Page No.

Page No.

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 programs 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

Page No.

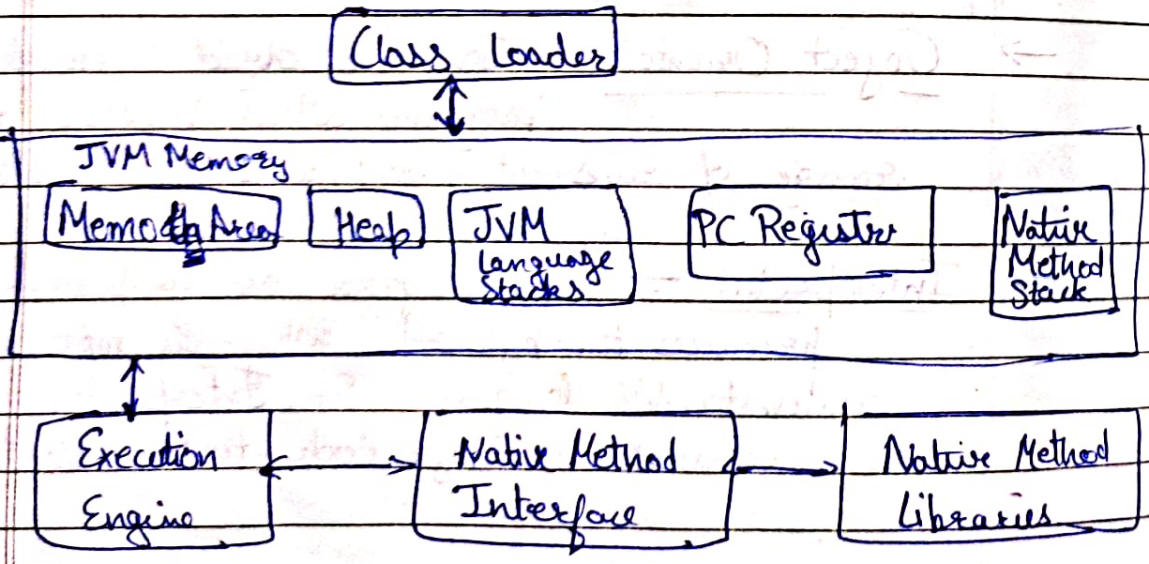
→ Robust and Secure :- Java programming is more reliable. At the time of execution, it shows all the error. Java also provides the lot of security as the computer will be attacked by the external program.

→ 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 :-



Page No.

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 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 registes.

Native Method Stacks :- It holds the instruction of native code depends on the native library. It is 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.

⇒ 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 immediate parent class.

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

↳ Modifier, variable and method information etc.

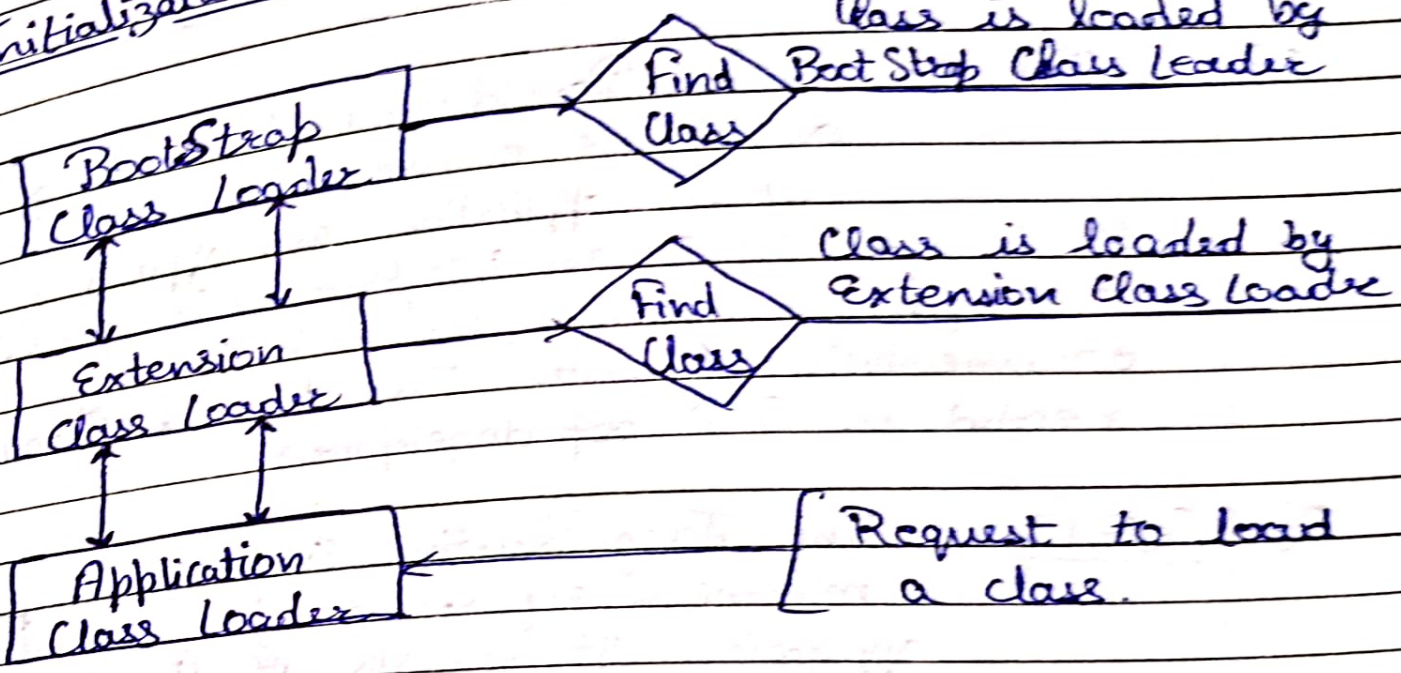
2) Linking :- Performs verification, preperation and resolution.

↳ 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.

Initialization :-



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 ~~FROM~~ Sun and other companies

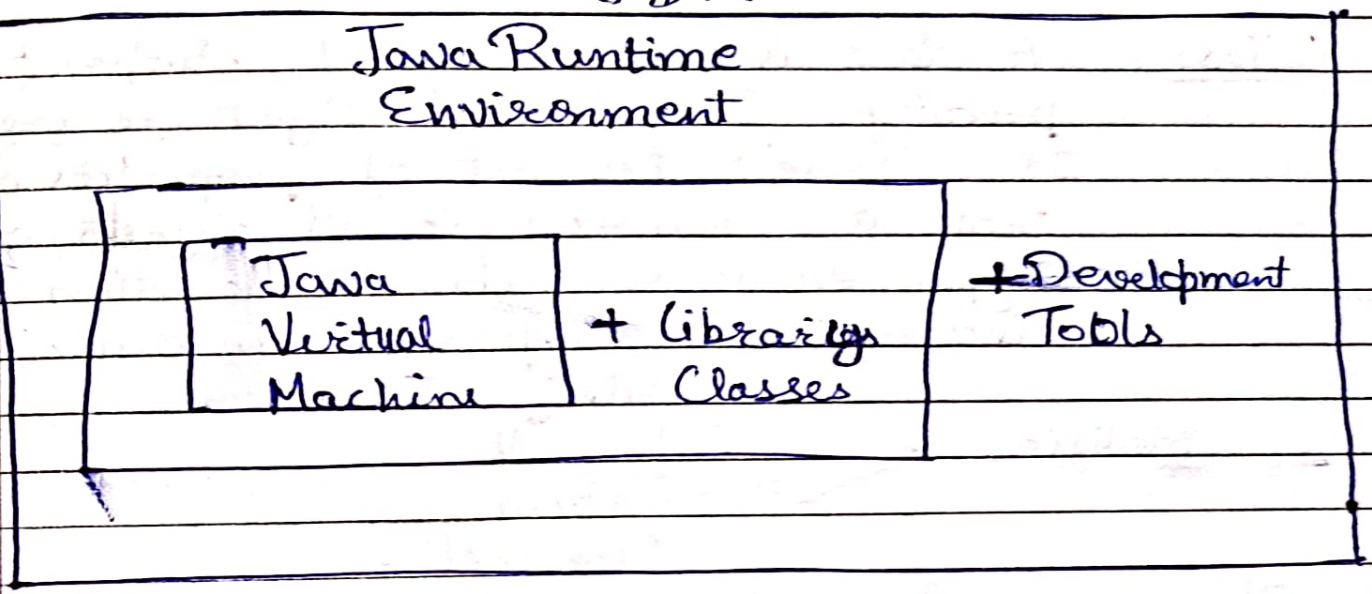
↳ A implementation is a computer program that meets the requirements of the JVM specification.

↳ 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.

Page No.

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.

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 behaviours 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.

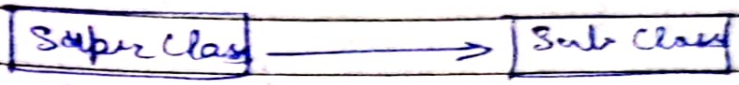
4) **Message Passing** : {Communication b/w Processes}

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.

Page No.

