Java

Unit - 1

→ Java is a general purpose, object oriented programming language developed by Sun Microsystem of USA in 1991.

→ original name of Java was "Oak" by James Gosling

→ "Oak" was renamed "Java" in 1995.

Java features

① **Compiled and Interpreted**

Java combines compiled and interpreter approaches thus making Java a two-stage system.
First, Java compiler translates source code into what is known as bytecode instructions. Bytecode are not machine instructions, therefore,
Second, Java interpreter generates machine code that can be directly executed by machine that is running the Java program.

② **Platform independent and Portal**

Portability, Java programs can be easily moved from one computer to another, anywhere and anytime.
Changes and upgrades in OS, processors, system resources will not force any changes in Java program

Portability in 2 ways :-

① Java compiler generates bytecode instructions that can be implemented on any machine.

② Size of primitive data types are machine independent.

③ Object-oriented

→ Java is object-oriented language. Almost every thing in Java is an object. All program code and data reside within objects and classes.

→ Java comes with an extensive set of classes, arranged in packages, that we can use in our programs by inheritance.

④ Robust and Secure

→ Java is a robust language. It provides many safeguards to ensure reliable code.

→ It also incorporates concept of exception handling which captures serious errors and eliminates any risk of crashing the system.

→ Java system not only verify all memory access but also ensure that no viruses are communi-cated with an applet.

→ The absence of pointers in Java ensures that programs cannot gain access to memory locations without proper authorization.

⑤ Distributed

→ Java is designed as distributed language for creating applications on networks. It has the ability to share both data and programs.

→ Java applications can open and access remote objects on Internet as easily as they can do in local system.

→ This enables multiple programmers at multiple remote ~~objects on Internet as easily as they~~ locations to collaborate and work together on single project.

(6) Simple, small and familiar

→ Java is small and simple language. Many features of C and C++ that are either redundant or sources of unreliable code are not part of Java.

→ Eg. Java doesn't use pointers, preprocessor header files, goto statement and many other. It also eliminates operator overloading, multiple inheritance.

→ Familiar :- To make the language look familiar, it was modelled on C and C++ languages.

(7) Multithreaded and Interactive

→ Multithreading means handling multiple tasks simultaneously.

→ Java supports multithreading, means we need not wait for application to finish one task before beginning another.

(8) High performance

- due to use of intermediate byte code.
- incorporation of multithreading enhances the overall execution speed.

(9) Dynamic & Extensible

→ Java is dynamic language. It is capable of dynamically linking in new classes, libraries, methods and objects.

→ Java can also determine type of class through a query, making it possible to either dynamically link or abort program, depending on response.

→ Java programs support functions written in other languages such as C and C++. These functions are known as native methods

→ Native methods are linked dynamically at runtime.

(10) Ease of development.
(11) Scalability and Performance
(12) Monitoring and Manageability

How Java Differs from C and C++

① Java and C

Java is object oriented and has mechanism to define classes and object (major difference)

* Java doesn't include keywords - sizeof & typedef.
* Java doesn't contain data types struct and union.
* Java doesn't define type modifier keywords - auto, extern, register, signed and unsigned.
* Java doesn't support an explicit pointer type
* Java doesn't have preprocessor & therefore can't use #define, #include, #ifdef statements
* Java requires that functions with no arguments must be declared with empty parenthesis and not with void keyword as done in C.
* Java adds new operators - instanceof and >>>.
* Java adds labelled break and continue.
* Java adds many features reqd for OOP.

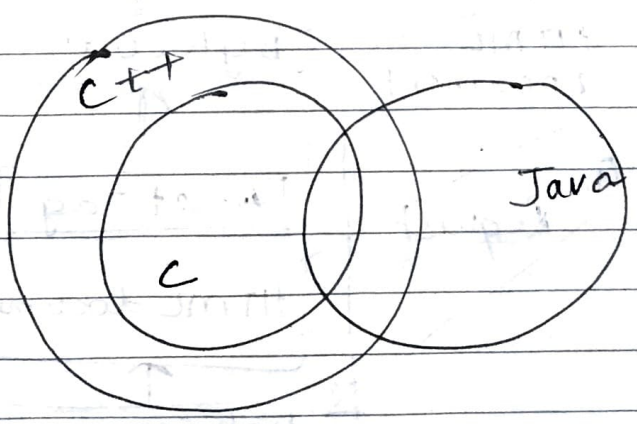
(2) Java and C++

Java is true object oriented language while C++ is basically C with object oriented extension.

* Java doesn't support operator overloading
* Java doesn't have template classes as in C++
* Java doesn't support multiple inheritance of classes. This is accompalished using new feature called 'interface'
* Java doesn't support global variables. Every variable and method is declared within class and forms part of that class.
* Java doesn't use pointers.
* Java has replaced distructor function with finalize() function.
* There are no header files in Java

→ Java also adds some new features. While C++ is a superset of C, Java is neither a superset nor a subset of C or C++.

→ Java may be considered as first cousin of C++ and second cousin of C.



Overlapping of C, C++ and Java.

