IAT1 - Scheme of Evaluation

Question number	Scheme	Marks
1	JDK JVM JRE	3 4 3
2 (a) (b)	 Output & explanation Output & explanation	 4 6
3	Multiplication program	10
4	Applet lifecycle Applet program	5 5
5	Java exception Exception Handling Example Program	2 ½ 5 2 ½
6	Final with example Finally with example Finalize with example	2+1 2+1 2+2
7 (a) (b)	 Life cycle of thread with diagram and explanation Factorial program with recursion	 2+3 5

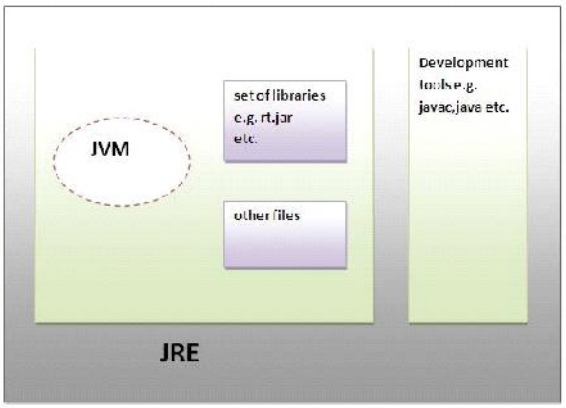
IAT1 – JAVA and J2EE Solution

Q1 Briefly Explain JDK with JVM & JRE

Solution:

Java Development Kit(JDK):

(JDK) is a software development environment used for developing Java applications and applets. It includes the Java Runtime Environment (JRE), an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development.

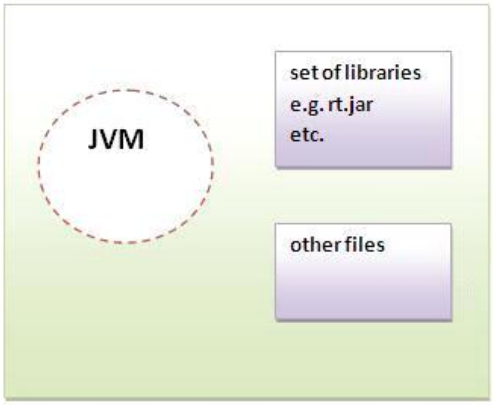


JDK

JDK=JRE + Tools (compiler,debugger,tools necessary for developing java application)

Java consists of JRE(Java Runtime Environment) that has JVM, platform,core classes, libraries.

Java Runtime Environment (**JRE**) is a set of software tools for development of Java applications. It combines the Java Virtual Machine (JVM), platform core classes and supporting libraries. **JRE** is part of the Java Development Kit (JDK), but can be downloaded separately.



JRE

JVM (Java Virtual Machine)

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

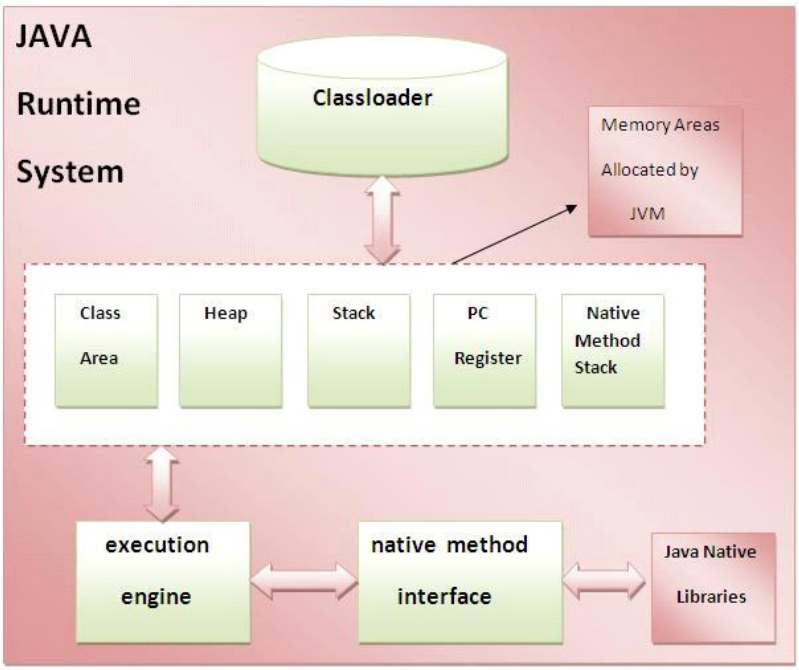
The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

- Internal Architecture of JVM



1) Classloader

Classloader is a subsystem of JVM that is used to load class files.

2) Class(Method) Area

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3) Heap

It is the runtime data area in which objects are allocated.

4) Stack

Java Stack stores frames.It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5) Program Counter Register

PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.

6) Native Method Stack

It contains all the native methods used in the application.

7) Execution Engine

It contains:

1) A virtual processor

2) Interpreter: Read bytecode stream then execute the instructions.