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

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 functions 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)        // Method Name; Method with two integer parameters
    {
        return a * b;
    }
    Static double Multiply (double a, double b)   // Method with same name but two double parameter
    {
        return a * b;
    }
}
```

Page No.

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.

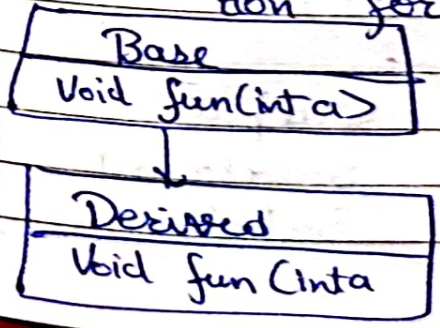
In Java Only '+' operator is overloaded

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.



Page No.

Ex:-

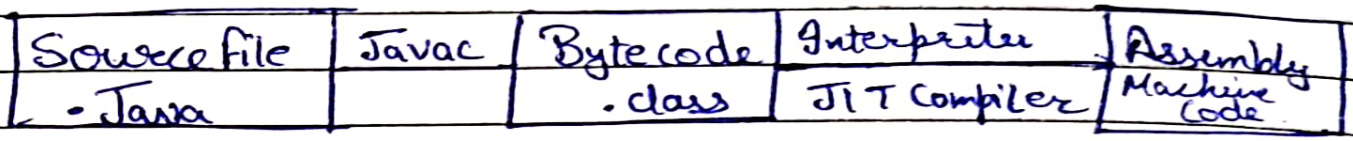
```
class Parent
{
  void Print()
  {
    System.out.println("Parent class");
  }
class Subclass1 extends Parent
  {
    void Print()
    System.out.println("Subclass1");
  }
class Subclass2 extends Parent
  {
    void Print()
    System.out.println("Subclass2");
  }
class Testpolymorphism
  {
  public static void main(String[] args)
    {
      Parent a;
      a= new Subclass1();
      a.Print()
      a=new Subclass2();
      a.Print;
    }
}
```

Output
Subclass1
Subclass2

Page No.

JVM Class file format :-

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



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
  {
  }
```

Page No.

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	

17

2>

3>

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 OXCAFEBABE. 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 preceded 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

Page No.

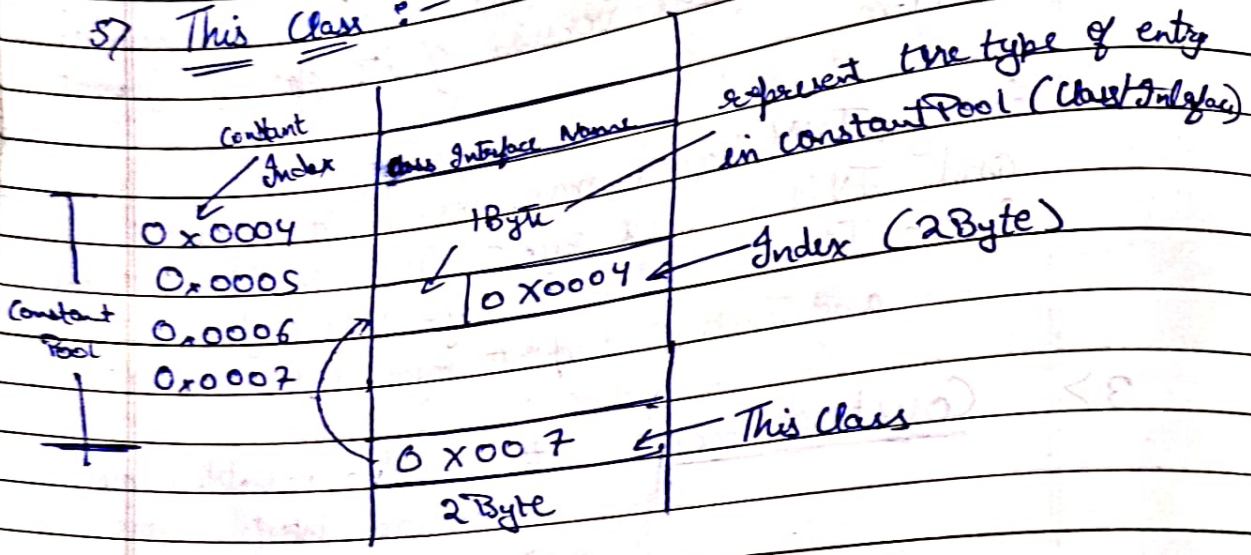
7>

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 :-



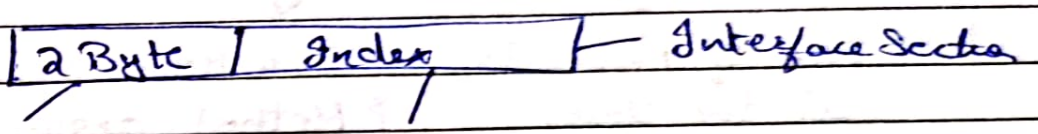
6> Super Class

9>

Next 2Byte after this class is of super class.
Value of two bytes is a point on that points to constant pool which has entry for super class of the class.

Page No.

Interface :-

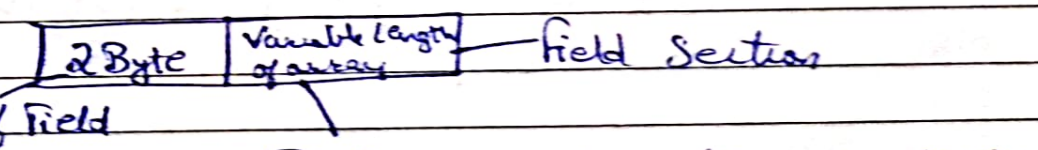


Total No. of Interface Being

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 ~~Sup~~ super interface.

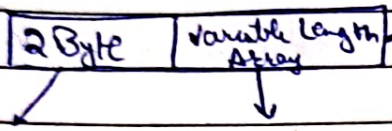


Total No. of field

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



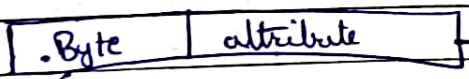
No. of Methods in the class/ Interface

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.



No of Attributes

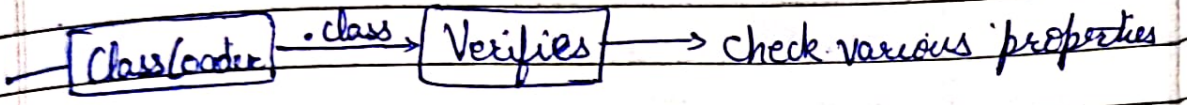
Page No.

Page No.

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.



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 references have valid names, classes and type

Page No.

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 beginning 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

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

Step 2

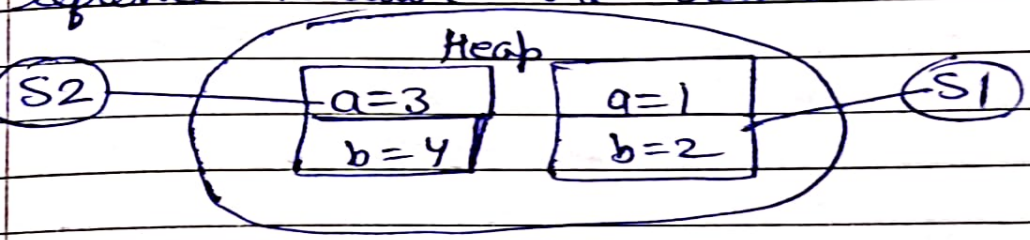
Step 3

Step 4

Step 5:-

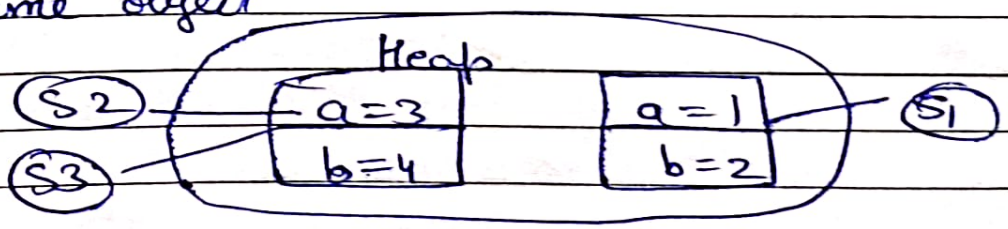
```
S1.SetData(1,2);
S2.SetData(3,4)
S1.ShowData
S2.ShowData
//Student S3;
// S3 = S2;
//S3.ShowData();
// S2 = null;
//S3 = ShowData()
//S3 = null;
//S3 = ShowData();
}}
```

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



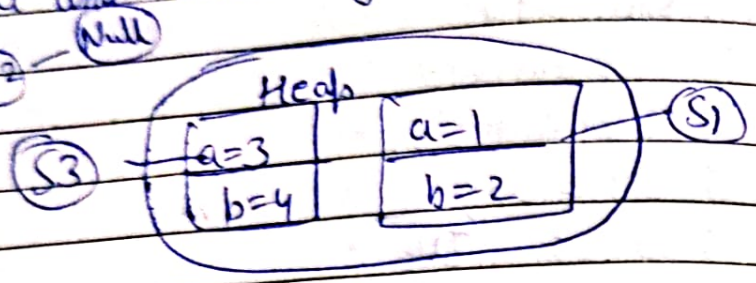
Uncomment line //20, 21, 22..... Save compile and run the code.

Here two reference variables are pointing to the same object



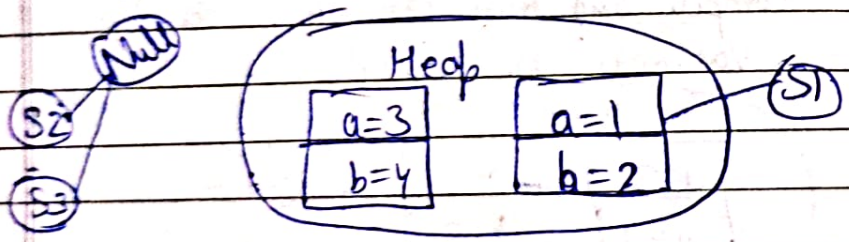
Uncomment line #/23, //24. Compile line & Run the code.

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 //2 S & //6 Save Compile & Run the code

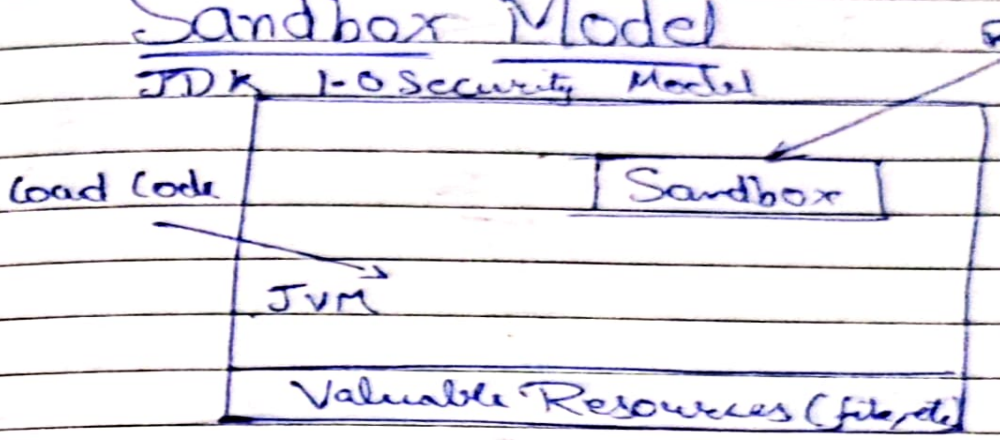
Step 8 :- At this point, there are no references pointing to the object and becoming eligible for garbage collection. It will be removed from memory and there is no way of retrieving it back.



Page No.

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.

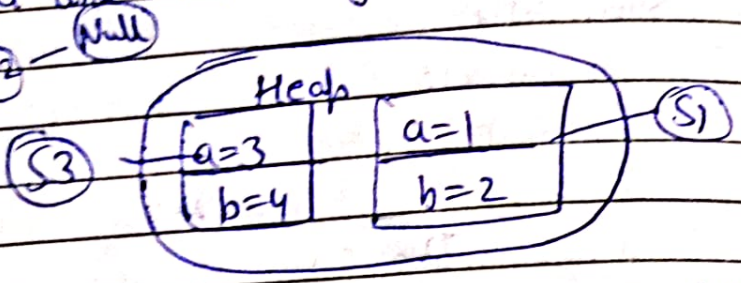
It was developed through JDK and was generally adopted by application built with JDK 1.0 including java-enabled web browser.

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.

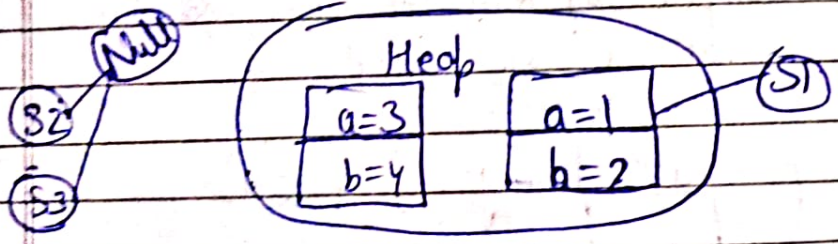
Step6 :- S2 become null but S3 is still pointing to the object and is not eligible for java garbage collector.



Step7 :- Uncomment line 1/25 & 26 save compile & Run the code

Step8 :- 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.

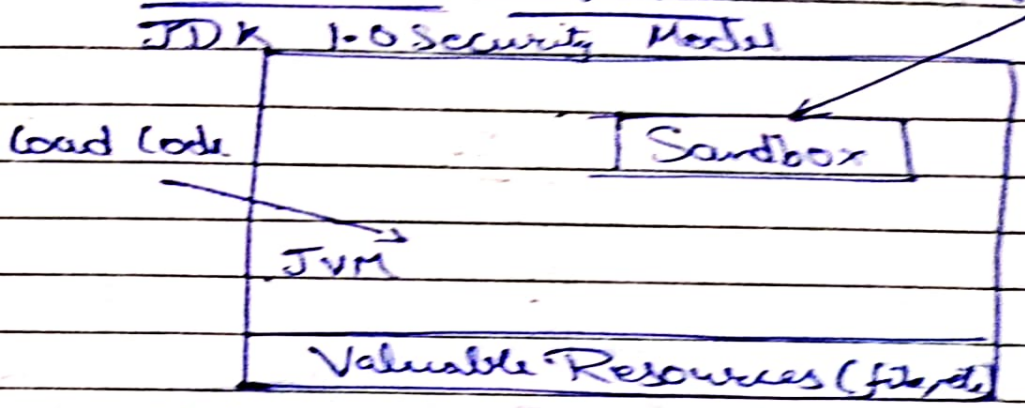
Step



Page No.:

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.

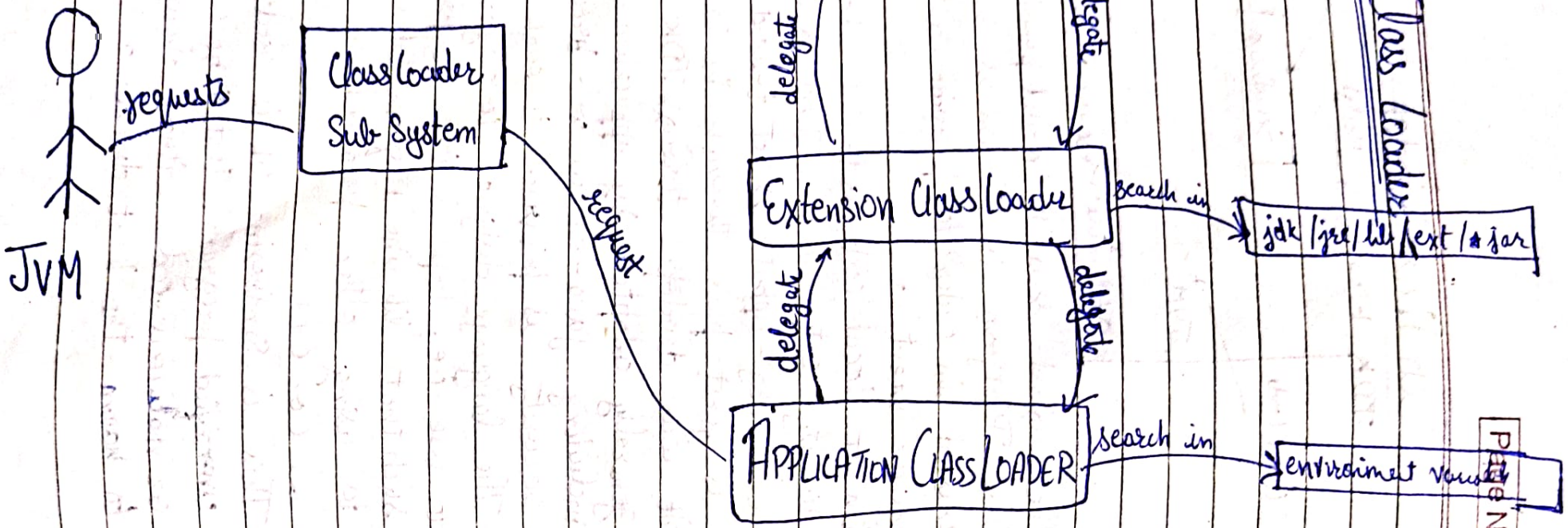
It was developed through JDK and was generally adopted by application built with JDK 1.0 including java-enabled web browser.

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.

It is based on Delegation Hierarchy



Page No.

Page No.

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 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

Page No.

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

17

12

For Practise go to www.w3schools.com

JAVA (Apnikaksha)

Page No.

Installation (Lecture 2)

Download JDK

- MAC OS (JDK-Version 11)
- WINDOWS
  - 32 Bit (JDK-Version-8)
  - 64 Bit (Version-11)

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

FileName should be this and ends with .java

Command Prompt

First come to desktop
then type dir
It helps to show all the files present in desktop

Page No.

Types Of Variables in Java

- Primitive Variables – Used to represent primitive values
  Eg:- int x = 100;
- Reference Variables – used to 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

Page No.

Intance Variable :->

↳ Scope of object
↳ Should be declared within the class directly but Can be not 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)

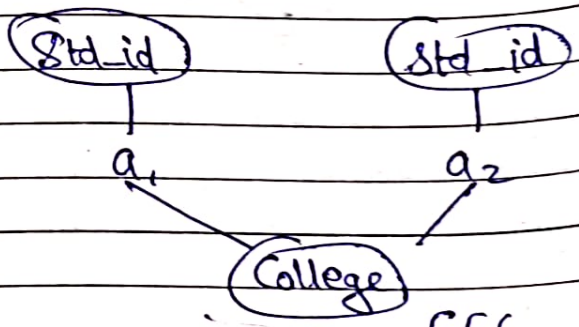
Output
0

Page No.

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.

Page No.

Local Variables :->

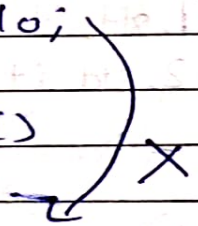
Create inside a

- Block
- Method
- Constructor

- Initialization is mandatory.