Java and Internet

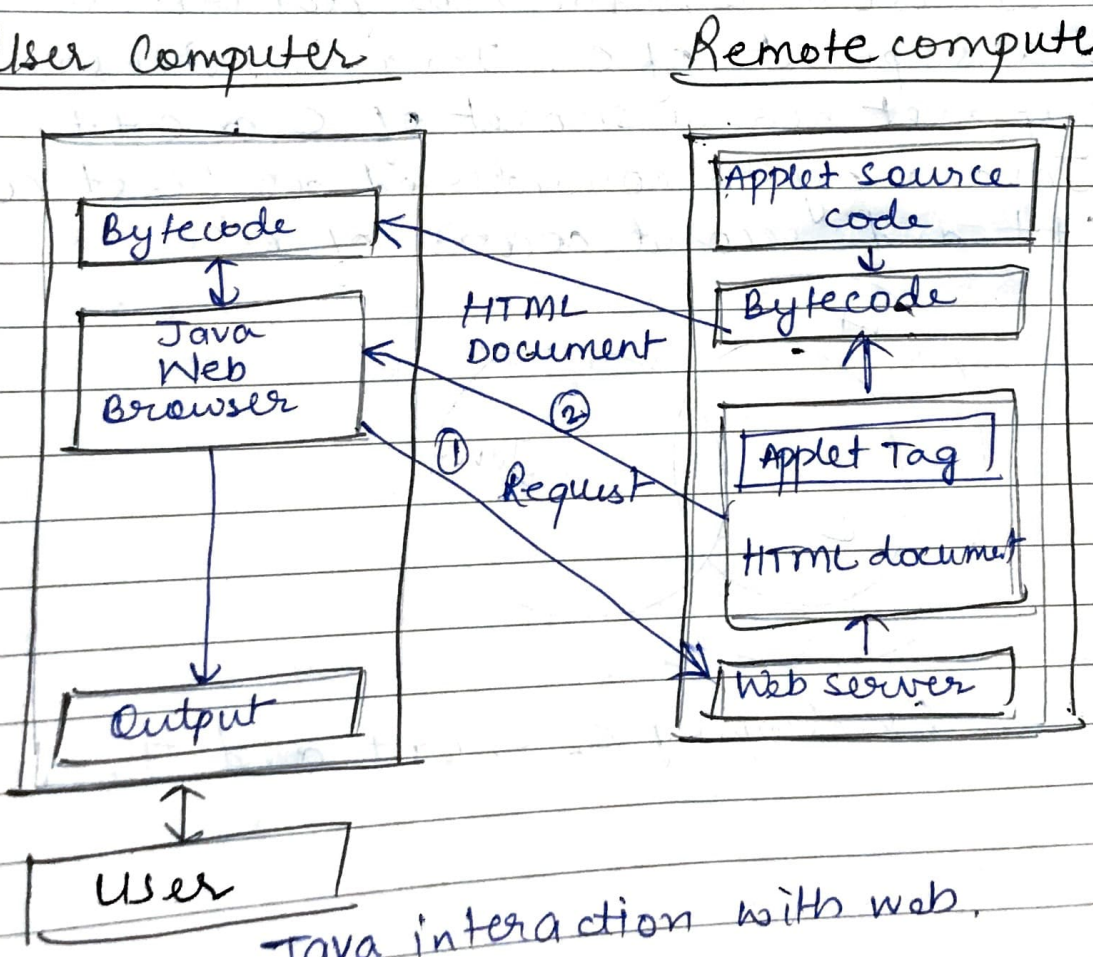
Java is strongly associated with Internet because first application program written in Java was HotJava, a web browser to run applets on Internet.

Internet users can use Java to create applet programs and run them locally using "Java-enabled browser" such as HotJava and download applets also from internet and save locally.

Java is popularly known as "Internet language"

Java and WWW

Java communicates with web page through a special tag called <APPLET>



Java interaction with web.

Communication steps :-

① The User sends a request for an HTML document to remote computer's web server. The web server is a program that accepts a request, processes the request and sends the required document.

② HTML document is returned to the user's browser. The document contains APPLET tag which identifies the applet.

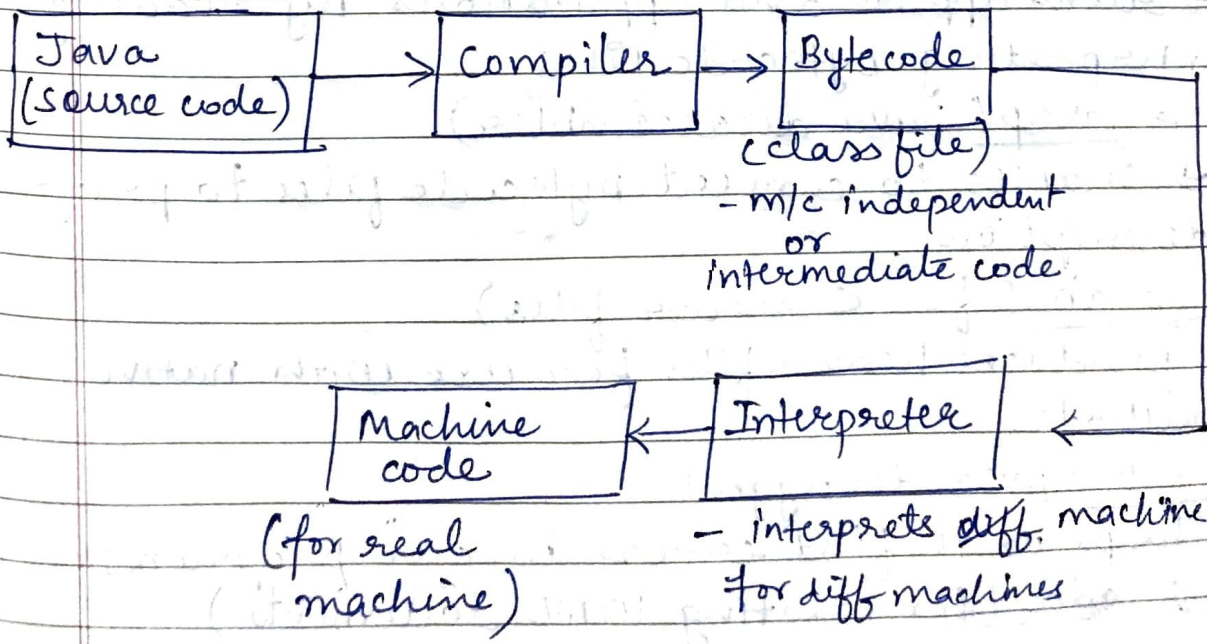
③ The corresponding applet bytecode is transferred to user's computer. The bytecode had been prev. created by Java compiler using Java source code for that applet.

④ The Java enabled browser on user's computer interprets bytecode and provides output.

⑤ User may have further interaction with applet but no further downloading from provider's web server, because bytecode contains all the information necessary to interpret applet.

JVM (Java Virtual Machine)

- software w/c interpret bytecode and generates machine code



→ It is known as virtual machine because it is a simulator computer inside a computer. It completes the machine work.

Java Environment

→ Includes large no. of development tools and hundreds of classes and methods. The development tools are part of system known as Java Development Kit (JDK) and classes and methods are part of Java Standard Library (JSL) also known as Application Programming Interface (API).

Java Development Kit

① appletviewer (for viewing Java applets)
→ without actually using Java-compatible browser

② javac (Java compiler)
→ translates Java source code to bytecode files that the interpreter can understand.

