JAVA

History Of Java

Initially Java language is name as 'Oak' in 1991. The first version of JDK was 1.0 & the latest release of Java Standard Edition is Java SE10.

Characteristics :-

Simple = It is simple than C & C++ and it also eliminates the pointer concept which is earlier present in C & C++.
Java also has a properties like automatic allocation of memory and garbage collection where as in C/C++ the garbage collection and allocation of memory will be done by the programmer which is a complex task.

Object Oriented = Java is object Oriented program which provides great range of reusability, modularity and flexibility.

Interpreted = When Java program are compiled, it produces the byte code which is m/c understandable language. So, Interpreter will convert the byte code to the m/c language

Scanned with CamScanner

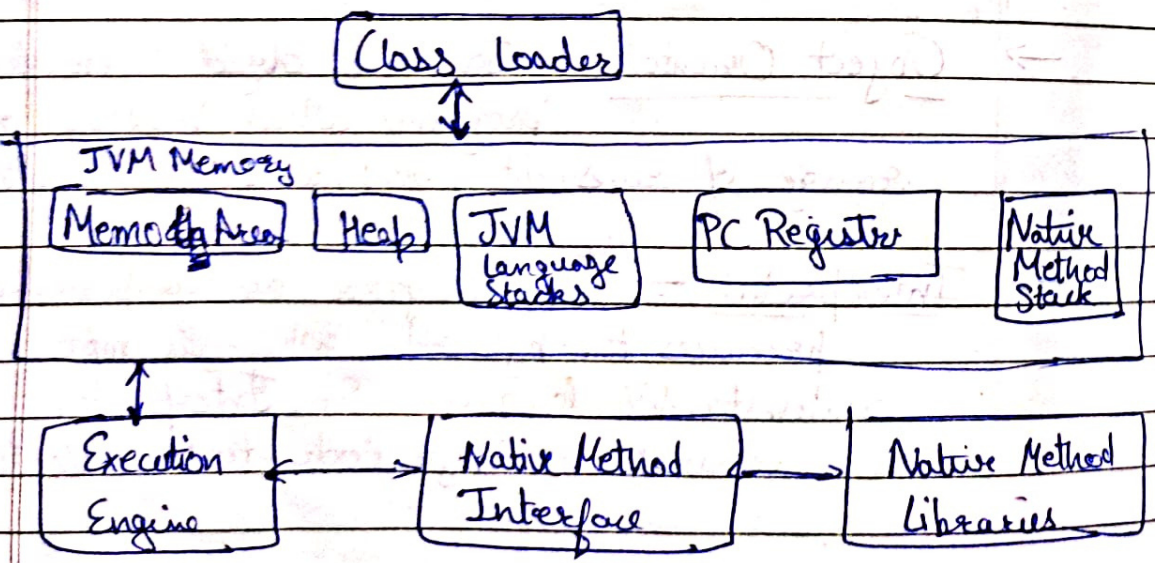
→ **Robust and Secure** :- Java programming is more reliable. At the time of execution, it shows all the error. Java also provides the lot of security as the computer will be attacked by the external program.

→ **Multi-threaded** :- It is defined as the programs ability to perform several tasks or function simultaneously.

JAVA VIRTUAL MACHINE (JVM)

↳ It is a engine that provides runtime environment to drive the Java Code or applications.
↳ It converts Java byte code into m/c language.
↳ It is responsible for allocating memory space.

Architecture :-



- **Class Loader :-**
  - loading
  - Linking
  - Initialization

- **Method Area :-** It stores method area class structure like Meta Data, the constant runtime and code for methods

- **Heap :-** All the ~~of~~ objects, their related instance variables and arrays are stored in the heap.

- **JVM Language Stack :-** It stores local variables & its partial results. Each thread has its own JVM stack, created simultaneously as the thread is created.

A new frame is created whenever a method is invoked and deleted when method invocation is complete.

- **PC Registers :-** It stores the address of the JVM instruct. which is currently executing. In JAVA each thread has its seperate PC register.

Native Method Stacks :- It holds the instruction of native code, depends on the native library. It is ~~instead~~ of written in another language instead of JAVA.

Execution Engine :- It is a type of S/W used to test hardware, software, or complete system.

- **Native Method Libraries :-** It is a collection of Native Lib. which needed by the execution engine.

Native Method Interface :- It is a programming framework. It allows JAVA code which is running in a JVM to call by libraries and native applications.

Scanned with CamScanner

⇒ Class Loader Subsystem
- Loading
- Linking
- Initialization

1) Loading :- Reads the class file, generating corresponding binary data and save it to method area.
For each class file, JAVA stores following informations in Method Area.

↳ fully qualified name of the loaded class and its intermediate parent class.

↳ whether class file is related to Class or interface or Enum.

↳ Modifier, variable and method information etc.

2) Linking :- Performs verification, preperation and resolution.

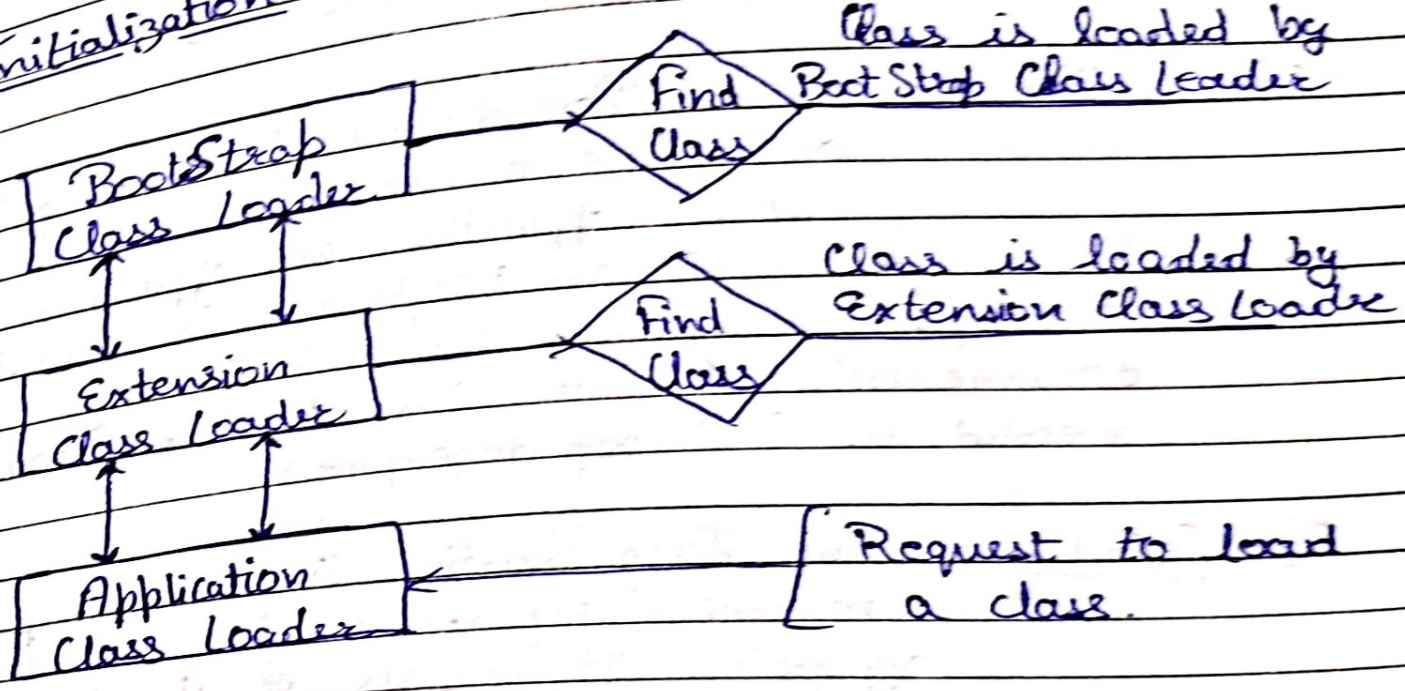
↳ Verification = Ensure correctness of class file i.e. it check whether this file is properly formated and generated by valid compiler or not.

↳ Preperation = JVM allocate memory for class variables and initializing the memory to default values

↳ Resolution = It is the process of replacing symbolic refrences from the type with direct refrences.

Scanned with CamScanner

Initialization :-



Scanned with CamScanner

Page No.

Diff b/w JDK, JRE, JVM

1> **JDK** :- Java Development Kit is a S/w development environment used for developing JAVA application and applets.
It includes JRE, an interpreter / loader (JAVA), a compiler (JAVAC), an archiever (jar), a documentation generator (Java doc) and other tools needed in JAVA ~~dep~~ development.

