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

* Public (public):
  + Classes: Public classes can be accessed from any other class in any package.
  + Methods: Public methods can be accessed from any class in any package.
  + Variables: Public variables can be accessed from any class in any package.
* Protected (protected):
  + Classes: While classes cannot be declared as protected, all classes in the same package can access protected members of other classes in the same package.
  + Methods: Protected methods can be accessed by classes in the same package and by subclasses (even if they are in different packages).
  + Variables: Protected variables have similar accessibility as protected methods.
* Default (Package-Private or No Modifier):
  + Classes: If no access modifier is specified, the class is given "default" access, which means it can only be accessed by classes in the same package.
  + Methods: Methods without an access modifier (package-private) can be accessed by classes in the same package.
  + Variables: Variables without an access modifier (package-private) have similar accessibility as package-private methods.
* Private (private):
  + Classes: Classes cannot be declared as private.
  + Methods: Private methods can only be accessed within the same class.
  + Variables: Private variables can only be accessed within the same class.

Access modifiers play a crucial role in encapsulation, which is a fundamental principle in object-oriented programming. Encapsulation helps control the visibility and access to internal details of a class, allowing developers to create well-defined and maintainable code by limiting the interaction between different parts of the program. By choosing appropriate access modifiers, you can ensure that your classes, methods, and variables are only accessible where they are intended to be used, reducing the risk of unintended side effects and enhancing code security.

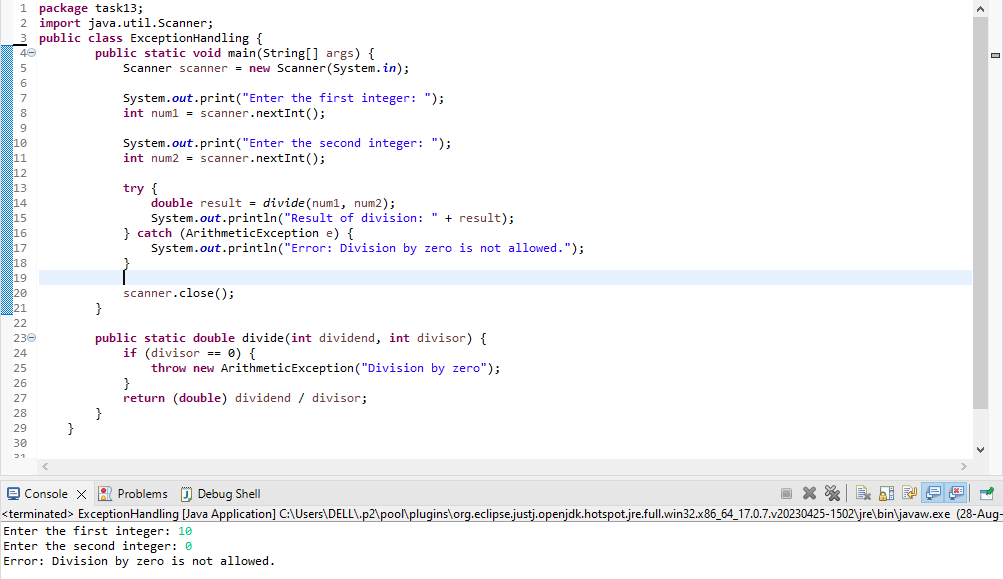
**2. What is the difference between Exception and error?**

| Aspect | Checked Exception | Unchecked Exception |
| --- | --- | --- |
| Handling Requirement | Must be either caught or declared with throws keyword. | No explicit handling or declaration required. |
| Inheritance | Subclass of Exception (excluding RuntimeException). | Subclass of RuntimeException. |
| Compile-time Checked | Compiler enforces handling or declaration. | Compiler does not enforce handling. |
| Examples | IOException, SQLException | NullPointerException, ArrayIndexOutOfBoundsException |
| Typical Use Case | Recoverable scenarios, like I/O errors. | Programming errors, like null references. |
| Try-Catch Block | Often used with try-catch blocks. | Can be used with try-catch, but not required. |
| Code Clarity | Explicitly indicates potential issues. | Might result in cleaner but riskier code. |
| Forced Handling | Enforced by compiler. | Not enforced by compiler. |
| RuntimeException | Does not include RuntimeException. | Includes RuntimeException and subclasses. |

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

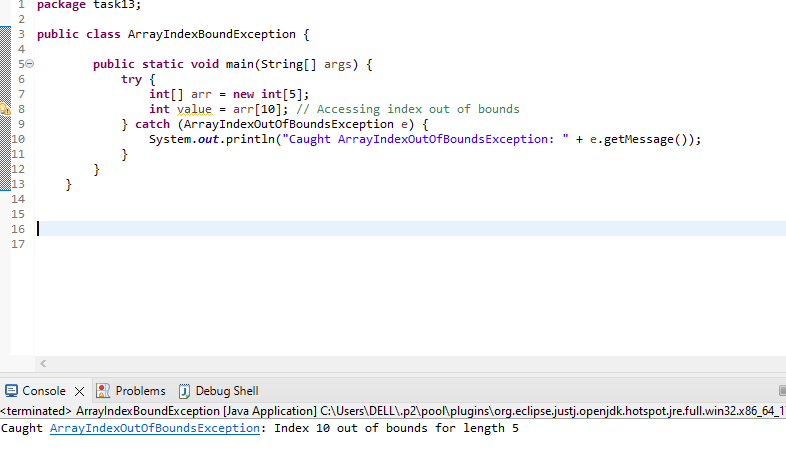
| Aspect | Exception | Error |
| --- | --- | --- |
| Definition | An exception is an unexpected event or condition that disrupts the normal flow of a program during runtime. | An error is a broad term encompassing any deviation from the expected behavior of a program, including syntax, runtime, and logical errors. |
| Occurrence | Occurs during runtime when an unforeseen condition arises that the program cannot handle. | Can occur at various stages, including compilation, runtime, and logical reasoning. |
| Type | Specific type of error that represents a distinct exceptional scenario. | Generic term that covers various types of deviations from expected behavior. |
| Handling | Typically handled using try-catch blocks or similar mechanisms. The program can attempt to recover from an exception. | Handling varies depending on the error type. Some errors may not be recoverable and can lead to program termination. |
| Propagation | Exceptions can be propagated up the call stack until caught or handled. | Errors can propagate as well, often leading to the termination of the program. |
| Granularity | Generally more granular, representing specific issues that can be addressed individually. | Broader term that may include specific exceptions but also encompasses other types of issues. |
| Examples | Division by zero, file not found, null pointer dereference. | Syntax errors (typos, missing semicolons), runtime errors (memory allocation failure), logical errors (incorrect calculations). |