Types
-
-

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



Page No.

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

p.s.v.m = Public Static void main

S.o.p = System.out.println

Page No.

Examples

Primitive to Wrapper

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

Wrapper to Primitive

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

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

Page No.

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

Page No.

do-while loop

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

Output

```
11
```

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

Page No.

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);
    }
}
```

Page No.

```
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

Page No.

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 :

Child

Case 2
Output :

Parent

Page No.

Var-Arg Methods in Java

↳ Method with variable number of arguements

↳ Declaration is like `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
```

Page No.

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) ← Last Priority ← m1(10,20)
m1(int i, int j) ← First Priority

For Each Loops

Page No.

↳ 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.

Page No.

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()); O/P 5	Ex :- int[] ar = new int[10]; SOP.(ar1.length) O/P 10

Page No.

Java Programming (Inheritance)

↳ No Multiple Inheritance as it show ambiguity or error

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

↳ Old class is know as Superclass

↳ New class is know as Subclass

Benefits :-

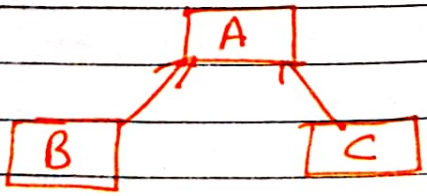
↳ Reusability of code
↳ Code Sharing
↳ Consistency in using an interface.

Types :-

<i> Single Inheritance

<ii> Multilevel Inheritance

<iii> Hierarchial Inheritance



Page No.

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

Page No.

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

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
{  ====        4methods
   ====
}
class C1 implements i
{

}
```

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
```

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 ~~instantiate~~ instantiate

Interface has no constructor

All the methods of interface are abstract.

Interface uses implement where class uses extend

An Interface can used as multiple inheritance.

Interface cannot contain instance fields.

Page No.

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");
    }
  }
```

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();
        }
    }
```

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.

Example of Existing packages in Java

↳ java.lang - Bundles fundamental classes

↳ java.io — class i/p, o/p etc.

Same class

Same Pack. Non Sub

Diff Pack Sub.

Diff Pack Non Sub

Page No.

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.

Page No.

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

1. try-catch block :-

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

Without try-catch block

```
class excepth
{
    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 );
    }
}
```

Page No.

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

It will run and it will give o/p which is relevant for us

Output

```
java.lang.Arithmatic Exaption : / 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.

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
x Name x Stack Trace

try { }
catch { }
catch { }
}

Order is imp

as * Catch allow to the error contains

this not

Page No.

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);
    }
  }
}
```

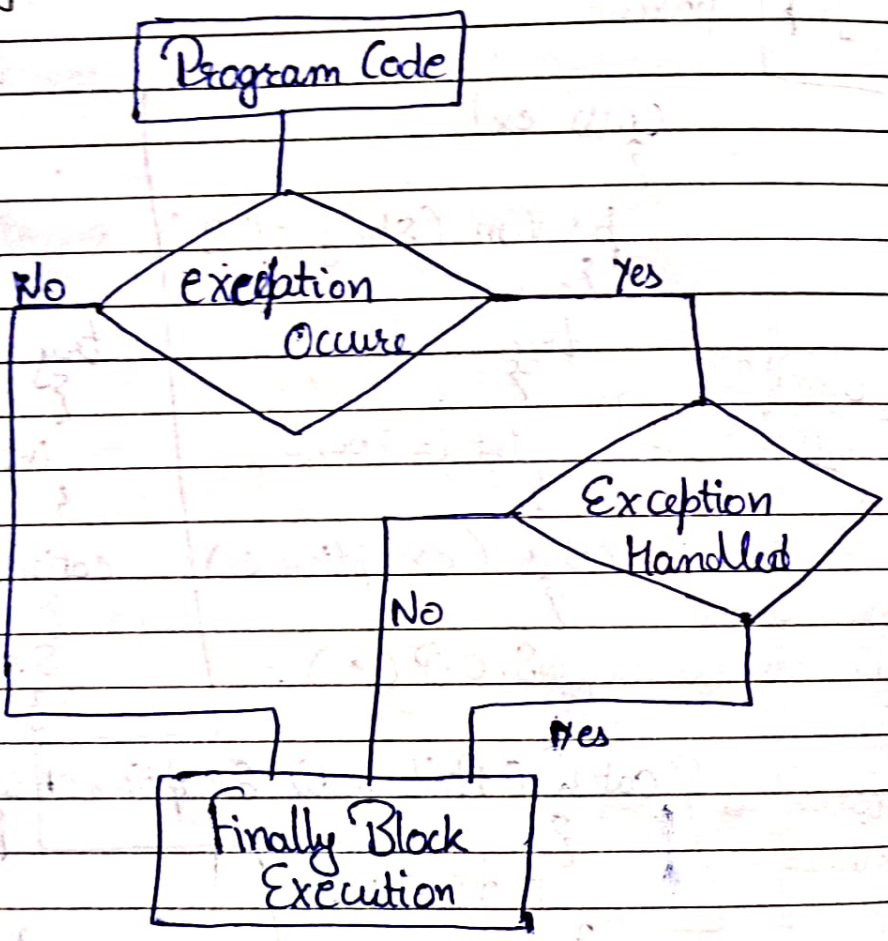
error will not forward as it the exception is

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

Page No.

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.



Page No.

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
```

Page No.

3> Throw Keyword :-

4>

It is used to explicitly throw an exception.

```
throw new Arithmetic Exception ("  ");
```

Example :-

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

Output

Not Valid

Page No.

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")
      }
  }
```

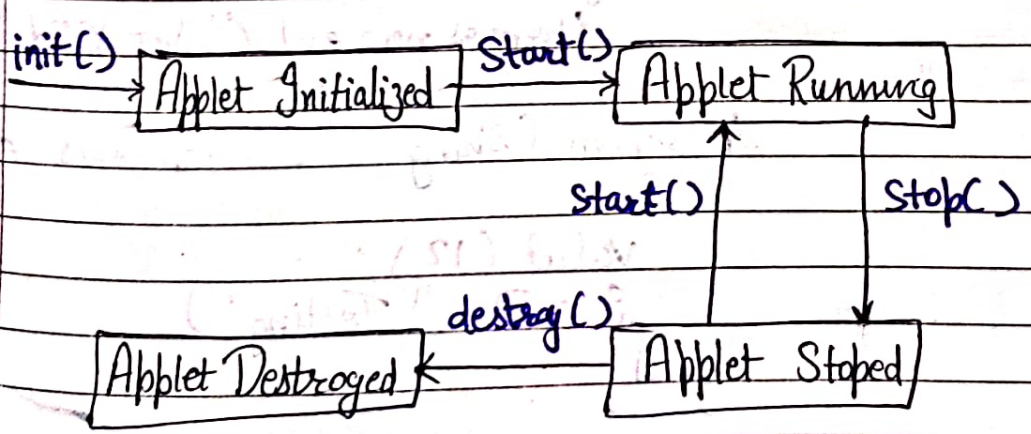
Page No.

APPLET PROGRAMMING

Java

- 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



Page No.

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

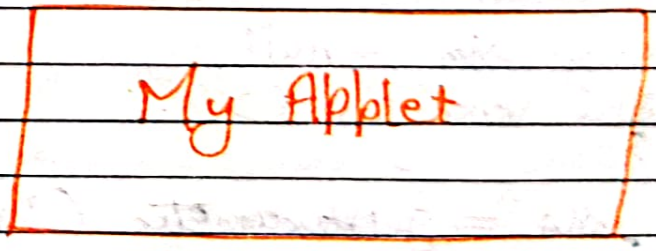
As this java code is save as app_image.class

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



Page No.

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 I </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")
       }
```

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

Getting I/P from User In Applet

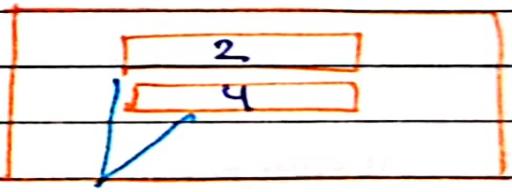
// Java Code

```
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



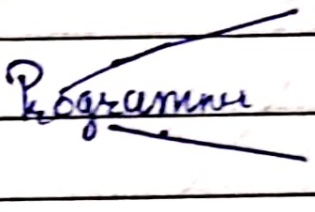
Textfield

Let suppose input is 2
the output will be 4

Page No.

Benefits of Multithreading :-

Enable Programmer to do multiple task at one time



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

Improved performance and concurrency.

Simultaneous access to applications.

Page No.

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

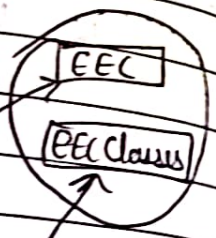
Page No.

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"); → EECClasses
        S.O.P(S);
    }
}
```



Output

EEC

Output

EEC Classes

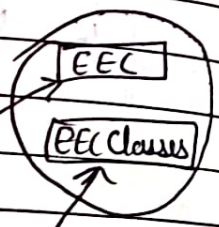
Page No.

