Page No.

Java

Unit-I

Overview

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

→ Invented by the team called 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 progeramming.

→ 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-consuming 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 Java 6

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

Compile 6 Javac Simple.Java
Run 6 Java Simple.

(Main is the first method that is seen).

Date :
Page No.

Use of Java (Application)

(1). 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 &

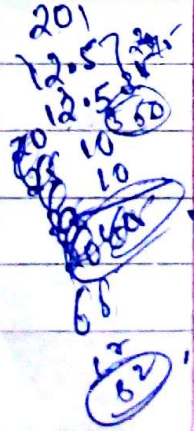
(1). Stand - alone application &

→ 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/load 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 Application :

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

→ Such as whatsap etc

→ Android. J2ME used to create mobile applications.

Date :
Page No.

Characteristics of Java

1. Java is Simple
2. Object-oriented
3. distributed
4. Architectural-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 clumpsy 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 seen on screen side.

Java's High Performance.
Java has high performance as it is architecture 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.

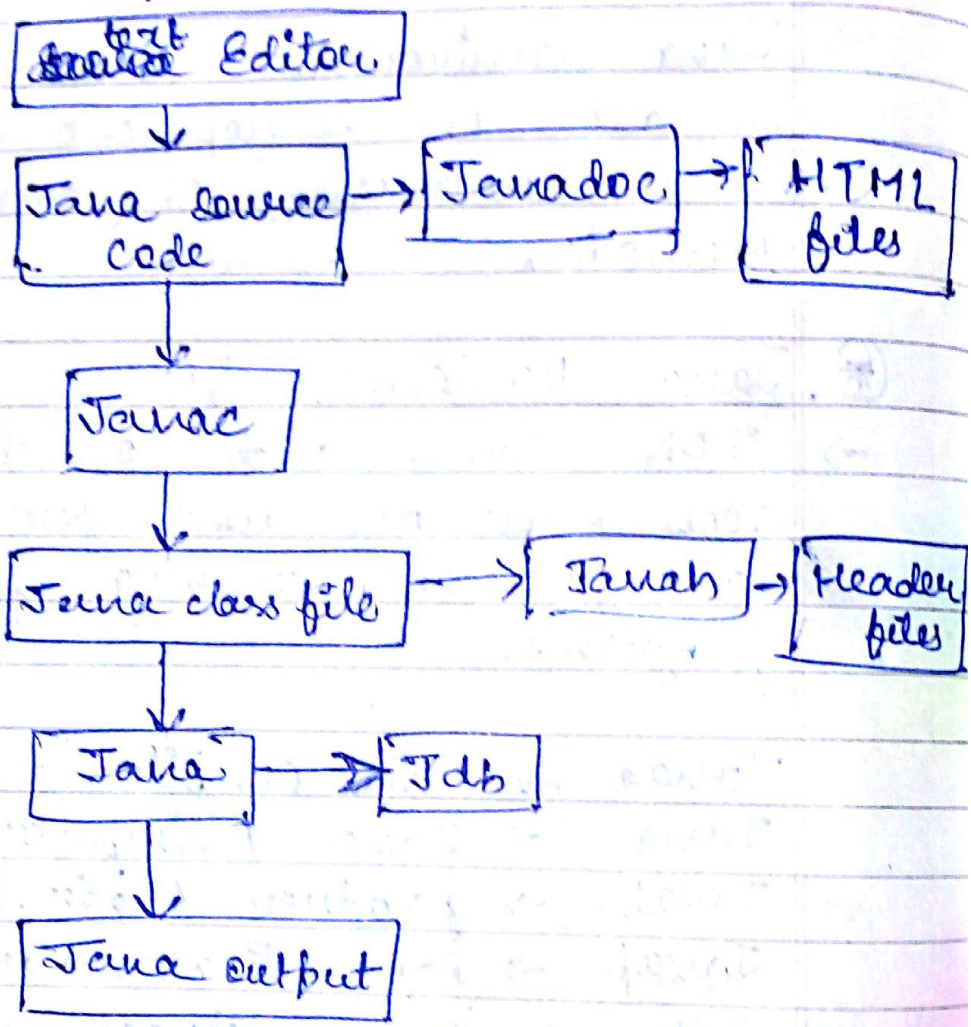
(*). Java Development Kit

→ 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.

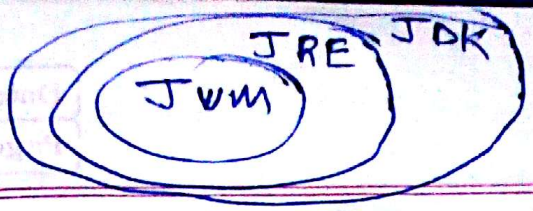
Flow of Java program.



Explain Each - One ——

JDK, JRE and JVM

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

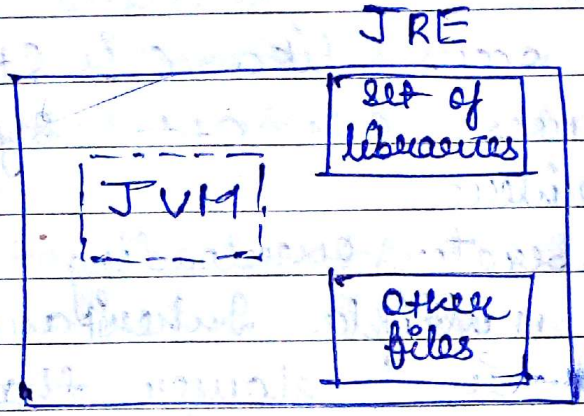


Date :
Page No.

JRE

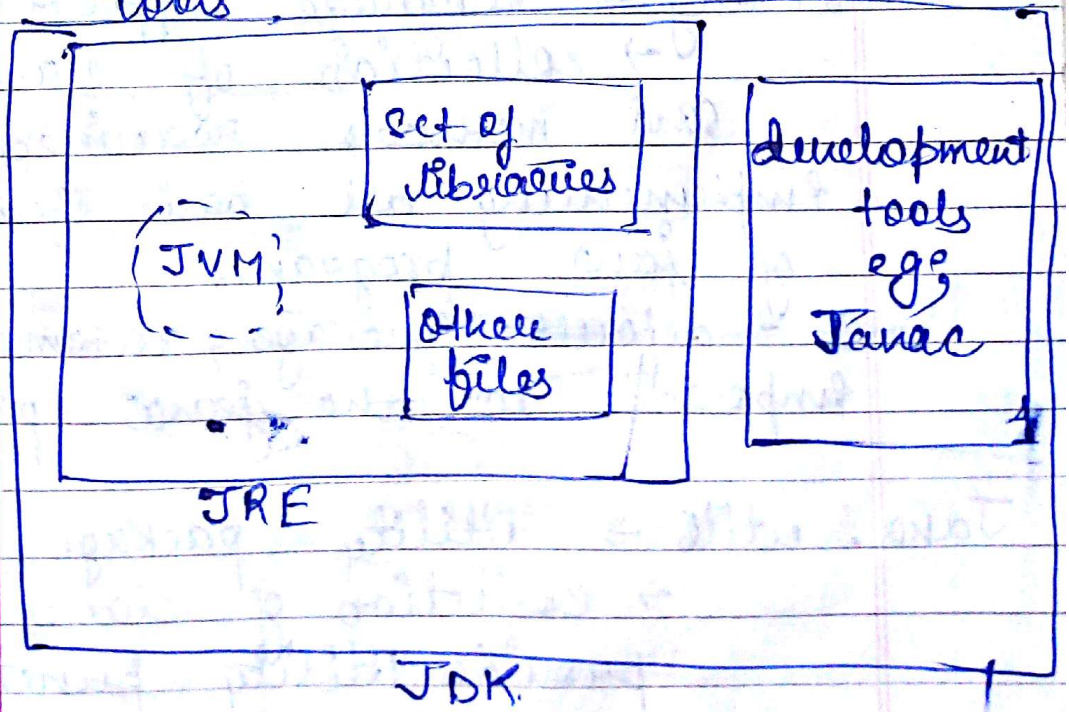
Java Run-time Environment.
used to provide run-time environment.

