*In Java, there are four access modifiers:*

*1. \*\*Public:\*\* Public access modifier makes a class, method, or variable accessible from any other class. This means that there are no restrictions on accessing the class, method, or variable from anywhere within the same package or from any other package.*

*2. \*\*Private:\*\* Private access modifier restricts the accessibility of a class, method, or variable to only within the same class. It cannot be accessed from outside the class, not even from subclasses.*

*3. \*\*Protected:\*\* Protected access modifier allows the class, method, or variable to be accessible within the same package or by subclasses in different packages. However, it is not accessible by non-subclasses outside the package.*

*4. \*\*Default (Package-private):\*\* If no access modifier is specified, then by default, the class, method, or variable has package-private access. This means it is accessible only within the same package and not from outside the package.*

*The significance of these access modifiers in terms of class, method, and variable accessibility can be summarized as follows:*

*- \*\*Class Accessibility\*\*:*

*- `public`: Accessible from anywhere.*

*- `protected`, default, and `private`: Accessible only within the same package.*

*- \*\*Method Accessibility\*\*:*

*- `public`: Accessible from anywhere.*

*- `protected`: Accessible within the same package and by subclasses, regardless of the package.*

*- Default (no modifier): Accessible within the same package.*

*- `private`: Accessible only within the same class.*

*- \*\*Variable Accessibility\*\*:*

*- `public`, `protected`, default (no modifier): Same accessibility rules as methods apply.*

*- `private`: Accessible only within the same class.*

*Here's a example to illustrate:*

*```java*

*package com.example;*

*public class MyClass {*

*public int publicVar;*

*protected int protectedVar;*

*int defaultVar; // package-private*

*private int privateVar;*

*public void publicMethod() {}*

*protected void protectedMethod() {}*

*void defaultMethod() {} // package-private*

*private void privateMethod() {}*

*}*

*```*

*In this example, `publicVar` and `publicMethod()` can be accessed from anywhere, `protectedVar` and `protectedMethod()` can be accessed within the same package or by subclasses, `defaultVar` and `defaultMethod()` can be accessed only within the same package, and `privateVar` and `privateMethod()` are accessible only within the `MyClass` class.*

*2.Difference between Exception and Error.*

*In Java, both exceptions and errors are subclasses of the `Throwable` class, but they serve different purposes and are handled differently in a program.*

*1. \*\*Exception\*\*:*

*- Exceptions represent exceptional conditions that can occur during the normal execution of a program. They are typically caused by external factors that are beyond the control of the program, such as incorrect user input, network issues, or file I/O errors.*

*- Exceptions are divided into two categories:*

*1. checked exceptions*

*2. unchecked exceptions (also known as runtime exceptions).*

*- Checked exceptions are checked at compile-time, and the compiler forces you to handle them using try-catch blocks or by declaring them in the method signature using the `throws` keyword.*

*- Unchecked exceptions are not checked at compile-time and typically represent programming errors, such as `NullPointerException` or `ArrayIndexOutOfBoundsException`.*

*2. \*\*Error\*\*:*

*- Errors, on the other hand, represent serious problems that are typically beyond the control of the application. They are often caused by system-level issues or by the Java Virtual Machine (JVM) itself. Examples include `OutOfMemoryError`, `StackOverflowError`, and `VirtualMachineError`.*

*- Unlike exceptions, errors are not meant to be caught or handled by the application. Instead, they indicate severe problems that usually require intervention at the system level, such as increasing system resources or fixing configuration issues.*

*- Catching errors is generally not recommended because they typically indicate unrecoverable problems. Attempting to catch errors can lead to unpredictable behavior and may not effectively resolve the underlying issue.*

*In summary, exceptions represent recoverable errors that can be handled by the application, while errors represent severe problems typically beyond the control of the application and are not meant to be caught or handled programmatically.*

*3.Difference between checked and unchecked exception*

*Checked exceptions and unchecked exceptions are two types of exceptions in Java, distinguished by how they are handled by the Java compiler and the JVM. Here are the main differences between them:*

*1. \*\*Checked Exceptions\*\*:*

*- Checked exceptions are checked at compile-time, and the compiler forces you to handle them using try-catch blocks or by declaring them in the method signature using the `throws` keyword.*

*- Typically, checked exceptions represent conditions that a well-behaved application should anticipate and handle gracefully. Examples include `IOException`, `SQLException`, and `ClassNotFoundException`.*

*- We have to handle checked exceptions explicitly, either by catching them or by declaring them in the method signature. This ensures that exceptions are handled appropriately and prevents them from being ignored.*

*- Failure to handle checked exceptions results in a compilation error.*

*2. \*\*Unchecked Exceptions\*\* (Runtime Exceptions):*

*- Unchecked exceptions, also known as runtime exceptions, are not checked by the compiler at compile-time. Instead, they occur at runtime and are typically caused by programming errors or unexpected conditions that may arise during program execution.*

*- Unchecked exceptions do not need to be explicitly caught or declared using `try-catch` blocks or the `throws` keyword. Programmers have the option to handle them, but it's not mandatory.*

*- Examples of unchecked exceptions include `NullPointerException`, `ArrayIndexOutOfBoundsException`, and `IllegalArgumentException`.*

*- Since unchecked exceptions are not checked by the compiler, failure to handle them does not result in a compilation error. However, if not handled, they propagate up the call stack until caught or until they cause the program to terminate.*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*