Java

Unit-I

Overview

→ Java the new programming language developed by sun microsystems in 1991.

→ Invented by the green team. One of the inventor was "James Gosling".

→ Originally java was called OOK later called HOT- Java.

→ In 1995, Internet & web was just emerging so, sun turned it into a language of Internet programming.

→ Java was not designed for Internet. The objective behind developement of java was to create a common developement environment for consumer electronic devices which was easily portable from one device to another.

Advantage of Java over C or C++

→ C & C++ (and most other languages) they are designed to be compiled for a specific target.

Date :
Page No.

→ Compilers are expensive and time-cosuming to create.

→ But, Java is platform Independent ie, it could be used to produce code that would run on a variety of CPU's under differing environment.

Eg of Javac

```
class Simple                                    // class
{
command line argument  public static void main (String args[])
    System.Out.Println("Hello Java");
}
```

Package.class.method.

Compile: Javac Simple.Java
Run: Java Simple.

(main is the first method that is seen).

Date :
Page No.

Use of Java (Application)

(c), Desktop Application : such as acrobat reader, media player, antivirus etc,

(2), Web - application ⇒ Javatpoint.com etc

(3), Enterprise application ⇒ Banking applications.

(4) Robotics

(5), Games

(6), Smartcard etc

(7), Embedded Systems.

Types of Java Applications ⇒

(1), Stand - alone application ⇒

→ Stand - alone or desktop application.

→ An application that we need to install on any machine such as media player

→ AWT and swings are used to create Stand - alone application.

→ eg ⇒ media player etc.

Date :
Page No.

② Web - Application :

→ An application that will produce dynamic pages and run on the server side is called a web-application.

→ Eg; Javatpoint.com,

→ Servlet and Jsp are used to create web-application.

③. Enterprise Application :

→ An application that is used in distributed system such as banking application (System).

→ EJB is used to create Enterprise Application.

④. Mobile Applications :

→ An application that will run on any mobile is called mobile application

→ Such as whatsapp etc

→ Android. JME used to create mobile applications.

Date :
Page No.

Characteristics of Java

1. Java is Simple
2. Object - oriented
3. Distributed
4. Architecture-neutral
5. portable
6. Robust
7. secure
8. dynamic
9. High - performance
10. platform Independent
11. Interpreted
12. Multithreaded

Java is Simple.

→ Java has no pointers and has automatic garbage collector.

→ Java is partially modeled on C++ but is quite simple than C++

→ Java is C++ - -++.

It is like C++ language with more functionality and lesser negative aspects.

→ It fixes some of clumpy feature of C++.

Date :
Page No.

Java is Object - Oriented.

→ Java is strictly object-oriented.
→ An Object oriented language provides greater flexibility, reusability and modularity through Encapsulation, Inheritance and polymorphism.
→ Reusability is the central issue in software development.

Java is distributed.
Java program on compilation produces an output which is called Byte-code. Now, this bytecode can run on any machine with a JVM (Interpreter) in it. Java is distributed as anyone can have access to thousands of program with internet.
→ EJB is used to create distributed programs in Java.

Java is Architectural Neutral.
Neutral means change. In architecture, design will doesn't have any effect on the execution of program.

Date :
Page No.

Java follows "Write Once Run anywhere" notion. With a Java virtual machine, you can write any program that will run on any platform.

Java is portable.
Java is portable because it is Architectural- neutral and distributed. Java programs can be run on any platform with JVM installed.

Java is Robust.
Robust means strong.
Java is a strong language as it uses high memory management.
It has Run-time Exception handling feature.
All these points makes Java Robust.
Automatic garbage collector

Date :
Page No.

Java is secure.
Java has multilevel system of security.
Bytecode verifier and security managers adds security to java programs.
Java creates fire-wall between java programs and remaining of the computer system that neither allows computer system to interfere with java program nor does it allow java program to interfere with remaining of computer system.

Java is dynamic.
One can access thousands of java program and other computers using Internet, Java loads dynamic pages when it is taken on screen side.

Java's High Performance.
Java has high performance as it is architectural neutral and portable.

Date :
Page No.

Java is platform Independent.

platform is any hardware or software environment that is needed to run a program.
Java compilation provides bytecode which is neutral. Java has its own run-time Environment (JRE) and API, Hence It is platform Independent.

Java is Interpreted.

You need an Interpreter to run Java program. Java program produces Bytecode on compilation in JVM. This Bytecode is platform Independent and can be run on any machine which has Interpreter Installed in it.

Java is multithreaded.

Multithreading is integrated smoothly in Java. Whereas in other languages we have to call procedures specific to the operating system.

Date :
Page No.

Java Environment.

→ Java environment includes a large number of development tools and hundreds of classes and methods.

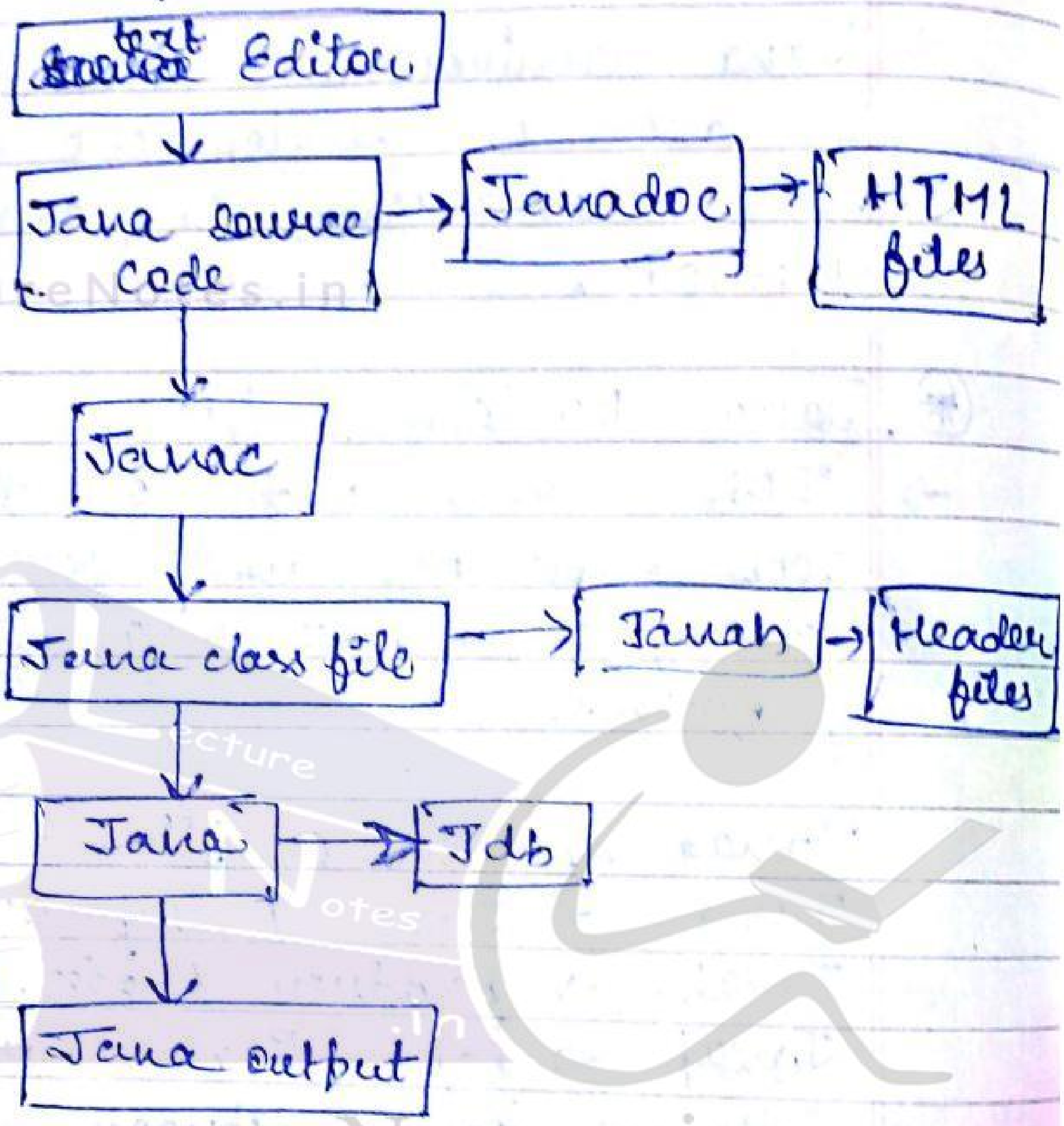
Java Development Kit

→ JDK comes with collection of tools that are used for developing and running java programs.

Javac → Java compiler.
Java → Java interpreter
Javah → produces header files
Javap → java disassembler
Jdb → java debugger.
Applet viewer → Enables to run java applets.
Javadoc → creates HTML format documentation.

Date :
Page No.

Flow of Java program.



Explain Each - One

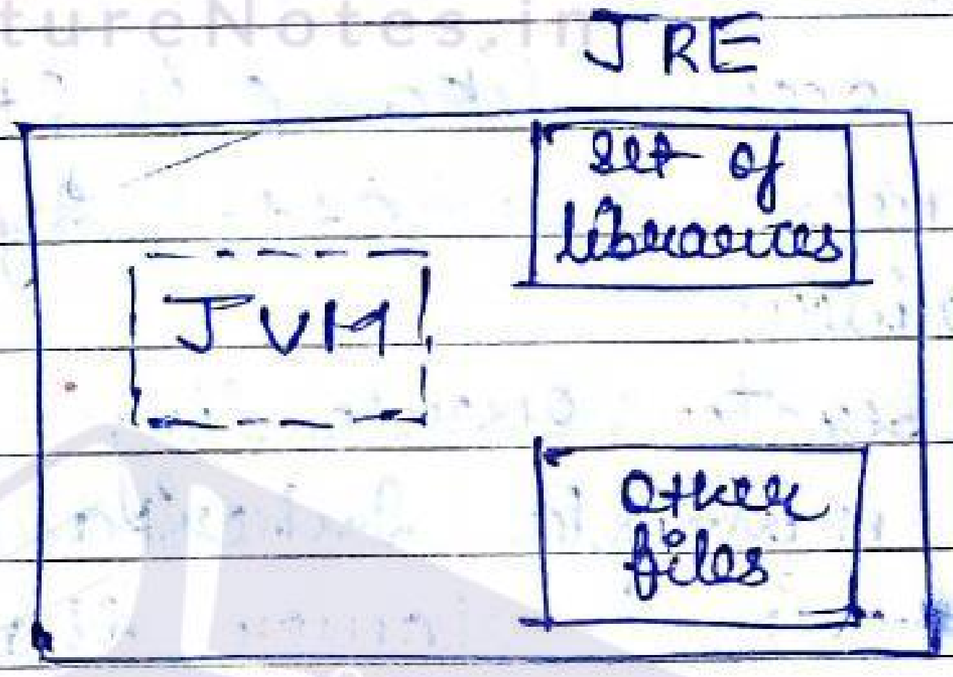
JDK, JRE and JVM

JDK → Java Development tool kit
JRE → Java run-time Environ-ment
JVM → Java Virtual Machine.

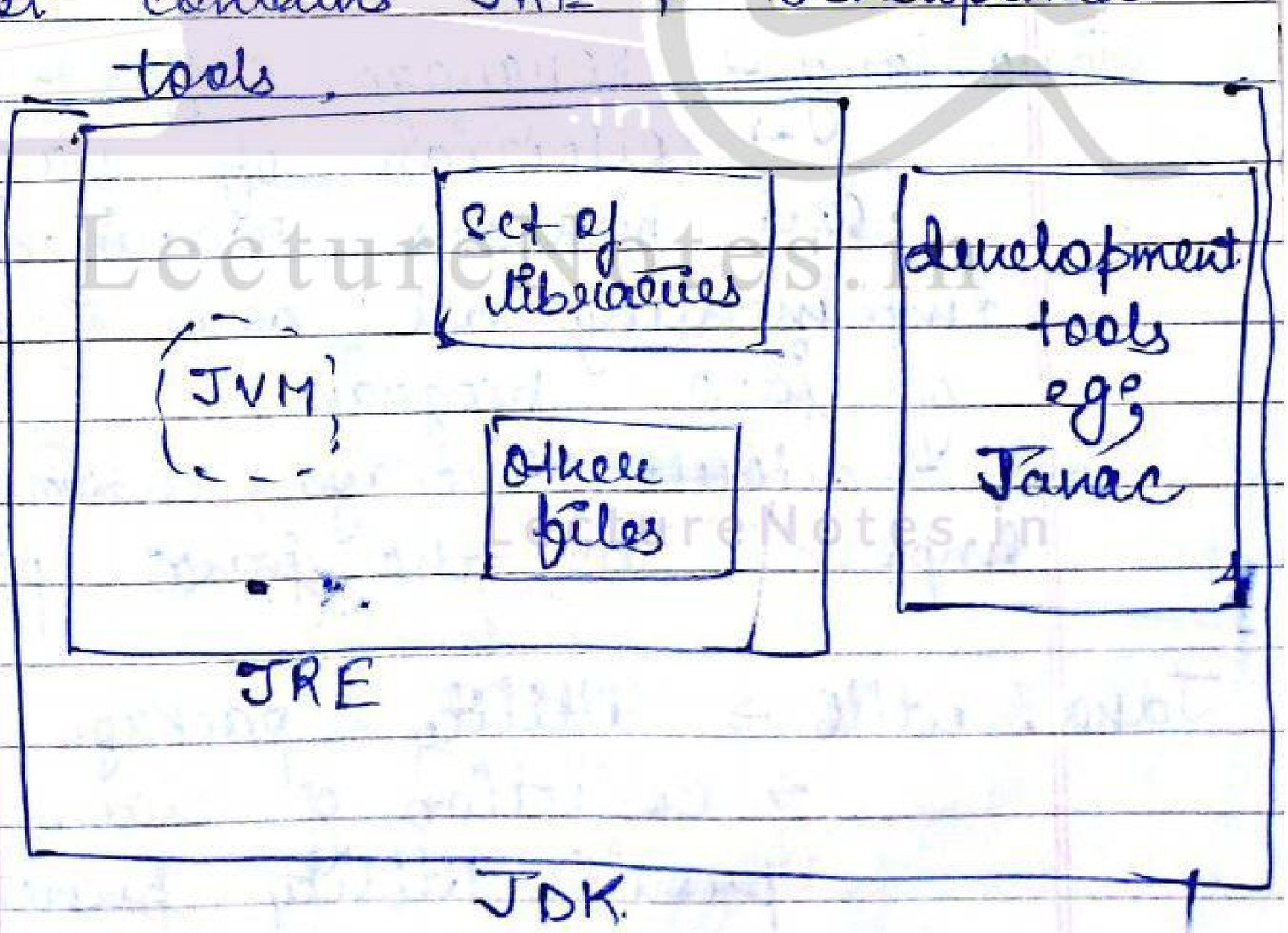
Date :
Page No.

JRE

Java Run-time Environment.
used to provide run-time environment.
→ It is an implementation of JVM.

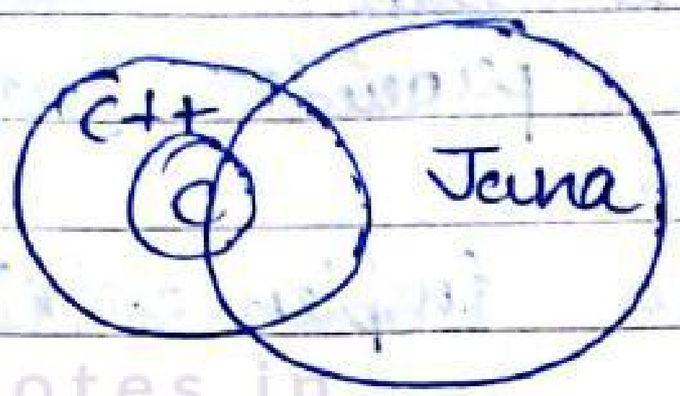


JDK → development tool Kit.
→ It contains JRE + Development tools.



Date :
Page No.

Overview of C, C++ & Java



→ Java seems like C & C++ as it uses C & C++ syntax
→ No pointers
→ No operator overloading
→ No multiple Inheritance
→ Java is slower than C++.

Packages in Java program

- Java.lang → Language support package
  → collection of classes and methods required for implementing all basic features of java program.
  → default package & automatically imported to the java program.
- Java.utils → Utility package.
  → collection of classes to provide utility functions such as date & time functions

Date :
Page No.

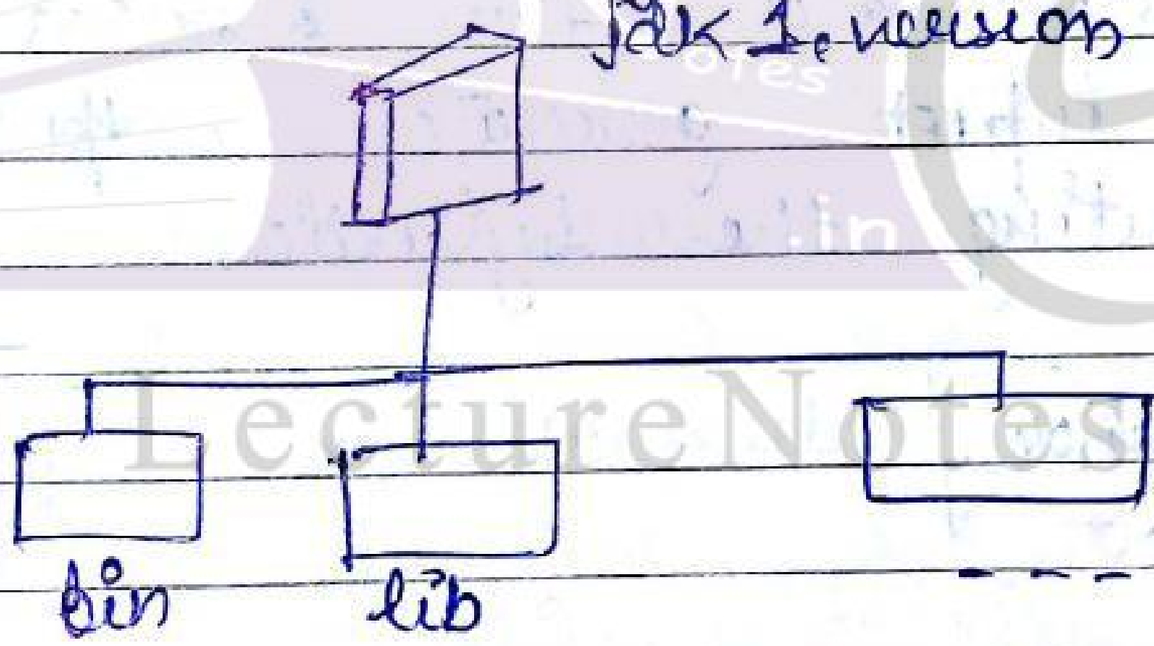
Java , Io 6

→ Input / Output Package →
→ Collection of classes required for I/O manipulation.

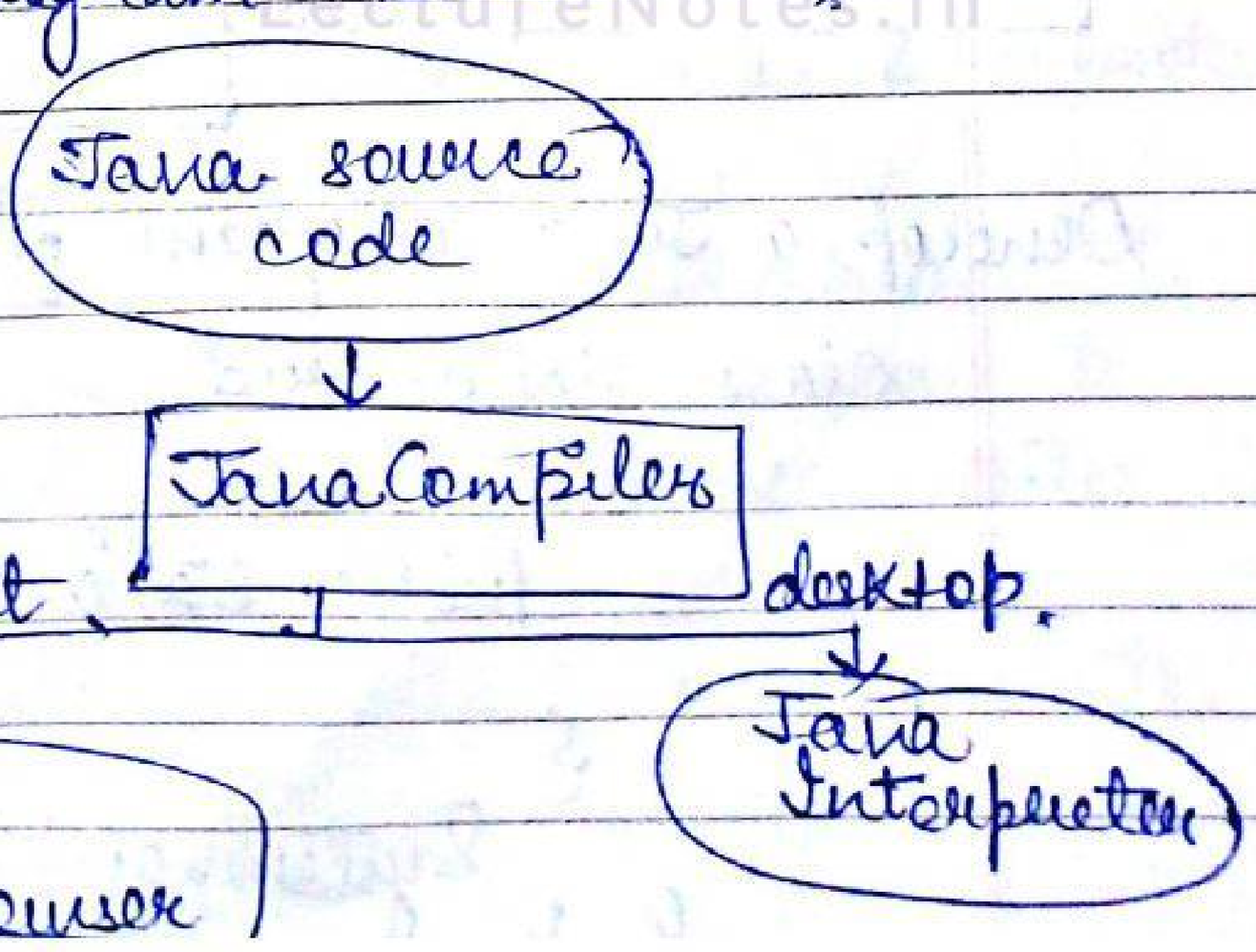
java.net → Networking package.
→ Collection of classes required for communicating with other computers via internet.