→ It is an implementation of JVM.



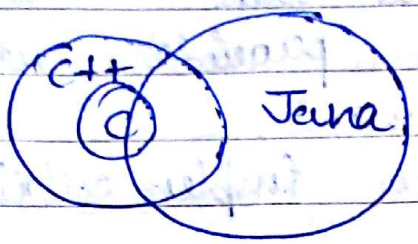
JDK → development tool Kit.

→ 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.util → Utility package.
→ collection of classes to provide utility functions such as date & time functions

Date :
Page No.

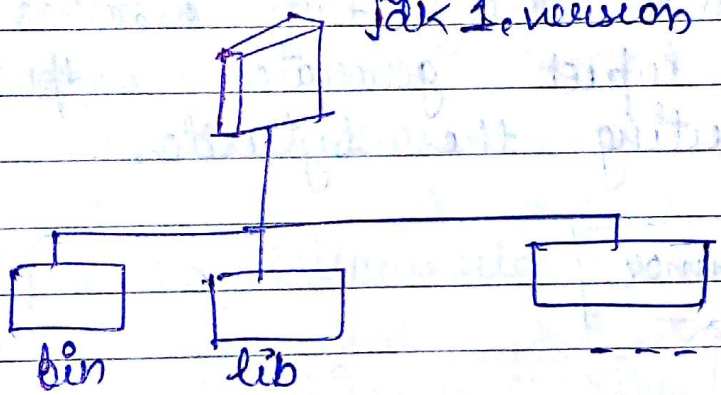
Java.io :

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

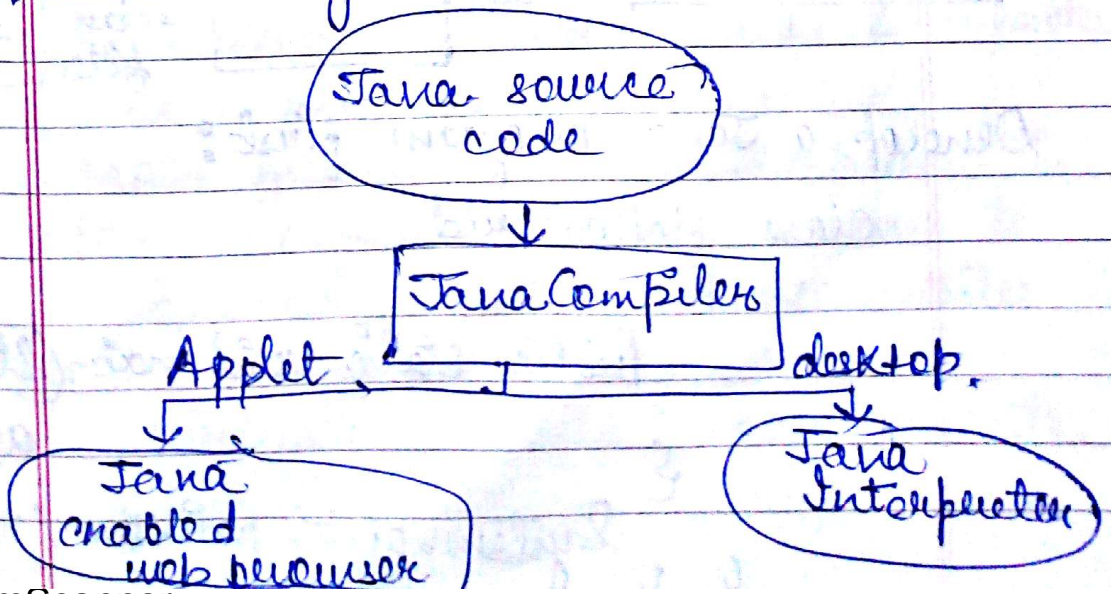
Java.net → Networking package.
→ Collection of classes required for communicating with other computers via internet.

AWT package = Java.Awt

#. JDK directory structure :



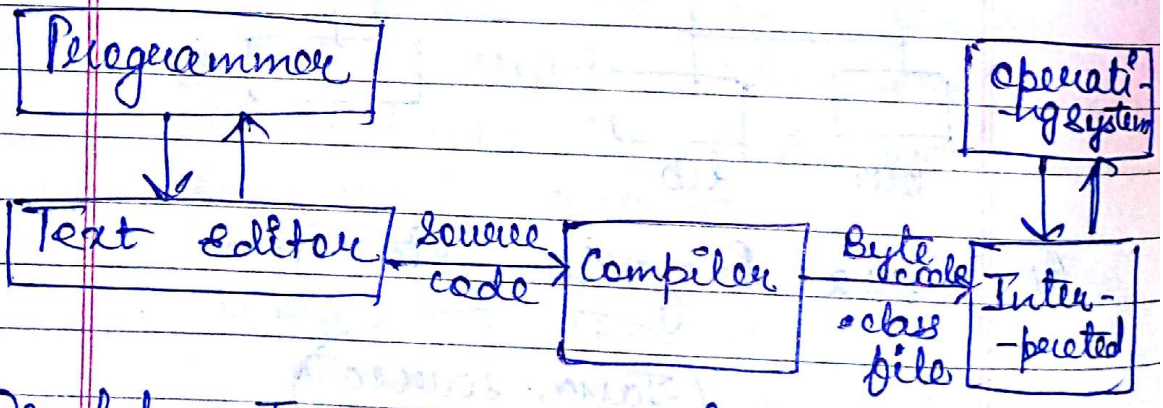
#. Java Program Structure :



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 is 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 execute.

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
→ finding 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 ~~affecting 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 methods ~~members~~ can be called without objects.

General structure

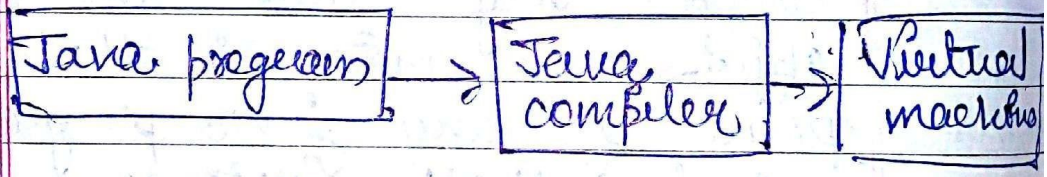
General str. of Java Program

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

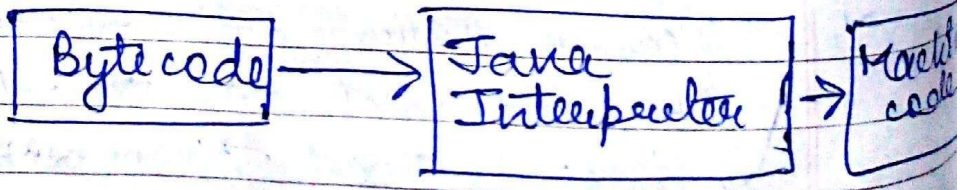
Date :
Page No.

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 reffered to as virtual machine code.

