**Practical No 6**

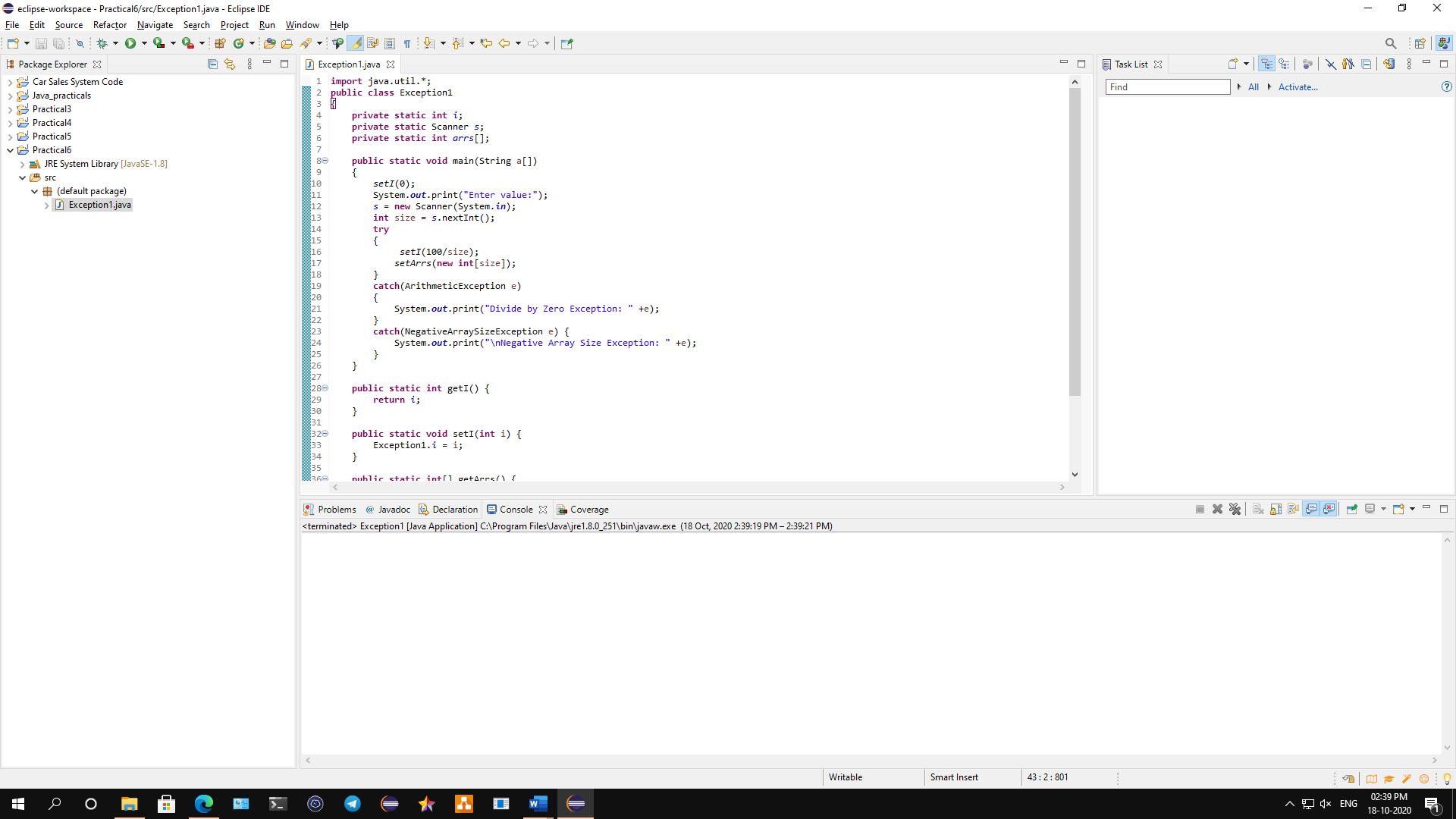
**A)** **To implement a program to demonstrate exceptions for negative array size and divide by zero etc.**

**Aim: Write a program to demonstrate exceptions for negative array size and divide by zero etc.**

**Description:**

An exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled. Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc. There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions: Checked Exception, Unchecked Exception and Error.

In this program we have defined the scanner class and int i for value and int arrs for exception classes. Then we defined the main method. After that we use defined the size of array if the array is more than 100 it will throw exceptions such as zero or if we get negative value it will throw negative array size expectation.



**Conclusion: We have written a program to demonstrate exceptions for negative array size and divide by zero etc.**

**Code:**

**import** java.util.\*;

**public** **class** Exception1

{

**private** **static** **int** *i*;

**private** **static** Scanner *s*;

**private** **static** **int** *arrs*[];

**public** **static** **void** main(String a[])

{

*setI*(0);

System.***out***.print("Enter value:");

*s* = **new** Scanner(System.***in***);

**int** size = *s*.nextInt();

**try**

{

*setI*(100/size);

*setArrs*(**new** **int**[size]);

}

**catch**(ArithmeticException e)

{

System.***out***.print("Divide by Zero Exception: " +e);

}

**catch**(NegativeArraySizeException e) {

System.***out***.print("\nNegative Array Size Exception: " +e);

}

}

**public** **static** **int** getI()

{

**return** *i*;

}

**public** **static** **void** setI(**int** i)

{

Exception1.*i* = i;

}

**public** **static** **int**[] getArrs()

{

**return** *arrs*;

}

**public** **static** **void** setArrs(**int** arrs[])

{

Exception1.*arrs* = arrs;

}

}

**Output:**

**Negative array size exception:**



**Divide by zero exceptions:**



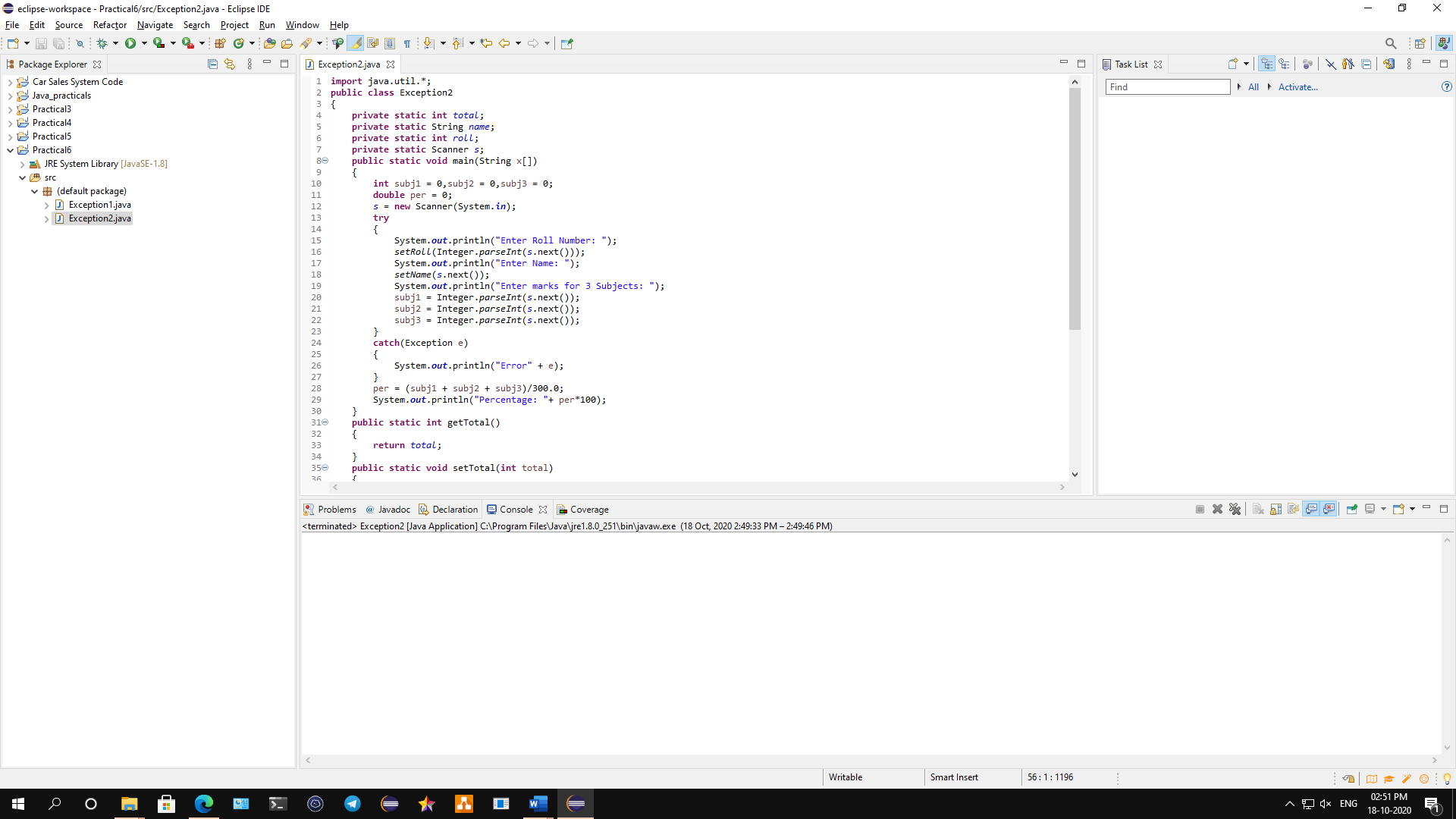
**B)** **To implement a program to demonstrate** **NumberFormatException: Enter Roll No, Name, Marks and Subject from command line and calculate percentage.**