③ java (Java interpreter)
→ runs applets and applications by reading and interpreting bytecode files.

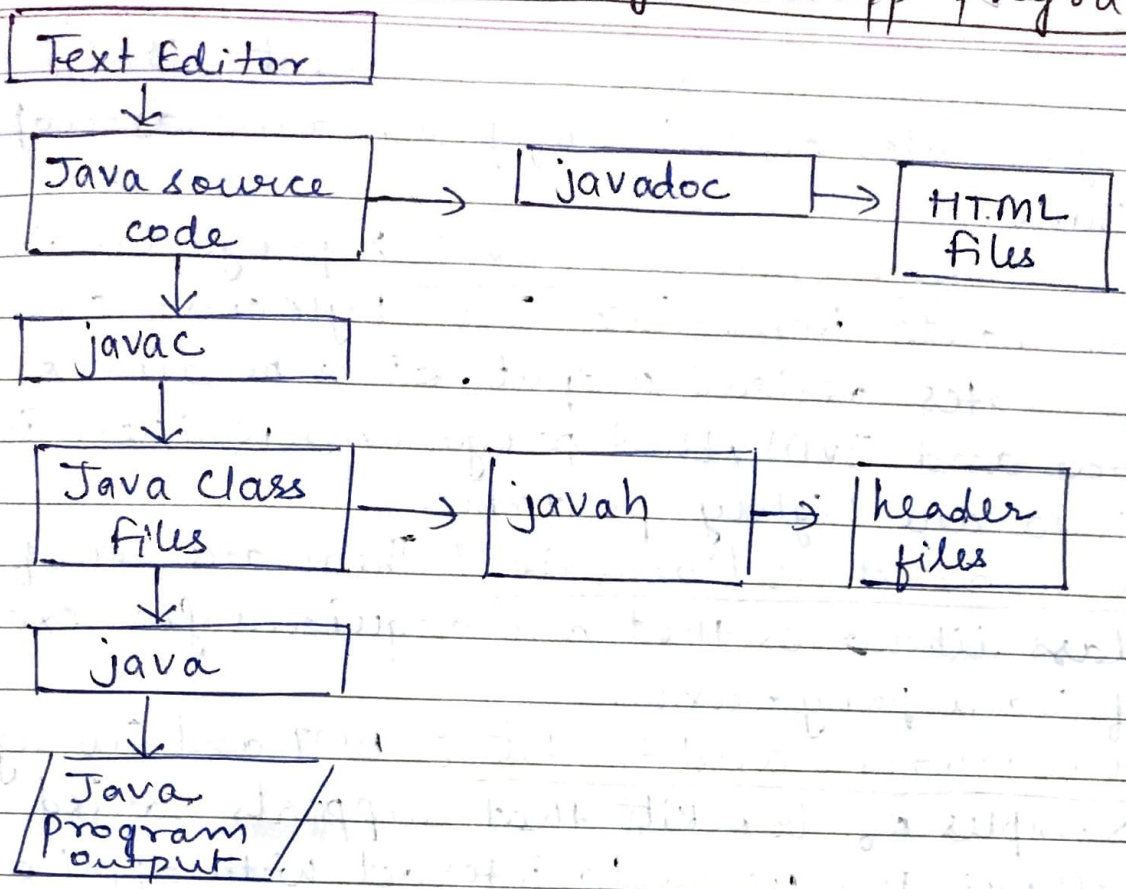
④ javap (Java disassembler)
→ enables to convert bytecode files to program description.

⑤ javah (for C header files)
- produces header files for use with native methods.

⑥ jdb (java debugger)
- helps us to find errors in our programs.

⑦ javadoc (for creating HTML documents)

Process of building, running Java app programs



API

① Language Support system: Collection of classes & methods reqd. for implementing basic features of Java.

② Utilities Package: Collection of classes to provide utility functions such as date and time functions.

③ Input & output package: Collection of classes reqd for input / output manipulation.

④ Networking Package: Collection of classes for communicating with other computers via Internet

⑤ AWT Package: Abstract Window Tool Kit package contains classes that implementing platform-independent graphical user interface.

⑥ Applet Package: Includes set of classes that allow us to create Java applets

Java Runtime Environment (JRE)

→ facilitates execution of programs developed in Java.

→ Java Virtual Machine (JVM) :- program that interprets immediate Java byte code and generates desired output. It is because of byte code and JVM, that programs written in Java are highly portable.

→ Runtime class libraries : These are set of core class libraries that are required for execution of Java programs.

→ User interface and toolkits : AWT and Swing are examples of toolkits that supports varied input methods for users to interact with application program.

→ Deployment technologies : JRE comprises the following key development technologies :

→ java plug-in : Enables the execution of Java applet on browser.

→ Java Web start : Enables remote deployment of an application.

Implementing Java Program

- JDK must be properly installed.

① Creating program

→ Text editor :

```
class Test
{
    public static void main (String args[])
    {
        =
        =
        y
    }
}
```

→ save this program in format Classname.java like 'Test.java' || This file is Source File

* If program has multiple classes, then filename must be the classname of the class containing main method.

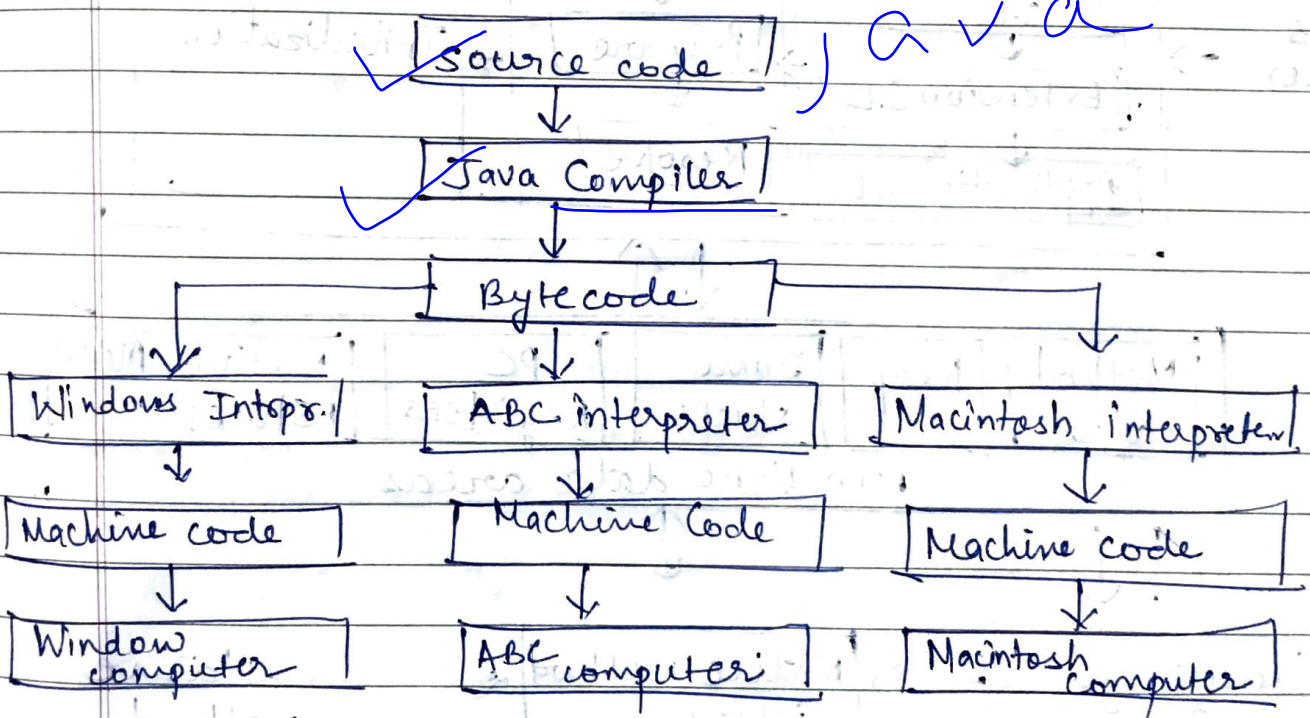
② Compiling program

→ In cmd :

javac Test.java || java compiler

* If everything is OK, the javac compiler creates a file Test.class containing the bytecodes of the program. Note, compiler automatically name the bytecode file as :-

<classname>.class

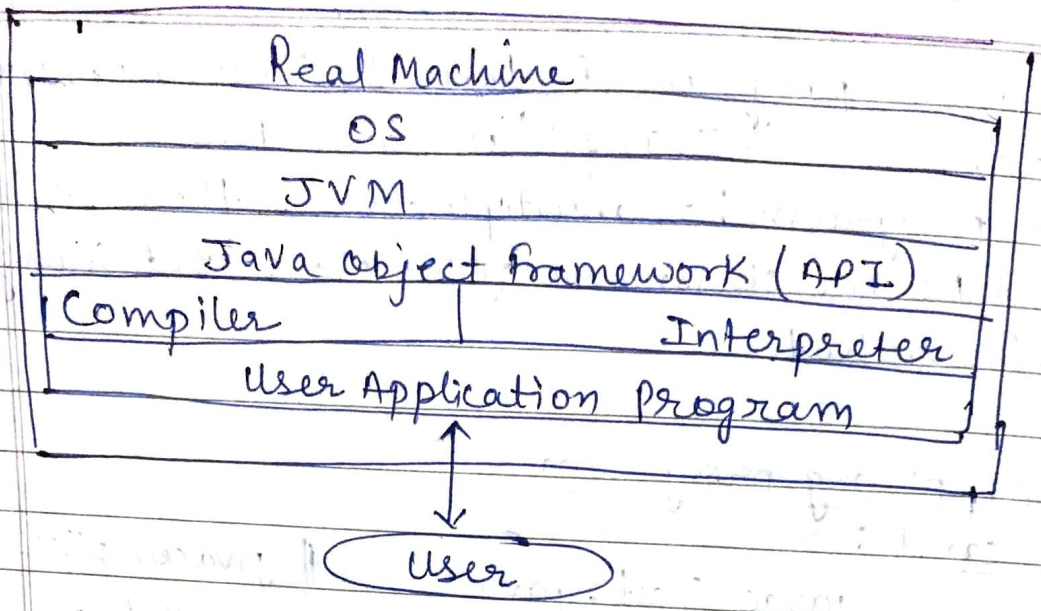


③ Running program

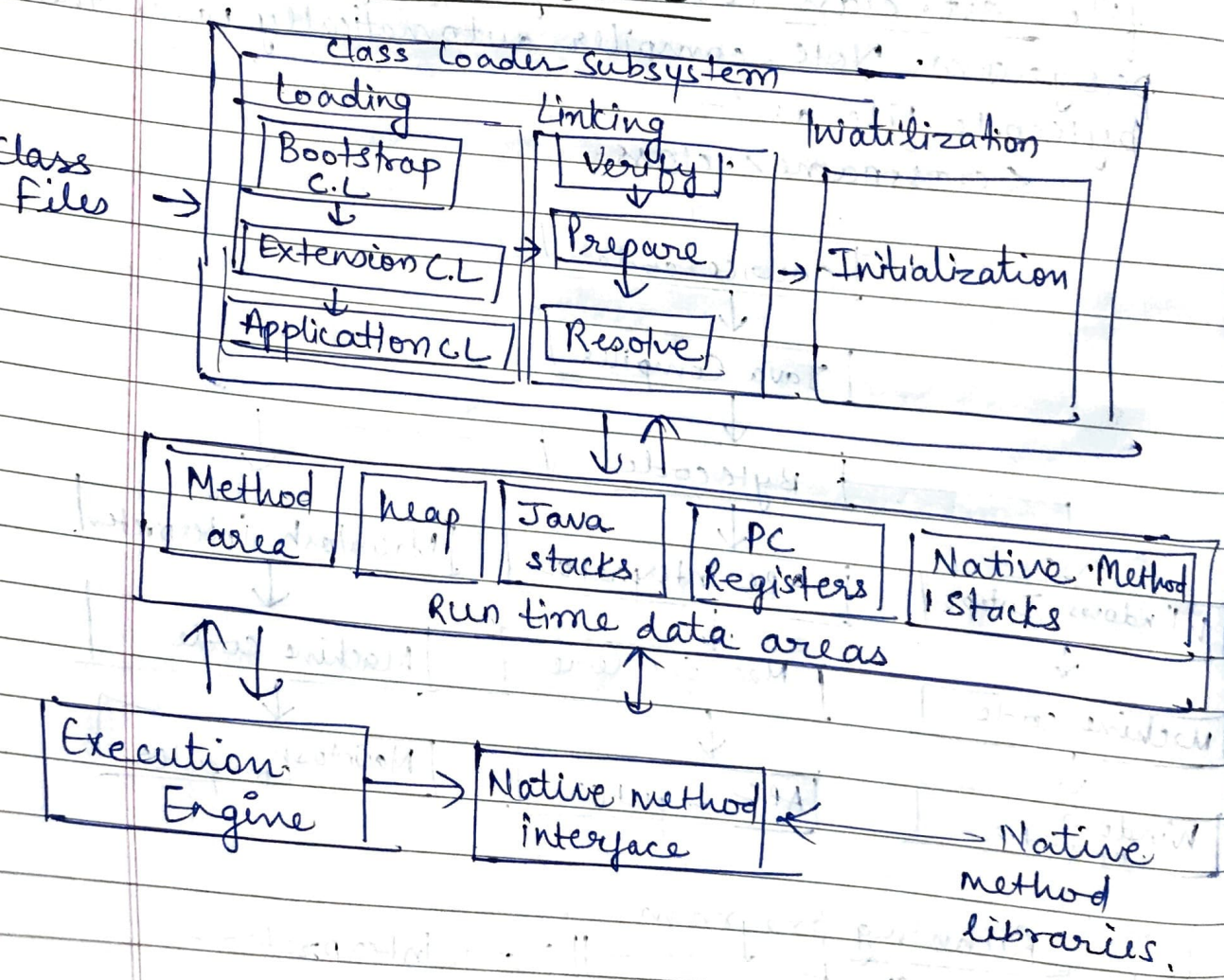
In cmd : java Test || Java Interpreter

Now, interpreter looks main method in program & begins execution from there.

Layers of Interaction for java program



JVM Architecture



→ Responsibility of JVM Loader is to load the application (class file) and execute the application.

1. Class Loader Subsystem

↳ responsibility is to load the bytecode, links and initializes class file when it refers to a class for the first time at run time, not compile time.

① Loading : classes will be loaded by this component There are 3 class loader :-

a) Bootstrap ClassLoader : Responsible for loading classes from bootstrap classpath, nothing but rt.jar. (Highest priority)

b) Extension ClassLoader : Responsible for loading classes which are inside the ext folder (jre\lib)

c) Application ClassLoader : Responsible for loading Application level Classpath, path mentioned Environment variable etc.

② Linking :

a) Verify : Bytecode verifier will verify whether the generated bytecode is proper or not if verification fails we will get the verification error

b) Prepare : For all static variables memory will be allocated and assigned with default values.

c) Resolve : All symbolic memory references are replaced with original references from Method Area

③ Initialization :- All static variables will be assigned with original values and static block will be executed.

② Runtime Data Area

JVM divided into 5 major components

1. Method area (or Class Area)
2. Heap area
3. Stack area
4. PC Registers
5. Native Method Stacks

① Class Area

- Stores the classes that are loaded into system. Method implementation kept in space called as Method area and constants kept in a space called constant pool.
- Class definitions serve as templates for objects. Objects stored in heap.

Classes have properties:

→ Superclass
→ List of field
→ List of constants
→ List of interfaces
→ List of methods & implementation

→ All properties of classes → immutable (ie no way to change any property of class once it has been brought into system., makes m/c stable.

② Heap area

:- Objects are stored in heap. and their corresponding instance variable & arrays store here.

③ Java stack :- Keeps track of which methods have been called and the data associated with each method invocation.

→ Each time a method is invoked, a new space called stack frame is created. Collectively, the stack frames are called the Java Stack.

→ The stack frame on top of the stack is called active stack frame

→ Each stack frame has operand stack, an array of local variables, and a pointer to currently executing instruction. This instruction pointer is called Program counter (PC). PC points to method area.

JVM As Emulator And Interpreter

→ Java processor can be implemented as software. It is implemented as a program that reads bytecodes and performs operations they specify.

→ Some interpreter run source code written in HLL like Basic, others (like Java interpreter) run bytecodes.

Emulator :- Interpreter is called as emulator because it emulates hardware, but in fact is software. A java bytecode interpreter can be created for any computer system. Once you have Java compiler and Java interpreter, you can run any Java program no matter what type of computer you have.