AWT package = Java.Awt

#. JDK directory structure &



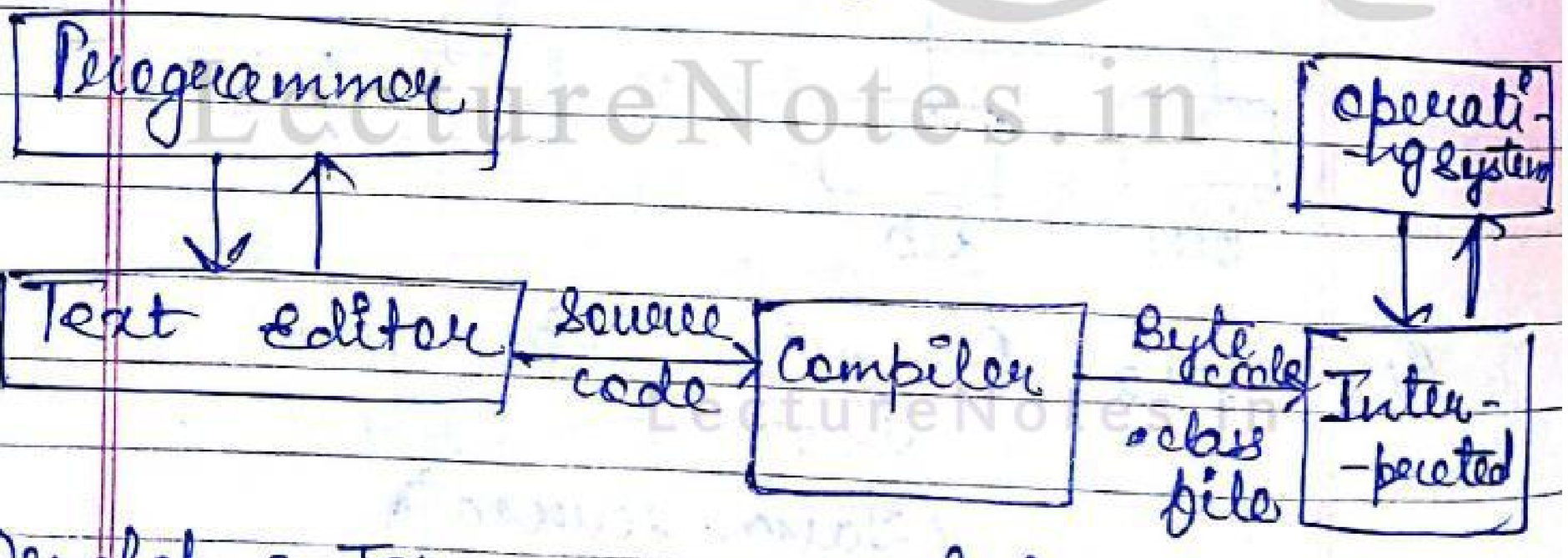
#. Java Program Structure.



Date :
Page No.

*. Java Program compilation and execution process:-

Compilation and execution of a java program is a two-step process. During compilation, Java Compiler compiles the source code and generates bytecode.
This intermediate Bytecode is saved in the form of .class file.
In second phase, the Java interpreter takes this .class file as input generates outputs by executing the bytecode.



Develop a Java program first:

```
Class HelloWorld
{
    Public static void main(String args[])
    {
        System.out.println("Hello");
    }
}
```

Date :
Page No.

Compile the Java program :

Once the Java program ~~comp~~ written and saved, it needs to be compiled. To compile a Java program, we need to invoke the java compiler by supplying javac command.

Java Compiler comes with JDK ie, java development kit. JDK is collections of tools needed for development & running of java programs. It includes JRE and set of API classes.

javac HelloWorld.java.

java compiler creates a .class file ie, bytecode which contains instructions that java interpreter will execute.

Main function of compiler is to convert source code to bytecode or source files to class file that the virtual machine can executes.

Run → Java virtual machine
↓
Interpreter

Date :
Page No.

Run Java Program.

After successful compilation of Java program from HelloWorld.java to HelloWorld.class to actually run the program we need the Java interpreter i.e. 'java'. To do so, we pass the command line argument as follows:

Java HelloWorld.

→ The message HelloWorld will be printed on the screen as a result of above command.

The JVM takes the following steps to run a program:

① → loading of classes and Interfaces
loading the bytecode.

② → Linking of classes and Interfaces
taking the bytecode and combining it to the run-time state of Java virtual machine, so that it can be executed.

Date :
Page No.

Ques Why main method is declared as public and static?

Ans main() method is declared as "public" to provide access to the JVM so that the JVM can call it without being member of that class.
main() method is declared as "static" to allow JVM to call main() method without ~~that~~ creating the object.
In Java, methods can be called only with the help of objects and main is the starting point of any java program And that instance, ~~there~~ no object can be created. Hence, main is declared as static as only static, ~~members~~ methods can be called without objects.

General structure

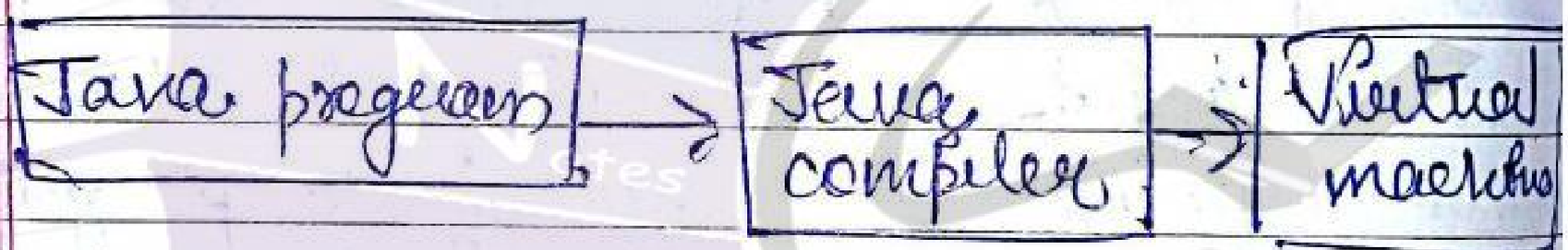
General str. of Java Program

Document Section → Suggested
Package statement → optional
Import statement → optional
Interface statement → Optional
class Definition → optional
main method class → Essential
{Main method definition}

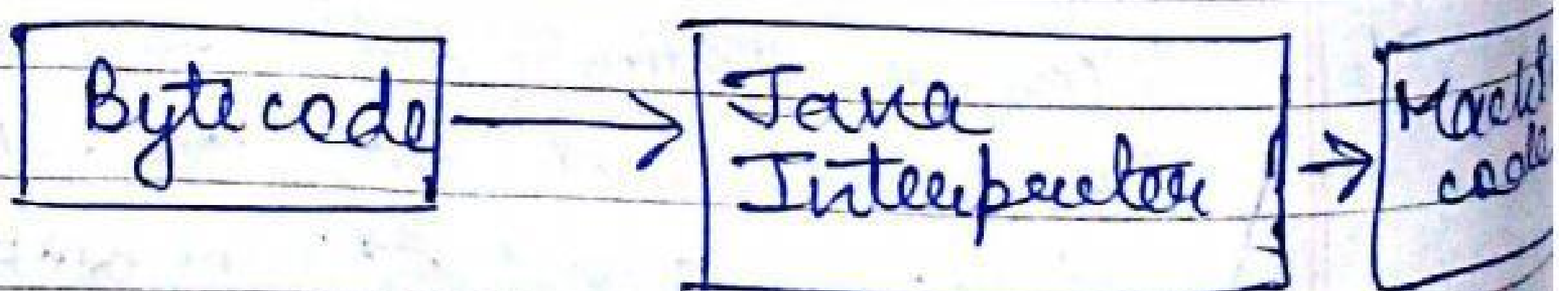
Date :
Page No.

Java Virtual machine.

Java compiler generates intermediate code called Bytecode for a machine that dosnt exist.
This machine is called virtual machine and it exists only inside the computer memory.
Byte code is also refferred to as virtual machine code.

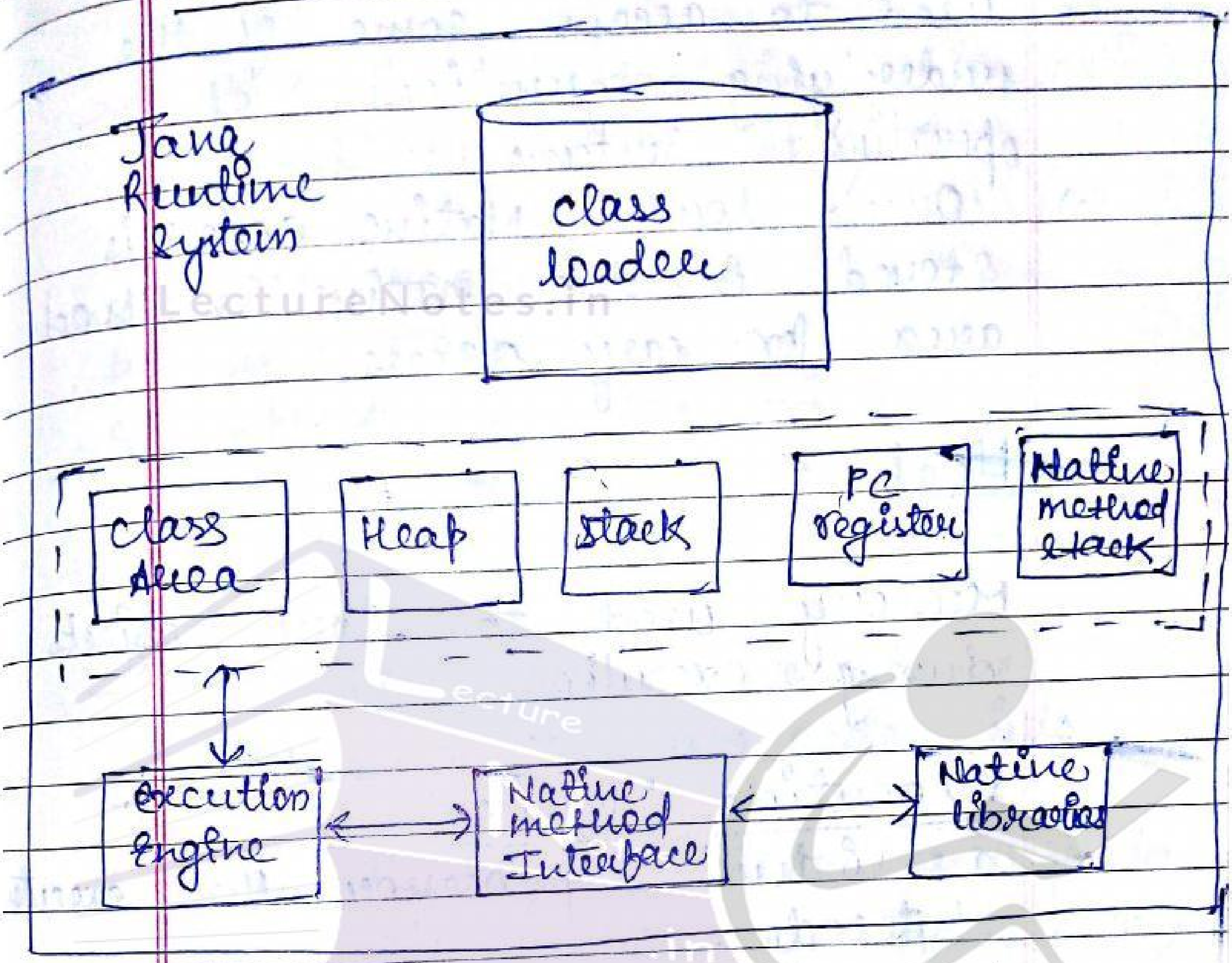


Virtual machine code is not machine specific code.
The machine specific code is generated by the java interpreter by acting as the intermediatory between the virtual machine and real machine.



Date :
Page No.

Internal Architecture of JVM



Class loader

→ locates and loads classes into the JVM.
→ loads java classes into the run-time environment.
→ loads trusted class file.
→ loads untrusted classes from the local file system and passes them to class file verifier.

Date :
Page No.

Native - Method loader

→ used to access some of the underlying functions of operating systems.

→ Once loaded, Native code is stored in the native method area for easy access.

Heap

Memory used to store objects during execution.

Execution Engine

- a virtual processor that executes bytecode.
- has virtual registers, stack etc.
- Contains JIT
- performs memory management etc.

JIT - Just In Time compiler

→ translates bytecode into machine code at run-time.

→ performance overhead due to interpreting bytecode.

→ works on a method-by-method basis.

Date :
Page No.

Organisation of JVM.

→ JVM is divided into three conceptual data spaces:

a. Class Area
b. Java stack
c. Heap

Class Area:

- The class Area stores the classes that are loaded into the system.

→ Method implementations are kept in the space called method area, and constants are kept in a space called constant pool.

→ Objects are stored in the heap.

~~A class~~

Date :
Page No.

Classes have general properties.
- Superclass
- List of interfaces
- List of fields
- List of methods
- List of constants

- All properties of classes are immutable. i.e, ~~they~~ there's no way to change any property of class once it has been brought to the system. This makes the machine more stable.

→ In Java, ~~multithreaded~~ we have multithreaded architecture as java supports multithreading so, a program counter 'PC' is created every time a new thread is created.

PC → Keeps track of the current instruction executing at any moment.

If the method is native, then the value of 'PC' counter is undefined

Date :
Page No.

Java stack :

→ JVM stacks are used to ~~create~~ store JVM frames. The JVM frame on the top of the stack is called the active stack frame.

→ JVM frames are created ; every time a method is invoked. Each stack / JVM frame has an operand stack, an array of local variables and a PC. PC → points to the current instruction being executing.

→ Only the operand stack & local variable array for the active stack frame is used.

~~The program counter~~. Each time, the method is invoked, a stack frame is created and becomes the top of java stack. The PC is saved as part of older java stack frame. The new frame has its own new PC.

Date :
Page No.

Heap

Stack-overflow error: This error occurs in fixed JVM stacks.

→ If the memory size is not sufficient, during the program execution.

Out of memory - Error: When dynamic sized stacks, when they try to expand for more memory need & there is no memory available to allocate and, we get out of memory error.

Heap:

- Objects are stored in the Heap.
- Each object is associated with a class in class Area.
- Heap areas are used to store objects of ~~data~~ array and classes.

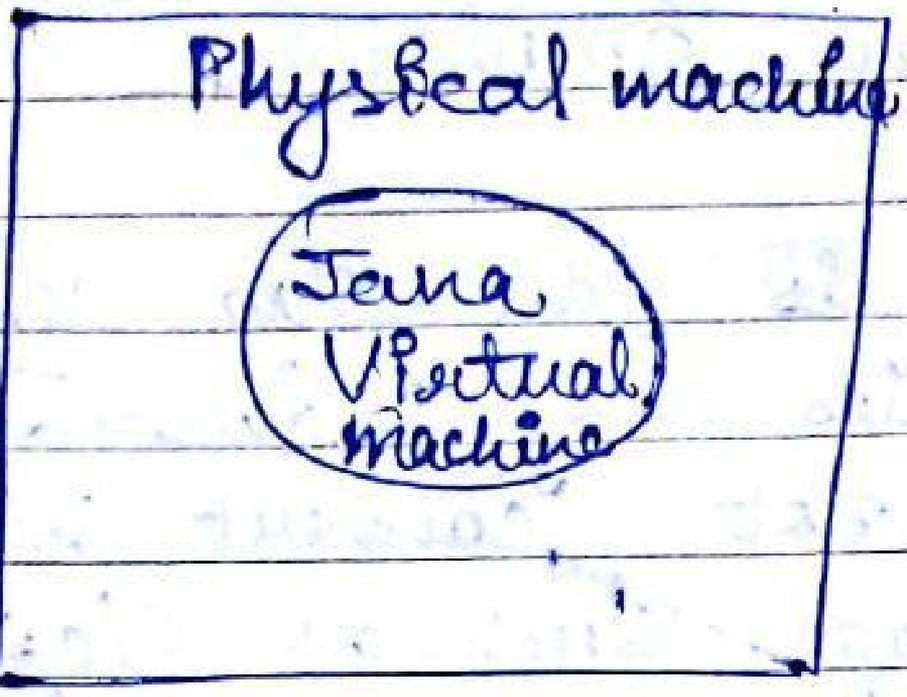
Date :
Page No.

JVM as an Emulator

JVM is an ~~also~~ emulator that emulates bytecode. Java Compiler does not convert high-level language such as C/C++ to the machine level language & it converts the java language to the java bytecode that the JVM understands.
Since, java bytecode has no platform-dependent code; it is executed on the hardware on which JVM is installed, even when the CPU or OS is different.

The difficult part of creating java bytecode is that the source code is compiled for the machine that doesn't exist. This machine is called java virtual machine, and it exists only in computer's memory.

Emulator: A hardware or software that enables one computer system (host) to behave like another computer system (Emulator). Typically



The Java virtual machine is responsible for interpreting byte code and translating this into actions or Operating System calls.

JVM as an Interpreter

An Interpreter is a program that acts much like a CPU, with a kind of fetch-and-execute cycle.

→ In order to execute a program, An Interpreter reads an instruction, decides what is necessary to carry out the instruction and performs the require command to do so.

→ One use of interpreter is to use high-level language.

Source code → Byte code → Interpreter → (Scene)

Date :
Page No.

The Java virtual machine Instructions set.

Java virtual machine Instruction consists of an opcode specifying the operations to be performed.

Types of Java Virtual machine Instructions set:

The JVM is stack-oriented interpreter that creates a local stack frame of fixed size each time a method is invoked. The size of the local stack is to be computed by the compiler. The values may also be stored intermediately in a frame area containing local variables. These variables are numbered from 0 to 65535 i.e, we have a maximum of 65536 of local variables per method.

- The Instruction set can be roughly grouped as follows:

1. **Stack Operations:**
   In stack, we have load & store instructions.

Date :
Page No.

load & takes something from the augruments and pushes it to the top of the stack area.

different kinds of "load" instructions.

iload → Integer load
fload → float load
l load → long load
d load → double load.

→ Each instruction has different "forms" for different kinds of operands.
eg; different forms of iload instruction.

Eg:

iload - 0	26	
iload - 1	27	
iload - 2	28	
iload - 3	29	
iload - n	21	n

Store & pop something from the top of operand stack and places it into args / local area.

Date :
Page No.

(2). Arithmetic operations :
The instruction set of JVM distinguishes its operand type using different instructions.

Add : iadd, ladd, fadd, dadd
Subtract : isub, lsub, fsub, dsub
Multiply : imul, fmul, lmul, dmul.

(3). Control flow : There are branch instructions like if_icmpeq which compares if two integers are equal.
another eg : jsr instruction (jump to sub-routine)

(4). Field access :
→ In Jvm, different kinds of memory can be accessed by different kind of instructions.

1. Accessing locals and arguments : load & store instruction.
2. Accessing fields in object : getfield, putfield.
3. Accessing static field : getstatic, putstatic.

5. Method invocation :

1. invokevirtual : usual instruction for calling a method on an object.

2. Invokeinterface : Same as invokevirtual but used when a called method is declared in an interface.

3. Invokespecial : also called invokenonvirtual
→ used for calling things such as "constructor".

4. Invokestatic : used for calling method that have the "static" modifier.

6. Conversion :

- i2l → Integer to long
- i2f → Integer to float
- i2d → integer to double
- l2f → long to float
- l2d → long to double
- f2d → float to double.

Date :
Page No.

Class file format

My Very Cute Animal Turns Savage In Full Moon Areas.

Magic no.	version
Constant pool	
Access flag	
This class	
Super class	
Interfaces	
Fields	
Methods	
Attributes	

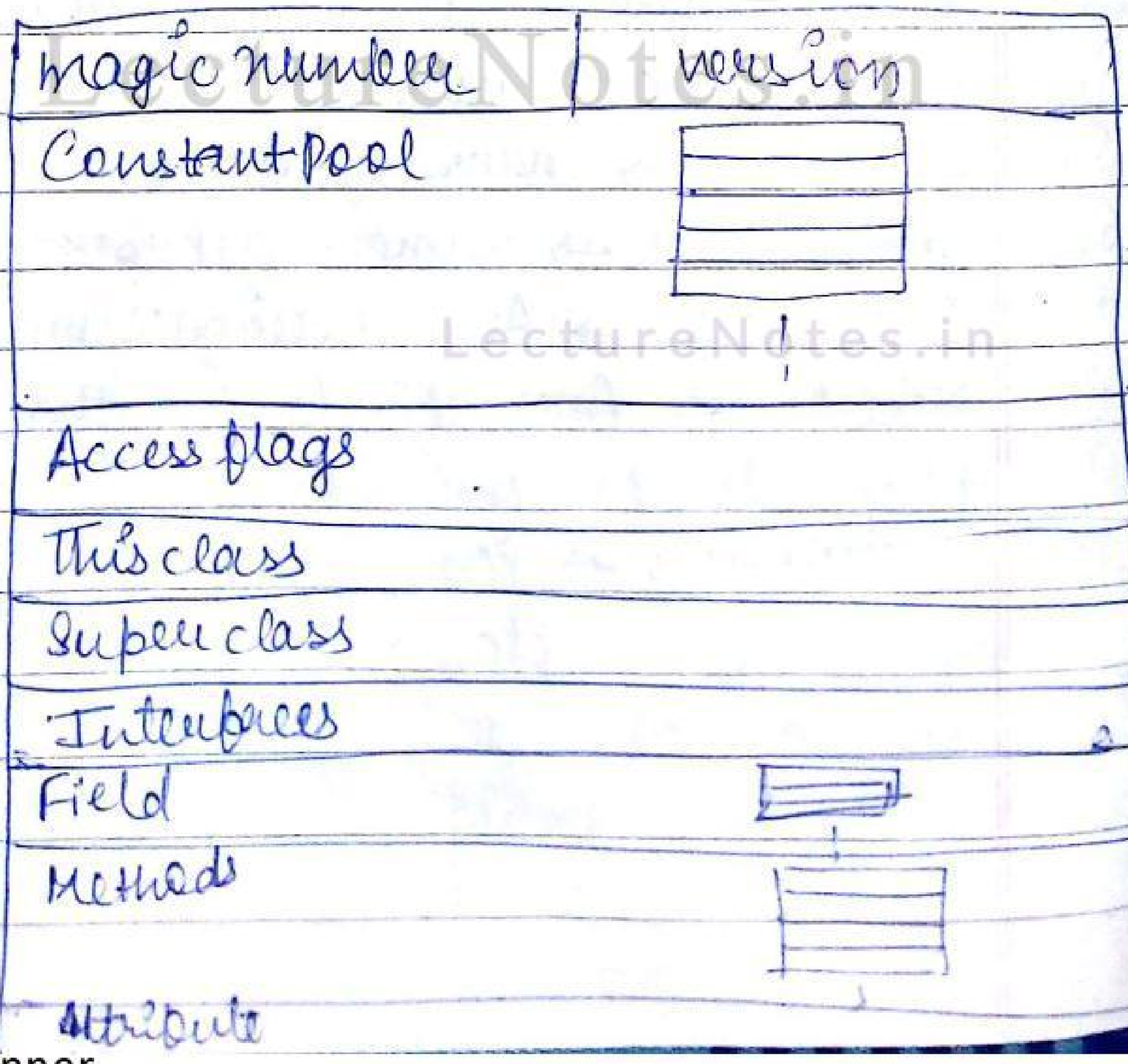
order is strictly defined.