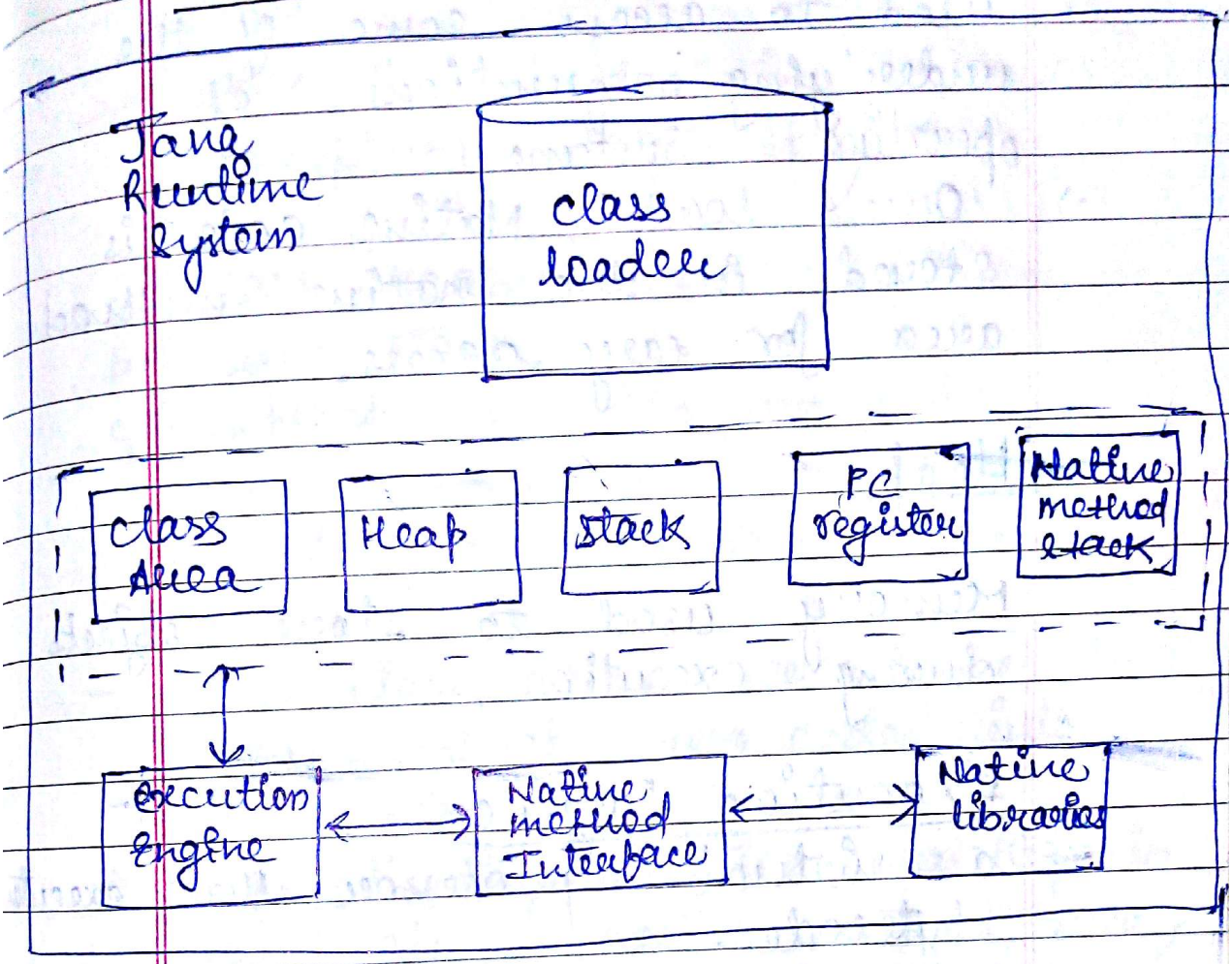


Virtual machine code is not machine specific code.
The machine specific code is generated by the java interpreter by acting as the intermediatory between the virtual machine and real machine.



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 system.

→ 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 Areas

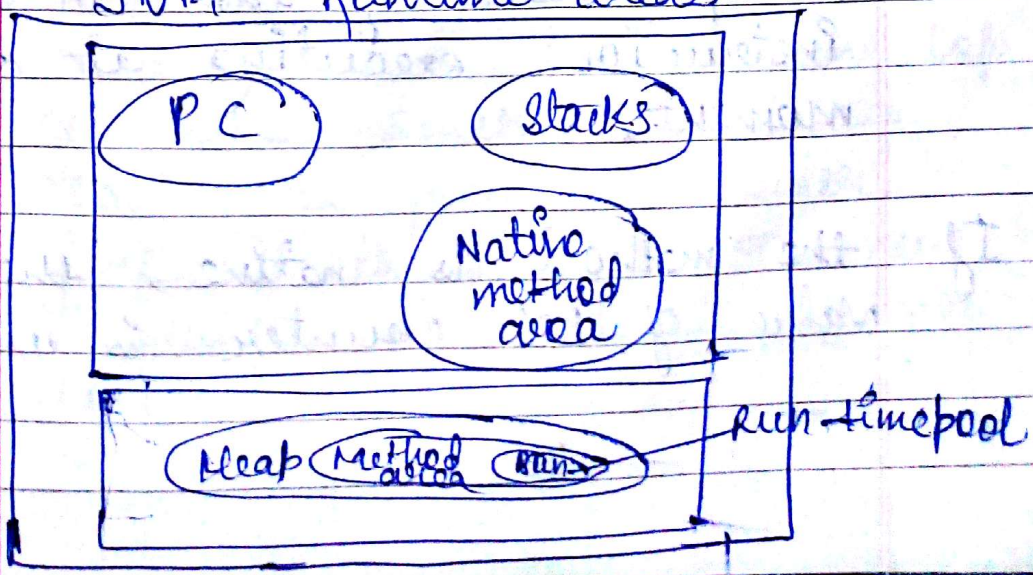
- The class Area stores the classes that are loaded into the system.

→ 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~~

JVM Runtime areas



Date :
Page No.

Classes have several 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 of 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 of saved as part of older Java stack frame. The new frame has its own new PC.

Date :
Page No.

Heap

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

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

Out of Memory Errors: 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.

Heaps

- 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.

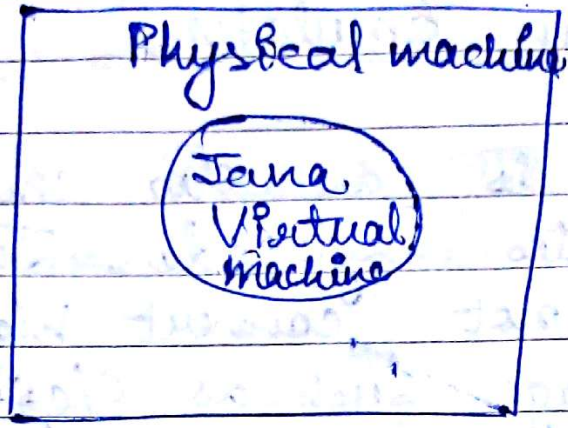
JVM as an Emulator

JVM is an 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 — same

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 intermediatory 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 arguments 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 ex : jsr instruction
(jump to sub-routine)

(4). Field access :
→ In Jana, 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 invocations

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.

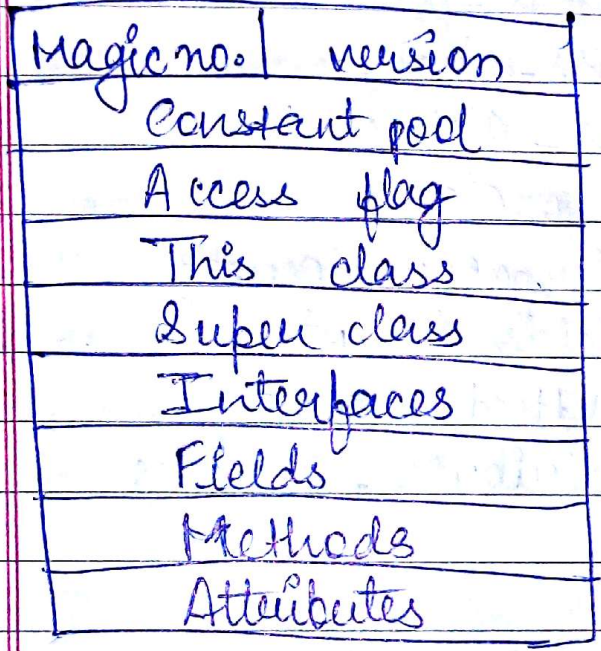
6. Conversions

- 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.