**Aim: Write a java program to implement a program to demonstrate** **NumberFormatException: Enter Roll No, Name, Marks and Subject from command line and calculate percentage.**

**Description:**

An exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled. There are three types of exceptions: Checked Exception, Unchecked Exception and Error.

In this program we use private class to define total, name and roll number, then we used the main functions. After that we defined the int for subject, after than we have defined the try keyword to try the exception if the exception is found in the code the error will throw and the program will terminated, if there are no error the code will compile and output will be displayed.



**Conclusion: We have written a program to implement a program to demonstrate** **NumberFormatException: Enter Roll No, Name, Marks and Subject from command line and calculate percentage.**

**Code:**

**import** java.util.\*;

**public** **class** Exception2

{

**private** **static** **int** *total*;

**private** **static** String *name*;

**private** **static** **int** *roll*;

**private** **static** Scanner *s*;

**public** **static** **void** main(String x[])

{

**int** subj1 = 0,subj2 = 0,subj3 = 0;

**double** per = 0;

*s* = **new** Scanner(System.***in***);

**try**

{

System.***out***.println("Enter Roll Number: ");

*setRoll*(Integer.*parseInt*(*s*.next()));

System.***out***.println("Enter Name: ");

*setName*(*s*.next());

System.***out***.println("Enter marks for 3 Subjects: ");

subj1 = Integer.*parseInt*(*s*.next());

subj2 = Integer.*parseInt*(*s*.next());

subj3 = Integer.*parseInt*(*s*.next());

}

**catch**(Exception e)

{

System.***out***.println("Error" + e);

}

per = (subj1 + subj2 + subj3)/300.0;

System.***out***.println("Percentage: "+ per\*100);

}

**public** **static** **int** getTotal()

{

**return** *total*;

}

**public** **static** **void** setTotal(**int** total)

{

Exception2.*total* = total;

}

**public** **static** String getName()

{

**return** *name*;

}

**public** **static** **void** setName(String name)

{

Exception2.*name* = name;

}

**public** **static** **int** getRoll()

{

**return** *roll*;

}

**public** **static** **void** setRoll(**int** roll)

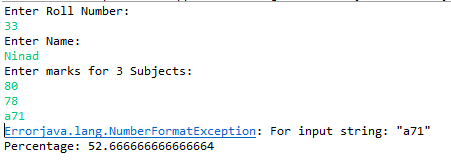
{

Exception2.*roll* = roll;