1. magic number. → 0x CAFEBABE.
2. Version → minor and major versions of class file.
3. Constant pool → pools of constants for the class.
4. Access flag → wether the class is abstract or static
5. This class → name of current class.
6. Super class → name of super class
7. Interfaces → Any Interfaces in the class
8. Fields → Any fields in the class.
9. Methods → Any methods in the class
10. Attributes → Any attributes in the class!

Date :
Page No.

Eg:
```
ClassFile {
    u4 magic;
    u2 minor_version;
    u2 major_version;
    u2 constant_pool_count;
    u2 access_flags;
    u2 this_class;
    u2 super_class;
    u2 interfaces_count;
    u2 fields_count;
    u2 methods_count;
    u2 attributes_count;
}.
```

→ Length of class is not fixed.
→ Variable length sections are constant pools, methods etc.



Date :
Page No.

Verification :

→ The purpose of verification is to ensure that programs follows a set of rules that is designed for the security of the JVM.

→ Program's can try to subvert the security of JVM by stack overflowing, corrupting the memory they are not allowed to access.

→ The verification algorithm ensures that this does not happen by checking that each object in the class is used according to their proper type. It traces the code to do so.

→ The verification algorithm is applied to the classes as ~~before~~ it is loaded, before its instances are created. This ensures the JVM ~~to no~~ implementation to assume that classes have certain safety properties.

Date :
Page No.

→ The verification algorithm makes it possible to safely downdo-ad the javaapplets from the Internet.

⇒ How verification algorithm works?

The Java virtual machine specification contains a large set of rules that programs have to follow if they want to run in the machine. It is the job of the verification algorithm to prove that these rules are followed by each class.

The verification algorithm works by asking a set of questions about class files.

The questions can be answered by just looking at the bytecodes without executing the program.

→ The enables java virtual machine to stop badly behaving programs before they start.

→ It permits faster program execution.

Date :
Page No.

base example,

```
int i = 70;
float j = 111.1 f;
double k = i + j;
```

```
0  ldc 70 f;        // Initialize i
   fstore_1 ;
   ldc 111.1 ;      // Initialize j
   fstore_1 ;
   iload_1 ;        // loads i
   fload_2 ;        // loads j
   fadd ;
   dstore_3 ;
```

→ Give error because at the top of stack, there is float so fadd → will create error.

Java Garbage Collection

The allocation and deallocation of memory for objects is done by garbage collector in an automated way by Java.

The developer not need to write code for garbage collection process as in 'C'.

It is an automatic process to manage memory at run time used by programs.

Garbage collector takes into consideration the memory space no longer avaibles for use & empties it for future references.

Java Garbage Collection - gc Initiation.

Developers need not to initialize garbage collection explicitly in code.

System.gc() & request.gc() are the

Date :
Page No.

methods that are used to call garbage collection process.

JVM can also choose to reject these ~~processes~~ requests so it is not guaranteed that ~~called~~ those requesed can call the garbage collection process.

~~He~~ Java ~~decide~~
Decision is taken by the JVM based on the "eden" space availability in the Heap memory.

HOTspot Virtual machines garbage collector uses generational garbage collection process.

It separates JVM memory into ~~three~~ two parts &

- Young generation
  - Eden space
  - Survivor space
- Old generation
  - tenured gens
  - permanent gen.

Date :
Page No.

Eden Space :- When an instance is created it is firstly stored in eden-space, or in young generation of heap area.

Survivor Space (S0 & S1) :- Instances stored in Eden spaces are scanned; The instances that are live (still referenced) are move to S0 from Eden Space. And instances which are not live (not referenced) are marked for garbage collection (eviction process).

Garbage collector

Similarly, ~~S0 to~~ scans the move all the live instances to S1 from S0.

The dereferenced instances (not live) marked for eviction process.
The JVM decides either ~~to check~~ the dereferenced instances to be removed from the memory or go on the eviction process will be done in the separate process.

Date :
Page No.

Old generation : The live instances in the S1 are further promoted to old generation.

→ Tenured generation is the logical part of old generation in Heap memory area.

→ This process is called minor GC process.

Major GC : Old generation is the last phase of the instance cycle.

Major GC is the process that scans the old generation part of heap memory.

Memory fragmentation : Once the instances are deleted from the heap memory it empties the memory space to be used for in future for live instances.

This memory spaces are as a fragmented across the memory areas.

Finalization of Instances is done by calling the final() method just before the eviction process.

Java Security :

The need of Java security :

→ Downloaded executable content can be dangerous.

⇒ The term security might lead us to expect that Java programs would be :

- programs should not be allowed to harm the users computing environment (Trojan horses, virus)
- programs should be prevented from disconnecting private information on the host computer.
- Data sent and received by the program should be encrypted.
- The identity of parties involved in the program must be verified
- Rules of operations should be set & verified.

Date :
Page No.

Security promises of the JVM :

There are some security promises that JVM made about programs that passed verification.

1. Every object is created exactly once, before it is used.
2. Every object is an instance of exactly one class, which can't be change through the life of object.
3. If field or method, is marked as private, then it can only be accessed within its class itself.
4. If field or method is marked protected, then it can only be accessed through the code which is an implementation of that class.
5. Every field is initialized before it used
6. Every field & local variable is initialized, before it is used.
7. It is impossible to underflow or overflow a stack.
8. An attempt to use a null reference as a receiver of method invocation always throw a null-pointer Exception.

Date :
Page No.

9. Final methods cannot be overridden and final class cannot be made subclass.

10. It is impossible to change the length of the array, ~~before~~ once it is created.

11. It is impossible to write past the end of the array or at the beginning of the array.

Security architecture and security policy

The Java platform builds a security architecture on the top of precautions premised by JVM.

The A security architecture is the way of organising a platform that makes up java platform so that potentially harmful operations are isolated from unprevileledge code and only available to the previleged code.

Most Code are unprevileledged; Security architecture is responsible for ensuring that unprevileledge

Date :
Page No.

code doesnot masquerade the previledge code.

~~The core of Java~~
There are three pillars of Java security :

1. Security Manager.
2. Class loader.
3. Bytecode Verifier.

The core of Java platform security architecture is security manager. This class decides which piece of code can perform certain operations and which cannot.

These decisions are called security policies.

Security manager ensures that the permission specified in policy file cannot be overridden.

It uses a checkpermission() method.

- This method uses permission object as an input and outputs a 'yes' or 'NO' (based on the source code & permissions)

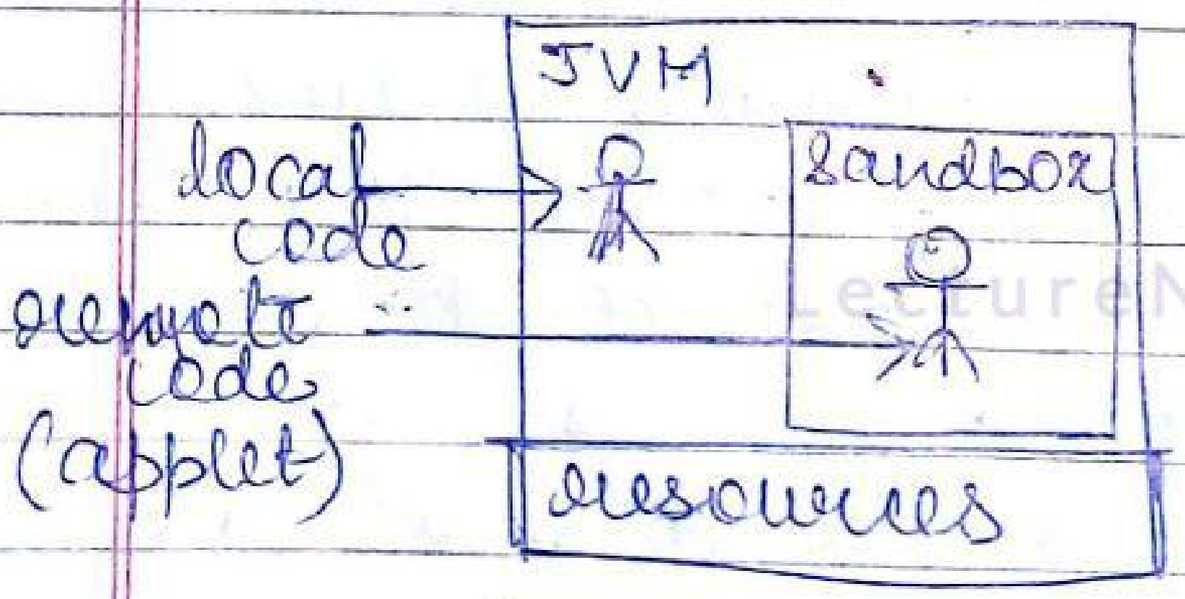
Date :
Page No.

checkpermission() is called from trusted classes.

There can only be one security manager active at a time. JDK has its own security manager. (default SM).
Java programs can have their own security manager if they have permission to do so.

SANDBOX security model

Idea: limit the access to system resources by applet (remote code).



→ Introduced in Java 1.0
→ local code has unrestricted access to system resources.
→ Remote code (applet) has

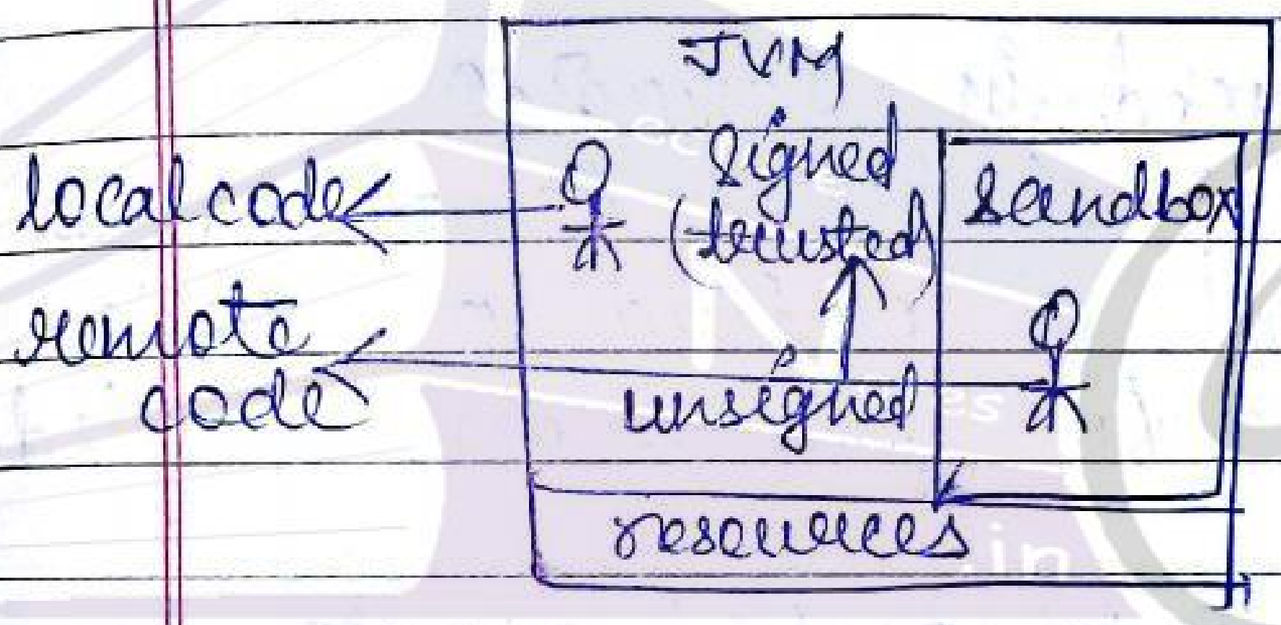
Date :
Page No.

restricted access to system resources.

→ separate code (applet) is limited to sandbox.

The Concept of trusted code.

→ introduced in java 1.1



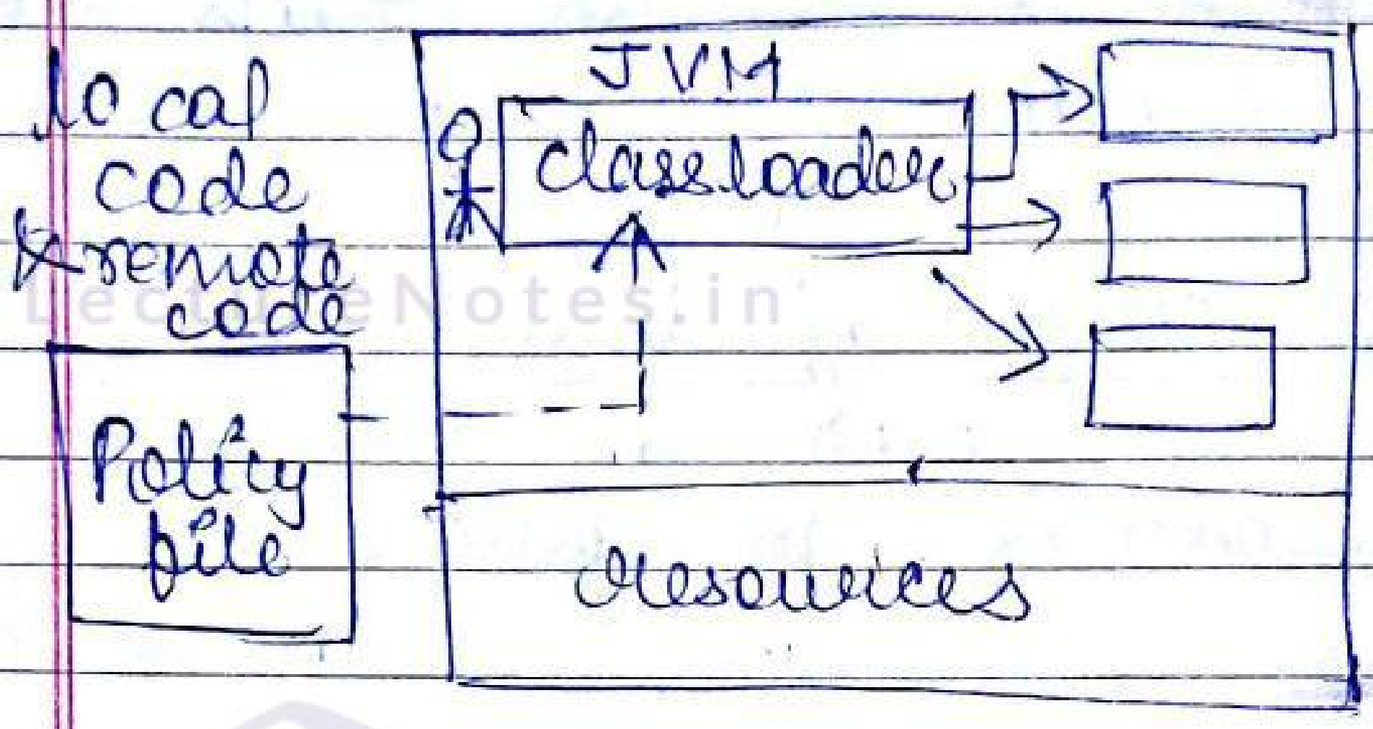
→ Based on the idea, that local code has unrestricted access to the systems resources.

→ Remote code (applet) signed by unauthorized principals unsigned applet has restricted access to system-resources

→ United to sandbox.

Date :
Page No.

Fine-grain access control



→ Introduced in Java 2.0

→ The protection domain was created according to code source & permission granted.

→ Every code (local & remote) has unrestricted access to the system resources based on the permissions written in policy file.

→ Code source is consists of URL.

→ permissions granted to the code source are specified in the policy file.

UNIT-2

#. Why 'main' is made static

→ Because to call main function we need to create object which will consume memory for the class, so, In order to save memory, we avoid to create object we make main as static

new Keyword is not used with primitive datatype because 'new' creates references wether primitive datatype stores / assigns value dynamically

#. General structure of a Java program

Document Section	→ suggested
package statement	→ optional
Import "	→ "
Interface "	→ "
class Definition	→ "
Main method class	→ Essential
{	
main method Defind	
}	

If we access a variable with class name → it is static.

String concatenation → + left to right.

Page :
Date: / /

Scope of variable → accessibility or life-time of a variable.

* String → Sequence of characters. Anything written in double quotes " " is String is inbuilt class in Java. Hence it is not object but it is a class in java.

To access first character -
Selection.charAt(first);

* Wrapper classes :-

→ For connection.
→ For converting string input to objects & vice-versa for accessing them.

In Java we have primitive data types.
↳ (are not classes)

Syntax :-

Wrapper.valueof( ).
↓
wrapper class name.

Abstract" → Data Hiding.

Page :
Date: / /

Collections → getting all items

Simple Example.

```
List myList = new ArrayList();
// we add items.
Iterator iterator = myList.iterator();
while (iterator.hasNext())
{
String
```

Syntax for class

```
class classname { extends Sub-class
```

}

Page :
Date: / /

Ex: Java class ✓

class Box
{
double w, l, h;
double area()
{
return (w*l*h);
}
void setvalue(double w, double l, double h)
{ this.width = width
this.length = length
this.height = height }
};

class BoxDemo
{
public static void main(String[] args)
{
Box b = new Box();
Box c;
b.setvalue(10.2, 11.4, 12.1);
c = b;
d = b.addvalue(c);

Invoking object.

Page :
Date: / /

Primitive type → call by value
, objects → " " reference

```
Box add volume (Box b)
{
    width = width + b.width
    height = length + b.length.
        return this;
}
```

Sun looks up for
(1). name of the method
(2). no. of arguments.
(3). Type of arguments.

private → accessible to each & every sub-class

All Instance of the class share the same static variable.

Reverse no. = Reverse no * 10 + temp;

no. = no/10.

Page:

Date: / /

Restrictions on static blocks

① static methods can only call other static methods.

② They can only access static data.

③

Final

Syntax :-

final type name.

→ alternate for function overloading

Varargs → concept in java through which we can pass multiple ~~different~~ arguments

→ Varargs is done with the ... operator

Date :
Page No.

Example 6

lang. Package

↓

Math {→ class}

↓

sqrt, Pow, sin etc {→ functions or methods}

Eg- Class Power
{
public static void main (String arg []).

{

double a = 5;
double b = 2;
double c;
c = math. pow (5, 2);
System. out. print ("power = " + c + "ans");

}

}

output :→ power = 25.0 Ans.

String

Date :
Page No.

Round

$\frac{4}{3} = 1.33 \rightarrow$ Round (4.0)

(#) - class Abc
{
public static void main (String arg[])
{

double a = 3.533;
double b;
b = Math.Round (3.533);
System.out.print ("rounded" + b + "ans");
}
}

O/P :- rounded = 4.0 ans
=.

(#). class Xyz
{
public static void main (String arg[])
{
double a = 2.6;
double b;
b = Math.floor (2.6);
System.out.print ("req." + b + "ans");

Date :
Page No.

Conditional Statements :-

If, else, nested if, ladder, switch.

(1).

```
import java.util.*;
class compare
{
public static void main (String arg[]):
{
Scanner sc = new Scanner (System.in);
int a, b;
System.out.print ("enter a =");
a = sc.nextInt();
System.out.print ("Enter b =");
b = sc.nextInt();
{
if (a>b)
{
System.out.print ("a" + "is" + "greator");
}
if (b>a)
{
System.out.print ("b" + "is" + "greator");
}
}
}
```

Date :
Page No.

Even/odd.

```
import java.util.*;
class Compare
{
public static void main(String args[])
{
Scanner sc=new Scanner(System.in);
int n;
System.out.print("Enter n=");
n=sc.nextInt();
if(n%2==0)
{
System.out.print("n"+"is"+"even");
}
if(n%2!=0)
{
System.out.print("n"+"is"+"odd");
}
}
}
```

Date :
Page No.

#.

```
import java.util.*;
class grade
{
public static void main (String arg[])
{
Scanner sc = new Scanner (System.in);
int m1, m2, m3, M4, m;
System.out.print("Enter m1 =");
m1 = sc.nextInt();
System.out.print("Enter m2 =");
m2 = sc.nextInt();
System.out.print("Enter m3 =");
m3 = sc.nextInt();
M4 = m1 + m2 + m3;
System.out.print("Total =" + M4 + "\n");
m = M4/3;
System.out.print("Average =" + m + "\n");
}
if (m <= 100 && m >= 70)
{
System.out.print("A");
}
else if (m < 70 && m <= 60)
{
System.out.print("B");
}
```

Date :
Page No.

```
}
else if (m<60 && m<=50)
{
System.out.print("C");
}
else
{
System.out.print("Fail");
}
}
```

vowels / consonent

or

→ This program was to find the total marks of a student and to determine the result i.e, fail or pass.

Q) Perform vowel / consonent.

Date :
Page No.

```
import java.util.*;
class Letters
{
  public static void main(String args[])
  {
  Scanner sc = new Scanner(System.in);
  String N;
  System.out.print("Enter any character");
  N = sc.next();
  {
  if (N=a||e||i||o||u)
  {
  System.out.print("Vowel");
}
else
{
System.out.print("Consonant");
}
}
}
```

Date :
Page No.

```
import java.util.*;

FIR by Scanner class.

m, n

obj = Input();
obj.Output();
```

Date / /
Page No.

Adapter class

→ used to simplify process of Event Handling in Java.

→ As we know, when we implement any interface, we need all the methods defined in that Interface needs to be overrided In the class, which is not desirable In the case of Event Handling.

→ Adapter classes are useful as they provide an Empty Implementation of all the methodes defined in an event listener Interface.

Date
Page No.

Anonymous Inner class

class → class id, not assigned any name

Example:

```
class Anony_test extends Frame
{
        Frame f;
        Button b;
        TextField tf;
        Anony_test()
        {
```

Frame

```
f = new Frame();
f.setVisible(true);
f.setLayout(new FlowLayout());
f.setSize(300,300);
tf = new TextField("");
b = new Button("click me");
f.add(b);
f.add(tf);
```

Date / /
Page No.

```
b. AddActionListener (new ActionListener()
{
public void Actionperformed (ActionEvent e)
{
String tf.setText( "welcome");
}
}
}};
}.
```

Date / /
Page No.

Multiheritance Inheritance in Interface

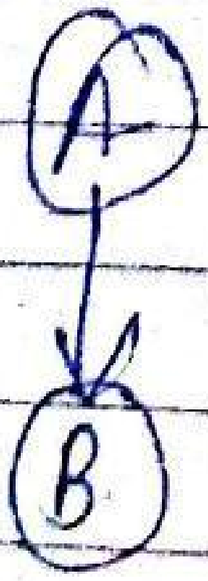
Subclass are detailed version of super class.

Benefits of Java Inheritance

1. Reusability of code
2. code sharing
3. consistency in using an interface.

Types

1. Single Inheritance



```
class employee
{
    private int eno;
    private string ename;

public void set emp(int no, string ename)
  {
     eno = no;
     ename = name;
  }

public void put emp()
  {
     System.out.println("Empno :" + eno);
     System.out.println("Ename :" + ename);
  }
}

class department extends employee
{
    private int dno;
    private string dname;
```

```
public void set dept(int no, String name)
{
    dno = no;
    dname = name;
}
public void put dept()
{
    System.out.println("deptno = " + dno);
    System.out.println("deptname = " + dname);
}
}
public static void main(String args[])
{
    department d = new department();
    d.setemp(100, "aaa");
    d.setdept(20, "sales");
    d.putemp();
    d.putdept();
}
}
```

3. multilevel

A

↓

B

↓

C

Ex:

```
class student
{
private int stno;
private String stname;
public void set stid(int no, int
                     name)
{
stno = no;
stname = name;
}
public void put stud()
{
Sy.O.P("Studno :" +(stno)
C.OP ("Studname :" + stname);
```

Date / /
Page No.

