Assignment No 5

Q1 Answer:

An exception in Java is an event that occurs during the execution of a program and interrupts the normal flow of instructions. It represents an abnormal condition or error condition that arises during the execution of the program. When a special event occurs, a known special object is created. It contains information about the exception object's error, such as its type and the state of the process when the error occurred. The exception is then thrown, which means it is passed to a special piece of code called an exception handler. The exception handler can catch the exception and take appropriate actions, such as displaying an error message or recovering from an error.

Q2. Answer :

Exception handling is a mechanism provided by Java to handle and recover from exceptions or errors that occur during execution. It can catch and handle exceptions to prevent the program from stopping abruptly. Exception handling has three main components: try, catch, and finally. The "try" clause is used to bind code that can throw an exception. The "catch" statement is used to catch and handle exceptions. Identify the type of exception it can handle and the actions to take when that exception occurs. The "end" clause is optional and is always used to refer to the execution of some code regardless of whether an exception occurs or not. It is often used for release or cleaning purposes. Using exception handling allows you to handle errors gracefully, provide meaningful error messages to users, and take appropriate actions to recover from exceptional situations.

Q3.Answer:

In Java, exceptions are categorized into three kinds: checked exceptions, unchecked exceptions, and mistakes.

Checked Exceptions: Checked exceptions are exceptions which are checked at collect-time. The compiler guarantees that those exceptions are both stuck and handled or declared to be thrown by means of the technique the usage of the "throws" clause. Examples of checked exceptions in Java consist of IOException, SQLException, and ClassNotFoundException. Checked exceptions are commonly recoverable, and this system need to be capable of handle them gracefully.

Unchecked Exceptions: Unchecked exceptions, also referred to as runtime exceptions, are exceptions that are not checked at collect-time. The compiler does now not enforce any handling of these exceptions. Examples of unchecked exceptions in Java consist of Null Pointer Exception, Array Index Out Of Bounds Exception, and Arithmetic Exception. Unchecked exceptions are commonly resulting from programming errors or logical errors in the code and are frequently considered deadly to the utility.

Errors: Errors are super conditions that aren't expected to be caught or dealt with with the aid of the software. Errors typically get up from the JVM or the underlying system and indicate extreme problems that are unlikely to be recovered. Examples of mistakes include OutOfMemoryError and StackOverflowError. Errors are generally now not stuck or handled explicitly by application code.

Q4Answer;

Throw: The "throw" keyword is used to explicitly throw an exception from a technique or a block of code. It is accompanied by an instance of an exception class or subclass. When a "throw" announcement is encountered, the ordinary waft of this system is interrupted, and the required exception is thrown. It allows you to create and throw custom exceptions or propagate built-in exceptions to the caller of a way.

Throws: The "throws" keyword is used inside the technique statement to signify that the method may also throw one or greater exceptions. It specifies the kind of exceptions that a method would possibly throw, permitting the caller of the method to be privy to the viable exceptions and handle them for this reason. Multiple exceptions may be declared the use of a comma-separated list. When a way broadcasts a checked exception with "throws," it's far a signal to the caller that they want to handle or propagate the exception.

In precis, "throw" is used to throw an exception explicitly within a method or block of code, at the same time as & quot; throws & quot; is used inside the approach statement to suggest the exceptions that is probably thrown by that method.

Q5. Answer:

Multithreading in Java refers back to the concurrent execution of multiple threads inside a unmarried software. A thread is an independent unit of execution that represents a chain of commands. Multithreading lets in distinct parts of a software to execute simultaneously, making it feasible to perform more than one responsibilities concurrently.

Advantages of multithreading in Java include:

Concurrency: Multithreading lets in concurrent execution of duties, permitting higher usage of system resources and stepped forward overall performance. It permits one of a kind components of a application to execute independently, making the program more responsive and efficient.

Responsiveness: Multithreading facilitates in constructing responsive packages by maintaining the user interface (UI) responsive at the same time as appearing time-eating obligations inside the heritage. It prevents the UI from freezing or becoming unresponsive, supplying a smoother user enjoy.

Improved throughput: By using multiple threads, it is feasible to execute more than one responsibilities simultaneously, leading to improved throughput and faster crowning glory of tasks.

Resource sharing: Multithreading permits efficient sharing of sources among multiple threads. Threads can speak and share facts via shared variables or synchronized information structures, enabling coordination and cooperation among special elements of a application.

However, it is vital to note that multithreading additionally introduces demanding situations which includes thread synchronization, race situations, and deadlock, which need to be cautiously treated to ensure correct and dependable application behavior.

Q6Answer:

```
// Custom exception class
class CustomException extends Exception {
    public CustomException(String message) {
        super(message);
    }
}
// Main program
public class CustomExceptionExample {
    public static void checkNumber(int number) throws CustomException {
        if (number < 0) {
            throw new CustomException("Number cannot be negative");
        }
        System.out.println("Number is valid");
    }
    // Main method
    public static void main(String[] args) {
        int number = -5;
        try {
            checkNumber(number);
        } catch (CustomException e)
          {
            System.out.println("Caught CustomException: " + e.getMessage());
        } } }
```

Q7. Answer:

In Java, exceptions can be handled the use of the attempt-capture-finally mechanism. Here's how you may manage exceptions:

try: The code that could throw an exception is enclosed within a attempt block. If an exception takes place within the attempt block, the manipulate straight away transfers to the capture block.

Catch: The trap block is used to catch and manage the exception. It specifies the kind of exception it could take care of and the actions to be taken whilst that exception happens. Multiple trap blocks may be used to handle special kinds of exceptions.

Ultimately: The finally block is optionally available and is used to specify a block of code this is continually achieved, no matter whether an exception happens or now not. It is normally used to release sources or perform clean up tasks that need to be executed no matter an exception being thrown.

```
public class ExceptionHandlingExample {

    public static void main(String[] args) {

        try {

            // Code that might throw an exception

            int result = 10 / 0; // ArithmeticException: division by zero

            System.out.println("Result: " + result); // This line is skipped

        } catch (ArithmeticException e) {


            System.out.println("An error occurred: " + e.getMessage());

        } finally {

            System.out.println("Finally block executed");

        }

    }

}
```

Q8. Answer:

In Java, a thread represents a separate waft of execution inside a software. It is an impartial unit of execution that could concurrently run along different threads in a software. Each thread has its personal name stack, program counter, and local variables.

Threads allow concurrent execution, permitting multiple duties to be performed concurrently. In a Java application, the principle thread is created mechanically whilst this system starts off evolved. Additional threads may be created using the Thread elegance or by imposing the Runnable interface.

Q9. Answer:

Extending the Thread class: This approach involves creating a subclass of the Thread class and overriding its run() method. The run() method contains the code that will be executed when the thread starts. To start the thread, an instance of the child class is created, and the start() method is invoked on that instance.

Implementing the Runnable interface: This approach involves implementing the **Runnable** interface and providing an implementation for the **run()** method. The **run()** method contains the code that will be executed when the thread starts. To start the thread, an instance of the class that implements **Runnable** is created, and that instance is passed to the **Thread** constructor. The **start()** method is then invoked on the **Thread** object.

Q10. Answer:

Garbage collection in Java is the process by which Java programs perform automatic memory management. Java programs compile to byte code that can be run on a Java Virtual Machine, or JVM for short. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory.