Ques . 1

1. **public**:
   * **Class**: A public class can be accessed from any other class in any package. This is the most accessible level.
   * **Method**: A public method can be called from any other class, regardless of the package it belongs to.
   * **Variable**: A public variable can be accessed directly from any other class.
2. **protected**:
   * **Class**: Classes cannot be declared protected. This modifier only applies to class members (methods and variables).
   * **Method**: A protected method can be accessed within its own package and by subclasses (including those in different packages).
   * **Variable**: A protected variable can be accessed within its own package and by subclasses (including those in different packages).
3. **Default (no modifier)**:
   * **Class**: A class with no access modifier (often called package-private) is only accessible within its own package. It cannot be accessed from classes in other packages.
   * **Method**: A method with no access modifier (package-private) is accessible only within the same package. It is not accessible from outside the package.
   * **Variable**: A variable with no access modifier (package-private) is also accessible only within the same package.
4. **private**:
   * **Class**: Classes cannot be declared private (except in nested classes, where the outer class can have private nested classes).
   * **Method**: A private method is accessible only within the class in which it is defined. It is not visible to any other class.
   * **Variable**: A private variable is accessible only within the class in which it is defined. It cannot be accessed by any other class.

**Summary of Accessibility**:

* **public**: Accessible from everywhere.
* **protected**: Accessible within the same package and by subclasses.
* **Default**: Accessible only within the same package.
* **private**: Accessible only within the same class.

**Ques .2**

**Exceptions**

1. **Definition**: Exceptions are conditions that a program should handle. They are usually the result of issues that occur during the normal operation of a program but are recoverable if handled properly.
2. **Hierarchy**: Exceptions are part of the java.lang.Exception class hierarchy. This class is a subclass of java.lang.Throwable. Exceptions are further divided into:
   * **Checked Exceptions**: These are exceptions that must be either caught or declared in the method signature using the throws keyword. Examples include IOException, SQLException, etc.
   * **Unchecked Exceptions**: These are exceptions that do not need to be explicitly handled or declared. They are subclasses of RuntimeException. Examples include NullPointerException, ArrayIndexOutOfBoundsException, etc.
3. **Handling**: Exceptions are generally anticipated and can be handled using try-catch blocks. If not handled, checked exceptions will cause the program to fail to compile, whereas unchecked exceptions can cause runtime failures.
4. **Use Cases**: Exceptions are used to handle errors in a program's logic or external factors (like I/O operations) that may need special handling. They often represent conditions that are not necessarily fatal and can be managed programmatically.

**Errors**

1. **Definition**: Errors represent serious problems that a program typically cannot recover from. They are usually indicative of a fundamental issue in the environment in which the application is running or a critical failure.
2. **Hierarchy**: Errors are part of the java.lang.Error class hierarchy, which is a subclass of java.lang.Throwable. Examples include OutOfMemoryError, StackOverflowError, etc.
3. **Handling**: Errors are generally not meant to be caught or handled by applications. They are usually outside the control of the application and often indicate that the JVM or the environment is in a problematic state.
4. **Use Cases**: Errors indicate severe problems such as resource exhaustion or internal JVM issues. Since they usually represent unrecoverable states, application-level handling of errors is typically not recommended. The focus is more on preventing conditions that might lead to errors or addressing the root cause of the error if possible.