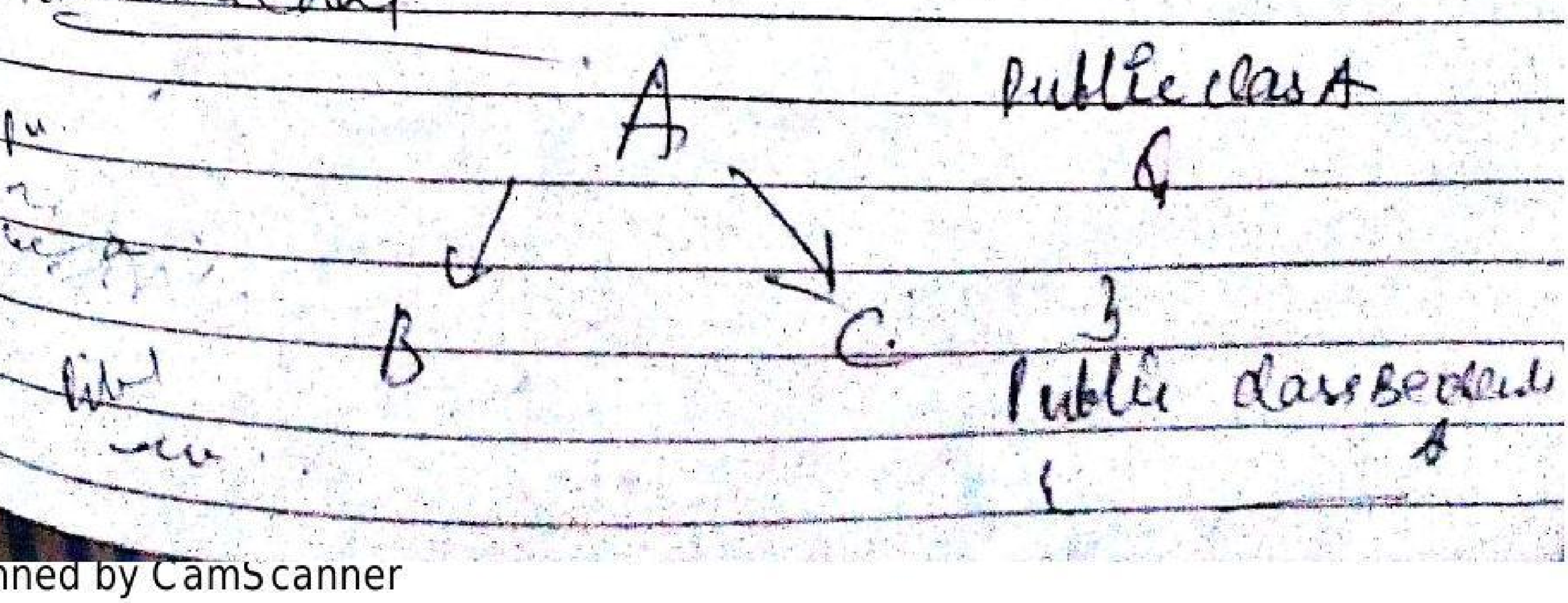
```
}

class marks extends Student
{
    protected int marks1, marks2;
public void set marks(int m1, int m2)
{
    marks1 = m1;
    marks2 = m2;
}
Public void put marks()
{
S.O.P("marks1 = " + marks1);
S.O.P("marks2 = " + marks2);
}
}
class Result extends marks
{
    private int total;
public void calc()
{
    total = marks1 + marks2;
```

Date / /
Page No.

```
}
public void putTotal()
{
    S.O.P("Total" + total);
}
P.S.V.M.(String args[]);
{
    Final fat f = new Finalfat();
    f.setStud(70, "Sunny");
    f.setmarks(78,89);
    f.calc();
    f.putstud();
    f.putmarks();
    f.putTotal();
}
}
```

Himanshu



Date / /
Page No.

Construction In Inheritance

```
class Parent {

Parent()
{
S.O.P ("S1");
}

class child extends Parent {

child() {
SOP ("S2");
}

Public class Test5 {
P.S.V.M (
Child c1 = new child();
}
```

(44). Using "super" keyword :

⇒ super is a reference variable that is used to refer immediate parent class object.

Uses of super keywords are as follows :

(1). super() is used to invoke immediate parent class constructor

(2). super is used to invoke immediate parent class method

(3). super is used to refer immediate parent class variable.

Example :

```
class base
{
    int a = 100;
}
class sup2 extends base
{
    int a = 200;
    void show()
```

Date / /
Page No.

System.out.println (super.a);
S.i.l. ... (a);

}

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

{

Sup2 obj = new Sup2();

obj.show();

}

}

O/P => 100
200

Method overridding:

In a class hierarchy, when a method in a subclass has same name and type argument as a method in superclass, then the method in the subclass is said to override the method in the superclass.

Method defined by the super class will be hidden

Date / /
Page No.

Example 8

```
class A
{
    int q, j;
    A(int a, int b)
    {
        i=a;
        j=b;
    }
    void show()
    {
        S.o.P.("i and j = "+i+" "+j);
    }
}
class B extends A {
    int k;
    B(int a, int b, int c)
    {
        super(a,b);
        k=c;
```

Date / /
Page No.

```
void show(){
    S.O.P.("K");
}
public class Override{
    P.S.V.M.(---){
    B b1 = new B(1,2,3);
    b1.show();
    }
```

O/P ⇒ K

Overloading V/s Overriding

Overloading	Overriding
method overloading is compile time polymorphism.	run time polymorphism.
→ way to provide more than one method in one class with same name but diff arg. to distinguish between them. (same name diff arg.)	→ same name same arg same return type

Date / /
Page No.

3. Method overriding forms the basis of dynamic method dispatch →

Dynamic method dispatch is an mechanism by which a call to an overridden method is resolved at run time, rather than the compile time.
Dynamic method dispatch is imp because it is how Java implements run-time polymorphism.

A superclass reference variable can refer to a sub-class object.
→ Java uses this fact to resolve calls to overridden methods at run time.

→ class Object

P int i = 10;

void

Date / /
Page No.

-Abstract class

An abstract class is a class that is declared abstract

- It may or may not include abstract methods
- → cannot be instantiated.
- → An abstract method has no body.
- → cannot create object.
- → An abstraction method is a method that is declared without an implementation.

Syntax:

```
abstract void moveTo (double deltaX, double deltaY);

Public abstract class GraphicObject {
    abstract void draw();
}
```

Date / /
Page No.

In order to use abstract-classes, they have to be sub-classed.

Example:

```
Abstract class parent-class
{
    abstract void meth1();
    void meth2()
    {
    S.O.P("method of abstract class");
    }
}
class child extends parent
{
    void meth1()
    {
    S.O.P("method of child class");
    }
}
class abstract
{
    P.S.V.M. ___
    {
    child c1 = new child();
    c1.meth1()
```

e2.meth2();

}

}

6. Interface

An Interface is a collection of abstract methods.

A class implements an interface thereby implementing the abstract methods of the interface.

An interface is not a class.

A class describes the attributes and behaviours of an object.

An interface contains behaviours that a class implements.

Date / /
Page No.

→ Interface is not extended by a class it is implemented by a class.
→ An Interface can extend multiple Interfaces.

Declaring Interface

```
Exp Interface Emp
{
    public void salary();
    int Increment = 10;
}
```

Implementing Interface

```
class Interfacetest Implaments Emp
{
    public void salary()
    {
        S.O.P("salary of Emp is" + (Increment * 5000))
    }
    public static void main(String args[])
    {
        Interfacetest I1 = new Emp();
        I1.salary();
    }
}
```

Date / /
Page No.

How Interface provides multiple Inheritance in java?-

```
Interface emp
{
  public void salary();
    int increment = 10;
}

Interface director
{
  public void salary();
    int increment = 100000;
}

class Interfacetest implements emp, director
{
  System.out.println("Salary of emp:"
      + (emp.increment+500));
  System.out.println("Salary of director");
      + (director.increment + 5000
  );
}

public static void main (String args[])
  Interfacetest I1 = new Interface.test
```

Date / /
Page No.

I1.salary(); }

}

}

Package

Package achin-java;

public class add

{

int a;

public int b

public add-num()

{

S.O.P("sum is "+(a+b));

}

}

Date / /
Page No.

Use of "final"

① Final variable → once a variable is declared as final, its value cannot be changed during the scope of the program.
final int i=10;

② Final method → cannot be final void methods overridden

③ Final class → cannot be introduced.
final class finalmethod

Date / /
Page No.

Exception Handling

Exception → problem arises during execution

Reasons → Invalid I/P
→ file to be open is not found
→ lost of network connection

&

E. Handling ⇒ try to catch the exception object thrown by the the error condition.

Types
Checked Exception
↳ IO exception, SQL Exception

Unchecked Exception ⇒ Runtime Exception
↳ Null pointer exception
↳ Arithmetic exception

Error → not exceptions at all

Date / /
Page No.

Hierarchy of Exceptions

Object

Throwable

Exception

IO Exception

SQL Exception

Runtime Exception

Arithmetic Exception

Null pointer Exception

Number format Exception

Error

Virtual machine error

Assertion error

Date / /
Page No.

Following five keywords are used to handle exception in Java

1. try
2. catch
3. ~~throw~~ finally
4. throw
5. throws.

try-catch block

→ placed around the code that might cause exception.

→ code within this block is referred to as protected code.

Syntaxe

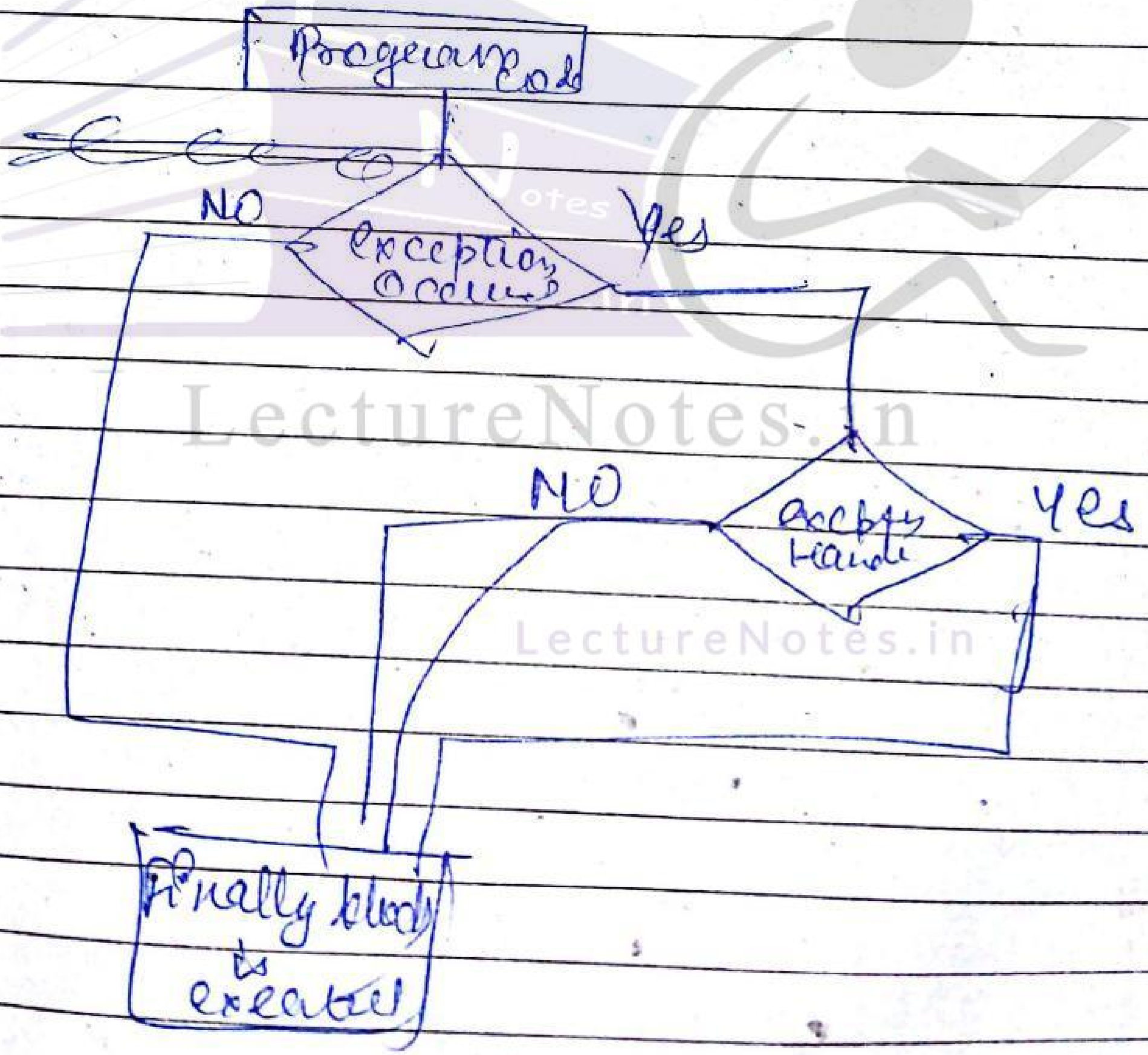
```
try
{
// protected code
}catch (exception e)
{
// catch block.
}
```

Date / /
Page No.

Catch Statement ⇒ declaring the type of exception we are trying to catch.

Finally

→ always executed whether or not an exception has occured.



Date / /
Page No.

try
{

}
catch
{

}
finally
{

}

sequence

Date / /
Page No.

Applets

Applet is a special type of program that is embeded in the webpage to generate the dynamic content.

→ It runs inside the browser.

→ Applets are used to provide interactive features to webapplication that cannot be provided by HTML alone.

→ A java applet extends the java.applet.Applet

→ The class must override the methods of the applet to set up a user interface.

→ An applet is automatically loaded and executed when you open a web-page that contains it.

→ Environment of the applet is known as the context of the applet.

Date 1. 1
Page No.

→ Use javac to compile and appletviewer to execute them.

→ All applets are subclasses of Applet.
→ Applets are not standalone programs. They run within either a web-browser or an applet viewer.
Execution of applet does not begin at main().
→ Use AWT methods.

→ To use an applet, it is specified in an HTML file.

#. Java Applet Advantages :

1. It is simple to make it work on any platform.
2. It is fast - Can have even have similar performance as native performed software.
3. It works at client side, so less response time.
4. Secured.

Date / /
Page No.

Java Applet disadvantages :

→ requires java plug-in.
→ Java applets with specified requirements are impossible to run unless Java is manually updated.

Difference between Applet & Application

Application	Applet
We cannot make web-pages by using application	We can make web-pages using applets.
cannot We use HTML tags in application.	We can use HTML tags in Applet.
For execution of applet there is no need for web-browser.	For executing applet web-browser is required.
To execute application java interpreter is needed	To execute applet java-web enabled browsers is needed.

Date / / .
Page No.

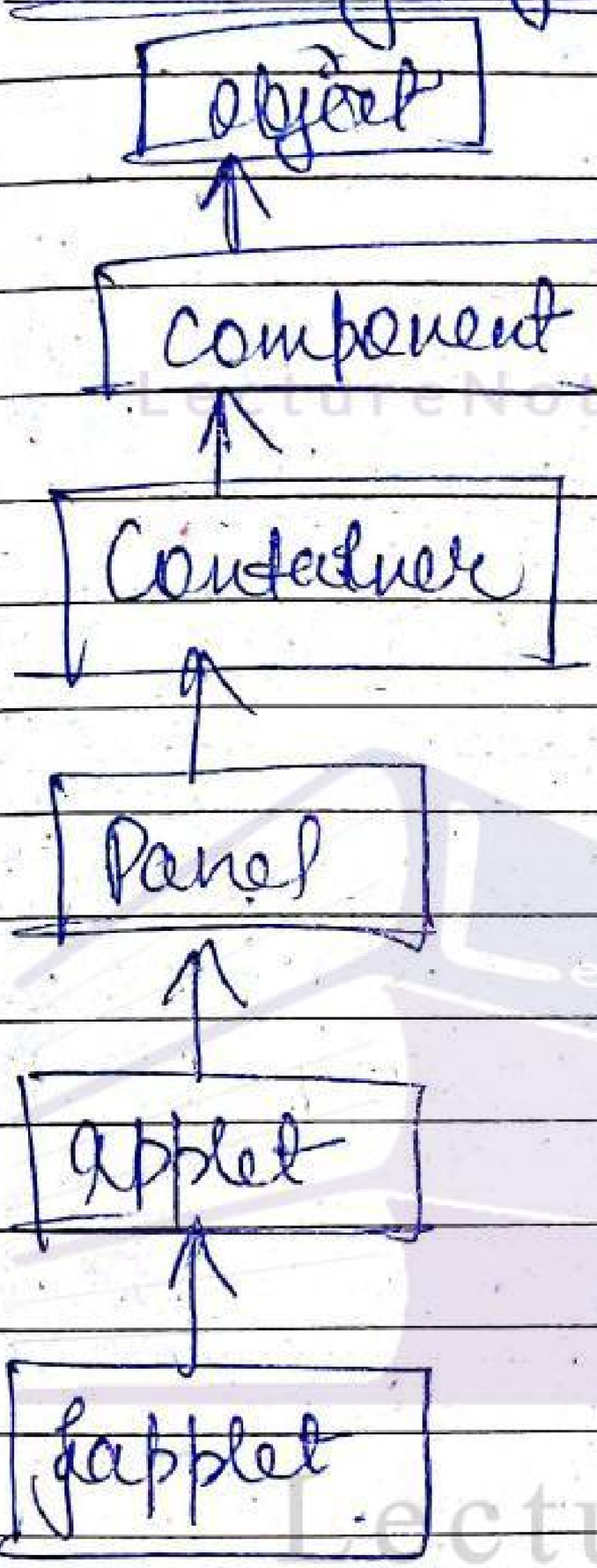
In application program execution begins with main() method.	In applet, execution doesnot begins with main() method.
No necessarily to use inheritance in a data program	applet is inherited from Applet.

Ex of Applet:

```
import java.awt.*;
import java.applet.Applet;
public class First extends Applet{
<applet code =
public void paint(Graphics g){
g.drawString("welcome", 150, 150);
}
}
```

Date / /
Page No.

Hierarchy of Applet



→ Applet class extends Panel
→ Panel class extends container
→ Container is a sub-class of component.

→ Applet interact with the user through the AWT not console-based I/O classes.

Date / /
Page No.

→ Applet derived from Awt.

* steps to incorporate and run an applet

1. have Myapplet.java
2. javac Myapplet.java
3. have Myapplet.class
4. Create Myapplet.HTML.

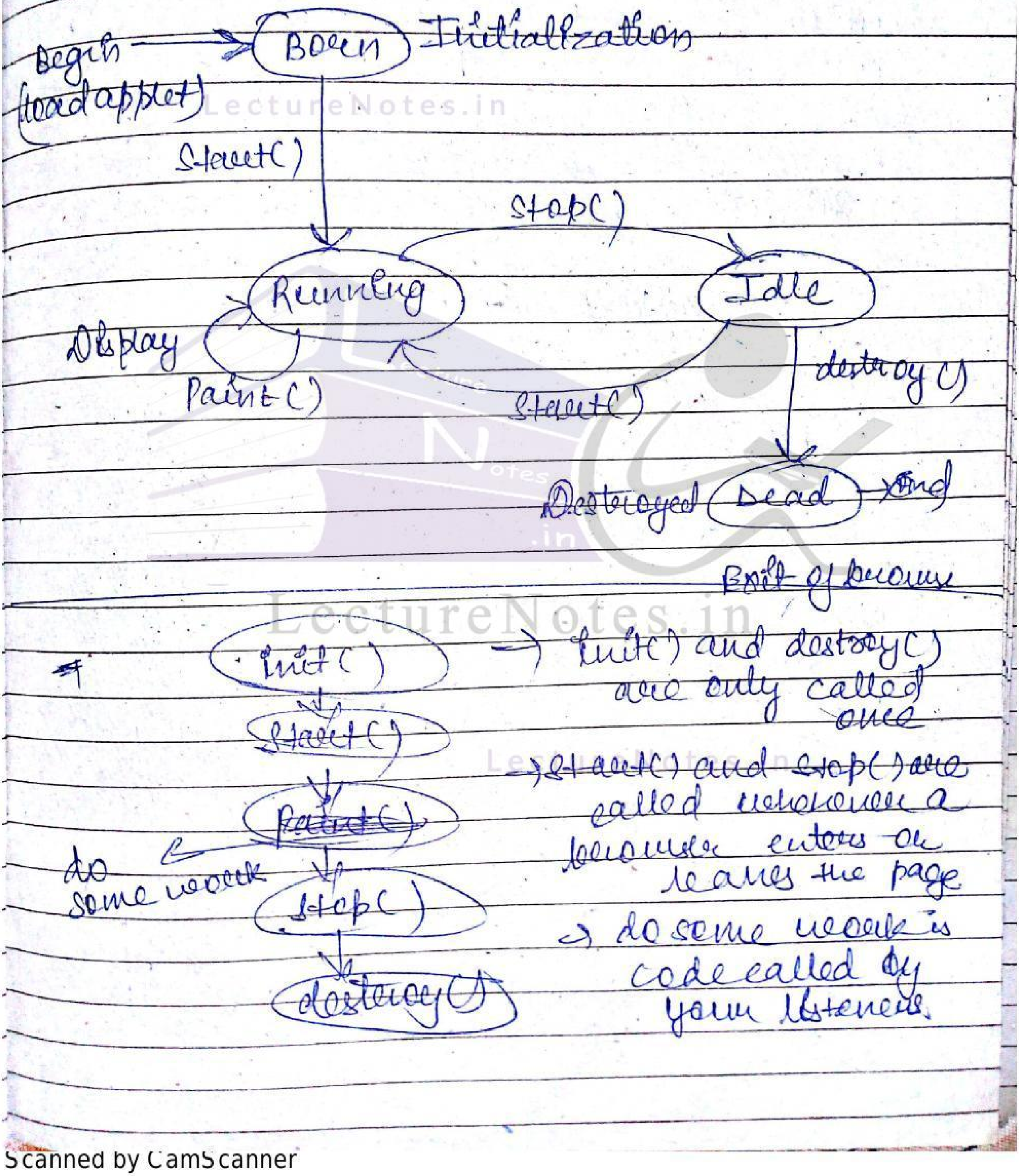
<applet code="Myapplet.class" width=200 height=300> </applet>

* Applets do not have main instead they have.

init()
start()
paint()
stop()
destroy()
update()
repaint()

Date / /
Page No.

Applet life - Cycle



Date / /
Page No.

Applet methods

public void init()
public void start()
public void stop()
public void destroy()
public void paint(Graphics)

(7). Why an applet needs?

→ We write an applet by extending Applet
→ Applet defines methods — init(), start(), paint(Graphics), stop(), destroy()

→ These methods are do nothing — they are stub.
→ We make the applet do something by overriding these methods

Date / /
Page No.