2> **JRE** :- The Java Runtime Environment provides the minimum requirements for executing a JAVA application. It consists of the Java Virtual Machine, core classes and supporting files.

Lecture 4

3> **JVM** :- A specification where working of JVM is specified. But implementation provider is independent to choose the algorithm. It implementation has been provided by ~~JRM~~ Sun and other companies

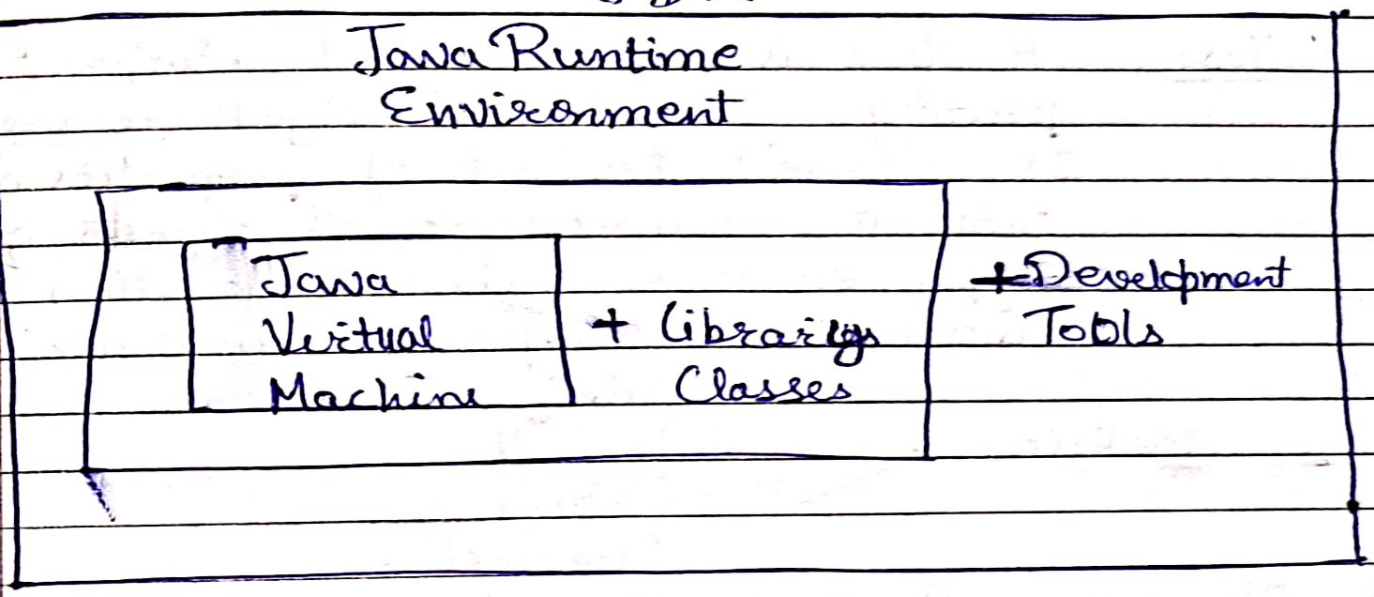
↳ A implementation is a computer program that meets the requirements of the JVM specification.

↳ Runtime Instance whenever you write java command on the compand prompt to run the Java class, an instance of JVM is created.

1.
2.
3.
4.
5.
6.

Scanned with CamScanner

JDK



JDK = JRE + Development Tools
JRE = JVM + Library classes

OOPS Concept In JAVA

Object Oriented Programming

OOPS refers to languages that uses objects in the programming.

The main aim of OOP is to bind together the data and the functions that appreciate on them. So, that no other part of the code can access this data except that function.

OOPS Concepts

Class
Object
Method
Message Passing
Abstraction
Inheritance
7. Encapsulation
8. Polymorphism.

Scanned with CamScanner

Page No.

1. **Class** :- A class is a user defined blueprint or prototype from which object are created. It represent the set of properties or method that are common to all objects of one type. In general, class declaration, can include these components in order:

Modifier
- Public
- Default
- Private
- Protective

2) **Object** :- An entity that has state and behaviour of class is known as object.

3) **Method** :- It is a collection of statements that perform some specific task and return the result to the caller.
Method can perform some specific task without returning anything.
Methods are the time savers and help us to reuse the code without retyping the code.

4) **Message Passing** : {Communication b/w Processes}

Message passing in JAVA is like sending an object i.e. message from thread to another thread.

5) **Abstraction** :- It is a property by which only the essential details are displayed to the users.

Scanned with CamScanner

Inheritance :- It is the mechanism in java by which one class is allow to inherit the features of another class.

Super Class ⟶ Sub Class

Encapsulation :- Wrapping up of data in a single unit. It is the mechanism that binds together code and the data is manipulated.

Polymorphism :- One to many forms. It allows us to perform a single action in different ways.

Ex :-
'+' Operator in Java is used for addition purpose as well as it is also used for string concatenation.

Types Of Polymorphism

- Run Time Polymorphism
- Compile Time Polymorphism.

Scanned with CamScanner

Page No.

lecture 5

TYPES OF POLYMORPHISM

1) Compile time Polymorphism (Static Polymorphism)

A polymorphism i.e. resolved during compile time is known as static polymorphism

Types
- Method Overloading
- Operator Overloading

a) Method Overloading :-

When there are multiple method function with same name but different parameters then these methods are called method overloading.
Methods can be overloaded by change in number of arguments / change in type of arguments.

Ex :-
```
class MultiplyFun
{
    Static int Multiply (int a, int b)
    {
        return a * b;
    }
    Static double Multiply (double a, double b)
    {
        return a * b;
    }
}
```

Method Name

Method with two integer parameters

Method with same name but two double parameter

Operator Overloading :-

Java also provide option to overload operators

Ex:- Single operator '+' when place b/w integer operands adds them and when placed b/w strings operands, concatenate them.

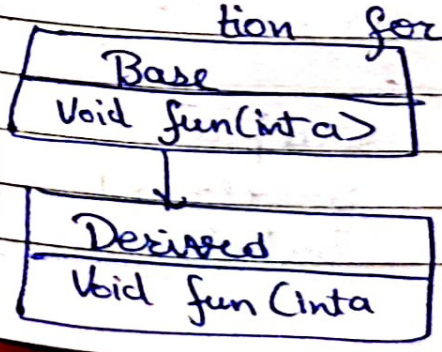
In Java Only '+' operator is overloaded

Run Time Polymorphism (Dynamic Polymorphism)

It is a process in which a function call the overridden method, is resolved at Runtime. This type of polymorphism is achieved by method overridding

Advantage of Run Time Polymorphism

Support Method Overriding
Common Method Specification
↳ It allows subclasses to add specific method and allow subclasses to define specific implementation for the same.



Scanned with CamScanner

Page No.

Ex:-

```
class Parent
{
  void Print ()
    System.Out.Println("Parent Class");
}
class Subclass 1 extends Parent
{
  void Print ()
  System.Out.Println ("Subclass 1");
}
class Subclass 2 extends Parent
{
  void Print ()
  System.Out.Println ("Subclass 2");
}
class Testpolymorphism
{
  public static void main (String [] args)
  {
    Parent a;
    a= new Subclass 1();
    a.Print ()
    a= new Subclass 2();
    a.Print;
  }
}
```

Output
Subclass 1
Subclass 2

Scanned with CamScanner

JVM Class File Format :-

JVM ~~class~~ source code is compiled into Byte Codes using Javac (compiler)

Source file .Java	Javac	Bytecode .class	Interpreter / JIT Compiler	Assembly / Machine Code

Byte code are saved in a binary file format. These class files (bytecodes) are then loaded dynamically into memory by class loader component of JVM. Java execution engine executes these bytecode one by one on the host machine CPU.

Ex:-
```
Public Class classOne
{
Public Static void main (String[] args)
  {
    System.out.Println ("Hello World");
  }
Static class Static Nested class
  {
  }
Class class Two
  {
    interface Interface One
    {
    }
```

After compilation we get

1> class one & Static Nested class . class
2> Class One . class
3> ClassTwo . class
4> Interface One . class

Java Class File Format :-

Magic Number	Version
Constant Pool	
Access Flags	
This class	
Super Class	
Interface	
Fields	
Methods	
Attributes	

Scanned with CamScanner

We can remember the 10 section with
"My Very Cute Animal Turn Savage In Full Moon Area"

Page No.

Magic Number:- The first four bytes of every class file are always 0XCAFEBABE. These four bytes identify the class file format from others.

Major and Minor Version:-

This is the second four byte of the class file containing the major and minor version number. We denote the version of its class file format as M.m

Every JVM has a max. version, it can load and JVM's will reject files with later versions.
Ex:- Java 11 supports major version from 45 to 55 and Java 12 supports major version from 45 to 56.