**Ques . 3**

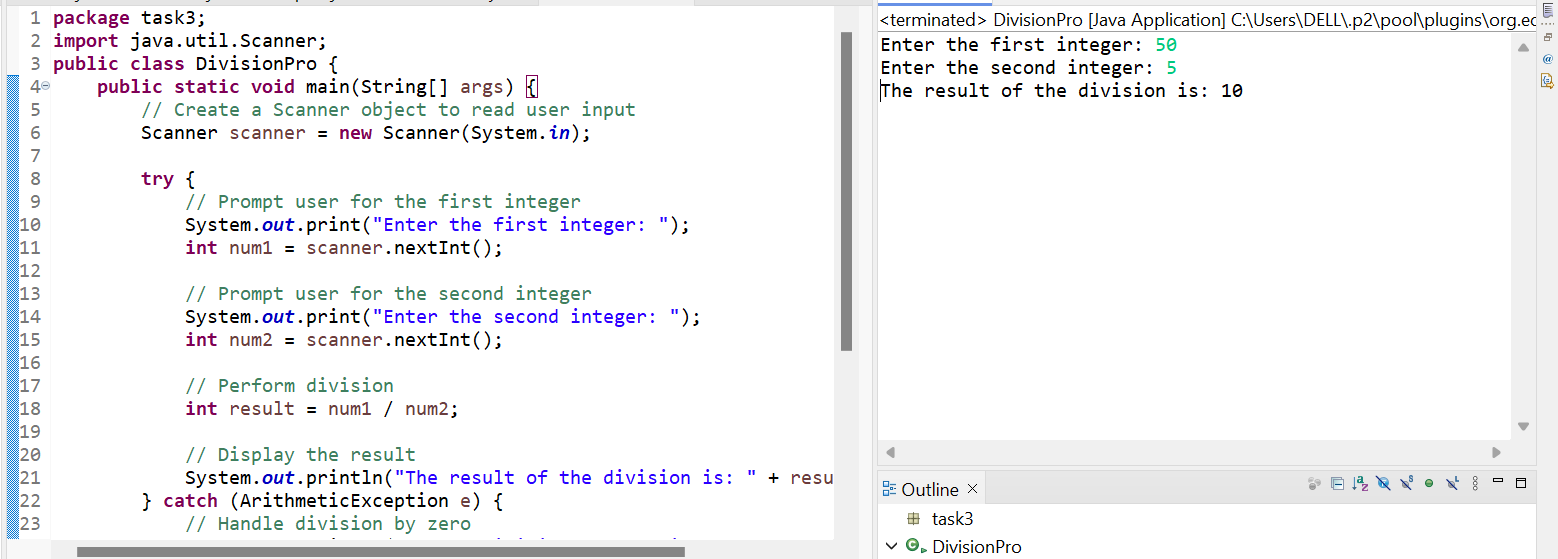
**Checked Exceptions**

1. **Definition**: Checked exceptions are exceptions that must be either caught or declared in the method signature using the throws keyword. These exceptions represent conditions that a reasonable application should anticipate and handle.
2. **Inheritance**: Checked exceptions are subclasses of Exception but not subclasses of RuntimeException.
3. **Handling**:
   * **Compilation Requirement**: The compiler enforces that checked exceptions must be handled. If a method throws a checked exception, it must either catch the exception using a try-catch block or declare it in its throws clause.
   * **Declaration**: If a method calls another method that throws a checked exception, it must either handle the exception or declare that it throws the exception in its own method signature.
4. **Examples**: Common examples include:
   * IOException: Thrown when an I/O operation fails or is interrupted.
   * SQLException: Thrown when there is a database access error.
   * ClassNotFoundException: Thrown when an application tries to load a class that cannot be found.
5. **Usage**: Checked exceptions are typically used for conditions that are expected and can be recovered from or handled gracefully, such as file handling errors or network issues.

