**Q1. what are the four access modifiers available in java and what is their significance in terms of class , methods and variable accessibility?**

1. Public: This access modifier allows unrestricted access to the class, method, or variable from any other class or package. Classes, methods, and variables declared as public can be accessed from anywhere in the Java program.
2. Protected: The protected access modifier allows access to the class, method, or variable within the same package or by subclasses in other packages. This means that if a class, method, or variable is declared as protected, it can be accessed by classes in the same package or by subclasses (regardless of the package) of that class.
3. Default (no modifier): If no access modifier is specified, the default access modifier is applied. This means that the class, method, or variable is accessible only within the same package. It is not accessible outside the package.
4. Private: The private access modifier restricts access to the class, method, or variable to only within the same class. It is not accessible from outside the class, including subclasses.

* Public allows access from anywhere.
* Protected allows access from the same package or subclasses.
* Default allows access from the same package only.
* Private allows access only from within the same class.

These access modifiers play a crucial role in encapsulation, which is a fundamental principle in object-oriented programming. They help control the visibility and accessibility of classes, methods, and variables, thereby promoting better code organization, security, and reusability.

**Q2. what is the difference between exception and error?**

Exceptions and errors are both types of exceptional conditions that can occur during the execution of a program in Java, but they serve different purposes and are handled differently.

1. **Exceptions:**  
 —  > Exceptions are events that occur during the execution of a program that disrupt the normal flow of the program’s instructions.  
 —  > They are typically caused by conditions that are recoverable, such as invalid user input, network problems, or file not found.  
 — >  Exceptions in Java are represented by classes that extend either `Exception` (checked exceptions) or `RuntimeException` (unchecked exceptions).  
 —  > Checked exceptions must be caught or declared in the method’s signature using the `throws` keyword, while unchecked exceptions do not require this.  
 —  > Examples of checked exceptions include `IOException`, `SQLException`, etc., while examples of unchecked exceptions include `NullPointerException`, `ArrayIndexOutOfBoundsException`, etc.

2. **Errors:**  
 — >  Errors, on the other hand, are typically severe issues that occur at runtime and are not easily recoverable by the program.  
 —  > They are often caused by external factors or internal system failures that are beyond the control of the program, such as out-of-memory errors, stack overflow errors, or hardware failures.  
 —  > Errors in Java are represented by classes that extend the `Error` class, which itself extends `Throwable`.  
 —  > Unlike exceptions, errors are typically not caught or handled by the program because they usually indicate serious problems that require intervention at a higher level, such as system administrators or developers.

In summary, exceptions are used to handle exceptional conditions that occur during the normal operation of a program and are recoverable, while errors represent severe problems that are typically not recoverable and may require intervention at a higher level. Exceptions are part of the expected flow of a program and are designed to be caught and handled, whereas errors are often indicative of underlying issues that need to be addressed at a system level.

**Q3. what is the difference between checked exception and unchecked exception?**

The main difference between checked and unchecked exceptions in Java lies in how they are handled by the compiler and the programmer:

1.**Checked Exceptions:**  
 — Checked exceptions are exceptions that the compiler forces you to handle. This means that if a method can throw a checked exception, the method must either catch and handle the exception using a `try-catch` block or declare the exception using the `throws` clause in its signature.  
 — Examples of checked exceptions include `IOException`, `SQLException`, `ClassNotFoundException`, etc.  
 — Checked exceptions are typically used for conditions that are outside the immediate control of the program but are expected to occur in normal circumstances, such as file I/O errors, database connection issues, etc.  
 — Handling checked exceptions ensures that the program’s flow is not disrupted abruptly and allows for graceful recovery or alternative actions.

2. **Unchecked Exceptions:**  
 — Unchecked exceptions, also known as runtime exceptions, are exceptions that the compiler does not force you to handle explicitly.  
 — These exceptions extend the `RuntimeException` class or one of its subclasses.  
 — Examples of unchecked exceptions include `NullPointerException`, `ArrayIndexOutOfBoundsException`, `IllegalArgumentException`, etc.  
 — Unchecked exceptions typically arise due to programming errors or logical mistakes in the code, such as attempting to access a null object reference, accessing an array index out of bounds, passing invalid arguments to a method, etc.  
 — Because unchecked exceptions are not required to be caught or declared, they can propagate up the call stack until they are caught or result in the program terminating abnormally.

The key differences are:  
- Checked exceptions must be handled or declared, while unchecked exceptions are not required to be explicitly handled.  
- Checked exceptions are typically used for external conditions beyond the program’s immediate control, while unchecked exceptions are often due to programming errors or logical mistakes.  
- Checked exceptions are subclasses of `Exception` (excluding subclasses of `RuntimeException`), while unchecked exceptions are subclasses of `RuntimeException` or directly extend `RuntimeException`.