**H180128H**

**GIFT W. T RUWENDE**

**ICS 212 OOP IN JAVA ASSINGMENT 01**

**COMPUTER SCIENCE**

CLASS LOADER

NATIVE METHOD STACK

EXECUTION ENGINE

NATIVE METHOD INTERFACE

NATIVE METHOD LIBRARIES

PC REGISTERS

JVM LANGUAGE STACK

HEAP

METHOD AREA

* Part of the java runtime environment
* Acts as a runtime engine to run java applications
* Calls the man method in a java file

Class loader subsystem

Mainly responsible for three activities:

1. Loading
   * The class loader reads two class file, generate corresponding binary data
2. Initialization
   * All static variables are assigned their values defined in the code and static block

The three class loaders are:

1. Bootstrap class loader
2. System application class loader
3. Extension class loader
4. Linking

* Performs verification, resolution and preparation.

JVM Memory

Native Memory stacks

* It stores runtime method information
* For every thread, a separate native stack is created

PC Register

* Stores address at current execution instruction of a thread
* Each thread has separate registers

Heap Area

Information of all objects is stored here

It is a shared resource

Stack Area

JVM creates a runtime stack for every thread, which is stored here

The runtime stack is then destroyed after the termination of a thread.

It is not a shared resource.

Native Method Stacks

Creates a separate native stack for every thread.

Stores native method information.

Execution engine

Executes the byte code

Can be defined in three parts:

1. Interpreter -> interprets the byte code line by line and executes
2. Just - In – Time Compiler -> used to increase efficiency of the interpreter.
3. Garbage Collector -> it destroys unreferenced objects

Method Area

* All class level information like class, name, methods and variables are stored including all static variables

Java Native Instruction

* Interface which interacts with the Native Method Libraries and provide native libraries required for execution.

Native Method Libraries

* A collection of the native libraries (C, C++) which are required by the execution engine

2.

public static void main(String[] args){

//declaring first matrix called numbers

int[][] numbers = new int[2][2];

//declaring first matrix called numbers2

int[][] numbers2 = new int[2][2];

//adding elements to numbers using the random function

for(int r =0;r<2;r++){

for(int s = 0; s < 2; s++){

int rand = (int)(Math.random()\*10 );

numbers[r][s] = rand;

System.out.print(numbers[r][s]);

}

System.out.println("\n");

}

System.out.println("THE SECOND MATRIX IS");

//adding elements to numbers2 using the random function

for(int t = 0; t < 2; t++){

for(int u = 0; u < 2;u++){

int random = (int)(Math.random()\*10);

numbers2[t][u]=random;

System.out.print(numbers2[t][u]);

}

System.out.println("\n");

}

//declaring temporary array to store the product

int[][] temp = new int[2][2];

//calculating the product

for(int q = 0; q < 2;q++){

for(int r = 0; r < 2;r++){

temp[q][r]=numbers[q][r] \* numbers2[q][r];

}

}

for(int t = 0; t < 2; t++){

for(int u = 0; u < 2; u++){

System.out.print(temp[t][u] + " ");

}

System.out.println("\n");

}

}

3.

when an object, in memory, is no longer needed it must be removed or deleted from memory to free the memory space that can be reused for other objects.

This is called garbage collection, JVM implicitly sweeps out abandoned objects from the memory.

Code snippets

public class garbage collection{  
 public void finalize(){

System.out.println(“Object is Garbage, needs cleaning”);

}

public static void main(String[] args){

garbage newgarbage = new garbage();

garbage nextgarbage = new garbage();

newgarbage = null;

nextgarbage = null;

System.gc();// invokes garbage collector

}

}

Finalize

Is a protected and non-static method.

Used to perform final operations or change operations on an object before it is removed from memory.

It can be overrided to keep operations before an object is destroyed.

Code snippet

Protected void finalize(){

}

Data hiding is used to ensure exclusive data access to class members and provides object integration by presenting unintended or intended changes. Data hiding hidding helps to protect members of a class.

Access modifiers

In java some of the access modifiers used are:

* Public
* Private
* Protected
* Private

NB all are written in small letters

Objects of other classes can access public data members and methods of other classes.

Protected members are accessible by objects in the same class and in the subclasses of the class in which they exist. Private members are only accessible within the class.

Code snippets of access modifiers:  
private static String name; // declares a String variable called name with an access modifier of

//type private

public String name; //the string now is accessible outside of its class;

protected int age; //declares an integer with the protected access modifier.