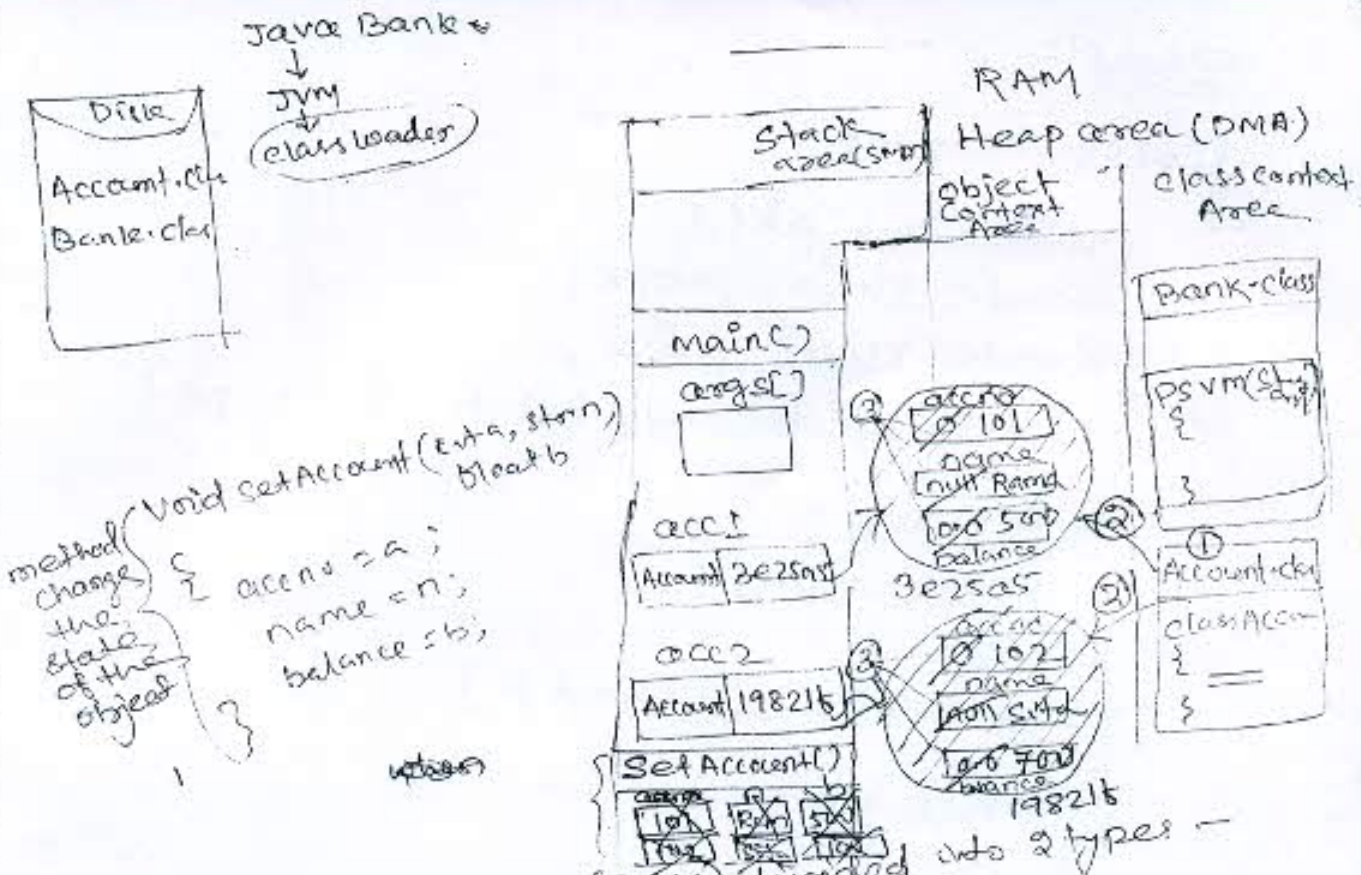
→ One method communicate with another method by passing values.

→ Java supports two ways of calling method.

1) Pass by value.
2) Pass by reference.

→ method parameters are called local variables.
→ These variables exist until execution of method.

```
class Account
{
    private int accno;
    private string name;
    private float balance;
    void setAccount(int a, String n, float b)
    {
        accno = a;
        name = n;
        balance = b;
    }
    void printAccount()
    {
        S.O.P(accno);
        S.O.P(name);
        S.O.P(balance);
    }
}
class Bank
{
    public static void main(String args[])
    {
        Account acc1 = new Account();
        Account acc2 = new Account();
        S.O.P(acc1);
        S.O.P(acc2);
        acc1.setAccount(101, "Rama", 5000f);
        acc2.setAccount(102, "Sita", 6000f);
        acc1.printAccount();
        acc2.printAccount();
    }
}
```



→ The primary memory (RAM) divided into 2 types —

① Stack area. - object context area (object stored)

② Heap area — class context area (class is stored)
(or) method area.
(or) static area.

→ When stack area is full it send a msg that "Stack overflow."

→ methods are stored in stack area.

→ JVM calls the method by sending hashcode of object on which it perform operation or currently bind.

Q) what is binding :—

Linking method call with method body is called binding.

example:-

```
class Product
{
    private int pid;
    private string pname;
    private float price;
    void SetpidName(int p, String pn)
    {
        pid = p;
        pname = pn;
    }
    void setPrice(float P)
    {
        price = P;
    }
    void print()
    {
        S.O.Pf("\n%d %s %f", pid, pname,
                price);
    }
}
class Shopping
{
    public static void main(String args[])
    {
        Product P1 = new Product();
        Product P2 = new Product();
        P1.setPidName(101, "Mouse");
        P1.SetPrice(200f);
        P1.print();
        P2.SetPidName(102, "keyboard");
        P2.setprice(300f);
        P2.print();
    }
}
```

Q) What is immutable object?

- An object whose state (values) cannot be changed after creating with initial values is called immutable object
- This class doesn't provide setter methods.

Q What is mutable object?

- An object whose state (values) can be changed after creating with initial values is called mutable object.

Date - 25/9/12

ex-1 -

```
class A
{
    private int x;
    private int y;
}
```

```
class B
{ psvm( )
  A obj1 = new A();
  obj1.x = 10; → ⊗
  obj1.y = 20 → ⊗.
}
```

→ The above is an example of immutable ~~native~~ object bcoz the initial values cannot change.

⇒ No setter method in immutable object.

ex-2 :-

```
class A
{
    private int x;
    private int y;
    void setXY()
    {
        x = 10;
        y = 20;
    }
}
```

```
Class B
{ psvm(String args[])
  { A ob = new A();
    ob.x = 10; → X
    ob.y = 20; → X
    ob.setXY();
  }
}
```

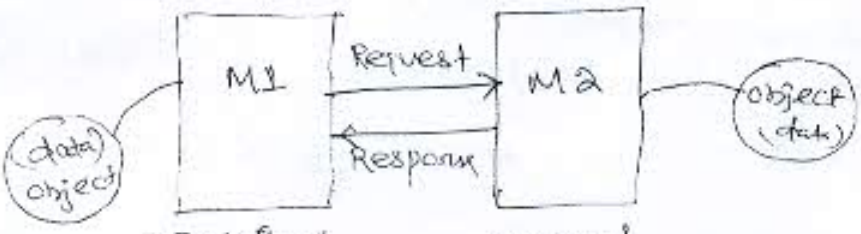
→ The above is an example of mutable object

→ Setter method must be present

Method with returntype :-

→ A method having returntype as void can't return any value.

→ A method perform operation on object and return values.

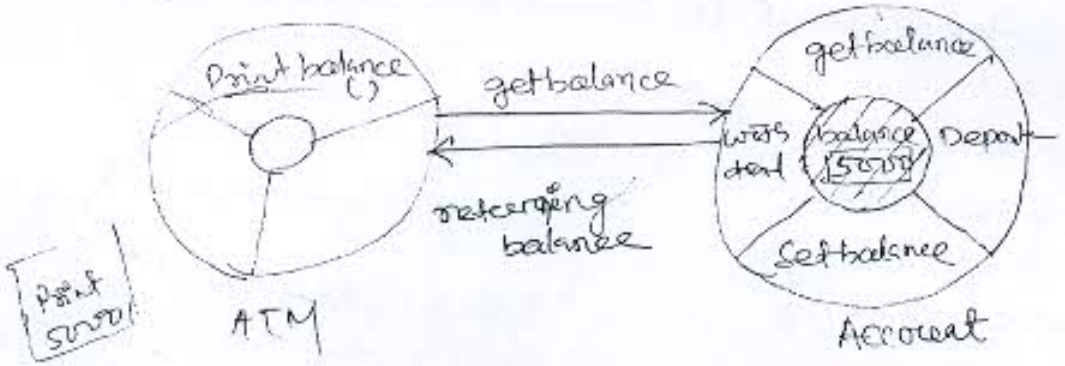


- Calling method
- This method have return type.

- Called method
- always having parameters.

→ A method can have return type as —

1) Primitive type → it returns value.
2) Reference type → it returns object.



ex :-

```
class Account{
{
    private int accno;
    private float balance;
    void setAccount(int a, float b)
    {
        accno = a;
        balance = b;
    }
```

```
    float getBalance()
    {
      return balance;
    }
}
class ATM
{ public static void main(String args[])
  {
    Account acc1 = new Account();
    acc1.setAccount(101, 5050f);
    float bal = acc1.getBalance();
    System.out.println("Balance" +bal);
  }
}
```

javaATM ↵

→ Every method should have a method stackarea, so that it must create their local variables.

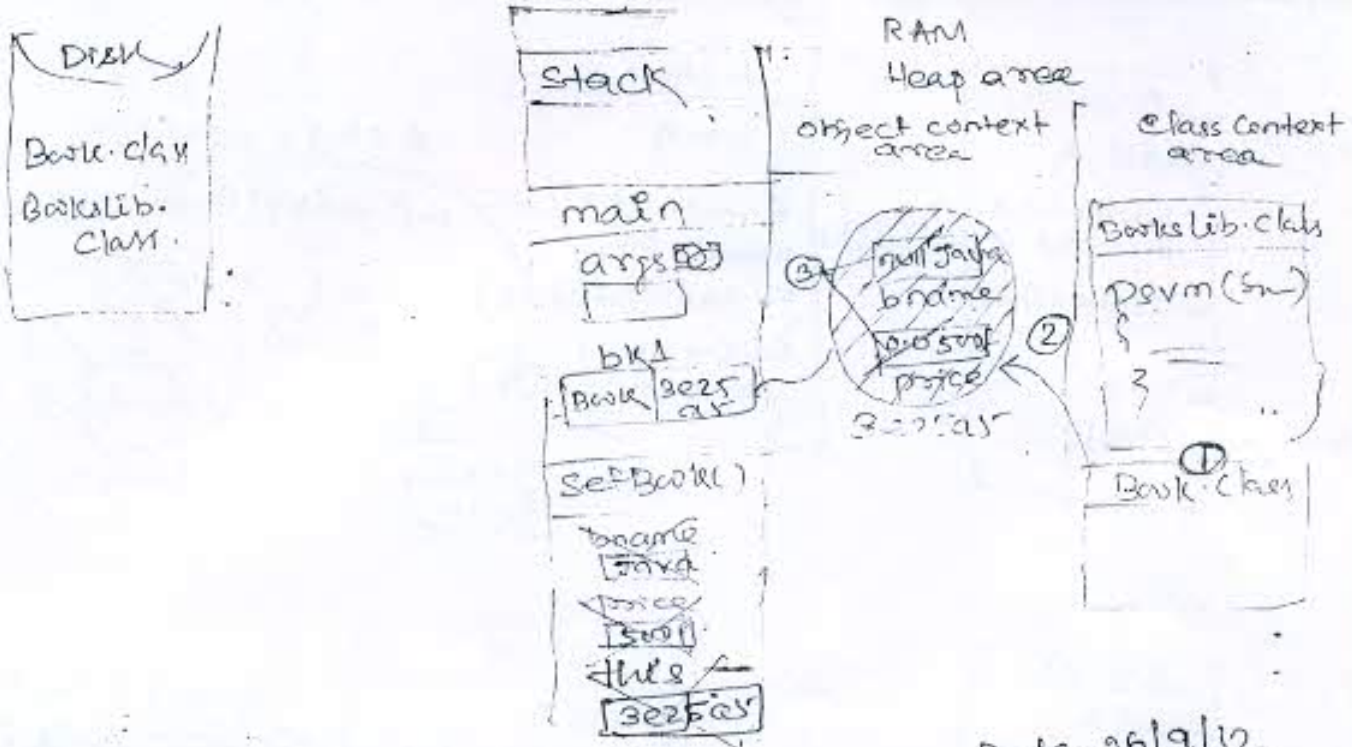
example:

```
class Book
{
  private String bname;
  private float price;
  void setBook(String bname, float price)
  {
    this.bname = bname;
    this.price = price;
  }
  String getBook()
  {
    return bname+" "+price;
  }
}
class BooksLib
{
  public static void main(String args[])
  {
    Book bk1 = new Book();
    bk1.setBook("java", 500f);
    String b = bk1.getBook();
    System.out.println(b);
  }
}
```

Note:-

Statements

- executable
  - it always resides inside method.
- declaring



Date - 26/9/12

this reference :-

→ Every non static method of class, java compiler add a refrence variable "this".

→ "this" is a keyword.

→ It is a refrence variable which stores/holds, the hashcode of current object, on which method perform opr^n.

