**Java Exceptions and Error Handling**

**1.Access modifiers in Java and their significance in class,method and variable accessibility**

In Java there are four main access modifiers that control the visibility and accessibility of classes,methods and variables.These access modifiers determine which parts of your code can access and interact with the class ,method or variable.The four access modiifers are

* **public**: The public access modifier allows the class, method, or variable to be accessed from any other class. It has the widest scope of accessibility. For example, a public class can be accessed from any other class, and a public method or variable can be used by any other class that has visibility to the class containing the public member.

* **protected**: The protected access modifier allows a class, method, or variable to be accessed by classes within the same package and by subclasses, regardless of the package they are in. It's more restrictive than public but provides a level of access suitable for extending classes and maintaining encapsulation.

* **default (no modifier)**: When no access modifier is specified, it is considered package-private (also called default access). This means that the class, method, or variable can only be accessed by other classes in the same package. It restricts visibility to a specific package, making it useful for internal package implementation details.

* **private**: The private access modifier restricts the visibility of a class member to only the class in which it is declared. It is the most restrictive access level and is used to hide the implementation details of a class from other classes. Private members are not accessible from outside the class.

            The Significance of the access modifiers are

* **Class Access**:

-public: Accessible from anywhere.

-protected: Accessible within the same package and by subclasses (even if they are in different packages).

-Default: Accessible within the same package.

-private: Accessible only within the same class.

* **Method and Variable Access**:

            -public: Accessible from anywhere.

   -protected: Accessible within the same package and by subclasses

-Default: Accessible within the same package.

-private: Accessible only within the same class.

