**Answer 1 :** In Java, there are four main access modifiers, each of which serves to establish the visibility of classes, methods, and variables within the program. These access modifiers are:

1. **‘public’**: The **‘public’** access modifier specifies that the class, method, or variable is accessible from any other class in the Java environment, regardless of the package it belongs to. If a class is declared as public, it can be accessed from any other class.
2. **‘protected’**: The **‘protected’** access modifier allows the class, method, or variable to be accessed within its own package and by subclasses of its class in other packages. If a variable or method is declared as protected, it can be accessed by any class in the same package or by a subclass located in any package.
3. **‘default’** (no modifier): When no access modifier is specified for a class, method, or variable, it is treated as having default access. Default access means that the class, method, or variable is accessible only within its own package and is not available to classes from different packages.
4. **‘private’**: The **‘private’** access modifier is the most restrictive. If a class member (method or variable) is declared as private, it can only be accessed within the class in which it is declared. Private members are not visible to subclasses or classes from different packages.

The significance of these access modifiers in terms of class, method, and variable accessibility is that they provide a way to encapsulate the data and enforce a level of abstraction in the program. This encapsulation protects the integrity of the object by preventing external classes from interacting with internal workings that they should not be concerned with, allowing the developer to change internal workings without affecting code that uses the class. Access modifiers are a foundational concept of object-oriented programming and help with the maintenance and scalability of the software.

**Answer 2 :** In Java, Exception and Error are both subclasses of the Throwable class, which means they can be thrown and caught. However, they represent different issues that can arise during the execution of a program.

1. **Exception**: An Exception indicates a condition that a reasonable application might want to catch. Exceptions are conditions that occur as a result of the program execution but are not fatal to the program. They can be either checked or unchecked.

* **Checked Exceptions**: These are exceptions that are checked at compile-time. They must be either caught or declared in the method signature using the throws keyword. Examples include IOException, SQLException, and ClassNotFoundException.
* **Unchecked Exceptions**: These are exceptions that are not checked at compile time, so the compiler does not require methods to catch or declare them. They usually indicate programming bugs, such as logic errors or improper use of an API. Examples include NullPointerException, ArrayIndexOutOfBoundsException, and IllegalArgumentException.

1. **Error**: An Error indicates serious problems that a reasonable application should not try to catch, typically because they are outside the control of the application. Errors are usually thrown by the Java Virtual Machine (JVM) and indicate problems that are not expected to be caught under normal circumstances. Examples include OutOfMemoryError, StackOverflowError, and LinkageError.

The main difference is in their intended use: exceptions are meant for conditions that the application can recover from, while errors are intended to signal issues that are likely beyond the application’s control, often indicating that the program should terminate. In practice, you will typically write code to catch and handle exceptions, but it's very rare to write code to catch errors.

**Answer 3 :** In Java, exceptions are divided into two main categories: checked and unchecked exceptions, which are differentiated by how the compiler handles them.

1. **Checked Exceptions:** Checked exceptions are exceptions that are checked at compile time. This means that the compiler requires that these exceptions must be either caught within a try-catch block or declared to be thrown in the method signature using the throws keyword. Failure to do so will result in a compile-time error.

**Purpose**: They are used to represent conditions that a program can reasonably be expected to recover from or handle in some way. They often represent invalid conditions in areas outside the immediate control of the program (e.g., I/O errors).

**Examples**: IOException, FileNotFoundException, ClassNotFoundException, SQLException, etc.

1. **Unchecked Exceptions**: Unchecked exceptions are not checked at compile time, meaning that the compiler does not require methods to catch or declare them. They are also known as runtime exceptions.