3) Just-In-Time(JIT) compiler: It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term ?compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

Q2 (a) Correct Error in following code segment with explanation

```
byte b;
b=b*2;
```

Solution: Error:incompatible types :possible lossy conversion from int to byte.

When multiplying b(byte) with 2 (int) by java's automatic type promotion result will be an integer.But,'b'is of type byte .An integer value cannot be assigned to a byte variable without typecasting.

(b) Write the output of following code segment with explanation:

```
byte b;
int i = 200;
b = (byte) i;
System.out.println(b);
```

Solution:

Output: -56

Explanation: byte occupies 1 byte(8 bits) and integer occupies 4 bytes (32 bits) in java

200 in binary will be stored in int as below

00000000 00000000 00000000 11001000

After type casting into byte datatype the higher order MSBs are truncated and we are left with 11001000

Byte is a signed data type, and every signed number is represented by 2's complement method hence the given number is signed as there is a 1 on MSB so reversing the 2's complement will give below:

11001000

00000001 –

11000111, now convert this to get the ones complement

00111000 (1’s complement)

This is 56 in decimal and hence after applying the sign it will be -56.

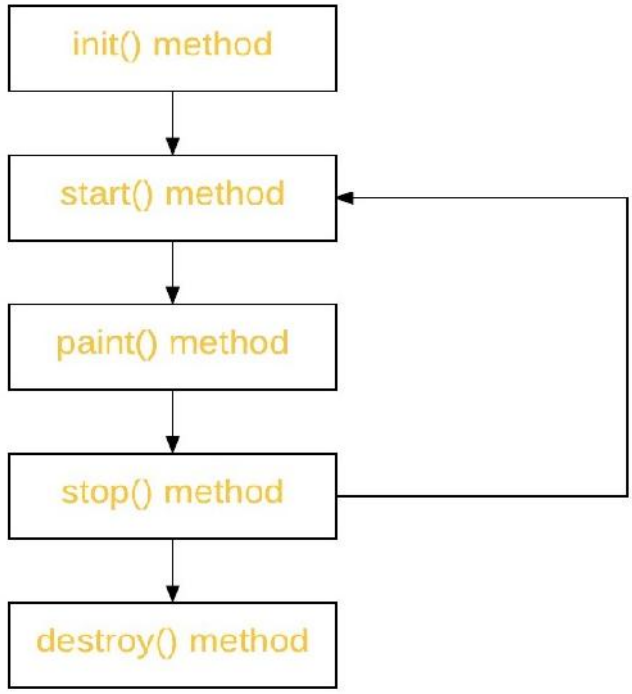
Q3 Write a java program to create multiplication table of number given by user during execution through keyboard

Solution:

```
//To create multiplication table of number entered through keyboard
import java.io.*;
class Multable{
  public static void main(String args[]) throws IOException {
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
  String str;
  int n,x;
  System.out.println("Enter number to create Multiplication Table: ");
  str=br.readLine();
  n=Integer.parseInt(str);
  System.out.println(" Multiplication Table of  " + n);
  for(x=1;x<=20;x++)
    System.out.println(n + " * " + x + " = " + n*x);
  }
}
```

Q4 Explain life cycle of an applet. Develop a Java applet (With HTML Tag) that sets background colour cyan, foreground colour red and outputs a string message “Welcome to CMRIT”.

Solution:



Life Cycle of an applet

It is important to understand the order in which the various methods shown in the above image are called. When an applet begins, the following methods are called, in this sequence: 1. init( ) 2. start( ) 3. paint( ) When an applet is terminated, the following sequence of method calls takes place: 1. stop( ) 2. destroy( ) Let’s look more closely at these methods.

1. init( ) : The init( ) method is the first method to be called. This is where you should initialize variables. This method is called only once during the run time of your applet.

2. start( ) : The start( ) method is called after init( ). It is also called to restart an applet after it has been stopped. Note that init( ) is called once i.e. when the first time an applet is loaded whereas start( ) is called each time an applet’s HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at start( ).

3. paint( ) : The paint( ) method is called each time an AWT-based applet’s output must be redrawn. This situation can occur for several reasons. For example, the window in which the applet is running may be overwritten by another window and then uncovered. Or the applet window may be minimized and then restored. paint( ) is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, paint( ) is called. The paint( ) method has one parameter of type Graphics. This parameter will contain the

graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required.

4. stop( ) : The stop( ) method is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example. When stop( ) is called, the applet is probably running. You should use stop to suspend threads that don't need to run when the applet is not visible. You can restart them when start( ) is called if the user returns to the page.

destroy( ) : The destroy( ) method is called when the environment determines that your applet needs to be removed completely from memory. At this point, you should free up any resources the applet may be using. The stop( ) method is always called before destroy( ).

```
import java.applet.Applet;
import java.awt.*;
public class myapplet1 extends Applet{
    public void paint(Graphics g){
  setBackground(Color.cyan);
  setForeground(Color.red);
   g.drawString("Welcome to CMRIT",50,100);
}
}
/* <applet code="myapplet1.class" width="300" height="300">   </applet> */
```

Q5 What is Java exception ? Explain exception handling mechanism with try, catch and throw with example program.

