

2. What is the effect of
temperature on the rate of
chemical reaction?

Ans. The rate of chemical reaction increases with increase in temperature.

Explain how a catalyst affects the rate of reaction.

Ans. A catalyst increases the rate of reaction by providing an alternative pathway for the reaction.

Explain how a buffer solution resists change in pH.

Ans. A buffer solution resists change in pH because it contains a weak acid and its salt.

Explain how a precipitate forms.

Ans. A precipitate forms when two soluble salts are mixed in such a way that they form an insoluble salt.

Explain how a solubility product constant is determined.

Ans. A solubility product constant is determined by dissolving a salt in water and then titrating the solution with a strong acid or base.

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 :-

Java

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 collector 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.

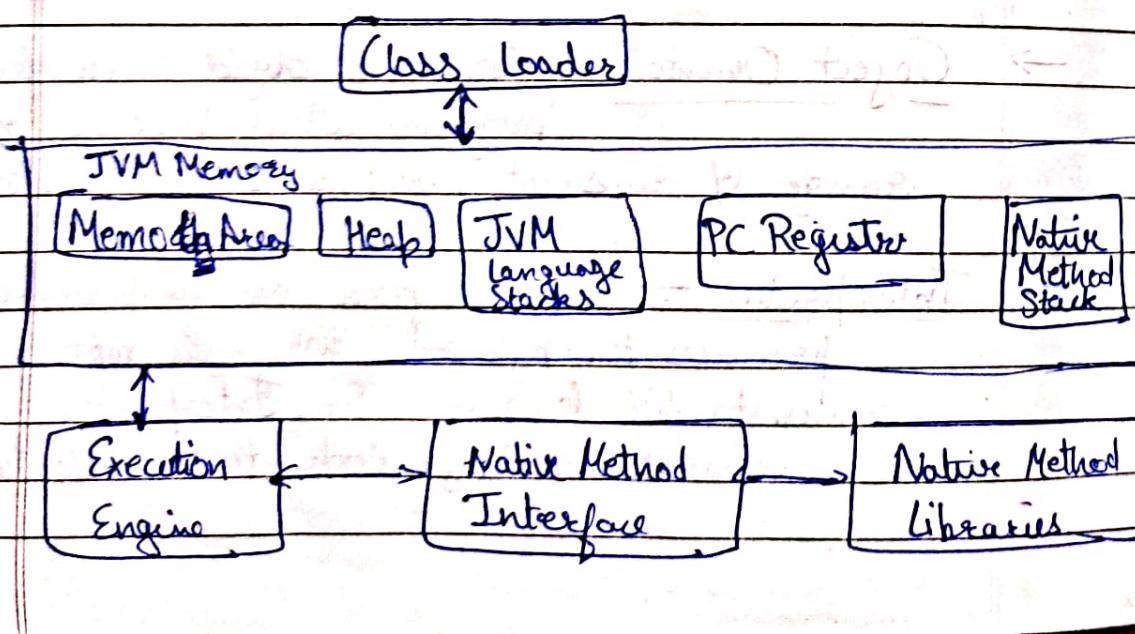
→ Robust and Secure :- Java programming is more reliable. At the time of execution, it shows all the error. Java also provides the lot of security as the computer will be attacked by the external program.

→ Multi-threaded :- It is defined as the program's ability to perform several tasks or functions 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 ~~obj~~ 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 is complete.

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

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

Execution Engine :- It is a type of SW 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 information 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.

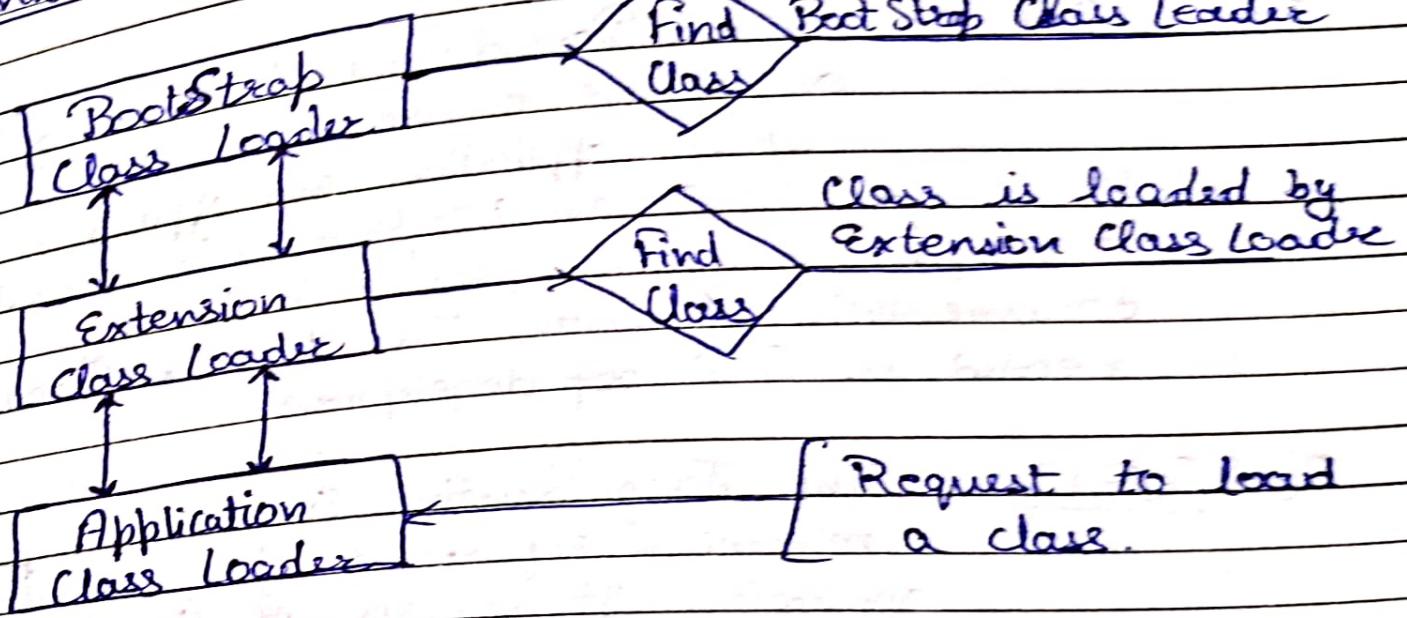
2) Linking :- Performs verification, preparation and resolution.

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

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

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

Initialization :-



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 (JAVA C), an archiver (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 1

3) JVM :- A specification where working of JVM is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by ~~Sun~~ 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 command prompt to run the Java class, an instance of JVM is created.

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

JDK

Java Runtime
Environment

Java Virtual Machine	+ Library Classes	+ Development Tools
----------------------	-------------------	---------------------

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 operate on them. So, that no other part of the code can access this data except that function.

OOPS Concepts

Class

7. Encapsulation

Object

8. Polymorphism

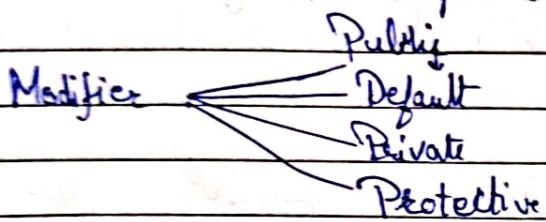
Method

Message Passing

Abstraction

Inheritance

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:



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 returns the result to the caller.

Method can perform some specific task without returning anything.

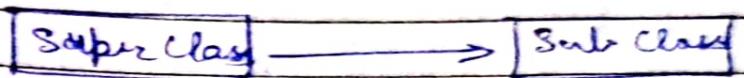
Methods are the time savers and help us to reuse the code without retyping the code.

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 user.

Inheritance :- It is the mechanism in Java by which one class is allowed 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 as 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

→ Runtime Polymorphism

→ Compile Time Polymorphism.

lecture 5

TYPES OF POLYMORPHISM

b) Compiletime Polymorphism (Static Polymorphism)

A polymorphism i.e. evolved 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

{ MethodName }

 Static int Multiply (int a, int b)

{

 ↳ Method with
 ↳ two integer parameters

 return a * b;

}

 Static double Multiply (double a, double b)

{

 return a * b;

}

 ↳ Method with same
 name but two double
 parameters

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)

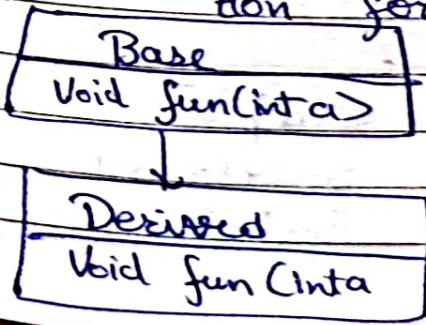
It is a process in which a function call the overridden method , is resolved at Runtime . This type of polymorphism is achieved by method overriding

Advantage of Runtime Polymorphism

Support Method Overriding
Common Method Specification



It allows subclasses to add specific method and allow subclasses to define specific implementation for the same .



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

```
public static void main (String [] args)
```

```
{
```

```
    Parent a;
```

```
    a = new Subclass 1();
```

```
    a.Print();
```

```
    a = new Subclass 2();
```

```
    a.Print();
```

```
}
```

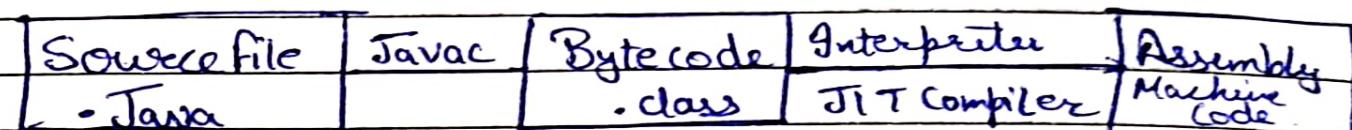
Output

Subclass 1

Subclass 2

JVM Class file format :-

JVM ~~class~~ source code is compiled into ByteCodes 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 byte code one by one on the host machine CPU.

Ex:- Public Class classOne

 {
 Public Static void main (Strings [] args)
 }

 System.out.println ("HelloWorld");
 }

 Static class Static Nested class

 {
 }

 class classTwo

 {

 interface InterfaceOne

 {
 }

After compilation we get

- 1> class one & static Nested class - class
- 2> Class One.class
- 3> classTwo.class
- 4> InterfaceOne.class

Java Class File Format :-

Magic Number	Version
Constant Pool	
Access Flags	
This class	
Super class	3>
Interface	
Fields	
Methods	
Attributes	

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 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

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.

