# 1.Access Modifiers in Java :

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers.**

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

### **1) Private**

The private access modifier is accessible only within the class.

Example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.

1. **class A{**
2. **private int data=40;**
3. **private void msg(){System.out.println("Hello java");}**
4. **}**
6. **public class Simple{**
7. **public static void main(String args[]){**
8. **A obj=new A();**
9. **System.out.println(obj.data);//Compile Time Error**
10. **obj.msg();//Compile Time Error**
11. **}**
12. **}**

### **2) Default**

If you don't use any modifier, it is treated as default by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public.

**Example of default access modifier**

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

1. //save by A.java
2. package pack;
3. class A{
4. void msg(){System.out.println("Hello");}
5. }
6. //save by B.java
7. package mypack;
8. import pack.\*;
9. class B{
10. public static void main(String args[]){
11. A obj = new A();//Compile Time Error
12. obj.msg();//Compile Time Error
13. }
14. }

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

### **3) Protected**

The protected access modifier is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

It provides more accessibility than the default modifer.

**Example of protected access modifier**

**//save by A.java**

1. **package pack;**
2. **public class A{**
3. **protected void msg(){System.out.println("Hello");}**
4. **}**
5. **//save by B.java**
6. **package mypack;**
7. **import pack.\*;**
9. **class B extends A{**
10. **public static void main(String args[]){**
11. **B obj = new B();**
12. **obj.msg();**
13. **}**
14. **}**

**OUTPUT : HELLO**

### **4) Public**

**The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.**

**Example of public access modifier**

1. **//save by A.java**
3. **package pack;**
4. **public class A{**
5. **public void msg(){System.out.println("Hello");}**
6. **}**
7. **//save by B.java**
9. **package mypack;**
10. **import pack.\*;**
12. **class B{**
13. **public static void main(String args[]){**
14. **A obj = new A();**
15. **obj.msg();**
16. **}**
17. **}**

**OUTPUT : Hello**

**2.Difference between Exception and Error :**

In Java, errors and exceptions are both types of throwable objects, but they represent different types of problems that can occur during the execution of a program.

Errors are usually caused by serious problems that are outside the control of the program, such as running out of memory or a system crash. Errors are represented by the Error class and its subclasses. Some common examples of errors in Java include:

* OutOfMemoryError: Thrown when the Java Virtual Machine (JVM) runs out of memory.
* StackOverflowError: Thrown when the call stack overflows due to too many method invocations.
* NoClassDefFoundError: Thrown when a required class cannot be found.

Since errors are generally caused by problems that cannot be recovered from, it’s usually not appropriate for a program to catch errors. Instead, the best course of action is usually to log the error and exit the program.

Exceptions, on the other hand, are used to handle errors that can be recovered from within the program. Exceptions are represented by the Exception class and its subclasses.