(i). The init() method

→ Executes only one time in the life of an applet.

→ init() method is called before the first time applet is loaded into the memory of the computer.

→ Initialize variables & components like buttons and check boxes

(ii). start() method

start() method is called immediately after the init() method and every time the applet receive focus as a result of scrolling in the active window.

→ We use this method, if we want to restart a process.

(iii). stop() method

→ used to reset variable and stop the threads that are running.

(iv). destroy() method.

→ destroy() method is called by the browser when the user moves to another page.

→ this method is used to perform cleanup operations like closing a file.

Date / /
Page No.

(v). Painting the applet :-

(vi). update() method

→ takes graphic class object as a parameter

→ update() method is called to clear the screen and call the paint() method.

→ screen is then refreshed by the system.

(vii) paint() method

→ draws the graphics of an applet in the drawing area.

→ The method is automatically called the applet is displayed on the screen & every time applet receives the focus.

→ The paint() method can be triggered by invoking the repaint() method.

(viii). repaint() method.

→ called when applet area to be redrawn.

repaint() method calls the update() method that the applet has to be redrawn. The default action of update()

Date / /
Page No.

method is to ~~call~~ represents the screen & call the paint() method.

drawString() method,
↳ member of graphic class.

Syntax;

void drawString(String message, int x, int y)

Ex: import java.awt.*;
import java.applet.*;

// <applet code = "applet2.class" width=200, height=200> </applet>
//

public class applet2 extends Applet {
int in=0, st=0, sto=0, des=0;
public void init()
{ in++;
repaint()
}
public void start()
{ st++;
repaint();
}

Date / /
Page No.

```
public void stop()
{
    sto=0;
    repaint();
}
public void destroy()
{
    des=0;
    repaint();
}
public void paint(Graphics g)
{
    g.drawString("Init has invoked"+String.valueOf(in),10,20);
    g.drawString("start has invoked"+String.valueOf(st),10,35);
    g.drawString("stop has invoked"+String.valueOf(sto),10,50);
    g.drawString("destroy has invoked"+String.valueOf(des),10,65);
}
}
```

Advantages of Java.

Thread || Applet

Date :
Page No.

10. Threads.

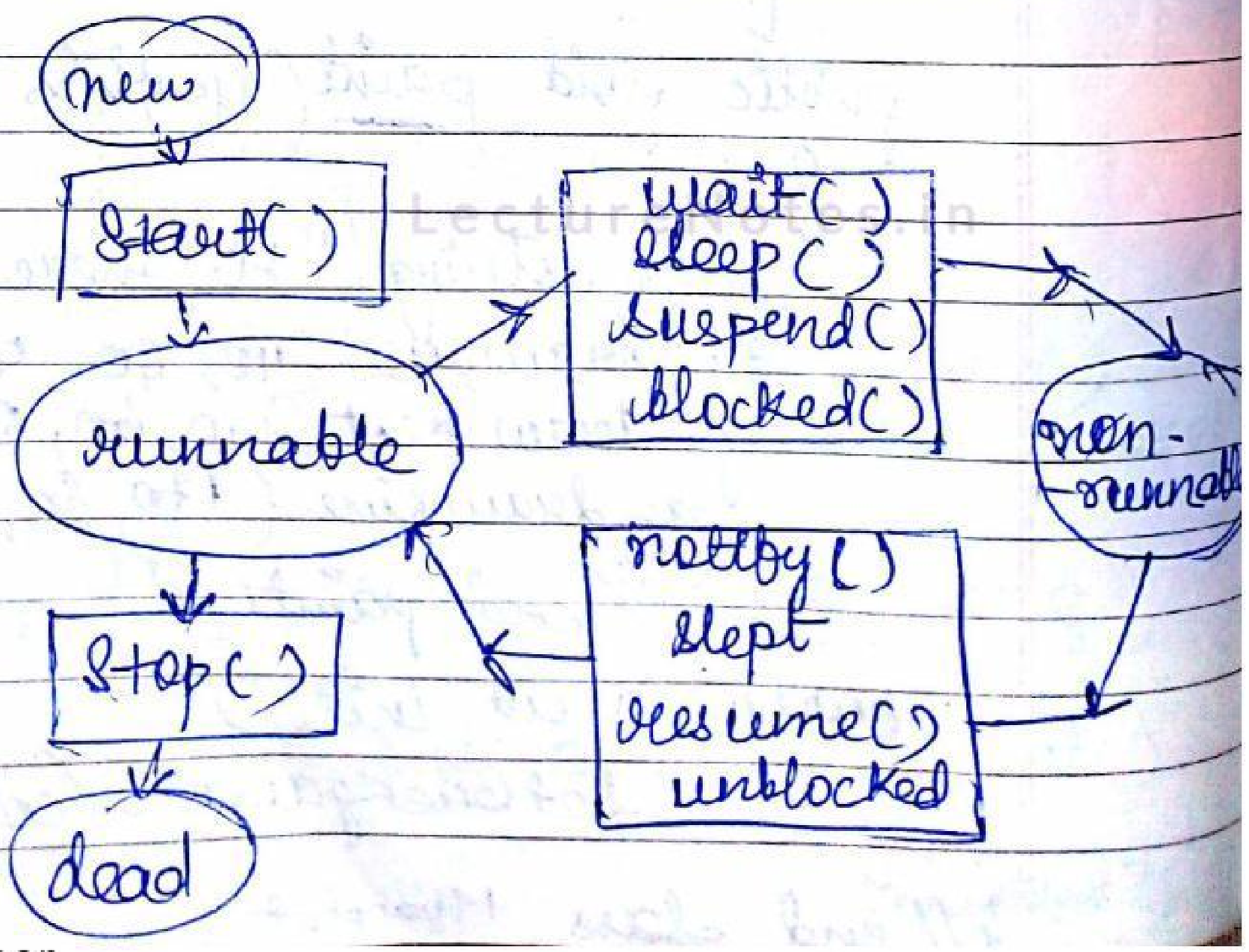
→ Java has built in thread support for multithreading.

→ Synchronisation.
→ Thread Scheduling.
→ Inter thread Communication

Current thread	start	setpriority
yield	run	getpriority
sleep	stop	suspend
		Resume

→ Java - Garbage collector is a low priority thread.

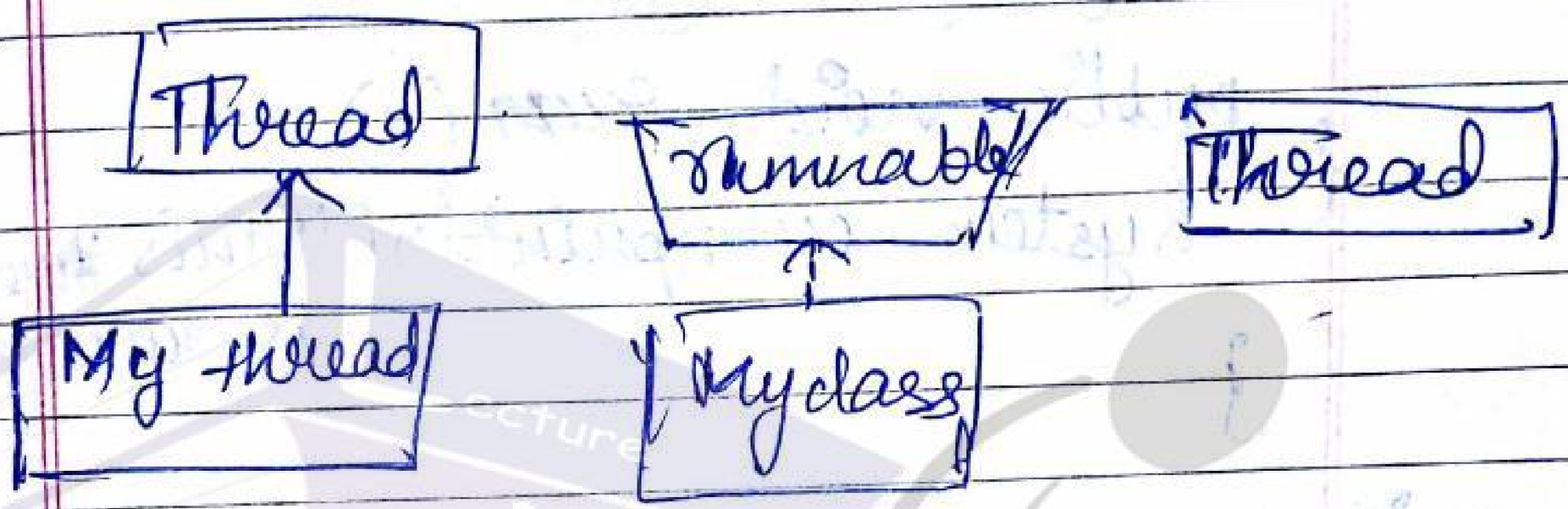
11. Thread States :-



Date :
Page No.

Threading mechanisms :-

- Create a class that extends the thread class
- Create a class that implements the runnable interface



1st method :- Extending the thread class.

```
class Mythread extends Thread
{
    public void run()
    {
        // thread body execution
    }
}
```

⇒ Creating Thread :
MyThread thr1 = new MyThread();

Date :
Page No.

Start Execution :

thr1.start();

Example 6-

```
class MyThread extends Thread
{
public void run()
{
System.out.println("This thread is running");
}
} // end class MyThread
```

Example 7 - // utilization of a thread.

```
class ThreadEx2 {
    public static void main(String[] a
    {
    MyThread t = new MyThread();
    t.start();
    }
}
```

Date :
Page No.

2nd method — for runnable Interf-
—ace.

```
class MyThread implements Runnable
{
public void run()
{
// - - -
}
}
```

⊕ Creating object &
MyThread myobject = new
MyThread()

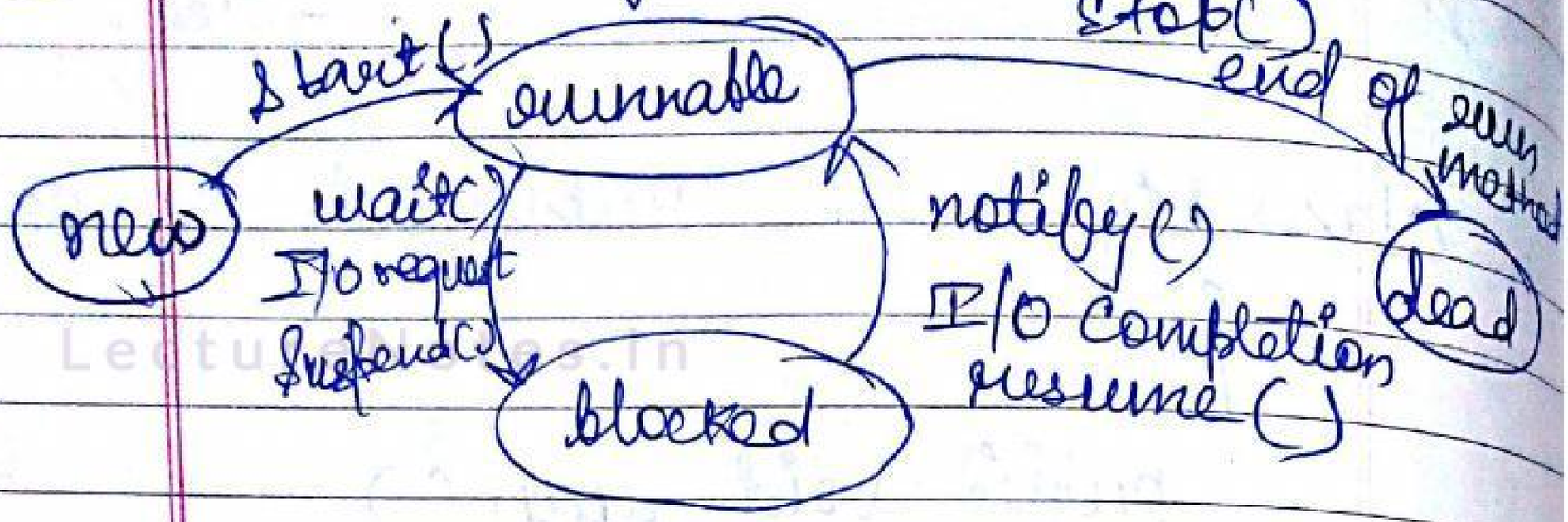
as)

e Creating Thread object.

Execution → top to bottom.

Date :
Page No.

4. States of Java Threads



5. Creating Threads Example -

```
class Outputs extends Thread {
    private String tosay;
    public Outputs (String st) {
        tosay = st;
    }

    public void run() {
        try {
            for (;;) {
                System.out.println (tosay);
                sleep (1000);
            }
        }
        catch (InterruptedException e) {
            System.out.println(e);
        }
    }
}
```

Date :
Page No.

```
class program{
public static void main(String[] args)
{
Output thr1 = new Output("Hello");
Output thr2 = new Output("There");
thr1.start();
thr2.start();
}
}
```

#. class Output implements Runnable
{
private string toSay;
public Output(String st) {
toSay = st;
}
public void run(){
try {
for(;;){
System.out.println(toSay);
Thread.sleep(1000);
}

Date :
Page No.

```
} catch(InterruptedException e){
        System.out.println(e);
    }
}

class program {
public static void main (String[] arg)
{
Output out1 = new Output ("Hello");
Output out2 = new Output ("There");
Thread thr1 = new Thread (out1);
Thread thr2 = new Thread (out2);
thr1.start();
thr2.start();
}
}
```

Multithreading

Multithreading is a conceptual programming concept where a program (process) is divided into two or more subprograms (process), which can be implemented at the same time in parallel. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.

A process consists of the memory space allocated by the operating system that can contain one or more threads. A thread cannot exist on its own; it must be a part of a process. There are two distinct types of Multitasking i.e. Processor-Based and Thread-Based multitasking.

Q: **What is the difference between thread-based and process-based multitasking?**

Ans: As both are types of multitasking there is very basic difference between the two. Process-Based multitasking is a feature that allows your computer to run two or more programs concurrently. For example you can listen to music and at the same time chat with your friends on Facebook using browser. In Thread-based multitasking, thread is the smallest unit of code, which means a single program can perform two or more tasks simultaneously. For example a text editor can print and at the same time you can edit text provided that those two tasks are perform by separate threads.

Q: Why multitasking thread requires less overhead than multitasking processor?

Ans: A multitasking thread requires less overhead than multitasking processor because of the following reasons:

Processes are heavyweight tasks where threads are lightweight

- Processes require their own separate address space where threads share the address
- space Interprocess communication is expensive and limited where Interthread
- communication is inexpensive, and context switching from one thread to the next is lower in cost.

Benefits of Multithreading

1. Enables programmers to do multiple things at one time.
2. Programmers can divide a long program into threads and execute them in parallel which eventually increases the speed of the program execution
3. Improved performance and concurrency
4. Simultaneous access to multiple applications

Life Cycle of Thread

A thread can be in any of the five following states

1. Newborn State: When a thread object is created a new thread is born and said to be in Newborn state.
2. Runnable State: If a thread is in this state it means that the thread is ready for execution and waiting for the availability of the processor. If all threads in queue are of same priority then they are given time slots for execution in round robin fashion
3. Running State: It means that the processor has given its time to the thread for execution.

A thread keeps running until the following conditions occurs

a. Thread give up its control on its own and it can happen in the following situations

i. A thread gets suspended using suspend() method which can only be revived with resume() method

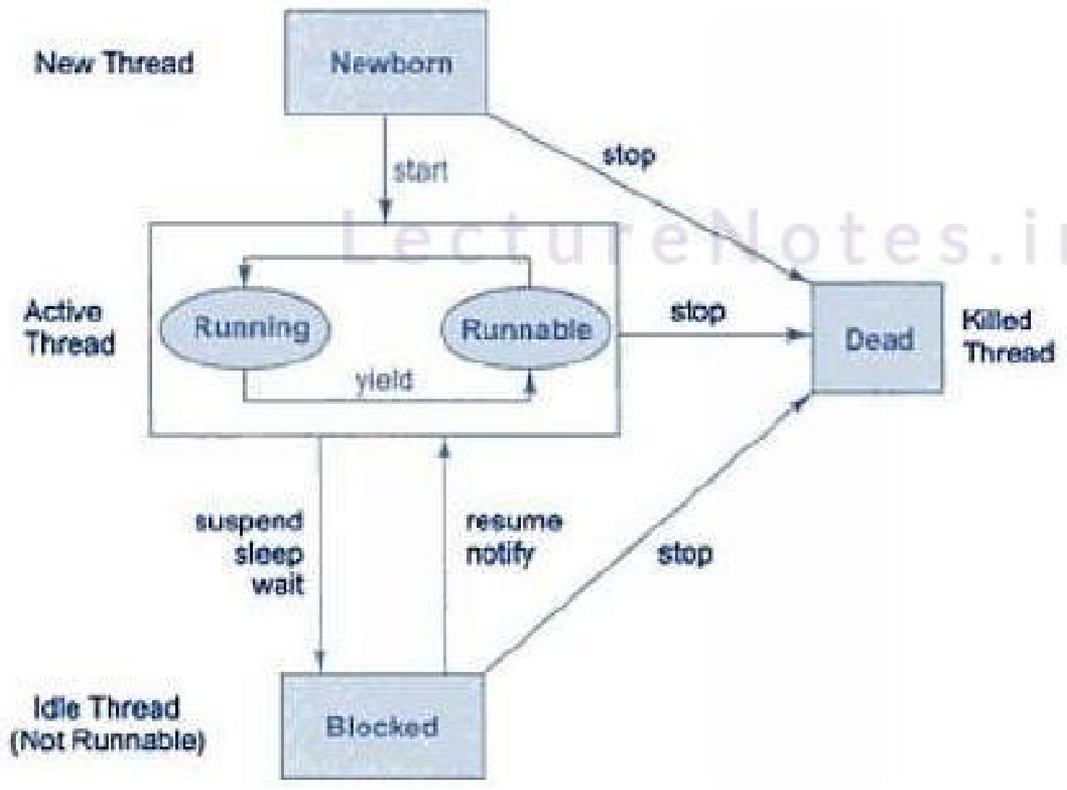
ii. A thread is made to sleep for a specified period of time using sleep(time) method, where time in milliseconds

iii. A thread is made to wait for some event to occur using wait () method. In this case a thread can be scheduled to run again using notify () method.

b. A thread is pre-empted by a higher priority thread

4. Blocked State: If a thread is prevented from entering into runnable state and subsequently running state, then a thread is said to be in Blocked state.

5. Dead State: A runnable thread enters the Dead or terminated state when it completes its task or otherwise terminates.



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.

Child threads are produced from main thread

- Often it is the last thread to finish execution as it performs various shut down operations

- Creating a Thread Java defines two ways in which this can be accomplished: You can implement the Runnable interface.

- You can extend the Thread class, itself.
- Create Thread by Implementing **Runnable**

The easiest way to create a thread is to create a class that implements the Runnable interface. To implement Runnable, a class need only implement a single method called run( ), which is declared like this:

public void run( )

You will define the code that constitutes the new thread inside run() method. It is important to understand that run() can call other methods, use other classes, and declare variables, just like the main thread can.
After you create a class that implements Runnable, you will instantiate an object of type Thread from within that class.
Thread defines several constructors.
The one that we will use is shown here:

Thread(Runnable threadOb, String threadName);

Here threadOb is an instance of a class that implements the Runnable interface and the name of the new thread is specified by threadName. After the new thread is created, it will not start running until you call its start( ) method, which is declared within Thread.
The start( ) method is shown here:

void start( );

Example to Create a Thread using Runnable Interface

```
class t1 implements Runnable
{
    public void run()
    {
        System.out.println("Thread is Running");
    }
    public static void main(String args[])
    {
        t1 obj1 = new t1();
        Thread t = new Thread(obj1);
        t.start();
    }
```

Create Thread by Extending Thread

The second way to create a thread is to create a new class that extends Thread, and then to create an instance of that class. The extending class must override the run( ) method, which is the entry point for the new thread. It must also call start( ) to begin execution of the new thread.

Example to Create a Thread by Extending Thread Class

```
class t2 extends Thread
{
    public void run()
    {
        System.out.println("Thread is Running");
    }
    public static void main(String args[])
    {
        t2 obj1 = new t2();
        obj1.start();
    }
}
```

Output:

```
C:\NIEC Java>javac t2.java

C:\NIEC Java>java t2
Thread is Running

C:\NIEC Java>
```

SN	Methods 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 was 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. The possible values are between 1 and 10.
5	**public final void setDaemon(boolean on)** A parameter of true denotes this Thread as a daemon thread.
6	**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.
7	**public void interrupt()** Interrupts this thread, causing it to continue execution if it was blocked for any reason.
8	**public final boolean isAlive()** Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

Q: Can we start a thread twice?

Ans: No, if a thread is started it can never be started again, if you do so, an illegal Thread State Exception is thrown. Example is shown below in which a same thread is coded to start again

```java
class t2 extends Thread
{
    public void run()
    {
        System.out.println("Thread is Running");
    }
    public static void main(String args[])
    {
        t2 obj1 = new t2();
        obj1.start();
        obj1.start();
    }
}
```

As you can see two statements to start a same thread is written in the code which will not give error during compilation but when you run it you can see an Exception as shown in the Output Screenshot.

Output:

```
C:\NIEC Java>javac t2.java

C:\NIEC Java>java t2
Thread is Running
Exception in thread "main" java.lang.IllegalThreadStateException
        at java.lang.Thread.start(Unknown Source)
        at t2.main(t2.java:11)
```

Important Thread Methods

1. **Yield Method** - Causes the currently running thread to yield to any other threads of the same priority that are waiting to be scheduled.

2. **Stop Method** - kills the thread on execution

3. **Sleep Method** - Causes the currently running thread to block for at least the specified number of milliseconds. You need to handle exception while using sleep() method.

4. **Suspend and Resume Method** - A suspended thread can be revived by using the resume() method. This approach is useful when we want to suspend a thread for some time due to certain reason but do not want to kill it.

Example

Thread Priority

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled. Java priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).

Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and very much platform dependent.

Example

```
class prioritytest
{
    public static void main(String args[])
    {
        C c = new C();
        A a = new A();
        a.setPriority(10);
        c.setPriority(1);
        c.start();
        a.start();
    }
}
```

In the above code, you can see Priorities of Thread is set to maximum for Thread A which lets it to run to completion ahead of C which is set to minimum priority.