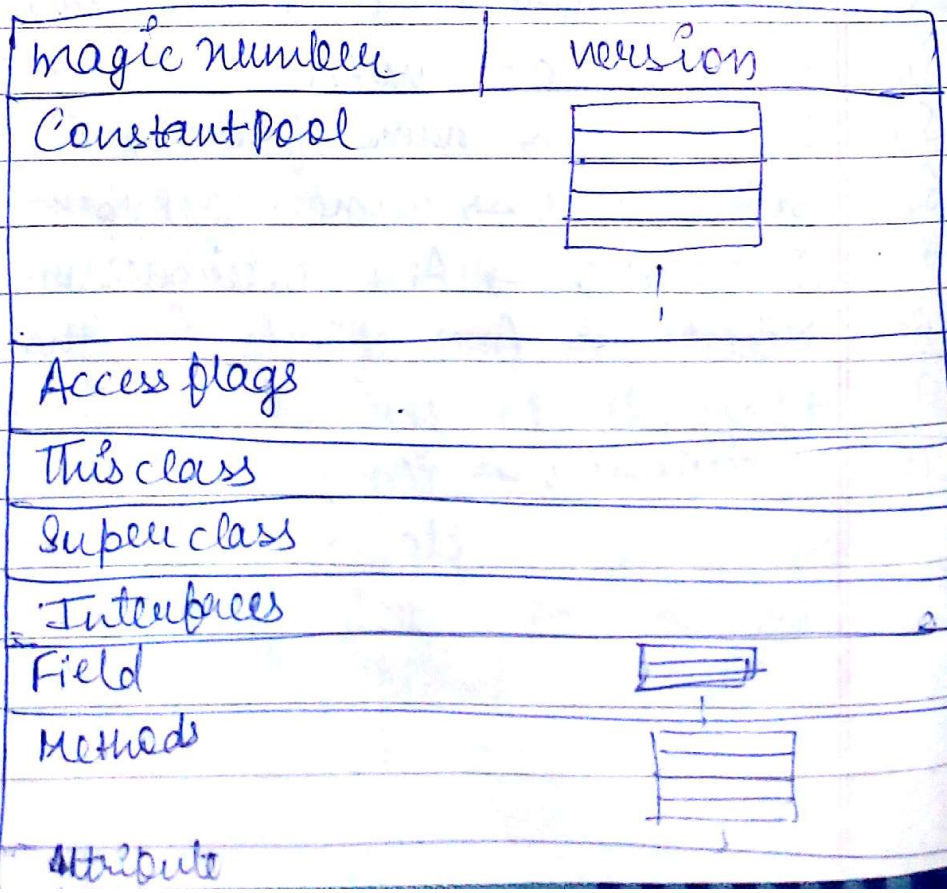


1. magic number → 0x CAFEBABE.
2. Version → minor and major versions of class file.
3. Constant pool → pools of constants for the class.
4. Access flag → wether the class is abstract or static
5. This class → name of current class.
6. Super class → name of super class
7. Interfaces → Any Interfaces in 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 it is loaded, before its instances are created. This ensures the JVM implementation to assume that classes have certain safety properties.

Date :
Page No.

→ The verification algorithm makes it possible to safely download 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 execution.

Date :
Page No.

for example,

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

```
                    Initialize
ldc 70 ;  // loads i
istore_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 avaible 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 ~~calling~~ these request can call the garbage collection process.

~~How~~ Java ~~work~~

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 gener.
  - permanent gener.

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).

Similarly, Garbage collector 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 ~~choosed~~ 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 in future for the instances.
This memory spaces are as a fragmented across the memory area.

Finalisation 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 user's computing environment (Trojan horses, virus)
- programs should be prevented from discovering 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 local variable is initialized, before it is used.
7. It is impossible to underflow or overflow the stack.
8. An attempt to use a null reference as a reciever of method invocation always throws 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 unprevileldge code and only available to the previleldged code.

Most code are unprevileldged; Security architecture is responsible for ensuring that unprevileldge

Date :
Page No.

code doesnot masquerade the priviledge 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.

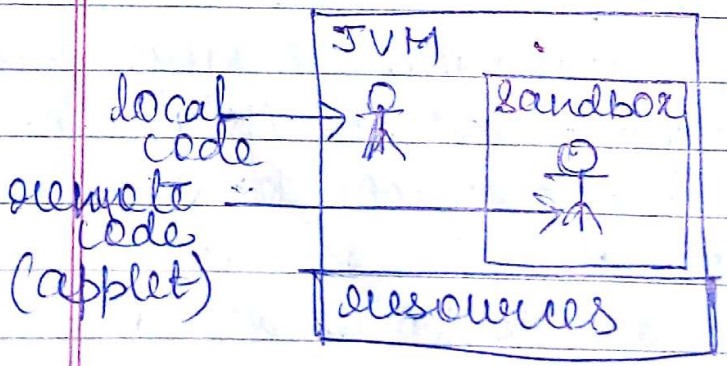
- This method uses permission object as an input and returns a 'yes' or 'NO' (based on the source code & permissions)

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~~ replace the default security manager by their own security manager if they have permission to do so.

SANDBOX Security Model :

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



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

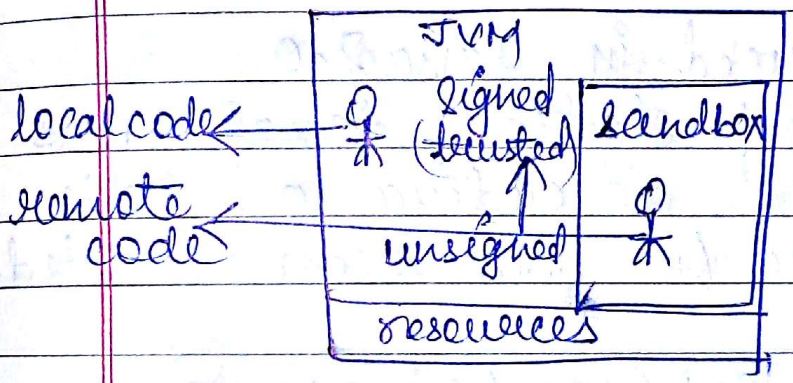
Date :
Page No.

restricted access to system resources.

→ remote code (applet) is limited to sandbox.

The Concept of trusted code

→ introduced in java 1.1



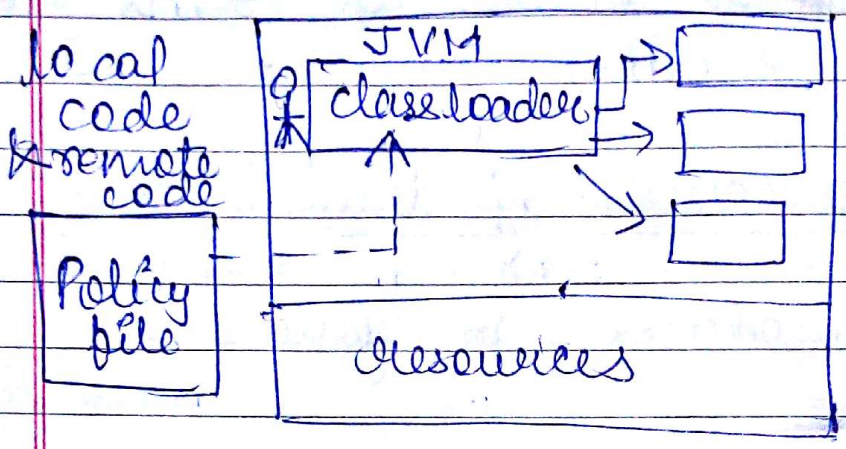
→ Based on the idea, that local code has unrestricted access to the system resources.

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

→ limited 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, & avoid to create object we make main as static

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

#. General structure of a Java program.

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

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

In Java, we have 8 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: for 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, d;
        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
    length = length + b.length.
        return this;
}
```

Jvm looks up for
1. name of the method
2. no. of arguments.
3. Type of arguments.

private protected → accessible to each & every sub-class

All Instance of this 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 ... oper-ator

Date :
Page No.