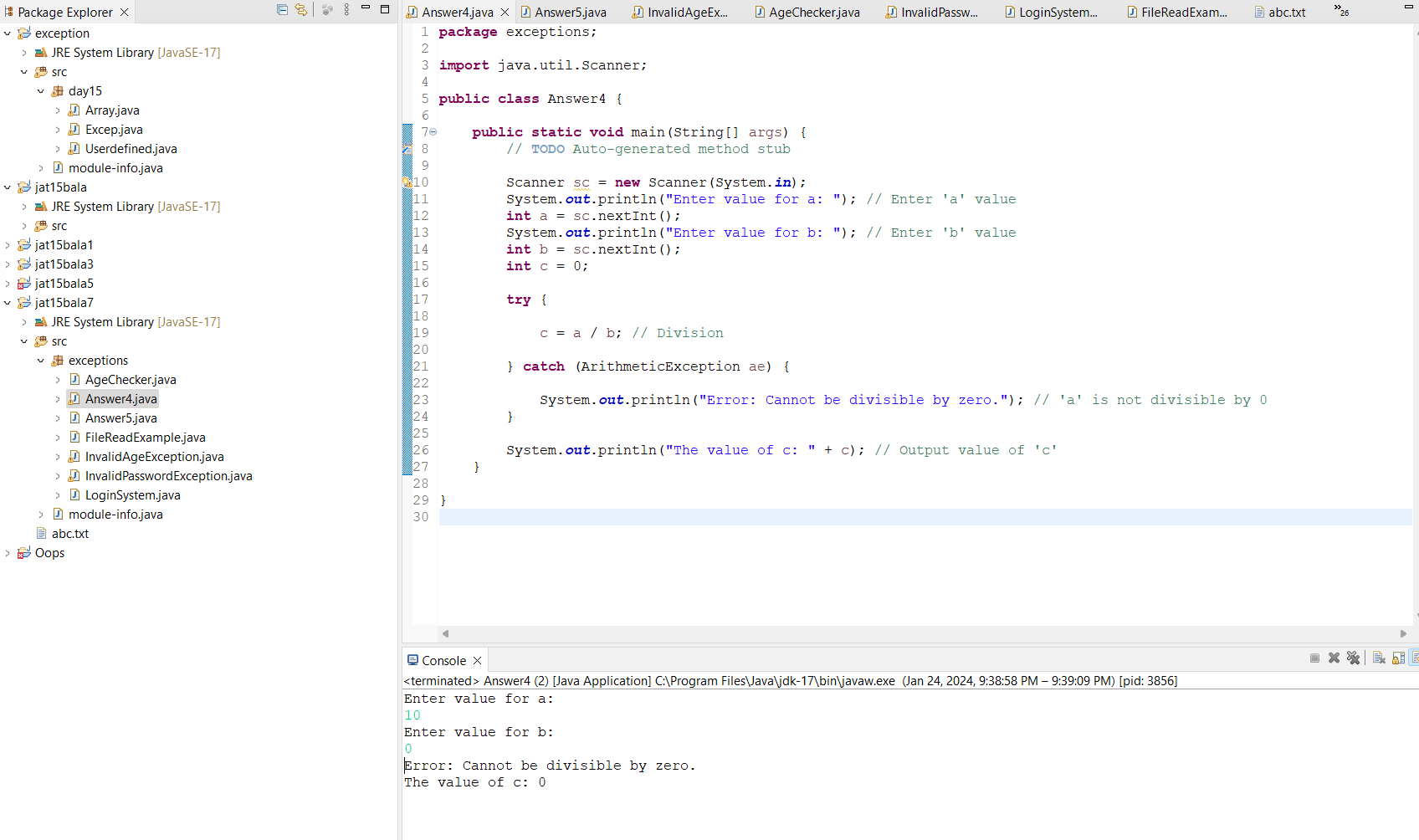
**Purpose**: They are used to indicate problems that are typically the result of a programming error, such as bugs in the code. They are usually the program’s fault and could have been avoided by the programmer.

