# Assignment 5 (Core Java)

**Q1.** What is Exception in Java?

**Sol:-**

In Java, an exception is an event that occurs during the execution of a program and disrupts the normal flow of the program's instructions. It represents an abnormal condition or error situation that may arise during program execution.

==========

**Q2.** What is Exception Handling?

**Sol:-**

Exception handling in Java is a mechanism that allows you to handle and manage runtime errors and exceptional conditions in a controlled and structured manner. It provides a way to gracefully handle exceptional situations, preventing the program from terminating abruptly and allowing you to take appropriate actions when errors occur.

=> Java exception handling revolves around the following key concepts:

**1. Exception:-** An exception is an object that represents an exceptional condition or error that occurs during program execution. Exceptions are derived from the **Throwable** class and can be categorized into two types: checked exceptions and unchecked exceptions.

**2. Try-Catch Blocks:-** The **try-catch** block is used to catch and handle exceptions. The **try** block contains the code that may throw an exception, and the **catch** block(s) specify the type of exception(s) to catch and provide the code to handle the exception. Multiple **catch** blocks can be used to handle different types of exceptions.

**3.Throw Statement:-** The **throw** statement is used to explicitly throw an exception within a method or block of code. It allows you to create and throw custom exceptions or propagate exceptions to the calling method.

**4. Finally Block:-** The **finally** block is used to define a block of code that is always executed, regardless of whether an exception occurs or not. It is commonly used to release resources or perform cleanup operations.

==========

**Q3.** What is the difference between Checked and Unchecked Exceptions and Error?

**Sol:-**

**1.Checked Exceptions:-**

- Checked exceptions are exceptions that must be declared in the method signature or handled explicitly using try-catch blocks.
- These exceptions are subclasses of **Exception** but not subclasses of **RuntimeException**.
- Examples of checked exceptions include **IOException**, **SQLException**, and **ClassNotFoundException**.

**2.Unchecked Exceptions (Runtime Exceptions):-**

- Unchecked exceptions are exceptions that do not need to be explicitly declared or handled using try-catch blocks.
- These exceptions are subclasses of **RuntimeException** or its subclasses.
- Examples of unchecked exceptions include **NullPointerException**, **ArrayIndexOutOfBoundsException**, and **ArithmeticException**.
- Unchecked exceptions are usually caused by programming errors or violations of language semantics.

**3.Errors:-**

- Errors represent exceptional conditions that are usually beyond the control of the programmer and cannot be easily recovered from.
- Errors are typically caused by serious problems that may prevent the program from functioning properly.
- Examples of errors include **OutOfMemoryError**, **StackOverflowError**, and **NoClassDefFoundError**.
- Errors are subclasses of **Error**, which is a subclass of **Throwable**.

==========

**Q4.** What are the difference between throw and throws in Java?

**Sol:-**

**1.throw**:-

- The **throw** keyword is used to explicitly throw an exception from a block of code.
- It is used when you encounter an exceptional condition or error within your code and want to raise an exception to indicate the occurrence of that exceptional condition.
- The **throw** statement is followed by an instance of an exception or a subclass of **Throwable**.
- The exception thrown using **throw** can be caught and handled using try-catch blocks up the call stack.

**2.throws**:-

- The **throws** keyword is used in a method signature to declare that the method may throw one or more types of exceptions.
- It is used to indicate that a method can potentially generate exceptions during its execution that it does not handle internally.
- When a method declares **throws** for a specific exception type, it means that it does not handle that exception itself, but it delegates the responsibility of handling to the calling method or the caller of the method.
- The caller of the method must either handle the declared exceptions using try-catch blocks or propagate the exceptions further up the call stack by declaring them with **throws** in their own method signature.

==========

**Q5.** What is multithreading in Java? mention its advantages

**Sol:-**

Multithreading in Java refers to the concurrent execution of multiple threads within a single program. A thread is an independent path of execution within a program, allowing multiple tasks to be performed simultaneously.

**Advantages of multithreading in Java include:-**

- **1.Concurrency:-** Multithreading allows concurrent execution of tasks, enabling multiple operations or computations to run simultaneously. This can lead to improved performance and efficiency by utilizing available system resources effectively.

- **2.Responsiveness:-** Multithreading enables the application to remain responsive and interactive while executing time-consuming operations in the background. For example, a user interface can remain responsive even when performing heavy computations or network operations concurrently.

