Task-11

1)In Java, there are four access modifiers used to control the visibility and accessibility of classes, methods, and variables. They are:

**a)Public**: The **public** access modifier allows the class, method, or variable to be accessed from any other class in the same package or from any other package. Public members are widely accessible.

**b)Protected**: The **protected** access modifier allows the class, method, or variable to be accessed from within the same package, as well as by subclasses (regardless of the package). Protected members are accessible within the same package and by subclasses, even if they are in a different package.

**c)Default (Package-private)**: If no access modifier is specified (i.e., no modifier is used), it is known as the default access level or package-private access. This means that the class, method, or variable is accessible only within the same package. It is not accessible outside the package, even to subclasses.

**d)Private**: The **private** access modifier restricts the visibility of the class, method, or variable to within the same class only. Private members are not accessible from outside the class, not even by subclasses.

Here's a summary of the accessibility of members based on their access modifiers:

* **Public**: Accessible from anywhere (same class, same package, different package).
* **Protected**: Accessible from the same class, same package, and subclasses (regardless of the package).
* **Default (Package-private)**: Accessible only within the same package.
* **Private**: Accessible only within the same class.

Top of Form

2)In Java, both exceptions and errors are types of Throwable objects, but they serve different purposes and have different implications:

**a)Exceptions**:

* + Exceptions represent exceptional conditions that occur during the execution of a program, but from which the program can potentially recover.
  + Exceptions are typically caused by issues that occur at runtime, such as invalid input, file not found, network failure, etc.
  + Exceptions are intended to be caught and handled by the program. They can be anticipated and handled through try-catch blocks or propagated up the call stack using throws clauses.
  + Examples of exceptions include **IOException**, **NullPointerException**, **ArrayIndexOutOfBoundsException**, **NumberFormatException**, etc.

**b)Errors**:

* + Errors represent serious problems that occur at runtime and usually indicate problems that are beyond the control of the program.
  + Errors are typically caused by issues that are difficult or impossible to recover from, such as JVM-related issues, out-of-memory errors, stack overflow, etc.
  + Errors are not intended to be caught and handled by the program. Attempting to catch errors is discouraged because they usually indicate serious problems that may render the program in an unstable state.
  + Examples of errors include **OutOfMemoryError**, **StackOverflowError**, **NoClassDefFoundError**, **VirtualMachineError**, etc.

In summary, exceptions are used to handle exceptional conditions that occur during the normal execution of a program and can be recovered from, while errors indicate serious problems that usually cannot be recovered from and may require intervention at the system level.

Top of Form

Bottom of Form

Bottom of Form

3)The difference between checked and unchecked exceptions in Java lies primarily in their handling requirements:

**a)Checked Exceptions**:

* + Checked exceptions are exceptions that are checked at compile time by the compiler.
  + These exceptions are subclasses of **Exception** (excluding **RuntimeException** and its subclasses) but not subclasses of **RuntimeException**.
  + Checked exceptions represent conditions that a well-behaved program should anticipate and handle, such as I/O errors, network errors, etc.
  + Methods that may throw checked exceptions must declare them in their **throws** clause, or handle them using a try-catch block.
  + Examples of checked exceptions include **IOException**, **SQLException**, **FileNotFoundException**, etc.

