Idris Rattemwala
2003145 - C31

PAGE NO.
DATE / /

JAVA ASSINGMENT. 1

Write Short note on:

1. Features of java :

1) Simple : Java is easy to learn and its syntax is quite simple, clean and easy to understand. The ambigous concepts of C++ are either left out in java or they have been re-implemented in cleaner way.

2) Object Oriented : In java everything is an object ~~oriented~~ which has some data and behaviour. Java can be easily extended as its based on Object Model. following are basic concepts of OOP's.
i) Object
ii) Class
iii) Inheritance
iv) Polymorphism
v) Abstraction
vi) Encapsulation

3) Robust : Java makes an effort to eliminate error prone codes by emphesising mainly on compile time error checking and runtime checking. But the main areas which java improved were Memory management and mishandeled exceptions.

4) Platform independent : Unlike other programing languages such as C, C++, ~~[illegible]~~ which are compiled into platform specific machines, Java is gaurenteed to be write-once & run anywhere language.

On compilation, java program is compiled into bytecode. This bytecode is platform independent. and can be run on any device.

PAGE No.
DATE / /

5) Secure : when it comes to security, jave is always the first choice. With jave secure features, it enable us to develop virus free, temper free system. It always runs in $\overset{jave}{}$ runtime environment with almost null interactions with system OS, hence more secure.

6) Multi threading : Java multithreading makes it possible to write program that can do many task simultanously. Benifit of multithreading is that it utilizes same memory/ other resources to execute multiple threads at same time.

7) Potable : Java byte code can be carried to any platform. No implementation dependent features. Everything related to storage is predefined, example :- size of primitive data types.

8) Distributed : Java is also distributed language. Programs can be designed to run on computer networks. Java has a special class library for communicating ~~protocol~~ using TCP / IP protocols.

