**TASK - 11**

**1) What are the access modifiers available in java and what is their significance in terms of class, method and variable access ability?**

In Java, access modifiers are keywords used to define the accessibility or visibility of classes, methods, and variables in different parts of a program. There are four main access modifiers in Java, each with its significance:

They are,

1. Public
2. Private
3. Protected
4. Default

1. Public:

- Class: A public class is accessible from any other class. It acts as an entry point for the execution of the program.

- Method: A public method can be called from any other class.

- Variable: A public variable is accessible from any other class.

Example:

public class ExampleClass {

public void publicMethod() {

// Code here

}

public int publicVariable = 10;

}

2. Private:

- Class: A private class cannot be accessed from outside the class it is defined in.

- Method: A private method can only be called within the same class. It is not visible to external classes.

- Variable: A private variable is only accessible within the class it is defined in.

Example:

public class ExampleClass {

private void privateMethod() {

// Code here

}

private int privateVariable = 20;

}

3. Protected:

- Class: A protected class cannot be accessed from outside the package, except by subclasses.

- Method: A protected method can be accessed within the same package or by subclasses, regardless of the package.

- Variable: A protected variable has the same accessibility rules as a protected method.

Example:

public class ExampleClass {

protected void protectedMethod() {

// Code here

}

protected int protectedVariable = 30;

}

4.Default (Package-Private):

- Class: If no access modifier is specified, it is considered "package-private." The class is accessible only within the same package.

- Method: A default (package-private) method can be accessed within the same package but not outside it.

- Variable: A default (package-private) variable has the same accessibility rules as a default method.

Example:

class ExampleClass {

void defaultMethod() {

// Code here

}

int defaultVariable = 40;

}

Significance of Access Modifiers:

- Encapsulation: Access modifiers help in encapsulating the implementation details, exposing only what is necessary.

- Code Organization: They provide a way to organize and control the visibility of classes, methods, and variables within a program.

- Security: By restricting access, they contribute to the security and integrity of the codebase.

- Inheritance: Access modifiers play a crucial role in inheritance, allowing controlled access to superclass members in subclasses.

Choosing the right access modifier depends on the design principles, encapsulation needs, and the desired level of abstraction for the code.

**2) What is the difference between exception and error?**

In Java, both exceptions and errors are subclasses of the `Throwable` class, but they serve different purposes and are meant to handle distinct types of issues in a program.

**Exception:**

It’s a run time error.

1. Definition:

- An exception is an abnormal event or condition that occurs during the execution of a program but is generally expected to be handled by the program itself.

- Exceptions can occur due to various reasons, such as user input errors, network issues, file I/O problems, or unexpected conditions in the program logic.

2. Handling:

- Exceptions are typically caught and handled using try-catch blocks. The idea is to gracefully handle exceptional situations to prevent the program from terminating abruptly.

3. Types:

- Exceptions in Java are further divided into two main categories: checked exceptions and unchecked exceptions.

- Checked Exceptions: These are exceptions that are checked at compile-time. The programmer is required to handle these exceptions explicitly using try-catch blocks or declare them in the method signature using the `throws` keyword.

- Unchecked Exceptions: These are exceptions that are not checked at compile-time. They usually indicate programming errors or runtime issues and extend from `RuntimeException` or its subclasses.

Example:

try {

// Code that may throw an exception

} catch (ExceptionType e) {

// Handle the exception

} finally {

// Code that runs whether an exception occurs or not

}

**Error:**

1. Definition:

- An error is an abnormal event or condition that occurs during the execution of a program and is typically beyond the control of the program itself.

- Errors usually represent severe issues that can lead to the termination of the program. They are often caused by external factors or problems at the system level.

2. Handling:

- Unlike exceptions, errors are not meant to be caught and handled by the program. They are usually unrecoverable, and attempting to handle them might not be practical or meaningful.

3. Types:

- Examples of errors include `OutOfMemoryError`, `StackOverflowError`, and `LinkageError`. These issues are often related to resource exhaustion, system failures, or problems with the Java Virtual Machine (JVM).

// Errors are typically not caught and handled

// They are often indicative of critical issues that should be addressed at a system level

**Key Differences:**

1. Cause:

- Exception: Typically caused by issues that a well-designed program can anticipate and handle.

- Error: Generally caused by factors external to the program, such as system failures or resource limitations.

2. Handling:

- Exception: Meant to be caught and handled by the program using try-catch blocks.

- Error: Not intended to be caught and handled programmatically. Often indicates a severe issue that may require system-level intervention.

3. Examples:

- Exception: `IOException`, `NullPointerException`, `ArrayIndexOutOfBoundsException`.

- Error: `OutOfMemoryError`, `StackOverflowError`, `LinkageError`.

In summary, exceptions are meant to handle recoverable issues within the program, while errors indicate severe problems that may lead to the termination of the program and are often beyond the program's control.

**3) What is the difference between checked exception and unchecked exception?**

In Java, exceptions are broadly categorized into two types: checked exceptions and unchecked exceptions (also known as runtime exceptions). These categories define how the compiler enforces handling or declaration of exceptions in a program.

**Checked Exceptions:**

1. Definition:

- Checked exceptions are exceptions that are checked at compile-time.

- They extend the `Exception` class (directly or indirectly) but not the `RuntimeException` class.

**2. Handling:**

- Checked exceptions must be either caught and handled using try-catch blocks or declared in the method signature using the `throws` clause.

- If a method can potentially throw a checked exception, the programmer is obligated to either handle it within the method or declare it in the method signature.

**3. Examples:**

- `IOException`, `FileNotFoundException`, `SQLException`.

**4. Syntax:**

- Handling checked exceptions using try-catch:

Example

try {

// Code that may throw a checked exception

} catch (ExceptionType e) {

// Handle the exception

}

- Declaring checked exceptions in method signature:

public void exampleMethod() throws IOException {

// Code that may throw an IOException

}

**Unchecked Exceptions (Runtime Exceptions):**

1. Definition:

- Unchecked exceptions are exceptions that are not checked at compile-time.

- They extend the `RuntimeException` class (directly or indirectly).

2. Handling:

- Unchecked exceptions are not required to be caught or declared. The compiler does not enforce handling or declaration for unchecked exceptions.

3. Examples:

- `NullPointerException`, `ArrayIndexOutOfBoundsException`, `ArithmeticException`.

4. Syntax:

- Handling unchecked exceptions using try-catch:

Example:

try {

// Code that may throw an unchecked exception

} catch (RuntimeException e) {

// Handle the exception

}

- Unchecked exceptions are not declared in the method signature:

public void exampleMethod() {

// Code that may throw an unchecked exception

}

**Key Differences:**

1. Checked Exceptions

- Checked at compile-time.

- Must be either caught and handled or declared in the method signature.

- Extend the `Exception` class (excluding subclasses of `RuntimeException`).

2. Unchecked Exceptions (Runtime Exceptions):

- Not checked at compile-time.

- Handling or declaration is optional.

- Extend the `RuntimeException` class (or its subclasses).

3. Examples:

- Checked Exceptions: `IOException`, `SQLException`.

- Unchecked Exceptions: `NullPointerException`, `ArrayIndexOutOfBoundsException`.

4. Usage:

- Checked exceptions are typically used for scenarios where the program can anticipate and recover from the exceptional condition.

- Unchecked exceptions are often used for programming errors or runtime issues that may be difficult to predict and handle explicitly.

In practice, checked exceptions are used for situations where recovery is possible and the programmer needs to be aware of potential issues, while unchecked exceptions are used for scenarios where recovery may be challenging, and the emphasis is on fixing the code.