**2.Difference between Exception and Error**

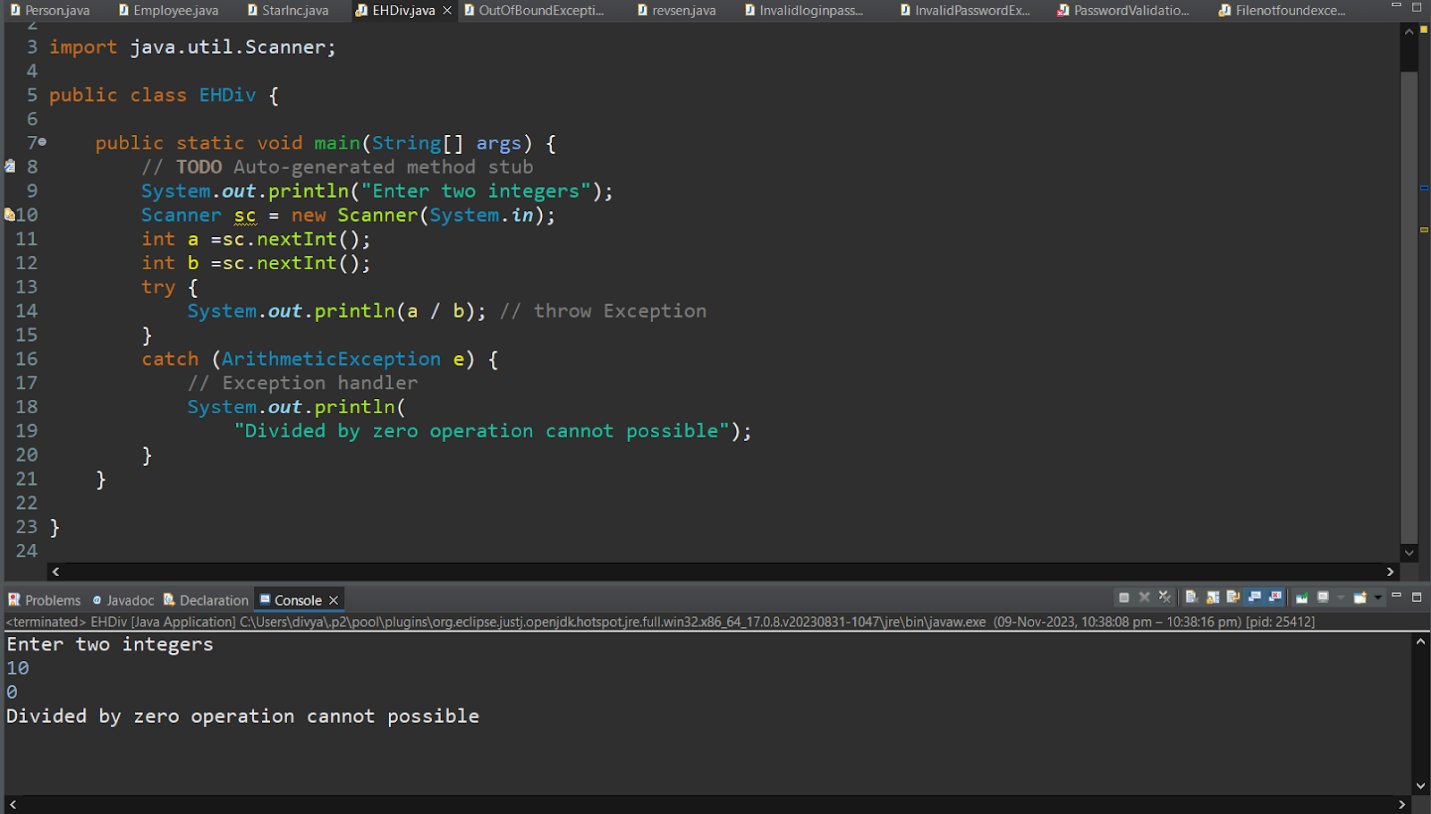
|  |  |  |
| --- | --- | --- |
| **No** | **Exception** | **Error** |
| 1 | Exceptions represents events  or conditions that can occur during the normal execution of the program. | Errors represent serious and often unrecoverable issues that are not typically handled by the program. |
| 2 | They are typically caused by problems that a well-written program can anticipate and handle, such as invalid input or network issues. | They usually indicate problems at the system or JVM level, such as out-of-memory errors or hardware failures. |
| 3 | Exceptions are meant to be caught and handled by the program to prevent it from crashing. | Errors are not meant to be caught or handled by regular program code because doing so may not be effective, and the program might not be in a reliable state. |
| 4 | Exceptions are usually recoverable, and the program can continue to execute after handling an exception. | Errors are typically reserved for unrecoverable issues at a higher system level. |
| 5 | Example  public class ExampleException {      public static void main(String[] args) {          try {              int result = divide(10, 0);              System.out.println("Result: " + result);          } catch (ArithmeticException e) {              System.out.println("An error occurred: " + e.getMessage());          }      }      public static int divide(int a, int b) {          if (b == 0) {              throw new ArithmeticException("Division by zero is not allowed.");          }          return a / b;      }  }  In this example, an ArithmeticException is thrown when attempting to divide by zero, and it is caught and handled in the catch block. | Example  public class ExampleError {      public static void main(String[] args) {          try {              recursiveMethod(10);          } catch (StackOverflowError e) {              System.out.println("Stack overflow error: " + e.getMessage());          }      }      public static void recursiveMethod(int n) {          if (n == 0) {              return;          }          recursiveMethod(n - 1);      }  }  In this example, I intentionally create a Stackoverflow by making a recursive call without a base case. |

**3. Difference between Checked and Unchecked exceptions**

|  |  |  |
| --- | --- | --- |
| **No** | **Checked Exception** | **Unchecked Exception** |
| 1 | Checked exceptions are exceptions that are checked at compile time by the Java compiler. | Unchecked exceptions, also known as runtime exceptions, are not checked at compile time. The compiler does not enforce catching or declaring them. |
| 2 | When a method may throw a checked exception, the compiler enforces that you handle the exception, either by catching it with a try-catch block or declaring that the method itself throws the exception using the throws clause. | These exceptions usually indicate programming errors or issues that should be avoided through proper coding. |
| 3 | Examples of checked exceptions include IOException, SQLException, and ClassNotFoundException. | Examples of unchecked exceptions include NullPointerException, ArrayIndexOutOfBoundsException, and ArithmeticException. |
| 4 | Example:  import java.io.FileReader;  import java.io.IOException;  public class CheckedExceptionExample {      public static void main(String[] args) {          try {              FileReader fileReader = new FileReader("nonexistentfile.txt");          } catch (IOException e) {              System.out.println("File not found or cannot be read.");          }      }  } | Example:  public class UncheckedExceptionExample {      public static void main(String[] args) {          int[] numbers = {1, 2, 3};          System.out.println(numbers[4]); // This will throw an ArrayIndexOutOfBoundsException      }  } |