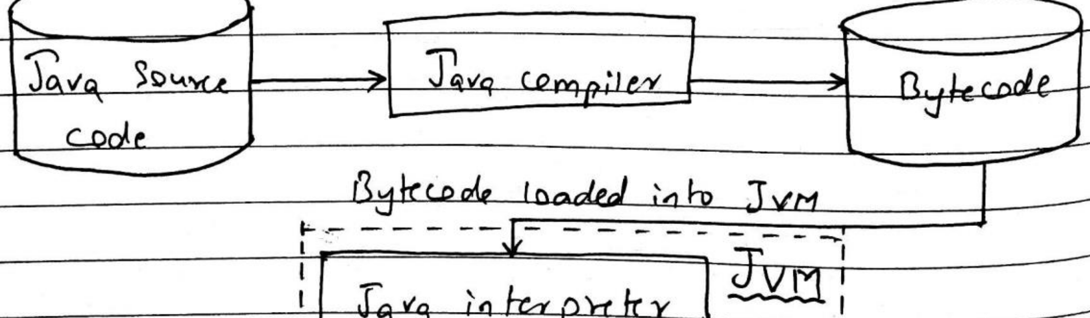
PAGE NO.
DATE / /

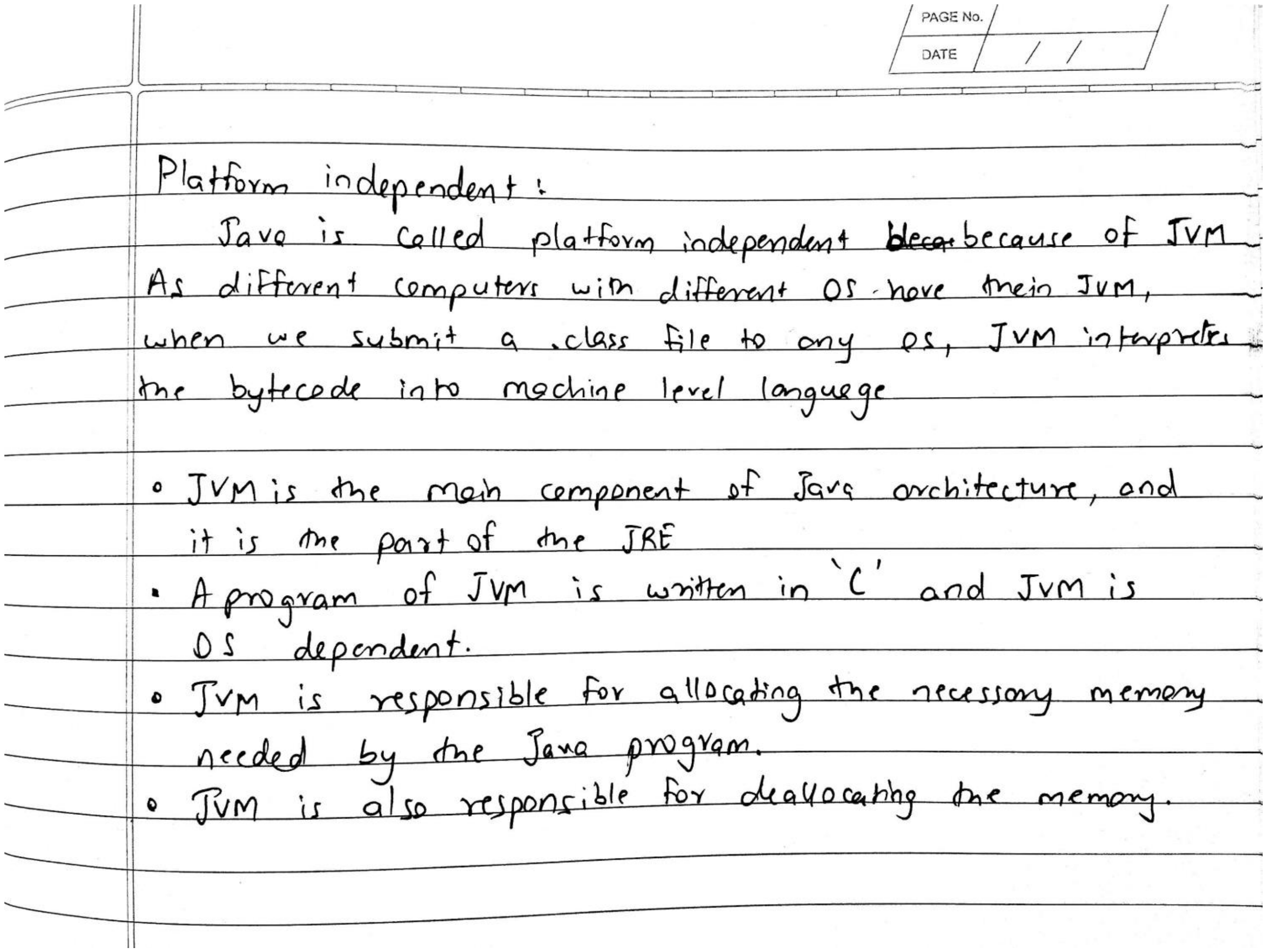
2. JVM :

The java virtual machine is called as JVM, is an abstract computing machine or virtual machine ~~that~~ interface that drives the java code.

- Mostly in other compiling languages, compiler produces a code for a particular system, but java ~~produces~~ compiler produces byte code for a java virtual machine.
- When we compile a Java program, then a bytecode is generated. Bytecode is the source code that can be used to run on any platform.
- Bytecode is an intermediate language between java source and the host systems.
- It is the medium which compiles javacode to bytecode which gets interpreted on diffrent machines and hence makes it platform/OS independent.

JVM's work can be explained in the following manner:
- Reading the bytecode.
- verifing bytecode.
- Linking the code with the library.





PAGE No.