**b)Unchecked Exceptions**:

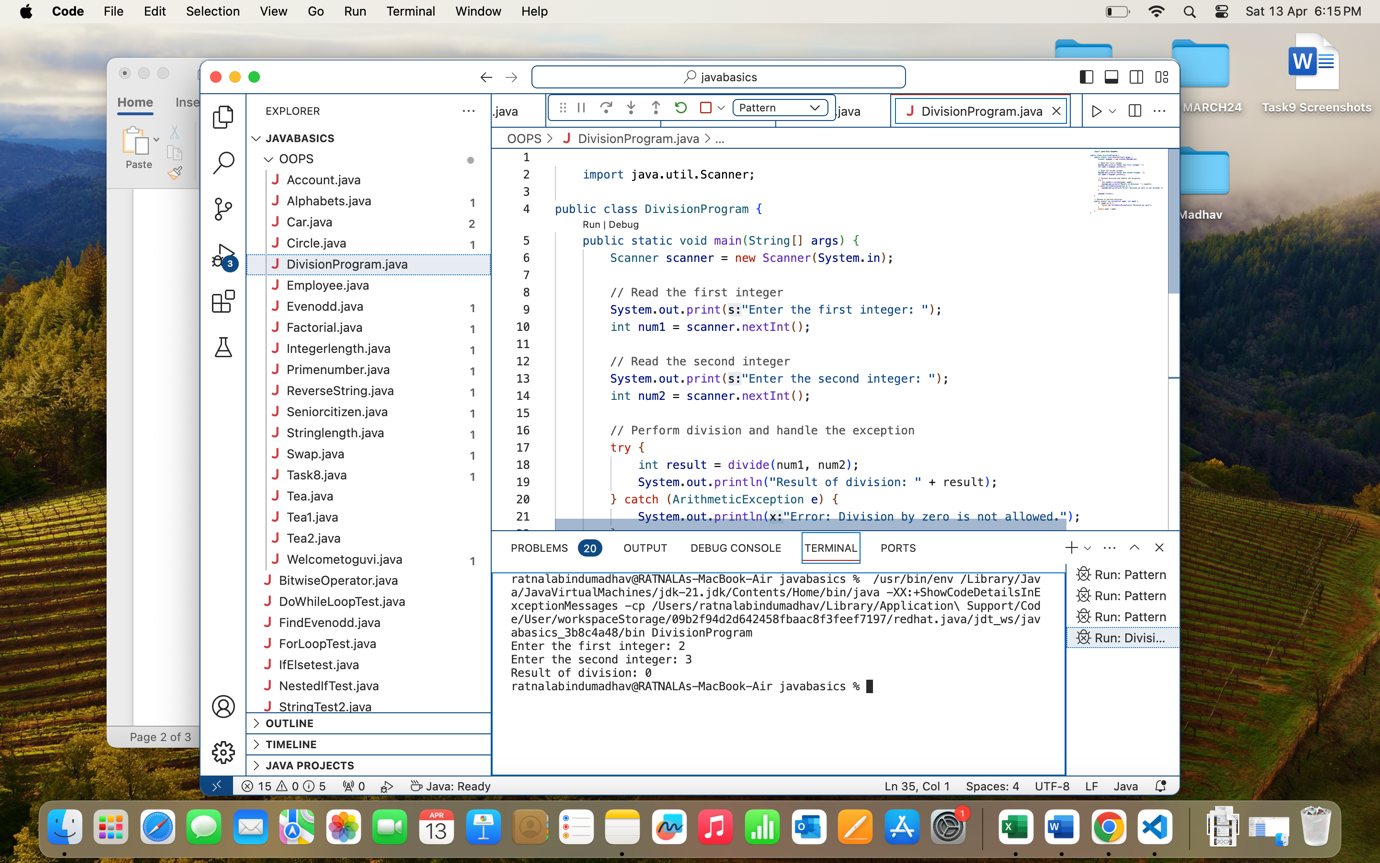
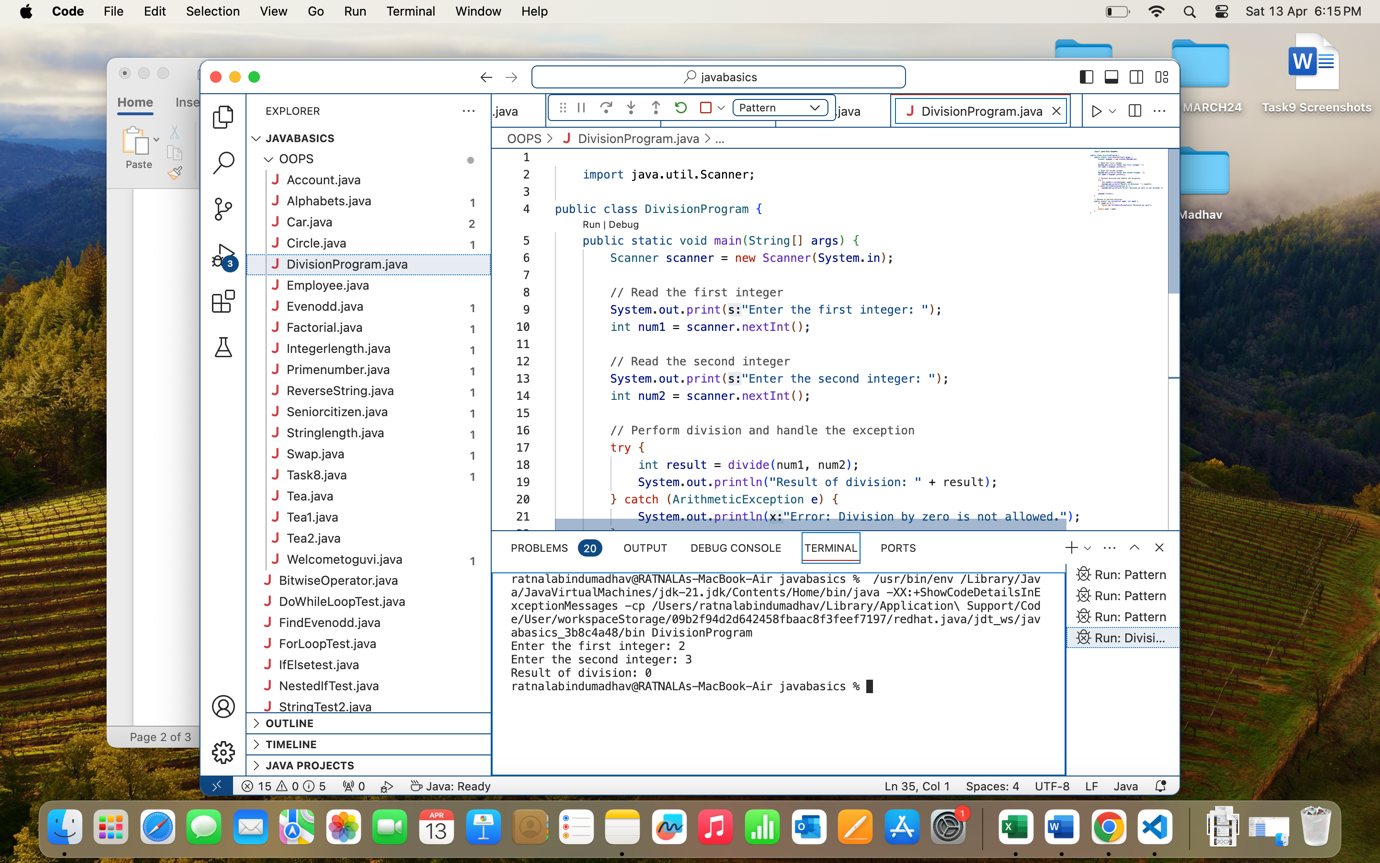
* + Unchecked exceptions are exceptions that are not checked at compile time by the compiler.
  + These exceptions are subclasses of **RuntimeException** and its subclasses.
  + Unchecked exceptions typically represent programming errors or conditions that are not recoverable, such as null pointer dereference, array index out of bounds, etc.
  + Methods are not required to declare unchecked exceptions in their **throws** clause or handle them using a try-catch block.
  + Examples of unchecked exceptions include **NullPointerException**, **ArrayIndexOutOfBoundsException**, **IllegalArgumentException**, etc.