Output:

```
C:\Achin Jain>java prioritytest
A:1
A:2
A:3
A:4
A:5
Exit from A
C:1
C:2
C:3
C:4
C:5
Exit from C
```

Use of isAlive() and join() method

The java.lang.Thread.isAlive() method tests if this thread is alive. A thread is alive if it has been started and has not yet died. Following is the declaration for java.lang.Thread.isAlive() method

public final boolean isAlive()

This method returns true if this thread is alive, false otherwise.

join() method waits for a thread to die. It causes the currently thread to stop executing until the thread it joins with completes its task.
Example

```
class A extends Thread
{
    public void run()
    {
        System.out.println("Status:" + isAlive());
    }
}
class alivetest
{
    public static void main(String args[])
    {
        A a = new A();
        a.start();
        try
        {
            a.join();
        }
        catch(InterruptedException e)
        {}
        System.out.println("Status:" + a.isAlive());
    }
}
```

At this point Thread A is alive so the value gets printed by **isAlive()** method is "***true***"

join() method is called from Thread A which stops executing of further statement until A is Dead

Now isAlive() method returns the value false as the Thread A is complete

Output

```
C:\Achin Jain>java alivetest
Status:true
Status:false
```

Synchronization

When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this synchronization is achieved is called thread synchronization. The synchronized keyword in Java creates a block of code referred to as a critical section. Every Java object with a critical section of code gets a lock associated with the object. To enter a critical section, a thread needs to obtain the corresponding object's lock.

synchronized(object) { // statements to be synchronized }

Problem without using Synchronization

In the following example method updatesum() is not synchronized and access by both the threads simultaneously which results in inconsistent output. Making a method synchronized, Java creates a "monitor" and hands it over to the thread that calls the method first time. As long as the thread holds the monitor, no other thread can enter the synchronized section of the code. Writing the method as synchronized will make one thread enter the method and till execution is not complete no other thread can get access to the method.

```
synchronized void updatesum(int i)
{
    Thread t = Thread.currentThread();
    for(int n=1; n<=5; n++)
    {
        System.out.println(t.getName()+" : "+(i+n));
    }
}
```



```
class update
{
    void updatesum(int i)
    {
        Thread t = Thread.currentThread();
        for(int n=1; n<=5; n++)
        {
            System.out.println(t.getName()+" : "+(i+n));
        }
    }
}
class A extends Thread
{
    update u = new update();
    public void run()
    {
        u.updatesum(10);
    }
}
class syntest
{
    public static void main(String args[])
    {
        A a = new A();
        Thread t1 = new Thread(a);
        Thread t2 = new Thread(a);
        t1.setName("Thread A");
        t2.setName("Thread B");
        t1.start();
        t2.start();
    }
}
```

```
synchronized void updatesum(int i)
```

Output when method is declared as synchronized

```
C:\Achin Jain>java syntest
Thread A : 11
Thread A : 12
Thread A : 13
Thread A : 14
Thread A : 15
Thread B : 11
Thread B : 12
Thread B : 13
Thread B : 14
Thread B : 15
```


Interthread Communication

It is all about making synchronized threads communicate with each other. It is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter in the same critical section to be executed. It is implemented by the following methods of Object Class:

wait( ): This method tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify( ).

notify( ): This method wakes up the first thread that called wait( ) on the same object.
notifyAll( ): This method wakes up all the threads that called wait( ) on the same object. The highest priority thread will run first. These methods are implemented as final methods in Object, so all classes have them. All three methods can be called only from within a synchronized context.

Example

```
class customer
{
int amount = 0;
int flag = 0;
public synchronized int withdraw(int amount)
{
    System.out.println(Thread.currentThread().getName()+" : is going to withdraw");
    if(flag==0)
    {
        try
        {
            System.out.println("Waiting....");
            wait();
        }
        catch(Exception e)
        {
        }
    }
    this.amount-=amount;
    System.out.println("Withdraw Complete");
    return amount;
}
public synchronized void deposit(int amount)
{
    System.out.println(Thread.currentThread().getName()+" : is going to Deposit");
    this.amount+=amount;
    notifyAll();
    System.out.println("Deposit Complete");
    flag=1;
}
}
class threadcomm
{
    public static void main(String args[])
    {
        final customer c = new customer();
        Thread t1 = new Thread()
        {
            public void run()
            {
                c.withdraw(5000);
                System.out.println("After withdraw Amount is :"+ c.amount);
            }
        };
        Thread t2 = new Thread()
        {
            public void run()
            {
                c.deposit(10000);
                System.out.println("After Deposit Amount is :"+ c.amount);
            }
        };
        t1.start();
        t2.start();
    }
}
```

If both these methods are commented which means there is no communication, output will be inconsistent. See **Output 2**

Page :
Date: / /

3

8 ;

AWT

GUI → elements of GUI include windows, dropdown list, buttons, scroll bar, icons, images & wizards.

AWT is a package that provides an integrated set of classes to manage user interface component like window, dialog box, push button, list, menus, scrollbar, & text-field,

→ provide support for UI containers
→ provides an event system for ~~provide~~ managing events.

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, radioButtons, textfield etc.

Page :
Date: / /

→ 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 sub classes 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 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 with given alignment

Page :
Date: / /

given text alignment.

available alignment

Label.Left
Label.Right
Label.center

Exp
c

```
import
import
public class Label1Demo extends Applet
{
    public void init()
    {
        Label one = new Label("one");
        add(one);
    }
}
```

Page :
Date: / /

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

Button()
Button(String)

Ex: Imp ——————
Imp ——————
public class Button1Demo extends Applet
{
——— Button 'yes'
public class init()
{
Button yes
Yes = new Button("yes");
add(yes);
}
}

(3). CheckBoxes :

→ selected or deselected to provide options.

Constructors :

Checkbox()
Checkbox(String) → used as a placeholder for a group
Checkbox(String, null, boolean).

Page:
Date: / /

To retrieve current state of checkbox call getstate().
To set its state, call setstate().

Methods:

Boolean getstate()
void setstate(boolean on) → if on is true, box is checked otherwise false
String getlabel()
void setlabel(String str)

Eg:

```
import java.awt.event.*;
import java.applet.*;
public class checkboxDemo extends
                                Applet
{
  Checkbox win1;
{
public void init()
{
win1 = new CheckBox("Windows",
                    null, true);
add(win1);
}
```

Page :
Date: / /

(4). Radio buttons :

→ Radio buttons are a variation on the checkbox.

→ have same appearences as checkboxes but only one in a series can be selected at a time.

ex:

```
Import ————————;
Import ————————;
public class CBGroup extends
                        Applet
{
    checkBox cb1, cb2, cb3;
    checkBoxGroup cbg;
    TextField tf;

public void init()
{

cbg = new checkBoxGroup();
tf = new TextField("", 30);
cb1 = new checkBox("Red", cbg,
                    null, true);
cb2 = new checkBox("Green", cbg,
                    null, false);

add(tf);
add(cb1);
add(cb2);
}
}
```

Page:
Date: / /

(5). Choice menus

→ pull-down menus that enable us to select an item from that menu.

Ex6 Cmp

Cmp — choiceDemo

public class extends Applet

{

choice OS, Browsers

String msg = " ";

public void init();

OS = new choice();

OS.add("WindowsXP");

OS.add("WindowsVista");

OS.add("Macos");

OS.add("Solaris");

add(OS);

}

Page:
Date: / /

6. Textfield → handles a single line text.

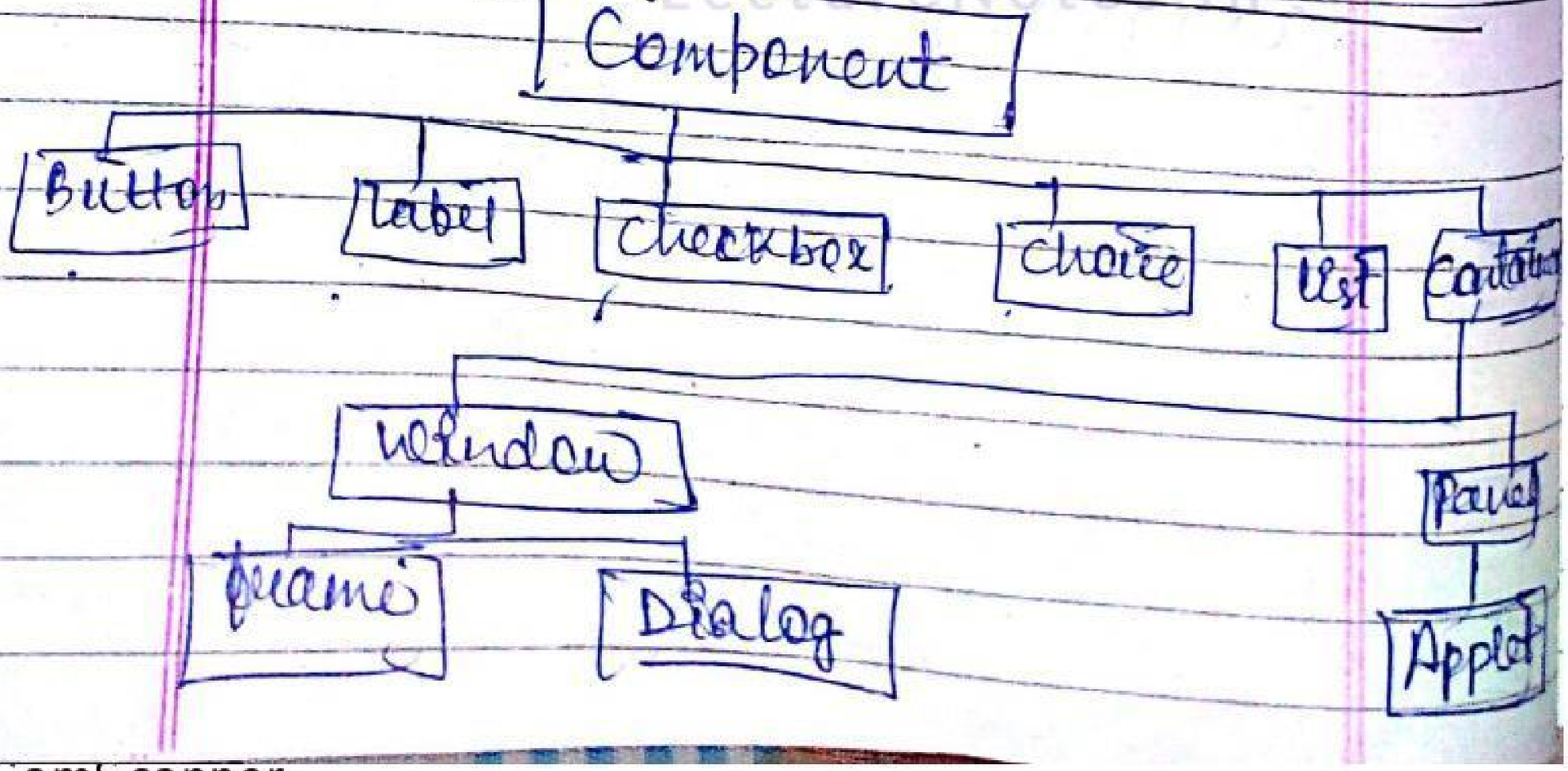
~~Method:~~

TextArea → handles multiple lines of text.

Methods:

```
textfield tf = new Textfield();
                └Empty constructor.
Textfield tf = new Textfield(10);
Textfield tf = new Textfield("Enter name");
add(tf);
```

Class Hierarchy of Component class:



Page:
Date: / /

#. Container :

→ Container is a Component which contains other Components.
→ Containers are themselves Components.
→ Components are contained in container.
→ Swing Components subsist directly or indirectly from the JComponent class, which is an indirect subclass of old AWT Component class.

#. Panel

→ Sub-class of container
→ doesn't add any methods.
→ Implements Container
→ when screen output is directed to an applet, it is drawn on the surface of panel object.
→ components can be added by add() method.
→ doesn't contain little bar, menu bar or border.

Create a panel;
Panel p1 = new Panel();

Page :
Date: / /

add panel to a window or frame

```
Frame F1 = new Frame ("Frame window");
Panel P1 = new Panel ();
F1.add (P1);
```

Page :
Date: / /

Layout management

→ layout management is the process that determines the size and position of components in a container.

→ manges the layout of components in a container.

→ Every component has a default layout manager.

→ Each time we create a container, java automatically creates a default layout manager.

→ layout managers determines the placing of control in the applet display.

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

→ Set by setLayout() method.

→ If no call is made then default layout manager is used.

Page :
Date: / /

void setLayout(Layout Manager layout obj)

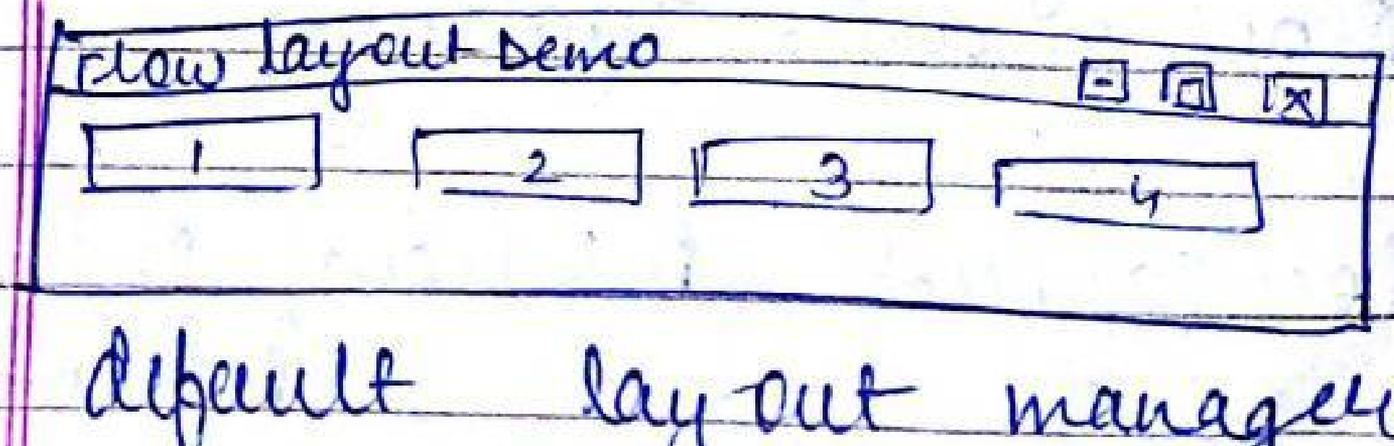
AWT provides following layout managers

Flow layout
Grid layout
Border layout
Card layout
Grid Bag layout

Flow layout is most basic of layout.

→ Components are added to the panel one at a time, row by row.

→ has an alignment, which determines the alignment of each row.



→ default layout manager.

Page :
Date: / /

Constructors for flow layout :
FlowLayout()
FlowLayout(int how)
FlowLayout(int how, int hor, int ver)

(1) FlowLayout()
→ default layout, which centres components and leaves five pixels of space between each component.

(2) FlowLayout(int how)
→ lets us specify how each line is aligned. Valid values for how are :

FlowLayout.LEFT
FlowLayout.Center
FlowLayout.Right
FlowLayout.leading
FlowLayout.Trailing

(3) FlowLayout(int how, int hor, int ver)
→ specify the horiz. & vertical space left between components.

Page :
Date: / /

Ex6 import java.applet.Applet;
import java.awt.*;
public class Flow1 extends Applet
{
public void init()
{
Button b1, b2, b3;
FlowLayout flow;
Flow = new FlowLayout(FlowLayout.
LEFT, 10, 10);
setLayout(flow);
b1 = new Button("Button 1");
b2 = " " (" " 2");
b3 = " " (" " 3");
add(b1);
add(b2);
add(b3);
}
};

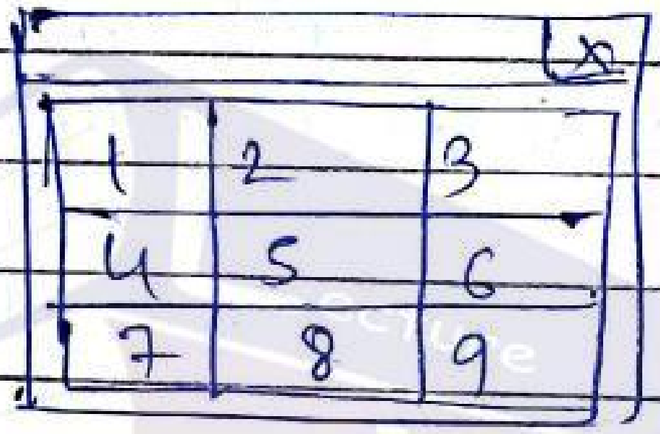
and one used //

Page :
Date: / /

Grid Layout

→ layouts component in two dimensional grid.

→ we defines number of rows & columns.



GridLayout() → single column grid layout created

GridLayout(int numRows, int numColumns)

GridLayout(int numRows, int numColumns, int horz, int vert),

→ creates grid layout with the specified number of rows & columns.

specify the horizontal & vertical space left between columns.

Page:
Date: / /

Ex6
```
Import java.applet.Applet;
Import java.awt.*;
Public class Grid extends Applet
{
Public void init()
{
Button b1,b2,b3;
GridLayout G1;
G1 = new GridLayout(2,2,10,10);
setLayout(G1);
Button B1 = new Button("Button 1");
B2 = "  "  "Button 2";
B3 = "  "  "Button 3";
add(b1);
add(b2);
add(b3);
}
}
```

Page :
Date: / /

Border Layout :

⇒ ~~Bro~~ Indicate geographic direction
o north, south, east, west and
centre.

Borderlayout() → default border layout
Border.Layout(int horz, int vert)
Specify how to vertical
space between components.

Regions —
BorderLayout.Center
" . East
" . West
" . North
" . South,

.add()
Void add (Component compobj, object regid)
SetLayout (new BorderLayout());

Page :
Date: / /

Ex6
```
import java.applet.Applet;
import java.awt.*;
public class Border1 extends
                        Applet
{
public void init()
{
    Button b1, b2, b3;
    BorderLayout bord1;
bord1 = new BorderLayout();
setLayout (bord1);
   b1 = new Button ("Button 1");
   b2 = "  "  ("  "  2");
   b3 = "  "  ("  "  3");
add ("North", b1);
 add ("South", b2);
 add ("East", b3);
 }
}.
```

Page :
Date: / /

Card Layout :

→ Different from other layouts.
Unlike the other three layouts, when you add component to a Card layout, they are not all displayed on the screen.
→ Card layout are used to produce slide shows of components, one at a time.
→ Only one component is visible at a time.

Constructors :

CardLayout ( )
CardLayout (int horz, int vert)

Ex :
```
void import java.applet.Applets
import a java.awt.*;
public class Card1 extends Applet
{
    public void init()
    {
        Button b1, b2, b3;
        CardLayout c1;
        c1 = no
        panel P1 = new Panel ( );
        add (P1);
```

Page :
Date: / /

```
C1 = new CardLayout();
P1.setLayout(C1);
    b1 = new Button("Button 1");
    b2 =  "       "     ("   "  2");
    b3 =  "       "     ("   "  3");
P1.add("Button 1", b1);
P2.add("   "  2", b2);
P3.add("   "  3", b3);
}
}
```

GridBag Layout :-

→ most flexible & complex.

→ Can resize the components by assigning weights to individual components in the gridbag layout.

→ when we specify the size & position of components you also need to specify the constraints for each component.

Page :
Date: / /

To specify the constraints, we need to specify variables. set the setConstraints() method.

→ Has Single Constructor, GridBagConstraints con = new GridBagConstraints();

* Specifying Constraints :

- anchor
  GridBagConstraint.East
  " . North
  " . West
  " . South
  " . North East
  " . North West
  " . South East
  " . South West
  " . Centre

- Fill
  - None
  - Horizontal
  - Vertical
  - Both.

Page :
Date: / /

Examples

```
Emp ——————
Emp ——————
public class gb1 extends App——
{
Public void init()
{
Button b1, b2, b3, b4, b5, b6;
GridBagLayout g1;
g1 = new GridBagLayout();
GridBagConstraints gbc;
setLayout(g1);
gbc = new GridBagConstraints();
b1 = new Button("Button 1");
b2 = "      "  ("  2
b3 = "      "  (
b4 = "      "  (
b5 = "      "  (
b6 = "      "  (
gbc.fill = GridBagConstraints.BOTH;
gbc.anchor = "      ".CENTER;
gbc.gridwidth = 1;
gbc.weightx = 1.0;
g1.setConstraints(b1, gbc);
```

Page :
Date: / /

Insets: If we want to leave the space between the container's components and the windows that contains it. To do this, override the getInsets() method.

Insets(int top, int left, int bottom, int right)

specify amount of space between the container & enclosing windows.

Eg:

```
Imp
Imp
Public class InsetsDemo extends
Applet
{
Public void init()
{
setBackground(Color.cyan);
}
Public Insets getInsets()
{
return new Insets(10,20,10,10);
}
```

Event Handling

→ An event occurs when something changes within a graphical user interface.

→ Java Events are the part of the Java AWT package.
An event is the way the AWT communicates to you.

Page :
Date: / /

Process Involves In Event Handling Includes:

1. Identifying where an Event should be processed
2. forward the Event
3. receive the forward event
4. Take some appropriate action in response.
5. Event Handler may ultimately forward the Event to an Event Customer.

Components of An Event:

Event Object

When the user interacts with the application by pressing a key or a clicking a mouse button, an event is generated.
The operating System traps the event & data associated with it.
This data is then passed onto the application to which the event belongs.
In Java, Events are represented

Page:
Date: / /

by objects that describes the events themselves.

Event Sources

→ An Event source is an object that generates an event.

Event Handler (Event listener)

→ Method that understands the event & processes it.
→ the Event Handler method takes an event object as a parameter.
→ Listener is an object which is notified when an event occurs.

The Event Model (Delegation Event Model)

In this model, we can specify the objects that are to notify when specific event occurs.

Page :
Date: / /

→ Advantage of this design is that the application logic that processes event is clearly separated from the User Interface logic that generates those events.

This model works in following manner.

1. Source generates an event and sends it to one or more listener
2. The listener simply waits until it receives an event. Once it receives an event, it process event and returns back.
3. Listener must register with a source, in order to receive event notifications.

Date :
Page No.

Event classes	Listener Interface
Action Event	Action Listener
Mouse Event	Mouse listener and Mouse motion Listener
Mouse Wheel Event	Mouse wheel listener
Key Event	Key listener
Item Event	Item Listener
Text Event	Text Listener
Adjustment Event	Adjustment listener
Window Event	Window listener
Component Event	Component listener
Container Event	Container listener
Focus Event	Focus listener

Swing → used to create window based appl
↳ build at the top of awt.
↳ javax.swing

Eg6 Java Event Handling

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements
                                ActionListener
{
  TextField tf;
  AEvent()
  {

    tf = new TextField();
    tf.setBounds(60, 50, 170, 20);

    Button b = new Button("clickme");
    b.setBounds(100, 120, 80, 30);
    b.addActionListener(this);

    add(b);
    add(tf);
    setSize(300, 300);
    setLayout(null);
    setVisible(true);
  }
  public void actionPerformed(ActionEvent
                                e)
  {
    tf.setText("welcome");
  }
```

Date :
Page No.

