**Theory**

1. **Why is Java platform independent?**

 It is a **platform-independent programming language**. Java doesn’t require the entire code to be rewritten for all the different platforms. It supports platform independence using **Java bytecode** and **Java Virtual Machine**. Java compiler javac converts the program code into byte code. This byte code is platform-independent and can run on any JVM operating system. JVM interprets the byte code to machine code, and the program is executed.

1. **What are the default values assigned to variables and instances in java?**
   1. There are no default values assigned to the variables in java. We need to initialize the value before using it. Otherwise, it will throw a compilation error of (Variable might not be initialized).
   2. But for instance, if we create the object, then the default value will be initialized by the default constructor depending on the data type.
   3. If it is a reference, then it will be assigned to null.
   4. If it is numeric, then it will assign to 0.
   5. If it is a boolean, then it will be assigned to false. Etc
2. **If you define a class and instantiate two objects with same field values, what result will equals(==) operator will return?** (false)
3. **What is method overloading and method overriding?**

When the method signature (name and parameters) are the same in the superclass and the child class, it’s called *overriding*. When two or more methods in the same class have the same name but different parameters, it’s called *overloading*.

The only difference in the return type of the method does not promote method overloading. The following example will furnish you with a clear picture of it

1. **Explain the use of final keyword in variable.**
   1. In Java, the final keyword is used as defining something as constant /final and represents the non-access modifier.

final variable:

When a variable is declared as final in Java, the value can’t be modified once it has been assigned. If any value has not been assigned to that variable, then it can be assigned only by the constructor of the class.

1. **When does a memory get qualified for garbage collection?**
2. **What are two different types of exceptions in java.**

1) **Checked Exception**

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) **Unchecked Exception**

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) **Error**

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

1. **Discuss Java Concurrency**

Java is a multi-threaded programming language which means we can develop multi-threaded program using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs. Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

1. **How can we create a Thread in Java?**

There are two ways to create Thread in Java - first by implementing Runnable interface and then creating a Thread object from it and second is to extend the Thread Class.

1. **What are the different stages in the thread lifecycle**

Java thread life cycle is as follows:

**New** – When the instance of the thread is created and the start() method has not been invoked, the thread is considered to be alive and hence in the NEW state.

Runnable – Once the start() method is invoked, before the run() method is called by JVM, the thread is said to be in RUNNABLE (ready to run) state. This state can also be entered from the Waiting or Sleeping state of the thread.

**Running** – When the run() method has been invoked and the thread starts its execution, the thread is said to be in a RUNNING state.

**Non-Runnable (Blocked/Waiting)** – When the thread is not able to run despite the fact of its aliveness, the thread is said to be in a NON-RUNNABLE state. Ideally, after some time of its aliveness, the thread should go to a runnable state.

A thread is said to be in a **Blocked** state if it wants to enter synchronized code but it is unable to as another thread is operating in that synchronized block on the same object. The first thread has to wait until the other thread exits the synchronized block.

A thread is said to be in a **Waiting** state if it is waiting for the signal to execute from another thread, i.e it waits for work until the signal is received.

**Terminated** – Once the run() method execution is completed, the thread is said to enter the TERMINATED step and is considered to not be alive.

1. **What are the main concepts of OOPS in Java?**

**Inheritance**: Inheritance is a process where one class acquires the properties of another.

**Encapsulation**: Encapsulation in Java is a mechanism of wrapping up the data and code together as a single unit.

**Abstraction**: Abstraction is the methodology of hiding the implementation details from the user and only providing the functionality to the users.

**Polymorphism**: Polymorphism is the ability of a variable, function or object to take multiple forms.

1. **What is a Java Semaphore, that is used in the thread synchronization?**

A Semaphore is used to limit the number of threads that want to access a shared resource. In other words, it is a non-negative variable that is shared among the threads known as a counter. It sets the limit of the threads. A mechanism in which a thread is waiting on a semaphore can be signaled by other threads.

If counter > 0, access to shared resources is provided.

If counter = 0, access to shared resources is denied.

In short, the counter keeps tracking the number of permissions it has given to a shared resource. Therefore, semaphore grants permission to threads to share a resource.

**Practical**

1. **Declare an infinite loop**
   1. For (;;) {}
   2. While(true){}
2. **Write a Java Program to find factorial of a given number**

public class FindFactorial {

public static void main(String[] args) {

int num = 10;

long factorialResult = 1l;

for(int i = 1; i <= num; ++i)

{

factorialResult \*= i;

}

System.out.println("Factorial: "+factorialResult);

}

}

1. **Write a Java program to create and throw custom exceptions.**

class Interview {

public static void main(String args[]) throws CustomException {

// Throwing the custom exception be passing the message

throw new CustomException(" This is my custom Exception ");

}

}

//Creating Custom Exception Class

class CustomException extends Exception{

//Defining Constructor to throw exception message

public CustomException(String message){

super(message);

}

}

1. **Write a Java Program to reverse a string**

class Interview{

public static void main(String[] args){

//Input String

String str = "Welcome to Interview";

//Pointers.

int i = 0, j = str.length()-1;

//Result character array to store the reversed string.

char[] revString = new char[j+1];

//Looping and reversing the string.

while(i < j){

revString[j] = str.charAt(i);

revString[i] = str.charAt(j);

i++;

j--;

}

//Printing the reversed String.

System.out.println("Reversed String = " + String.valueOf(revString));

}

}

1. **Create an example of java exception handling using try-catch statement. Is there a case when finally will not be executed? (System.exit(), fatal error)**

public class JavaExceptionExample{

public static void main(String args[]){

try{

//code that may raise exception

int data=100/0;

}catch(ArithmeticException e){System.out.println(e);}

//rest code of the program

System.out.println("rest of the code...");

}

}

1. **Create an example program to print 1 and 2 using threads by extending Thread class**

class ThreadDemo extends Thread {

private Thread t;

private String threadName;

ThreadDemo(String name) {

threadName = name;

System.out.println("Creating " + threadName );

}

public void run() {

System.out.println("Running " + threadName );

try {

for(int i = 4; i > 0; i--) {

System.out.println("Thread: " + threadName + ", " + i);

// Let the thread sleep for a while.

Thread.sleep(50);

}

} catch (InterruptedException e) {

System.out.println("Thread " + threadName + " interrupted.");

}

System.out.println("Thread " + threadName + " exiting.");

}

public void start () {

System.out.println("Starting " + threadName );

if (t == null) {

t = new Thread (this, threadName);

t.start ();

}

}

}

public class TestThread {

public static void main(String args[]) {

ThreadDemo T1 = new ThreadDemo("Thread-1");

T1.start();

ThreadDemo T2 = new ThreadDemo("Thread-2");

T2.start();

}

}