In summary, the key differences between checked and unchecked exceptions are:

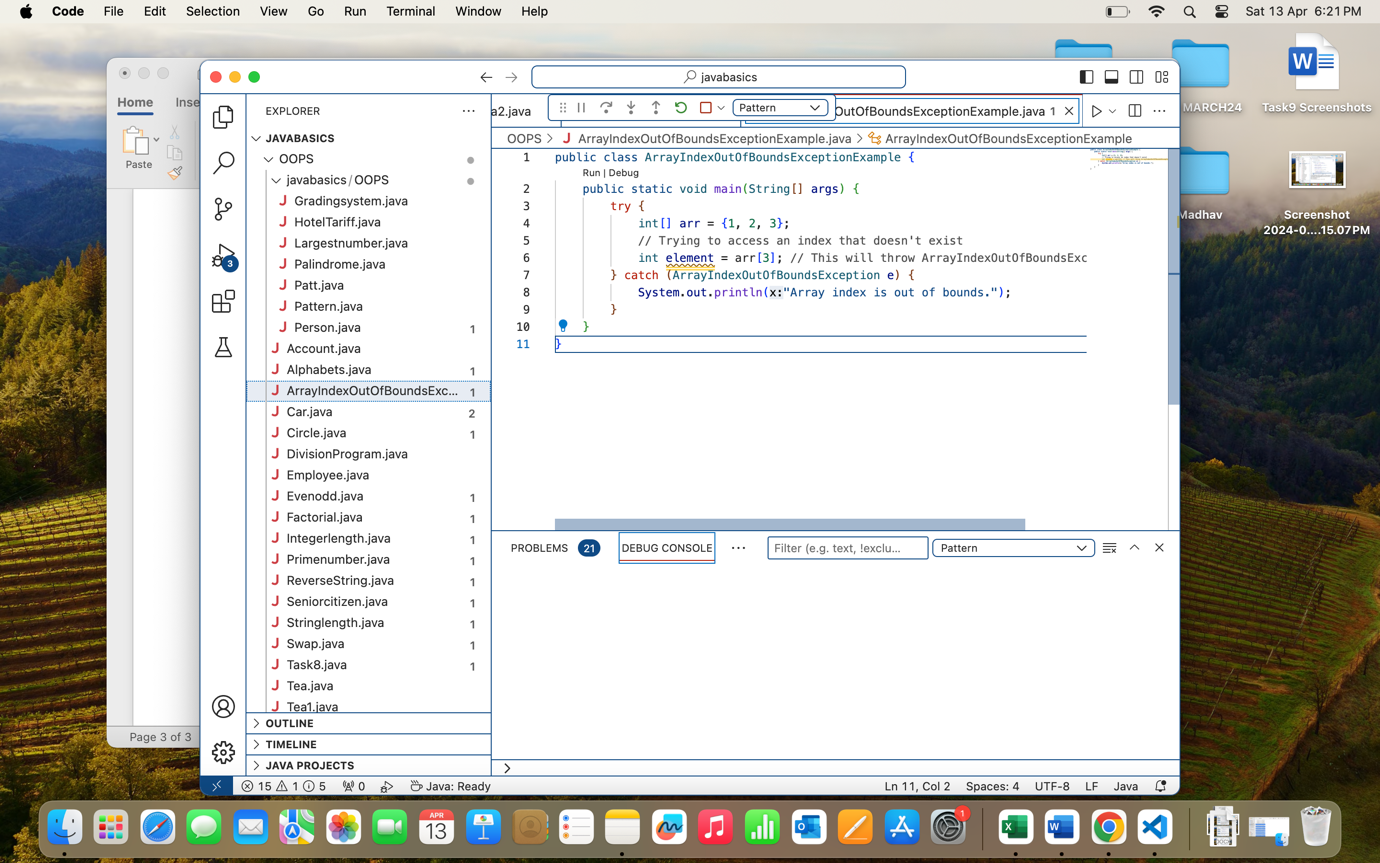
* Checked exceptions are checked at compile time and must be declared or handled, while unchecked exceptions are not checked at compile time and do not need to be declared or handled explicitly.
* Checked exceptions are subclasses of **Exception** (excluding **RuntimeException** and its subclasses), while unchecked exceptions are subclasses of **RuntimeException** and its subclasses.