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);
  }
}
```



Output

EEC

Output

EEC Classes

17

2>

3>

Page No.

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

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

O/P

== operator

Compares references not values.

S1 == S2 — true

S1 == S3 — false

Compare To() method.

Compares value 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

Page No.

String Buffer class

Used to create mutable (modifiable) string.

Methods :-

StringBuffer Sb = new StringBuffer ("YouTube");
0 1 2 3 4 5 6

1) Append ()

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

2) Insert ()

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

3) Replace ()

Sb.replace (1, 2, "Shame")
Starting index, Ending index
YShameTube

4) delete ()

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

5) Reverse ()

Sb.reverse (); Pallindrome

1)
2)
3)
4)

Page No.

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.

UNIT-3

Page No.

Multithreading :-

Here a program (process) is divided into two or more subprograms (process), which can be implemented at the same time in parallel.

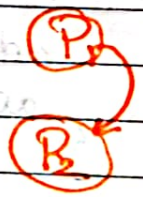
```
A
{
    m1  — thread
    m2  — thread
}
```

Multitasking :-

It is the process of executing several task simultaneously.

↳ **Process - based Multitasking :-**

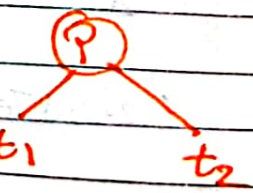
Eg :- Facebook chat, Music



- Heavy weight
- Own memory address space.
- Interprocess communication is expensive

↳ **Thread Based Multitasking :-** eg:- text editor — edit text — Print

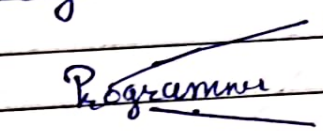
- Light weight
- Address space is shared
- Inter thread communication is cheap.



Page No.

Benefits of Multithreading :-

<i> Enable Programmer to do multiple task at one time



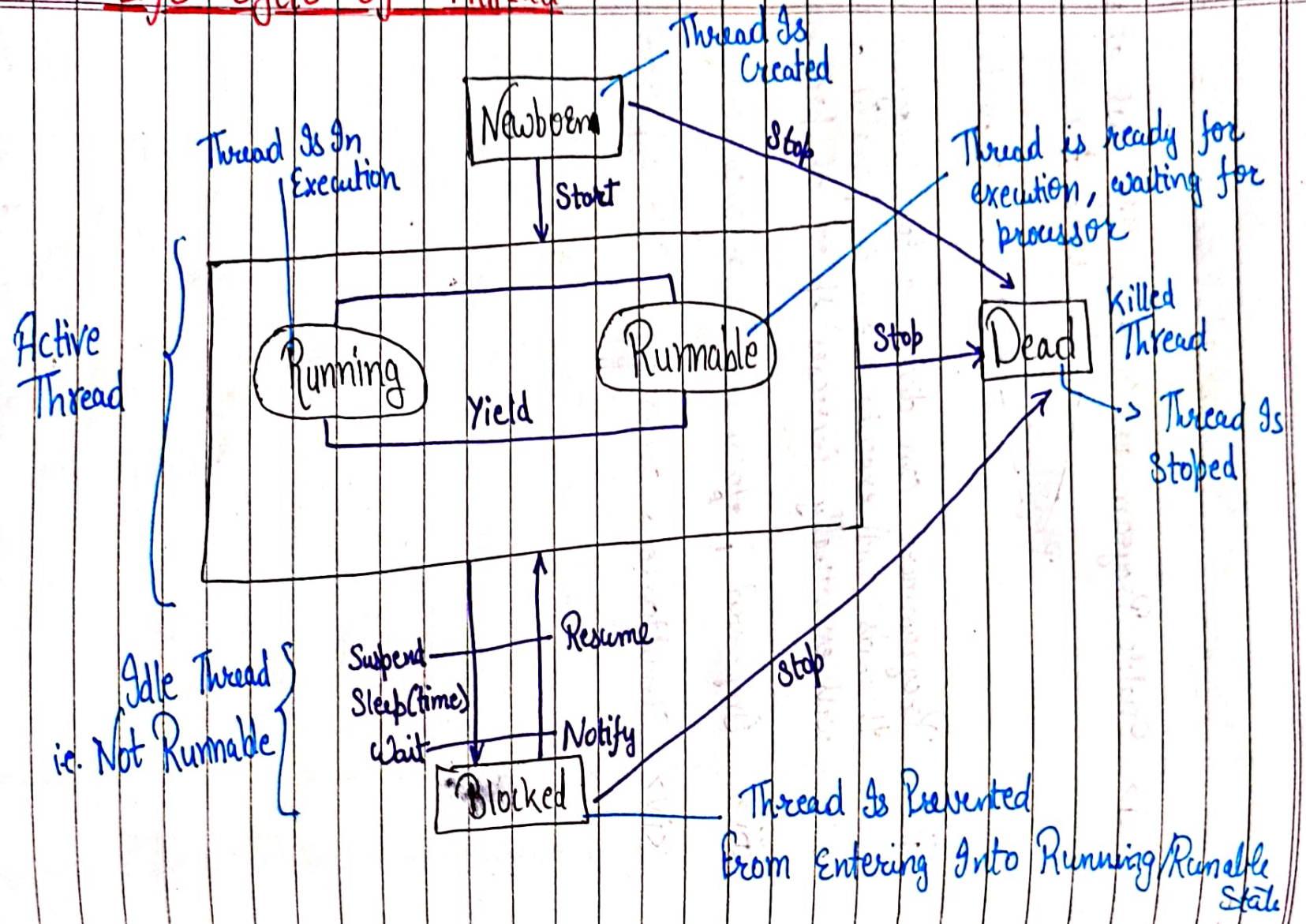
<ii> Programmers can divide a long program into threads and execute them in parallel which will eventually increases speed of program execution.

<iii> Improved performance and concurrency.

<iv> Simultaneous access to multiple applications.

Imp

Life Cycle Of Thread



Page No.

Page No.

Thread Creation

Main Thread :-

Every time a Java program starts up, one thread begins running which is called as the main thread of the program because it is the one that is executed when your program begins

You can implement Runnable Interface

Class t1 implements Runnable

You can extend the Thread class.

Class t1 extend Thread.

Thread Class

```
Class t2 extends Thread
{ public void run()
  { S.O.P("Thread t2");
  }
p.s.v.m (String a[])
  {
    t2 obj1 = new t2();
    obj1.start();
  }
}
```

start

Implementing Runnable Interface

```
class t1 implements Runnable
{ public void run()
  { S.O.P("Thread t1")
  }
p.s.v.m (String a[])
{
  t1 obj1 = new t1();
  Thread t = new Thread(obj1);
  t.start();
}
}
```

Page No.

Use Of Stop() Method :-

Stop() method kills the thread on execution

Ex :-

```
class A extends Thread
{
    public void run()
    {
        for (int i=1; i<=5; i++)
        {
            if (i==2)
                Stop();
            S.O.P("A:" +i);
        }
    }
    public static void main (String a[])
    {
        A a = new A()
        a.Start();
    }
}
```

Thread Gets Terminated ←

Output

A:1

A:
← loop
A:

Page No.

Use of Sleep() Method

It causes the currently running thread to block for atleast the specified number of milliseconds.

You need to handle exception while using Sleep() method

Example :-

```
class A extends Thread
{
    public void run()
    {
        for (int i=1; i<=5; i++)
        {
            try()
            {
                if (i==2) Sleep(1000);
            }
            catch (Exception e)
            {
            }
        }
        S.o.P("A" + i);
    }
}
p.s.v.m (String a[])
{
    A a = new A();
    a.Start();
}
}
```

Milliseconds

oms -->

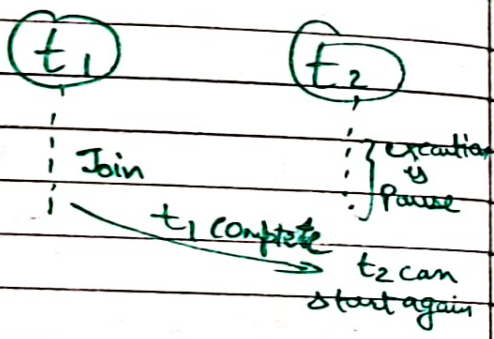
Page No.

Use of isAlive() and Joint() Method :-

isAlive() method tests if the thread is alive.

{ True / False } public final boolean isAlive()

joint() method waits for a thread to die. It causes currently running thread to stop executing until it joins complete its task.



Example :-

```
class A extends Thread
  {
   public void run()
    {
      S.O.P("Status: " + isAlive());
    }
}
class alivetest
   {
     p.s.v.m(String[] arg)
       {
          A a = new A();
          a.start();
          S.O.P. ("New Status" + a.isAlive();
       }
}
```

Output
Status : True
NewStatus : True

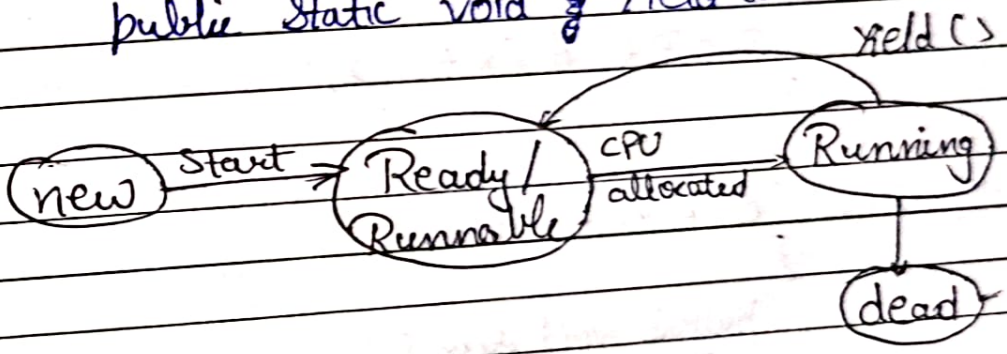
Page No.

Use Of Yield () Method :-

Yield method causes to pause current executing thread for giving the chance to remaining waiting threads of same priority

-> If there are no waiting threads or all waiting threads have low priority then the same thread will continue its execution once again.

public static void Yield ()



Example :-

```
class A extends Thread
 {
  public void run()
   {
    for (int i=1; i<=5; i++)
     {
      if (i==2) yield();
       S.O.P("A"+i);
     }
    S.O.P("A Exit");
   }
}
class B extends Thread
   {
    public void run()
     {
      for (int j=1; j<=5; j++)
       {
        S.O.P("B"+j);
       }
      S.O.P("B Exit");
     }
}
```

Page No.