5) This Class :-

		represent the type of entry in constant pool (Class/Interface)	
		Index (2Byte)	
Constant pool	Content Index	Class Interface Name	1Byte
	0x0004		
	0x0005		
	0x0006		
	0x0007		
		0x0004	2Byte
			0x0007
			This Class

6) Super Class

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.

Interface :-

[2 Byte]	Index	Interface Sector
----------	-------	------------------

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~~ interface.

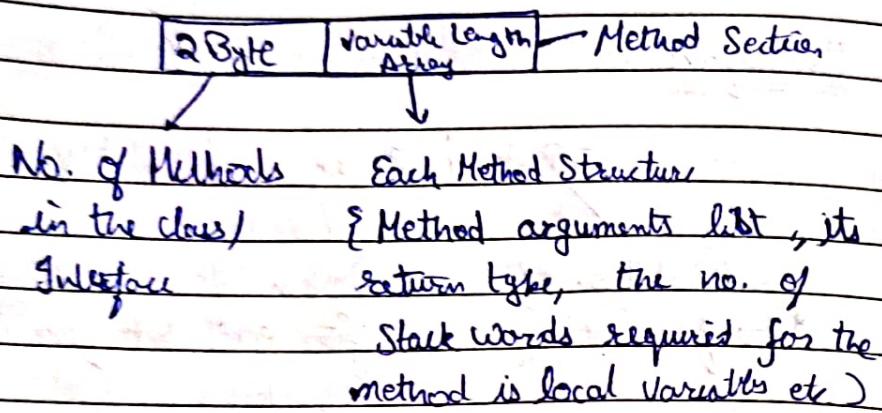
[2 Byte]	Variable length of array	field Sector
----------	-----------------------------	--------------

Total No. of Field

Reference to the constant
pool entry.

Method :-

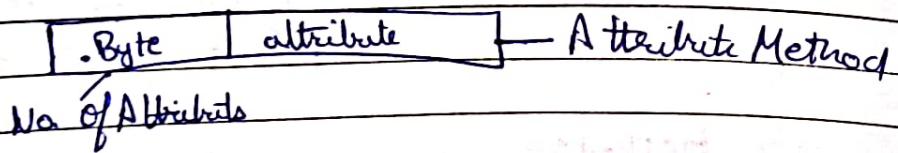
Method components host the methods that are explicitly defined by this class, not any other methods that may be inherited from super class



10) Attributes :-

Attribute Section contains several attribute about the class file.

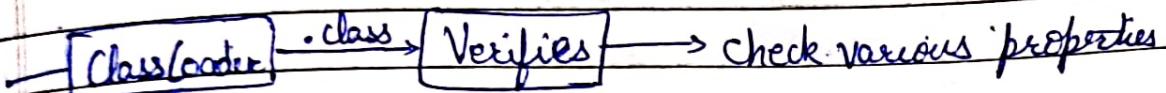
Eg :- One of the attribute is the source code attribute which reveals the name of the source file from which this class file was compiled.



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.

Security Promises of the JVM :-

Here are some of the promises the JVM makes about program that have passed 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.

Lecture 9Garbage 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 runs 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;
```

Step 2

```
(4>     Public void setData(int c, int d)
```

```
5> {  
6>     a=c;  
7>     b=d;  
8> }
```

Step 3

```
9>     Public void ShowData(){  
10>        System.out.println ("Value of a = " + a);  
11>        System.out.println ("Value of b = " + b);  
12>    }
```

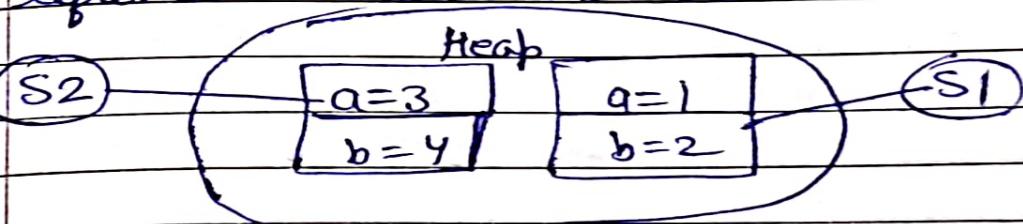
Step 4

```
13> Public static void main (String args[]){  
14>     Student S1 = new Student ();  
15>     Student S2 = new Student ();
```

Step 5

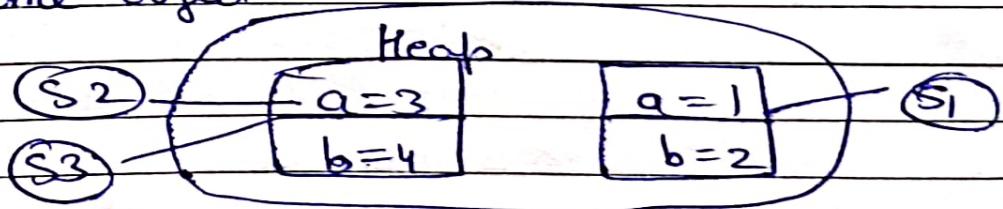
```
> S1. SetData(1,2);  
> S2 . Set Data(3,4)  
S1. ShowData  
> S2 . ShowData  
> // Student S3,-  
> // S3 = S2;  
> // S3 . Show Data();-  
> // S2 = null;  
> // S3 = ShowData()  
> // S3 = null;  
> // S3 = ShowData();  
> } }
```

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



Uncomment line 1120, 21, 22 ~~2020~~ : Save
complete and run the code.

Here two reference variables are pointing to the same object

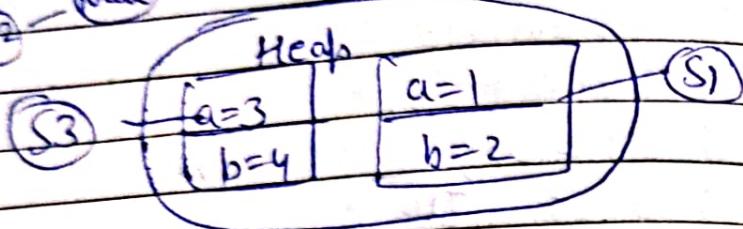


Uncomment line #123, 1124. Compile line & Run the code.

Step

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

(S2) - Null

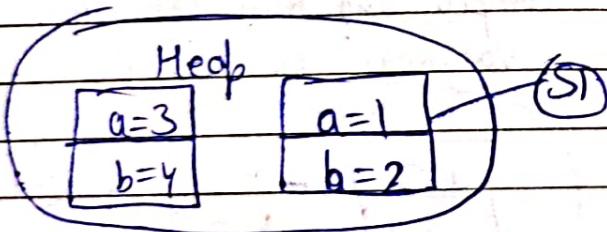


Step 7: Uncomment line 11/25 8/26 save Compile & Run this ↳

Step

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. ↳

(S2) - Null
(S3)



Sandbox Model

JDK 1.0 Security Model

Load Code

JVM

Sandbox

Remote Code

Valuable Resources (files)

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 whereas Remote Code (Eg:- Applet) is not trusted, that's why, it is first pass through the Sandbox.

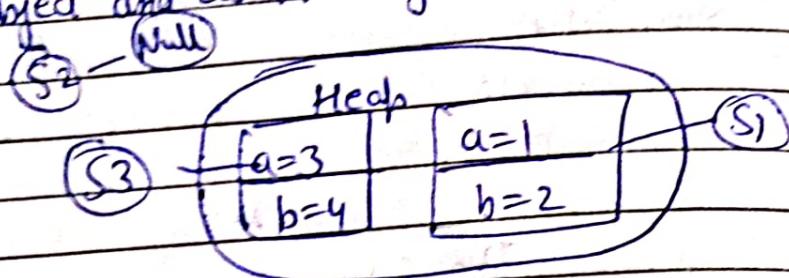
It was developed through JDK and was generally adopted by application built with JDK 1.0 including java-enabled web browser.

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 compare 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.

↓

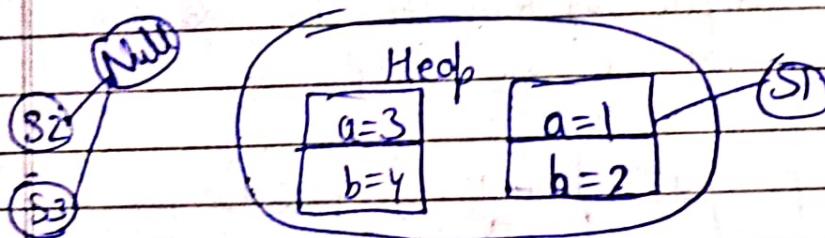
Step 6: S2 become null but S3 is still pointing to the object and is not eligible for garbage collection.



Step 7: Uncomment line 11/25 8/26 gave Compile & Run time ↳

Step

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

Remote Code

Local Code

Sandbox

JVM

Valuable Resources (files)

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

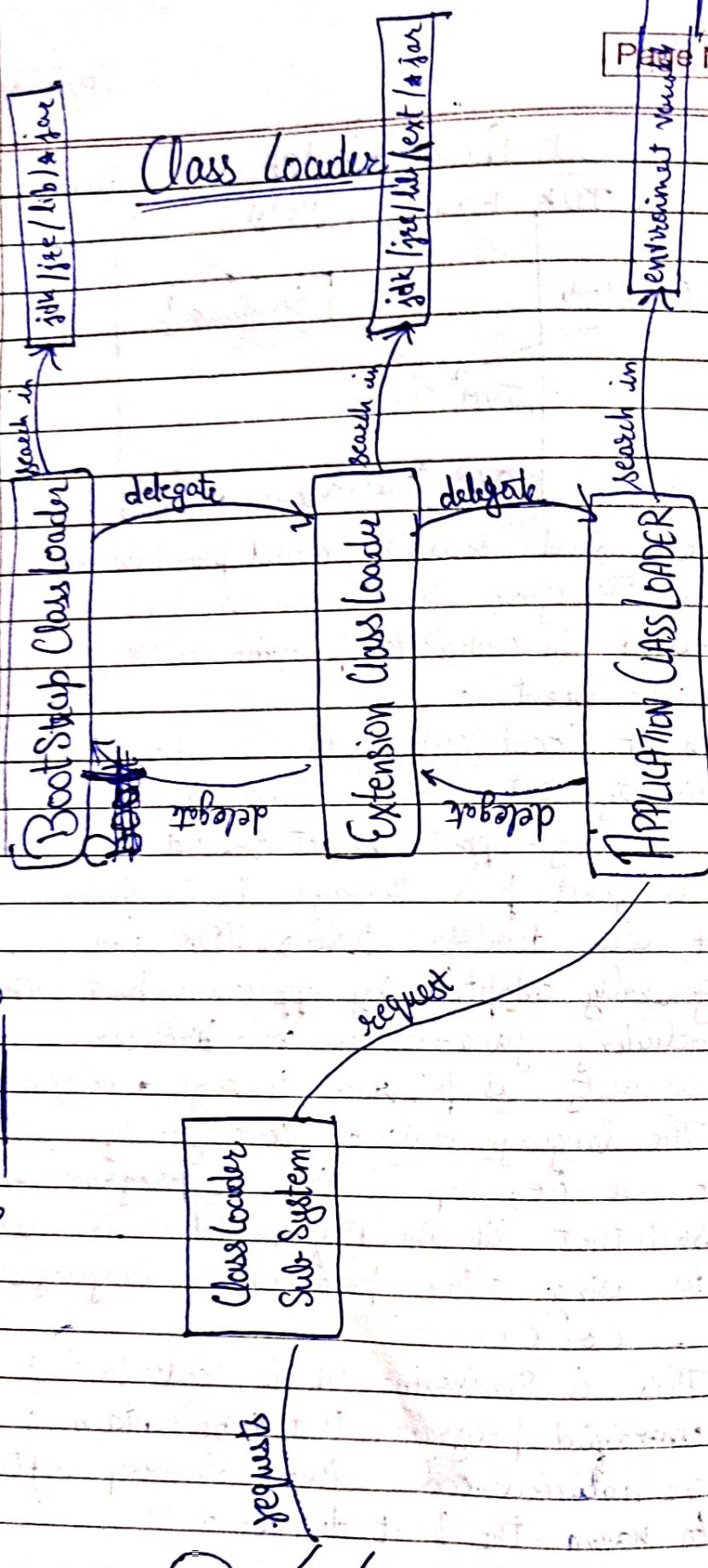
In it local code is trusted, so that it is directly enter to the JVM whereas Remote Code (Eg:- Applet) is not trusted, that's why, it is first pass through the sandbox.

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



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. ~~ext~~ 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 search 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

87 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

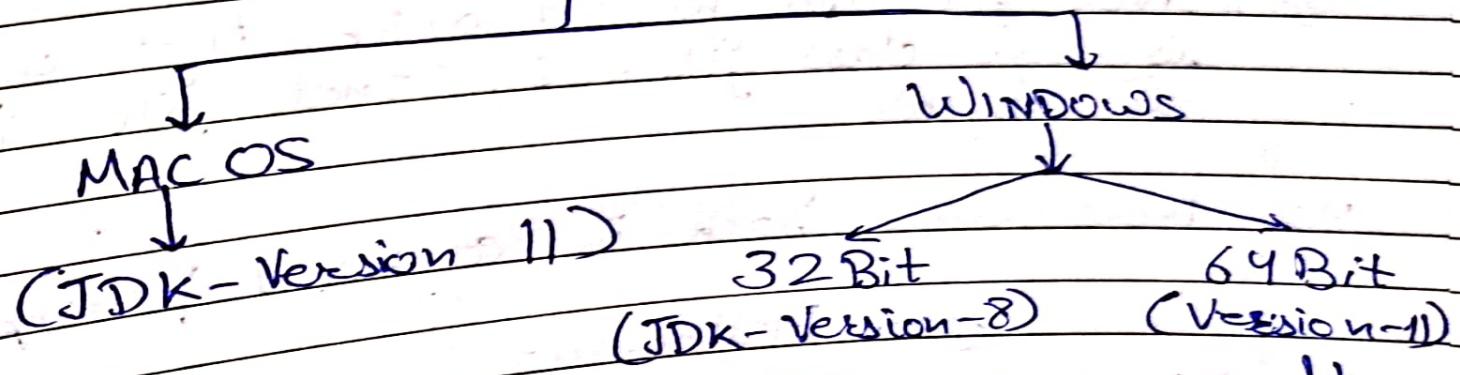
For Practice go to www.w3schools.com

JAVA (ApniKaksha)

Page No.

Installation (Lecture 2)

Download JDK