Example :

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

9990836998
Poonam.

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 +
```

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" + "greater");
}
if (b>a)
{
System.out.print("b" + "is" + "greater");
}
}
}
}
```

Date :
Page No.

Even/odd

```
import java.util.*;
class Compare1
  {
  public static void main(String arg[]
    {
    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");
  }
  }
  }
  }
```

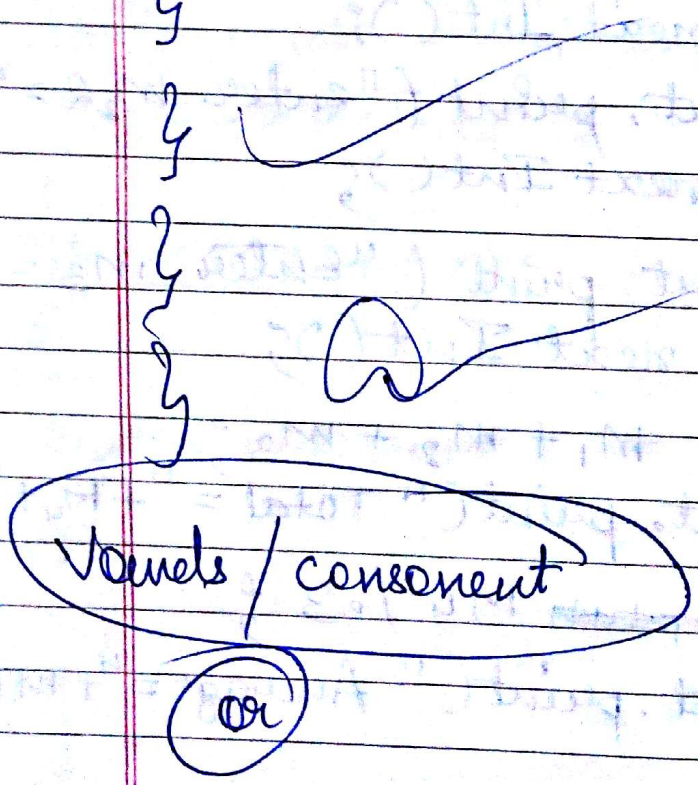
Date :
Page No.

11.

```
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 + "avg");
  m = M4 / 3;
System.out.print("Average = " + m + "avg");
{
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");
}
}
}
}
}
```



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

Q) Review

vowel / consonent,

Date :
Page No.

#.
```
import java.util.*;
class Letters
{
  public static void main(String
                          arg[])
{
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("Consonent");
 }
}
}
}
```

Date :
Page No.

Import java.util.*;

I/P 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 overrride 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.

Anonymous Inner class

class → class ie, not assigned any name

Example:

```
class Anony_test extends Frame
{
    Frame f;
    Button b;
    TextField tf;
    Anony_test()
    {
        f = new Frame();
        f.setVisible(true);
        f.setLayout(new FlowLayout());
        f.setSize(300,300);
        tf = new TextField(15);
        b = new Button("click me");
        f.add(b);
        f.add(tf);
```

(Frame)

Date / /
Page No.

b. AddActionListener(new ActionListener()

{

public void ActionPerformed(ActionEvent e)

{

tf.setText("welcome");

}

};

});

Date / /
Page No.

Multithread Inheritance & Interfaces

Subclass are specialized version of super class.

Benefits of Java's 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 name)
{
    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 setdept(int no, String name)
{
    dno = no;
    dname = name;
}
public void putdept()
{
System.out.println("deptno = " + dno);
                  ("deptname = " + dname);
}

Public static void main(String args[])
{
    department d = new department();
    d.setemp(100, "aaa");
    d.setdept(20, "sales");
    d.putemp();
    d.putdept();
}
```

2. multilevel

A

↓

B

↓

C

Ex e

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

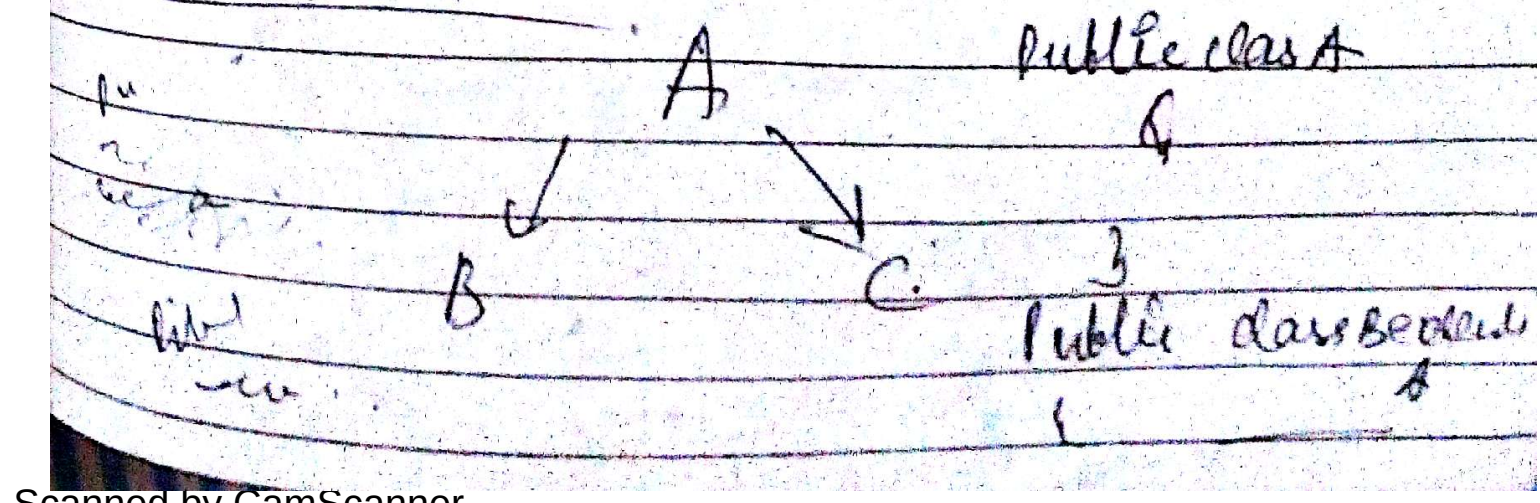
Date / /
Page No.

```
}
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("mark2 = " + marks2);
}
}
Class Finaltot 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[]);
{
    FinalTot f = new FinalTot();
    f.setStud(70, "Sunny");
    f.setMarks(78,89);
    f.calc();
    f.putStud();
    f.putMarks();
    f.putTotal();
}
}
```

Himanchal



Date / /
Page No.

Construction In Inheritance

```
class Parent {

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

class child extends Parent {
    child() {
        S.O.P ("S2");
    }

}

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

11. Using "super" keyword :

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

Uses of super keywords are as follows :

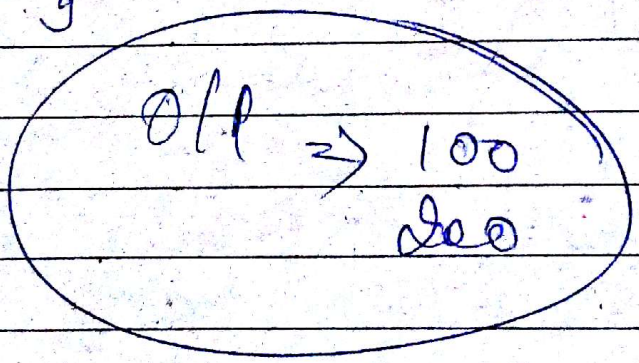
1. super() is used to invoke immediate parent class constructor
2. super is used to invoke immediate parent class method
3. ~~super is used to refer immediate parent class variable.~~

Example :

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

Date / /
Page No.

System.out.println (super.a);
" " " (a);
}
p.s.v.m (String [] args)
{
Sup2 obj = new Sup2();
obj.show();
}
}



(#) ; 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 B

```
class A
{
int i, 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;
```

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

O/P ⇒ K : 3

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 b/w 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 →

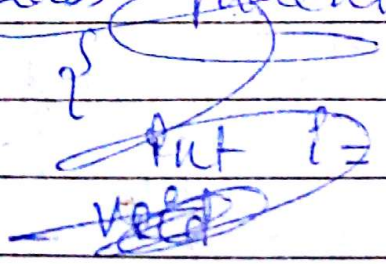
Dynamic method dispatch is an mechanism by which a call to an overridden method is resolved at run time, rather than the compile time.

Dynamic method dispatch is important 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



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();
```

c2.meth2();

}

}

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 behaviour 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

```
interface Interface_emp
{
    public void salary();
    int increment = 10;
}
```

Implementing Interface

```
class Interfacetest implements emp
{
    public void salary()
    {
        S.O.P("salary of emp is " + (increment * 5000));
    }
    public static void main(String args[])
    {
        Interfacetest I1 = new Interfacetest();
        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 = 1000000;
}

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

  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 6

```
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 inherited.
   final class finalmethod

Date / /
Page No.

Exception Handling

Exception → problem arises during execution.

reasons → Invalid I/p
→ file to be opened not found
→ lost of network connection

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

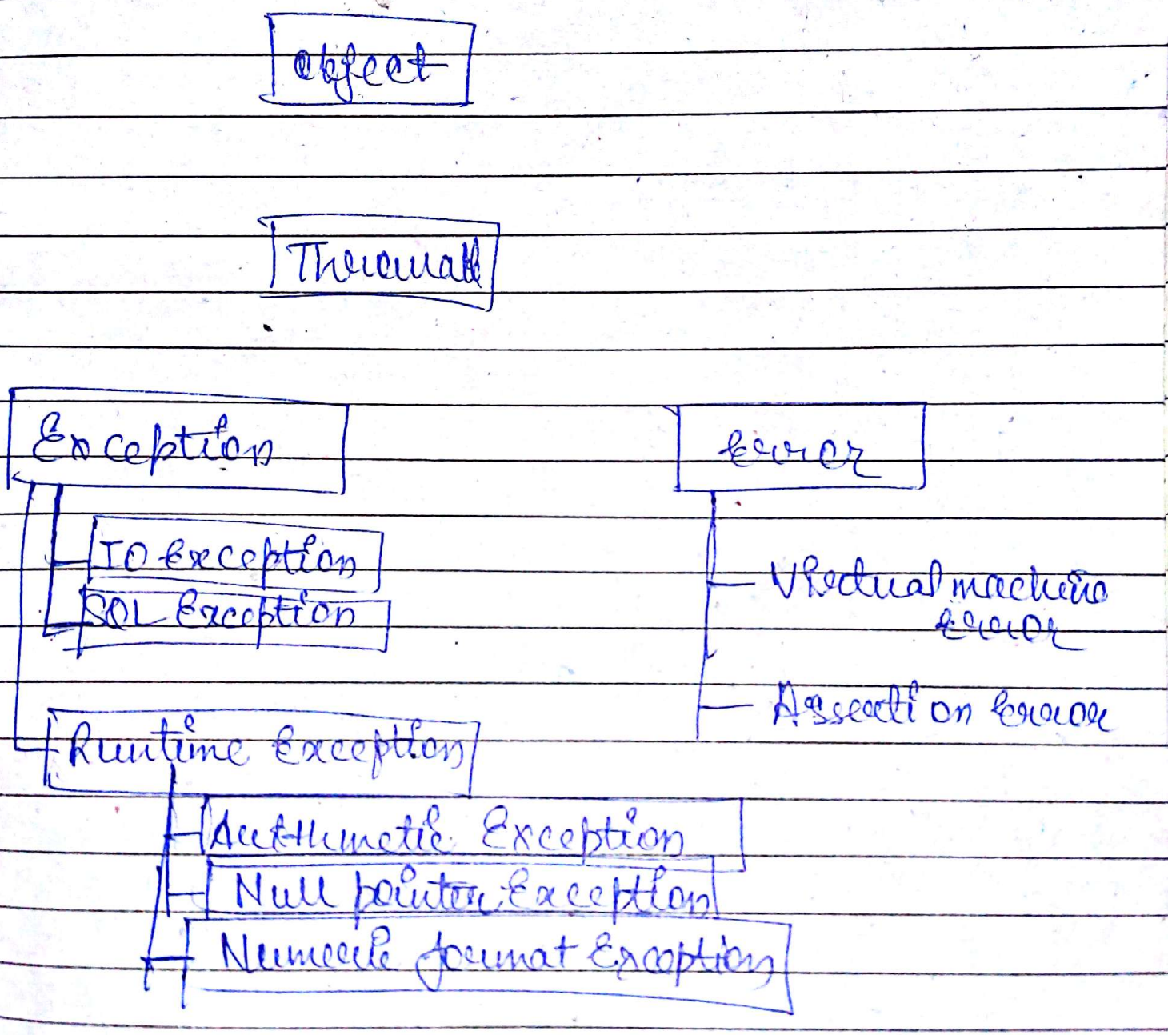
Types
Checked Exception
↳ IO exception, SQL Exception

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

Error → not exceptional at all

Date / /
Page No.

Hierarchy of Exceptions



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.

Syntax

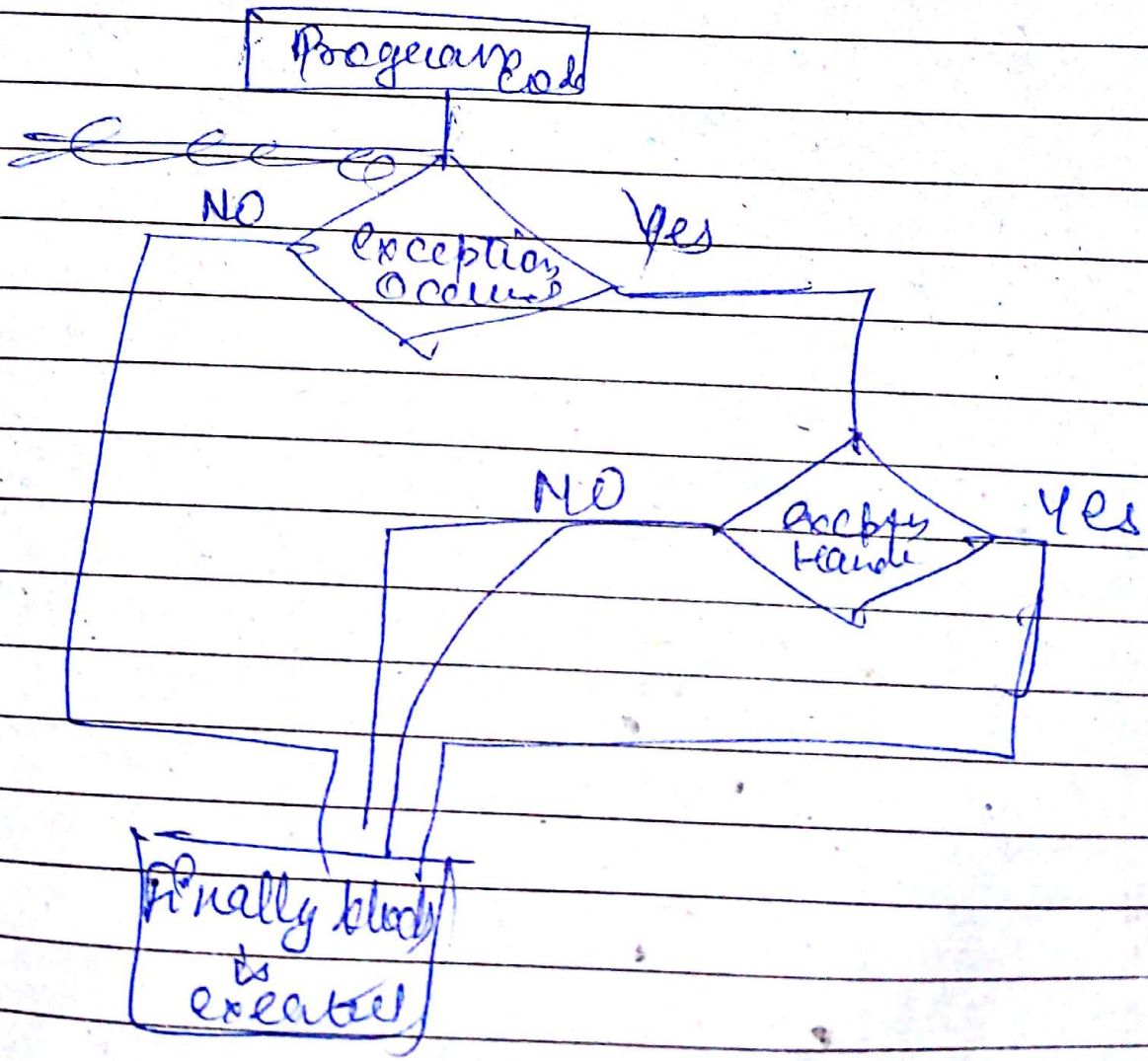
```
try
{
    // protected code
}
catch (exception e)
{
    // catch block.
}
```

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.



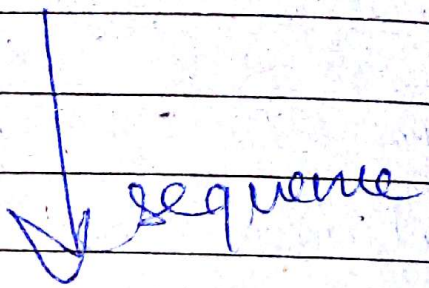
Date / /
Page No.

try
{

}
catch
{

}
finally
{

}



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 web application that cannot be provided by HTML alone.

→ A java applet extends the java.applet.Applet

→ The class must override the methods from 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 / /
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.

4. Java Applet Advantages :

1. It is simple to make it work on any platform.

2. It is fast - Can 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 specific 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.
We cannot use HTML tags in application.	We can use HTML tags in Applet.
For execution of application there is no need for web-browsers	For executing applet web-browsers is required.
To execute application java interpreter is needed	To execute applet java-web enabled browsers is needed.

In application program execution begins with main() method.	In applet, execution does not begins with main() method.
No necessarily to use inheritance in a basic program	applet is inherited from Applet.

End of Applet

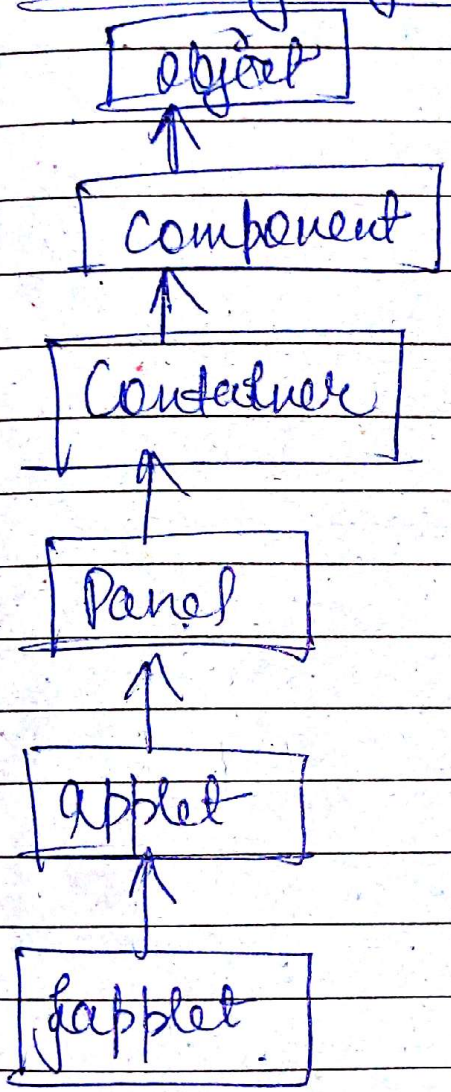
```
Import java.awt.*;
Import java.applet.Applet;
public class First extends Applet{
```

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. Name Myapplet.java
2. javac Myapplet.java
3. Name Myapplet.class
4. Create Myapplet.HTML.

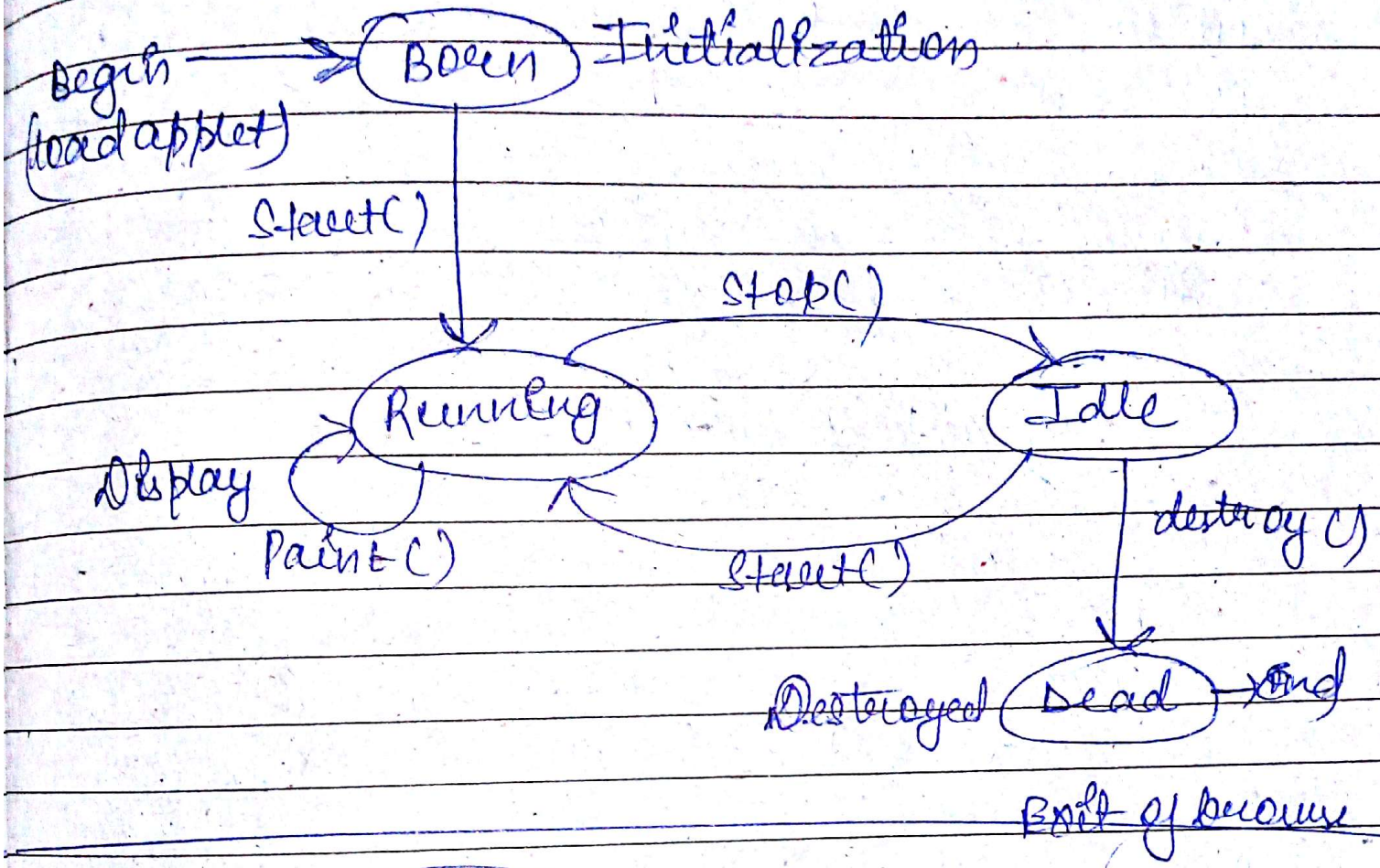
<applet code="Myapplet.class" width=200 height=300> </applet>

→ Applets do not have main. Instead they have:

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

Date / /
Page No.

Applet life-cycle



- init() → init() and destroy() are only called once
- start() and stop() are called whenever a browser enters or leaves the page
- do some work is code called by your listeners.

Date / /
Page No.

Applet methods :

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

(1). Why an applet works ?

→ 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.

1. 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

2. 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.

3. stop() method

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

4. 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.

(iv). Painting the applet :-

(i). 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.

(ii) 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.

(iii). 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 the~~ refreshes 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(
             10,65);
}
}
```