```
class Yieldtest
  {
    p.s.v.m (String a[ ])
    {
      A a = new A();
      B b = new B();
      a.start();
      b.start();
    }
  }
```

Can we Start a thread twice?

No

```
class A extends Thread
  {
    public void run()
    {
      S.O.P("Thread A");
    }
    p.s.v.m (String [] ar)
    {
      A a1 = new A();
      a1.start();
      a1.start();
    }
  }
```

Output

Error :- IllegalThreadState Exception

Page No.

Synchronization :-

↳ Used to solve data inconsistency problem.

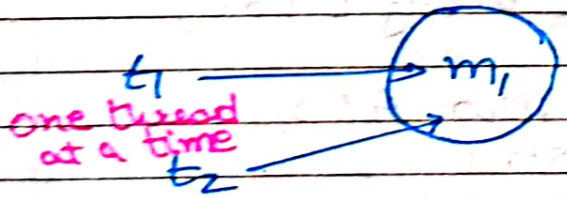
↳ Synchronization is the modifier applicable only for methods and blocks.

↳ Can't apply for classes and ~~objects~~ variables.

↳ Synchronization keyword in Java creates a block of code known as critical section.

↳ To enter a critical section, a thread needs to obtain the corresponding object's lock.

```
Synchronized (object)
{
    // Statements
}
```



Page No.

Program without Synchronization

```
class A
{
Synchronized void addnew(int i)
    {
      Thread t = Thread.CurrentThread;
       for(int n=1; n<=5; n++)
          {
            S.O.P(t.getName()+" - "(i+n);
          }
    }
class B extends Thread
    {
      A a1 = new A();
      public void run()
        {
          a1.addnew(100);
        }
    }

class Syntest
  { p.s.v.m(String a[])
  { B b = new B();
  Thread t1 = new Thread(b);
  Thread t2 = new Thread(b);
   t1.setName("T1");
   t2.setName("T2");
   t1.start();
   t2.start();
  }
}
```

O/P without Synchronized

```
T-101
T2-101
T2-102
T1-102
!
!
!
```

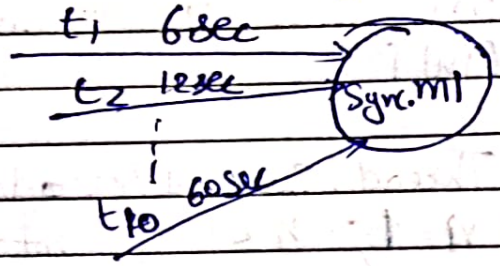
O/P with Synch.

```
T1 -101
T1 -102
T1 -103
T1 -104
T1 -105
T2 -101
T2 -102
T2 -103
T2 -104
T2 -105
```

Page No.

Problem Of Synchronization

↳ Increases waiting time



↳ Effect the performance of the system

S.No	Method with discription
1>	public void start()
	Starts the thread in a seperate path of execution, then invokes the run() method on this Thread object.
2>	public void run()
	If this Thread object instantiated using a seperate Runnable target, the run() method is invoked on that Runnable object.
3>	public final void setName(String name)
	changes the name of the Thread object. There is also a getName() method for retrieving the name.

Page No.

4> public final void setPriority (int priority)

Sets the priority of this Thread object. There is also a getName() method for returning the name.

5> public final ~~void~~ void join (long millisec)

The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.

6> public void interrupt()

Interrupts this thread, causing it to continue execution if it was blocked for any reason.

7> public final boolean isAlive()

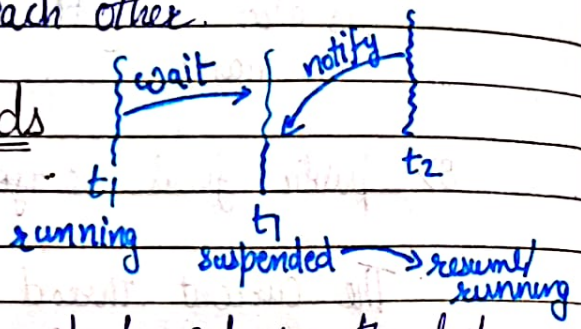
Returns true if the thread is alive.

Page No.

Interthread Communication

Used to allow synchronized thread to communicate with each other.

Methods



1) **Wait :-**

Causes current thread to release the lock and wait until

- specified time elapsed
- another thread calls notify or notifyAll method.

public void wait ()

2) **notify :-**

wakes up the single thread i.e. waiting on this object's monitor

public void notify ()

3) **notifyAll :-**

Wakes up all the threads waiting.

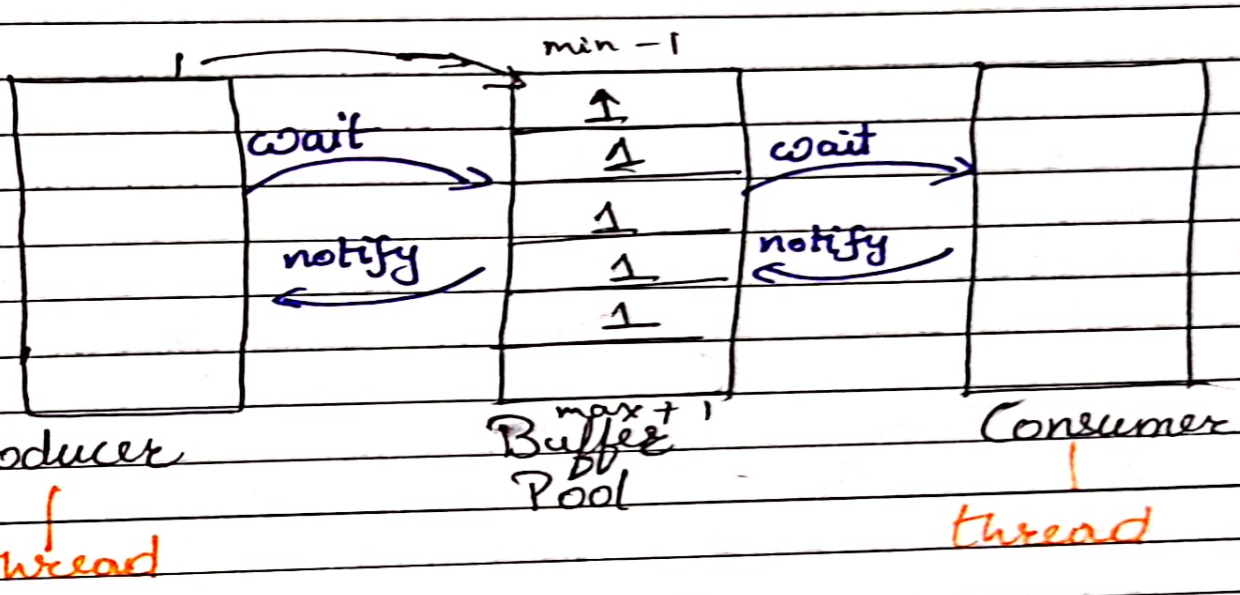
public void notifyAll ()

Page No.

Example of Inter thread communication

Producer - Consumer Problem

↳ multi process synchronization



Producer can't add data into Buffer if it is full.

Consumer can't consume data if it is empty.

Page No.

Page No.

Java AWT Programming

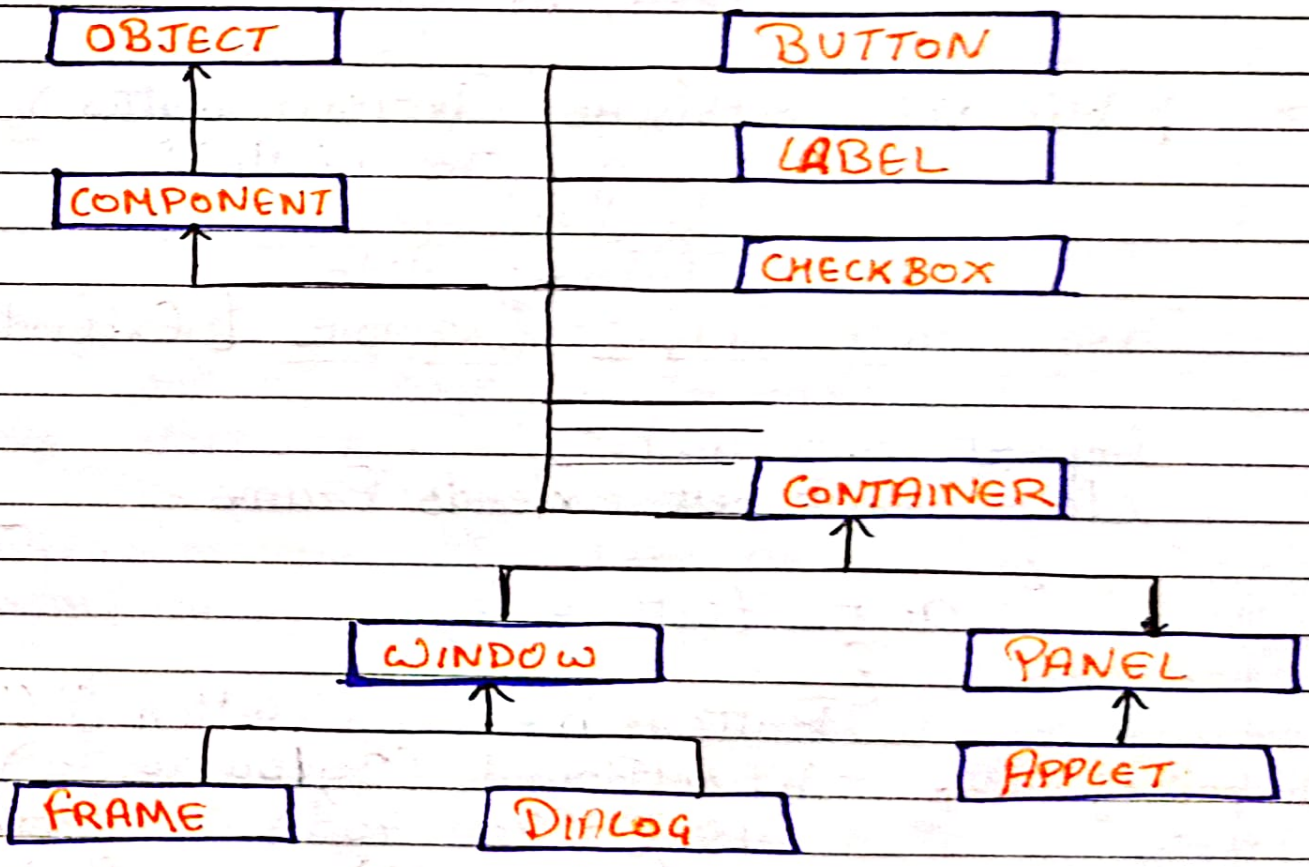
AWT - Abstract Window Toolkit

Used to develop GUI or Window based application.

AWT components are Platform dependent.

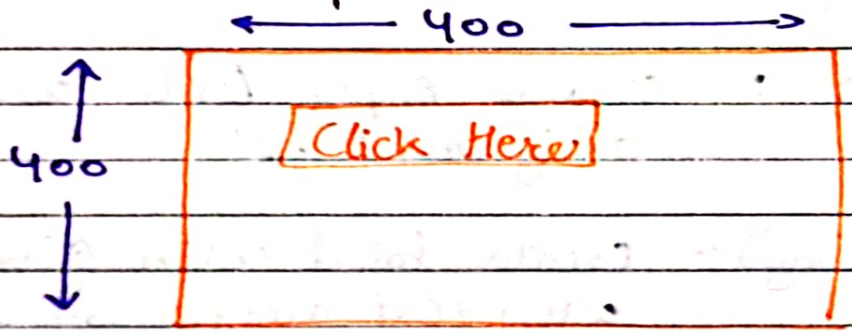
java.awt package is used.

Hierarchy



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

Output



Components Of AWT :-

Every GUI based program consists of a screen with set of objects. In java, these objects are called components.

Components frequently used as Buttons, Checkboxes, RadioButton, textfield etc.

At the top of AWT hierarchy is the component class. Component is an abstract class that encapsulates all the attributes of the component.

All User Interface elements that are displayed on the screen and interact with user are the subclasses of the component.

Page No.

IMPORTANT METHODS OF COMPONENT CLASS

<i> public void add (component c)
↳ Inserts component in container.

<ii> public void setSize (int width, int height)
↳ set the size.

<iii> public void setLayout (LayoutManager m)
↳ defines Layout but if not mention then, by default it is Flow Layout.

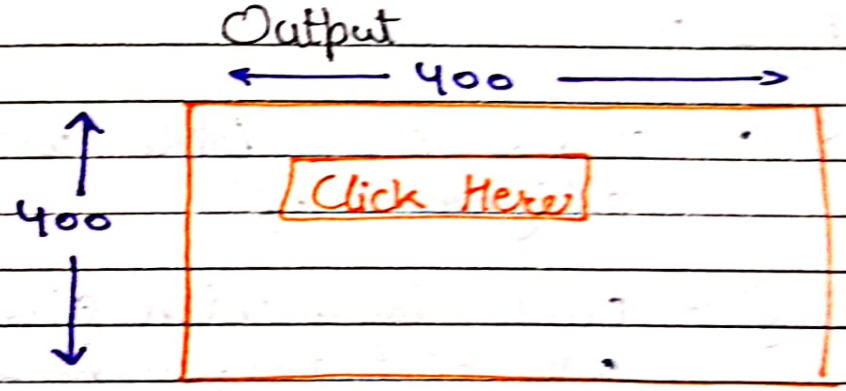
<iv> public void setVisible (boolean Status)
↳ Changes the visibility
True :- Show
False :- Hide.

Java AWT Simple Example [Extend Frame]