- **3.Enhanced throughput:-** By dividing a task into multiple threads, each performing a subtask, the overall throughput of the system can be increased. This is especially beneficial in scenarios where the execution of one task depends on the completion of another.

- **4.Resource utilization:-** Multithreading allows efficient utilization of system resources, such as CPU cores and memory. By distributing tasks across multiple threads, the workload can be balanced, and resources can be utilized optimally.

- **5.Asynchronous programming:-** Multithreading enables asynchronous programming, where tasks can execute independently, and the program can proceed with other operations while waiting for the completion of asynchronous tasks. This is useful in scenarios where time-consuming operations, such as I/O or network requests, are involved.

- **6.Modularity and code organization:-** Multithreading allows you to structure your code into smaller, more manageable units. Different threads can handle different aspects of the program, promoting modularity and code organization.

- **7.Real-time applications:-** Multithreading is crucial for developing real-time applications, where multiple tasks need to be executed concurrently with strict timing requirements. Examples include multimedia applications, gaming, and simulation systems.

==========

**Q6.** Write a program to create and call a custom exception

---

**Sol:-**

```
class MyException extends Exception {
    public MyException(String message) {
        super(message);
```

```
    }
}

public class example {
    public static void main(String[] args) {

        try {
            int age = 19;
            if (age < 18) {
                throw new MyException("you must be at least 18 years old");
            } else {
                System.out.println("Access granted. you are eligible");
            }
        } catch (MyException e) {
            System.out.println("Exceptioncaught: " + e.getMessage());
        }
    }
}
```

==========

**Q7.** How can you handle exceptions in Java?

**Sol:-**
In Java, exceptions can be handled using try-catch blocks or by declaring them to be thrown using the **throws** keyword.

**1.Try-Catch Blocks:-**

- The **try-catch** block is used to catch and handle exceptions that may occur during the execution of a block of code.
- The **try** block contains the code that may throw an exception, and the **catch** block(s) specify the type of exception(s) to catch and provide the code to handle the exception.
- Multiple **catch** blocks can be used to handle different types of exceptions.
- The **finally** block, if present, is executed regardless of whether an exception occurs or not. It is typically used for releasing resources or performing cleanup operations.

**2.Declaring Exceptions with throws:-**

- In Java, it is possible to declare exceptions to be thrown by a method using the **throws** keyword in the method signature.
- When a method declares **throws** for a specific exception type, it means that the method does not handle that exception itself but expects the calling code to handle it.
- The caller of the method must either handle the declared exceptions using try-catch blocks or propagate the exceptions further up the call stack by declaring them with **throws** in their own method signature.

==========

**Q8.** What is Thread in Java?

**Sol:-**

In Java, a thread refers to an independent path of execution within a program. It allows concurrent execution of multiple tasks or operations within the same program. Threads are lightweight and provide a way to achieve multitasking and concurrent programming.

==========

**Q9.** What are the two ways of implementing thread in Java?

**Sol:-**

**1.Extending the Thread class:-**

- In this approach, you can create a new class that extends the **Thread** class and override its **run()** method to define the behavior of the thread.
- The **run()** method contains the code that will be executed when the thread starts.
- To start the thread, you can create an instance of your custom thread class and call its **start()** method.

**2.Implementing the Runnable interface:-**

- In this approach, you can create a class that implements the **Runnable** interface and implement its **run()** method.
- The **run()** method contains the code that will be executed when the thread starts.
- To start the thread, you need to create an instance of your class that implements **Runnable** and pass it as a parameter to a **Thread** object. Then, call the **start()** method on the **Thread** object.

==========

**Q10.** What do you mean by garbage collection?

**Sol:-**

- Garbage collection in Java refers to the automatic memory management process where the Java Virtual Machine (JVM) automatically reclaims memory that is no longer in use by the program. It frees up memory occupied by objects that are no longer referenced by any active part of the program.

- In Java, objects are dynamically allocated on the heap memory using the **new** keyword. However, the programmer does not need to explicitly deallocate or free the memory occupied by objects. The JVM automatically handles the deallocation process through the garbage collector.

- The garbage collector operates in the background, identifying objects that are no longer reachable or referenced by any part of the program. It then reclaims the memory occupied by these unreachable objects, making it available for future allocations.