```
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

Types Of Variables in Java

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

Variables (Posⁿ 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 before using it

Instance 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.out.println(a1.i);
```

```
}
```

↳ for two variables

```
Abc a1 = new Abc();
```

```
a1.i = 200;
```

```
Abc a2 = new Abc();
```

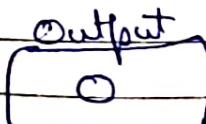
```
a2.i = 400;
```

↳ Initialization is not mandatory

```
int i;
```

```
Abc a1 = new Abc();
```

```
System.out.println(a1.i)
```



Static Variable :-

Scope of class

Initialization is not mandatory

class Abc

```
{ Static String college = "EFC" } static
```

```
int std_id;
```

```
public static void main (String ar[])
```

{

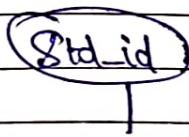
```
Abc a1 = new Abc ();
```

```
Abc a2 = new Abc ();
```

```
a1.std_id = 1;
```

```
a2.std_id = 2;
```

}



EFC

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

Local Variables :-

Create inside a

↳ Block

↳ Method

↳ Konstruktor

↳ Initialization is mandatory

class Abc

{

 public static void main (String [] a)

{

 int i = 10;

}

 void main ()

{

}

X

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; // Primitive  
        Integer f = Integer.valueOf(i);  
        Integer ab = f; // Autoboxing  
        S.o.p ();  
    }  
}
```

Wrapper to Primitive

```
class wrapper2 {  
    p.s.v.m (String []a)  
    {  
        Integer i = new Integer(10);  
        int ab = i.intValue();  
        int j = i; // Unboxing  
        S.o.p ();  
    }  
}
```

Loops

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

Types of loops

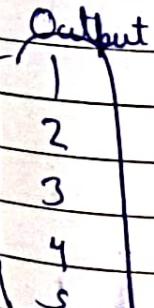
Entry Controlled loop
i.e. cond is checked at entry time

while loop

for loop

Exit Control loop
i.e. cond is checked during exit

do-while loop



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

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.

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 constructor.

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);
```

```
}
```

class supertest

{

public static void main (String a[])

{
child c1 = new child();
c1.m();}

3

Output
20
10

Example to call immediate Parent Class Constructor
using Super Keyword

class Parent

{

Parent ()

{ S.O.P. ("Hello"); }

}

class child extends Parent

{

class Supertest

if super() is not child()

written here, by default {

it will be super

as it call parent class Super ();

}

{ p.s.v.m (String a[]) }

{

child c1 = new child();

}

Output

Hello

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
```

```
{
```

```
b.s.v.m (String ar[])
```

```
{
```

```
child c1 = new child();
```

```
c1.m1();
```

```
}
```

Case 1

Output :-

Child

Patent

Case 2

If this block is empty then it calls parent class

Case 2

Output :-

Var-Ag Methods in Java

↳ Method with variable number of arguments

↳ Declaration is like [method (int... x)]

Ex:- class ABC

{

public static void m1 (int... i)

{

S.O.P ("ABC")

3

b.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

3

}

output

ABC

ABC

A BC

ABC

Important Points Regarding Var-Arg Methods

Var-arg parameter can be mix with normal parameter.

Ex:- $m1(\text{int } x, \text{String } \dots y)$ - Valid
 $m1(10)$
 $m1(10, "ABC")$

During mixing, Var-arg parameter should be last.

Ex:- $m1(\text{int} \dots x, \text{String } y)$ - Not Valid

There can be only one var-arg parameter.

Ex:- $m1(\text{int} \dots x, \text{String} \dots y)$ - 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:- $m1(\text{int.} \dots x)$ ← $m1(10, 20)$
 Last Priority $m1(\text{int } i, \text{int } j)$ ← First Priority

For Each Loops

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

Present only in 1.5 version or above

Example :-

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

for-loop

for-each loop

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

Output

1
2
3
4
5

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.

Difference B/w length() vs length

Length()

It is a final method applicable only for string objects.

Represents the no. of characters in string.

Ex :-

```
String s1 = "Hello"
S.O.P (s1.length());
```

O/P

5

length

It is final variable applicable only for arrays.

Represents the size of array.

Ex :-

```
int [] ar = new int [10];
S.O.P. (ar.length)
```

O/P

10

Java Programming (Inheritance)

→ No Multiple Inheritance
as it shows ambiguity or error

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

↳ Old class is known as Superclass

↳ New class is known as Subclass

Benefits :-

↳ Reusability of code

↳ Code sharing

↳ Consistency in using an interface.

Types:-

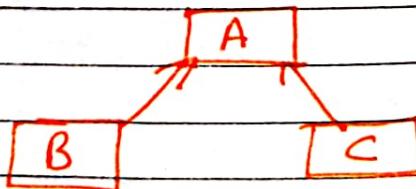
(i) Single Inheritance



(ii) Multilevel Inheritance



(iii) Hierarchical Inheritance



Final Keyword :-

Final Variable :- its value cannot be changed during the scope of program

Final Method :- Cannot be overridden

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 ac)

{

 finalvar f1 = new finalvar();

}

Another Example :-

class final method {

int i=10;

final void m1()

It shows error as
overridden is not
possible as m1() is
S.O.P ("Final"); already declared as
final in parent class.

}

class child extends final method

void m1()

S.O.P. ("Child");

class finaltest {

b.s.v.m (String a[])

child c1 = new child();

c1.m1();

abstract

}

Interface :-

{ Multiple Inheritance in Java }

An interface is a collection of abstract methods.

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 IT
class Name *Interface Name*

Ex :-

interface i1
{
 =====
 4methods

3
→ class c1 implements i1
{

3

How Interface Provide Multiple Inheritance in Java

interface i1

```
abstract public void m1();
int i=10;
```

interface i2

```
abstract public void m1();
int i=100;
```

class interfacetest implements i1, i2

```
public void m1()
```

No ambiguity

```
S.O.P("Child class");
```

```
S.O.P("i in i1 "+i1.i);
```

```
S.O.P("i in i2 "+i2.i);
```

}

b.s.v.m (String ar[])

```
interfacetest obj1 = new interfacetest();
```

```
obj1.m1();
```

}

child class
i in i1 10
i in i2 100

Similarity b/w Interface and class

They can have any number of methods
Both file name ends with .java extension.

Bytecode in .class file.

Both appear in packages

Difference b/w Interface and class

Interface cannot be instantiated

Interface has no constructor

All the methods of interface are abstract

Interface uses implement where class uses extend

An Interface can be used as multiple inheritance

Interface cannot contain instance fields

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 ~~atst~~ Abstract Method has no body.
- ↳ Child class needs to override the defⁿ of all abstract methods.

Example Of Abstract class / Methods :-

abstract class parent

{

 abstract void m1();

 void m2();

{

 S.O.P. ("Parent");

}

{

class child1 extends parent1

{

void m1()

{

S.O.P. ("Child")

}

}

class abstracttest

{

b.s.v.m (String ac[])

{

child1 c1 = new child1();

c1.m1();

c1.m2();

}

}

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/b, o/b etc.

Same class

Same Pack. Non Sub.

Diff Pack Sub.

Diff Pack Non Sub.

Example to Create
Package

Package Name
~~~~~  
package Hi package;

Importing Package in  
other class for all  
classes

import 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);

}

O/P

15

class package test

{

b.s.v.m (String a[])

{

C1 obj = new C1();

C1.m1();

}

## 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 be accessible outside the package. → Yes but only in inherited class.

## 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 block.

```
class except
{
```

b.s.v.m (String a[ ]) gt will not run

~~error/exception~~  
(Divide by zero)

```
int i = 100/0;
S.O.P. ("Value of i" + i);
```

~~Without  
try-catch  
block~~

class except1

{

p.s.v.m (String a[ ])

{

try

It will run

{

and it will give O/P int i = 100/0;  
which is ~~isn't~~ present

}

catch (Arithmetic Exception e)

{

S.O.P (e);

}

S.O.P ("After try - catch"),

3

}

Output

java.lang.ArithmeticException: / by zero

After try catch

Note:- Within the try block if anywhere an exception occurs, the rest of the try block won't be executed even though we handled that exception.

## 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 c.getMessage ( ),

description  
x Name x Stack Trace

try {  
} catch {  
} catch {  
} catch {  
} }  
order is imp

as catch allows 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 ex 1

b.s.v.m (String art)

try

error

will not 3

int i = 100/0;

try

int i=100/0;  
3

3

catch (ArithmeticException e)

S  
3

S.O.P. (e);

3

catch

catch (ArithmeticalException e) {  
 System.out.println("An error occurred: " + e.getMessage());  
}

3

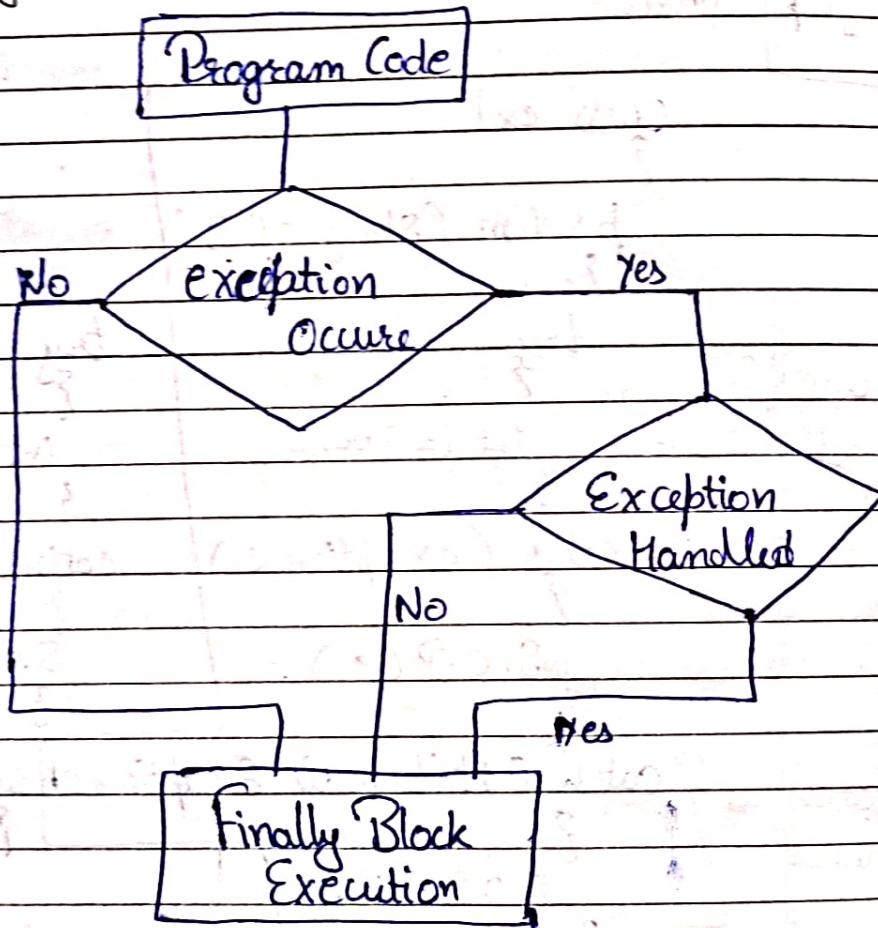
3

3

SOP(e1);

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

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



## Example of finally Block :-

class ex1

{

p.s.v.m (String ar[])

{

try

int i = 10/0;

}

catch (ArithmeticException e)

{

S.O.P (e);

}

finally

{

S.O.P ("finally");

}

}

### Output

| java.lang.ArithmeticException : / by zero |

Finally

### 3) Throw Keyword :-

47

It is used to explicitly throw an exception.

`throw new ArithmeticException (" - ");`

Example :-

```

class throw
{
    static void valid (int i)
    {
        if (i<18)
        {
            throw new ArithmeticException ("Not Valid");
        }
        else
        {
            S.O.P (" Welcom ");
        }
    }

    b.s.v.m (String a[])
    {
        try
        {
            Valid (16);
        }
        catch (ArithmeticException e)
        {
            S.O.P (e);
        }
    }
}

```

**Output**

**Not Valid**

## 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");  
    }  
}
```

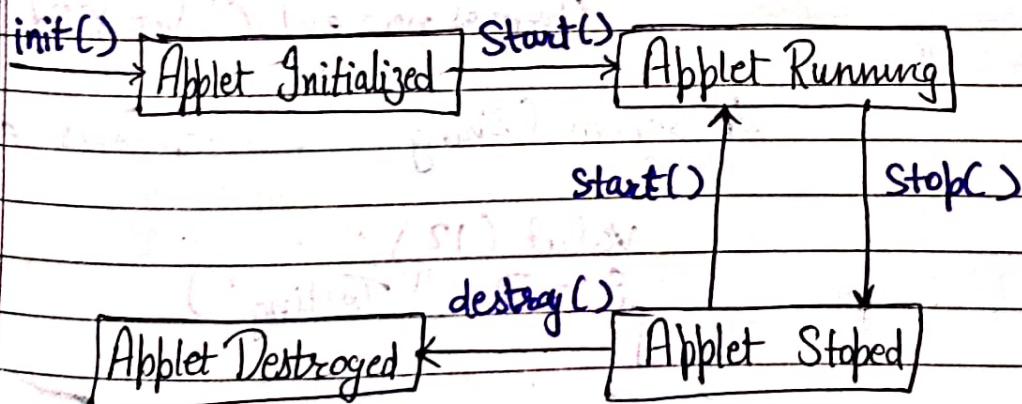
```
p.s.v.m (String a[]) throws ex1
```

```
{  
    valid (12);  
    S.O.P ("Testing")  
}
```

## 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



## How to create first Applet

```
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

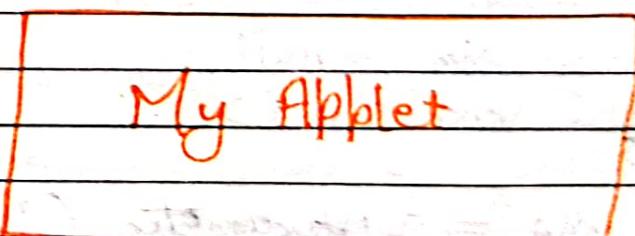
As this java code is save as app-image.class

### TML Code

```
<html>  
<title> Image Example</title>  
<applet code = "app-image-class" width=400 height=100>  
</applet>  
</html>
```

Save this file such as a.html

Open this file and the output will be



My Applet

## 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")
        }
    }
}
```

```

public void paint (Graphics g)
{
    g.drawString ("Strc , 100, 100)
}

```

## Getting I/P from User In Applet

### // Java Code

```

import java.applet.*;
import java.awt.*;
public class input extends Applet
{
    TextField t1, t2;
    public void init ()
    {
        t1 = new TextField (5);
        t2 = new TextField (5);
        add (t1);
        add (t2);
    }
}

```

### public void paint (Graphics)

```

{
    int x = 0;
    x = Integer.parseInt (t1.
        getText ());
    t2.setText (String.valueOf
        (x*x));
}

```

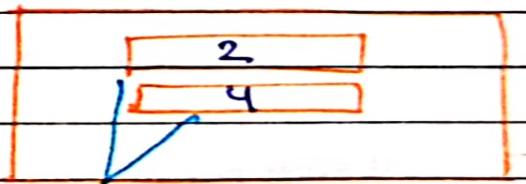
### // HTML Code

```

<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

## Benefits of Multithreading :-

Enable Programmers to do multiple task at one time

Programme

update user ( $t_1$ )

Print User Info ( $t_2$ )

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

Improved performance and concurrency.

Simultaneous access

applications.

6

6

6

## 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

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
{
```

```
b.s.v.m (String [] ar)
{
```

String s = "EEC";

S = S.concat("classes"); → EEC classes

S.O.P(S);

3

3

Output

EEC

Output

EEC classes

17

18

22

19

33

20

21

## 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
{
    b.s.v.m (String [] ar)
```

String s = "EEC";

s = s.concat(" classes"); → EEC classes  
System.out.println(s);

3

3 Output

EEC

Output

EEC classes

17

27

35

6

## 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.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 ~~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

## String Buffer class

Used to create mutable (modifiable) string.

### Methods :-

1) `StringBuffer sb = new StringBuffer("YouTube");`

2) `Append()`

`sb.append("classes");`

`S.O.P(sb);`

**YouTubeclasses**

3) `Insert()`

`sb.insert(3, "fire");`

**YouFireTube**

4) `Replace()`

`sb.replace(1, 2, "Shame")`

**YShameTube**

5) `delete()`

`sb.delete(1, 3)`

**Si S.i.**

6) ~~Reverse()~~

`sb.reverse();`

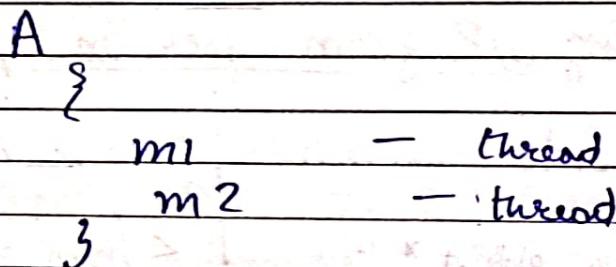
**Pallindrome**

## 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. |

## Multithreading :-

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



## 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 :-

- Light weight
- Address space is shared
- Interthread communication is cheap.

eg:- text editor — edit text  
— Paint



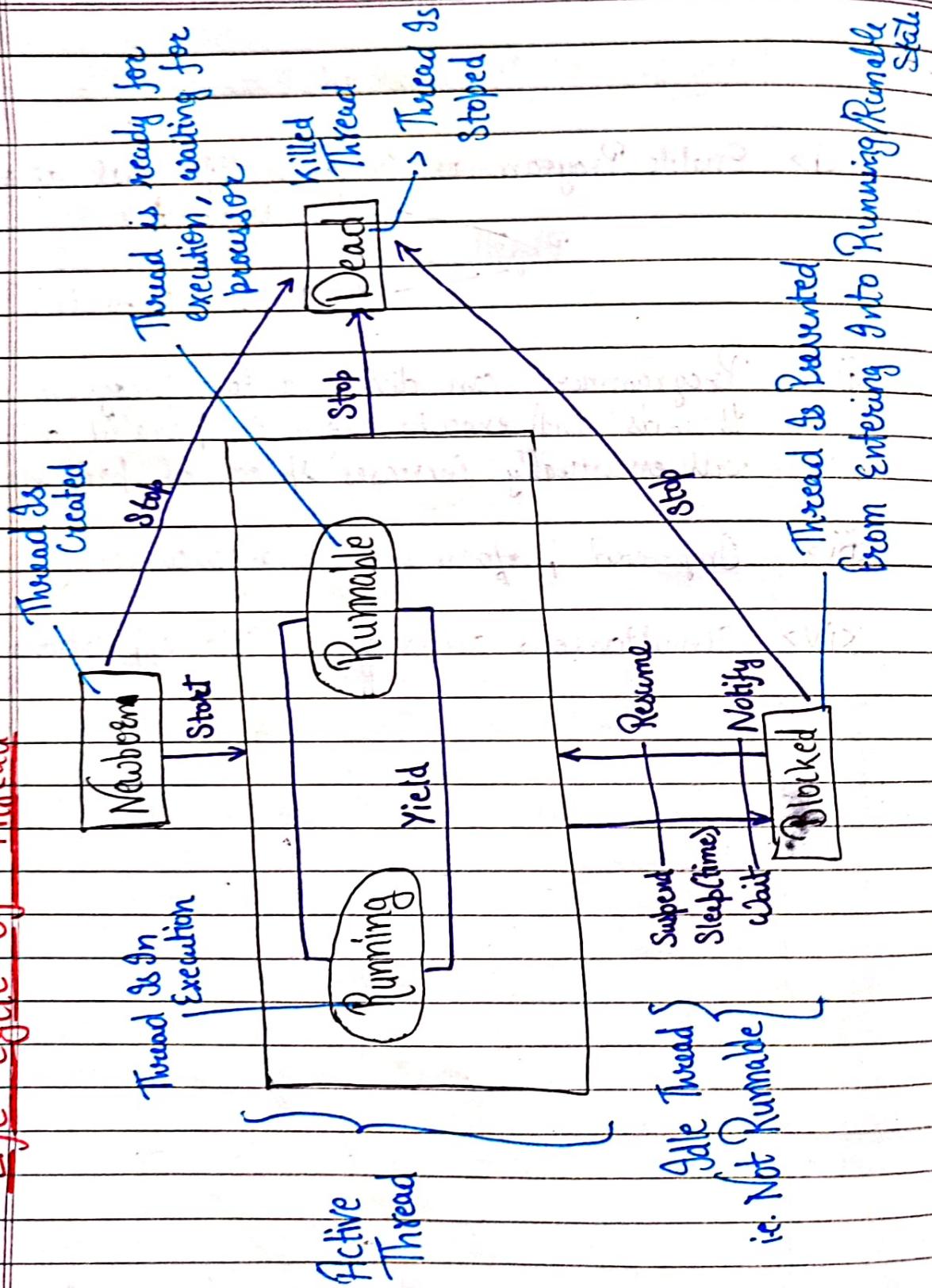
$t_1$

$t_2$

## Benefits of Multithreading:-

- (i) Enable Programmers to do multiple task at one time
  - update user ( $t_1$ )
  - Programmer
  - Print User Info ( $t_2$ )
- (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.

## Life Cycle Of Thread



## 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 t2 extends Thread.

Thread Class start

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

## 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)
```

Thread Gets Stop();

Terminated

```
S.O.P("A=" + i);
```

}

{

```
public static void main(String a[])
```

{

```
A a = new A()
```

```
a.start();
```

}

{

Output

A:1

A:

← 1000

A:

## 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)
    }
}
```

milliseconds

```
    S. O. P ("A" + i);
}
}
```

```
p. s. v. m (String a[])
{
    A a = new A();
    a. Start();
}
}
```

A a = new A();  
a. Start();

## Use of isAlive() and Join() Method :-

isAlive() method tests if the thread is alive.

[ True ] public final boolean isAlive()  
[ False ]

join() 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());

}

}

(t<sub>1</sub>)

(t<sub>2</sub>)

Join

Execution  
is Paused

t<sub>1</sub> complete

t<sub>2</sub> can  
start again

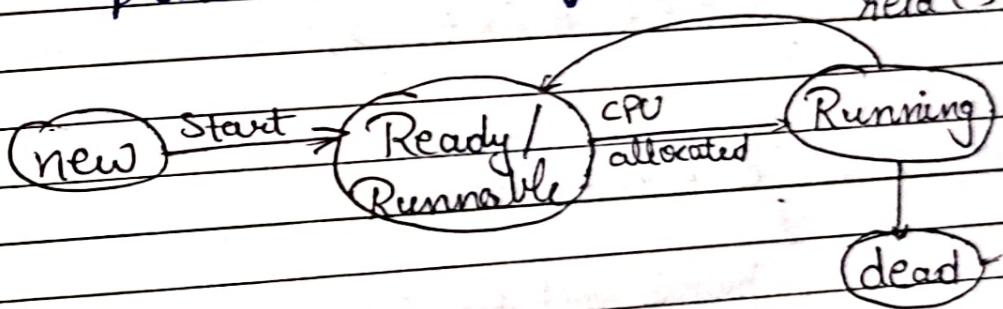
**Output**  
Status : True  
NewStatus : True

## 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");

{

3  
class B extends Thread

{

public void run()

{

for (int j=1; j<=5; j++)

{

S.O.P("B"+j);

{

S.O.P("B Exit");

{

3

Q  
Ans

class YieldTest

{  
p.s.r.m (String a[ ])

A a = new A();

B b = new B();

a.start();

b.start();

3

3

Can we start a thread twice?

No

class A extends Thread

{

public void run()

{  
S.O.P ("Thread A");

3

p.s.v.m (String [] ar)

{

A a1 = new A();

a1.start();

a1.start();

3

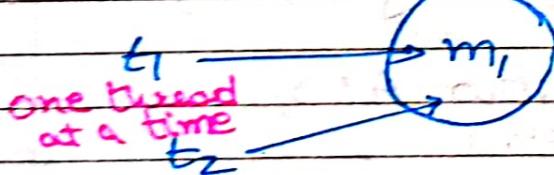
Output

Error :- Illegal Thread State  
Exception

## Synchronization :-

- ↳ Used to solve data inconsistency problem.
- ↳ Synchronization is the modifier applicable only for methods and blocks.
- ↳ Can't apply for classes and ~~object~~ 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)



## Program without Synchronization

class A  
{

Synchronized void address(int i)  
{

    Thread t = Thread.currentThread();

    for (int n = 1; n <= 5; n++)

        System.out.println(t.getName() + " - " + (i + n));

}

class B extends Thread

{

    A a1 = new A();

    public void run()

{

        a1.address(100);

}

O/P without Synchronization

T1-101

T2-101

T2-102

T1-102

class Syntext

{ p.s.v.m (String a[]) } O/P with Synch.

{ B b = new B(); }

T1-101

Thread t1 = new Thread(b);

T1-102

Thread t2 = new Thread(b);

T1-103

t1.setName("T1");

T1-104

t2.setName("T2");

T1-105

t1.start();

T2-101

t2.start();

T2-102

3 3

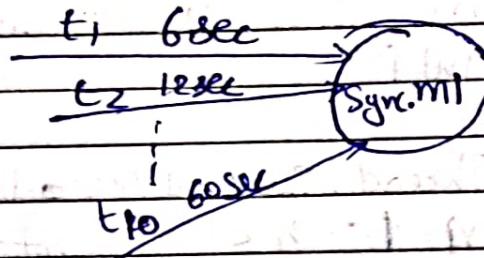
T2-103

T2-104

T2-105

## Problem of Synchronization

↳ Increases waiting time



↳ Effect the performance of the system

### S.No Method with description

1) public void start()

Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.

2) public void run()

If this Thread object instantiated using a separate 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.

4> public final void setPriority (int priority)

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

5> public final 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 ()

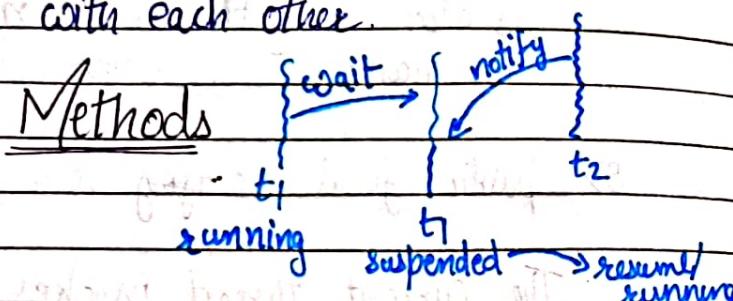
Interrupts this thread, causing it to continue execution if it was blocked for any reason.

7> public final boolean isAlive ()

Returns true if the thread is alive.

## Interthread Communication

Used to allow synchronized threads to communicate with each other.



17) Wait :-

Causes current thread to release the lock and wait until

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

public void wait()

23 notify :-

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

public void notify()

37) notify All :-

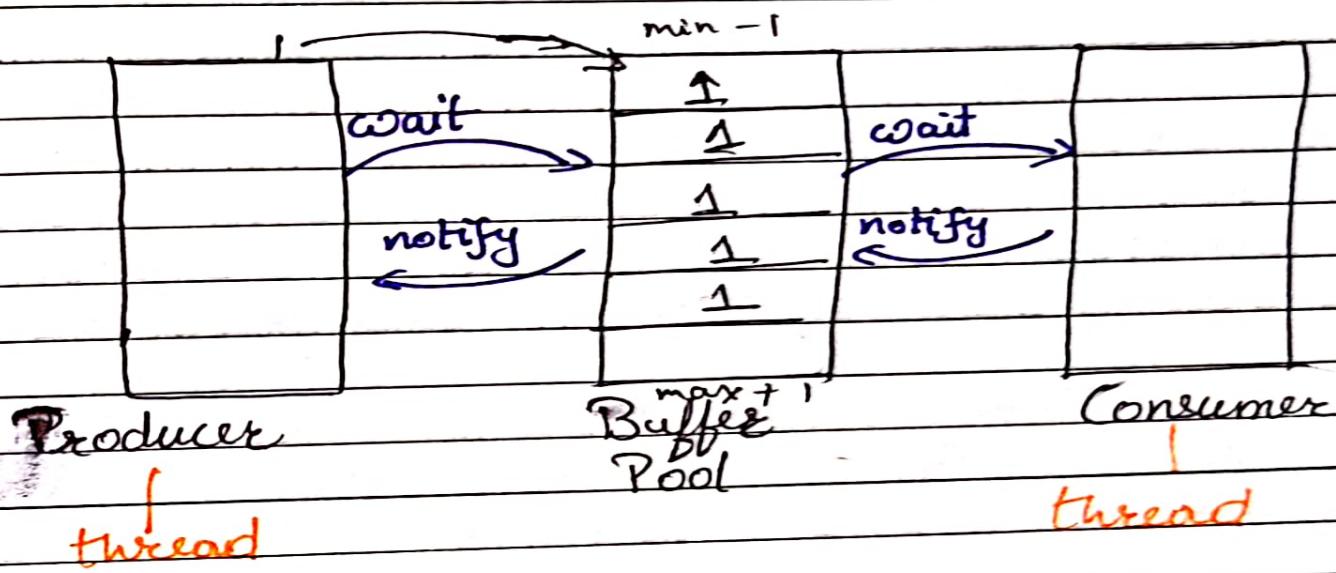
Wakes up all the threads waiting -

public void notifyAll ()

## 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.

6

6

6

## Java AWT Programming

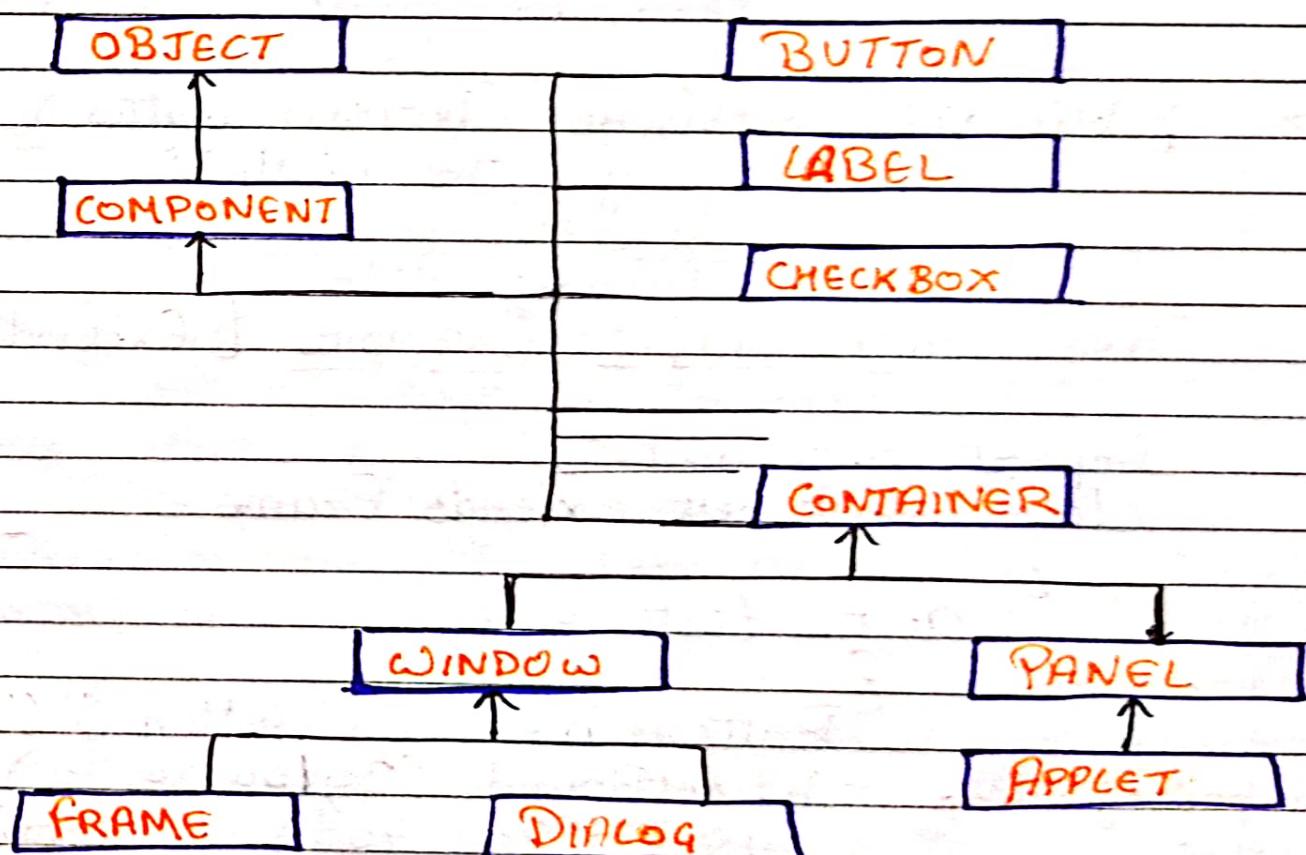
AWT - Abstract Window Toolkit

Used to develop GUI or window based application.

AWT components are Platform dependent.

java.awt package is used.

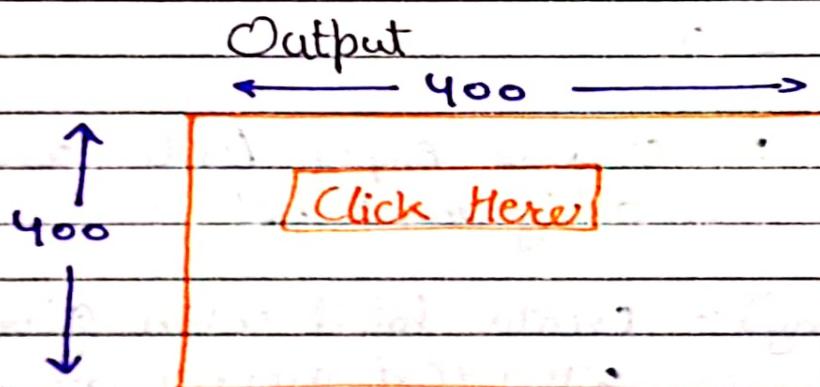
### Hierarchy -



```
public static void main (String [] ar)
{
```

```
    awt_test a1 = new awt_test ();
```

3



## 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.

## ~~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);
        setSize (400,400);
        setLayout (null);
        setVisible (true);
    }
}

```

Set the pos<sup>n</sup> of Button      // Frame Size  
                                   // By default FlowLayout

```
public static void main (String [] ar)
```

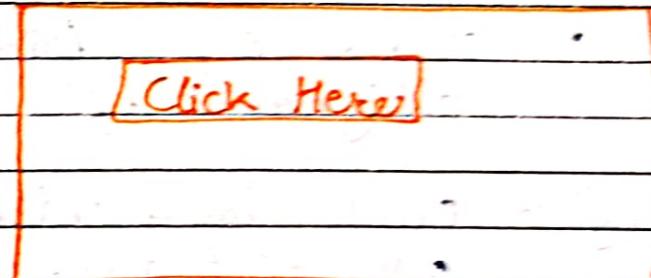
{

```
    cwt test a1 = new aut test ();
```

3

3

Output

 $\leftarrow 400 \longrightarrow$ 
 $\begin{array}{c} \uparrow \\ 400 \\ \downarrow \end{array}$ 


## 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 are 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.

①

Labels :-

- ↳ Object of type `Label`
- ↳ contains a string which it displays.
- ↳ do not support any interaction with the user.

Constructors :-

`Label()` :- creates empty label with its text alignment `left`.

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

`Label(String, int)` :- creates label with given string and with given alignment.

Available Alignment :-

`Label.left`

`Label.right`

`Label.center`

Example

```

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

public class LabelDemo extends Applet
{
    public void init () // init always uses
    {
        Label one = new Label ("One", Label.RIGHT);
        one.setBackground (Color.red); // for background color
        Label two = new Label ();;
        two.setBackground (Color.blue);
        two.setText ("Two");
        Label three = new Label ("Three")
        three.setForeground (Color.orange); // for character color
        Label four = new Label ();
        four.setText (three.getText ());
        add (one);
        add (two);
        add (three);
        add (four);
    }
}