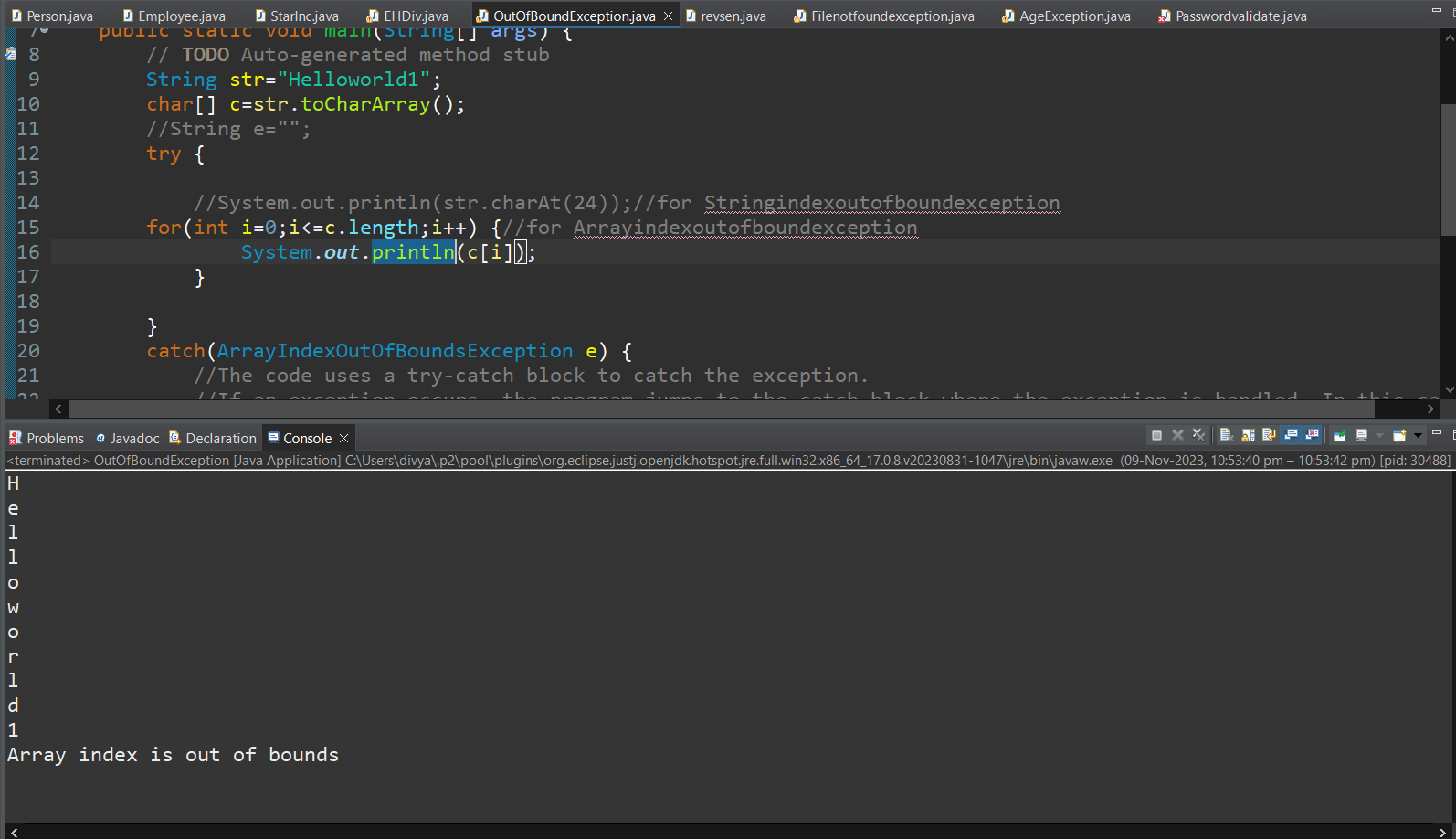
**Programs output:**

**4. Java program to handle exception while dividing a number by