**Task — 11**

**Ans 1**

**Public:**

Classes, methods, and variables with the public access modifier are accessible from any other class or package. They have the widest scope of accessibility.

Classes with the public access modifier can be accessed by any other class in the same project or in other projects.

Methods and variables with the public access modifier can be called or accessed from any other class or package.

**Protected:**

Classes, methods, and variables with the protected access modifier are accessible within the same package or by subclasses (even if they are in different packages).

They are not accessible by classes in other packages unless they are subclasses of the class containing the protected member.

This modifier is often used for variables and methods that need to be accessed by subclasses but not by classes outside the package.

**Default (no modifier):**

Classes, methods, and variables with no explicit access modifier (i.e., default access) are accessible within the same package.

They are not accessible by classes outside the package.

This is the default level of access if no access modifier is specified.

It provides a balance between encapsulation and visibility, limiting access to package-private members.

**Private:**

Classes, methods, and variables with the private access modifier are accessible only within the same class.

They are not accessible by any other class, even subclasses.

This modifier provides the highest level of encapsulation and data hiding, restricting access to only the class in which the member is defined.

It is commonly used to hide implementation details and prevent direct access to internal state.

**Ans-2**

**Exception:**

Exceptions represent exceptional conditions that can occur during the normal execution of a program.

They are generally caused by user input errors, incorrect logic, or resource availability issues (e.g., file not found, network failure).

Exceptions are expected to be caught and handled by the program using try-catch blocks or propagated up through the call stack using throws declarations.

Examples of exceptions include IOException, NullPointerException, ArrayIndexOutOfBoundsException, NumberFormatException, etc.

Exceptions are checked at compile-time, which means that methods must declare all checked exceptions that they might throw using the throws clause, or handle them within a try-catch block.

**Error:**

Errors represent serious, often unrecoverable problems that occur within the JVM or its environment.

They are typically caused by system-level issues, such as out-of-memory errors, stack overflow, hardware failures, or JVM internal errors.

Errors are not meant to be caught and handled by the application code because they usually indicate critical problems that cannot be recovered from.

Attempting to catch or handle errors is generally not recommended, as it may lead to unpredictable behaviour or make the application unstable.

Examples of errors include OutOfMemoryError, StackOverflowError, NoClassDefFoundError, VirtualMachineError, etc.

Errors are unchecked, which means that the compiler does not enforce any handling or declaration requirements for them. However, application code should focus on preventing errors rather than handling them.

**Ans-3**

**Unchecked Exception**

It will occur at the run time.  
**Checked Exception**:

Checked exception will occur at the Compile time