**Examples**: ArithmeticException, NullPointerException,

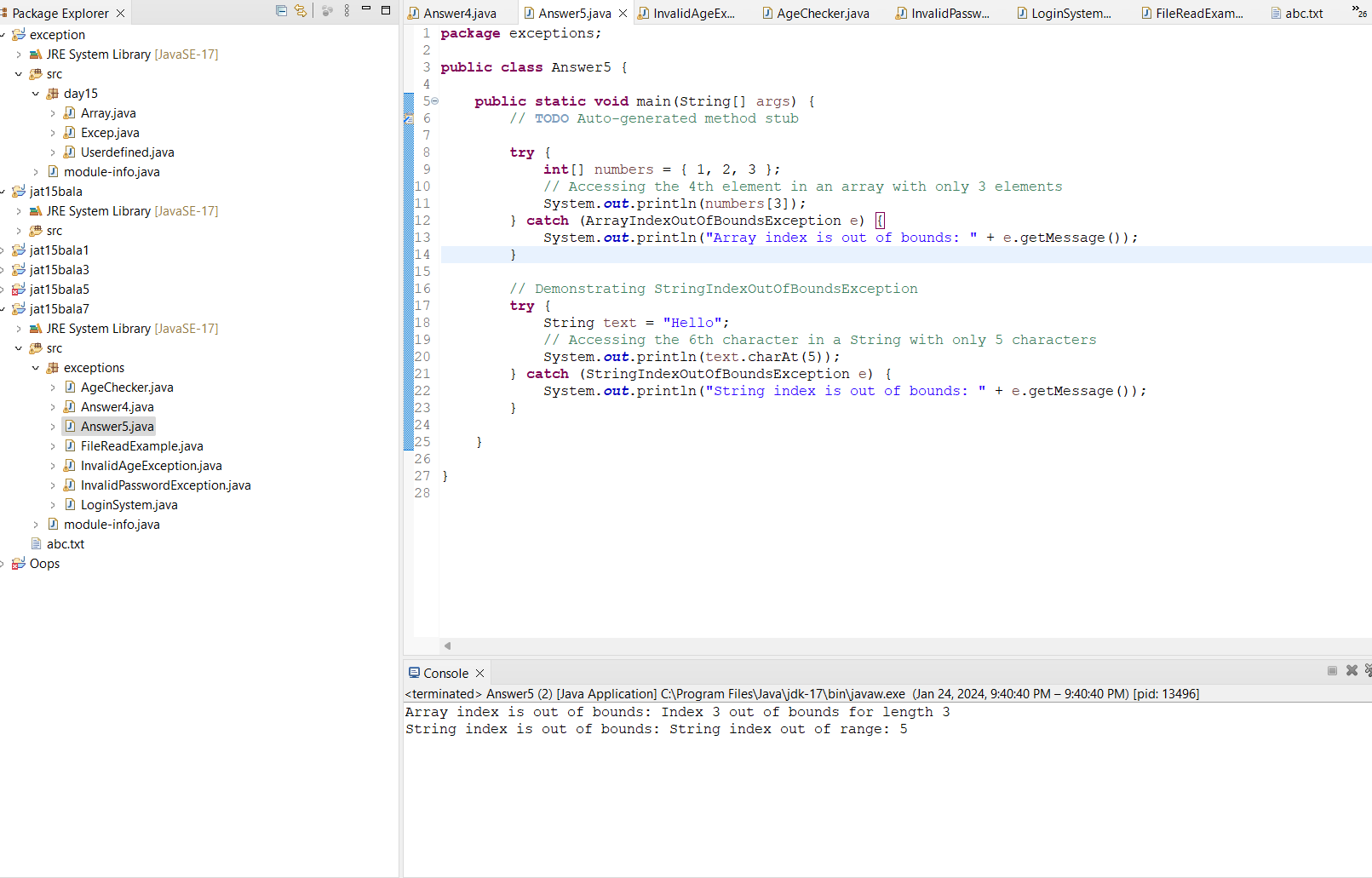
ArrayIndexOutOfBoundsException, IllegalArgumentException, etc.

The distinction between checked and unchecked exceptions is significant because it affects how a programmer writes code to deal with the potential problems that may arise during the execution of the program. Checked exceptions encourage error handling by explicitly forcing the programmer to address them through either try-catch blocks or throws clauses. Unchecked exceptions, on the other hand, allow the programmer to choose whether to handle them, with the understanding that they typically indicate programming errors that should be fixed in the code rather than being caught and handled at runtime.

