Practical 9 Questions

Q1. What are the different types of errors in Java?

→ there are 3 types of errors in JAVA :

* Compile time errors
* Run time errors
* Logical errors

Q2. What is the difference between compile time and run time error in JAVA?

* Compile time error :
  + These errors are errors which prevents the code from compiling because of error in the syntax such as missing a semicolon at the end of a statement or due to missing braces, class not found, etc. These errors will be detected by java compiler and displays the error onto the screen while compiling.
* Run time error :
  + These errors are errors which occur when the program is running. Run time errors are not detected by the java compiler. It is the JVM which detects it while the program is running.

Q3. What are the different types of exception (runtime error) in Java?

* ArithmeticException:
  + Dividing by zero
  + Int x = 100;
  + Int y = 0;
  + System.out.println(x/y);
* NullPointerException:
  + String data = null;
  + System.out.println(data.length());
* ClassNotFoundException:
  + This exception is raised when we try to access a class whose definition is not found.
* FileNotFoundException:
  + This exception is raised when a file is not accessible or does not open.
* ArrayIndexOutOfBoundsException:
  + String arr[] = new String[5];
  + Arr[6] = “More data”;
* InterruptedException:
  + It is thrown when a thread is waiting, sleeping, or doing some processing, and it is interrupted.
* NoSuchFieldException:
  + It is thrown when a class does not contain the field (or variable) specified.
* NoSuchMethodException:
  + It is thrown when accessing a method that is not found.
* NullPointerException:
  + This exception is raised when referring to the members of a null object. Null represents nothing.
* NumberFormatException:
  + This exception is raised when a method could not convert a string into a numeric format.
* RuntimeException:
  + This represents an exception that occurs during runtime.
* StringIndexOutOfBoundsException:
  + It is thrown by String class methods to indicate that an index is either negative or greater than the size of the string.
* IllegalArgumentException :
  + This exception will throw the error or error statement when the method receives an argument which is not accurately fit to the given relation or condition. It comes under the unchecked exception.
* IllegalStateException :
  + This exception will throw an error or error message when the method is not accessed for the particular operation in the application. It comes under the unchecked exception.
* IOException: It is thrown when an input-output operation failed or interrupted.

Q4. How to handle runtime exception in JAVA?

→ To handle runtime exceptions in JAVA write the piece of code in try, throw and catch block.

Q5. How to implement try throw and catch statements?

→

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

Text

Description automatically generatedText

Description automatically generated

Q6. Explain the concept of multiple catch blocks in JAVA with the help of an example?

→ Multiple catch blocks are used in Java to handle multiple exceptions that the try block throws.

For code, refer to the code of Q5.

Q7. What is the significance of the finally method?

→ This method is always executed no matter if there is an error in the try block or not.

For code, refer to code of Q5.

Q8. What is the concept of multithreading. How to implement it in Java programming?

→ Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms :

1. Extending the Thread class
2. Implementing the Runnable Interface

**Thread creation by extending the Thread class**  
We create a class that extends the **java.lang.Thread** class. This class overrides the run() method available in the Thread class. A thread begins its life inside run() method. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object.

Graphical user interface, text, application

Description automatically generated

**Thread creation by implementing the Runnable Interface**  
We create a new class which implements java.lang.Runnable interface and override run() method. Then we instantiate a Thread object and call start() method on this object.

Text

Description automatically generated

Text

Description automatically generated

**Thread Class vs Runnable Interface**

1. If we extend the Thread class, our class cannot extend any other class because Java doesn’t support multiple inheritance. But, if we implement the Runnable interface, our class can still extend other base classes.
2. We can achieve basic functionality of a thread by extending Thread class because it provides some inbuilt methods like yield(), interrupt() etc. that are not available in Runnable interface.
3. Using runnable will give you an object that can be shared amongst multiple threads.

Q9. What is applet code and explain the life cycle of applet code?

→ An applet is a Java program that can be embedded into a web page. It runs inside the web browser and works on the client-side. An applet is embedded in an HTML page using the APPLET or OBJECT tag and hosted on a web server. The entire life cycle of an applet is managed by the Applet Container. All applets are sub-classes (either directly or indirectly) of java.applet.Applet class. Applets are not stand-alone programs. They run either within a web browser or an applet viewer.