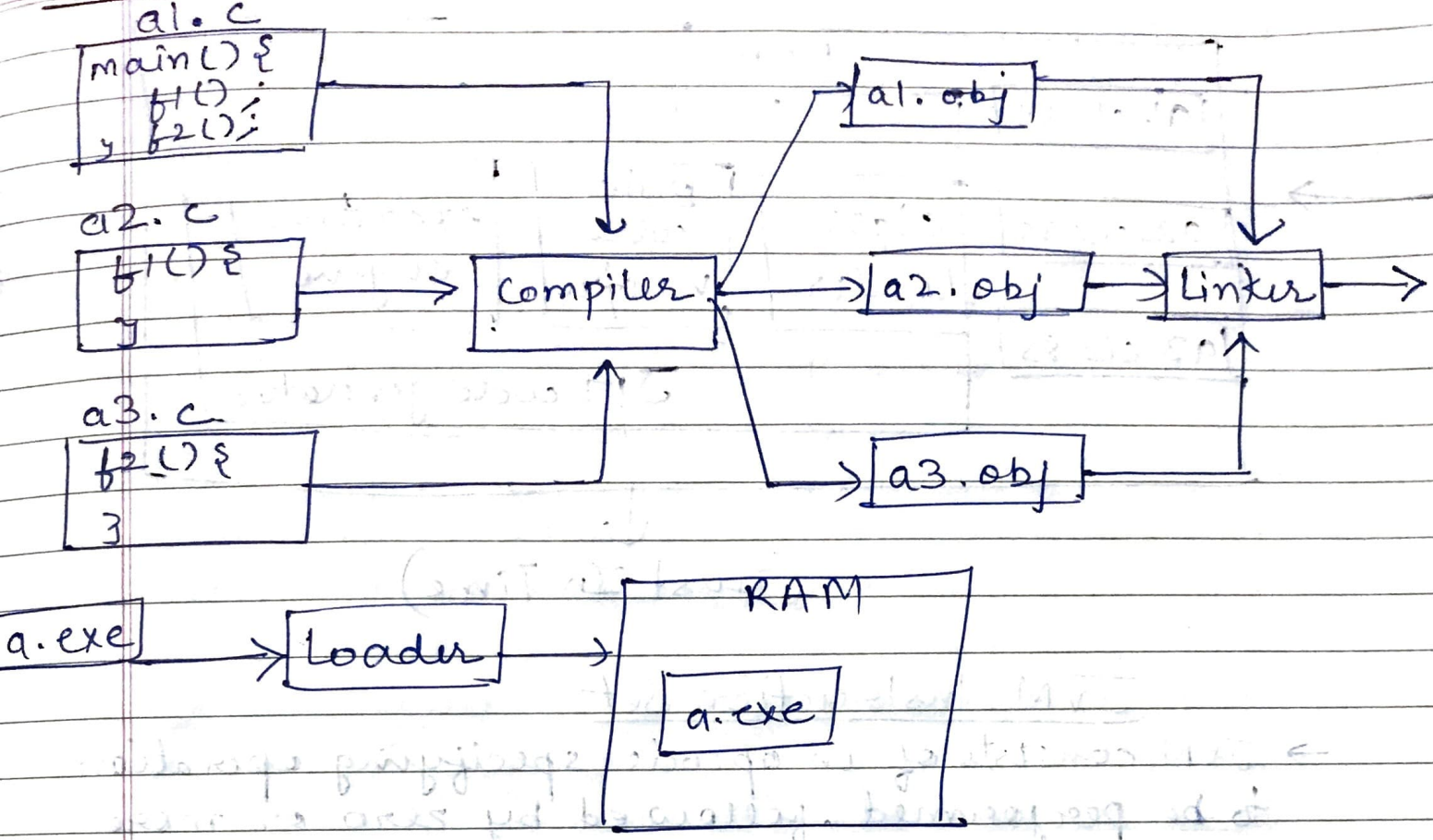
Java Interpreter → executable program ie running on ordinary computer system, such as desktop Intel system. Each type of computer system has its own Java interpreter and that can be run on that system.

* When Java interpreter is running on a computer system, that system acts just like a hardware Java bytecode processor. It is JVM.

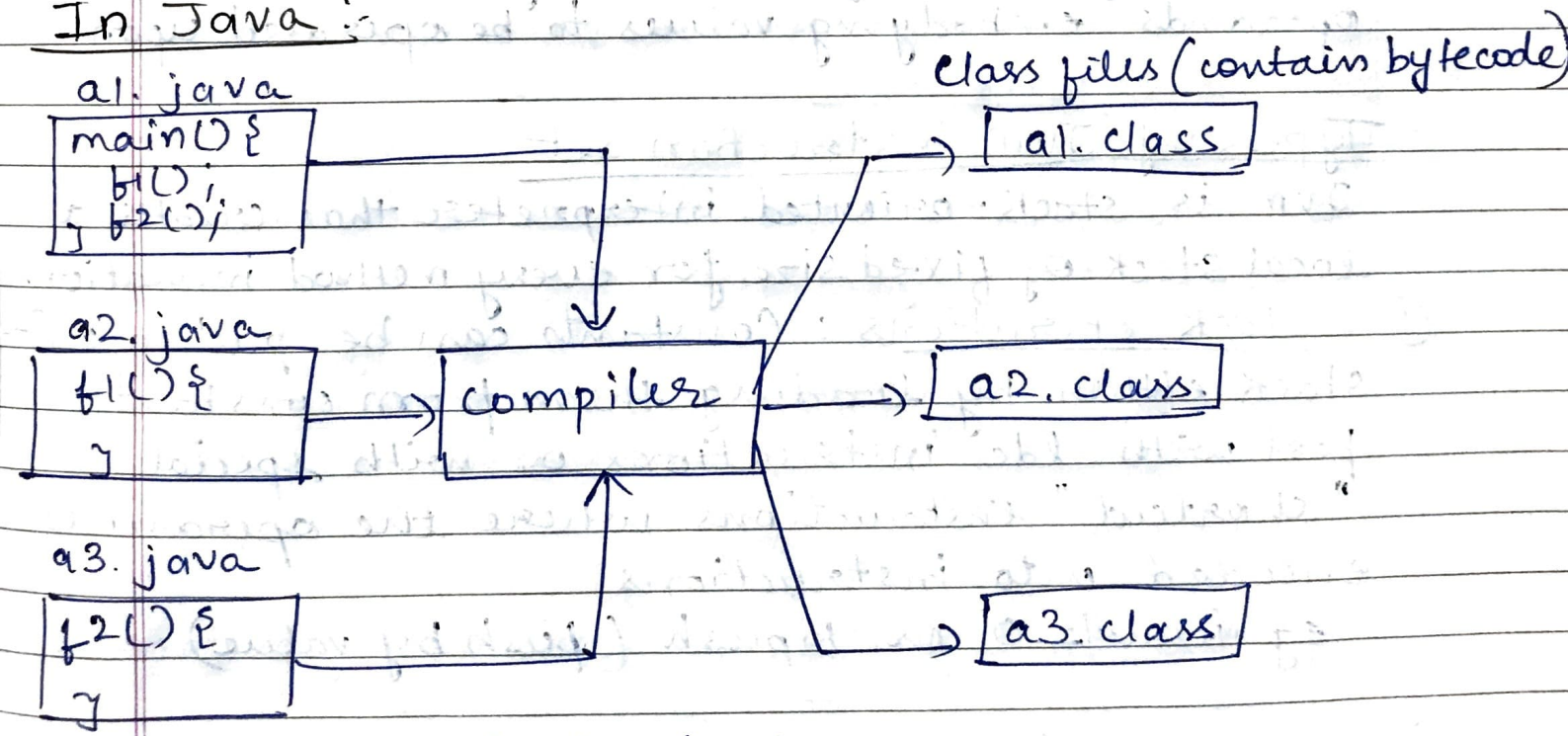
* Any computer system can execute a Java bytecode program by using Java interpreter. Java interpreter has to be specifically written for that particular computer system, but once ie done, computer system can become JVM. ie it looks like computer with H/W Java processor chip and run Java bytecode.

