**Task 11 – Java Exceptions and Error Handling**

Q1: Four Access Modifiers in Java

**Public:**

Class: A public class can be accessed from any other class in the same project or package, as well as from classes in other packages.

Method: Public methods can be called from any other class, regardless of its location. They are part of the class's public interface and are meant to be accessible to all classes.

Variable: Public variables can be accessed directly from any class. However, it's often considered a better practice to encapsulate data using private variables and providing public methods (getters and setters) to access and modify them.

**Protected:**

Class: Java doesn't allow class-level access control with the protected modifier. Classes can only be declared as public or default (package-private).

Method: Protected methods are accessible within their own package and by subclasses, whether they are in the same package or not. This facilitates method overriding in subclasses.

Variable: Protected variables can be accessed by classes within the same package and by subclasses, whether they are in the same package or not.

**Default (no Modifier- Private Package)**

Class: If no access modifier is specified (i.e., default access), the class is accessible only within its own package. It's not visible to classes in other packages, even if they are subclasses.

Method: Default methods are accessible only within the same package. They cannot be accessed from classes outside the package, even if they are subclasses.

Variable: Default variables can also be accessed only within the same package. They are not visible to classes outside the package, including subclasses in other packages.

**Private:**

Class: Java doesn't allow class-level access control with the private modifier. Classes can only be declared as public or default (package-private).

Method: Private methods are accessible only within the same class. They cannot be accessed or overridden by subclasses or other classes.

Variable: Private variables are accessible only within the same class. They cannot be accessed from outside the class, including subclasses. Encapsulation is often achieved using private variables to control access to data and prevent unintended modifications.

Q2: **Difference between Exceptions and Error**

**Exceptions:**

Exceptions represent exceptional conditions that occur during the execution of a program. These conditions might be recoverable, and the program can usually continue its execution after handling the exception.

Exceptions in Java are further divided into checked exceptions and unchecked exceptions. Checked exceptions are those that must be caught or declared by the method, while unchecked exceptions (runtime exceptions) do not need to be caught or declared.

Examples of exceptions include FileNotFoundException, NullPointerException, and ArrayIndexOutOfBoundsException.

Exceptions are intended to handle abnormal conditions in a program's execution flow, such as invalid user input, file not found, or division by zero.

**Errors:**

Errors, on the other hand, represent serious, unrecoverable problems that typically cannot be handled by the program itself. They often indicate problems with the Java runtime environment or underlying system.

Unlike exceptions, errors are not meant to be caught or handled by application code. Attempting to handle errors may not be effective, and it's generally better to allow the program to terminate and log the error for investigation.

Examples of errors include OutOfMemoryError, StackOverflowError, and VirtualMachineError.

Errors are typically caused by issues outside the control of the program, such as running out of memory, hardware failures, or JVM crashes.

Q3**: Difference between Checked Exceptions and Unchecked Exceptions**

The key difference between checked exceptions and unchecked exceptions lies in how they are handled by the compiler and the programmer. Checked exceptions must be explicitly handled or declared, while unchecked exceptions do not need to be. Checked exceptions are typically used for recoverable error conditions, while unchecked exceptions are often used for programming errors or exceptional conditions that are beyond the programmer's control.

**Checked Exceptions:**

* Checked exceptions are exceptions that must be either caught or declared in the method signature using the throws keyword.
* They are typically used to represent anticipated error conditions that a method might encounter and that the caller should handle.
* Checked exceptions are subclasses of java.lang.Exception, excluding subclasses of RuntimeException and its subclasses.
* Examples of checked exceptions include IOException, SQLException, and ClassNotFoundException.
* The compiler enforces handling or declaring of checked exceptions, which means that the programmer must either catch the checked exception using a try-catch block or declare it in the method signature.

**Unchecked Exceptions (Runtime Exceptions):**

* Unchecked exceptions are exceptions that do not need to be caught or declared in the method signature.
* They are typically used to represent programming errors or conditions that the programmer might overlook or cannot reasonably anticipate.
* Unchecked exceptions are subclasses of java.lang.RuntimeException and its subclasses.
* Examples of unchecked exceptions include NullPointerException, ArrayIndexOutOfBoundsException, and NumberFormatException.
* The compiler does not enforce handling or declaring of unchecked exceptions. Programmers are not required to catch or declare unchecked exceptions, although they still can if desired.