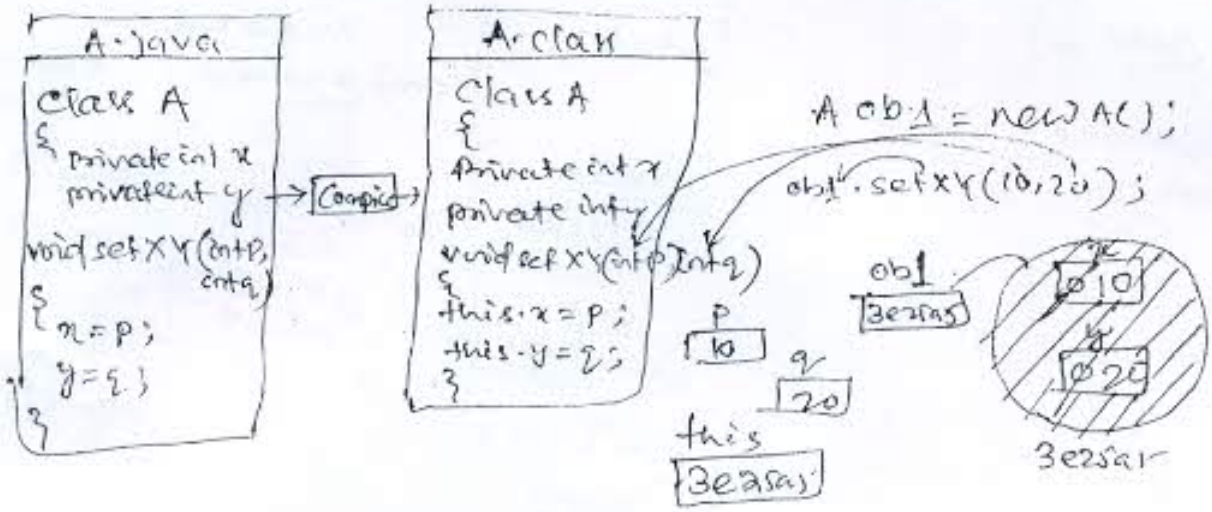
→ "this" is a constant variable whose value is never changed.

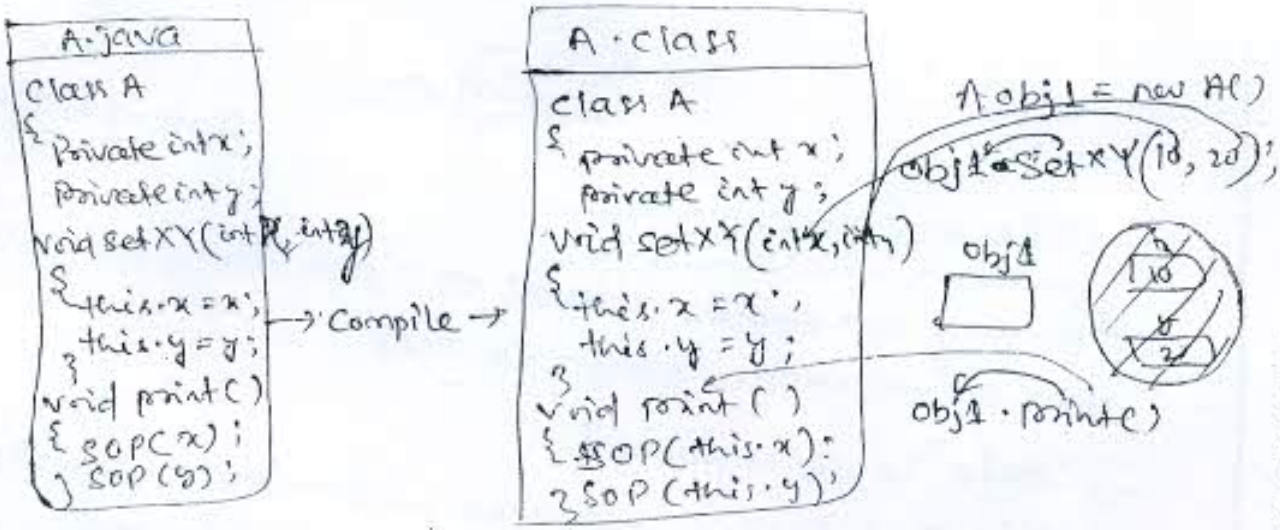
→ If instance & local variables are declared with same, instance variable of current object are refer using "this" reference. (It must explicitly done)

→ "this" is a local variable.

→ "this" refrence can be explicitly call. i.e if should be call by compiler implicitly

→ Instance method always called by sending the hashcode of object i.e stored in "this" pointer.





Program :—

```
Class Accocent
{ private int accno ;
  private float balance ;
   void setAccocent (int accno, float balance)
   {
     this.accno = accno ;
     this.balance = balance;
   }
   void deposite (float tamt)
   {
     balance = balance + tamt ;
   }
   void withdraw (float tamt)
   {
```

```
        if(tamt > balance)
        S.o.p("Insufficient balance");
        else
        balance = balance - tamt;
    }
    String getAccount()
    {
        return accno + "   " + balance;
    }
}
class Bank
{
    public static void main(String args[])
    {
        Account acc1 = new Account();
        acc1.setAccount(101, 5000f);
        String a;
        a = acc1.getAccount();
        S.o.p(a);
        acc1.deposite(5000f);
        a = acc1.getAccount();
        S.o.p(a);
        acc1.withdraw(2000f);
        a = acc1.getAccount();
        S.o.p(a);
    }
}
```

Method overloading :-

→ Define more than one method with same name by changing :

1. no. of parameter.
2. types of parameters.
3. order of parameters.

is called ~~overide~~ method overloading.

~~→ When more than~~

Q) When to overload method ?

When more than one method having similar operations but different implementation, method is overloaded.

Q) Why should I overload method ?

1) Simplecity :-

As a developer / programmer does not required to remember no. of method name ;

2) Extensibility :-

Adding new features in to existing methods without modifying it.

3) Reusability :-

Allows to reuse method name & functionality.

→ Overloaded method having Polymorphic behaviour.

→ ~~Defining~~ one thing in many forms is called "polymorphism".

→ This is called name polymorphism or ad-hoc polymorphism.

```
class A
{  void m1() { }
   void m2() { }
}
```

error:-
becoz the compiler cannot write two methods with same signatureee.

Date-27/9/12

→ Compiler recogrize method with its signatureee, which include no. of parameters, types of parameters and order of parameteres.

```
class A
{  int  m1() { }      → same signatureee
   float m1() { }
}
```

O/P Error. (Invalid overloading)

~~compiler~~

```
class A
{  void m1() { }      signatureee = method name + parameters
   int  m1(int x) { }
}
```

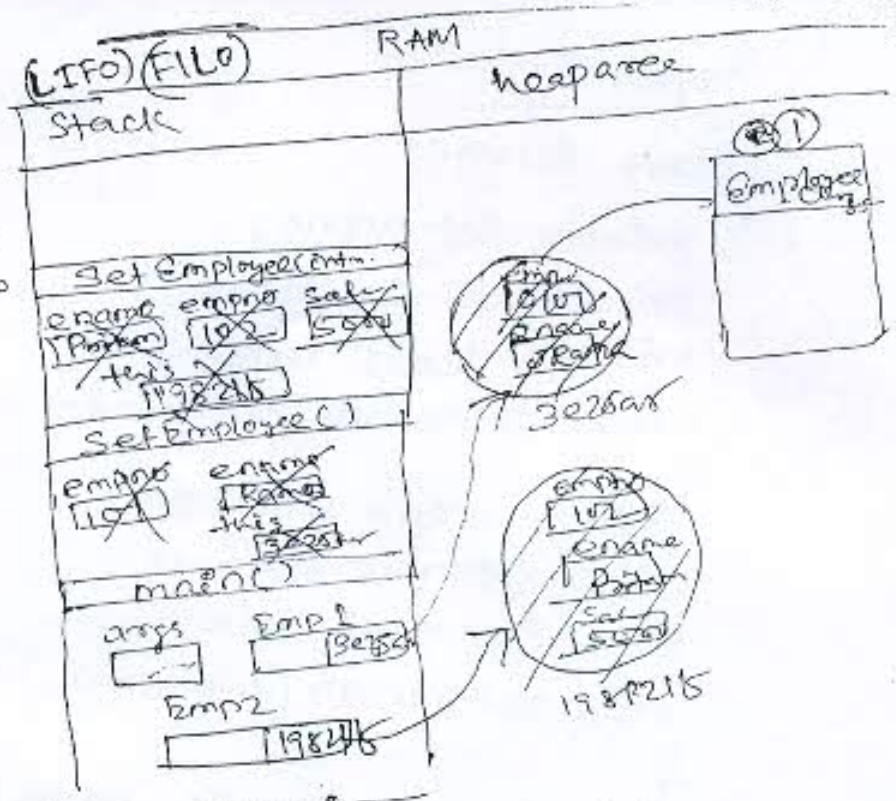
O/P :- valid overloading.

Program :-

```
class Employee
{  private int empno;
   private String ename;
   private float Salary;
   void setEmployee(int empno, String ename)
   {  this.empno = empno;
      this.ename = ename;
   }
```

```
void setEmployee(int empno, String ename,
                 float salary)
{
    this.ename = ename;
    this.empno = empno;
    this.salary = salary;
}
String getEmployee()
{
    return empno + "  " + ename + "  " +
                                salary;
}
}
class Payroll
{ public static void main (String args[])
  {
    Employee Emp1 = new Employee();
    Employee Emp2 = new Employee();
    emp1.setEmployee(101, "Rama");
    emp2.setEmployee(102, "Krishna", 50000);
    String e1 = emp1.getEmployee();
    String e2 = emp2.getEmployee();
    System.out.println(e1);
    System.out.println(e2);
  }
}
```

→ The behaviour (selection of method) is done during compile time. So this is called compile time polymorphism.



O/P

101 Rama 0.0
102 Pritam 50,000

Compile time polymorphism:-

→ method overloading exibit compile time polymorphism.

→ As selection of method is done during compile time based on no. of parameters, types of parameters or order of parameters.

→ The method which is selected by compiler same method is executed by JVM during runtime.

→ As behaviour of this method is changed at compile time, it is called compile time polymorphism.

Program:-

```
Class Account
{
  private int accno;
  private String name;
  private float balance;
  void setAccount(int accno, String name)
  {
    this.accno = accno;
    this.name = name;
  }   // without balance
  void setAccount(int accno, String name, float balance)
  {
    setAccount(accno, name);   // to avoid the code redundancy we call a method inside another method.
    this.balance = balance;
  }  // with balance
  String getAccount()
  {
    return accno+" "+name+" "+balance;
  }
}
Class Bank
{
  public static void main(String args[])
  {
    Account acc1 = new Account();
    Account acc2 = new Account();
    acc1.setAccount(101, "Rama");
    acc2.setAccount(102, "Sita", 5000f);
    String a1 = acc1.getAccount();
    String a2 = acc2.getAccount();
    S.O.P(a1);
    S.O.P(a2);
  }
}
```

Note :—

① class A
{ void print (double d)
{ S.O.P ("Inside double");
}
void print (float f)
{ S.O.P ("Inside float");
}
}
class ObjectDemo9
{ public static void main (String args[])
{
A obj1 = new A();
obj1.print(10);
}
}

O/P :-
javac objectDemo9 ↵
java objectDemo9
Inside float method.

→ Compiler always look for a method having similar type parameteres.
→ If it doesnot found similar type parameter it look for method having border datatype parameters.
→ If method with border datatype parameters not available it leads to compile time error.

Ex :-
② class A
{ void print (byte b)
{ SOP ("Inside byte");
}
void print (int x)
{ S.OP ("Inside int");
}

```
Class ObjectDemo10
{
  public static void main(String args[])
  {
    A obj1 = new A();
    obj1.print(65);
  }
  {byte a = 65;
   obj1.print(a);
}
```

O/P Inside Integer method.
Inside Byte

→ If the type is not specified, then it is by default integer.

(3)
```
class A
{ void print(byte b)
  { sop("Inside byte");
  }
  void print(char ch)
  { sop("Inside character");
  }
}
class ObjectDemo11
{ psvm(String args[])
  { A obj1 = new A();
    obj1.print(65);

  }
}
```

O/P :- Compiler error.

→ This above program display compiler time error because there is no method of type integer.

Date - 25/9/12

Order of parameters :-

```
class A
{ void M1(int x, float y)
  {
  }
  void M1(float x, int y)
  {
  }
}
```

Static :-

→ Static is a modifier used for declaring,
- variables
- methods
- blocks

Static variables :-

→ Static variables are class variables for which memory is allocated on loading of class.

→ Java does not support global variables.

→ In order to declare global variable, which is global to more than one object, it is declared with static modifier.

Syntax :-

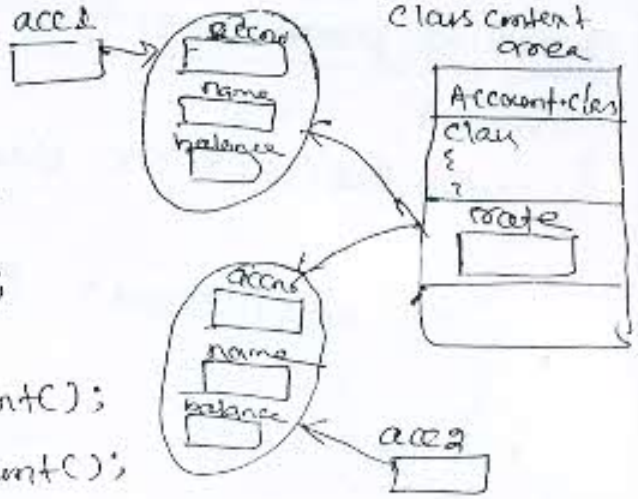
```
static data-type variable-name;
```

→ The static variable is loaded in class context area.

→ Static variables are shared by more than one object.

→ Static variables bind with class name or object name.

```
Class Accocent
{  int accno;
   String name;
   float balance;
   static float rate;
}
Account acc1 = new Account();
Account acc2 = new Account();
```



→ In order to declare a variable which is common for more than one object declared as static.