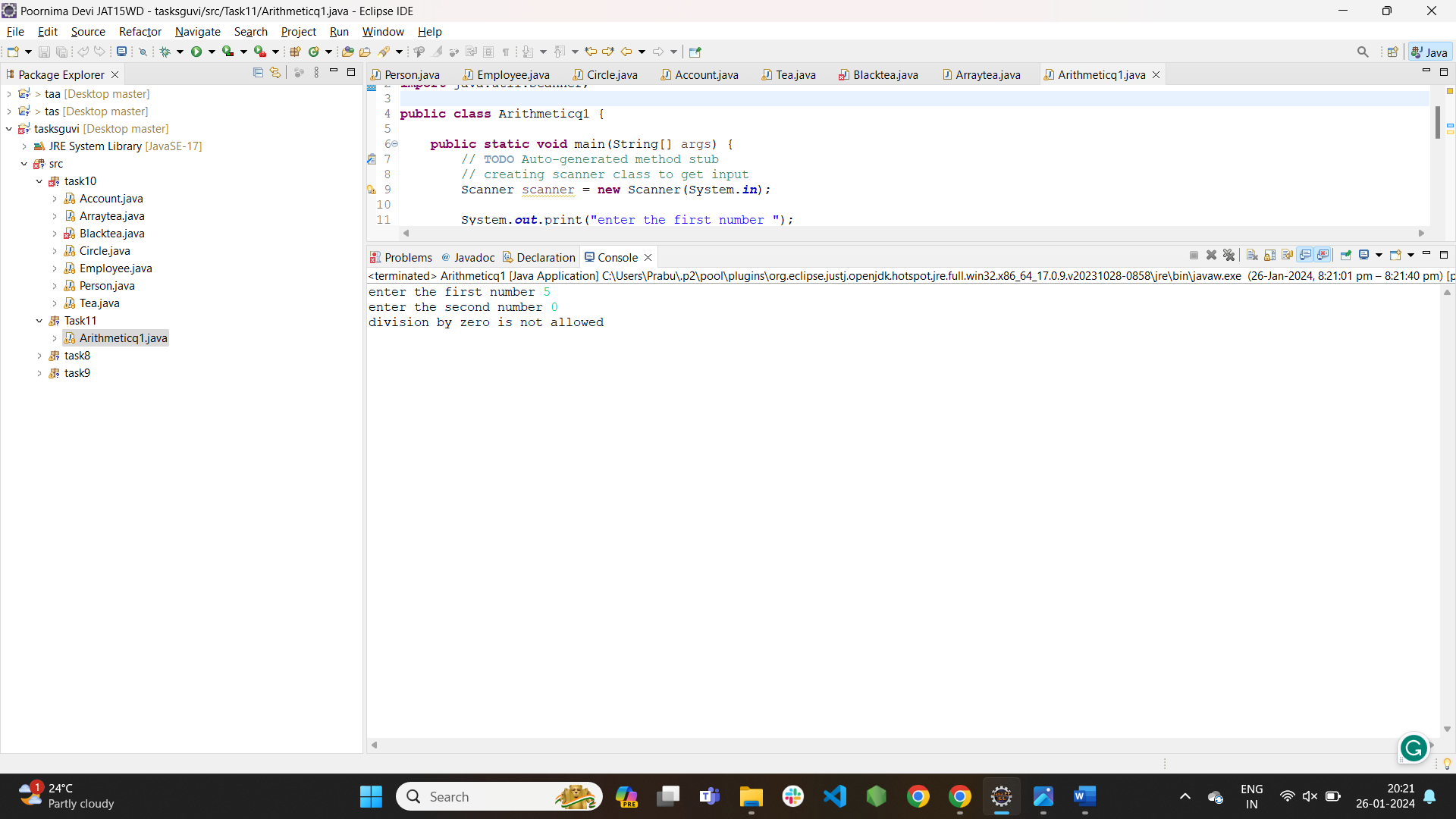
| **S.NO** | **EXCEPTION** | **ERROR** |
| --- | --- | --- |
| **01** | Exception and error are subclasses of Java throwable class. | Error represents different types of problem that occur during execution of program. |
| **02** | Recovering from exception is possible. | Recovering from error is impossible. |
| **03** | Exceptions are checked and unchecked in java. | Errors are unchecked type in java. |
| **04** | Example- IOException, Arithmetic exception etc | Example- StackOverflowError, Memory error etc |
| **05** | Exceptions are caused by the program itself. | Errors are caused by the environment its running. |
| **06** | It occurs at run time. | It occurs at compile time. |
| **07** | They are defined in java.lang.exception package. | They are defined in java.lang error package |
| **08** | This class represents concerning conditions. | This class represents critical conditions. |
| **09** | It is possible to recover from the error using try ..catch block | It is not possible to recover from this error. |
| **10** | Exception compiler has knowledge and errors and force to keep try and catch block. | Error compiler won't have knowledge of errors. |