Eg. Java source program (ie Hello.java) can be written and compiled on one computer (say Windows) to produce bytecode (say Hello.class) Now, the bytecode can be run on any computer that has Java interpreter.

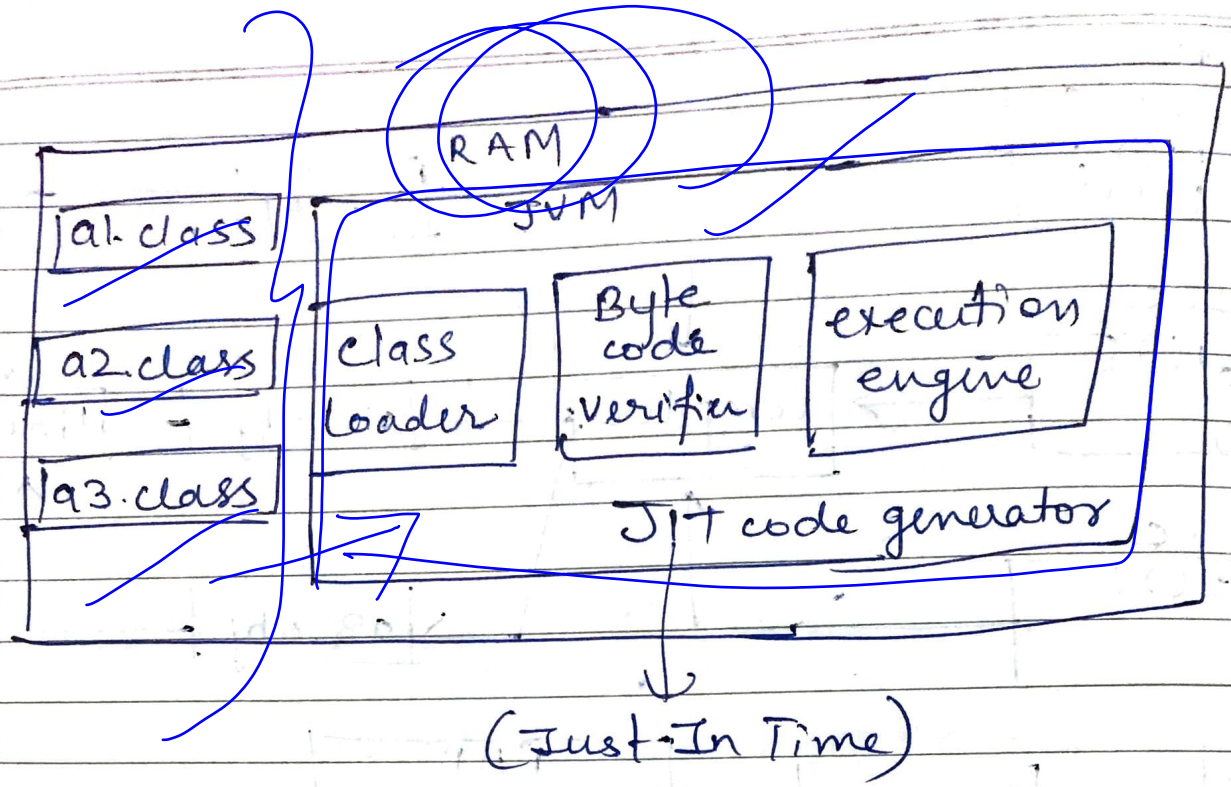
In C:



In Java:



No Linking is done



JVM Instruction Set

→ JVM consists of an opcode specifying operation to be performed, followed by zero or more operands embodying values to be operated upon.

Types of JVM Instruction set

JVM is stack-oriented interpreter that creates a local stack of fixed size for every method invocation.

① Stack operations : Constants can be pushed on stack either by loading them from constant pool with ldc instructions or with special "shortcut" instructions where the operand is encoded into instructions
eg iconst_0 or bipush (push by value)

② Arithmatic operations : It distinguishes its operand types using different instructions to operate on values of specific type.

③ Control flow: There are branch instructions like if_icmpeq, which compares two integers for equality to implement try-catch blocks. Exceptions may be thrown with athrow instruction.

④ Field access: Value of instance field may be retrieved with getfield and written with putfield.

⑤ Method invocation: Static methods may either be called via invokestatic or be bound virtually with invokevirtual instruction.

⑥ Object allocation: Class instances are allocated with new instruction, array of basic type like int[] with newarray, arrays of references like String[][] with anewarray or multianewarray.

⑦ Conversion and type checking: For stack operand of basic type there exists casting operations like f2i that converts float value into integer. The validity of typecast may be checked with checkcast and instance of operator can be directly mapped to equally named instruction.

Format of Instruction Descriptions

Format: mnemonic // (it means symbol)
operand1
operand2
---

forms: mnemonic = opcode. It is its numeric rep. & given in both decimal and hexadecimal form

Class File format

It starts with

1) Header containing "magic number" (COXCAFEBABE) and Version number followed by :-
2) Constant pool ie text segment of executable.
3) access right of class encoded by a bit mask
4) List of interfaces implemented by class
5) List containing fields
6) methods of class and
7) class attributes eg. Source file attribute telling the name of source file.

Attributes are way of putting additional user-defined information into class file data structures.

Verification

→ In order to ensure that certain parts of machi are kept safe from tampering, the JVM has verification algorithm to check every class.

→ Purpose: to ensure that programs follow se of rules that are designed to protect securit of JVM.

→ Programs try to subvert security of JVM in variety of ways :-
- → overflowing of stack
- → hoping to corrupt memory that are not allowed to access
- → try to cast an object inappropriately.
- → hoping to obtain pointers to forbidden memory.

→ Verification algo ensures this does not happen by tracing through code to check that objects are always used according to their propertype.

→ Verification algo is applied to every class as it is loaded into system, before instances are created or static properties used. This allows JVM implementation to assume that class has certain safety properties, which permit the implementation to make optimization based on that assumption.