}

}

**Output:**

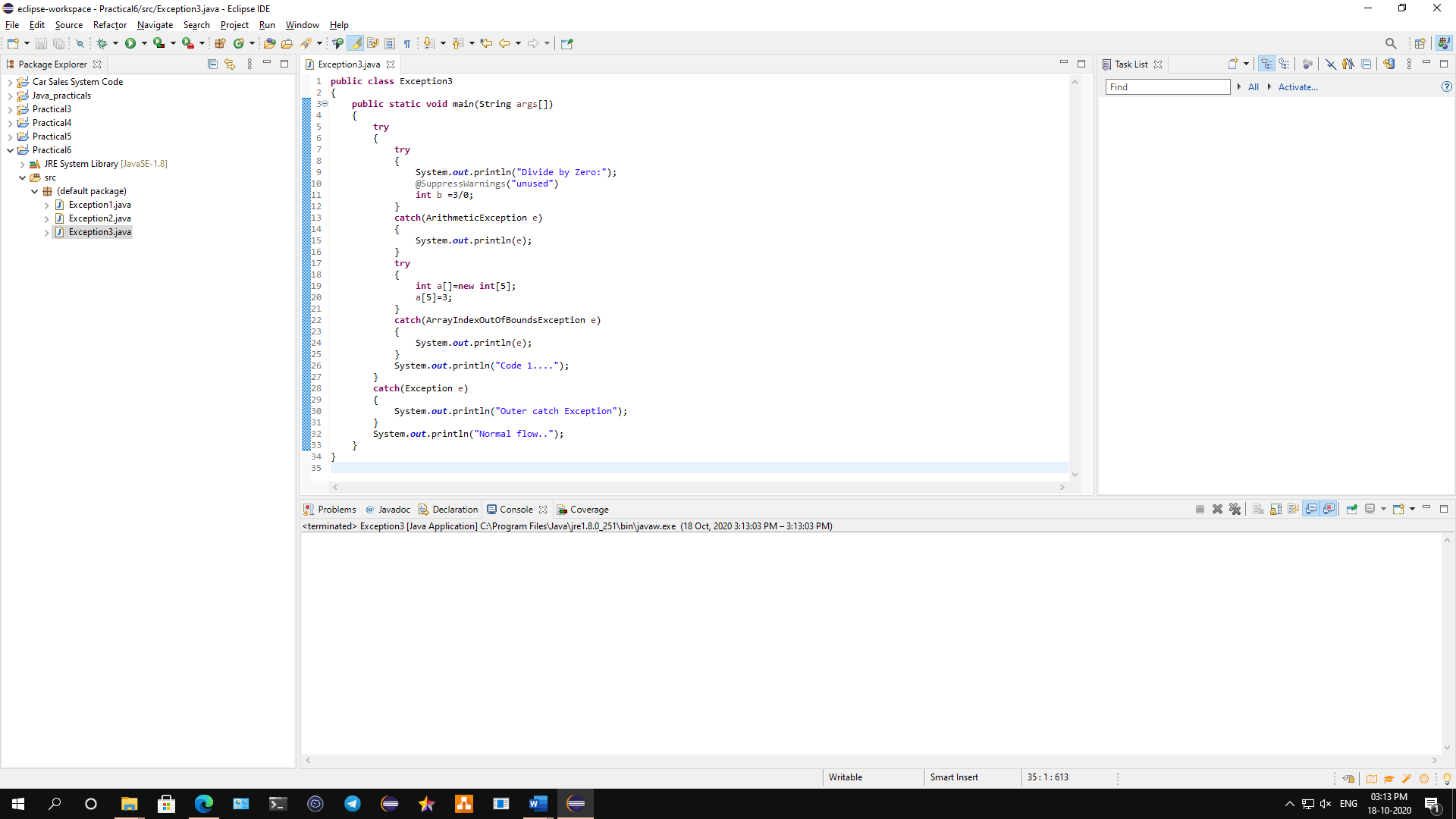


**C) To implement a program to implement nested try, multiply catch and finally.**

**Aim: Write a java program to implement nested try, multiply catch and finally.**

**Description:**

There are 5 keywords which are used in handling exceptions in Java. They are Try, Catch, Finally, Throw, Throws. The “try” keyword is used to specify a block where we should place exception code. The “catch” block is used to handle the exception. The "finally" block is used to execute the important code of the program. The “throw” keyword is used to throw an exception. The "throws" keyword is used to declare exceptions. In this program we defined the public class followed by the main method. Then we use try block to specify the block of exception divided by zero. Then we use catch block to handle the exception. Again, we define try block for specifying the new block of exception. The catch block will use to handle that exception for code 1. And it will catch that exception in the last one and then output will show.



**Conclusion: We have implemented a program on nested try, multiply catch and finally.**

**Code:**

**public** **class** Exception3

{

**public** **static** **void** main(String args[])

{

**try**

{

**try**

{

System.***out***.println("Divide by Zero:");

@SuppressWarnings("unused")

**int** b =3/0;

}

**catch**(ArithmeticException e)

{

System.***out***.println(e);

}

**try**

{

**int** a[]=**new** **int**[5];

a[5]=3;

}

**catch**(ArrayIndexOutOfBoundsException e)

{

System.***out***.println(e);

}

System.***out***.println("Code 1....");

}

**catch**(Exception e)

{

System.***out***.println("Outer catch Exception");

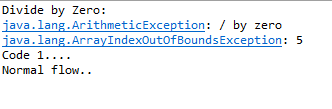
}

System.***out***.println("Normal flow..");