zero**

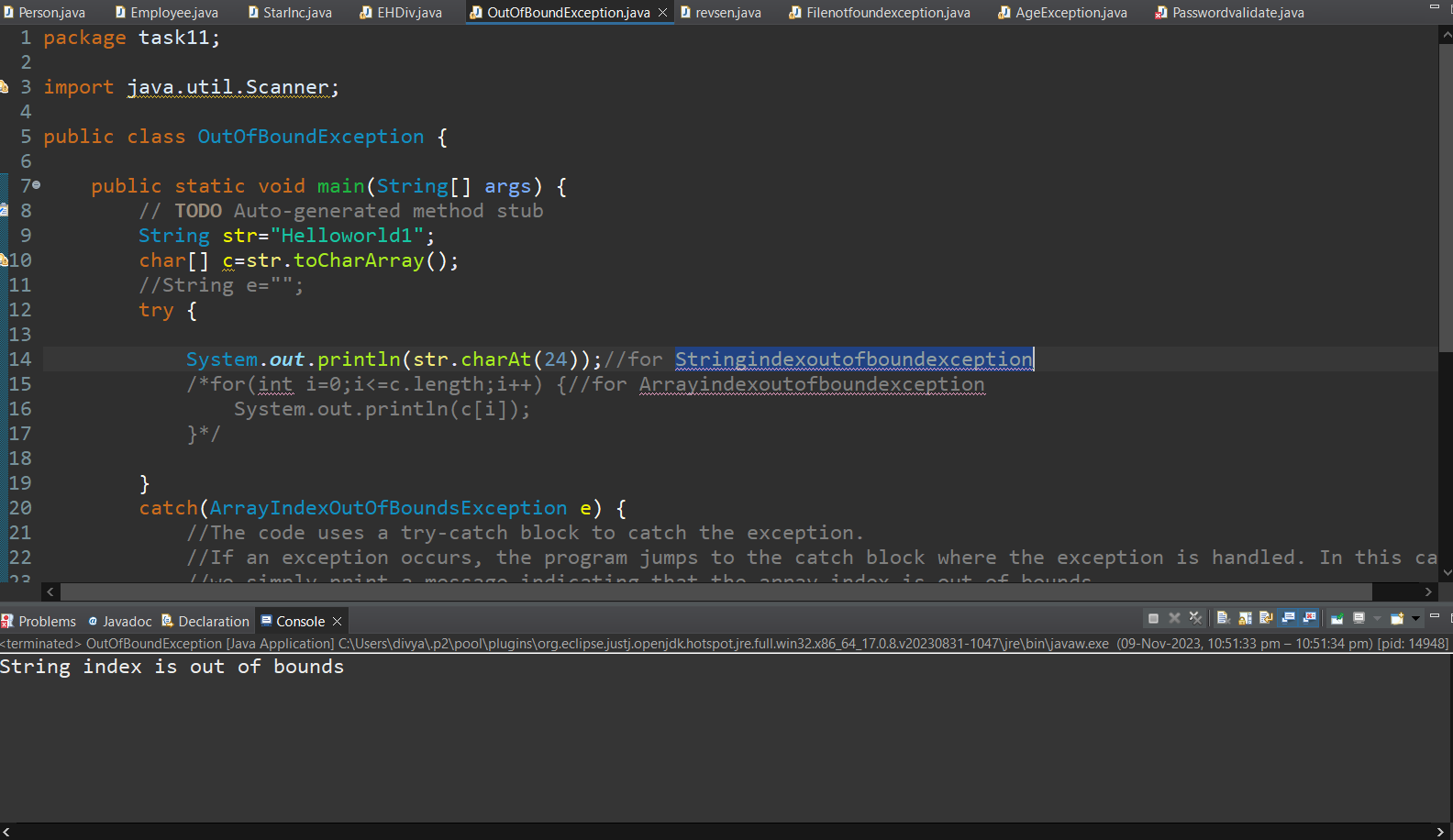


**5. Java program for ArrayIndexOutOfBoundException and StringIndexOutOfBoundException**

