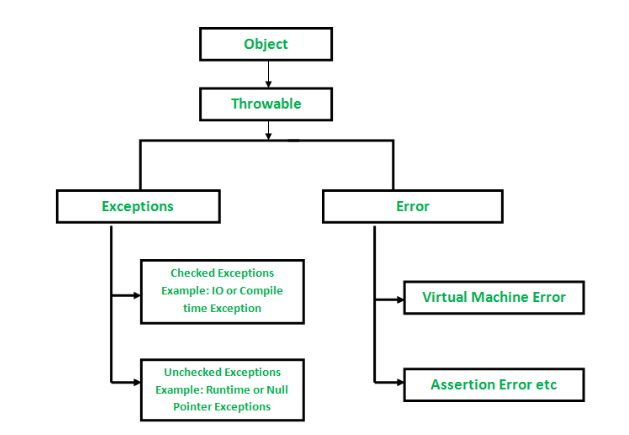
**1) What are the four access modifiers available in Java and what is their significance in terms of class, method, and variable accessibility?**

In Java, there are four access modifiers that control the visibility and accessibility of classes, methods, and variables within a program. These access modifiers specify the level of access that other classes or code components have to the elements they modify. Here are the four access modifiers and their significance:

* **Public:**
  + Significance: The public access modifier makes the class, method, or variable accessible from any other class or package.
  + Class: The public class can be accessed from any other class or package.
  + Method: The public method can be called from any other class or package.
  + Variable: The public variable can be accessed and modified from any other class or package.
* **Protected:**
  + Significance: The Protected access modifier restricts access to the class, method, or variable to its own package and subclasses (both in the same package and in subclasses in different packages).
  + Class: Classes cannot be declared with the protected access modifier. It only applies to methods and variables.
  + Method: A protected method can be accessed by classes within the same package or by subclasses, regardless of whether they are in the same package or a different package.
  + Variable: A protected variable can be accessed by classes within the same package or by subclasses, regardless of whether they are in the same package or a different package.
* **Default (Package-Private):**
  + Significance: The default access modifier (also known as package-private) restricts access to the class, method, or variable to only classes within the same package.
  + Class: If no access modifier is specified (i.e., no public, protected or private), the class has default access.
  + Method: A method with default access can be accessed only by classes within the same package.
  + Variable: A variable with default access can be accessed only by classes within the same package.
* **Private:**
  + Significance: The private access modifier restricts access to the class, method, or variable to only within the same class.
  + Class: Classes cannot be declared with the private access modifier. It only applies to methods and variables.
  + Method: A private method can be accessed and invoked only within the same class.
  + Variable: A private variable can be accessed and modified only within the same class.

**2) What is the difference between the Exception and Error ?**

In Java, errors and exceptions are both types of throwable objects, but they represent different types of problems that can occur during the execution of a program.



**Exceptions:**

* + Exceptions represent exceptional conditions that can be caught and handled by the application.
  + Exceptions can occur due to various reasons, such as invalid user input, network issues, or file I/O errors.
  + Exceptions are represented by the Exception class and its subclasses. Some common examples of exceptions in Java include:
  + **NullPointerException**: Thrown when a null reference is accessed.
  + **IllegalArgumentException:** Thrown when an illegal argument is passed to a method.
  + **IOException:** Thrown when an I/O operation fails.
  + Exceptions are checked or unchecked:
    - Checked exceptions must be declared in the method signature or handled using try-catch blocks.
    - Unchecked exceptions (also known as runtime exceptions) do not need to be declared or caught explicitly, but they can still be caught if desired.

Since exceptions can be caught and handled within a program, it’s common to include code to catch and handle exceptions in Java programs. By handling exceptions, you can provide more informative error messages to users and prevent the program from crashing.

**Errors:**

* Errors are usually caused by serious problems that are outside the control of the program, such as running out of memory or a system crash.
* Errors are typically not caught or handled by the application because they indicate fundamental problems with the execution environment or system.
* Errors are unchecked and usually indicate irrecoverable issues that may require intervention at a higher level, such as **system administrators or developers.**
* Errors are represented by the Error class and its subclasses. Some common examples of errors in Java include:
  + **OutOfMemoryError:** Thrown when the Java Virtual Machine (JVM) runs out of memory.
  + **StackOverflowError:** Thrown when the call stack overflows due to too many method invocations.
  + **NoClassDefFoundError:** Thrown when a required class cannot be found.

Since errors are generally caused by problems that cannot be recovered from, it’s usually not appropriate for a program to catch errors. Instead, the best course of action is usually to log the error and exit the program.

while both exceptions and errors represent abnormal conditions in Java programs, exceptions are meant to be caught and handled within the application's code, whereas errors are typically not recoverable and may require external intervention.

Handling exceptions allows for graceful error recovery and program execution, while errors indicate severe problems that may require system-level attention.

| **Errors** | **Exceptions** |
| --- | --- |
| Recovering from Error is not possible. | We can recover from exceptions by either using try-catch block or throwing exceptions back to the caller. |
| All errors in java are unchecked type. | Exceptions include both checked as well as unchecked type. |
| Errors are mostly caused by the environment in which program is running. | Program itself is responsible for causing exceptions. |
| Errors can occur at compile time. | Unchecked exceptions occurs at runtime whereas checked exceptions occurs at compile time |
| They are defined in java.lang.Error package. | They are defined in java.lang.Exception package |
| Examples : java.lang.StackOverflowError, java.lang.OutOfMemoryError | Examples : Checked Exceptions : SQLException, IOException Unchecked Exceptions : ArrayIndexOutOfBoundException, NullPointerException, ArithmeticException. |

**3.What is the difference between checked Exception and unchecked Exception?**

In Java, exceptions are categorized into two main types: checked exceptions and unchecked exceptions. Here's the difference between them:

**Checked Exceptions:**

* + Checked exceptions are exceptions that the compiler requires you to handle, either by catching them in a **try-catch block** or by declaring them in the method signature using the **throws clause.**
  + Checked exceptions represent conditions that are expected to occur during the normal operation of the program and that the programmer should anticipate and handle.
  + In checked exceptions, there are two types

fully checked and partially checked exceptions.

Fully checked exception is a checked exception where all its child classes are also checked, like IOException, and InterruptedException.

Partially checked exception is a checked exception where some of its child classes are unchecked, like an Exception.

* + Examples of checked exceptions include IO Exception and FileNotFoundException etc.
  + If a method throws a checked exception, the calling method must either catch the exception or declare it in its own throws clause.

**Unchecked Exceptions (Runtime Exceptions):**

* + Unchecked exceptions, also known as runtime exceptions, are exceptions that do not need to be declared in the method signature or caught explicitly using a try-catch block.
  + Unchecked exceptions typically represent programming errors or logic errors that occur at runtime, such as dividing by zero, accessing an array index out of bounds, or dereferencing a null pointer.

### In Java, exceptions under **Error and RuntimeException** classes are unchecked exceptions, everything else under **throwable** is checked.

* + Examples of unchecked exceptions include NullPointerException, ArithmeticException, ArrayIndexOutOfBoundsException, etc.
  + Unchecked exceptions are not required to be caught or declared, but they can still be caught and handled if desired.
  + Unchecked exceptions extend the **Run time Exception class or one of its subclasses.**

The main difference between checked and unchecked exceptions lies in how they are handled by the compiler and the programmer. Checked exceptions must be declared or caught, while unchecked exceptions do not need to be declared or caught explicitly.

Checked exceptions are typically used for recoverable conditions, while unchecked exceptions are usually indicative of programming errors or unexpected runtime conditions.