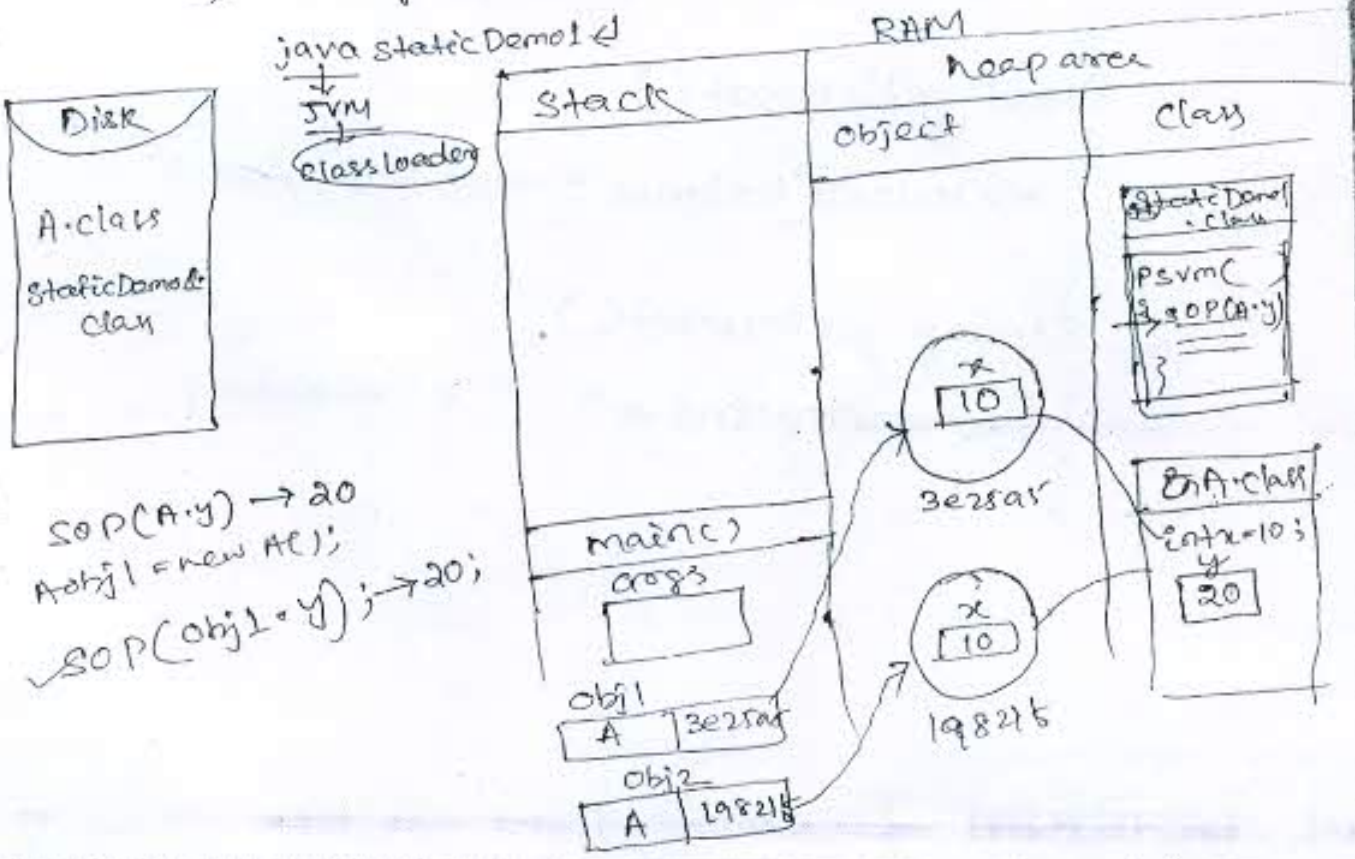
Q. Difference bet' static & non-static variables?

Non-static variables	static variables
① Non-static variables of a class are called instance variable.	① static variables of a class are called class variables.
② Memory for this variable allocated on creation of object.	② Memory for this variable allocated on loading of class.
③ These variable belongs to object. So every object having its own copy of instance variables.	③ These variables belongs to class, more than one object share static variable. becoz it is a Global variable.
④ It is bind with object name.	④ It is bind with class name or object name.
⑤ Memory for this allocated in object context area.	⑤ The memory for this allocated in class context area.

ex:—

```
class A
{ int x = 10;
  static int y = 20;
}
class staticDemo1
{ public static void main (String args[])
  {
    System.out.println (A.y); // valid
    System.out.println (A.x); // invalid non-static variable cannot access by class name.
    A obj1 = new A();
    A obj2 = new A();
    S.O.P (obj1.x);
    S.O.P (obj2.x);
  } S.O.P (obj1.y);
} S.O.P (obj2.y);
```

Note:
→ class can be loaded into RAM using 3 types —
by using -
1) new operator
2) class.forName ("class name");
3) Accessing static member of a class.



Note:-

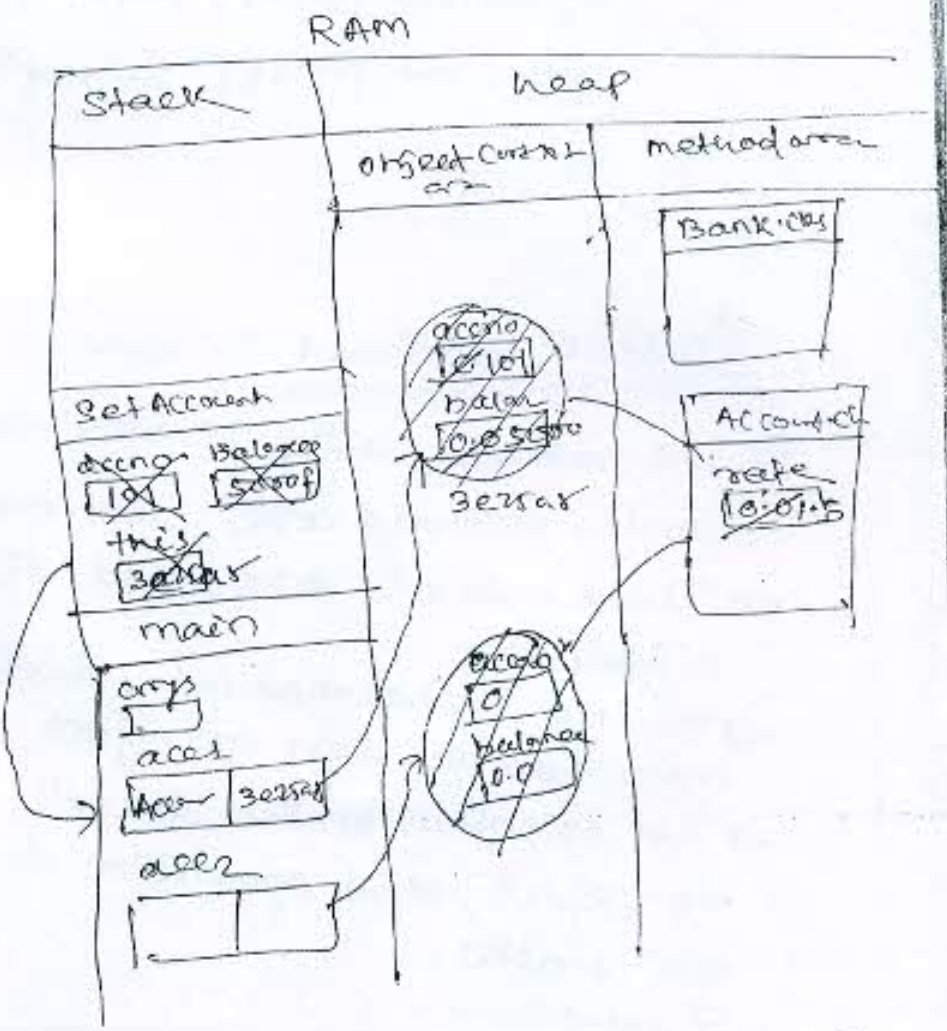
```
class A
{
    void M1()
    {
        static int x;
    }
}
```

→ This is an error.
→ The above program display compiletime error becoz static modifier is not allowed for local variables.
→ C, supports but java doesn't support it.

Program:-

```
class Account
{
    private int accno;
    private float balance;
    static float rate;
    void setAccount(int accno, float balance)
    {
        this.accno = accno;
        this.balance = balance;
    }
    float getInterest()
    {
        return(balance * rate * 6)/100;
    }
    String getAccount()
    {
        return accno + "  " + balance;
    }
}
```

```
Class Bank
{
  public static void main (String args[])
  {
    Account acc1 = new Account();
    Account acc2 = new Account();
    acc1.SetAccount(101, 5000f);
    acc2.setAccount(102, 6000f);
    Account.rate = 1.5f;
    float i1 = acc1.getInterest();
    float i2 = acc2.getInterest();
     S.O.P(i1);
     S.O.P(i2);
    String s1 = acc1.getAccount();
    String s2 = acc2.getAccount();
    System.out.printf("%s %s", s1, s2);
  }
}
```



```
Class A
{  int x = 10;
   Static int y = 20;
}
class B
{  public static void main (String args[])
   {  A obj1 = new A();
      A obj2 = new A();
      S.o.println("%d %d", obj1.x, obj1.y);
                          10      20
      S.o.println("%d %d", obj2.x, obj2.y);
                          10      20
      obj1.y = 40;
      obj2.y = 90;
      System.out.println("%d %d", obj1.x, obj1.y);
                                  10      40
      System.out.println("%d %d", obj2.x, obj2.y);
                                  10      90
   }
}
```

Date-29/9/12

Static methods :-

→ The methods define within class can be static.
→ Static methods are class methods.
→ These methods are bind with class name or objectname.
→ It is a Global method, which is Global to more than one class or object.
→ An operation which doesn't involb on creation of object, that operation or method declare as static.

Syntax :-

```
static return-type method_name (parameters)
{
   statements;
}
```

Q) what is the diff. betn static method & non-static methods?

Non-static	Static
(1) It is called instance method.	(1) It is called class method.
(2) It is bind with object name.	(2) It is bind with object name or class name.
(3) To call this method required to create object.	(3) This method is called without creating object.
(4) It is with "this" reference.	(4) It is without "this" reference.
(5) It access both static & non-static members of class.	(5) It can access only static members of a class; not access not static members.
(6) It allows Run-time polymorphism.	(6) It doesn't allows Run-time polymorphism.
(7) It defines object opr.	(7) It will defines class operations.

example:-

```
class A
{   int x = 10;
    static void print()
    { S.O.P (x);
    }
}
```

→ The above program displays an error at compile time because static method cannot access non static variable.

Ex-2 :-

```
class StaticDemo2
{
    int x = 10;
    public static void main(String args[])
    {
        S.O.P(x);
    }
}
```

→ This above program display compile time error because main is static method which can't access 'x'.

→ For invoking non-static we must create objects.

Ex-3 :-

```
class Math
{
    static int sqr(int n)
    {
        return n*n;
    }
    static int max(int x, int y)
    {
        if(x>y)
            return x;
        else
            return y;
    }
}
class StaticDemo3
{
    public static void main(String args[])
    {
        int s = Math.sqr(5);
        int m = Math.max(20,10);
        S.O.P("Sqr is " + s);
        S.O.P("Max is " + m);
    }
}
```

ex-4:

```
class A
{ void m1()
  { S.O.P("Inside non static method");
  }
  static void m2()
  { S.O.P("Inside static method");
  }
}
class static Demo 4
{ public static void main(String args[])
  { A.m1(); // error →
    A.m2();
  }
}
```

A.m1() ∴ non-static member can't access in static context error.

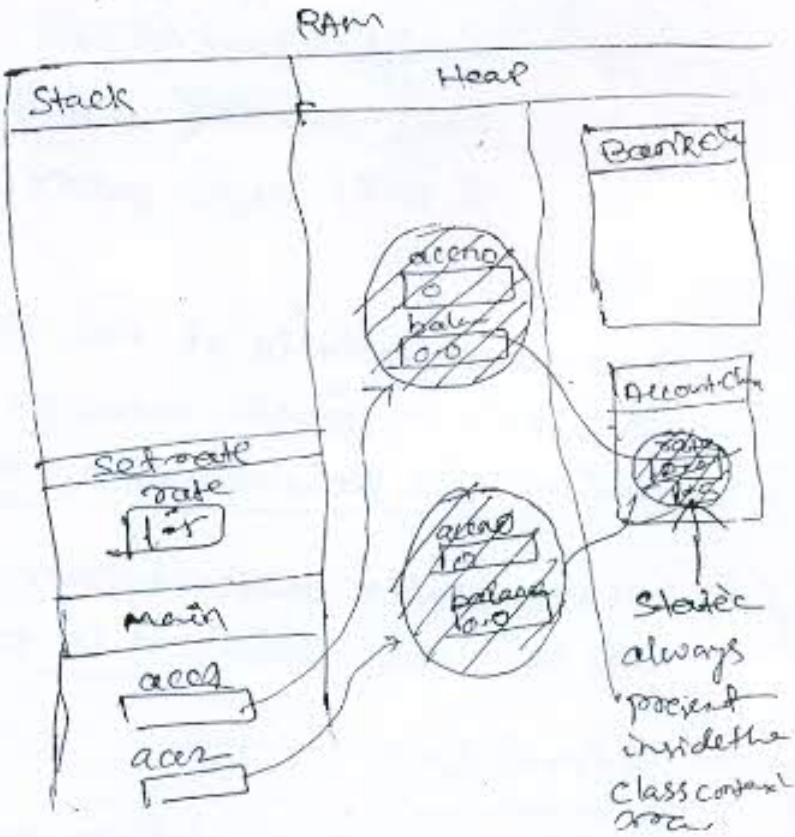
ex-5:

```
class A
{ void m1()
  {
  }
  static void m2()
  {
  }
}
class Static Demo 4
{ psvm(String args[])
  { A obj1 = new A();
    obj1.m1();
    obj2.m2();
  }
}
```

ex-6 :-

```
class Account
{
    private int accno;
    private float balance;
    private static float rate;
    static void setRate(float rate)
    {
        Account.rate = rate;
    }
    void setAccount(int accno, float balance)
    {
        this.accno = accno;
        this.balance = balance;
    }
    void deposite(float tamt)
    {
        balance = balance + tamt;
    }
    static float getRate()
    {
        return rate;
    }
    String getAccount()
    {
        return accno + " " + balance + " " + rate;
    }
}
class Bank
{
    public static void main(String args[])
    {
        Account.setRate(1.5f);
        S.O.P(Account.getRate());
        Account acc1 = new Account();
        acc1.setAccount(101, 5000f);
        S.O.P(Account.getAccount());
```

```
        acc1.deposite(2000f);
        S.O.P(acc1.getAccocont());
    }
}
```




```
@lass A
{   int x;
    Statec int y;
    void setXY(int x, inty)
    {  this.x = x;
       this.y = y;  (OR)  A.y = y;
    }
    String getXY()
    {  return x + " " + y;
    }
}
```

```
Class B
{
  psvm (String args[])
  {
    A obj1 = new A();
    obj1.setXY(10,20);
    S.O.P( obj1.getXY());
  }
}
```

→ Local variable of non-static method can be access static member of a class by using this reference or class name.

→ A non-static method can access static members of a class using this reference.

Final :

→ Final is a modifier used for declaring

- Final variable
- final parameters
- Final methods
- final class.