Garbage Collection

→ Each object consumes some memory, of which there is a limited amount. Eventually, the memory allocated to these objects automatically through process called garbage collection

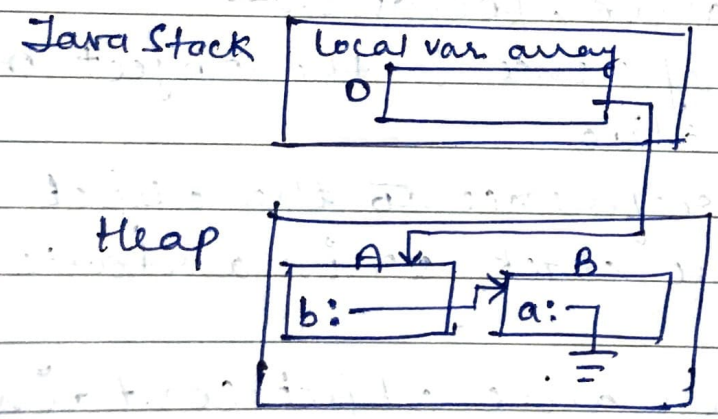
→ An object is ready to be garbage collected when it is no longer "alive". Rules for determining if an object is alive, follows:—

1) If there's a reference to the object on the stack, then it's alive
2) If there's a reference to the object in a local variable, on the stack or in a static field, then it's alive.
3) The field of an alive object contains a reference to object, then its alive.
4) JVM internally keep references to certain objects, for eg, to support native methods. These objects are alive.

Eg.
```
class A
{
    B b;    // A field pointing to an object of type B
}
class B
{
    A a;    // B field pointing to an object of type A
}

A foo = new A();   // Create object of class A called foo
foo.b = new B();   // Create an object of class B
                   // and put a reference to it in
                   // b field of foo.
```

Case: Code has 2 objects, one of class A and one of class B (we'll call these A & B). There is a reference of A in local variable foo (w/c is stored in local var. array slot 0) and a reference to B in b field of A.



In this, both A and B are alive.

Case 2: foo.b = null;
This sets b field to point to no object at all. Now, B doesn't meet any of 4 conditions for aliveness; there are no arrows pointing at all. This means, B is now dead, and it can be garbage collected. A is still alive, so it must not be garbage collected.

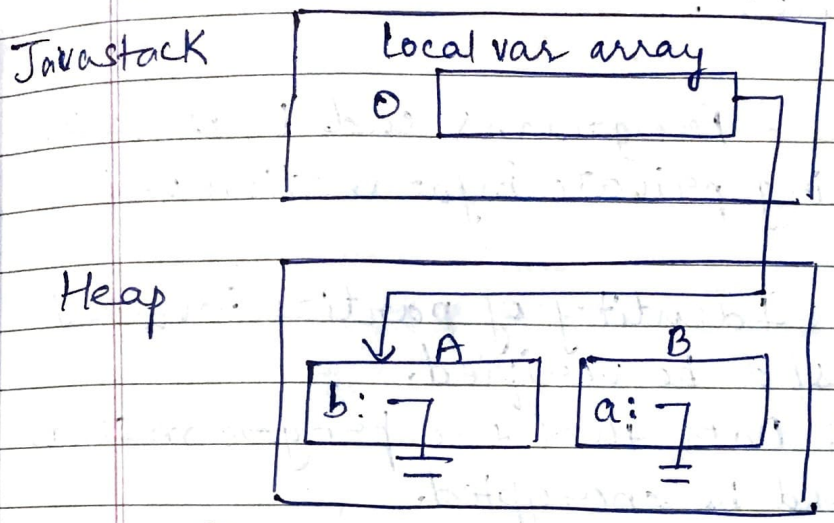


Fig: object on right is dead.

Finalization

→ In some cases, it is helpful to know when an object is no longer used so that you can do some final cleaning up.

→ To handle this, provide a method called finalize in your class. This method is called when the garbage collector notices that object is dead.

→ finalize() method is defined in class java.lang.Object to do nothing

```
public void finalize()
{
    this.close();    // close connection (in case of database)
}
```

Java Security

Java is a secure language.

Need of Java Security

1. Safe from malevolent programs: Program shd not be allowed to harm user's computing environment. This includes Trojan horses as well as harmful programs that can replicate themselves.
2. Non-intrusive :- Programs shd be prevented from discovering private information on the host computer.
3. Authenticated: Identity of parties involved in program shd be verified.
4. Encrypted: Data that the program sends and receives shd be encrypted.
5. Audited: Potentially sensitive operations shd always be logged.
6. Well defined: A well defined security specifi^{n} would be followed.
7. Verified: Rules of operations shd be set & verified.
8. Well behaved: Programs shd be prevented from consuming too many system resources

Security Promises of JVM

1. Every object is constructed exactly once before it is used.
2. Every object is an instance of exactly one class, which does not change through life of object
3. If a field or method is marked private,

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

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

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

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

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

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

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

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

Security Architecture & Secure Policy

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

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

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

proceeding with any operation under the control of security Manager.

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

The Java Sandbox Security Models:

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

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

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

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

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

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

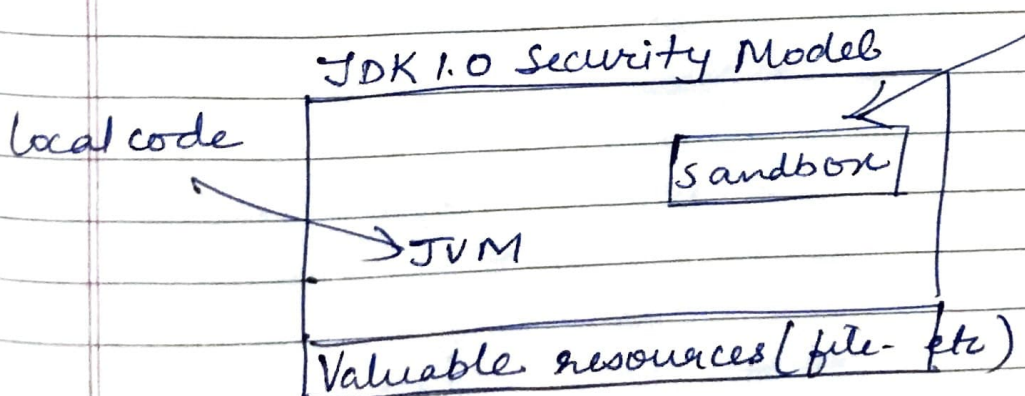
* In Java, sandbox is the program area and it has some set of rules that programmers need to follow when creating Java code (like applet) is sent as part of page.

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

Security (Sandbox)

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

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



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

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

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

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

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