Top of Form

4) Screenshot of 4th ProgramBottom of Form



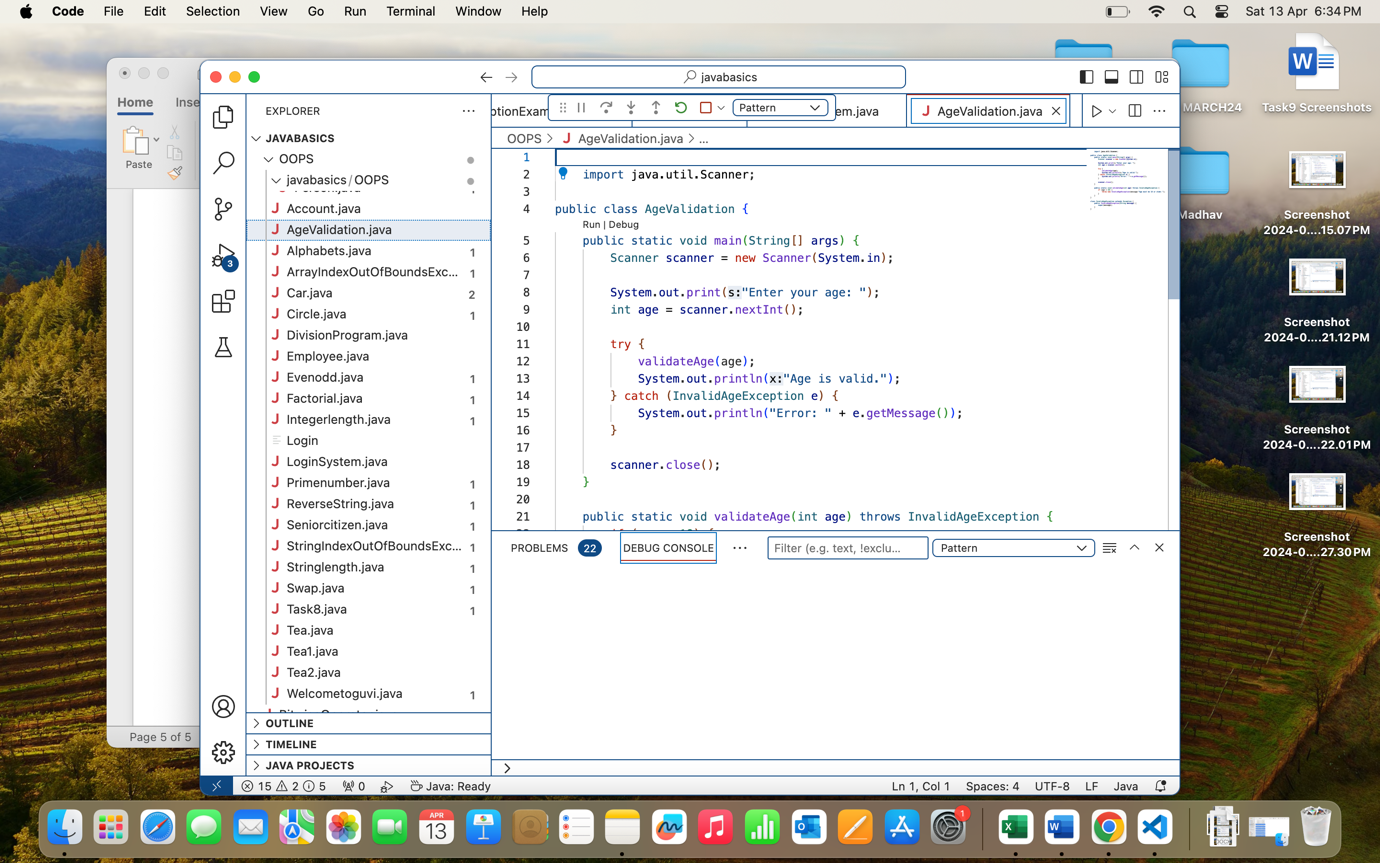
5)Screenshots of the code



A computer screen shot of a computer

Description automatically generated

6) Screenshot of the code



Explanation:

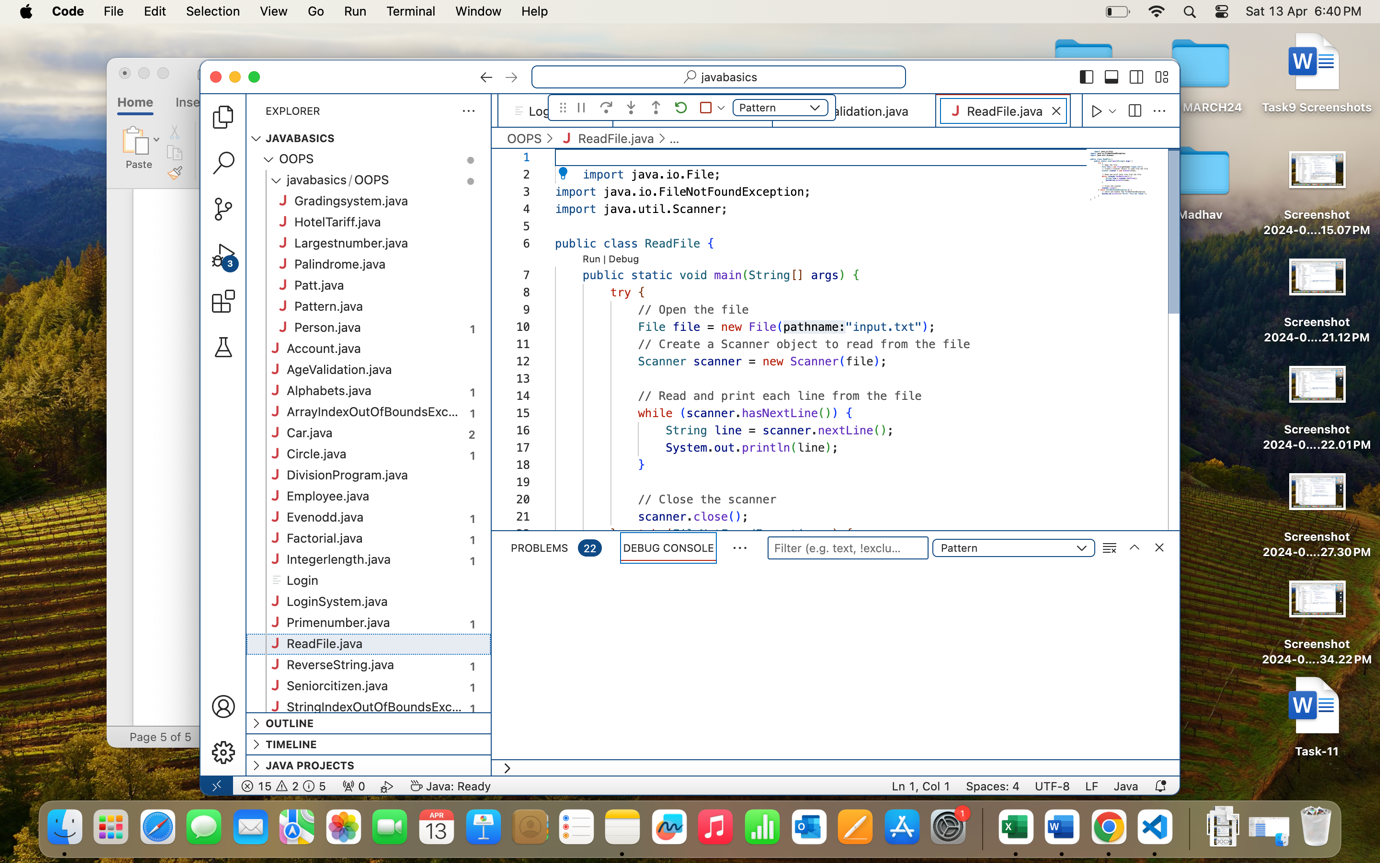
* We define a **AgeValidation** class that contains the main method.
* Inside the main method, we prompt the user to enter their age and read the input using a **Scanner**.
* We then call the **validateAge** method with the entered age as an argument.
* The **validateAge** method checks if the entered age is less than 18. If it is, it throws an **InvalidAgeException** with a message indicating that the age must be 18 or older.
* In the main method, we catch the **InvalidAgeException** and display the error message using **getMessage()** method.

This way, using exception handling, we can catch the **InvalidAgeException** and display an appropriate error message when the user enters an age less than 18.

Top of Form

Bottom of Form

8) Screenshot of code



Explanation:

* We use a try-catch block to handle the potential FileNotFoundException that may occur when attempting to open the file.
* Inside the try block, we attempt to open the file using the File object and create a Scanner object to read from the file.
* We then use a while loop to read each line from the file and print it to the console.
* If the FileNotFoundException occurs (i.e., if the file does not exist), control is transferred to the catch block where we catch the exception and display an error message to the user.

Top of Form

Bottom of Form