Final variable :

→ The value of final variables never changed.
→ Final variables are constants.
→ These variables can be,

1. Final instance variable
2. Final class variable
3. Final local variable.

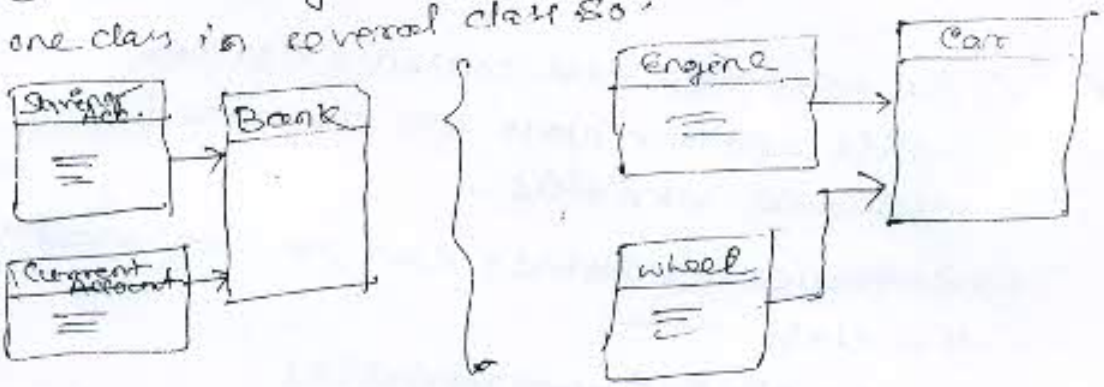
Date - 2/10/12

Class Reusability :

→ Object oriented allows to use content of one class inside another class using 2 approaches or methods —

1. Composition (bet^n unrelated classes)
2. Inheritance (bet^n related classes)

→ If we write all the data in one class then it leads to ① redundancy & ② inconsistency. So we divide the one class in several classes.



Composition :

→ It is a process of creating an object of one class inside another class.

→ It allows to use contents of one class inside another class but doesn't allow to extend. (i.e it allows only reusability but not extending)

→ The relationship between classes in composition is called has-a relation.

→ This relationship is between unrelated classes where one class doesn't share properties & behaviour of another class.

```
class A  → container class.
{
   int x ;
   B obj = new B();
}
```

has-a

```
class B  → contained class
{
}
```

obj → Reference variable

```
class System
{
static PrintStream out = new PrintStream();
}
System.out.println()
      "    print()
      "    print()
```

```
class PrintStream
{ void println()
  {
  }
  void print()
  {
  }
  void printf()
  {
  }
```

→ In order to use contents of one class inside another class we requirred a reference variable.

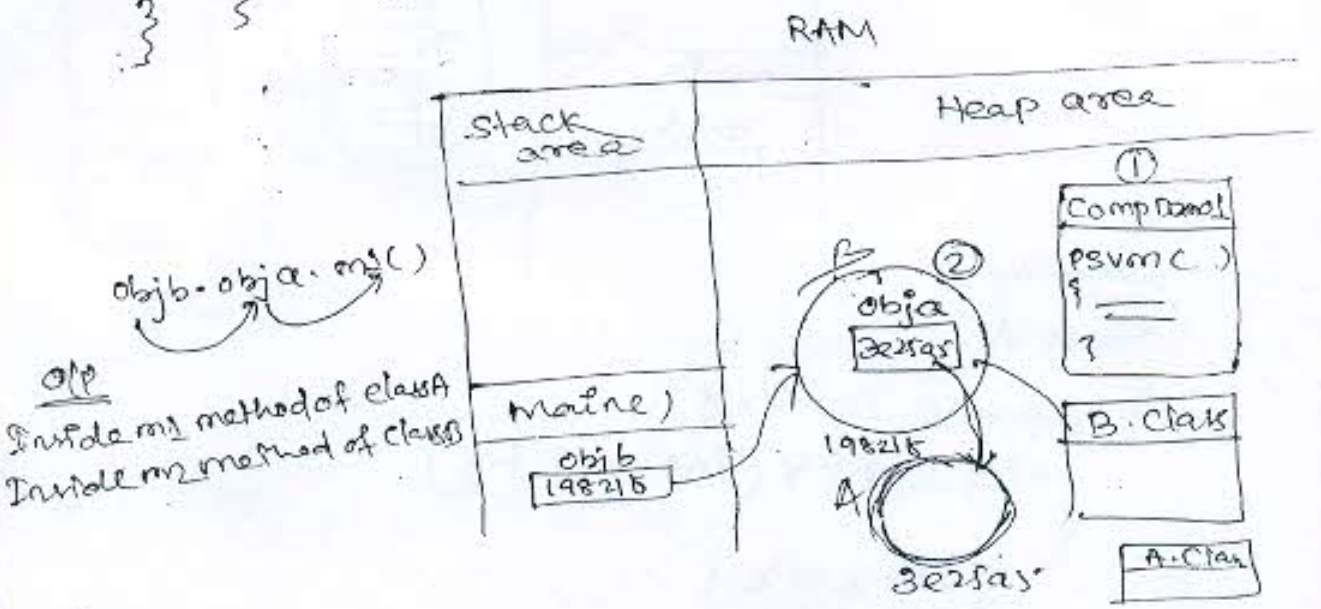
→ Refrence variables can be declareed within the class —

1. Steatic Refrence variables
2. non steatic reference variable
3. Local reference variables.

example:-

```
class A
{
  void m1()
  {
    System.out.println("Inside m1() of class A");
  }
}
class B
{
  A obj = new A();
  void m2()
  {
    S.o.p("Inside m2() of class B");
  }
}
```

```
class CompDemo1
{
    public static void main (String args[])
    {
        B objb = new B();
        objb.obja.m1();
        objb.m2();
    }
}
```



O/P
Inside m1 method of classA
Inside m2 method of classB

→ The reference variable may be private, public default etc.

→ If we want to hide the object within another object. then we declare inner object as private.

Example:

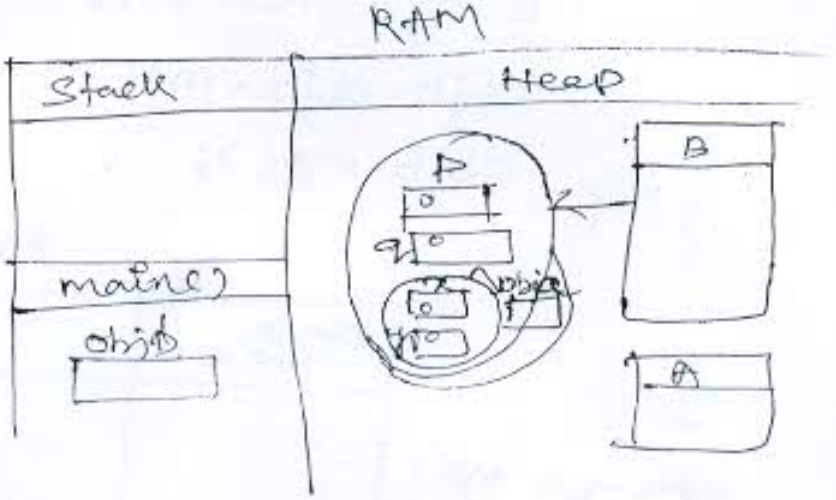
```
class A
{
    int x;
    int y;
}
class B
{
    int p;
    int q;
    A obja = new A();
}
```

```
class CompDemo2
{
    psvm(String args[])
    {
        B objb = new B();
        S.O.P(objb.p); → 0
        S.O.P(objb.q); → 0
        S.O.P(objb.obja.x); → 0
        S.O.P(objb.obja.y); → 0
    }
}
```

```
S.O.printf("\n y.d y.d", objb.p, objb.q);
S.O.println("\n y.d y.d", objb.obja.x,
            objb.obja.y);
```

→ When object is created it allocate memorey for non-static varciables.

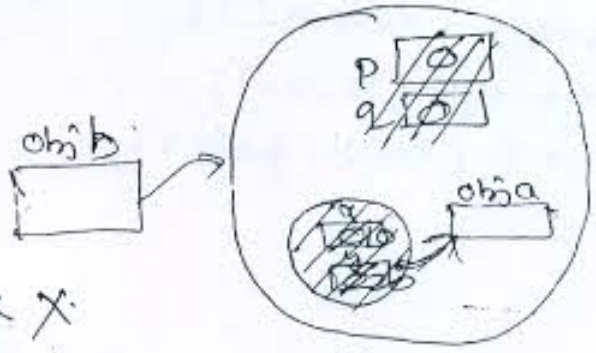
→ In 'B' class 3 variables are there.



Example-3

```
class A
{ private int x, y;
  void setXY (int x, int y)
  {
    this.x = x;
    this.y = y;
  }
  int getX()
  {
    return x;
  }
  int getY()
  {
    return y;
  }
}
class B
{
  private int p, q;
  {
    A obja = new A();
    void setPQ (int p, int q)
    {
      this.p = p;
      this.q = q;
    }
```

```
    int getP()
    { return p;
    }
    int getQ()
    { return q;
    }
}
class CompDemo3
{ public static void main (String args[])
  {
    B objb = new B();
    objb.setPQ(10,20);
    objb.objA.setXY(30,40);
    S.O.println("\n%d %d", objb.getP(), objb.getQ());
    S.O.println("\n%d %d", objb.objA.getX(),
                          objb.objA.getY());
  }
}
```



objb.objA.x X
objb.p X
objb.objA.setXY(10,20);

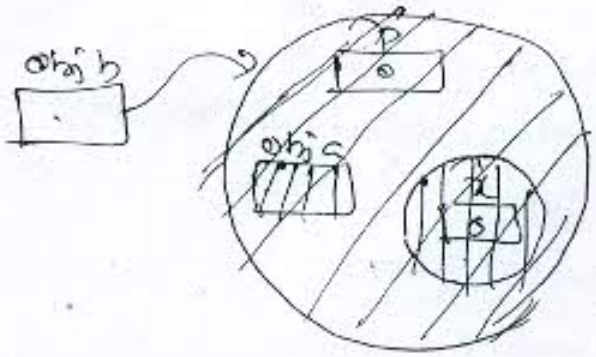
example-4 :—

```
class A
{ private int x;
  void setX(int x)
  { this.x = x;
  }
  int getX()
  { return x;
  }
}
```

```
class B
{
  private A obja = new A();
  private int p;
  void set(int x, int p)
  {
    obja.setX(x);
    this.p = p;
  }
  String get()
  {
    return obja.getX() + "  " + p;
  }
}
class CompDemo4
{
  public static void main(String args[])
  {
    B objb = new B();
    objb.set(10, 20);
    S.O.P(objb.get());
  }
}
```

// container class must read value from its own & also its contained class.

objb.obja ⇒ X



Date-3/10/12

~~Objective~~

Static Refrence Variable :-

→ ~~Static~~ Reference variable is static within class So it is bind with class name :

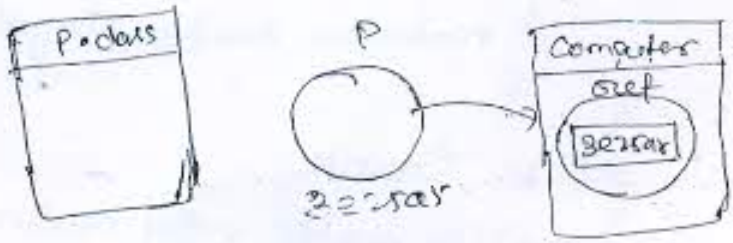
→ It allows to create global object, which is shared by more than one class or object.

example :-

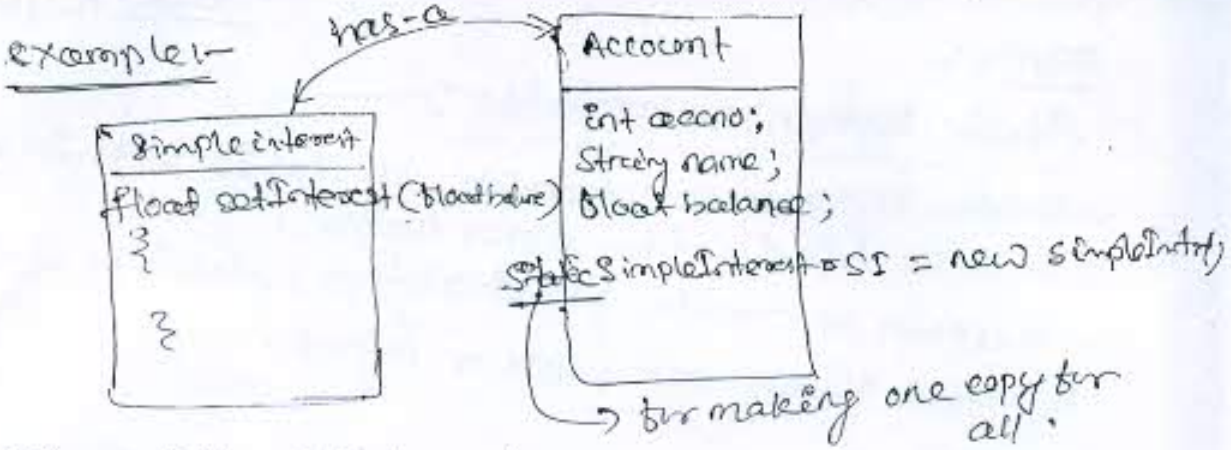
```
class P
{  void print(int x)
   {  S.O.P(x);
   }
   void print(float x)
   {  S.O.P(x);
   }
}
class Computer
{  static P oref = new P();
}
class CompDemo5
{  public static void main(String args[])
   {  Computer.oref.print(10);
      Computer.oref.print(1.5f);
   }
}
```

O/P

10
1.5f

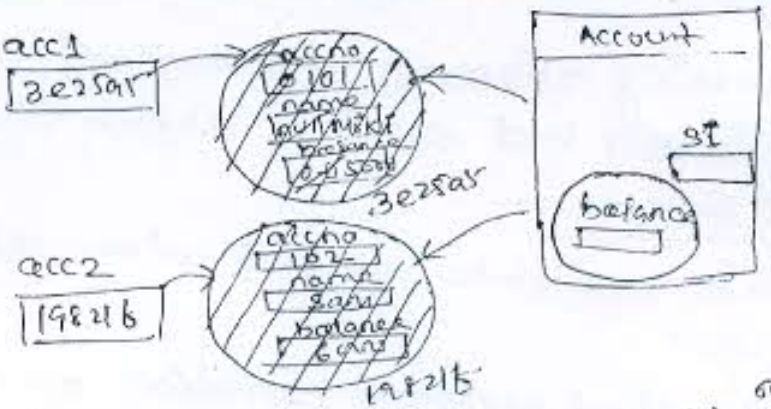


Example:-




```
class SimpleInterest
{
    float getInterest(float balance)
    {
        return (bal * 3 * 1.5f)/100;
    }
}
class Account
{
    private int accno;
    private String name;
    private float balance;
    private static SimpleInterest si =
                        new SimpleInterest();
    void setAccount(int accno, String name,
                        float balance)
    {
        this.accno = accno;
        this.name = name;
        this.balance = balance;
    }
    float getBalance()
    {
        return balance + si.getInterest(balance);
    }
}
class Bank
{
    public static void main(String args[])
    {
        Account acc1 = new Account();
        Account acc2 = new Account();
        acc1.setAccount(101, "Mike", 5000f);
        acc2.setAccount(102, "Sam", 6000f);
        float b1 = acc1.getBalance();
        float b2 = acc2.getBalance();
        System.out.printf("%f %f", b1, b2);
    }
}
```

O/P
5225.0
6270.0



→ Refrence variable always refer non-static members of a class.

```
class A
{ void m1()
  {
  }
  static void m2()
  { sop("Inside m2()");
  }
}
```

```
class B
{
  A obj = new A();
  void M3()
  {
    A.m2();
    obj.M1();
  }
}
```

→ Here m2() is static so we donot required to create an object it can be access by class name.

```
class B
{ A obj = new A(); ← it may be static or non-static instance variable.
  void m3()
  { A obj1 = new A(); ← local variable.
    =
  }
  void m4();
  {
  }
}
```

Method with parameters of type reference:-

(Pass by Reference)

→ Java allows to call a method into 2 ways

1. pass by value
2. pass by Refrence.