**3. Difference between Checked Exception and Un-Checked Exception :**

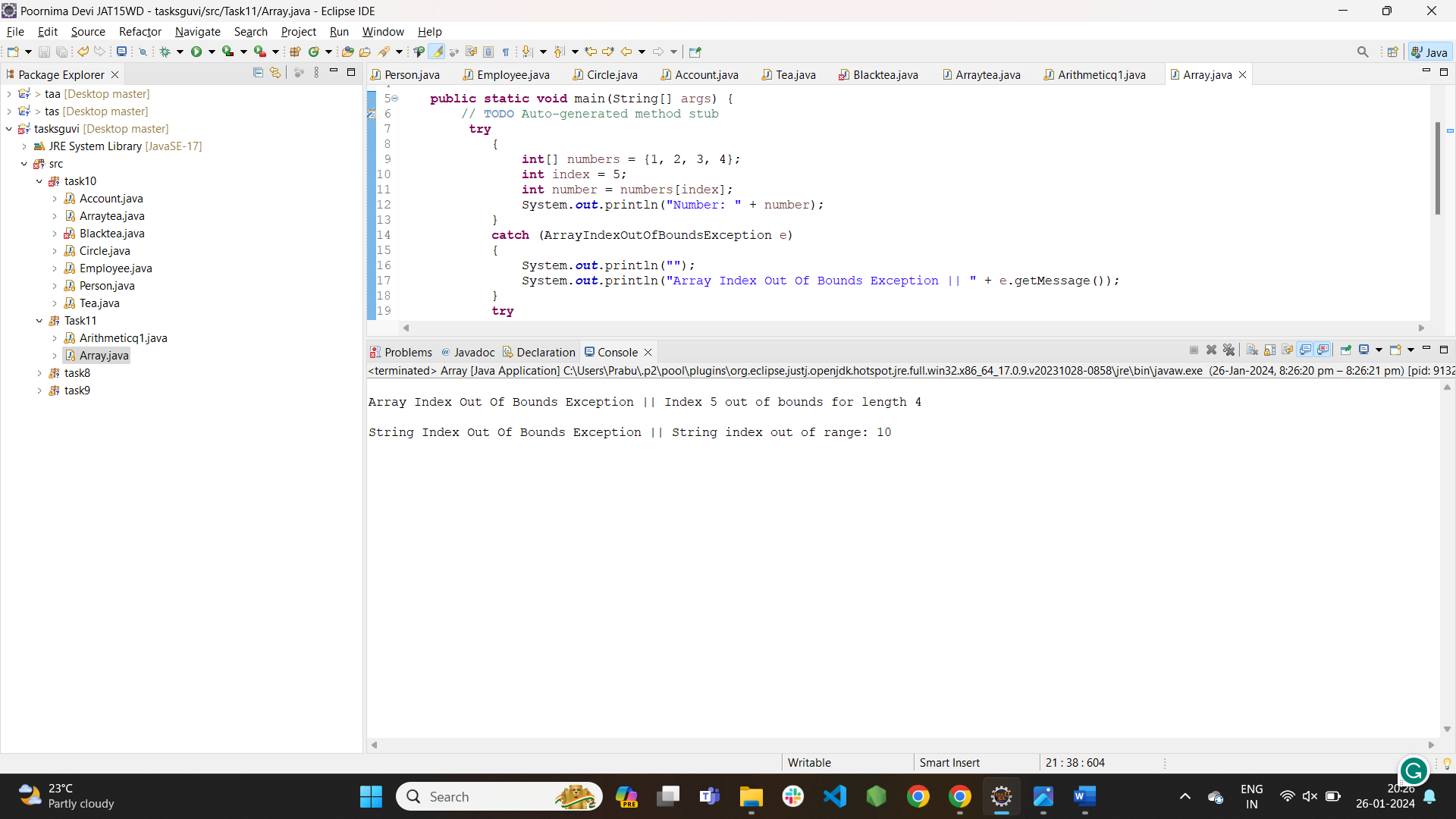
| Checked Exception | Unchecked Exception |
| --- | --- |
| **Checked exceptions occur at compile time.** | **Unchecked exceptions occur at runtime.** |
| **The compiler checks a checked exception.** | **The compiler does not check these types of exceptions.** |
| **These types of exceptions can be handled at the time of compilation.** | **These types of exceptions cannot be a catch or handle at the time of compilation, because they get generated by the mistakes in the program.** |
| **They are the sub-class of the exception class.** | **They are runtime exceptions and hence are not a part of the Exception class.** |
| **Here, the JVM needs the exception to catch and handle.** | **Here, the JVM does not require the exception to catch and handle.** |
| **Examples of Checked exceptions:**   * **File Not Found Exception** * **No Such Field Exception** * **Interrupted Exception** * **No Such Method Exception** * **Class Not Found Exception** | **Examples of Unchecked Exceptions:**   * **No Such Element Exception** * **Undeclared Throwable Exception** * **Empty Stack Exception** * **Arithmetic Exception** * **Null Pointer Exception** * **Array Index Out of Bounds Exception** * **Security Exception** |