```
public static void main(String arg
{
    new ABrent();
}
}
```

(#) Difference btw AWT and Swing

AWT	Swing
→ platform dependent	→ platform Independent
→ less Component	→ more component
→ heavyweight	→ lightweight

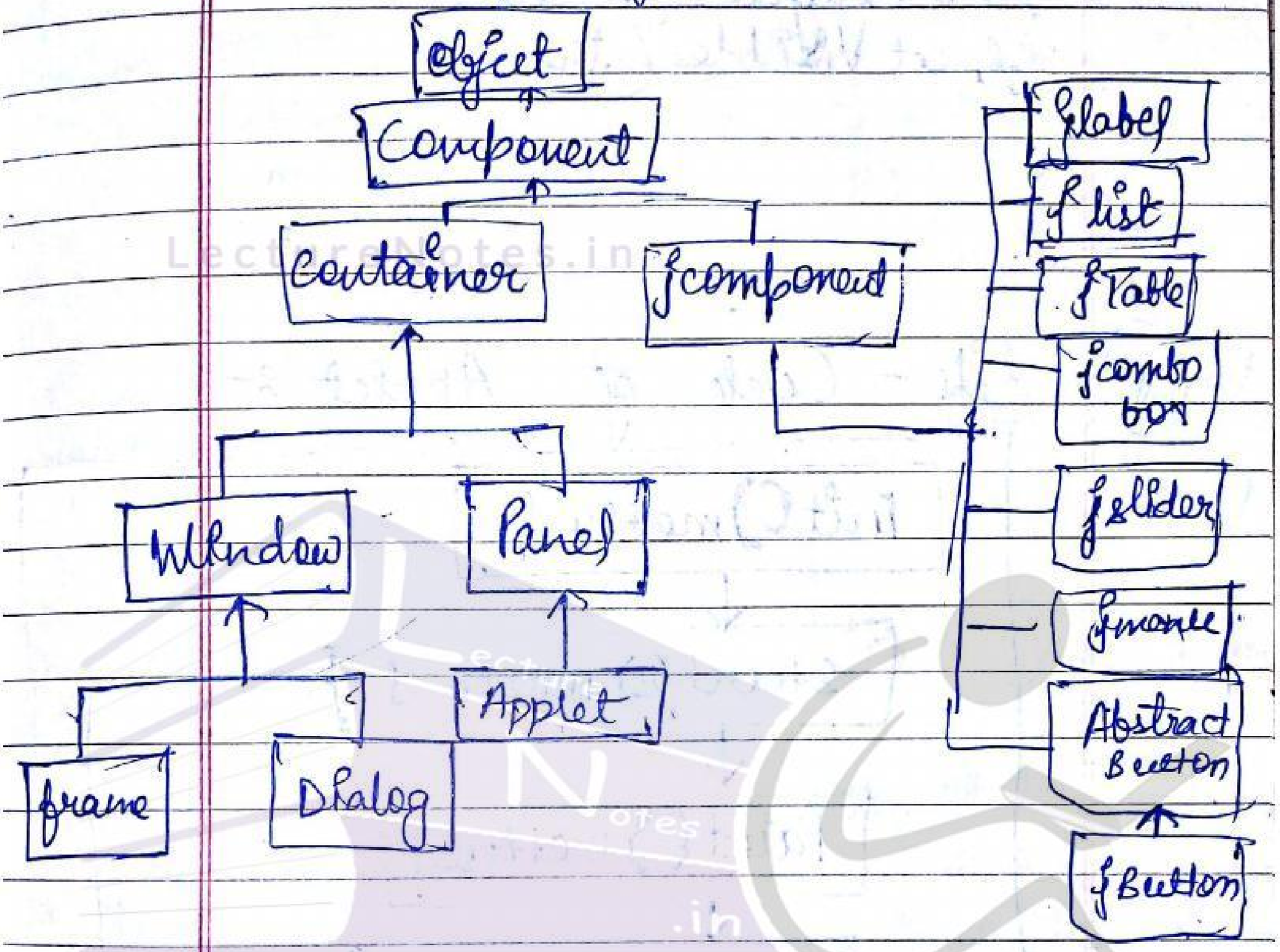
MVC
↓
model repres. data

→ View represents presentation

→ Controller represents Interface btw m & V

Date :
Page No.

D) Hierarchy of java Swing Classes



Eg:-

```
Import javax.swing.*;
public class FirstSwingExample
{
public static void main(String[]
                        args)
{
JFrame f = new JFrame();
 JButton b = new JButton("click");
b.setBounds(130, 100, 100, 40);
f.add(b);
```

Date / /
Page No.

Event Handling ex6

```
import java.awt.*;
import java.awt.event.*;

class AEvent extends Frame implements
                    ActionListener{
TextField tf();
AEvent();
{
tf= new TextField();
tf.setBounds(60,120,80,30);
Button b= new Button("click me");
b.setBound(120,180,60,30);
b.addActionListener(this);
add(b);
add(tf);

setSize(300,300);
setLayout(null);
setVisible(true);
}
}
public void ActionPerformed(ActionEvent
e)
{
tf.setText("welcome");
```

Date / /
Page No.

```
4
Public static void main (String args[])
{
newAfrent();
}
}
```

```
Q Outer class
import java.awt.*;
import java.awt.event.*;
class AFrame extends Frame implements ActionListener
{
TextField tf;
AFrame2();
{
tf = new TextField();
tf.setBounds(100,200,60,30);
Button b = new Button("click me");
b.setBounds(80,120,20,30);
Outer o = new Outer(this);
b.addActionListener(o);
add(b);
add(tf);
```

Date / /
Page No.

```
setSize(800,300);
setVisible(true);
setLayout(null);
}
public void static main(String args[])
{
new ABwent2();
}
}
{
import java.awt.event.*;
class Outer implements ActionListener{
ABwant2 obj;
Outer(ABwent2 obj){
this.obj = obj;
}
```

Page No.

Unit - IV

Binary I/O is more efficient than text I/O, because binary I/O does not require encoding and decoding.

Binary files are free from encoding ~~and~~ ~~its~~ on host machine & thus are portable.

Java I/O → is used to process the input and produce the o/p based on the input.

File

Data can be read from or stored in file. Data stored in file is often called persistent data.

→ File is collection of related records placed in a particular area in the disk.

→ File is a class that directly deals with file system. It does not specify how data is retrived from or stored in file. It just deals with properties of file.

Date :
Page No.

→ This object is used to obtain information for associated with file such as, date, time, directory path.

Directory is a file that contains list of other files and directories.

We call list() method to ~~call~~ extract the list of other files and directories.

file methods :-
exists()
isdirectory()
isfile()
canread()
canwrite()

Streams

→ Streams is an abstract demonstration of input and output devices.
→ Stream is an interface between I/O devices and users.
→ Stream helps in smooth transfer of data to the desired place.

Date :
Page No.

→ Data stored in binary file are represented in the binary form.

Eg: Java source programs are stored in text files & can be read by any text editor and Java classes are stored in binary file & can be read by JVM.

The Java Input/output (I/O) is a part of java.io package.

The java.io.package contains a relatively large number of classes that support input and output operations.

Java defines 2 types of streams Byte stream and character stream.

Standard Stream ↔

→ They read Input from keyboard and write output to the display.
Java supports three standard Streams ↔

Date :
Page No.

System.in → used to read I/P from keyboard
System.Out → used to write o/p to display
System.err → used to write error output to be display.

- System.in → Standard Input stream
- System.out → Standard output stream
- System.err → Standard error stream.
- both are used to write output;

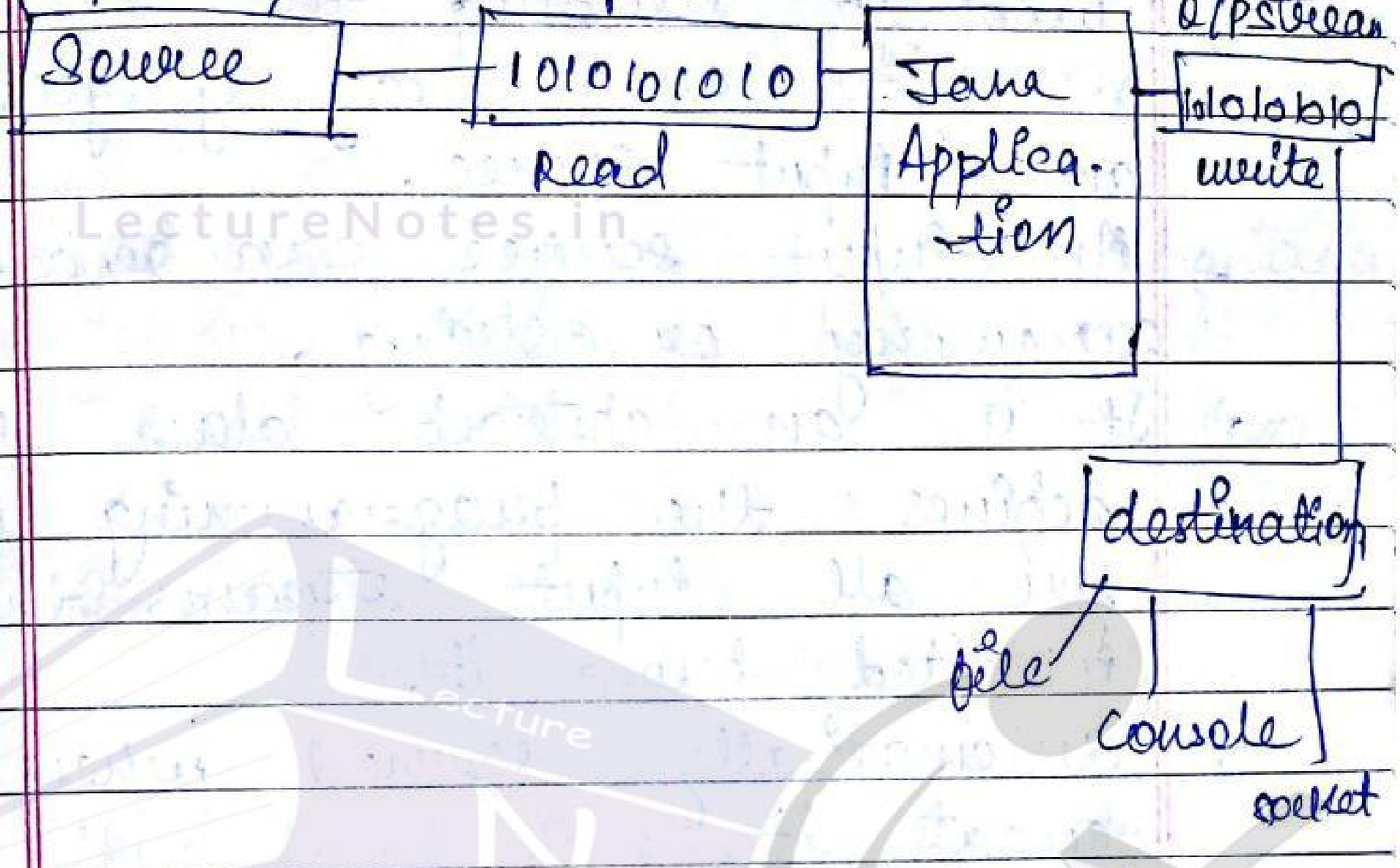
↳ SystemofIn → byte stream that has no character - stream features.

to use standard input as a character stream :-

InputStreamReader inp = new InputStreamReader(System.in);

Date :
Page No.

Working of Java output stream and Input stream :



Streams are unidirectional in nature.

There are two types of stream for file handling in Java
- Byte stream
- Character stream.

Byte stream : Supports 8-bit input and output operations.

There are two classes of byte stream.
- Input stream
- Output stream.

Date :
Page No.

Input stream :

→ Represents Input stream of bytes.
→ used for reading the data such as bytes or array of bytes from an Input source.
→ An Input source can be a file, a memory or string.
→ It is an abstract class that defines the programming Interface for all Input streams that are Inherited from it.
→ Automatically opened when we create it.
→ Explicitly closed by close() method and Implicitly closed when the object is bound as a garbage.

- Sub classes Inherited from Input Stream class can be seen in hierarchy manner

1, ByteArray Input stream
2, File Input stream
3, Object Input stream
4, Pipe Input stream
5. StringBuffer Input Streams
6. Filter Input stream

Date :
Page No.

7. Data Input Stream -
8. Buffered Input Stream
9. Line number Input Stream.
10. Push Back Input Stream.

Byte array Input Stream → It contains the internal buffer and read data from the stream

File Input Stream → contains Input bytes from a file.

Object Input Stream → used to recover objects to serialize previously.

Piped Input stream → Input pipe.

String Buffer → used to buffer the string that comes through Input device.

Filter Input Stream → Implements Input Streams

Buffered Input Stream → Used to create internal buffer array

Date :
Page No.

Data Input Stream → This class reads the primitive data type from Input stream

Line number Input Stream → Counts the lines.

PushBack Input Stream → allows characters to be returned to Input stream.

Output Stream :

This class is sibling to the Input stream class used to write byte and array of bytes to the output source.

- O/P Sauce can be - a file, a string, or a memory
- Opened automatically
- closed either explicitly by call() method or implicitly when object is garbage collected.

⇒ Subclasses / classes inherited from an output stream in hierarchy are :

1. ByteArray Output stream
   - data is written into byte array.

Date :
Page No.

2. File Output stream – data is written to a file.
3. Object output stream – write the object to read by object Ip stream
4. piped Output Stream – Output pipe
5. String Buffer Output stream – used to buffer string.
6. Filter Output stream – Implements Outputstream
7. Buffered output stream – writes byte to output stream
8. Data Output stream – writes the primitive data type
9. Print stream – Output stream that contains Print() and println() methods.

Date :
Page No.

Character Streams

→ Supports 16-bit unicode character input and output.

→ There are two classes of character streams ~~to be~~ :

- Reader
- writer

Reader

1. Buffered Reader
2. Line number Reader
3. CharArray Reader
4. Filter reader
5. Piped reader
6. String reader
7. Push back reader
8. File reader
9. Input stream reader

File Input Stream / File Output Stream

→ File Input stream and File Output stream are used to read or write data in file.

→ ~~There~~ All the methods in these classes are inherited from Input stream & Output stream. They do not introduce new methods.

Date :
Page No.

→ They are used for file handling in java.

Java FileOutputstream class :

Almost all the methods in the I/O classes throw java.io.IOexception.

therefore we have to place the code in a try-catch block as;

Syntax :

```
{ public static void main (String[] args)
    try {
        // perform IO exception.
    }
    catch (Exception e)
    {
        System.out.println (e);
    }
}
```

Date :
Page No.

Example of Java FileOutputStream class:-

```
import java.io.*;
class Test {
public static void main (String args[]){
{
try {
FileOutputStream fout = new FileOutputStream
("abc.txt");
String s = "Sachin is the best";
byte b [] = s.getBytes();
fout.write(b);
fout.close();
System.out.println("Success");
}
catch(exception e)
{System.out.println(e);
}
}
}
```

Date :
Page No.

Java File InputStream class :

→ obtains input bytes from a file
→ used for reading streams of byte.

Example :

```
import java.io.*;
class SimpleRead {
public static void main(String args[])
{
try {
FileInputStream fin = new FileInputStream("abc.txt");
int i = 0;
while ((i = fin.read()) != -1).
{
System.out.println((char)i);
}
fin.close();
}
Catch(e) { System.out.println(e); }
}
}
```

Date :
Page No.

Example

Reading the data of one java file and writing it to another file.

```
import java.io.*;
class CS{
public static void main (String args[]){
try{
FileInputStream fin = new FileInputStream
                          ("C.java");
FileOutputStream fout = new FileOutputStream
                          ("M.java");
int i=0;
while((i=fin.read())!=-1)
{System.out.print((char)i);}
fout.write(byte) i);
}
fin.close();
}
}
```

Date :
Page No.

Buffered Streams :

Buffered Input streams reads ~~data~~ data from a memory area called as buffer, Similarly, Buffered Output Streams writes ~~data~~ data ~~from~~ to buffer.

There are four buffered stream classes :
1. BufferedInputStream
2. BufferedOutputStream
3. BufferedReader
4. BufferedWriter

BufferedInputStream :

→ Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.

⇒ Constructions of buffered Input Stream.
- public BufferedInputStream(InputStream in, int size)

Date :
Page No.

Parameters :
in → the underlying input stream
size → the buffer size.

Example of Java BufferedInputStream

```
import java.io.*;
class SimpleRead {
  public static void main(String args[])
  {
    try {
FileInputStream fin = new FileInputStream
                        ("f1.txt");
BufferedInputStream bin = new BufferedInput
                          Stream
                          (fin);
int i;
while ((i=bin.read()) != -1)
  {
System.out.println((char)i);
  }
bin.close();
fin.close();
} catch (Exception e)
  {
    System.out.println(e);
  }
}
```

Date :
Page No.

Buffered Output Stream :

Ex6 import java.io.*;
class Test {

```
public static void main(String args[])
{
try {
FileOutputStream fout=new FileOutputStream
("f1.txt");
BufferedOutputStream bout = new
BufferedOutputStream
(fout);
String s="Sachin is the best";
byte b[] = s.getBytes();
bout.write(b);