→ Passing reference to a method doesn't send address but send a values called hashcode :

→ This hashcode mapped with address of object.

→ A method perform operation on object by receiving another object ~~by receiving~~ using method with parameters of type reference.

→ This is called "use-a" relation.

→ This allow to a method perform operation on more than one object.

Date - 4/10/12

→ Method with parameters of type primitive
→ pass by value.

→ Method with parameters of type reference → pass by reference. it is internally send a value not address called hashcode.

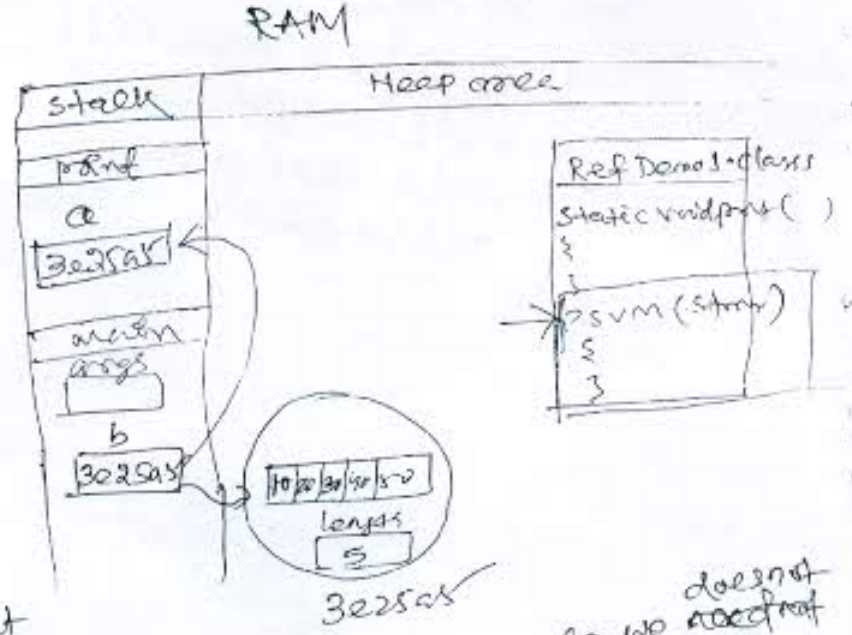
→ If method is having parameters of above & below 4 type (i.e type class) it receives hashcode of object -

1. Class
2. Interface
3. enum
4. Array.

example :-

```
Class RefDemo1
{
    Static void print(int a[])
    {
        for(i=0; i<a.length; i++)
            S.O.P(a[i])
    }
```

```
public static void main(String args[])
{ int b[] = {10,20,30,40,50};
  print(b);
}
}
```



print(b);

→ Here b contain hashcode. so we send reference.

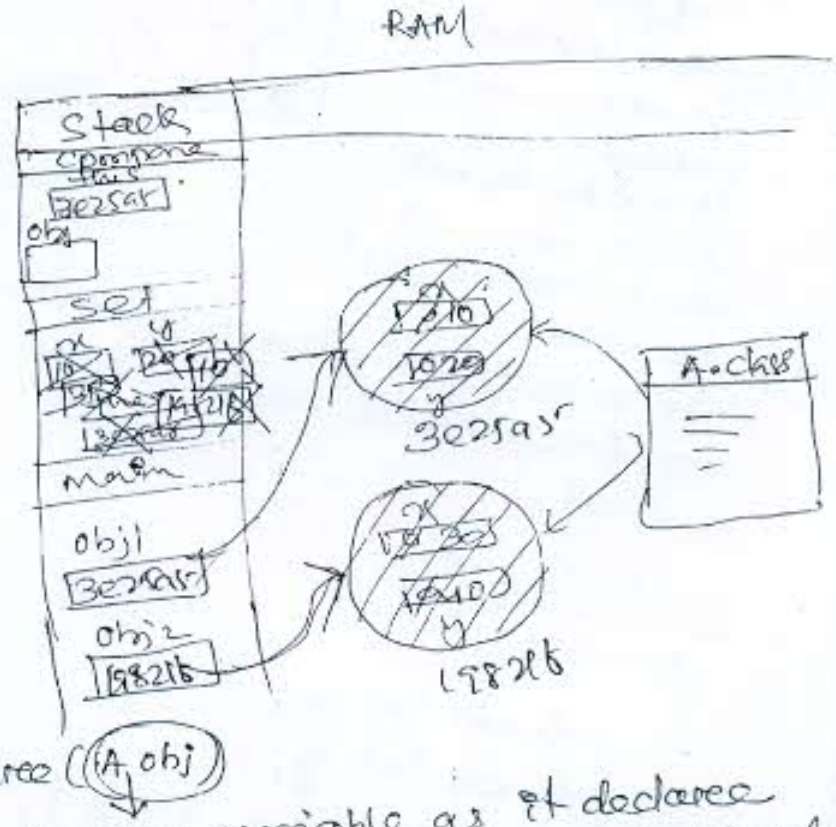
→ print() method is called by main() method. will if print() not static than we must create object for this.

→ As main() then present in the same class so we doesn't required to use the class name for calling the print() method.

example:—

```
class A
{ private int x;
  private int y;
  void set(int x, int y)
  { this.x = x;
    this.y = y;
  }
  void compare(A obj)
  { if(x == obj.x && y == obj.y)
      S.O.P("Equall");
    else
      S.O.P("not equall");
```

```
class RefDemo2
{ public static void main(String args[])
  {
      A obj1 = new A();
      A obj2 = new A();
      obj1.set(10,20);
      obj2.set(30,40);
      obj1.compare(obj2);
  }
}
```



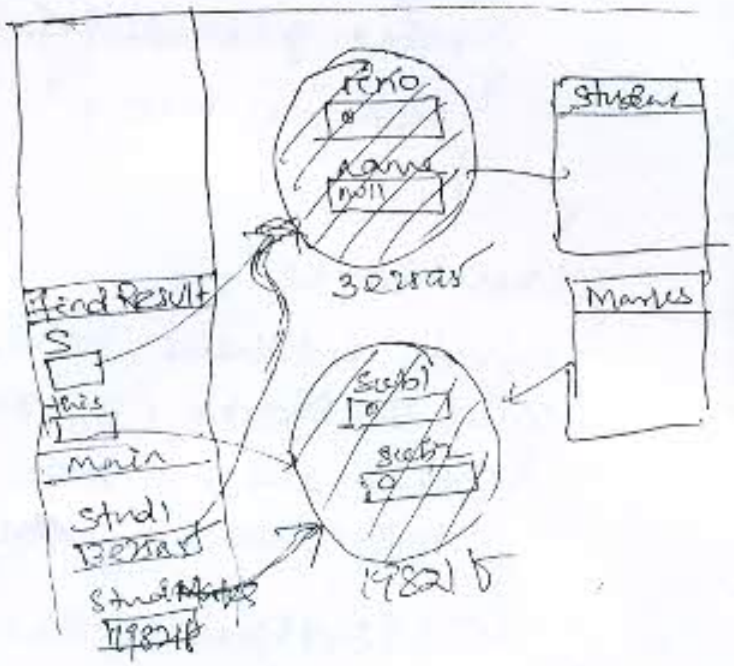
void compare((A obj)

→ obj is a reference variable as it declare with class name. It always stores the hashcode.

example:-

```
class Student
{
    private int rno;
    private String name;
    void setStudent(int rno, String name)
    {
        this.rno = rno;
        this.name = name;
    }
```

```
    String getStudent()
    {
      return rno + " " + name;
    }
}
Class Marks
{ private int sub1, sub2, sub3;
  Void SetMarks (int sub1, sub2)
  { this.sub1 = sub1;
    this.sub2 = sub2;
  }
  Void findResult (Student S)
  {
    S.O.P (S.rno); // invalid → becoz 'rno' is private cannot access outside the class.
    S.O.P (S.getStudent());
    if(sub1<40 || sub2<40)
    S.O.P ("fail");
    else
     S.O.P ("pass");
  }
}
Class RefDemo3
{
  public static void main (String args[])
  {
    Student stud1 = new Student();
    Marks stud1Marks = new Marks();
    Stud1.setStudent (101, "Rama");
    Stud1.M
    Stud1Marks.setMarks (40, 30);
    Stud1Marks.findResult (stud1);
  }
}
```



Constructors :-

→ Block of code which has to be executed on creation of object is defined inside constructors.

→ Constructor is a special method executed on creation of object.

→ Constructor is a initializer method, which defines initial state of an object / initial resources of object.

→ An object is never created without invoking constructors.

→ Automatic initialization of object is done using Constructors :

Types of Constructors :-

Java provides 2 types of Constructors.

1. Default Constructors
2. Parameterized Constructor.

→ Default constructors are of 2 types —
a. Compiler defined
b. User defined.

Date-5/10/12

Properties of Constructor:—

→ Constructor name is same as classname.
→ It can be declared as a private, default, protected or public.
→ It is invoked on creation of object.
→ It cannot be called explicitly.
→ It doesnot have any return type not even void.
→ It doesn't uses any modifiers (static, final, abstract etc.)
→ Its operation is initialization.

Q) Write the diff. bet^n a method and Constructors?

Method	Constructor
→ Method name may be same name as a class name or any name.	→ Constructor name must be same name as class name.
→ It having return type.	→ doesn't have return type
→ It allows access specifier as well as modifiers.	→ It allows only access specifier but doesn't have access modifiers.
→ It is called explicitly.	→ It is called implicitly at the time of creating object.
→ It is called any number of times.	→ It is called only once on object.
→ Operation performed by methods are setters & getters opⁿ.	→ Here the operation is initialization.
→ Method can called with name.	→ Constructor never called with name.

Method	Constructor
→ Methods are inherited.	→ Constructors are not inherited.
→ Constructor doesn't create object. but when the object created it invokes constructor. whose duty is initialization.	

Default Constructor :—

→ Non-parameterized constructor of class is called "default constructor."

(OR) → A constructor which doesnot have any parameters is called default constructor.

→ It is divided into 2 types —

(a) Compiler written default Const.

(b) User-defined default constructor.

Compiler defined default Constructor :—

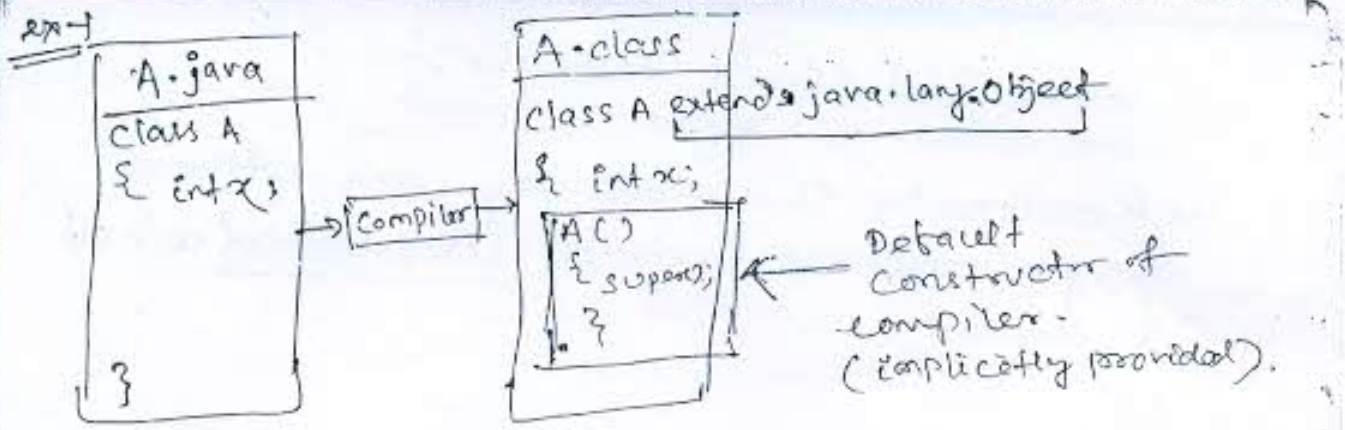
→ If there is no constructor defined within the class; compiler provides a constructor without any parameters, which is called default Constructors.

→ This constructor calls the constructor of superclass.