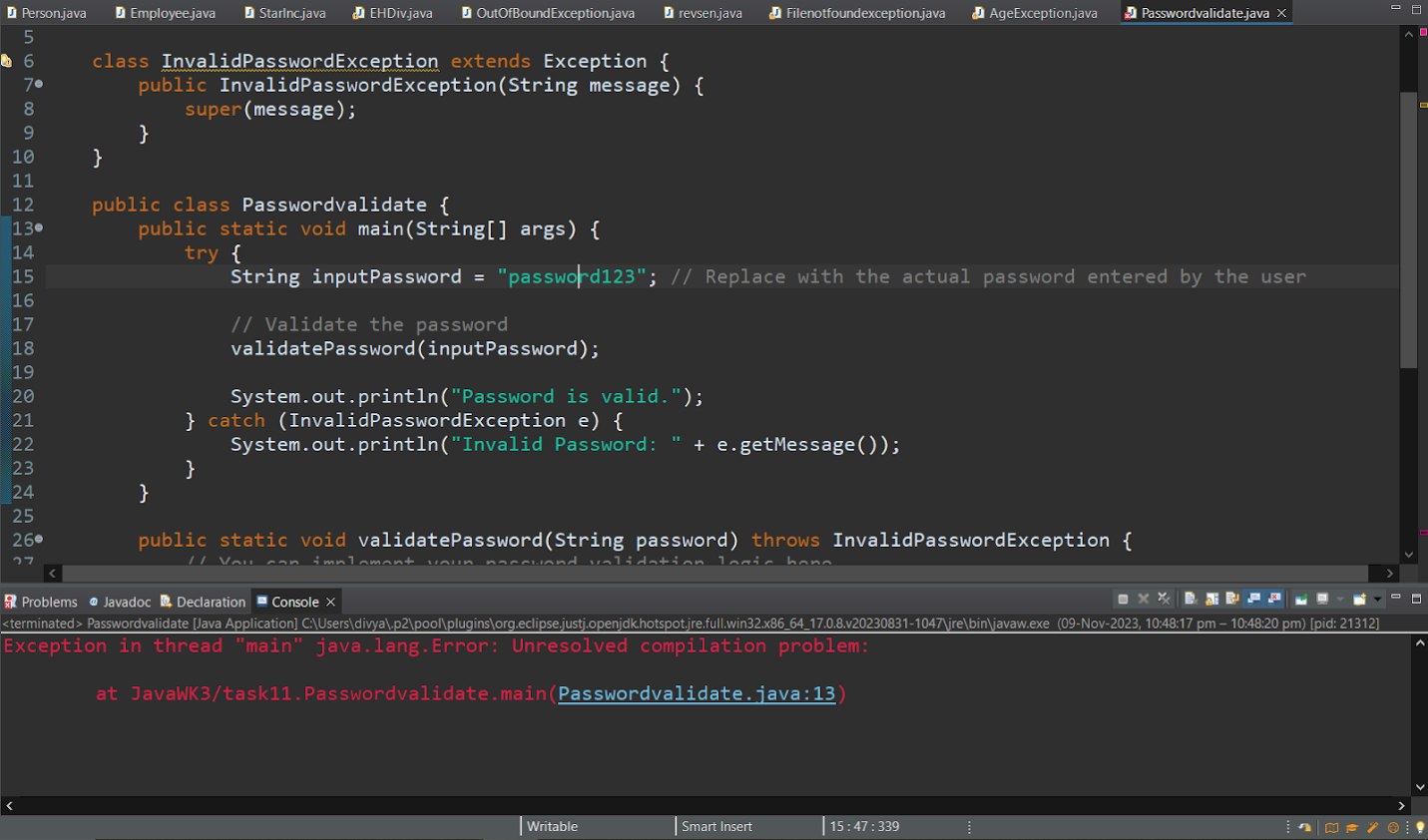
**ArrayIndexOutOfBoundException**



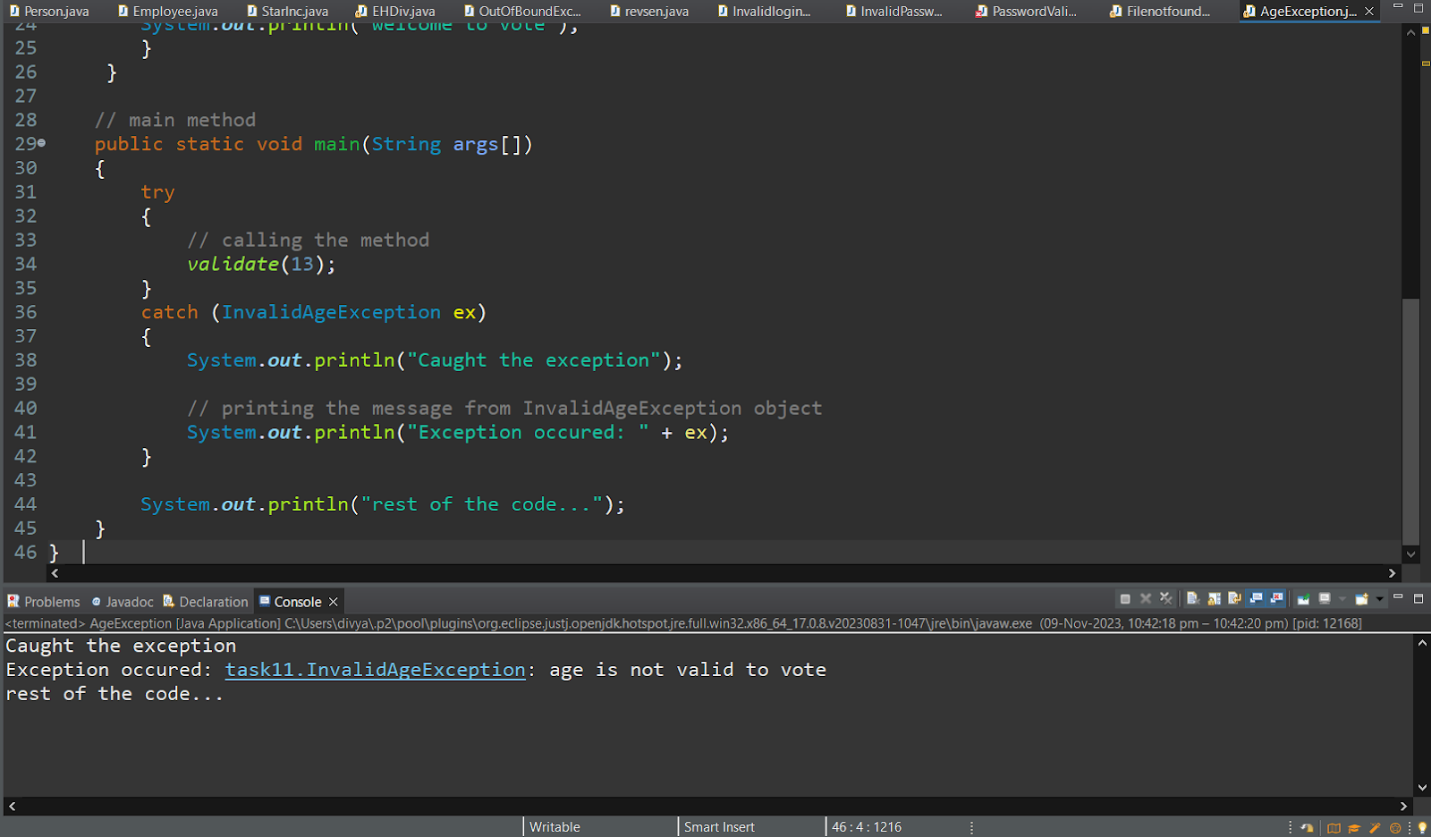