Constant Pool:- # include, variable name, class name, interface name, method name, string literals etc.

The constants are stored as a variable length array. elements in the constant pool. The array of constants are proceded by its array size.
Ex:- Minor Version = 0
Major Version = 50
Constant Pool = Const #1 = Method #4 #13;
Const #2 = int 16707053 - integer value
Const #4 = class #14 } this class/
Const #5 = class #15; } super class

Scanned with CamScanner

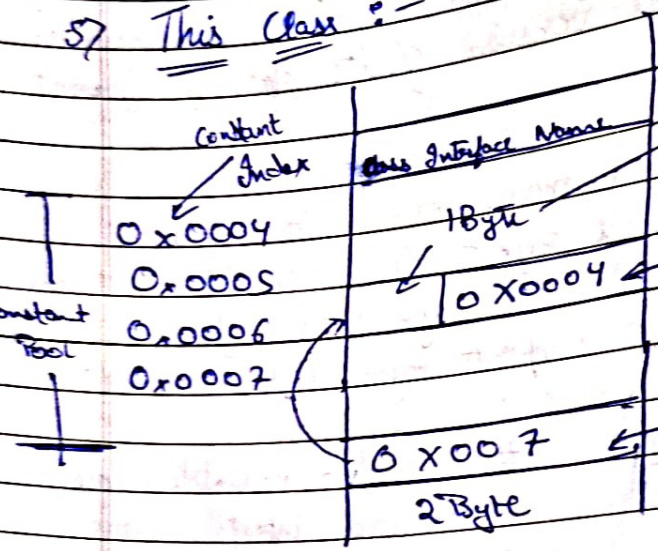
7>

4> Access Flag :-

Access flags follow the constant pool. It is a two byte entry that indicates whether the file defines a class or an interface. Whether it is public or abstract or final in case it is a class.

8>

5> This Class :-



6> Super Class

Next 2Byte after this class is of super class.

9>

Value of two bytes is a point on that points to constant pool which has entry for super class of the class.

Scanned with CamScanner

Interface :-

2 Byte	Index

Interface Section

Total No. of Interface Being

Index into the constant Pool for each interface implemented by class

Fields :-

A field is an instance or a class level variable of the class or interface. Field section contains only those fields that are designed by the class or an interface of the file and not those fields which are inherited from the super class or ~~super~~ super interface.

2 Byte	Variable length of array

Field Section

Total No. of Field

Reference to the constant pool entry.

Method :-

Method components host the methods that are explicitly defined by this class, not any other methods that may be inherited from super class

Scanned with CamScanner

2 Byte	Variable Length Array

— Method Section

No. of Methods in the class/ Interface

Each Method Structure {Method arguments list, its return type, the no. of Stack words required for the method is local variables etc)

10) Attributes :-

Attribute Section contains several attribute about the class file.

Eg :- One of the attribute is the source code attribute which reveals the name of the source file from which this class file was compiled.

1 Byte	attribute

— Attribute Method

No of Attributes

Scanned with CamScanner

lecture 8

Verification In Java :-

After the class loader in the JVM loads, the byte code of .class file to the machine the ByteCode is first checked for validity by the verifier and this process is called as verification.

[ClassLoader] —.class→ [Verifies] → check various properties

Some of checks that verifier performs :-

1) Uninitialized variables
2) Access rules for private data and methods are not violated.
3) Method calls match the object reference.
4) There are no operand stack overflows or underflows.
5) The arguments to all the Java Virtual Machine instructions are of valid types.
6) Ensuring that final classes are not subclassed and that final methods are not overridden
7) checking that all field references and method reference have valid names, classes and type

Scanned with CamScanner

Security Promises of the JVM :-

Here are some of the promises the JVM makes about program that have passes the verification algorithm

Every object is constructed exactly once before it is used.
Every object is an instance of exactly one class, which does not change through the life of the object
If a field or method is marked (Private) then the only code that ever accesses it is found with the class itself.
Fields and methods marked protected are used only by code that participated in the implementation of the class.
Every local variables is initialized before it is used.
Every field is initialized before it is used.
It is impossible to underflow or overflow the stack.
It is impossible to read or write past the end of an array or before the begining of the array.
It is impossible to change the length of an array once it has been created
Final methods can not be overridden and final classes can not be subclassed

Scanned with CamScanner

Page No.

Lecture 9

Garbage Collection:

Java garbage collection is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a JVM. When program run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory.

Ex:- To learn garbage collector mechanism in Java

Step 1

```
1> Class Student {
2> int a;
3> int b;
4> Public void SetData (int c, int d)
5> {
6> a=c;
7> b=d;
8> }
9> Public void ShowData ( ){
10> System.out.Println ("Value of a=" +a);
11> System.out.Println ("Value of b=" +b);
12>}
13>Public Static void main (String args[ ]){
14> Student S1 = new Student ( );
15> Student S2 = new Student ( );
```

Step 2

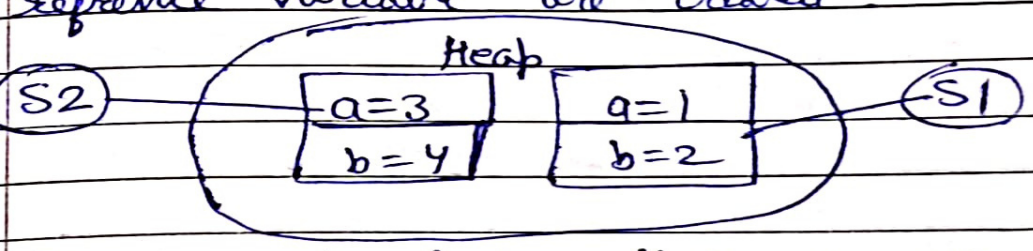
Step 3

Step 4

Step 5:

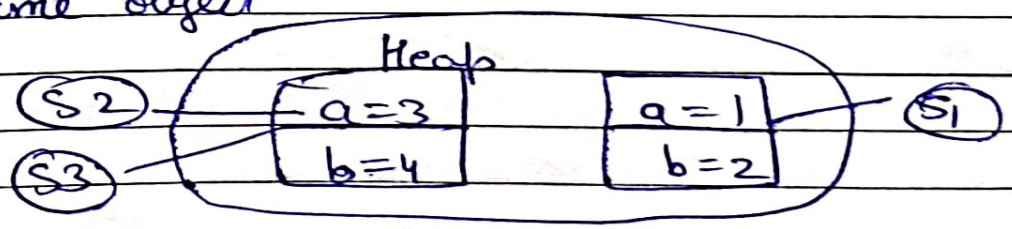
```
> S1.SetData(1,2);
> S2 . Set Data(3,4)
  S1. ShowData
> S2 . ShowData
> //Student S3;
> // S3 = S2;
> //S3 . ShowData ();
> // S2 = null;
> //S3 = ShowData( )
  //S3 = null;
> //S3 = ShowData( );
> } }
```

Save, compile and Run the code. Two objects & two refrence variable are created



Uncomment line // 20, 21, 22. Save compile and run the code.

Here two reference variables are pointing to the same object

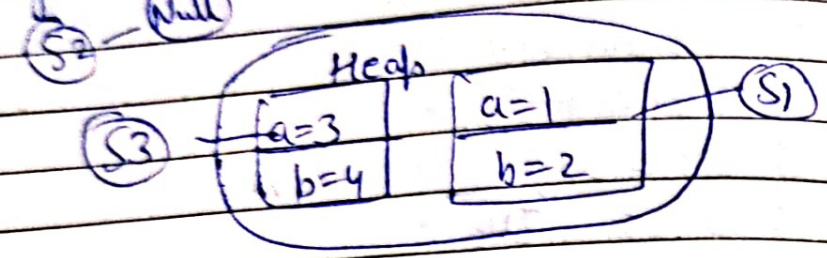


Uncomment line #/23, //24. Compile line & Run the code.

Scanned with CamScanner

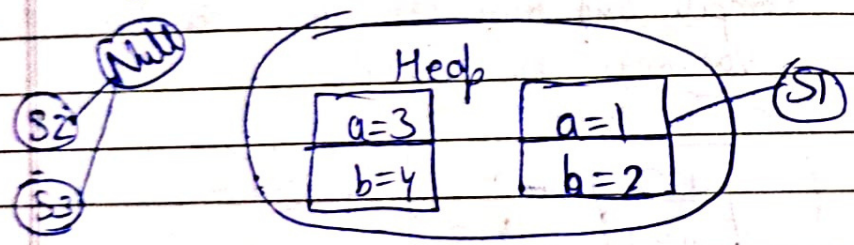
Page No.