Solution:

Exceptions in Java : An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program's instructions.

Exception Handling : Java exception handling is managed via five keywords: try, catch, throw, throws, and finally. Briefly, here is how they work. Program statements that you think can raise exceptions are contained within a try block. If an exception occurs within the try block, it is thrown. Your code can catch this exception (using catch block) and handle it in some rational manner. Systemgenerated exceptions are automatically thrown by the Java run-time system. To manually throw an exception, use the keyword throw. Any exception that is

thrown out of a method must be specified as such by a throwsclause. Any code that absolutely must be executed after a try block completes is put in a finally block. Need of try-catch clause(Customized Exception Handling) Consider the following java program.

```
// java program to demonstrate
// need of try-catch clause
class DEMO {
    public static void main (String[] args) {

        // array of size 4.
        int[] arr = new int[4];
        // this statement causes an exception
        int i = arr[4];
        // the following statement will never execute
        System.out.println("Hi, I want to execute");
    }
}
```

Output :

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
    at DEMO.main(DEMO.java:9)
```

Explanation : In the above example an array is defined with size i.e. you can access elements only from index 0 to 3. But you trying to access the elements at index 4(by mistake) that’s why it is throwing an exception.In this case, JVM terminates the program abnormally. The statement System.out.println(“Hi, I want to execute”); will never execute. To execute it, we must handled the exception using try-catch. Hence to continue normal flow of the program, we need try-catch clause. How to use try-catch clause

```
try {
// block of code to monitor for errors
// the code you think can raise an exception

}
catch (ExceptionType1 exOb) {
// exception handler for ExceptionType1
```

```
}
catch (ExceptionType2 exOb) {
// exception handler for ExceptionType2
}
// optional
finally {
// block of code to be executed after try block ends
}
```

Q6. Explain final, finally and finalize of java.

Solution: There are many differences between final, finally and finalize. A list of differences between final, finally and finalize are given below:

No.	Final	Finally	Finalize
1)	Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	Finally is used to place important code, it will be executed whether exception is handled or not.	Finalize is used to perform clean up processing just before object is garbage collected.
2	Final is a keyword.	Finally is a block.	Finalize is a method.

Java final example

```
class FinalExample{
public static void main(String[] args){
 final int x=100;
x=200;//Compile Time Error
```

}}

Java finally example

```
class FinallyExample{
public static void main(String[]args){
try{
int x=300;
}
catch(Exception e){
System.out.println(e);
}
finally{
System.out.println("finally block is executed");
} }}
```

Java finalize example

```
Class FinalizeExample{
public void finalize(){
System.out.println("finalize called");
}
public static void main(String[] args){
FinalizeExample f1=new FinalizeExample();
FinalizeExample f2=new FinalizeExample();
f1=null;
f2=null;
System.gc();
}}
```

Q7 (a) Describe life cycle of a thread with diagram ?

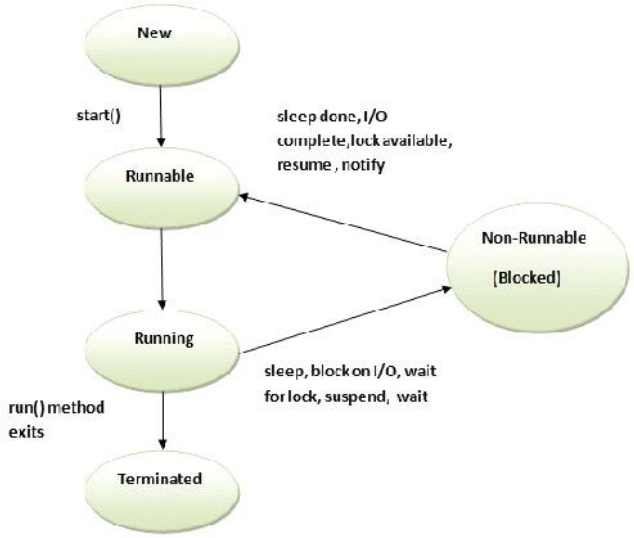
Solution:

A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.

But for better understanding the threads, we are explaining it in the 5 states.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

- New
- Runnable
- Running
- Non-Runnable (Blocked)
- Terminated



Life cycle of a Threaad

1) New

The thread is in new state if you create an instance of Thread class but before the invocation of start() method

2) Runnable

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

3) Running

The thread is in running state if the thread scheduler has selected it.

4) Non-Runnable (Blocked)

This is the state when the thread is still alive, but is currently not eligible to run.

5) Terminated

A thread is in terminated or dead state when its run() method exits.

(b) Develop a Java program to find Factorial of a number using recursion.

Solution:

```
import java.util.Scanner;
class Fact
{
  public static void main(String args[])
  {
    int num;
  Scanner s=new Scanner(System.in);
  System.out.println("Enter a number:");
  num=s.nextInt();
  //Fact f=new Fact();
  System.out.println("Factorial of"+num+" is "+fact(num));
  }
   static int fact(int n)
  {
    if (n==1||n==0)
  return 1;

        else
  return (n*fact(n-1));
   }
}
```