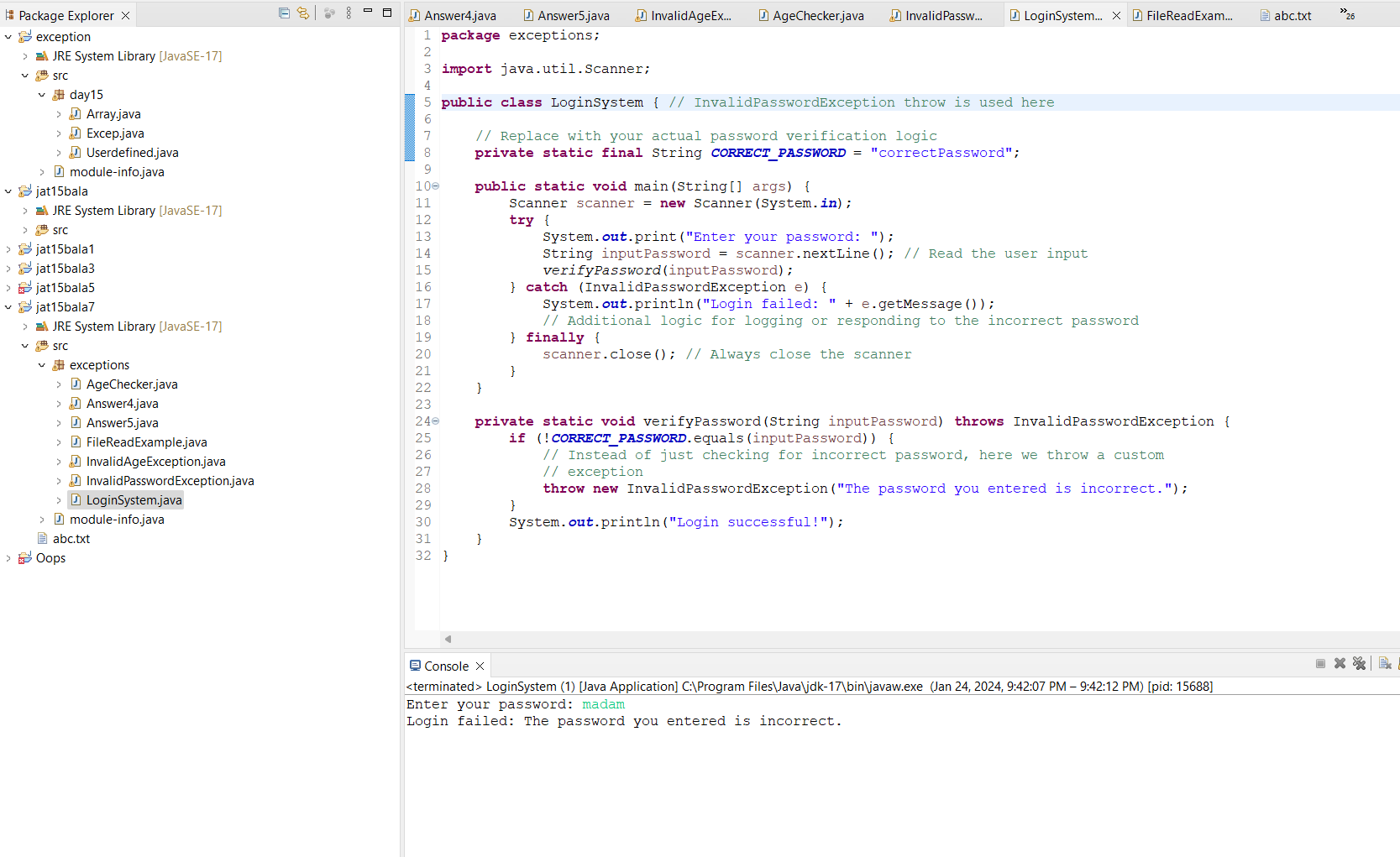
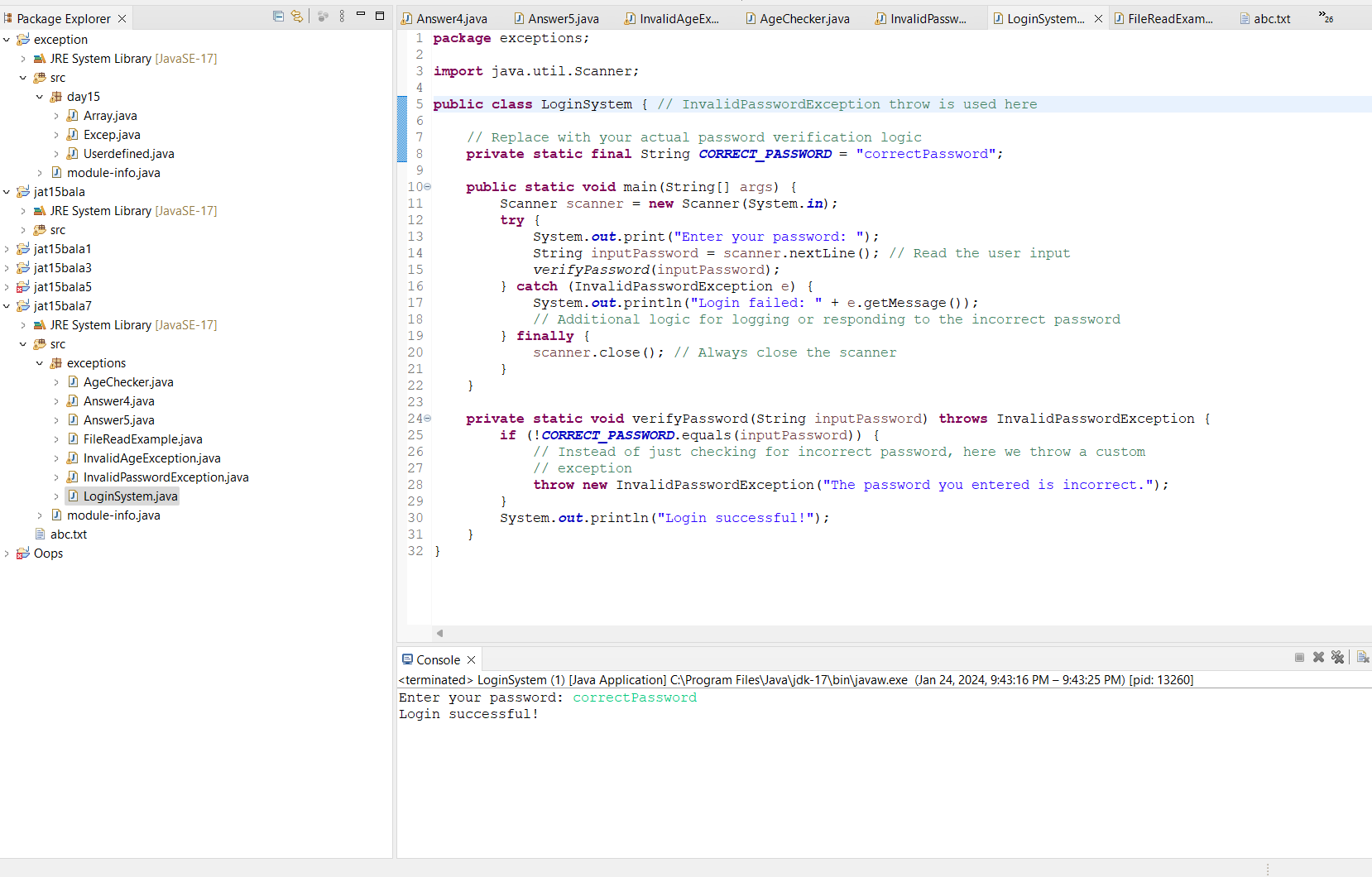
**Answer 4 :** Output Screenshot of Answer4.java