DATE / /

Platform independent :

Java is called platform independent ~~becau~~ because of JVM As different computers with different OS have their JVM, when we submit a .class file to any OS, JVM interpretes the bytecode into machine level language

- JVM is the main component of Java architecture, and it is the part of the JRE
- A program of JVM is written in 'C' and JVM is OS dependent.
- JVM is responsible for allocating the necessary memory needed by the Java program.
- JVM is also responsible for deallocating the memory.

3) Wrapper class :

Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc. Let us see the different scenarios, where we need to use the wrapper classes.

- **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.
- **Serialization:** We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.

- **Synchronization:** Java synchronization works with objects in Multithreading.
- **java.util package:** The java.util package provides the utility classes to deal with objects.
- **Collection Framework:** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

The eight classes of the *java.lang* package are known as wrapper classes in Java. The list of eight wrapper classes are given below:

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Autoboxing

The automatic conversion of primitive data types into its wrapper class objects is known as autoboxing.

```
class Autoboxing {
    public static void main( String args[] ) {
     int a = 15; // Primitive data type
     Integer I = a; // Autoboxing will occur internally.
    }
}
```

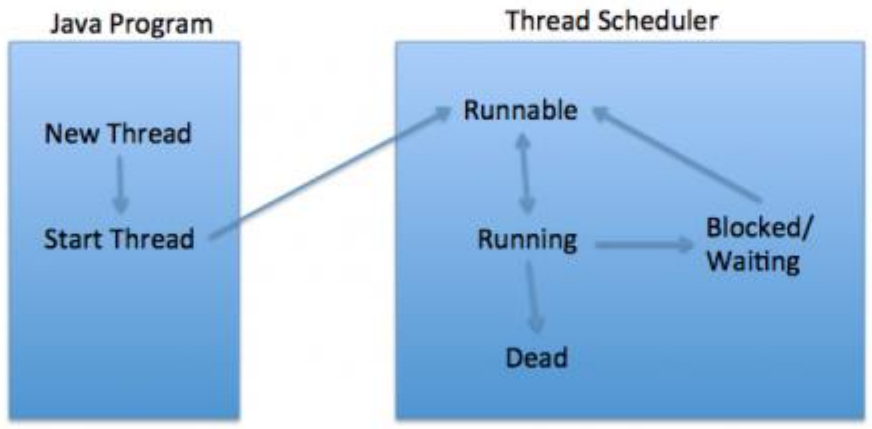
Unboxing

The automatic conversion of wrapper class objects into its primitive data types is known as unboxing.

```
class Autoboxing {
    public static void main( String args[] ) {
      Integer a = new Integer(15); // Wrapper class object
      int I = a;// Unboxing will occur internally.
    }
}
```

4) Life cycle of a thread :

Below diagram shows different states of thread life cycle in java. We can create a thread in java and start it but how the thread states change from Runnable to Running to Blocked depends on the OS implementation of thread scheduler and java doesn't have full control on that.



New

When we create a new Thread object using *new* operator, thread state is New Thread. At this point, thread is not alive and it's a state internal to Java programming.

Runnable

When we call start() function on Thread object, it's state is changed to Runnable. The control is given to Thread scheduler to finish it's execution. Whether to run this thread instantly or keep it in runnable thread pool before running, depends on the OS implementation of thread scheduler.

Running

When thread is executing, it's state is changed to Running. Thread scheduler picks one of the thread from the runnable thread pool and change it's state to Running.

Then CPU starts executing this thread. A thread can change state to Runnable, Dead or Blocked from running state depends on time slicing, thread completion of run() method or waiting for some resources.

Blocked/Waiting

A thread can be waiting for other thread to finish using thread join or it can be waiting for some resources to available. For example producer consumer problem or waiter notifier implementation or IO resources, then it’s state is changed to Waiting. Once the thread wait state is over, it’s state is changed to Runnable and it’s moved back to runnable thread pool.

Dead

Once the thread finished executing, it’s state is changed to Dead and it’s considered to be not alive.

Above are the different **states of thread**. It’s good to know them and how thread changes it’s state. That’s all for thread life cycle in java.