bout.flush();
bout.close();
fout.close();
System.Out.println("success");
}
}
```

Date / /
Page No.

JDBC

Introduction to JDBC

JDBC stands for database connectivity. JDBC is a set Java API's (Application programming interface) used for executing SQL statements.

→ API contains a set of classes and interfaces to enable the programmer to develop pure Java database applications.

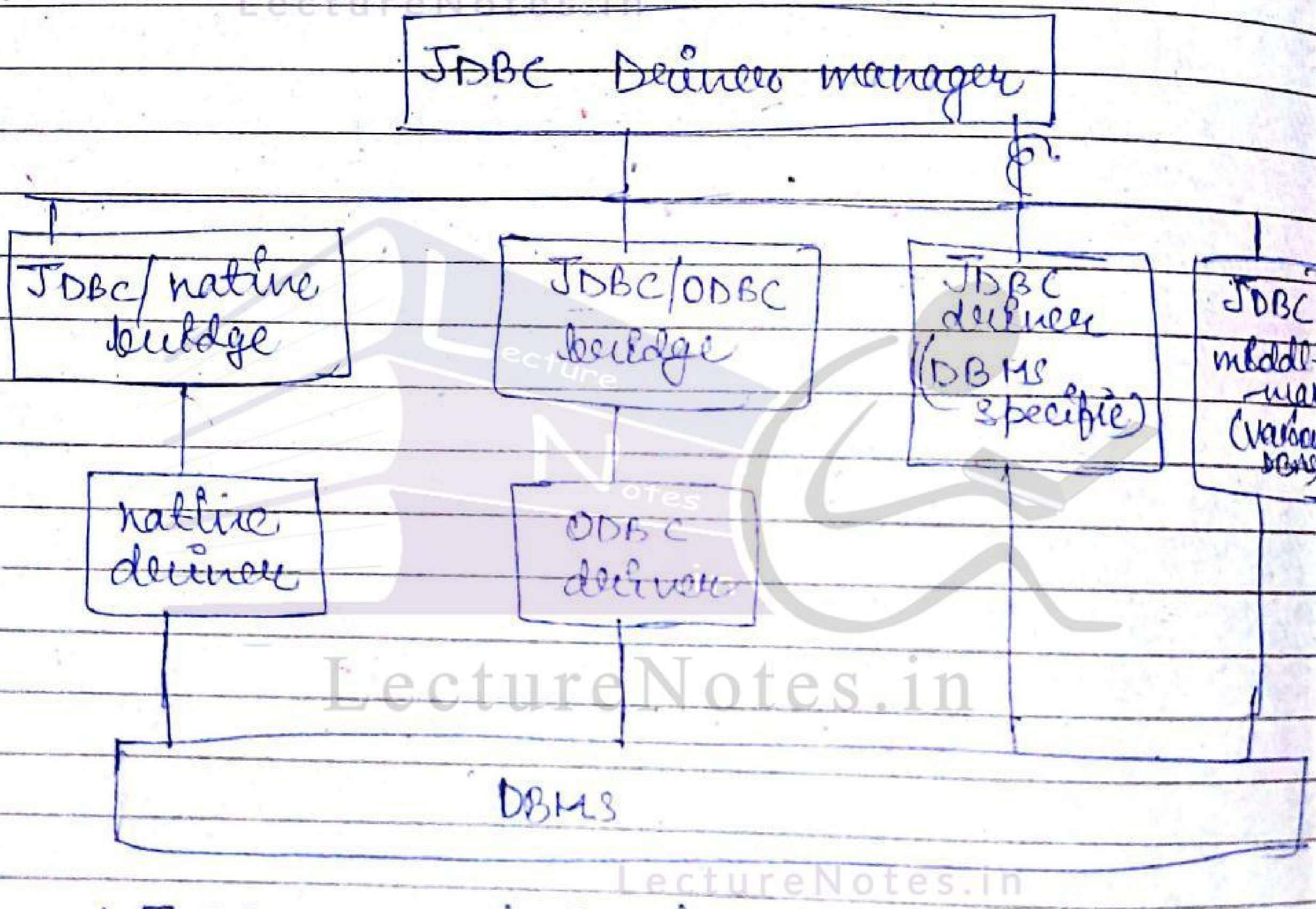
→ Allows developers to write real client-server projects in java.

→ JDBC API defines how ~~any connection~~ application opens a connection, communicates with database, executes SQL statements & retrieve query results.

Date / /
Page No.

Role of JDBC

Java Application



→ JDBC was designed to be very compact, simple interface focusing on the execution of raw SQL-statements & retrieving the results.

Date / /
Page No.

Characteristics of JDBC

→ It is a call-level SQL interface in Java.
→ JDBC mechanism are simple to understand & use.
→ It does not restrict the type of queries passed onto the underlying DBMS driver.
→ provides Java interface that stays consistent to the rest of the Java system.

⁂. Java and JDBC

→ The combination of Java with JDBC is very useful for the programs to run ~~desktop~~ his/her program on any platform.
→ Java programs are secure, robust and Java is a good language to create database application.

Advantages of using Java with JDBC
- Easy and Economical
- continued usage of already installed database.

Date / /
Page No.

- Development time is short
- Installation is simplified.

How JDBC Works ?

→ JDBC defines set of objects & methods to interact with underlying database.

- A Java program first opens a connection, makes a statement object, passes the SQL statement using these statement object to the underlying DBMS and retrieve the result as well as information about result set.

There are two type of interfaces ⊖
low level interface. (not user friendly)
high level interface.
(user-friendly)
JDBC is a low level API interface.

Date / /
Page No.

JDBC v/s ODBC

~~The most use~~

→ used to access relational database.

JDBC is preffered of the two &

1. ODBC cannot be directly used with Java because it uses a C interface. Calls from Java to native C code have a number of drawbacks in the security, implementation etc.

2. ODBC makes use of pointers which have been totally removed from Java.

3. ODBC mixes simple and advanced features together and has complex options for simple queries. But JDBC is designed to keep simple.

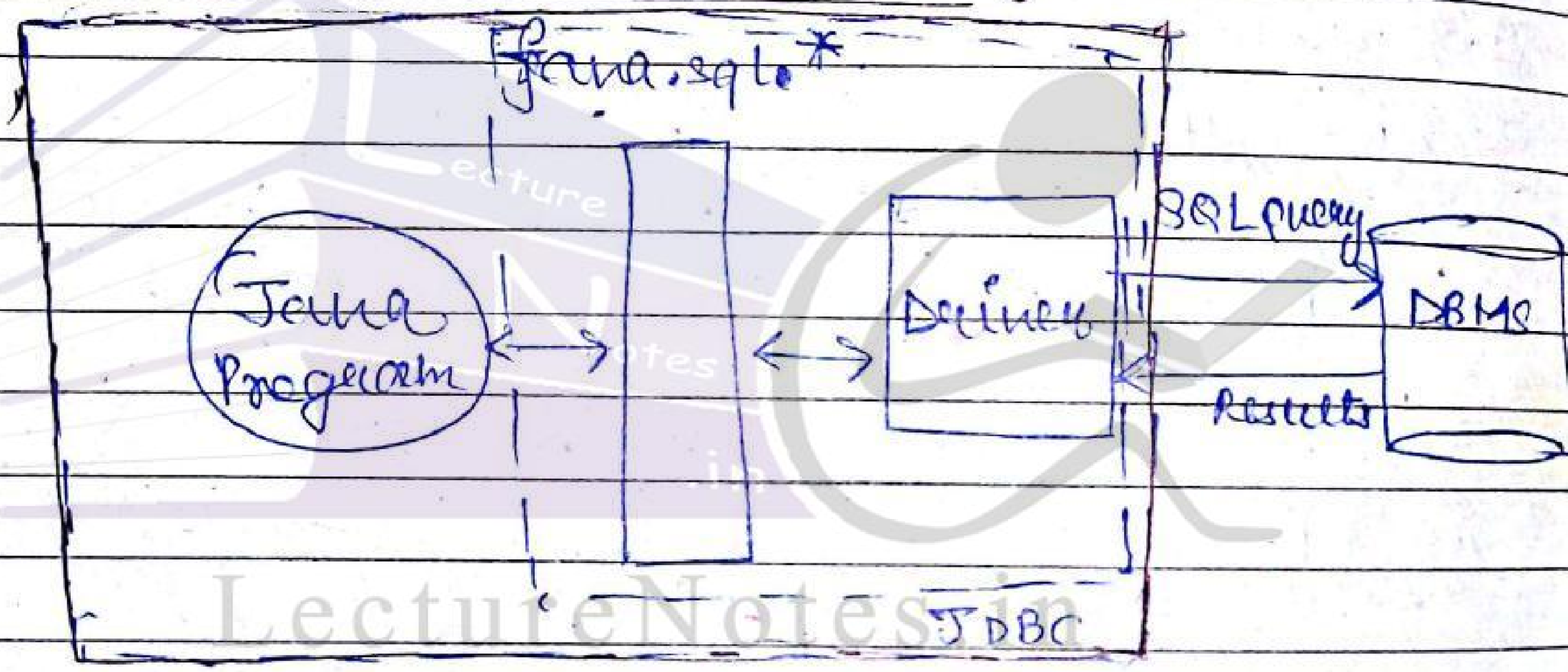
4. JDBC is to Java programs and ODBC is to programs written in other languages than Java.

5. ODBC is used between applications and JDBC is used by Java programmers to connect to database.

Details about JDBC :

→ JDBC API is in the package
java.sql.
→ 8 interfaces
6 classes
3 exceptions. In JDK 1.1

JDBC driver model :



JDBC Drivers Types :

There are 4 types of JDBC drivers.

- JDBC Type 1 Drivers : They are JDBC-ODBC bridge drivers i.e. Translate JDBC into ODBC and send. They delegate the work of data access to ODBC API.

JDBC-native Bridge

- JDBC Type-2 Drivers They mainly use native API for data access i.e; Connects JDBC to data base vendors native SQL calls. ~~and provide Java wrapper classes~~

JDBC-net Bridge

- JDBC Type-3 Drivers Translates JDBC to a DBMS independent network protocol.
  → Uses vendor independent Net-protocol to to access a vender independent call listener. The listen server maps the vendor independent calls to the vendor dependent ones. This extra step adds complexity and decreases the data access efficiency

Direct JDBC drivers

- JDBC type-4 Drivers Most efficient among all driver types.
  → directly translates JDBC to native API used by RDMS.
  → doesn't require anything to be installed on client's machine.

JDBC classes (6 classes)

Ex: (1) DriverManager
→ used to obtain connections to the database.

(2) Types
→ Defines constants which identify SQL types.

(3) Date
→ used to map between java.util.Date to SQL DATE type

(4) Time
→ used to map between java.util.Time to SQL TIME type

(5) TimeStamp
→ used to map between java.util.Timestamp to SQL TIMESTAMP type

Date / /
Page No.

JDBC Interfaces (6 Interfaces)

(1). Drivers
→ All JDBC drivers must implement the driver interface.
→ used to obtain the connections to the specific database type.

(2). Connection
→ Represents the connection to a specific database.
→ used for creating statements
→ used for calling stored procedures
→ used to create callable statements.

(3). Statement
→ used do execute SQL statements against the database.

(4). Result set
→ represents the result of an SQL Statement.
→ provides methods for navigating through the resulting data.

Date / /
Page No.

(5). prepared statement.
→ Similar to stored procedures.
* An SQL statement is compiled & stored in a database.

(6). Callable statements
→ used for executing stored procedures

(7). Database metadata
→ provide access to database system catalouge

(8). Resultset meta data.
→ provide information about the data contained by the result set.

Date / /
Page No

Advantages

→ Better performance than all other drivers.

→ NO software is required at client side or server side.

Basic steps to use a database in Java

① Establish a connection.

- load / register the driver class using forName() method.

eg: Class.forName("oracle.jdbc.driver.OracleDriver");

- Establish connection with getConnection() method.

eg: Connection con = DriverManager.getConnection
("jdbc:oracle:thin:@localhost:8152:XE", "System", "Password");

↓ class name

↓ string URL

↓ string name

↓ string password

Date / /
Page No.

(2). Create the ~~object~~ statement Object :-

→ Create statement() method is used,
→ responsible to execute queries with the database.

eg; Statement stmt = con.createStatement();

(3). Execute the query & get result set

→ executeQuery() method is used.

eg;

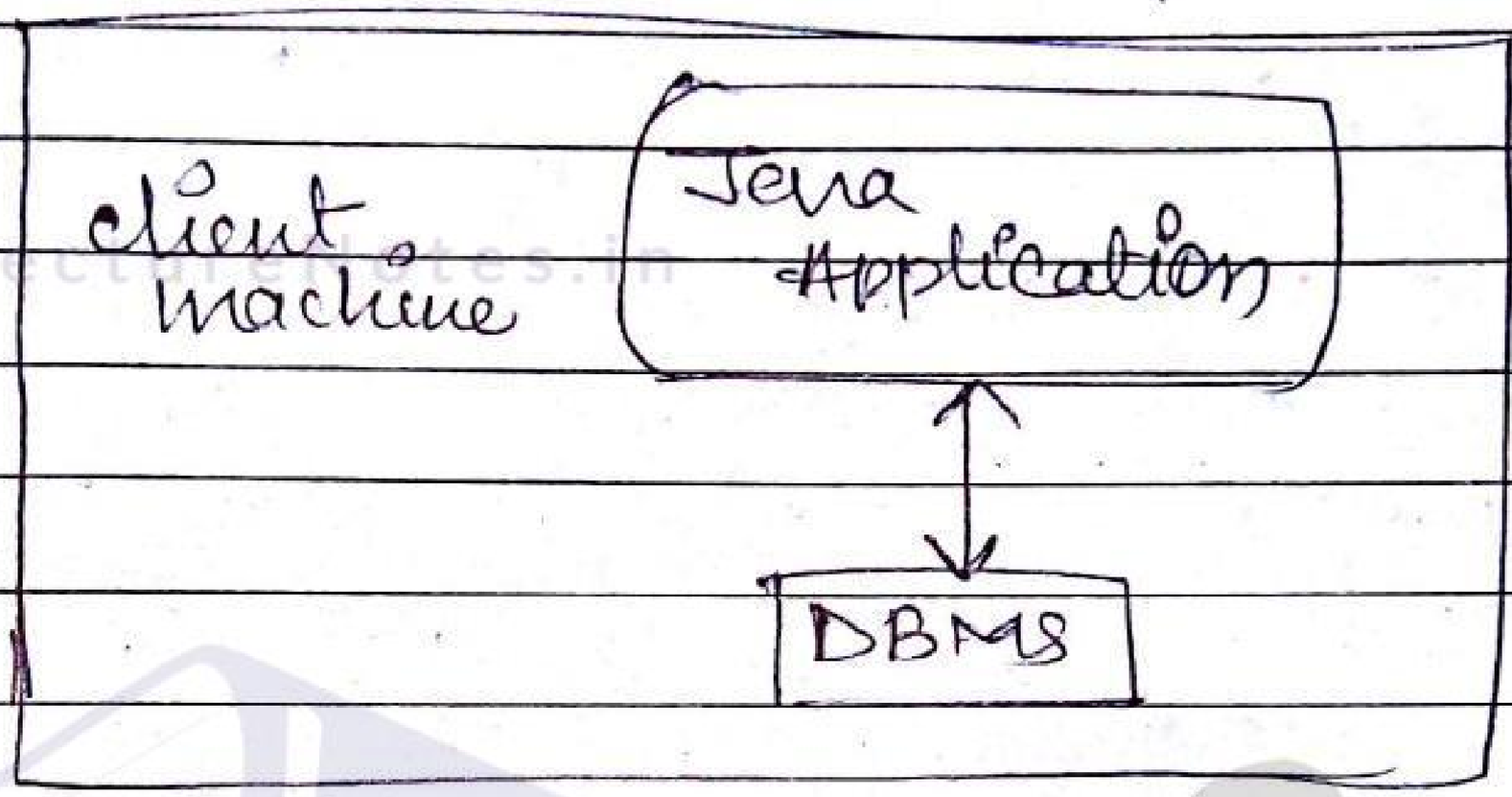
ResultSet rs = stmt.executeQuery("select * from emp");

(4) Close Connection :

stmt.close();
con.close();

Date / /
Page No.

11. Two-tier Architecture for Data Access



→ In a two-tier model, a java application talks directly to the data source.

→ This requires a JDBC driver that can communicate with the particular data source being accessed.

→ User's commands are send to the database & then those results are send back to the user.

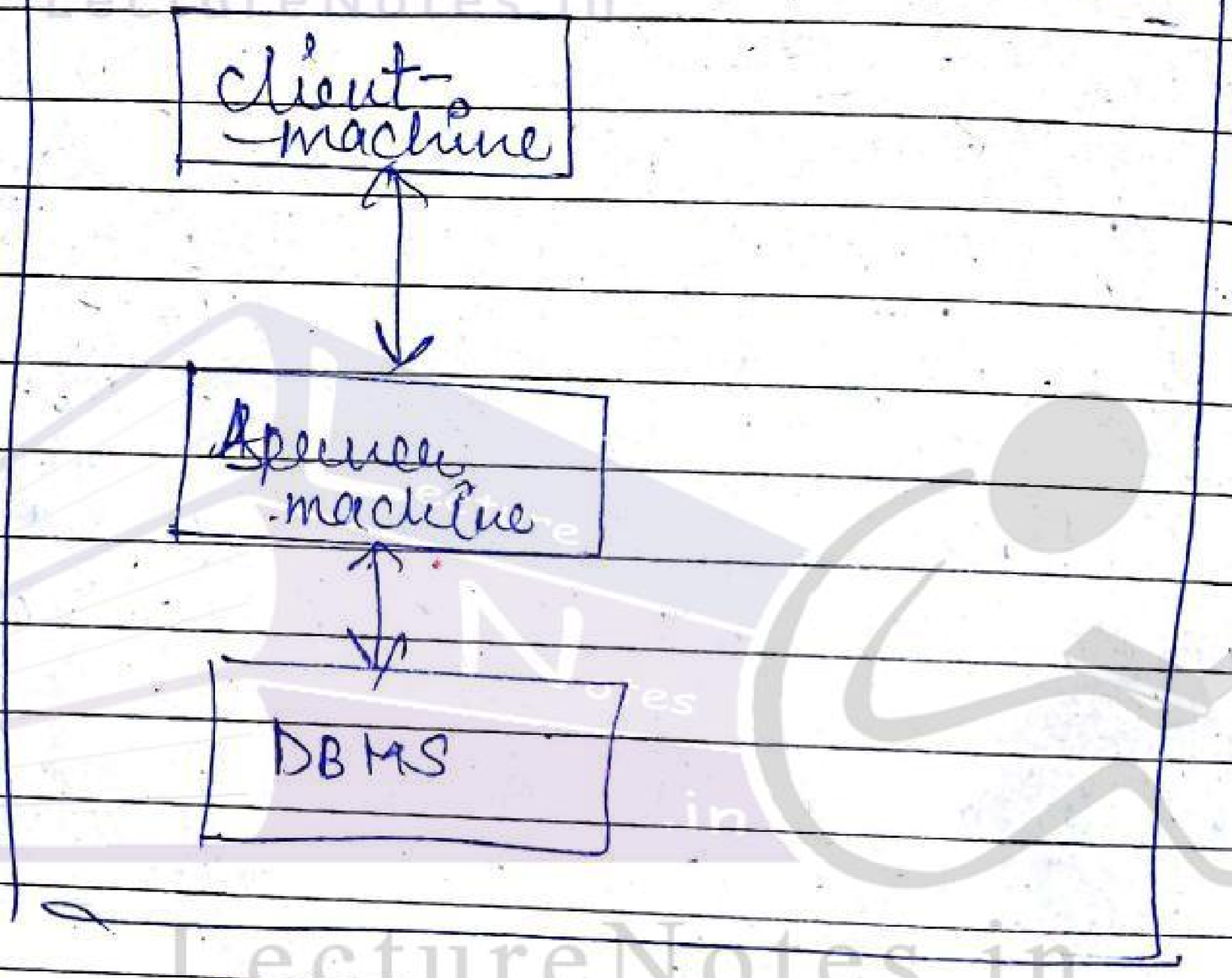
→ The data source may be located on another machine to which the user is connected via a network.

This is referred to as a client/server configuration with the user's machine as a client & the machine

Date / /
Page No.

the data source as server.

Three-tier architecture for Data Access.



→ In three-tier model, Commands are send to the middle-tier which then sends the commands to data source. The data source processes the command, these commands are then sent to the middle-tier & then it sends them to the User.

→ This model is very attractive because the middle tier makes it possible to control access, maintain

Date / /
Page No.

→ provides performance advantages.

Database Connectivity with Oracle

For connecting java application with the oracle database, you need to follow five steps for database connectivity;

(1). Driver class:
→ 'oracle.jdbc.OracleDriver'

(2). Connection URL:
~~jdbc:host~~
jdbc:oracle:thin@localhost:1521:xe

where,
jdbc → API
oracle → database
thin → driver
localhost → servername
1521 → Port number
xe → oracle service name

(3). Username & System
(4). password → given by User at the time of installation.

Example:

```
import java.sql.*;
class OracleCon{
public static void main(String args[])
{
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection
("jdbc:oracle:thin@localhost:1521:xe",
"system","password");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+" "+
rs.getString(2)+" "+
rs.getString(3));
stmt.close();
con.close();
}catch(exception e){System.out.println(e);}
}}
```

Date / /
Page No.

Database Connectivity with MySQL

→ Three steps are to be performed following to perform MySQL connectivity.

1. Driver class :- The driver class for mysql database is com.mysql.jdbc.Driver.

2. Connection URL :
jdbc:mysql://localhost:3306/sonoo

where,
jdbc → API
mysql → database
localhost → ip address on which mysql is running.
3306 → port number
sonoo → database name.

3. Username → default user name for mysql is "root".

4. password → given by the user at the time of installation

Date / /
Page No.

(8). Create a table in mysql database, firstly create database

```
Create database sunny;
Use sunny;
Create table emp( id int(10), name varchar
                                (15), age int(3));
```

Example:

```
import java.sql.*;
class MySQLCon{
public static void main(String args[]){
try{
Class.forname("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection
("jdbc:MySQL://localhost:3306/sunny",
"root","root");
Statement stmt = con.createStatement();
```

Date / /
Page No.

```
ResultSet rs = stmt.executeQuery("select * from emp");
while (rs.next()){
System.out.println(" rs.getInt(1) + "  "+
                   " rs.getString(2) + "  "
                   " rs.getString(3) ");
stmt.close();
con.close();
}
catch(Exception e)
{ System.out.println(e);
}
}
```

Date / /
Page No.

Database connectivity with MS-access 6-

There are two ways to connect java application with access database
(1) without DSN
(2). with DSN.

(1). Without DSN.

→ We have created login table in the access database.

Ex6
```
import java.sql.*;
class Test{
public static void main(String args[])
{
try{
String database="Student.mdb";
String url="jdbc:odbc:Driver={Microsoft Access Driver(*.mdb)};
DBQ="+database+";DriverID=22;READONLY=true";
```

Class.forname("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection(URL);
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("select * from emp");

while(rs.next())
System.out.println(rs.getInt(1));

catch(exception e)
System.out.println(e);

② With DSN

→ assume your dsn name is mydsn.

import java.sql.*;
class Test5
public static void main(String args[])
try

String url = "jdbc.odbc.mydsn";

class.forname("sun.jdbc.odbc.jdbc odbc driver");

Connection con = DriverManager.getconnection(url);

Statement stmt = con.Create statement();

ResultSet rs = stmt.executeQuery("select * from login");

while (rs.next())

System.out.println(rs.getString(1));

}

catch(exception e)

System.out.println(e);

}

}

Date / /
Page No.

Networking

→ The term network programming refers to writing programs that execute across multiple computers (systems), in which all devices are connected to each other using a network.

The java.net package provides support for the common network protocols

0. TCP : TCP stands for transmission control protocol, which allows for reliable communication between two applications.
TCP is typically used over the Internet protocol, which is referred to as TCP/IP.

(2). UDP : stands for User datagram protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

Date / / .
Page No.

Socket Programming

→ Java socket programming is used for communication between the applications running on different JRE.

→ Java Socket programming can be connection-oriented or connected-less.

→ Socket classes are used to represent the connection between client program and a server program.

```
ocket() {
}
```

Socket and ServerSocket ⇒ classes are used for connection-oriented socket programming.

DatagramSocket and DatagramPacket classes are used for connection less SOCKET programming.

Socket → at client side of the connection
ServerSocket → at server side of the connection.

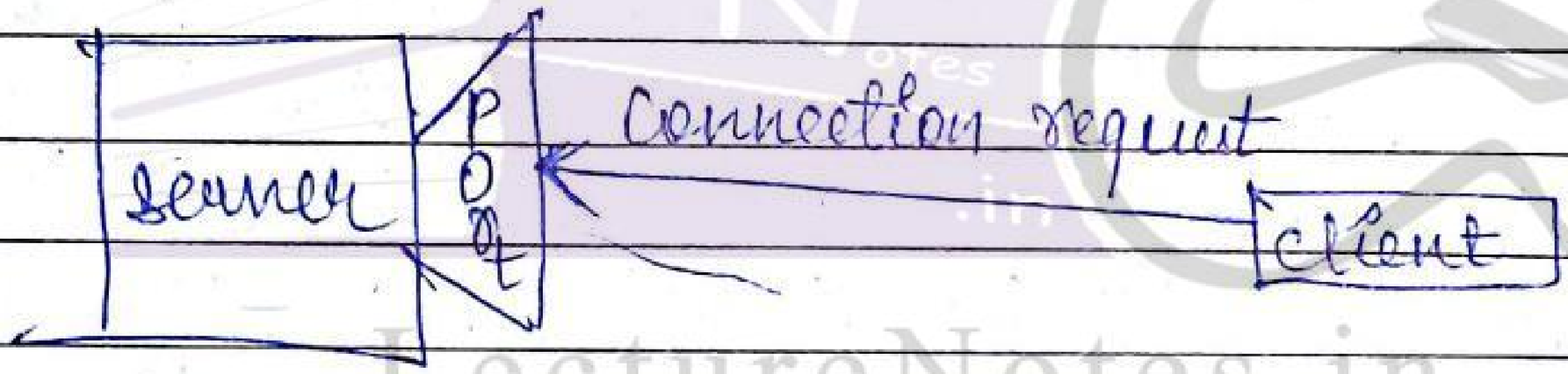
Date / /
Page No.

Socket is simply an end-point for communications between the machines.

Socket Communication :

A server runs on a specific computer and has a socket that is bound to a specific port.

The server waits and listens to the socket for a client to make a connection request.

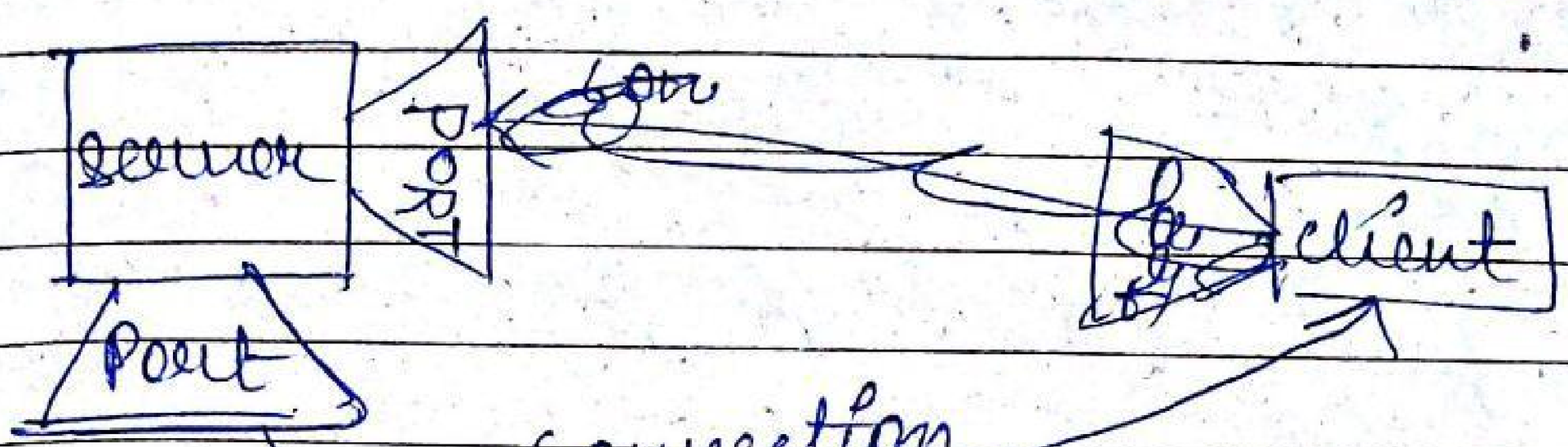


If everything goes well, the server accepts the connection.

Upon acceptance, the server gets a new socket bound to a different port with different port number.

So that it can be continue to listen to the original socket for connection requests while serving the connected client.

Date / /
Page No.



Important methods of

① socket class

public InputStream getInputStream()
→ returns the InputStream attached with this socket.

public OutputStream getOutputStream()
→ returns the OutputStream attached with the socket.

public synchronized void close()
→ closes this socket.

② ServerSocket class

public Socket accept()
→ establishes connection between client & server.

Date / /
Page No.

Ex. of Java socket programming -

Read fruits both side.

```
import java.net.*;
import java.io.*;
class MyServer{
public static void main(String args[])
                throws Exception{
ServerSocket ss = new ServerSocket
                (3333);
Socket s = ss.accept();
DataInputStream dis = new DataInputStream
                (s.getInputStream());
DataOutputStream dos = new DataOutputStream
                (s.getOutputStream());
BufferedReader br = new BufferedReader
                (new InputStreamReader
                (System.in));
```