Step 6 :- S2 become null but S3 is still pointing to the object and is not eligible for java garbage collection.



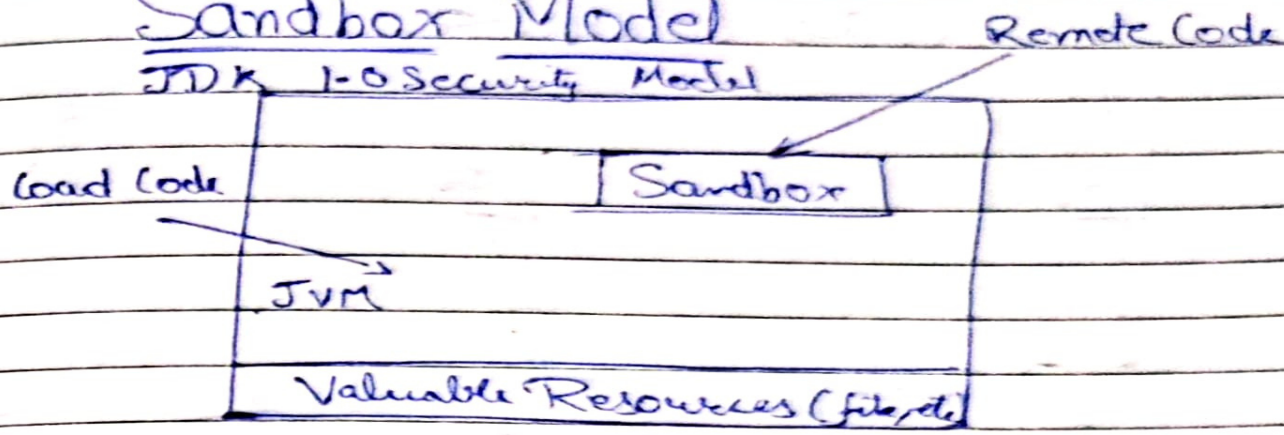
Step 7 :- Uncomment line //25 & //26 save compile & Run the code

Step 8 :- At this point, there are no references pointing to the object and becoming eligible for garbage collection. It will be removed from memory and there is no way of retrieving it back.



Sandbox Model

JDK 1.0 Security Model



The original security model provided by the Java Platform is known as sandbox model which existed in order to provide a very restricted environment.

In it local code is trusted, so that it is directly enter to the JVM where as Remote Code (Eg:- Applet) is not trusted, that's why, it is first pass through the sandbox.

It was developed through JDK and was generally adopted by application built with JDK 1.0 including java-enabled web browser.

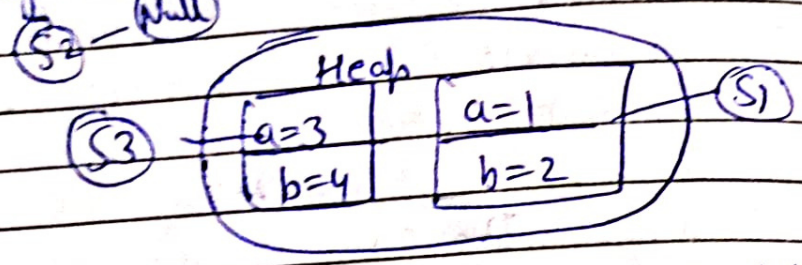
Security is provided through a no. of mechanism.

The language designed to a type safe and easy to use i.e. hop is that the program is such that likelihood of mistakes is less compared to using other programming languages such as C or C++.

This is Sandboxing. It is used to test unverified programs that may contain a virus or malicious code without allowing software to harm the host device.

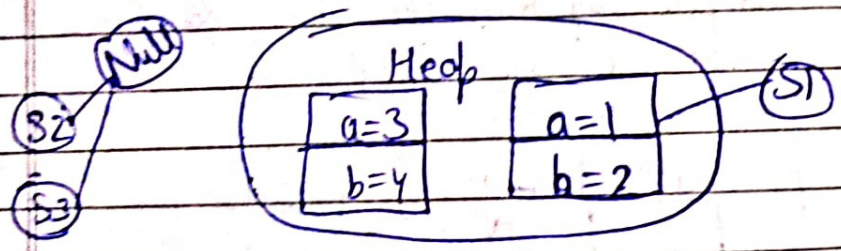
Scanned with CamScanner

Step 6 :- S2 become null but S3 is still pointing to the object and is not eligible for java garbage collector.



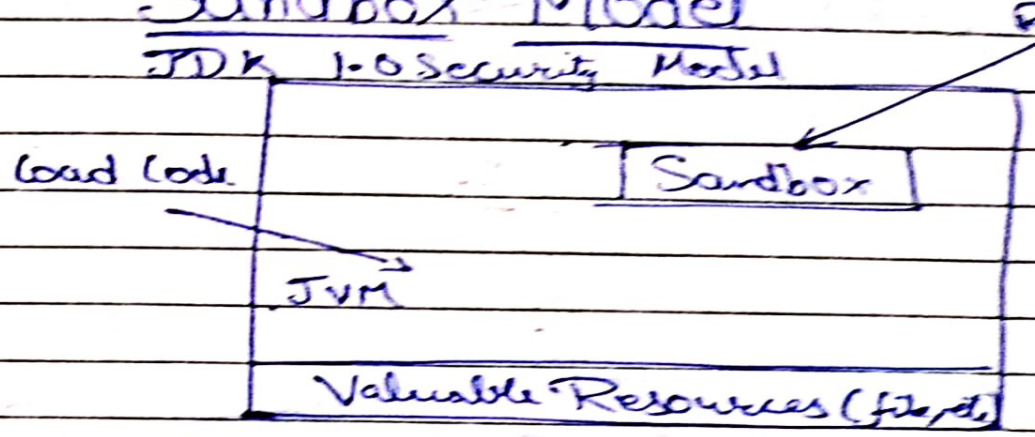
Step 7 :- Uncomment line //25 & //26 save compile & Run the code

Step 8 :- At this point, there are no references pointing to the object and becoming eligible for garbage collector. It will be removed from memory and there is no way of retrieving it back.



Sandbox Model

JDK 1.0 Security Model



The original security model provided by the Java Platform is known as sandbox model which existed in order to provide a very restricted environment.

In it local code is trusted, so that it is directly enter to the JVM where as Remote Code (Eg:- Applet) is not trusted, that's why, it is first pass through the sandbox.

It was developed through JDK and was generally adopted by application built with JDK 1.0 including java-enabled web browser.

Security is provided through a no. of mechanism

The language designed to a type safe and easy to use i.e. hop is that the program is such that likelihood of mistakes is less compared to using other programming languages such as C or C++.

This is Sandboxing. It is used to test unverified programs that may contain a virus or malicious code without allowing software to harm the host device.

Scanned with CamScanner

Java Security

Java is a secure language

Need of Java Security

① Safe from malevolent programs: Program shd not be allowed to harm user's computing environment. This includes Trojan horses as well as harmful programs that can replicate themselves.

② Non-intrusive:- Programs shd be prevented from discovering private information on the host computer.

③ Authenticated: Identity of parties involved in program shd be verified.

④ Encrypted: Data that the program sends and receives shd be encrypted.

⑤ Audited: Potentially sensitive operations shd always be logged.

⑥ Well defined: A well defined security specific^n would be followed.

⑦ Verified: Rules of operations shd be set & verified.

⑧ Well behaved: Programs shd be prevented from consuming too many system resources.

Security Promises of JVM

① Every object is constructed exactly once before it is used.

② Every object is an instance of exactly one class, which doesnot change through life of object

③ If a field or method is marked private,

then only code that ever accesses it is found within the class itself.

(4) Fields and methods marked protected are used only by code that participates in implementation of class

(5) Every local variable is initialized before it is used.

(6) Every field is initialized before it is used.

(7) It is impossible to underflow or overflow the stack.

(8) It is impossible to read or write past the end of array or before the beginning of array.

(9) It is impossible to change the length of array once created.

(10) Attempts to use null reference as receiver of method invocation or source of a field cause a NullPointerException to be thrown.

Security Architecture & Secure Policy

→ A security architecture is a way of organizing the software that makes up the Java Platform so that potentially harmful operations are isolated from unprivileged code but available to privilege code.

→ Core of Java platform security architecture under Java platforms 1.0 and 1.1 is the SecurityManager class. This class decides which pieces of code can perform certain operations and which cannot. Collectively, these decisions are called security policy.

The security policy is enforced by java platform classes, which check the SecurityManager before

proceeding with any operation under the control of security Manager.

→ On Java 2 platform, the core of security architecture is shifted to a class called <u>AccessController</u>.

The Java Sandbox Security Models:

→ The idea behind the model is when you allow a program to be hosted on your computer, you want to provide an environment where program can play (ie run) and limit the resources that can be accessed by applets.