```

Output



Three

Three

Ques 1.  $\frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$   $\frac{1}{4} \times \frac{1}{2} = \frac{1}{8}$   $\frac{1}{8} \times \frac{1}{2} = \frac{1}{16}$   $\frac{1}{16} \times \frac{1}{2} = \frac{1}{32}$   $\frac{1}{32} \times \frac{1}{2} = \frac{1}{64}$   $\frac{1}{64} \times \frac{1}{2} = \frac{1}{128}$   $\frac{1}{128} \times \frac{1}{2} = \frac{1}{256}$   $\frac{1}{256} \times \frac{1}{2} = \frac{1}{512}$   $\frac{1}{512} \times \frac{1}{2} = \frac{1}{1024}$   $\frac{1}{1024} \times \frac{1}{2} = \frac{1}{2048}$   $\frac{1}{2048} \times \frac{1}{2} = \frac{1}{4096}$   $\frac{1}{4096} \times \frac{1}{2} = \frac{1}{8192}$   $\frac{1}{8192} \times \frac{1}{2} = \frac{1}{16384}$   $\frac{1}{16384} \times \frac{1}{2} = \frac{1}{32768}$   $\frac{1}{32768} \times \frac{1}{2} = \frac{1}{65536}$   $\frac{1}{65536} \times \frac{1}{2} = \frac{1}{131072}$   $\frac{1}{131072} \times \frac{1}{2} = \frac{1}{262144}$   $\frac{1}{262144} \times \frac{1}{2} = \frac{1}{524288}$   $\frac{1}{524288} \times \frac{1}{2} = \frac{1}{1048576}$   $\frac{1}{1048576} \times \frac{1}{2} = \frac{1}{2097152}$   $\frac{1}{2097152} \times \frac{1}{2} = \frac{1}{4194304}$   $\frac{1}{4194304} \times \frac{1}{2} = \frac{1}{8388608}$   $\frac{1}{8388608} \times \frac{1}{2} = \frac{1}{16777216}$   $\frac{1}{16777216} \times \frac{1}{2} = \frac{1}{33554432}$   $\frac{1}{33554432} \times \frac{1}{2} = \frac{1}{67108864}$   $\frac{1}{67108864} \times \frac{1}{2} = \frac{1}{134217728}$   $\frac{1}{134217728} \times \frac{1}{2} = \frac{1}{268435456}$   $\frac{1}{268435456} \times \frac{1}{2} = \frac{1}{536870912}$   $\frac{1}{536870912} \times \frac{1}{2} = \frac{1}{1073741824}$   $\frac{1}{1073741824} \times \frac{1}{2} = \frac{1}{2147483648}$   $\frac{1}{2147483648} \times \frac{1}{2} = \frac{1}{4294967296}$   $\frac{1}{4294967296} \times \frac{1}{2} = \frac{1}{8589934592}$   $\frac{1}{8589934592} \times \frac{1}{2} = \frac{1}{17179869184}$   $\frac{1}{17179869184} \times \frac{1}{2} = \frac{1}{34359738368}$   $\frac{1}{34359738368} \times \frac{1}{2} = \frac{1}{68719476736}$   $\frac{1}{68719476736} \times \frac{1}{2} = \frac{1}{137438953472}$   $\frac{1}{137438953472} \times \frac{1}{2} = \frac{1}{274877906944}$   $\frac{1}{274877906944} \times \frac{1}{2} = \frac{1}{549755813888}$   $\frac{1}{549755813888} \times \frac{1}{2} = \frac{1}{1099511627776}$   $\frac{1}{1099511627776} \times \frac{1}{2} = \frac{1}{2199023255552}$   $\frac{1}{2199023255552} \times \frac{1}{2} = \frac{1}{4398046511104}$   $\frac{1}{4398046511104} \times \frac{1}{2} = \frac{1}{8796093022208}$   $\frac{1}{8796093022208} \times \frac{1}{2} = \frac{1}{17592186044416}$   $\frac{1}{17592186044416} \times \frac{1}{2} = \frac{1}{35184372088832}$   $\frac{1}{35184372088832} \times \frac{1}{2} = \frac{1}{70368744177664}$   $\frac{1}{70368744177664} \times \frac{1}{2} = \frac{1}{140737488355328}$   $\frac{1}{140737488355328} \times \frac{1}{2} = \frac{1}{281474976710656}$   $\frac{1}{281474976710656} \times \frac{1}{2} = \frac{1}{562949953421312}$   $\frac{1}{562949953421312} \times \frac{1}{2} = \frac{1}{1125899906842624}$   $\frac{1}{1125899906842624} \times \frac{1}{2} = \frac{1}{2251799813685248}$   $\frac{1}{2251799813685248} \times \frac{1}{2} = \frac{1}{4503599627370496}$   $\frac{1}{4503599627370496} \times \frac{1}{2} = \frac{1}{9007199254740992}$   $\frac{1}{9007199254740992} \times \frac{1}{2} = \frac{1}{18014398509481984}$   $\frac{1}{18014398509481984} \times \frac{1}{2} = \frac{1}{36028797018963968}$   $\frac{1}{36028797018963968} \times \frac{1}{2} = \frac{1}{72057594037927936}$   $\frac{1}{72057594037927936} \times \frac{1}{2} = \frac{1}{144115188075855872}$   $\frac{1}{144115188075855872} \times \frac{1}{2} = \frac{1}{288230376151711744}$   $\frac{1}{288230376151711744} \times \frac{1}{2} = \frac{1}{576460752303423488}$   $\frac{1}{576460752303423488} \times \frac{1}{2} = \frac{1}{1152921504606846976}$   $\frac{1}{1152921504606846976} \times \frac{1}{2} = \frac{1}{2305843009213693952}$   $\frac{1}{2305843009213693952} \times \frac{1}{2} = \frac{1}{4611686018427387904}$   $\frac{1}{4611686018427387904} \times \frac{1}{2} = \frac{1}{9223372036854775808}$   $\frac{1}{9223372036854775808} \times \frac{1}{2} = \frac{1}{18446744073709551616}$   $\frac{1}{18446744073709551616} \times \frac{1}{2} = \frac{1}{36893488147419103232}$   $\frac{1}{36893488147419103232} \times \frac{1}{2} = \frac{1}{73786976294838206464}$   $\frac{1}{73786976294838206464} \times \frac{1}{2} = \frac{1}{147573952589676412928}$   $\frac{1}{147573952589676412928} \times \frac{1}{2} = \frac{1}{295147905179352825856}$   $\frac{1}{295147905179352825856} \times \frac{1}{2} = \frac{1}{590295810358705651712}$   $\frac{1}{590295810358705651712} \times \frac{1}{2} = \frac{1}{1180591620717411303424}$   $\frac{1}{1180591620717411303424} \times \frac{1}{2} = \frac{1}{2361183241434822606848}$   $\frac{1}{2361183241434822606848} \times \frac{1}{2} = \frac{1}{4722366482869645213696}$   $\frac{1}{4722366482869645213696} \times \frac{1}{2} = \frac{1}{9444732965739290427392}$   $\frac{1}{9444732965739290427392} \times \frac{1}{2} = \frac{1}{18889465931478580854784}$   $\frac{1}{18889465931478580854784} \times \frac{1}{2} = \frac{1}{37778931862957161709568}$   $\frac{1}{37778931862957161709568} \times \frac{1}{2} = \frac{1}{75557863725914323419136}$   $\frac{1}{75557863725914323419136} \times \frac{1}{2} = \frac{1}{151115727451828646838272}$   $\frac{1}{151115727451828646838272} \times \frac{1}{2} = \frac{1}{302231454903657293676544}$   $\frac{1}{302231454903657293676544} \times \frac{1}{2} = \frac{1}{604462909807314587353088}$   $\frac{1}{604462909807314587353088} \times \frac{1}{2} = \frac{1}{1208925819614629174706176}$   $\frac{1}{1208925819614629174706176} \times \frac{1}{2} = \frac{1}{2417851639229258349412352}$   $\frac{1}{2417851639229258349412352} \times \frac{1}{2} = \frac{1}{4835703278458516698824704}$   $\frac{1}{4835703278458516698824704} \times \frac{1}{2} = \frac{1}{9671406556917033397649408}$   $\frac{1}{9671406556917033397649408} \times \frac{1}{2} = \frac{1}{19342813113834066795298816}$   $\frac{1}{19342813113834066795298816} \times \frac{1}{2} = \frac{1}{38685626227668133590597632}$   $\frac{1}{38685626227668133590597632} \times \frac{1}{2} = \frac{1}{77371252455336267181195264}$   $\frac{1}{77371252455336267181195264} \times \frac{1}{2} = \frac{1}{154742504910672534362390528}$   $\frac{1}{154742504910672534362390528} \times \frac{1}{2} = \frac{1}{309485009821345068724781056}$   $\frac{1}{309485009821345068724781056} \times \frac{1}{2} = \frac{1}{618970019642690137449562112}$   $\frac{1}{618970019642690137449562112} \times \frac{1}{2} = \frac{1}{1237940039285380274899124224}$   $\frac{1}{1237940039285380274899124224} \times \frac{1}{2} = \frac{1}{2475880078570760549798248448}$   $\frac{1}{2475880078570760549798248448} \times \frac{1}{2} = \frac{1}{4951760157141521099596496896}$   $\frac{1}{4951760157141521099596496896} \times \frac{1}{2} = \frac{1}{9903520314283042199192993792}$   $\frac{1}{9903520314283042199192993792} \times \frac{1}{2} = \frac{1}{19807040628566084398385987584}$   $\frac{1}{19807040628566084398385987584} \times \frac{1}{2} = \frac{1}{39614081257132168796771975168}$   $\frac{1}{39614081257132168796771975168} \times \frac{1}{2} = \frac{1}{79228162514264337593543950336}$   $\frac{1}{79228162514264337593543950336} \times \frac{1}{2} = \frac{1}{158456325028528675187087900672}$   $\frac{1}{158456325028528675187087900672} \times \frac{1}{2} = \frac{1}{316912650057057350374175801344}$   $\frac{1}{316912650057057350374175801344} \times \frac{1}{2} = \frac{1}{633825300114114700748351602688}$   $\frac{1}{633825300114114700748351602688} \times \frac{1}{2} = \frac{1}{1267650600228229401496703205376}$   $\frac{1}{1267650600228229401496703205376} \times \frac{1}{2} = \frac{1}{2535301200456458802993406410752}$   $\frac{1}{2535301200456458802993406410752} \times \frac{1}{2} = \frac{1}{5070602400912917605986812821504}$   $\frac{1}{5070602400912917605986812821504} \times \frac{1}{2} = \frac{1}{10141204801825835211973625643008}$   $\frac{1}{10141204801825835211973625643008} \times \frac{1}{2} = \frac{1}{20282409603651670423947251286016}$   $\frac{1}{20282409603651670423947251286016} \times \frac{1}{2} = \frac{1}{40564819207303340847894502572032}$   $\frac{1}{40564819207303340847894502572032} \times \frac{1}{2} = \frac{1}{81129638414606681695789005144064}$   $\frac{1}{81129638414606681695789005144064} \times \frac{1}{2} = \frac{1}{162259276829213363391578010288128}$   $\frac{1}{162259276829213363391578010288128} \times \frac{1}{2} = \frac{1}{324518553658426726783156020576256}$   $\frac{1}{324518553658426726783156020576256} \times \frac{1}{2} = \frac{1}{649037107316853453566312041152512}$   $\frac{1}{649037107316853453566312041152512} \times \frac{1}{2} = \frac{1}{1298074214633706907132624082305024}$   $\frac{1}{1298074214633706907132624082305024} \times \frac{1}{2} = \frac{1}{2596148429267413814265248164610048}$   $\frac{1}{2596148429267413814265248164610048} \times \frac{1}{2} = \frac{1}{5192296858534827628530496329220096}$   $\frac{1}{5192296858534827628530496329220096} \times \frac{1}{2} = \frac{1}{10384593717069655257060992658440192}$   $\frac{1}{10384593717069655257060992658440192} \times \frac{1}{2} = \frac{1}{20769187434139310514121985316880384}$   $\frac{1}{20769187434139310514121985316880384} \times \frac{1}{2} = \frac{1}{41538374868278621028243970633760768}$   $\frac{1}{41538374868278621028243970633760768} \times \frac{1}{2} = \frac{1}{83076749736557242056487941267521536}$   $\frac{1}{83076749736557242056487941267521536} \times \frac{1}{2} = \frac{1}{166153499473114484112958822535043072}$   $\frac{1}{166153499473114484112958822535043072} \times \frac{1}{2} = \frac{1}{332306998946228968225917645070086144}$   $\frac{1}{332306998946228968225917645070086144} \times \frac{1}{2} = \frac{1}{664613997892457936451835290140172288}$   $\frac{1}{664613997892457936451835290140172288} \times \frac{1}{2} = \frac{1}{1329227995784915872903670580280344576}$   $\frac{1}{1329227995784915872903670580280344576} \times \frac{1}{2} = \frac{1}{2658455991569831745807341160560689152}$   $\frac{1}{2658455991569831745807341160560689152} \times \frac{1}{2} = \frac{1}{5316911983139663491614682321121378304}$   $\frac{1}{5316911983139663491614682321121378304} \times \frac{1}{2} = \frac{1}{1063382396627932698322936464224276608}$   $\frac{1}{1063382396627932698322936464224276608} \times \frac{1}{2} = \frac{1}{2126764793255865396645872928448553216}$   $\frac{1}{2126764793255865396645872928448553216} \times \frac{1}{2} = \frac{1}{4253529586511730793291745856897106432}$   $\frac{1}{4253529586511730793291745856897106432} \times \frac{1}{2} = \frac{1}{8507059173023461586583491713794212864}$   $\frac{1}{8507059173023461586583491713794212864} \times \frac{1}{2} = \frac{1}{17014118346046923173166983427588425728}$   $\frac{1}{17014118346046923173166983427588425728} \times \frac{1}{2} = \frac{1}{34028236692093846346333966855176851456}$   $\frac{1}{34028236692093846346333966855176851456} \times \frac{1}{2} = \frac{1}{68056473384187692692667933710353702912}$   $\frac{1}{68056473384187692692667933710353702912} \times \frac{1}{2} = \frac{1}{136112946768375385385335867420707405824}$   $\frac{1}{136112946768375385385335867420707405824} \times \frac{1}{2} = \frac{1}{272225893536750770770671734841414811648}$   $\frac{1}{272225893536750770770671734841414811648} \times \frac{1}{2} = \frac{1}{544451787073501541541343469682829623296}$   $\frac{1}{544451787073501541541343469682829623296} \times \frac{1}{2} = \frac{1}{1088903574147003083082686939365659246592}$   $\frac{1}{1088903574147003083082686939365659246592} \times \frac{1}{2} = \frac{1}{2177807148294006166165373878731318493184}$   $\frac{1}{2177807148294006166165373878731318493184} \times \frac{1}{2} = \frac{1}{4355614296588012332325747757462636986368}$   $\frac{1}{4355614296588012332325747757462636986368} \times \frac{1}{2} = \frac{1}{8711228593176024664651495514925273972736}$   $\frac{1}{8711228593176024664651495514925273972736} \times \frac{1}{2} = \frac{1}{17422457186352049329302991029850547945472}$   $\frac{1}{17422457186352049329302991029850547945472} \times \frac{1}{2} = \frac{1}{34844914372704098658605982059701095890944}$   $\frac{1}{34844914372704098658605982059701095890944} \times \frac{1}{2} = \frac{1}{69689828745408197317211964119402191781888}$   $\frac{1}{69689828745408197317211964119402191781888} \times \frac{1}{2} = \frac{1}{139379657490816394634423928238804383563776}$   $\frac{1}{139379657490816394634423928238804383563776} \times \frac{1}{2} = \frac{1}{278759314981632789268847856477608767127552}$   $\frac{1}{278759314981632789268847856477608767127552} \times \frac{1}{2} = \frac{1}{55751862996326557853769571295521753425508}$   $\frac{1}{55751862996326557853769571295521753425508} \times \frac{1}{2} = \frac{1}{111503725992653115707539142910423506850176}$   $\frac{1}{111503725992653115707539142910423506850176} \times \frac{1}{2} = \frac{1}{223007451985306231415078285820847013700352}$   $\frac{1}{223007451985306231415078285820847013700352} \times \frac{1}{2} = \frac{1}{446014903970612462830156571641694027400704}$   $\frac{1}{446014903970612462830156571641694027400704} \times \frac{1}{2} = \frac{1}{892029807941224925660313143283388054801408}$   $\frac{1}{892029807941224925660313143283388054801408} \times \frac{1}{2} = \frac{1}{1784059615882449851320626286566776109602816}$   $\frac{1}{1784059615882449851320626286566776109602816} \times \frac{1}{2} = \frac{1}{3568119231764899702641252573133552219205632}$   $\frac{1}{3568119231764899702641252573133552219205632} \times \frac{1}{2} = \frac{1}{7136238463529799405282505146267104438411264}$   $\frac{1}{7136238463529799405282505146267104438411264} \times \frac{1}{2} = \frac{1}{14272476927059598810565010292534208868224128}$   $\frac{1}{14272476927059598810565010292534208868224128} \times \frac{1}{2} = \frac{1}{28544953854119197621130020585068417726448256}$   $\frac{1}{28544953854119197621130020585068417726448256} \times \frac{1}{2} = \frac{1}{57089827708238395242260040170136835452896512}$   $\frac{1}{57089827708238395242260040170136835452896512} \times \frac{1}{2} = \frac{1}{114179655416476790484520080340273670905792024}$   $\frac{1}{114179655416476790484520080340273670905$