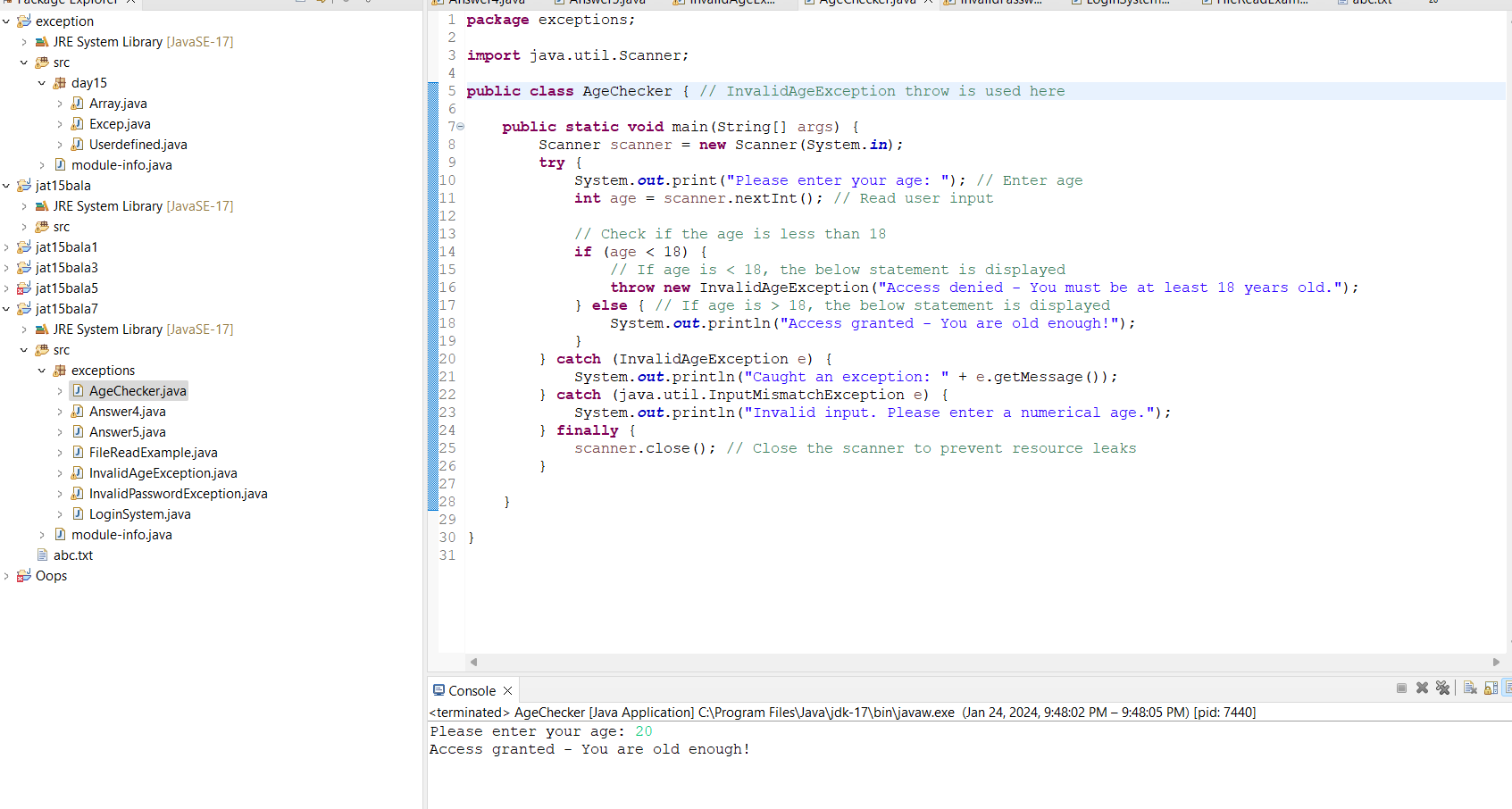
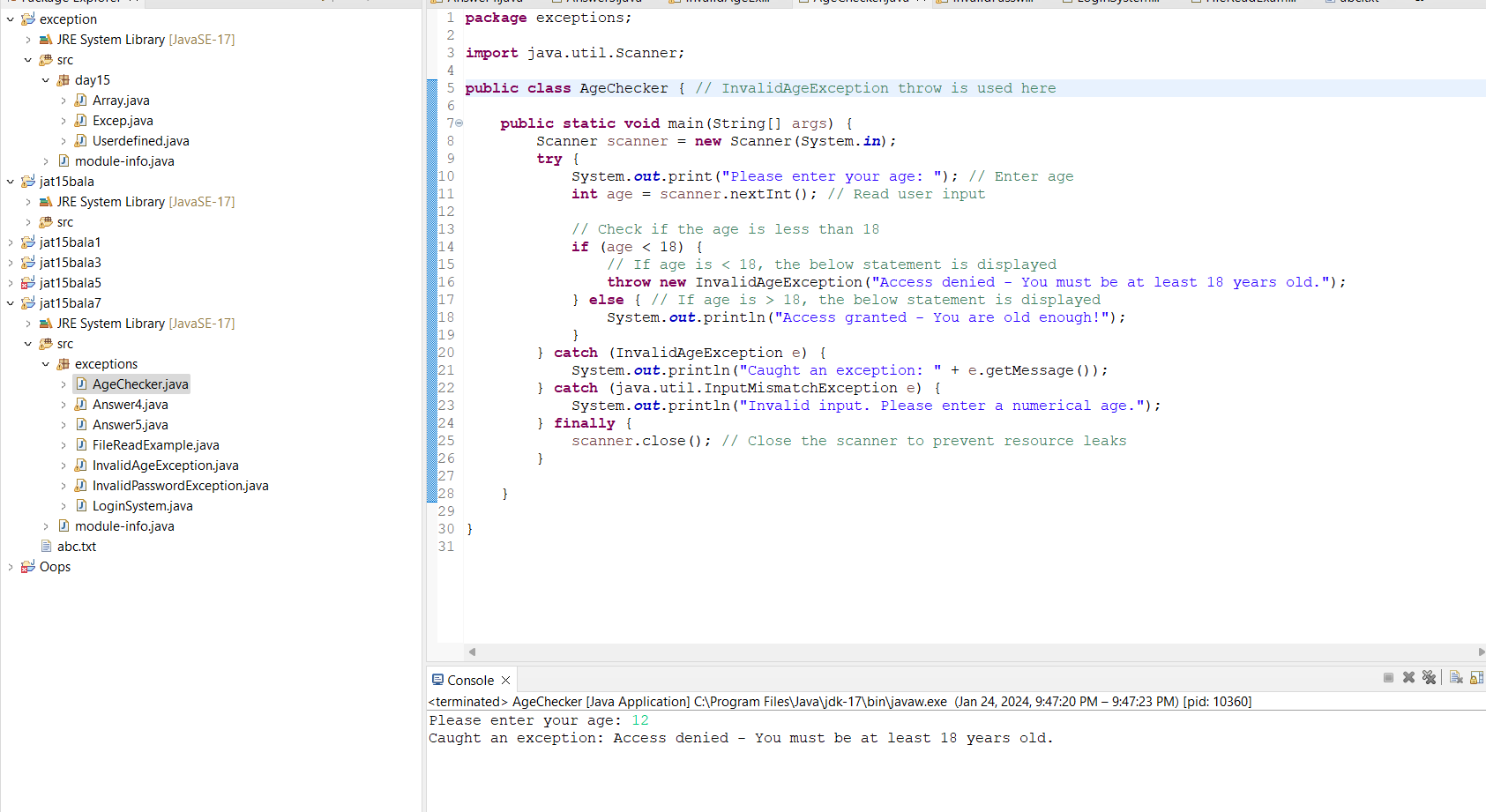
**Answer 5 :** Output Screenshot of Answer5.java



**Answer 6 :** Output Screenshots of LoginSystem.java



**Answer 7 :** Output Screenshots of AgeChecker.java



**Answer 8 :** Output Screenshot of FileReadExample.java

