# **Java Exception and error handling**

1. **What are the four access modifiers available in Java and what is their significance in terms of class, method, and variable accessibility?**

In Java, there are four access modifiers:

1. **public**: This modifier allows access from any other class or package. Classes, methods, and variables marked as public are visible and accessible everywhere.
2. **protected**: Protected members are accessible within the same package and by subclasses, even if they are in a different package. However, they are not accessible by non-subclass classes outside the package.
3. **default** (no modifier): When no access modifier is specified, the member has package-level visibility. It is accessible within the same package but not from outside.
4. **private**: Private members are accessible only within the class they are declared in. They are not visible or accessible from any other class, even subclasses.

Here's a breakdown of their significance in terms of class, method, and variable accessibility:

* Class:
  + **public** classes can be accessed from anywhere.
  + **default** and **protected** classes can be accessed within the same package or by subclasses.
  + **private** classes are not allowed at the top level (i.e., as top-level classes, they can only be declared with the **public**, **abstract**, or **final** modifiers).
* Method:
  + **public** methods can be called from anywhere.
  + **protected** methods can be called within the same package or by subclasses.
  + **default** methods can be called within the same package.
  + **private** methods are only accessible within the class they are defined in.
* Variable (field):
  + **public** variables can be accessed from anywhere.
  + **protected** variables can be accessed within the same package or by subclasses.
  + **default** variables can be accessed within the same package.
  + **private** variables are only accessible within the class they are defined in.

These access modifiers help control the visibility and accessibility of classes, methods, and variables in Java, allowing you to define the desired level of encapsulation and access control for your code.

1. **What is the difference between exception and error?**

In Java (and in programming in general), exceptions and errors are two different types of throwable objects, and they serve distinct purposes. Here's the key difference between them:

1. **Exceptions**:
   * Exceptions are events that occur during the execution of a program that disrupts the normal flow of the program.
   * Exceptions are typically caused by conditions that a programmer can anticipate and handle, such as invalid input, file not found, or network connection issues.
   * Exceptions are represented by classes that inherit from the **java.lang.Exception** class or its subclasses.
   * Programmers are encouraged to catch and handle exceptions using try-catch blocks to gracefully handle unexpected situations, recover from errors, or display meaningful error messages to users.
   * Examples of exceptions in Java include **NullPointerException**, **IOException**, **ArrayIndexOutOfBoundsException**, and **NumberFormatException**.
2. **Errors**:
   * Errors, on the other hand, are unexpected and usually indicate serious problems that are not recoverable at runtime.
   * Errors typically occur due to problems with the environment or the JVM itself, and they are not meant to be caught or handled by application code.
   * Errors are represented by classes that inherit from the **java.lang.Error** class or its subclasses.
   * Examples of errors in Java include **OutOfMemoryError**, **StackOverflowError**, and **VirtualMachineError**.
   * Errors are often caused by issues like running out of system resources (e.g., memory), hardware failures, or other critical issues beyond the control of the application.

In summary, exceptions are used to handle expected and recoverable problems in a program's execution, while errors represent unexpected and typically unrecoverable issues that often require intervention at the system or JVM level. It's important for developers to differentiate between exceptions and errors and handle them appropriately in their code to ensure robust and reliable software.

1. **what is the difference between checked exception and unchecked exception?**

In Java, exceptions are categorized into two main types: checked exceptions and unchecked exceptions (also known as runtime exceptions). The primary difference between them lies in when they are checked by the Java compiler and whether or not they require explicit handling in your code:

1. **Checked Exceptions**:
   * Checked exceptions are exceptions that the Java compiler mandates you to handle explicitly. This means you must either catch them using a **try-catch** block or declare that your method can throw them using the **throws** keyword.
   * Checked exceptions are typically used for situations that are external to your program and could occur, but you can reasonably anticipate and recover from them. These often include I/O-related issues, such as file not found, database connection problems, or network-related errors.
   * Checked exceptions are subclasses of **java.lang.Exception** (excluding **RuntimeException** and its subclasses).

***Example of a checked exception*:**

**try{**

**//code that may throw the checked exception**

**}catch(IO exception e){**

**//handle the exception**

**}**

1. **Unchecked Exceptions (Runtime Exceptions)**:
   * Unchecked exceptions are exceptions that the compiler does not require you to handle explicitly. You can catch them if you want, but it's not mandatory.
   * Unchecked exceptions are typically used for programming errors or issues that should not occur in well-written code, such as null pointer dereferences, array index out of bounds, or arithmetic exceptions (e.g., division by zero).
   * Unchecked exceptions are subclasses of **java.lang.RuntimeException**.

***Example of an unchecked exception***:

**Int[] array = {1,2,3,4}**

**Int result = array[5]; //this will throw ArrayIndexOutofBoundsException at runtime.**

In summary, the key difference is that checked exceptions are enforced by the compiler and must be handled explicitly, whereas unchecked exceptions (runtime exceptions) are not required to be handled, although it's a good practice to handle or prevent them when possible. Checked exceptions are typically used for recoverable external issues, while unchecked exceptions are used for programming errors or unexpected situations that indicate a bug in the code.