→ Java sandbox is responsible for protecting a number of resources and it does so at a number of levels.

→ In computer security, sandbox is a security mechanism for separating running programs, usually in order to minimize system failures or software vulnerabilities from spreading.

→ In general, sandbox is an isolated computing environment in which a program or file can be executed without affecting the application in which it runs.

→ Sandboxes are used by software developers to test new programming code.

* Sandbox provides tightly controlled set of resources for guest programs to run in, eg limited space on disk and memory.

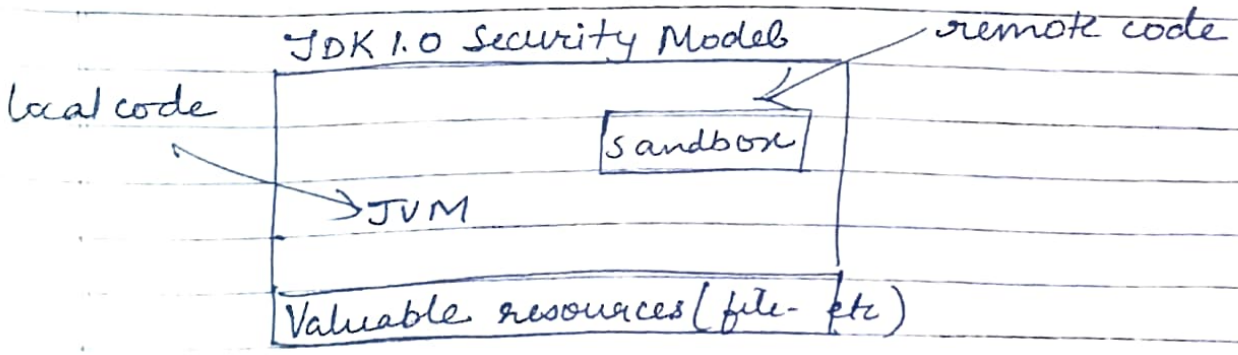
* In Java, sandbox is the program area and it has some set of rules that programmers need to follow when creating Java code (like applet) is sent as part of page.

* Sandbox restrictions provide strict limitations on which system resources the applet can request or access. The programmer must write code that "plays" only within the sandbox such as children are allowed to play within confined limits of space / place. The sandbox can be considered as small area within your computer where an applet's code can play freely but not allowed to play anywhere else.

Security (Sandbox)

→ It provides very restricted environment in which to run untrusted code obtained from open network.

→ Essence of sandbox :- local code is trusted to have full access to vital system resources (ie file ~~access~~ system) while downloaded remote code (an applet) is not trusted and can access only the limited resources provided inside sandbox.



→ Sandbox model is deployed through the JDK and generally adopted by applications built with JDK 1.0 including Java enabled web browsers.

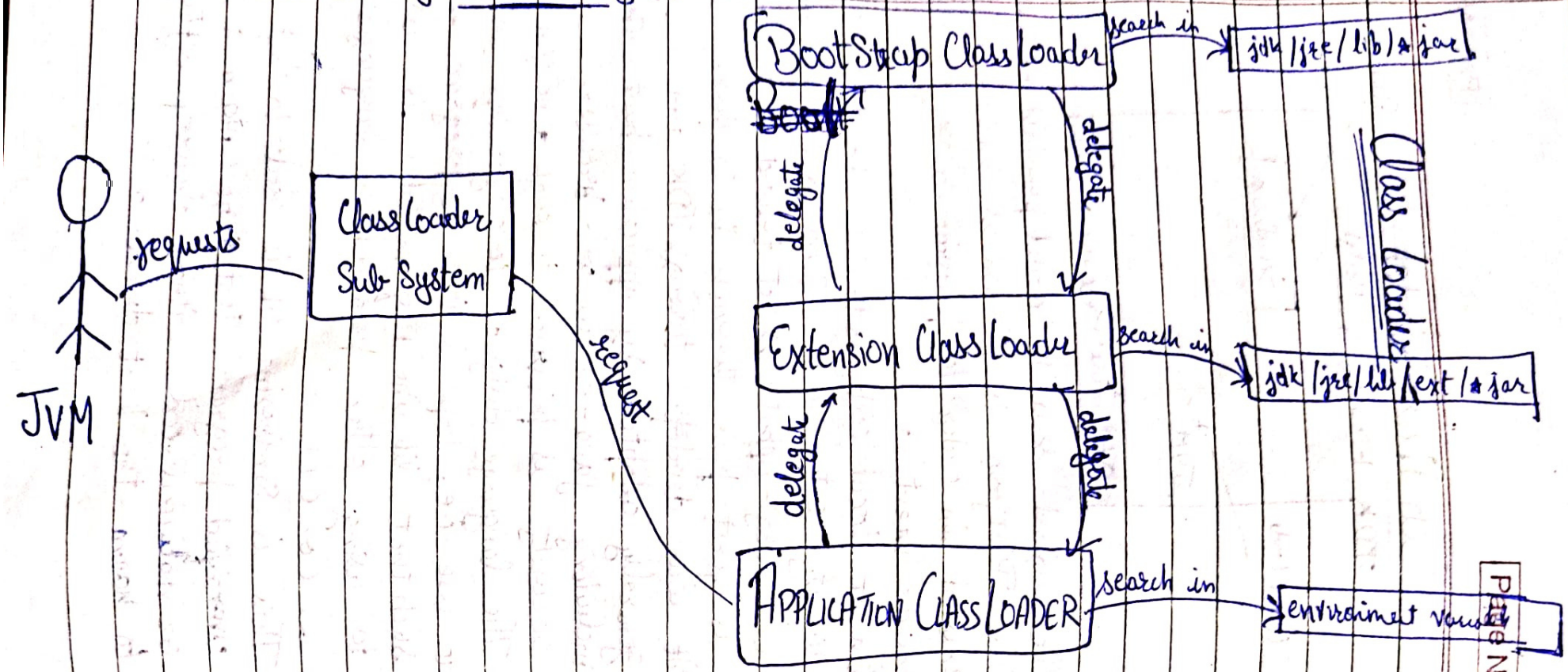
→ Overall security provided by language being disigned. Language features like automatic memory management, garbage collection and range checking on strings and arrays are example of how language helps programmer to write safe code.

→ Second, compilers & bytecode verifiers ensure only legitimate Java bytecodes are executed.

→ ClassLoader defines local name space, can be used to ensure untrusted applet can't interfere with running of other programs.

→ Finally, access to crucial system resources is mediated by JVM and is checked in advance by SecurityManager class that restricts the actions of piece of untrusted code to the bare minimum. [Sandboxing]

It is based on Delegation Hierarchy



Class Loader

Page No.

Scanned with CamScanner

How class loader works

Class loader follows delegation Hierarchy Principle

Whenever JVM come across a particular class, first it will check whether the corresponding class is already loaded or not

If it is already loaded in Method Area then JVM will use that loaded class.

If it is not already loaded then JVM requests class loader sub system to load that particular class, then class loader sub system handovers the request to Application class loader

Application C.L. delegates requests to Extension C.L ~~class~~ and E.C.L in turn delegates to BootStrap C.L.

BootStrap C.L. searches in BootStrap Class Path (JDK/JRE/Lib). If the specified class is available then it will be loaded. Otherwise, BootStrap C.L. delegates the request to E.C.L.

E.C.L. will searches in Extension class Path (JDK/JRE/Lib/Ext). If the specified class is available then it will be loaded. Otherwise, it delegates the Request to Application Class

Scanned with CamScanner

Page No.

8> Application C.L. will search in Application Class Path. If the specified class is already available then it will be loaded otherwise we will get RE: ClassNotFoundException.

17

Scanned with CamScanner

Types Of Variables in Java

- Primitive Variables – Used to represent primitive values
  Eg :- int x = 100;
- Reference Variables – used to ~~refr~~ refer objects
  Abc a1 = new Abc ()

Variables (Pos^n and declaration)

Instance Variable	Static Variable	Local Variable
↳ Values of variables is varied from object to object	Value is not varied from object to object	Created inside a Block or constructor, methods
↳ Cannot be accessed from Static area directly.	Known as Class-level Variables	Initialization is necessary befor using it

Scanned with CamScanner

Intance Variable :->

- Scope of object
- Should be declared within the class directly but ~~not~~ can be outside any block, method or constructor.

```
class Abc
{
  int i = 100;
  public static void main(String a[])
  {
    Abc a1 = new Abc();
    System.output.println (a1.i);
  }
```

- for two variables

```
Abc a1 = new Abc();
  a1.i = 200;
Abc a2 = new Abc();
  a2.i = 400;
```

- Initialization is not mandatory

```
int i;
Abc a1 = new Abc();
System.Output.println (a1.i)
```

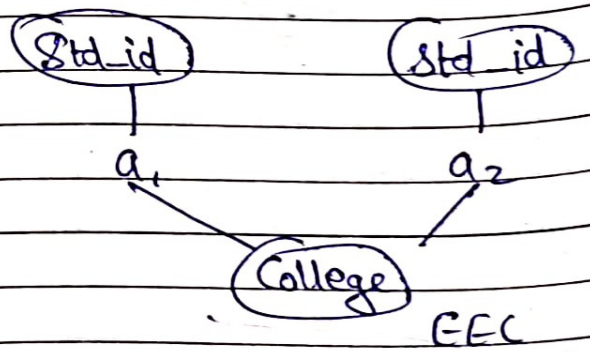
Output

0

Static Variable :->

Scope of class
Initialization is not mandatory

```
class Abc
{
  Static String college = "EEC"   } static
   int std_id;
  public Static void main (String a[ ])
   {
      Abc a1 = new Abc ();
      Abc a2 = new Abc ();
        a1.std_id = 1;
        a2.std_id = 2;
   }
}
```



If we want to change the subject of all subject we all have to do that change the college name.

Scanned with CamScanner

Page No.

Local Variables :->

Create inside a

↳ Block

↳ Method

↳ Constructor

↳ Initialization is mandatory.

```
class Abc
{
  public static void main (String [] a)
  {
    int i =10;
  }
  void main ()
  {
  }
```

Types

↳

↳

Wrapper Class

Used to convert primitive into object and vice-versa

Autoboxing :- Converting primitive to object.

Unboxing :- converting object to primitive.

Eight (8) classes of java language package are known as wrapper class in Java.

Primitive Type	Wrapper Class.
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Scanned with CamScanner

p.s.v.m = Public Static void main
S.o.p = System.output.println

Page No.

Examples

Primitive to Wrapper

```
class wrapper1
{
  p.s.v.m (String [] a)
  {
    int i=100;
  Integer j= Integer.ValueOf(i)
  Integer ab = j; //Autoboxing
    Sop ();
  }
}
```

Primitive

Wrapper to Primitive

```
Class wrapper 2
{
  p.s.v.m (String [] a)
{
Integer i= new Integer (10)
int ab = i.intvalue();
int j=i; //unboxing
S.op ()
}
}
```

Object

Loops

↳ A loop statement allows us to execute a statement or group of statements multiple times

Types of loops

- Entry Controlled loop
  i.e. cond^n is checked at entry time
  - while loop
  - for loop
- Exit Control loop
  i.e. cond^n is checked during exit.
  - do-while loop

Output

1
2
3
4
5

For Loop :-

```
class loopdemo
{
    public static void main (String a[ ])
    {
        for (i = 1; i<=5; i++)
        {
            System.out.println (i);
        }
    }
}
```

Output

```
1
2
3
4
5
```

While loop :-

```
class loopdemo
{
    public static void main (String a[ ])
    {
        int i = 1;
        while (i<=5)
        {System.out.println (i);
         i++;
        }
    }
}
```

int i = 11

Output

False

i.e. it first check the condition and then run

Scanned with CamScanner

do-while loop

```
class loopdemo
{
  public static void main (String a[])
  {
    int i = 11;
    do
    {
      System.out.println (i);
      i++;
    } while (i<=10);
  }
}
```

output

$\boxed{11}$

i.e. first it executes and then check the condition at the end.

Scanned with CamScanner

Super Key :-

Super Key is a reference variable that is used to refer immediate parent class object

Uses of Super key :-

Super () is used to invoke immediate Parent class constructors.

Super is used to invoke immediate Parent class method.

Super is used to refer immediate Parent class variable.

Example of Super keyword :-

```
class Parent
 { int i=10;
 }
class child extends Parent
  {
    int i=20;
    void m1()
    {
      System.out.println("Value of i in child"+i);
        S.O.P("Value of i in Parent"+ Super.i);
    }
  }
```

Scanned with CamScanner

```
class supertest
{
  public static void main(String a[])
  {
    Child c1= new Child();
    c1.m();
  }
```

Output
20
10

Example to call immediate Parent Class Constructor using Super keyword

```
class Parent
{
  Parent()
  { S.o.P. ("Hello");
  }
}

class child extends Parent
{
  Child()
  {
    Super();
  }
}
```

If super() is not written here, by default it will be super as it call parent class

```
class Supertest
{
  p.s.v.m(String a[])
  {
    child c1=new child();
  }
}
```

Output
Hello

Scanned with CamScanner

Method Overriding :-

Overriding means to override the functionality of an existing method.

```
class Parent
 {
   void m1 ( )
    S.O.P ("Parent");
 }
class child extends Parent
 {
   void m1 ( )
   S.O.P. ("Child");
 }
class overrideTest
 {
  p.s.v.m (String a[ ])
  {
   child C1 = new child ( );
    C1. m1( );
  }
 }
```

Case 2

If this block is empty then it calls parent class

Case 1

Case 1 Output:	Case 2 Output:
Child	Parent.

Scanned with CamScanner

Var-Arg Methods in Java

↳ Method with variable number of arguments.

↳ Declaration is like $\boxed{\text{method (int... x)}}$

Ex:-
```
class ABC
{
 public static void m1 (int... i)
  {
   S.O.P("ABC")
  }
 p.s.v. m(String a[])
  {
   m1();
   m1(1);
   m1(2, 3, 6);
   m1(34, 36);
  }
}
```

whether it is parametrized or unparametrized, it will always print ABC

output

```
ABC
ABC
ABC
ABC
```

Scanned with CamScanner

Important Points Regarding Var-Arg Methods

Var-arg parameter can be mix with normal parameter.

Ex:- m1(int x, String ... y) - Valid
m1(10)
m1(10, "ABC")

During mixing, Var-arg parameter should be last.

Ex:- m1(int ... x, String y) - Not Valid

There can be only one var-arg parameter.

Ex:- m1(int ... x, String ... y) - Not Valid

In General var-arg method will get last priority, if no other method matched then only var-arg method will be created.

Ex:- m1(int ... x) ← m1(10,20)
Last Priority
m1(int i, int j) ← First Priority

Scanned with CamScanner

Page No.

For Each Loops — Present ~~only~~ in 1.5 version or above

↳ It is the most convenient loop to retrieve the elements of Arrays and collections.

Example :-

~~int~~ int a[] = { 1, 2, 3, 4, 5 } ;

for-loop

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

Output

1
2
3
4
5

for-each loop

```
for (int x : a)
{
  S.O.P(x);
}
```

Output

1
2
3
4
5

Limitations :-

↳ Not a general purpose loop
↳ Only valid in arrays and collections.
↳ Can't retrieve particular set of values.

Scanned with CamScanner

Difference B/W length( ) vs length

Length( )	Length
It is a final method applicable only for string objects.	It is final variable applicable only for arrays.
Represents the no. of characters in string	Represents the size of array
Ex :- String S1 = "Hello" S.O.P (S1.length( ));	Ex :- int [ ] ar1 = new int[10]; SOP. (ar1.length)
O/P 5	O/P 10

Scanned with CamScanner

Page No.

Java Programming (Inheritance)

→ No Multiple Inheritance as it show ambiguity ~~ex~~ error

↳ Inheritance is the mechanism of deriving new class from old one

↳ Old class is know as Superclass

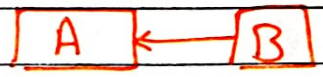
↳ New class is know as Subclass

Benefits :-

↳ Reusability of code
↳ Code Sharing
↳ Consistency in using an interface.

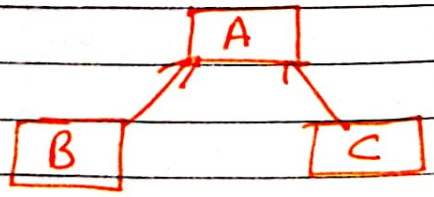
Types :-

<i> Single Inheritance



<ii> Multilevel Inheritance

<iii> Hierarchial Inheritance



Scanned with CamScanner

Final Keyword :-

- Final Variable :- its value cannot be changed during the scope of program.
- Final Method :- Cannot be overriden
- Final Class :- Cannot be inherited.