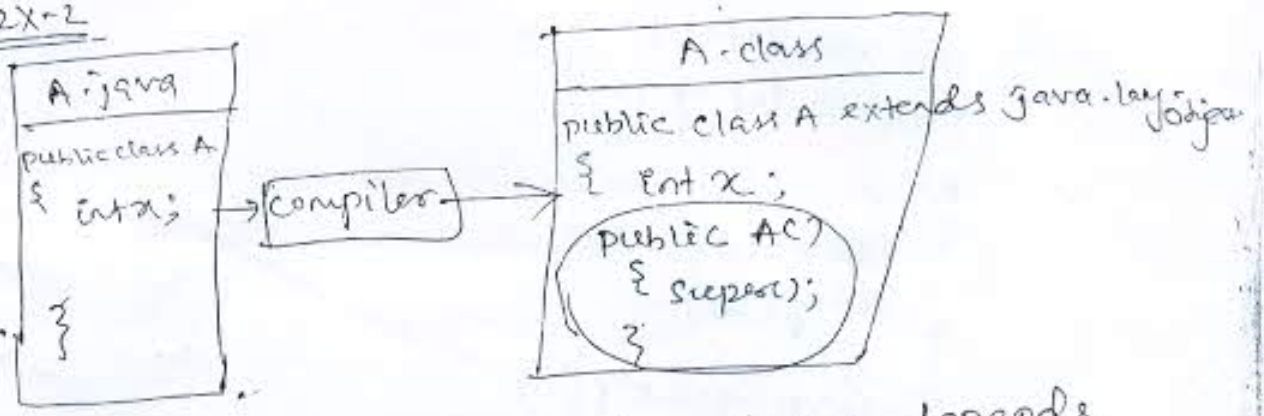
→ For all classes of java, root class is object.

→ Every class of java must be inherit properties & behaviour of object class.

ex-1



ex-2



→ Constructor access specifier always depends upon the access specifier of class. (extends as shown in above examples)

O/P → javap A ↵

Compiled from "A.java"
class A extends java.lang.object
{ int x;
A();
}

javaP :-

→ It means java class profiler.

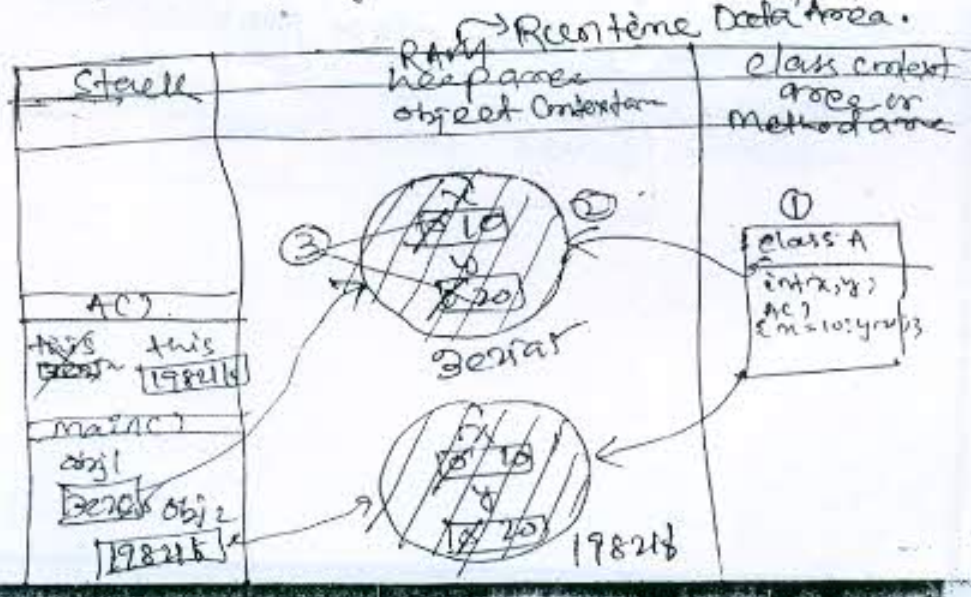
Syntax :- javaP class_name

User-defined default Constructor :—

→ A constructor is written by user without any parameters is called user defined default constructor.

ex

```
class A
{ private int x;
  private int y;
  A()
  { x=10;
    y=20;
  }
  String getXY()
  {
    return x+" "+y;
  }
}
class ConstDemo1
{
  public static void main(String args[])
  {
    A obj1 = new A();
    A obj2 = new A();
    S.o.pln(obj1.getXY());
    S.o.pln(obj2.getXY());
  }
}
```



→ Inside the RAM, Runtime Data area is present, which contain stack & heap area.
→ JVM reserves memory by using RAM.
→ Constructor is called along with class therefore constructor name is same as class name.
→ Constructor is a non-static.

A Obj1 = new A(); → calling the constructor along with new.

obj1.A(); → method.

→ Therefore when object created at that time Constructor invoke & initialize the member variables. So constructor is called only once on the object.

example:

```
class Date
{   private int dd;
    private int mm;
    private int yy;
    Date()
    {   dd = 1;
        mm = 1;
        yy = 2000;
    }
    void setDate(int dd, int mm, int yy)
    {   this.dd = dd;
        this.mm = mm;
        this.yy = yy;
    }
    void getDate()
    {   return dd + "/" + MM + "/" + YY;
    }
}
```

```
Class ConstDemo2
{
  public static void main(String args[])
  {
    Date date1 = new Date();
    S.o.pln(date1.getDate());
    date1.setDate(5, 10, 2012);
    S.o.pln(date1.getDate());
  }
}
```

example:-

```
class A
{
  A()
  { S.o.pln("Inside Constructor");
  }
  void A()
  { S.o.pln("Inside method");
  }
}
class ConstDemo3
{
  public static void main(String args[])
  {
    A obj1 = new A();
    obj1.A();
    obj1.A();
  }
}
```

O/P

```
Inside constructor
Inside method
Inside method.
```

Date- 5/10/12

Parameterized Constructor :—

→ A constructor with parameters is called parameterized constructor.

→ This is user-defined Constructor.

→ This constructor initializes object by receiving values.

example :-

→ Any statement that will be executed must be written inside a method.

→ By using constructor we can execute the statement without creating method.

example :-

```
class Login
{
    Button b1, b2;
    Textfield t1, t2;
    Label L1, L2;
    Login()
    {
        b1 = new Button();
        b2 = new Button();
        L1 = new Label();
```

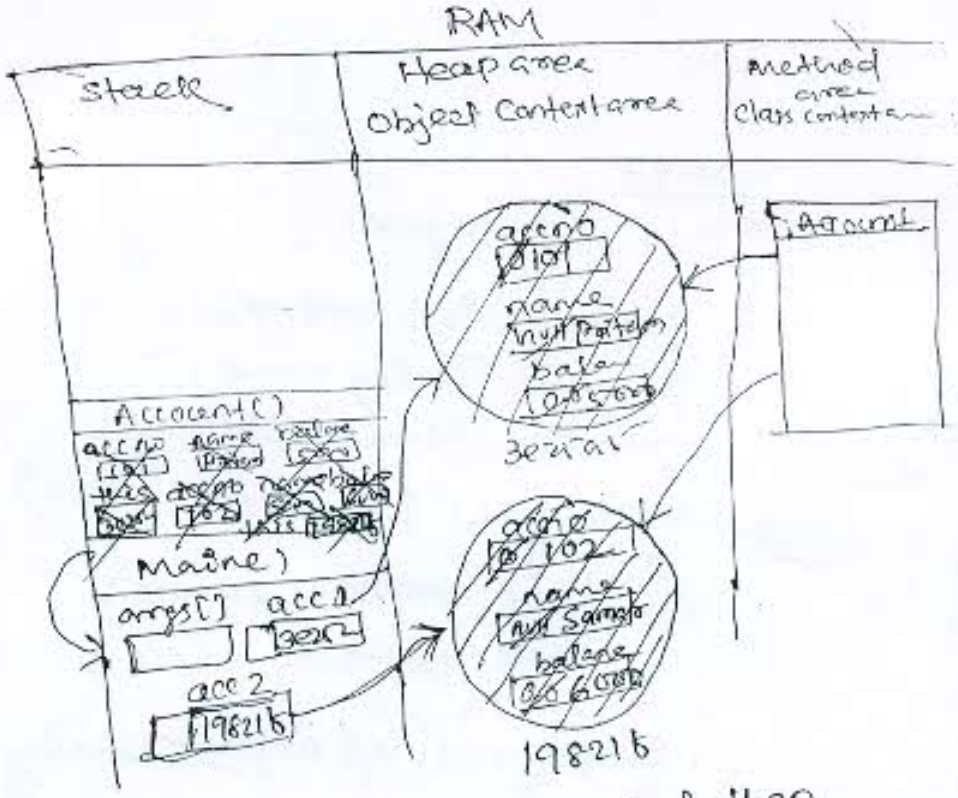
→ If we want to the change the state of an object by at the time of creation taking user input then we must declare parameterized constructor.

→ This parameters can be —

1. primitive
2. Referencetype.

```
class Account
{
  private int accno;
  private String name;
  private float balance;
  Account(int accno, String name, float balance)
  {
    this.accno = accno;
    this.name = name;
    this.balance = balance;
  }
  void deposit(float amt)
  {
    balance = balance + amt;
  }
  void withdraw(float amt)
  {
    if(amt > balance)
      S.o.pln("Insufficent balance");
      else
        balance = balance - amt;
  }
  String getAccount()
  {
    return accno + " " + name + " " + balance;
  }
}
class Bank
{
  public static void main(String args[])
  {
    Account acc1 = new Account();
    Account acc1 = new Account(101, "Ratan", 5000f);
    Account acc2 = new Account(102, "Kumar", 6000f);
    S.o.pln("acc1.getAccount());
    S.o.pln(acc2.getAccount());
    acc1.deposite(1000f);
```

```
acc2.withdraw(550f);
S.O.pln(acc1.getAccount());
S.O.pln(acc2.getAccount());
}
}
```



→ Once the constructor invoke and executed then memory reserved for constructor is deallocated & return to main().

Constructor Overloading :

→ Defining more than one constructor within class by changing • no. of parameters,
• type of parameters
• order of parameters
is called Constructor Overloading.

→ Constructor is overloaded inorder to extend the functionality of existing constructors.

→ Overloaded Constructor is having polymorphic behaviours.

example:

example:

```
class Employee
{
    private int empno;
    private String name;
    private float salary;
    Employee(int empno, String name)
    {
        this.empno = empno;
        this.name = name;
    }
    Employee(int empno; String name, float salary)
    {
        this.empno = empno;
        this.name = name;
        this.salary = salary;
    }
    void setSalary(float salary)
    {
        this.salary = salary;
    }
    String getEmployee()
    {
        return empno + " " + name + " " + salary;
    }
}
class Company
{
    public static void main(String args[])
    {
        Employee e1 = new Employee(101, "abc");
        Employee e2 = new Employee(102, "XYZ", 5000);
        S.o.println(" e1.getEmployee());
    }
}
S.o.println(" e2.getEmployee());
```

→ We can call a constructor inside another constructor to avoiding redundancy.

Date - 8/10/12

"this()" constructor call :-

→ Calling a constructor of a class within another constructor of same class is done using "this()" constructor call.

→ It allows to include functionality of one constructor inside another constructor in order to avoid redundancy

Rules :-

→ It must be a first statement within constructor.

→ It should not be a recursive.

→ It called only once within the constructors.

→ It is used only inside constructor but not inside method.

Q) whats the diff b/n "this" & "this()" ?

this	this()
→ It hold hash code of current object.	→ It refers to constructor of same class.

ex :-

```
Class A
{
   A()
   { S.o.pln("Inside Default constructor");
   }
   A(int a)
   { this();
     S.o.p("Inside parameterized constructor");
   }
}
```

```
class ConstDemo6
{
  public static void main (String args[])
  {
    A obj1 = new A(10);
  }
}
```

o/p :- inside default Const-
inside Parameterised Const.

* Always a method called with its name.
* By using this() we can call the default constructor.

ex-2 :

```
class A
{
  A()
  {
    S.o.p(inside default Const)
  }
  A(int p)
  {
    A();
    this();
    S.o.p(inside Param. Const");
  }
  void A();
  {
    S.o.p("inside method");
  }
}
```

* otherwise error

```
class ConstDemo7
{
  psvm(String args[])
  {
    A obj = new A(10);
  }
}
```

o/p :- inside ~~default Const~~ method 2
inside default Const. 1
inside param. Const. 3

ex-3 :-

```
class A
{
  A()
  {
    S.o.p("inside default Const");
  }
  A(int x)
  {
    this();
    A();
    S.o.p("inside param Const.");
  }
```

```
    void A()
    { s.o.p ("Inside method");
    }
}
Class ConstDemo8
{ psvm (String args[])
  { AObj1 = new A();
  }
}
```

O/P :- Inside default const.
Inside method.
Inside param. const.

Ex-4:-

```
Class Accocent
{ private int accno;
  private String cname;
  private float balance;
  Accocent (int accno, String cname)
  { this.accno = accno;
    this.cname = cname;
  }
  Account (int accno, String cname, float balance)
  {
    this (accno, cname);
    this.balance = balance;
  }
  String getAccocent()
  { return accno + " " + cname + " " + balance;
  }
}
Class ConstDemo9
{ psvm (String args[])
  { Account acc1 = new Account (101, "abc");
    // Account acc2 = new Account (102, "xyz", 5000f);
```

```
Account acc3 = new Account(102, "XYZ", 5000f);
S.O.P(acc1.getAccount());
S.O.P(acc2.getAccount());
}
}
```

O/P 101 abc 0.0
102 XYZ 5000f.

ex-5 :- (Rule-1)

```
class A
{ A()
  { S.O.P("Inside default Const");
  }
  A(int x)
  { S.O.P("Inside parameterized Const");
    this(); //error
  }
}
```

→ The above program display compile time error, becoz called to this must be first statement within constructor.

ex-6 :- (Rule-2)

```
class A
{ A()
  { this();
    S.O.P("Default Const");
  }
}
```

→ The above program display compile time error becoz a constructor cannot be recursive.

→ As the constructor is initialize once so constructor cannot be initialize more than one. (but method allows recursive).
The above example shows direct recursion.

ex-7 :-

```
Class A
{  A()
   {  this(10);
      S.o.p("Inside default");
   }
   A(int x)
   {  this();
      S.o.p("Inside parameterized");
   }
}
```

⇒ The above program display compile time error becoz recursive constructr call is not allowed.

→ It shows indirect recursive.

ex-8 :- (Rule-3)

```
Class A
{  A()
   {  S.o.p("Inside default");
   }
   A(int x)
   {  S.o.p("Inside one para.");
   }
   A(int x, int y)
   {  this();
      this(10); // error
      S.o.p("Inside Two para.");
   }
}
```

→ The above program display compile time error, becoz call to this must be first statement.