```
import java.awt.*;
class awt_test extends Frame
  {
    awt_test ()
      {
        Button b = new Button ("Click Here");
        b.setBounds (30,100,80,30);   // set the pos^n of Button
        setSize (400,400)             // Frame Size
        setLayout (null);             // By default Flow Layout
        setVisible (true);            // to add button
```

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

Output



Components Of AWT :-

Every GUI based program consists of a screen with set of objects. In java, these objects are called components.

Components frequently used as Buttons, Checkboxes, RadioButton, textfield etc.

At the top of AWT hierarchy is the component class. Component is an abstract class that encapsulates all the attributes of the component.

All User Interface elements that are displayed on the screen and interact with user are the subclasses of the component.

Page No.

① Labels :-

- Object of type label
- Contains a string which it displays.
- do not support any interaction with the user.

Constructors :-

Label() :- creates empty lable with its text allignment left.

Label(String) :- creates label with given string with text allignment left.

Label(String, int align) :- creates Label with given string and with given allignment.

Available Allignment :-

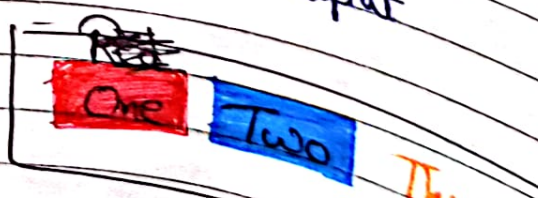
Label.left
Label.right
Label.center

Page No.

Example

```
import java.awt.*;
import java.applet.*;

public class LabelDemo extends Applet
  {
    public void init ()   // init always use
     {
      Label one = new Lable ("One", Label.RIGHT);
       One.setBackground (Color.Red);
                          // for Background color

      Label two = new Lable ();
      two.setBackground (Color.blue);
       two.setText ("Two");

      Lable three = new Label ("Three")
       three.setforeground (Color.orange);
                          // For Character color
      Lable four = new Label ();
      four.setText (three.getText ());

      add (one);
      add (two);
      add (three);
      add (four);
  }
```

Output



Three

Page No.

Page No.

② Buttons :-

Buttons are simple components that trigger some action on your interface.

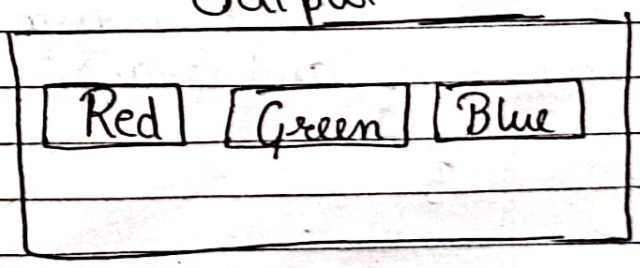
Button ()
Button (String)

Contains a label and generates an event when pressed

Example

```
import java.awt.*;
import java.applet.*;

public class ButtonDemo extends Applet
  {
    Button rbtn, gbtn, bbtn; // initialize button name.
    public void init ()
    {
      rbtn = new Button ("Red");
      gbtn = new Button ("Green");
      bbtn = new Button ("Blue");

      add (rbtn);
      add (gbtn);
      add (bbtn);
    }
}
```

Output



Page No.

③ Checkbox :-

Control used to turn option on or off

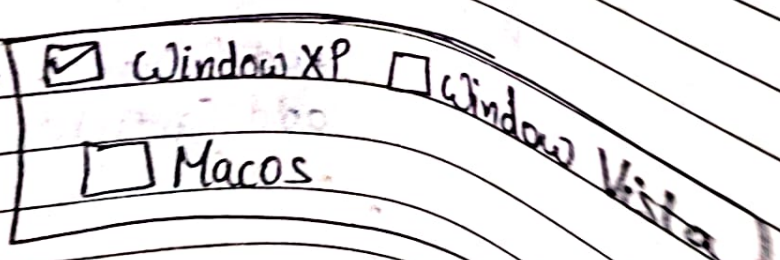
Checkbox()
Checkbox(String str)
Checkbox(String str, boolean on)

In this we can choose 1 or more options.

Example :-

```
import java.awt.*;
import java.applet.*;

public class checkboxDemo extends Applet
{
Checkbox winXP, winVista, mac; //initializing checkbox name it show
public void init()
{
winXP = new Checkbox("Window XP", true);
winVista = new Chuckbox("Window Vista");
mac = new Checkbox("Mac OS");

add(winXP);
add(winVista);
add(mac);
}
}
```

Output



Page No.

④ Checkbox Group :-

Helps to create a set of mutually exclusive check boxes
These check boxes are often called radio buttons
In this, we can only select 1 option.

Checkbox Group ( )

Constructor of checkbox class used for creating radio buttons.

Checkbox (String str, boolean on, CheckboxGroup cb)
or
Checkbox (String str, CheckboxGroup cb, boolean on)

Example