**StringIndexOutOfBoundException**

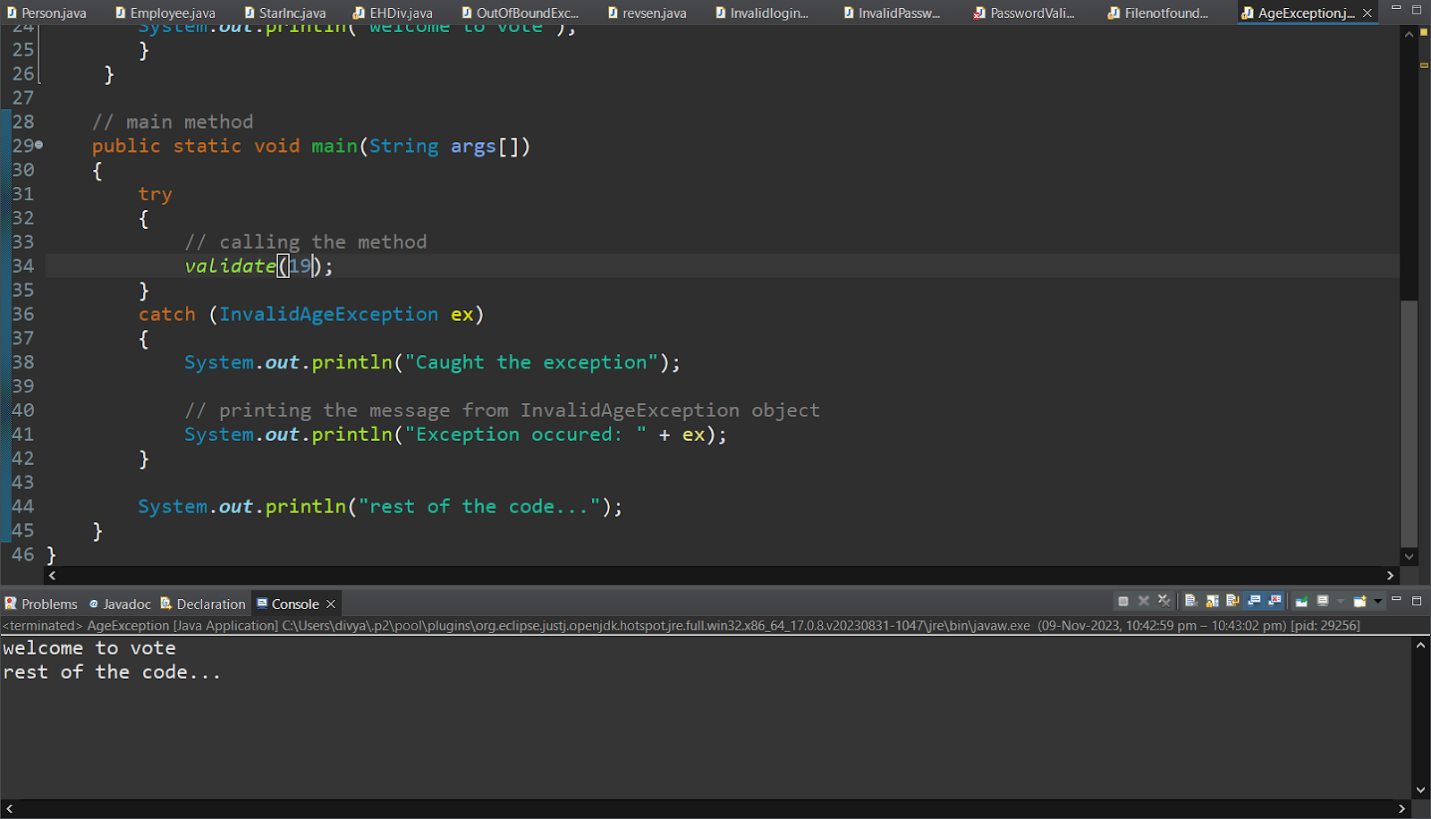


**6. Java program for Exception handling with incorrect password**



**7. Java program for Invalid Age exception**





**8. Java program for Filenotfoundexception**