Thread || Applet

Date :
Page No.

10. Threads.

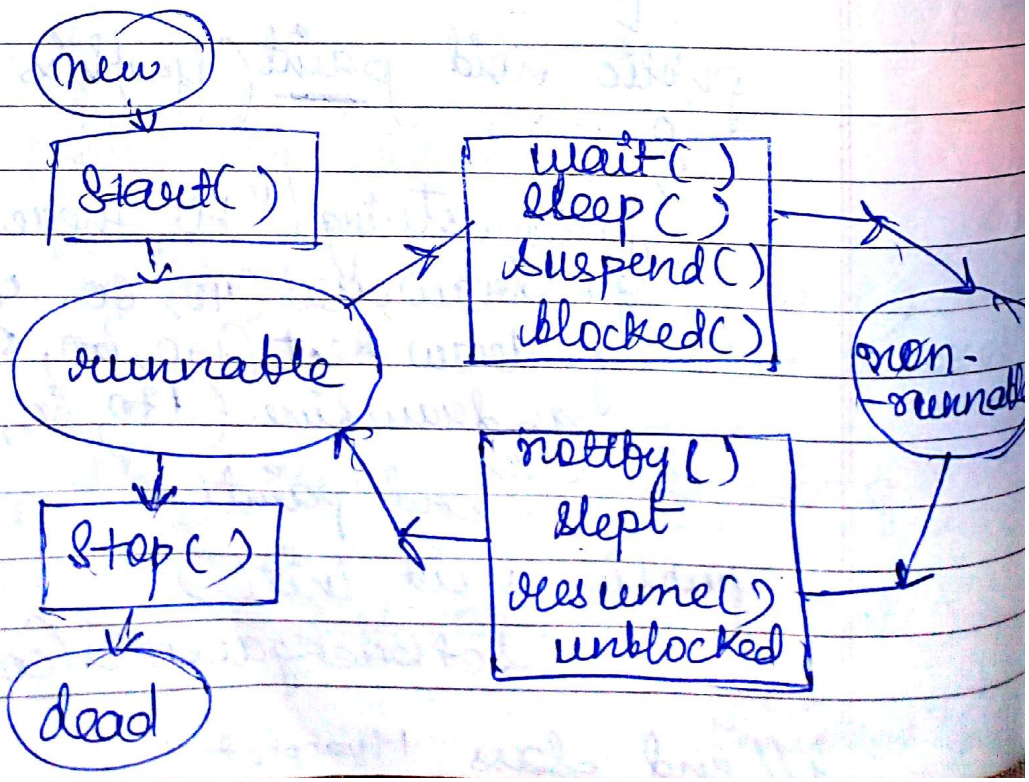
→ Java has built in thread support for multithreading.

→ 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.

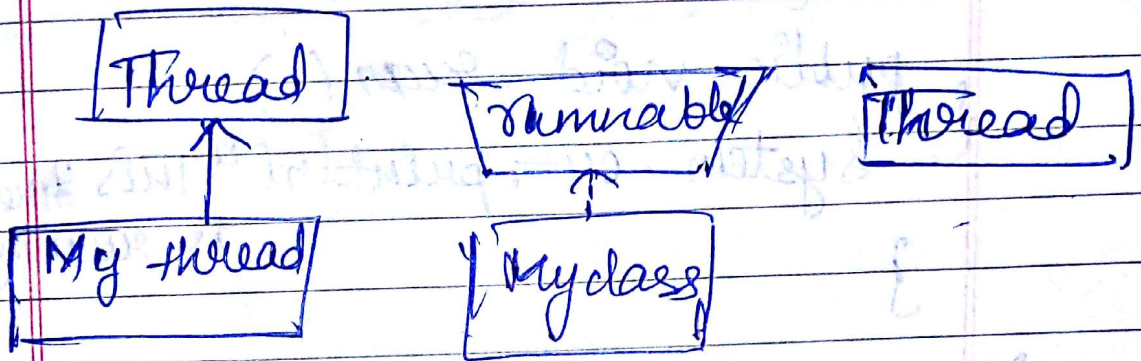
11. Thread States :-



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 2 :- // 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 Interface.

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

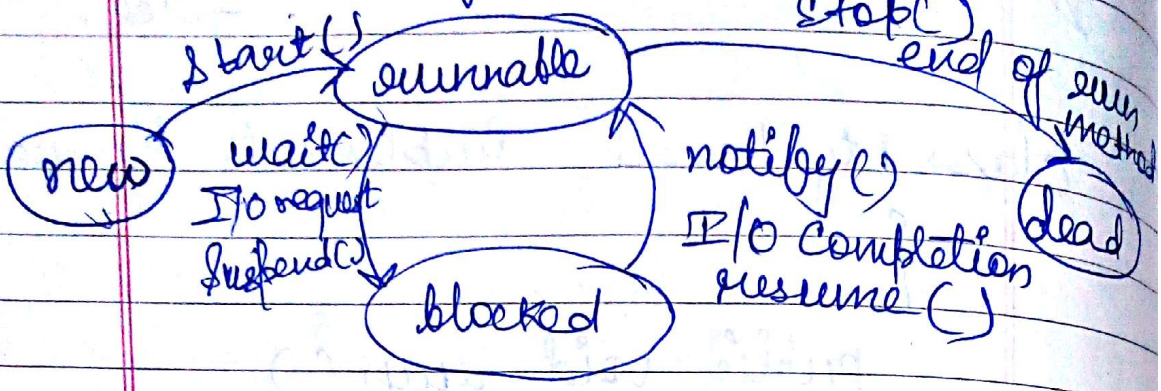
① Creating object of
MyThread myobject = new MyThread();

② Creating Thread object.

execution → top to bottom.

Date :
Page No.

11. States of Java Threads



12. Creating Threads Example 6-

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

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

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();
}
}
```

(11). 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();
  }
}
```