```
public class CBGroup extends Applet
{
    Checkbox winXP, winVista;
    CheckboxGroup cbg;
    public void init ()
    {
        cbg = new CheckboxGroup ();
        winXP = new Checkbox ("Window XP", cbg, true);
        winVista = new Checkbox ("Window Vista", cbg, false);

        add (winXP);
        add (winVista);
    }
}
```

Output

(•) Windows XP ( ) Windows Vista

Layout Manager

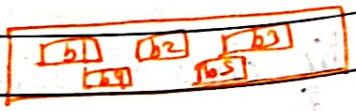
↳ Layout manager is the process that determines the size and position of components in a container.

↳ Each time we create a container java automatically creates a default layout manager i.e. Flow Layout.

↳ You can create different types of layout manager in order to control how your applet looks.

↳ Set by setLayout() method.

1) Flow Layout



↳ Flow layout is the default layout for JPanel

↳ It sets the components (button) as its preferred size in Java.

↳ It also sets the width of the button according to its content. (component)

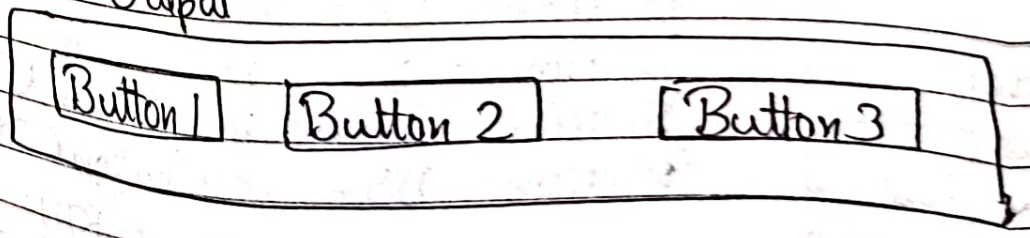
↳ If width of Jframe is less than it shifts the extra components to the next row.

↳ If width of Jframe is more than it aligns the components to the center.

Example :-

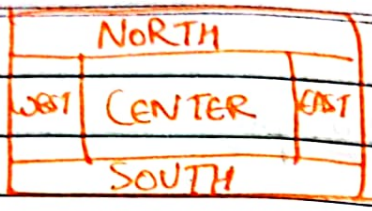
```
import java.awt.*;
import java.applet.*;

public class Flowlayout extends Frame
  { Flowlayout()
  {Button b1 = new Button ("Button 1");
   Button b2= new Button ("Button 2");
   Button b3= new Button ("Button 3");

   add(b1);
   add(b2);
   add(b3);
  }
public static void main(String ar[])
   {
   Flowlayout f = new Flowlayout();
} }
```

Output



Page No.

2) Boder Layout :-



It is used to arrange the components in five region :- north, south, east, west and centre Each region contain only 1 component.

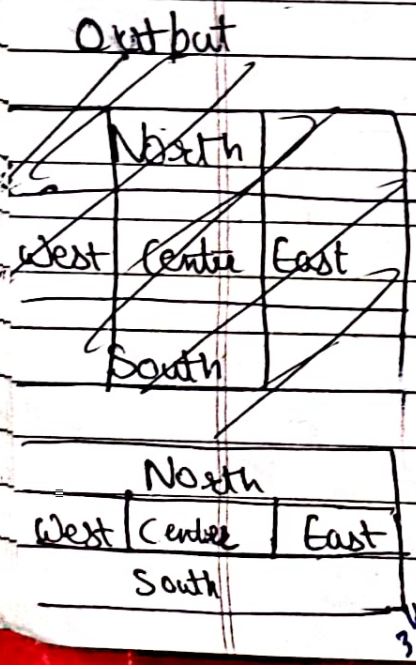
Border layout () :- Creates a boreder layout but with no gaps b/w the components.

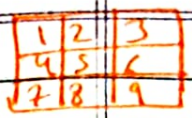
BBorderlayout (int hgap, int vgap) :

Creates a border layout with the given horizontal and vertical gaps b/w the components.

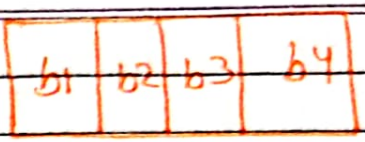
Ex :-
```
import java.awt.*;
public class Border extends Frame
 {
   Border ()
   {
    Button b1 = new Button ("North");
    Button b2 = new Button ("South");
    Button b3 = new Button ("West");
    Button b4 = new Button ("East");
    Button b5 = new Button ("Center");

    add (b1, Borderlayout.NORTH);
    add (b2, Borderlayout.SOUTH);
    add (b3, Borderlayout.WEST);
    add (b4, Borderlayout.EAST);
    add (b5, Borderlayout.CENTER);
   }
  public void main ()
  {
    Border b = new Border ();
```

Output





Grid Layout



↳ Just like flow layout, grid layout also sets the component left to right in a flow.

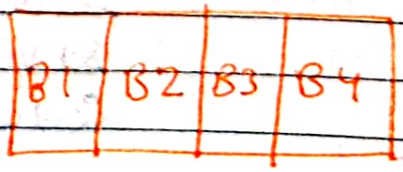
↳ In grid layout, all the available space is consumed by the components.

↳ Grid layout can be devided into the form of rows and columns.

↳ In grid layout, all the components have same size.

Ex:-

```
public class GridlayoutSample extends JFrame
{
    JButton b1= new JButton ("B1")
    JButton b2 = new JButton ("B2")
    JButton b3= new JButton ("B3")
    JButton b4 = new JButton ("B4")

    public GridlayoutSample ()
    {
        ~~Gridlayout g = new Gridlayout ();~~
        ~~c.get c.setlayout(g);~~
        add (b1);
        add (b2);
        add (b3);
        add (b4);
    }
    public static void main ()
    {
        GridlayoutSample g = new Gridlayout (2,2)
        g.setlayout (g);
    }
```



Page No.

Grid Bag Layout

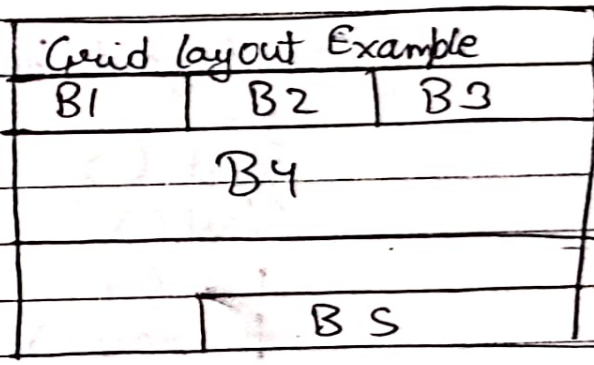
↳ It is the most powerful and flexible of all the layout managers but more complicated to use.

↳ Unlike grid layout, where the components are arranged in rectangular grid and each component in the container is forced to be same size, in GridBagLayout, components are also arranged in rectangular grid but can have different sizes and can occupy multiple rows and columns.

```
import java.awt.*;
import java.awt.event.*;
public class Mylayout extends JFrame
{
    JButton btn1 = new JButton("B1");
    "      btn2 = "    "    ("B2");
    "      btn3 = "    "    ("B3");
    "      btn4 = "    "    ("B4");
    "      btn5 = "    "    ("B5");

    public Mylayout()
    {
        setTitle("Grid Baglayout Example");
        setBounds(100,200,500,300)
        setVisible(true);
        Container c = getContentPane();
```

Page No.

```
GridBagLayout g = new GridBagLayout();
c.setLayout(g);

GridBagConstraints gbc = new GridBagConstraints();

gbc.gridx = 0;
gbc.gridy = 0;
c.add(btn1, gbc);

gbc.gridx = 1;
gbc.gridy = 0;
c.add(btn2, gbc);

gbc.gridx = 2;
gbc.gridy = 0;
c.add(btn3, gbc);

gbc.gridwidth = 3;
gbc.gridx = 0;
gbc.gridy = 1;
gbc.ipady = 40;
c.add(btn4, gbc);

gbc.gridwidth = 2;
gbc.ipady = 0;
gbc.gridx = 1;
gbc.gridy = 2;
gbc.anchor = GridBagConstraints.PAGE_END;
gbc.weightx = 1;
c.add(btn5, gbc);
```



Card Layout

↳ It treats each component in a container as a card.

↳ Only one card is visible at a time and the container acts as a stack of cards.

↳ The first component added to a CardLayout object is the visible component when the container is first displayed.

↳ CardLayout has some useful methods to flip the cards i.e.

first()
last()
next()
previous()
show()

Ex:-
```
class CardLayoutExample
{
    public static void main(String arg[])
    {
        JFrame frame = new JFrame();
        frame.setVisible(true);
        frame.setBounds(100, 100, 500, 400);
        frame.setTitle("Card Layout Example");
        Container c = frame.getContentPane();
```

Page No.

```
CardLayout Card = new CardLayout();
~~c.setLayout(CardLayout~~
c.setLayout(Card);

JButton B1 = new JButton("Page 1");
"  B2 = "  "  ("Page 2");
"  B3 = "  "  ("Page 3");
"  B4 = "  "  ("Page 4");

~~Center~~
c.add(B1, "1");   <-- ID
c.add(B2, "2");
c.add(B3, "3");
c.add(B4, "4");
}
}
```