Example :-

```
class finalvar
{
    final int i=10;
    finalvar()
    {
        i= i+10;  ---> Error as final variable cannot be changed.
    }
}
class finaltest
{
    p.s.v.m (String a[])
    {
        finalvar f1 = new finalvar();
    }
}
```

Scanned with CamScanner

Another Example :-

```
class final method {
  {
    int i=10;
    final void m1()
      {
        S.O.P ("Final");
      }
  }
class child extends final method
  {
    void m1()
      {
        S.O.P.("Child");
      }
  }
class finaltest {
  {
    p.s.v.m (String a[ ])
      {
        Child c1= new child ();
        c1.m1();
      }
  }
```

It shows error as overriden is not possible as m1() is allready declared as final in parent class.

abstract

Scanned with CamScanner

Interface :-

{ Multiple Inheritance in Java }

An interface is a collection of abstract method.

A class implements an interface.

Contains behaviours that a class implements

Unless the class that implements the interface is abstract, all the methods of the interface needs to be defined in the class.

class C1 implements i1

Interface Name

class Name

Ex :-

```
interface i1
{
  ____
  ____
  ____
}
class c1 implements i
{

}
```

4 methods

Scanned with CamScanner

Page No.

How Interface Provide Multiple Inheritance in Java

```
interface i1                          interface i2
{                                     {
public void m1()                      public void m1()
  int i=10                              int i=100;
}                                     }

class interfacetest implements i1, i2
{
  public void m1()
  {
    S.O.P("Child class");
    S.O.P("i in i1"+i1.i);
    S.O.P("i in i2"+i2.i);
  }
p.s.v.m(String a[])
{
  interfacetest obj1= new interfacetest();
    obj1.m1();
  }
}
```

abstract

abstract

No ambiguity

```
Child class
i in i1 10
i in i2 100
```

Scanned with CamScanner

Page No.

Similarity b/w Interface and class

They can have any number of methods

Both ~~have~~ file save ends with .java extension.

Bytecode in .class file

Both appear in packages

Difference b/w Interface and class

Interface cannot be ~~instantane~~ instantiate

Interface has no constructor

All the methods of interface are abstract.

Interface ~~isses~~ implement where class uses extend

An Interface can used as multiple inheritance

Interface cannot contain instance fields.

Scanned with CamScanner

Abstract Class :-

- An abstract class is the one whose instances cannot be created.
- Any class that has atleast one abstract class has to be compulsory declared as an abstract class.
- Abstract class can contain both abstract and non abstract ~~class~~ methods.
- An ~~abst~~ Abstract Method has no body.
- Child class needs to override the def^n of all abstract methods.

Example Of Abstract Class / Methods :-

```
abstract class parent1
  {
    abstract void m1( );
    void m2( )
    {
      S.O.P.("Parent");
    }
  }
```

Scanned with CamScanner

Page No.

```
class Child1 extends parent1
    {
        void m1()
        {
            S.O.P. ("Child")
        }
    }
class abstracttest
    {
        p.s.v.m (String a[ ])
        {
            child1 C1 = new child1 ( );
            C1.m1();
            C1.m2();
        }
    }
```

Scanned with CamScanner

Page No.

Packages :-

Packages in Java are used to prevent naming conflicts, to control access etc.

Or

A package can be defined as a grouping of related types (classes, interfaces etc) providing access protection and name space management.

Examples of Existing packages in Java

↳ java.lang - Bundles fundamental classes

↳ java.io — class i/p, o/p etc.

Imp

Same class
|
Same Pack. Non Sub
Diff Pack Sub
Diff Pack Non Sub

Scanned with CamScanner

Example to Create Package

Package Name

```
package Hi_package;

public class C1
{
  int a = 5;
  public int b=10;
  private int c;
  protected int d;
  public void m1()
  {
    S.O.P (a+b+c+d);
  }
}
```

Importing Package in other class

For all classes

```
import Hi_package.*;

class packagetest
{
  p.s.v.m (String a[])
  {
    C1 obj = new C1();
    C1 - m1();
  }
}
```

O/P

15

Access Modifier In ~~Class~~ Packages

Private	Default	Protected	Public
Yes	Yes	Yes	Yes
No	Yes	Yes	Yes
No	Yes	Yes	Yes
No	No	Yes	Yes
No	No	No	Yes

Can protected access modifier is accessible ~~out~~ outside the package. → Yes but only in inherited class.

Scanned with CamScanner

Exceptional Handling :-

Any exception is a problem that arises during the execution of a program.

↳ User entered invalid data
↳ file Not found

Exceptional Handling is a task to maintain Normal flow of the Program.
[Exceptional Handling does not repairing an exception, it is rather defining alternative way to continue rest of the program normally]

Handling Exception In Java

17. try-catch block :-

Risky code with in the try block and corresponding Handling code inside the catch book,

Without try-catch block

```
class exept1
{
    p.s.v.m (String a[]) It will not run
    {
        int i = 100/0 ;   Error/Exception (Divide by zero)
        S.O.P. ("Value of i" +i );
    }
}
```

Scanned with CamScanner

```
class except1
{
    p.s.v.m (String a[])
    {
        try
        {
            int i= 100/0;
        }
        Catch (Arithmetic Exception e)
        {
            S.O.P (e);
        }
        S.O.P ("After try-catch").
    }
}
```

It will run and it will give o/p which is correct

Output

```
java.lang.Arithmatic Exception : / by zero
After try catch
```

Note:- With in the try block if any where an exception occurs, the rest of the try block won't be executed eventhough we handled that exception.

Scanned with CamScanner

Page No.

Method To Print Exception Information

1> Print Stack Trace ( ) } By default

e.printStackTrace();

Name of Exception :- description
Stack Trace

2> toString ( )

e.toString ();

Name : description
X Stack Trace

3> getMessage( )

~~e.getM~~ e.getMessage ( );

description
xName xStack Trace

try { }
catch { }
catch { }
Order is imp

as catch / allow to the error contains

this not

Try with Multiple Catch Blocks :-

If try with multiple catch blocks is present than the order of catch blocks is very important and it should be from child to parent

```
class ex1
{
    p.s.v.m (String a[ ])
    {
        try
        {
            int i= 100/0;
        }
        catch (Exception e)
        {
            S.O.P (e);
        }
        catch (ArithmeticException e1)
        {
            S.O.P(e1);
        }
    }
}
```

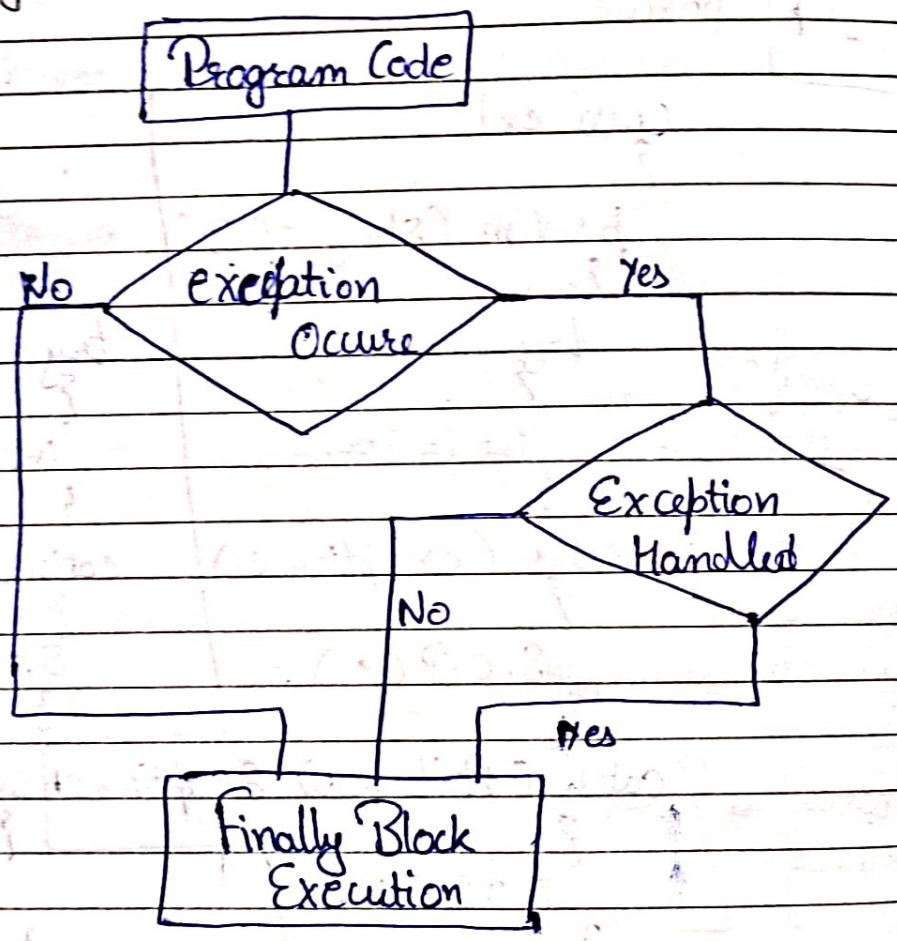
error will not forward as it the exception is

```
try
{
    int i= 100/0;
}
catch(ArithmeticException e)
{
    S.O.P. (e);
}
catch(Exception e1)
{
    S.O.P (e1);
}
```

Scanned with CamScanner

27 **Finally Block** :- Code in finally block always execute whether exception has occured or not.