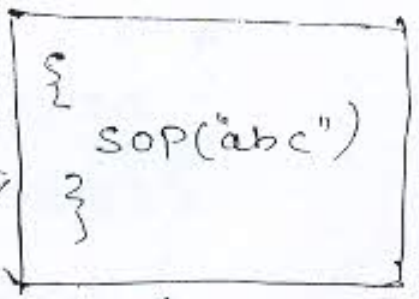
ex-09 :- (Rule-1)

```
class A
{
    A()
    {
      S.O.P("Inside default Const");
    }
    Void m1()
    {
      this();
      S.O.P("Inside method");
    }
}
```

→ The above program displays compile time error. becoz call to this() must be first statement inside constructor, but not inside method.

ex-10 :

```
class A
{
  A()
  { S.OP("abc");
    S.OP("XYZ");
  }
  A(int x)
  { SOP("abc");
    S.O.P("PQR");
  }
  A(int x, int y)
  { SO.P("abc");
    SOP("MNQ");
  }
}
```



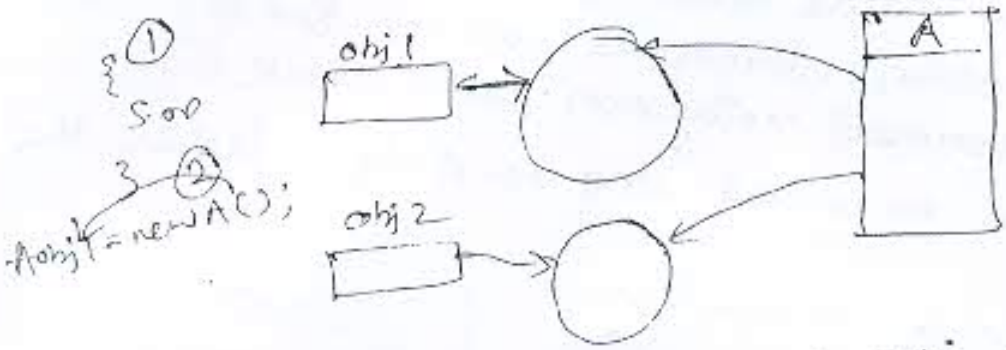
Non static block.

→ Non static block is used to initialize the object.

→ The block having no name.

→ If the code is common for all the constructors, then that code must be present in a non-static block.

O/P :- Inside non static block
Inside default Constructor.
Inside non static block
Inside parameterized constructor.



→ It is an anonymous block & this block has no name.
→ The class have any no. of non-static block.
→ The order of their execution is that the order in which they appear.

Ex-2

```
class A
{
    {
      S.O.P("Inside non-static block-1");
    }
    A()
    { S.O.P("Inside default Const");
    }
    {
      S.O.P("Non-static block 2");
    }
    A(int x)
    { S.O.P("Inside para. Const");
    }
}
```

```
class ConstDemo16
{
  public static void main (String args[])
  {
    A obj1 = new A();
    A obj2 = new A(10);
  }
}
```

O/P :- Inside non-static block-1
Non-static block-2
Inside default const.
Inside non-static block-1
Non-Static block-2
Inside para const.

→ Order of executing non-static blocks are the order in which they appear/declared within the class.

Static block :-

```
class System
{
  static PrintStream out;
  static
  {
    out = new PrintStream();
  }
}
```

→ A block of code which has to be executed on loading class included within static block.
→ It is used for static init or class init.
→ static block access only static members of a class but cannot access non static members.

Syntax :-

```
static
{
    statements;
}
```

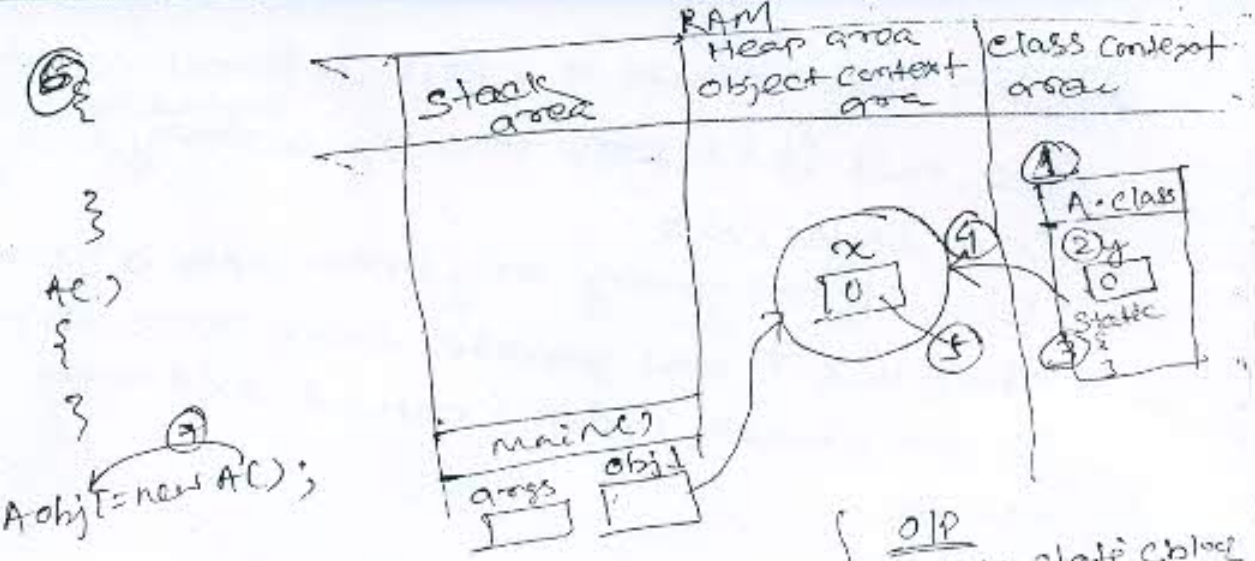
Note :-

→ Inside the class we can defined the following things -

1. Variables
   - non static variable
   - static variable.
2. methods
   - non static method
   - static method
3. Constructors.
4. block
   - static block
   - non-static block.
5. classes.

ex :-

```
class A
{   int x;
    static int y;
    {  S.O.P("Inside non-static block");
    }
    static
    {  S.O.P("Inside static block");
    }
    A()
    {  S.O.P("Inside Constructor");
    }
}
class ConstDemo17
{   public static void main(String args[])
    {   A obj1 = new A();
}   }
```



o/p
inside static block
inside non-static block
inside constructor

① → class is loaded
② → Allocate memory for static variable
③ → execute static block
④ → Creating object
⑤ → Initialize instance variable
⑥ → execute non-static block
⑦ → execute constructor.

→ when class is loaded 1st ①, ② & ③ are happen.
but when object is created all are execute sequentially.
→ sequence may change in inheritance.

Q) what is the diff. bet^n static & non-static block?

Static block	Non-static block
→ It is use for class initialization.	→ It is use for object initialization.
→ It access only static members of a class	→ It access static & non-static members of a class.
→ It get executed on loading of class.	→ It get executed on creation of object & before invoking constructor.
→ executed only once.	→ executed whenever object of a class is created.
→ It is without this.	→ It is with this.

2) Can you execute a class without main?

→ Yes, but it is done by writing everything including in a static block.

→ But it is not giving any error upto JavaV1.6 after this it was provide error because in java1.7 it first checks main() method exist or not then execute.

Ex :-

```
Class ConstDemo18
{
    static
    { S.O.P("Inside static block");
    }
}
```

O/P :- Inside static method
Exception in thread main()

Ex

```
Class ConstDemo18
{
    static
    { S.O.P("Inside static block");
      System.exit(0); // terminating execution of program.
    }
}
```

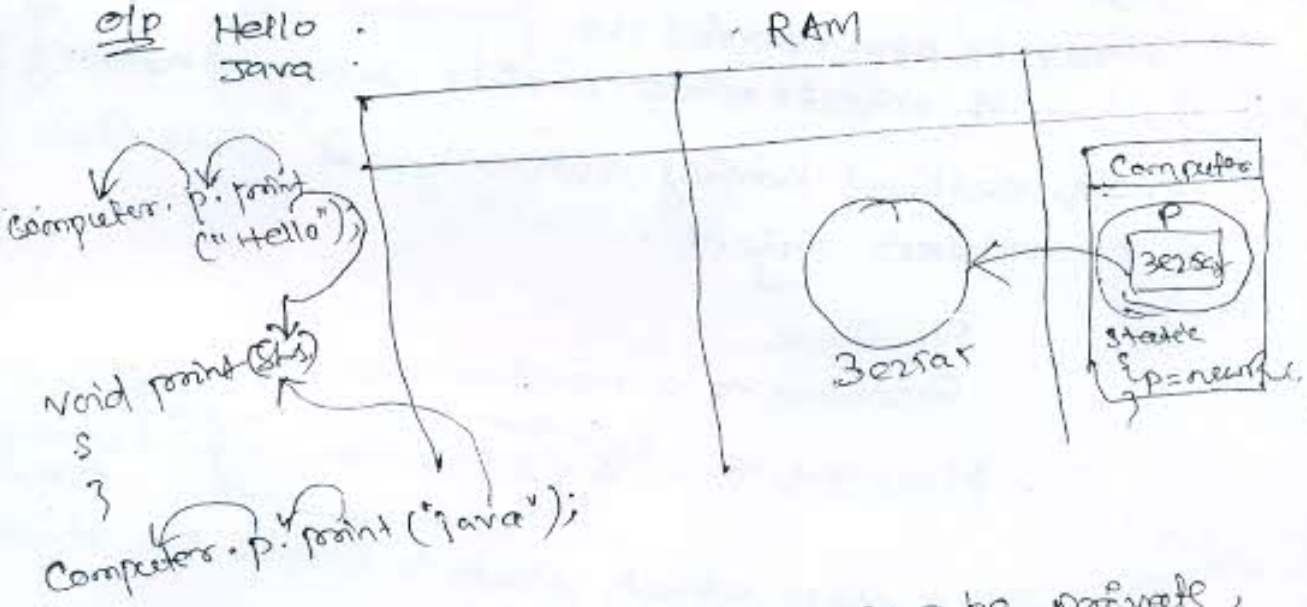
O/P Inside static block.

Ex :-

```
Class Pointers
{
    void Print(String s)
    { S.O.P(s);
    }
}
```

```
class Computer
{
    static Printer p;
    static
    {
        p = new Printer();
    }
}
class Word
{
    public static void main(String args[])
    {
        Computer.p.print("Hello");
        Computer.p.print("Java");
    }
}
```

o/p Hello
Java



→ A constructor within a class can be private, protected, public & default (called access specifiers).

10/10/12

Private Constructor :-

→ Constructor of a class can be declared as private.

→ Private constructor cannot access outside the class.

→ Private constructor is declared in order to restrict a class from creating object.

→ Programmer is not allowed to create object directly. It allows to create object indirectly with the help of methods.

→ We restrict the class from creating object for beoz to restrict the programmer creating 'n' no. of objects which avoids wastage of memory.

→ If method having return type of class then it return object.

Ex

```
Rentdome
Rentdome r = new Rentdome() X
Rentdome r = Rentdome.getRentdome(); → Factory method
```

Rentdome

Q) Develop a class which allows to create one object.

Q) What is Factory method ?

→ Factory method having following properties —

1. Static
2. Return type is class.

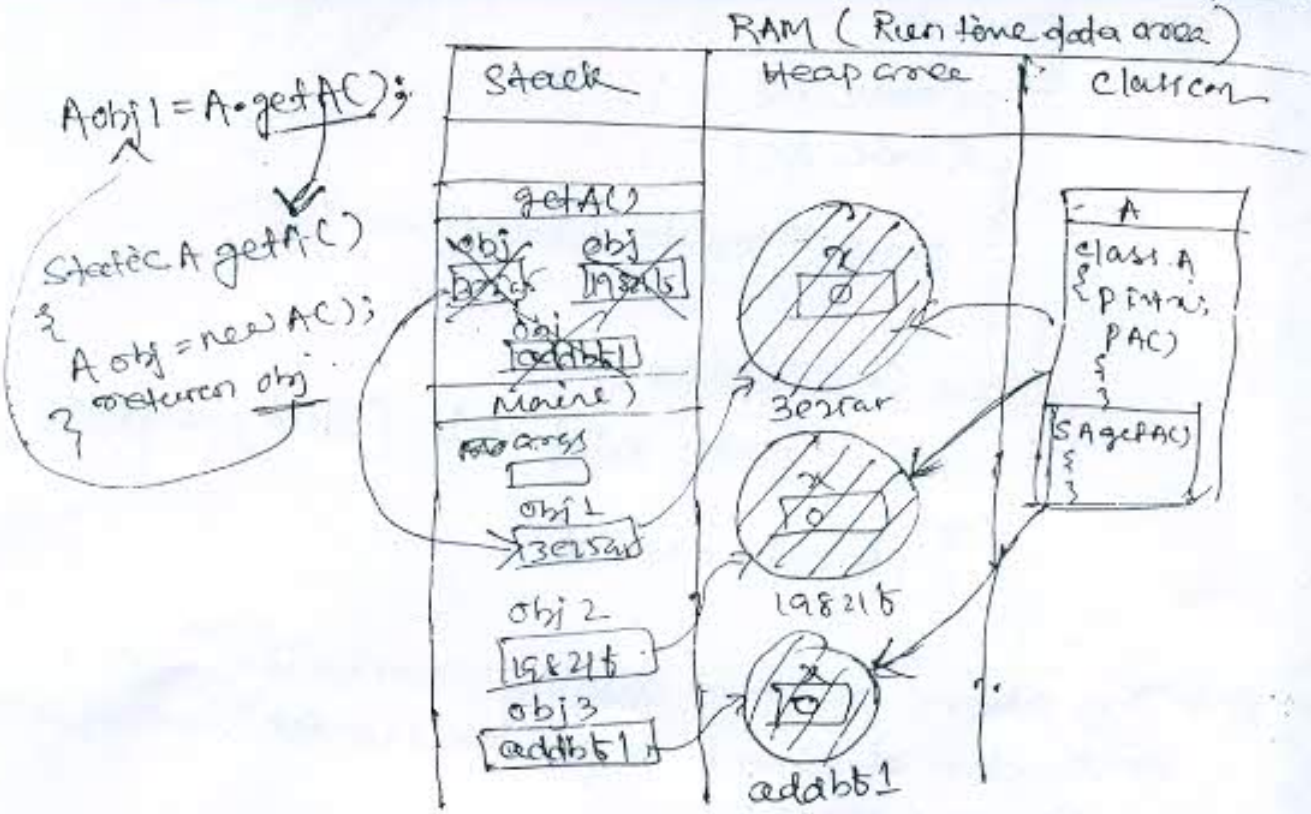
→ A method which create an object and return hashcode of that object is called factory method.