* Applets generate Dynamic content
* Applets work on the client-side
* The response time is fast

**Stages in the Life Cycle of Java Applet**

* Initializing an Applet
* Starting the Applet
* Painting the Applet
* Stopping the Applet
* Destroying the Applet

***Note:****In order to implement the Applet we need to import awt package :*

*java.awt.applet.\*;*

**Life Cycle of Applet**

**Step 1:** Initialization

public void init()

*There is no main method unlike our normal java programs. Every Applet will start it’s execution from init() method. It is executed only once*

**Step 2:**Start

public void start()

*After init() method start() method is invoked. Executed when the browser is maximized*

**Step 3:** Paint

public void paint (Graphics g)

*Paint method is used to display the content on the applet. We can create the objects or components to the applet or we can directly write a message on the applet. It will take Graphics class as a parameter.*

**Step 4:** Stop

public void stop()

*stop() method is used to stop the applet. It is executed when the browser is minimized.*

**Step 5:** Destroy

public void destroy()

*destroy() method is used to completely close the applet. It is executed when the applet is close****d.***

**Methods of Applet Life Cycle:**

There are five methods of an Applet Life Cycle namely;

1. init()
2. start()
3. paint()
4. stop()
5. destroy()

Q10. Explain the life cycle of thread?

# → **Life cycle of a Thread (Thread States)**

In Java, a thread always exists in any one of the following states. These states are:

1. New
2. Active
3. Blocked / Waiting
4. Timed Waiting
5. Terminated

## Explanation of Different Thread States

**New:** Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.

**Active:** When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is **runnable**, and the other is **running**.

* **Runnable:** A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time. It is the duty of the thread scheduler to provide the thread time to run, i.e., moving the thread the running state.  
  A program implementing multithreading acquires a fixed slice of time to each individual thread. Each and every thread runs for a short span of time and when that allocated time slice is over, the thread voluntarily gives up the CPU to the other thread, so that the other threads can also run for their slice of time. Whenever such a scenario occurs, all those threads that are willing to run, waiting for their turn to run, lie in the runnable state. In the runnable state, there is a queue where the threads lie.
* **Running:** When the thread gets the CPU, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.

**Blocked or Waiting:** Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.

For example, a thread (let's say its name is A) may want to print some data from the printer. However, at the same time, the other thread (let's say its name is B) is using the printer to print some data. Therefore, thread A has to wait for thread B to use the printer. Thus, thread A is in the blocked state. A thread in the blocked state is unable to perform any execution and thus never consume any cycle of the Central Processing Unit (CPU). Hence, we can say that thread A remains idle until the thread scheduler reactivates thread A, which is in the waiting or blocked state.

When the main thread invokes the join() method then, it is said that the main thread is in the waiting state. The main thread then waits for the child threads to complete their tasks. When the child threads complete their job, a notification is sent to the main thread, which again moves the thread from waiting to the active state.

If there are a lot of threads in the waiting or blocked state, then it is the duty of the thread scheduler to determine which thread to choose and which one to reject, and the chosen thread is then given the opportunity to run.

**Timed Waiting:** Sometimes, waiting for leads to starvation. For example, a thread (its name is A) has entered the critical section of a code and is not willing to leave that critical section. In such a scenario, another thread (its name is B) has to wait forever, which leads to starvation. To avoid such scenario, a timed waiting state is given to thread B. Thus, thread lies in the waiting state for a specific span of time, and not forever. A real example of timed waiting is when we invoke the sleep() method on a specific thread. The sleep() method puts the thread in the timed wait state. After the time runs out, the thread wakes up and start its execution from when it has left earlier.

**Terminated:** A thread reaches the termination state because of the following reasons:

* When a thread has finished its job, then it exists or terminates normally.
* **Abnormal termination:** It occurs when some unusual events such as an unhandled exception or segmentation fault.

A terminated thread means the thread is no more in the system. In other words, the thread is dead, and there is no way one can respawn (active after kill) the dead thread.