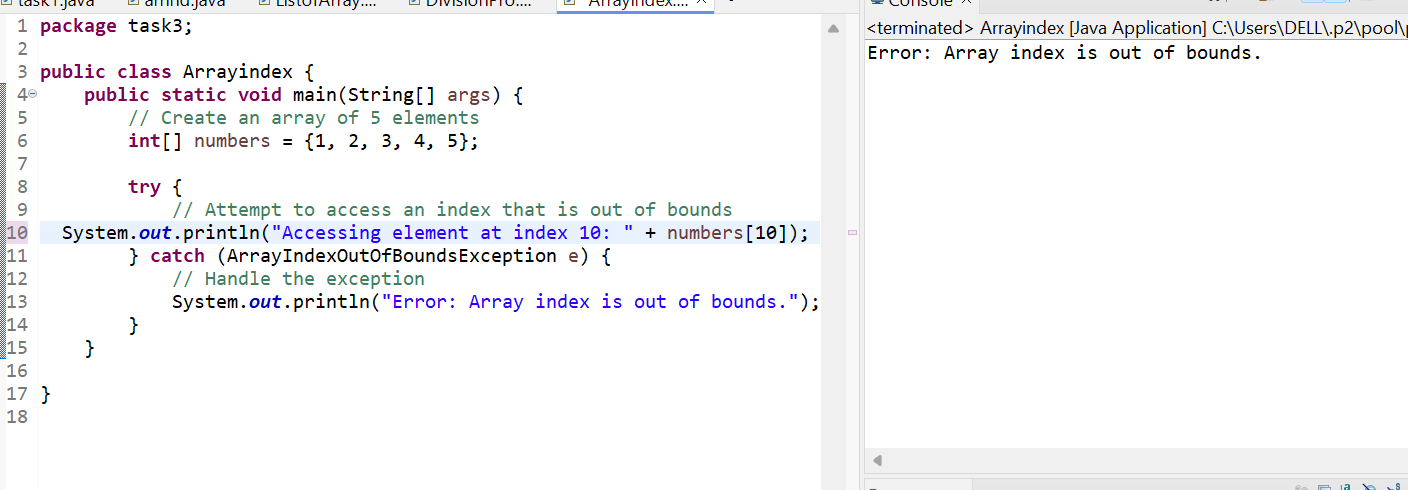
**Unchecked Exceptions**

1. **Definition**: Unchecked exceptions are exceptions that do not need to be explicitly handled or declared. They represent programming errors that are often due to bugs or logical errors in the code.
2. **Inheritance**: Unchecked exceptions are subclasses of RuntimeException, which is a subclass of Exception.
3. **Handling**:
   * **Compilation Requirement**: The compiler does not enforce the handling of unchecked exceptions. The developer has the option to handle them, but it is not mandatory.
   * **Declaration**: Methods are not required to declare unchecked exceptions in their throws clause, even if they can throw such exceptions.
4. **Examples**: Common examples include:
   * NullPointerException: Thrown when an application attempts to use null where an object is required.
   * ArrayIndexOutOfBoundsException: Thrown when an array is accessed with an illegal index.
5. IllegalArgumentException: Thrown when a method receives an argument that is inappropriate.
6. **Usage**: Unchecked exceptions are generally used for programming errors that the programmer should fix, such as logic errors or misuse of the API. They usually indicate a flaw in the code that should be corrected rather than handled at runtime.

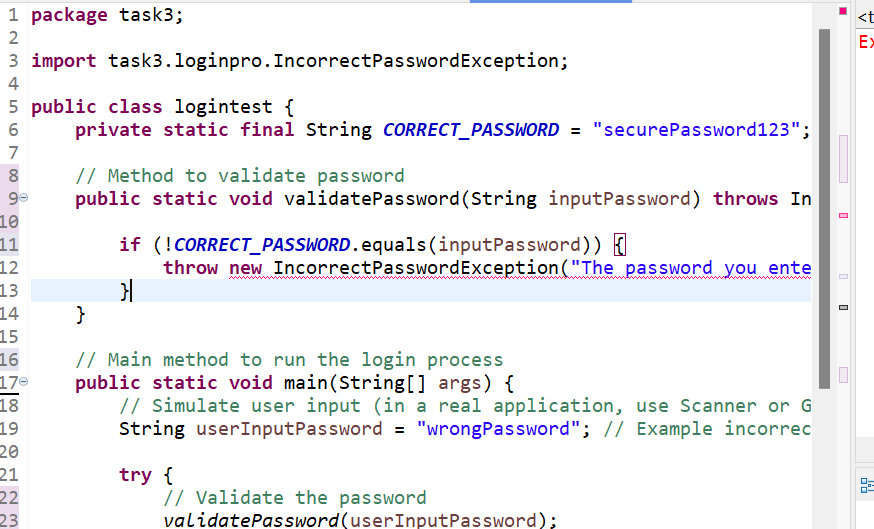
Ques . 4



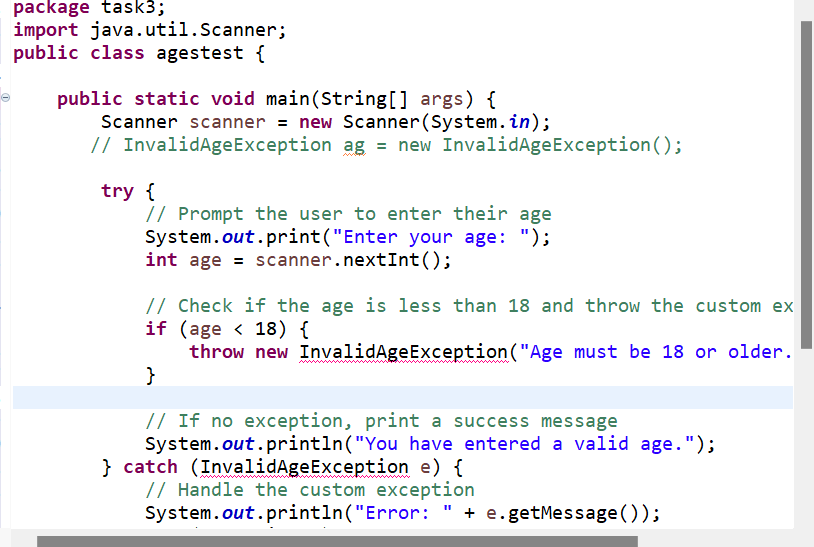
Ques .5



Ques . 6



Ques . 7



Ques . 8