→ If method having return type as class, then it returns hashcode of object.

ex1:
```
class A
{
  private int x;
  private A()
  {
    S.o.p("inside default const");
  }
}
class ConstDemo20
{
  public static void main(String args[])
  {
    A obj1 = new A();
  }
}
```

* → The above program display compile time error constructer of class A is private, which cannot access outside the class.

ex-2
```
class A
{
  private int x;
  private A()
  {
    S.o.p("inside default const");
  }
  static A getA()
  {
    A obj = new A();
    return obj;
  }
}
class ConstDemo21
{
  public static void main(String args[])
  {
    A obj1 = A.getA();
    A obj2 = A.getA();
    A obj3 = A.getA();
    S.o.p(obj1);
    S.o.p(obj2);
    S.o.p(obj3);
  }
}
```



Single-ton class :——

→ A class which is having following properties is called single-ton class.

1. private Constructor.
2. final class (not inherited).
3. Allows to create only one copy of object.
4. provide factory method to create object.

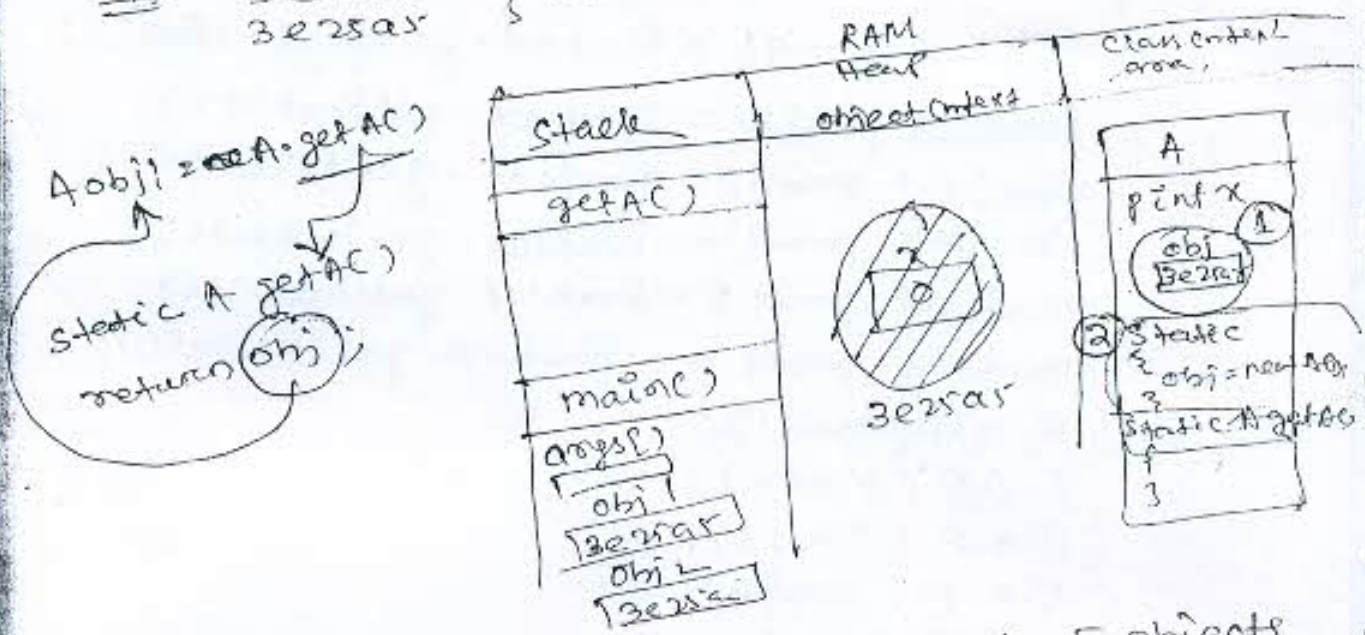
→ A class which allows to create only one copy of object is called <u>single-ton class</u>.

<u>ex</u>:-

```
class A
{ private int x;
  private static A obj;
  static
  { obj = new A();
  }
  static A getAC()
  { return obj;
  }
}
```

```
Class ConstDemo22
{ public static void main(String args[])
  { //A obj1 = new A(); // error.
    A obj1 = A.getA();
    A obj2 = A.getA();
    S.o.p(obj1);
    S.o.p(obj2);
  }
}
```

O/P:- 3e25a5 } returns same hashcode.
3e25a5



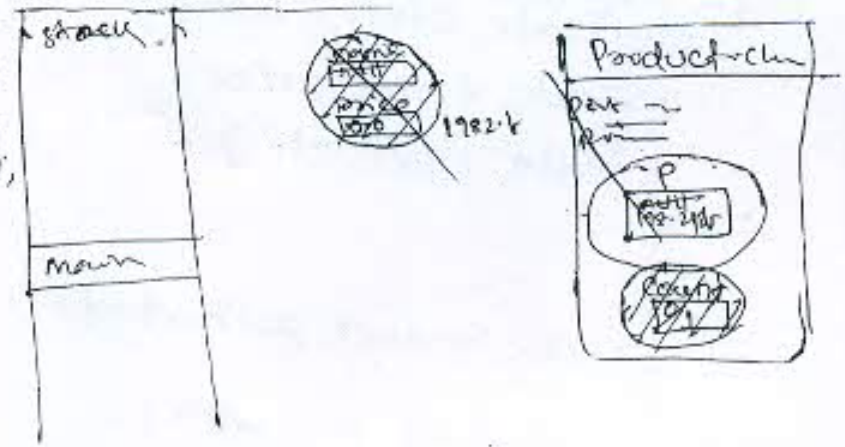
Q) Develop a class which allows to create 5 objects or instances.

```
Class Product
{ private static int count;
  private String name;
  private float price;
  private product()
  {
  }
  static Product getProduct()
  {
    Product p = null;
    &&
```

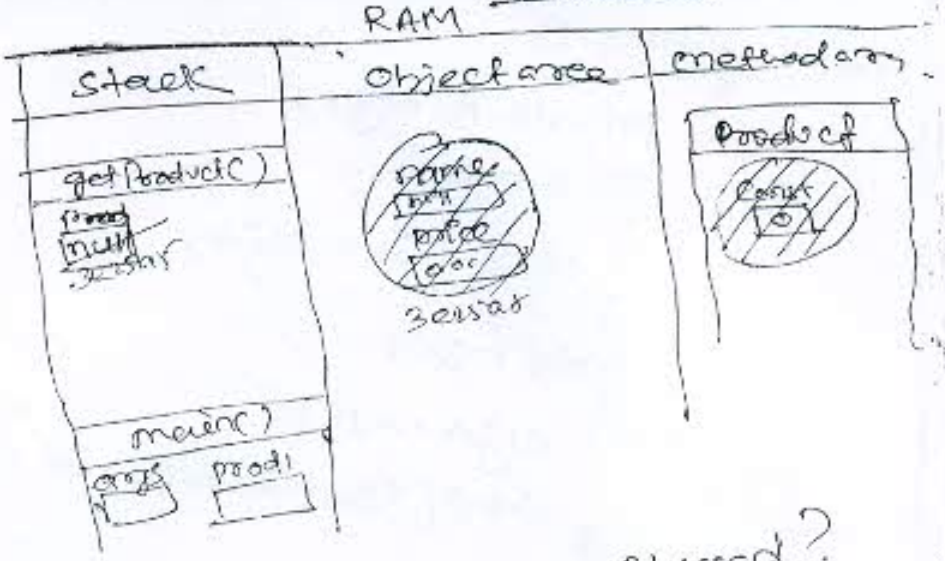
```
        if(cocent < 5)
        {
            p = new Product();
            Cocent = Cocent + 1;
        }
        return p;
    }
}
class ConstDemo23
{
    public static void main(String args[])
    {
        Product prod1 = Product.getProduct();
        Product prod2 = Product.getProduct();
        Product prod3 = Product.getProduct();
        product prod4 = Product.getProduct();
        product prod5 = Product.getProduct();
        product prod6 = Product.getProduct();
        S.O.P(prod1);
        S.O.P(prod2);
        S.O.P(prod3);
        S.O.P(prod4);
        S.O.P(prod5);
        S.O.P(prod6);
    }
}
```

O/P :- 1982.b
add bb1
A2e81b
930461

Prod prod1 = Product.getProduct();



11/10/12



2nd) What is a Singleton class? Where is it used?

(i) → Singleton is a design pattern meant to provide one & only one instance of an object. Other objects can get a reference to this instance through a static method (class constructors is kept private).

(ii) → Restricting the no. of instances may be necessary or desirable for technological or business reasons.
for example, we may only want a single instance of a pool of database connections.

Constructors with parameters of type reference :-

→ A constructor having parameters of type reference receive hashcode of object or object.

→ An object can be initialized with resources of another object

ex :-
```
class A
{
    void m1()
    {
        S.O.P("Inside m1 of class A");
    }
}
```

```
class B
{
  private A obja;
  B(A obja)
  {
    this.obja = obja;
  }
  void m2()
  {
    obja.m1();
    S.o.p("Inside m2() of class B");
  }
}
class ConstDemo2
{
  public static void main(String args[])
  {
    A obj1 = new A();
    B obj2 = new B(obj1);
    obj2.m2();
  }
}
```

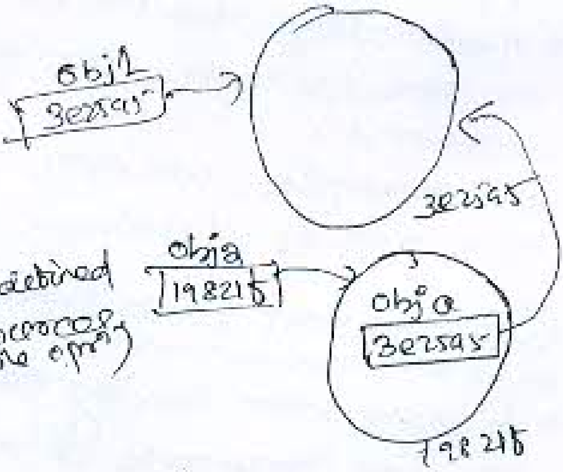
A obj3 = new A()
A obj2 = new A()
→ A obj1 = new A()
→ B obj2 = new B(obj1);

object may be defined dynamically (memory)

→ B obj2 = new B(obj1);

B(A obja)

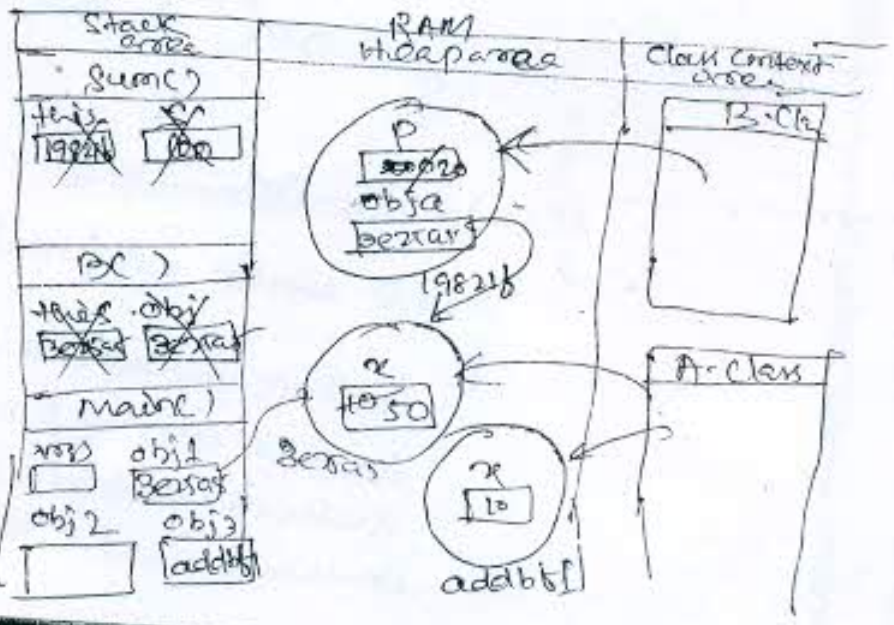
{



O/P :- Inside m1 of class A
Inside m2 of class B.

ex :-

```
class A
{
    int x = 10;
}
class B
{
    int p = 20;
    A objA;
    B(A objA)
    {
        this.objA = objA;
    }
    void sum()
    {
        int s = p + objA.x;
        S.O.P("sum is " + s);
    }
}
class ConstDemo2e
{
    public static void main(String args[])
    {
        A obj1 = new A();
        obj1.x = 50;
        A obj3 = new A();
        B obj2 = new B(obj1);
        obj2.sum();
    }
}
```



Ex:-

```
class Acthore
{
  private String aname;
  void setAcuent (String aname)
  {
    this.aname = aname;
  }
  String getAname()
  {
    return aname;
  }
}
class Book
{
  private String bname;
  private Acthor a;
  Book(Acthor a)
  {
    this.a = a;
  }
  void setBname (String bname)
  {
    this.bname = bname;
  }
  void printBook()
  {
    S.O.P("Book name" + bname);
    S.O.P("Acthor name" + a.getAname);
  }
}
class ConstDemo27
{
  public static void main (String args[])
  {
    Acthore acthor1 = new Acthore();
    Acthor acthor2 = new Acthore();
    acthor1.setAname("James");
    acthor2.GetAname("Scott");
```

```
Book book1 = new Book(authore1);
Book book2 = new Book(author1);
Book book3 = new Book(author2);
book1.setBname("Java");
book2.setBname("Complet Java");
book3.setBname("Oracle");
book1.printBook();
Book2.printBook();
book3.printBook();
                    // author1.setAname("Gosling")
                       book1.printBook();
}
```

```
}
```

O/P

Bookname Java
authorname James
B.n → Complete Java
A.n → James.
B.n → Oracle
A.n → Scott