Allows you to run any clean-up type statements that you want to execute.



Scanned with CamScanner

Example of Finally Block :-

```
class ex1
{
    p.s.v.m (String a[ ])
    {
      try
      {
          int i=10/0;
      }
      Catch (Aritmetic Exception e)
      {
          S.O.P (e);
      }
      finally
      {
          S.O.P ("Finally");
      }
    }
```

Output

```
java.lang.Aritmetic Exception : / by zero
Finally
```

Scanned with CamScanner

3> Throw Keyword :-

It is used to explicitly throw an exception.

throw new Arithmetic Exception ("___");

Example :-

```
class throw
{
  Static void valid (int i)
   {
      if (i<18)
       {
         throw new Arithmetic Exception ("Not Valid")
       }
      else
       {
         S.O.P ("Welcom");
       }
   }
  p.s.v.m (String a[])
   {
      try
       {
         Valid (16);
       }
      Catch (Arithmetic Exception e)
       { S.O.P (e); }
   }
}
```

Output

Not Valid

Scanned with CamScanner

Throws Keyword :-

Used to declare the exception, it provide information to the programmer that there may occur an exception. So during call of that method, programmer must use exception handling mechanism.

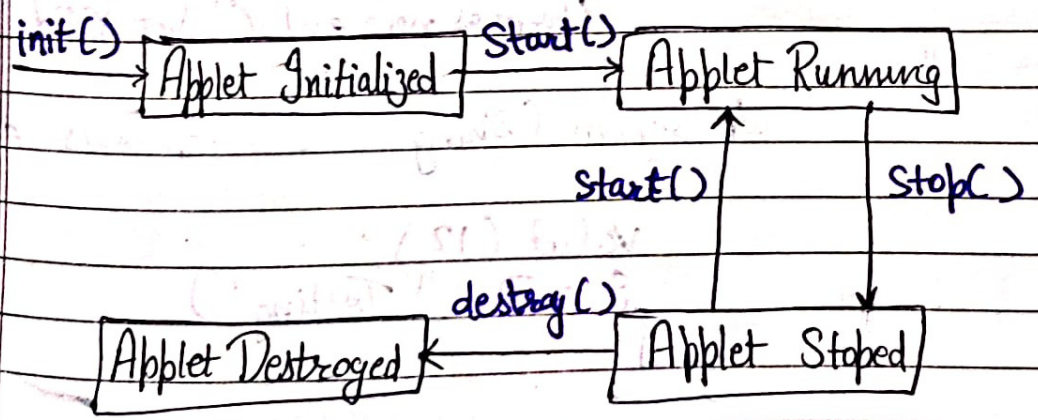
```
class ex1 extends Exception
  {
    ex1 (String s)
      {
        Super (s);
      }
  }

class ex2
  {
    static void valid (int i) throws ex1
      {
        if (i< 18)
        throws new ex1 ("Not Valid");
      }
    p.s.v.m (String a[]) throws ex1
      {
        valid (12);
        S.O.P ("Testing")
      }
  }
```

Scanned with CamScanner

APPLET PROGRAMMING

- An applet is Java Program that runs in a Web Browser.
- It extends the java.applet.Applet class.
- No main() method
- Applet are designed to be embedded within a HTML page
- JVM is required to view an applet.
- Applet have strict security rules that are enforced by the web browser.

Life Cycle Of Applet



Scanned with CamScanner

How to create first Applet

ode

```
import java.applet.*;
import java.awt.*;
public class app_image extends Applet
{
    public void paint (Graphics g)
    {
        g.drawString ("My Applet", 100, 100);
    }
}
```

String Coordinates

As this java code is save as app_image.class

TML Code

```
<html>
<title> Image Example <title>
<applet code = "app_image.class" width=400 height=400>
</applet>
</html>
```

Save this file such as a.html

Open this file and the output will be

My Applet

Scanned with CamScanner

Passing Parameters to Applets

<PARAM> tag is used to pass the parameter value from HTML file to Applet Code.

HTML file Code

```
<html>
<title> Test 1 </title>
<applet Code="Applet 2.class" width=400 height=400>
<PARAM Name="name" value="Hello Java">
</applet>
</html>
```

Java file Code

```
import java.applet*;
import java.awt*;
public class Applet 2 extends Applet
{
 String str=null;
 public void init()
  {
    String str =null;
    public void init ()
     {
       str = getParameter ("name")
     }
```

Scanned with CamScanner

```
public void paint (Graphics g)
    {
        g.drawString (Str, 100, 100)
    }
}
```

Getting I/P from User In Applet

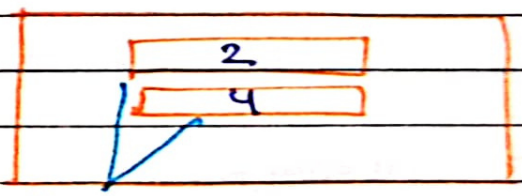
// Java Code

```
import java.applet.*;
import java.awt;
public class input1 extends Applet
{
  TextField t1, t2;
  public void init ()
    {
      t1 = new TextField (5);
      t2 = new TextField (5);
      add (t1)
      add (t2)
    }
public void paint (Graphics g)
    {
      int x = 0;
  x = Integer.parseInt (t1.getText());
t2.SetText (String.valueOf (x*x));
}
```

// HTML Code

```
<html>
  <title> "Check Input" </title>
<applet code = "input1.class"
      width=400  height=400>
</applet>
</html>
```

Output



Textfield

Let suppose input is 2
the output will be 4

Scanned with CamScanner

Benefits of Multithreading:-

Enable Programmer to do multiple task at one time

Programmer — Update User (t_1)
Programmer — Print User Info (t_2)

Programmers can divide a long program into threads and execute them in parallel which will eventually increase speed of program execution

Scanned with CamScanner

Strings

java.lang.String (import)

String as a object that represents sequence of char values.

It is immutable i.e. it cannot change once it is initialized.

Methods to create String :-

- String literal
- New keyword

String S = "Hello"

String S1 = New String ("Hello")

String S1 = "Hello"

Memory Efficient

Scanned with CamScanner

Q Why String objects are immutable ?

Ans Once string object is created its data or state can't be changed but a new string object is created.

Ex :-

```
class teststring
{
    p.s.v.m (String [] ar)
    {
        String s = "EEC";
        s=s.concat("Classes"); -> EEC Classes
        S.O.P(s);
    }
}
```

EEC

EEC Classes

Output

EEC

Output

EEC Classes

Scanned with CamScanner

Q Why String objects are immutable ?

Ans Once string object is created its data or state can't be changed but a new String object is created.

Ex :-

```
class teststring
{
    p.s.v.m (String [] ar)
    {
        String s = "EEC";
        s=s.concat("Classes"); -> EEC Classes
        S.O.P(s);
    }
}
```

EEC

EEC Classes

Output

EEC

Output

EEC Classes

Scanned with CamScanner

String Comparison

String S1 = "EEC";
String S2 = "EEC";
String S3 = new String("EEC");
String S4 = "classes";

equals() method

Compares original content of the string

Compares values of String for equality

O/P

~~String~~ S.O.P(S1.equals(S2)) = true
S.O.P(S1.equals(S3)) = true
S.O.P(S1.equals(S4)) = false.

= = operator

Compares references not values.

S1 = = S2 — true
S1 = = S3 — false

Compare To() method.
Compares value ~~the~~ lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string

S1 = = S2 -> 0
S1 > S2 -> +ve
S1 < S2 -> -ve

Scanned with CamScanner

String Buffer class

Used to create mutable (modifiable) string.

Methods :-

StringBuffer Sb = new StringBuffer("YouTube");
0123456

1) Append ()

Sb.append ("classes");
S.O.P (Sb);
YouTube classes

2) Insert ()

Sb.insert (3, "fire");
YouFireTube

3) Replace ()

Starting index End index

Sb.replace (1, 2, "Shame")
YShameTube

4) delete ()

Sb.delete (1, 3)
Si S.I.

5) ~~Sb~~ Reverse ()

Sb.reverse (); Pallindrome

1)
2)
3)
4)

Scanned with CamScanner

Difference B/w String and String Buffer Classes :-

String	String Buffer
Immutable	Mutable
Slow	Fast
Consumes more memory when you concat too many strings.	Consumes less memory.
Overrides equals( ) method of object class	Does not override equals() method.

Scanned with CamScanner