## (2) Buttons :-

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

Button ( )

Button (String)

Contains a label and generates an even 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

|     |       |      |
|-----|-------|------|
| Red | Green | Blue |
|-----|-------|------|

3

③ Checkbox :-

(control used to turn option on or off)

checkbox()

checkbox(String str)

checkbox(String str, boolean on)

In this we can choose ~~one~~ 1 or more options.

Example :-

import java.awt.\*;

import java.applet.\*;

public class checkboxDemo extends Applet

checkbox winXP, winVista, mac; //initializing checkbox  
public void init()

winXP = new Checkbox ("Window xp", true);  
winVista = new Checkbox ("Window Vista", false);  
mac = new Checkbox ("Mac Os");

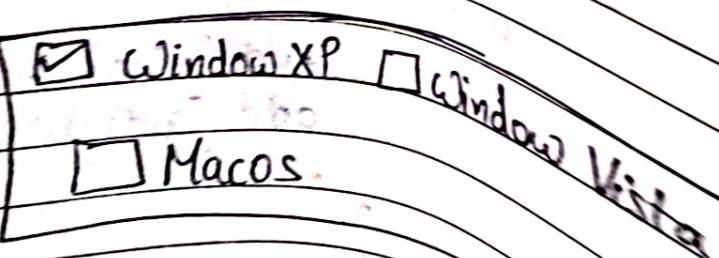
Output

add(winXP);

add(winVista);

add(mac);

}



## 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.

### CheckboxGroup()

Constructor of checkbox class used for creating radio buttons.

#### Example

Checkbox(String str, boolean on, CheckboxGroup  
or  
↳ a cbgrp)

public class CBGroup extends Applet

{

checkbox winXP, winVista;

CheckboxGroup cbg;

public void init()

{

cbg = new CheckboxGroup();

winXP = new Checkbox("Windows XP", cbg, true);

winVista = new Checkbox("Windows Vista", cbg, false);

add(winXP);

add(winVista);

}

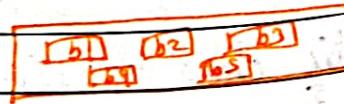
Output

① Windows XP ② 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 component's <sup>button</sup> as its preferred size in Java.
- ↳ It also sets the width of the button/according component to its content.
- ↳ 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

Button 1

Button 2

Button 3

## 2) Border Layout :-



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

BorderLayout () :- Creates a border layout but with no gaps b/w the components

JBorderLayout (int hgap, int vgap) :-

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

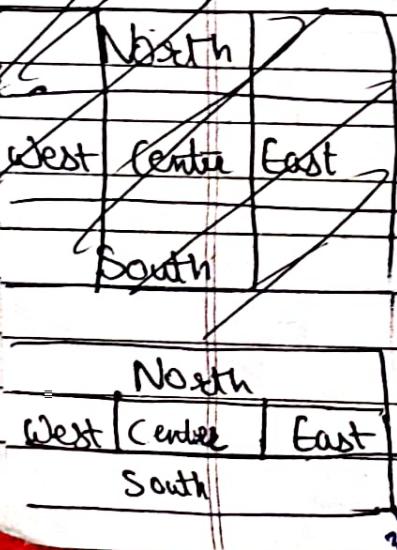
Ex:-

```
import java.awt.*;
public class Border extends Frame
{
```

Border ()

{

Output



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");

NORTH

add (b1, BorderLayout. NORTH);

add (b2, BorderLayout. SOUTH);

add (b3, BorderLayout. WEST);

add (b4, BorderLayout. EAST);

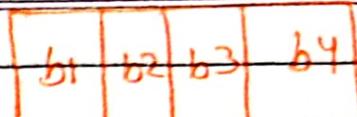
add (b5, BorderLayout. CENTER);

3 3 3 3 3

Border b = new Border();



## 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 divided into the form of rows and columns.
- ↳ In grid layout, all the components have same size.

Ex:-

public class GridLayout<sup>sample</sup> extends JFrame

{ JButton b1 = new JButton ("B1") }

JButton b2 = new JButton ("B2")

JButton b3 = new JButton ("B3")

JButton b4 = new JButton ("B4")

public GridLayout<sup>sample</sup> ()

GridLayout g = new GridLayout (),

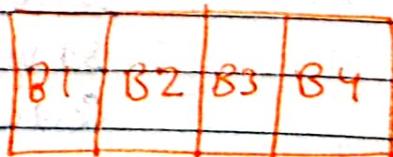
getLayout().setLayout (g),

add (b1),

add (b2),

add (b3),

add (b4),



public static void main ()

{ GridLayoutSample g = new GridLayout (2, 2),

getLayout (g);

B1 B2

## 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");
    JButton btn2 = " " ("B2");
    JButton btn3 = " " ("B3");
    JButton btn4 = " " ("B4");
    JButton btn5 = " " ("B5");
  
```

```

public Mylayout()
  
```

```

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

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

GridBagConstraints gbc = new GridBagConstraints();

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

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

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

| Grid layout Example |    |    |    |
|---------------------|----|----|----|
| B1                  | B2 | B3 | B4 |
|                     |    |    |    |

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();
```

CardLayout card = new CardLayout();

c. setLayout (CardLayout)

c. setLayout (card);

JButton B1 = new JButton ("Page1");

" B2 = " " ("Page 2"),

" B3 = " " ("Page3"),

" B4 = " " ("Page4"),

Container

ID

c. add (B1, "1");

c. add (B2, "2");

c. add (B3, "3");

c. add (B4, "4");

}

}