# 

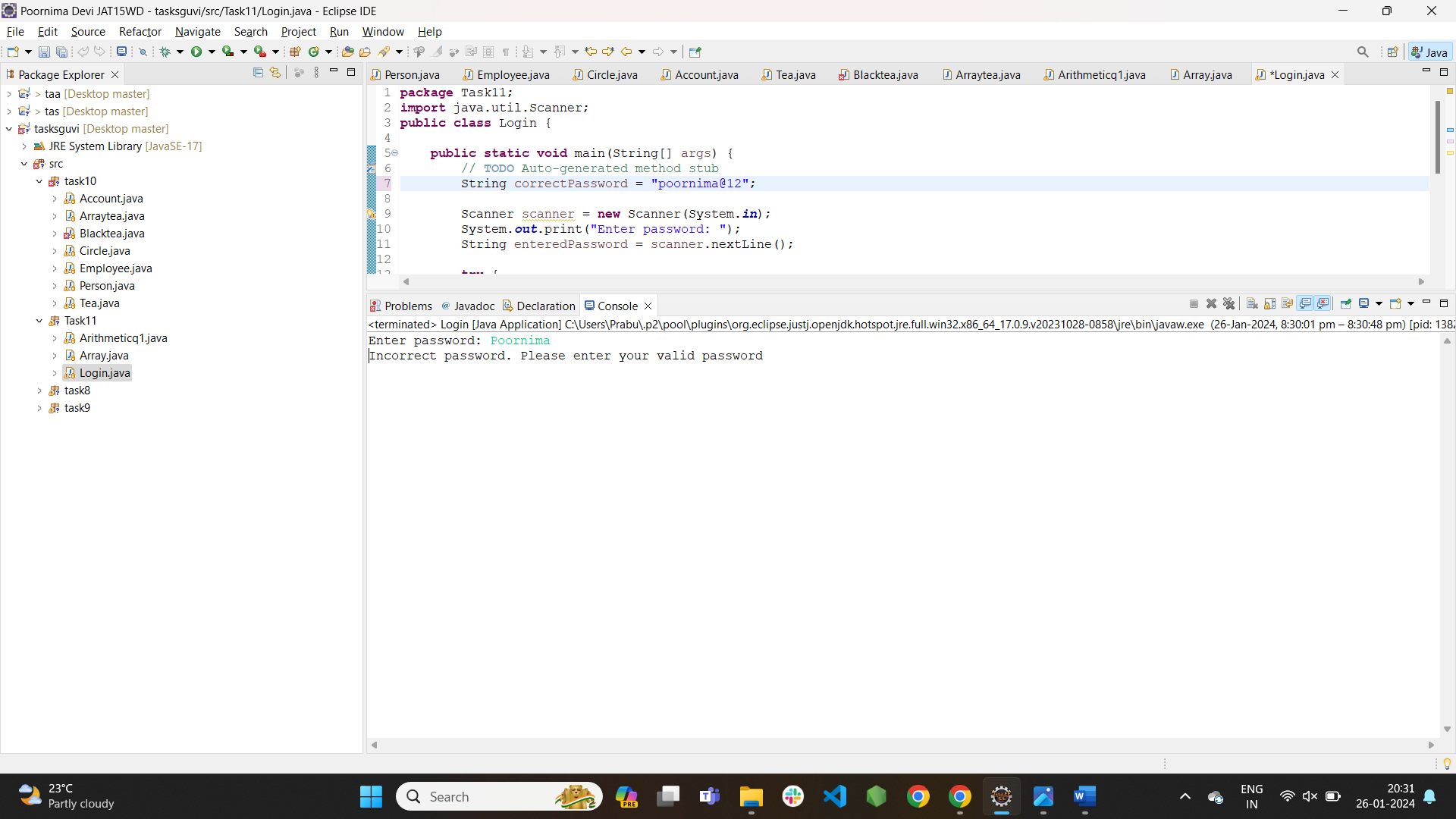
Q4. OUTPUT :



Q5 OUTPUT :



Q6 OUTPUT :



Q8 OUTPUT :