**4. Write a Java program that reads user input for two integers and performs division. Handle the exception that is thrown when the second number is zero, and display an error message to the user.**

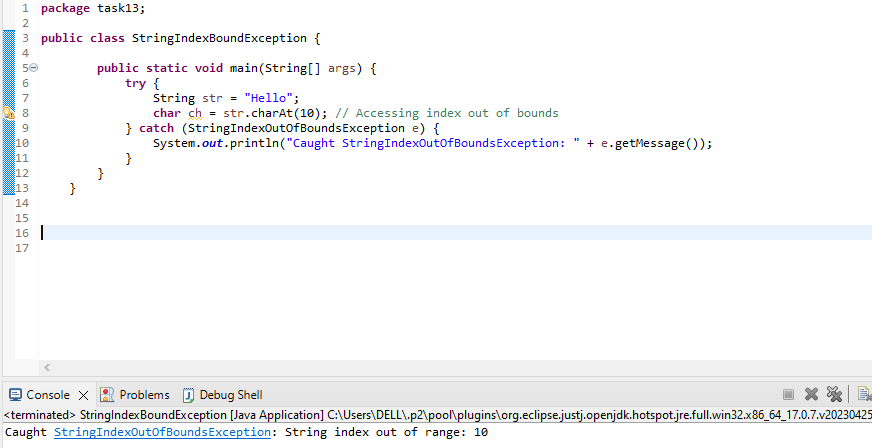
****

**5.Write the code of ArrayIndexOutOfBoundsException & StringIndexOutOfBoundsException?**

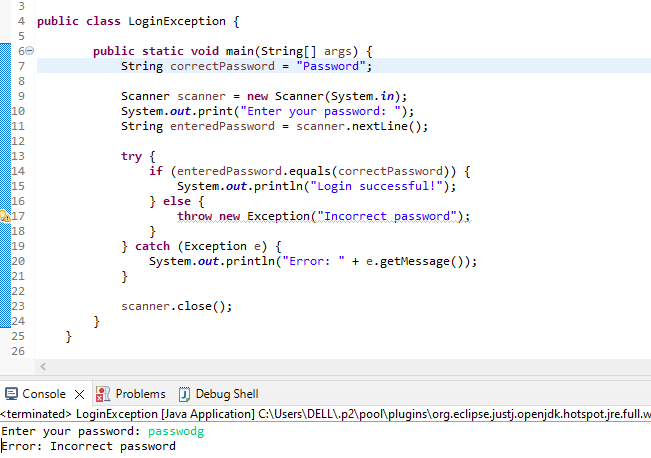
**ArrayIndexOutOfBoundsException**

****

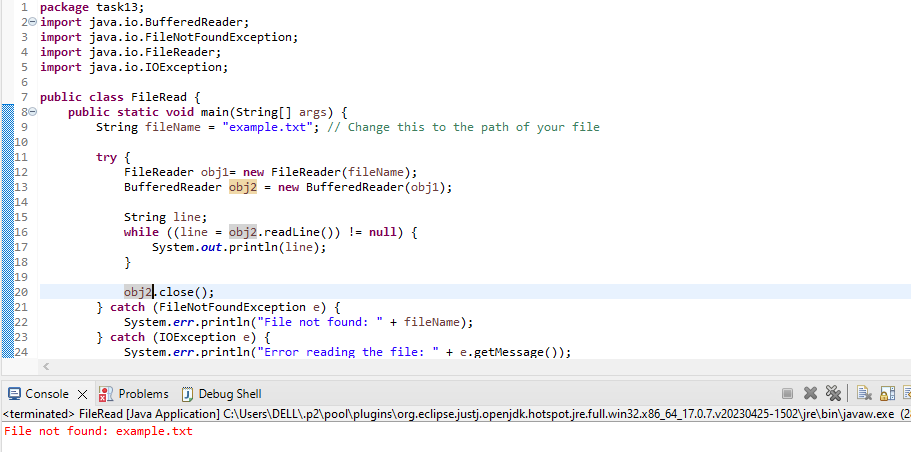
**StringIndexOutOfBoundsException**

****

**6. You are building a login system for a website using Java. If the user enters an incorrect password, you want to display a message informing them of the error. How would you use exception handling to handle this situation?**

****

**7.Implement exception handling in a Java program that reads data from a file. If the file does not exist, throw a "FileNotFoundException" and display an error message to the user**

****