}

}

**Output:**

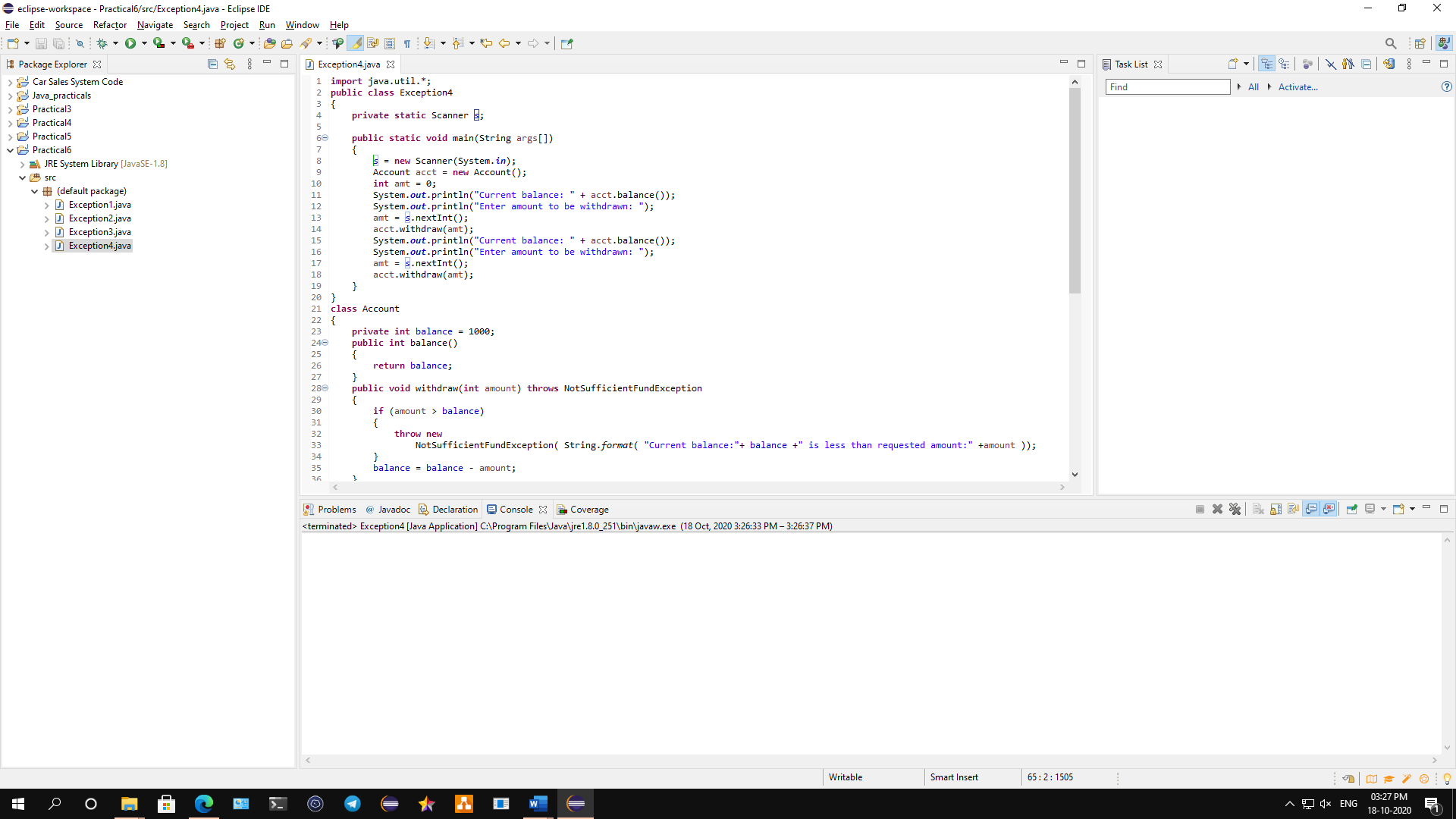


**D) To implement a program to create user define exception. Create class Bank and define methods open (), deposit() and withdraw() with minimum balance 500. Create an exception Payoutofbounds and fire exception.**

**Aim: Write a java program to implement a program to create user define exception. Create class Bank and define methods open (), deposit() and withdraw() with minimum balance 500. Create an exception Payoutofbounds and fire exception.**

**Description:**

There are 5 keywords which are used in handling exceptions in Java. They are Try, Catch, Finally, Throw, Throws. The “try” keyword is used to specify a block where we should place exception code. The “catch” block is used to handle the exception. The "finally" block is used to execute the important code of the program. The “throw” keyword is used to throw an exception. The "throws" keyword is used to declare exceptions. In this program we defined the public class followed by the main method and also the scanner class. Then we define account and int amount to get the output of current account balance. We use another class as Account which will read the current balance as 1000 and we use throws keyword to declare as exceptions if the balance is more than the remaining balance it will throw the exception. If the balance is less than the remaining balance it will display the output.



**Conclusion: We have implemented a program to create user define exception. Create class Bank and define methods open (), deposit() and withdraw() with minimum balance 500. Create an exception Payoutofbounds and fire exception.**

**Code:**

**import** java.util.\*;

**public** **class** Exception4

{

**private** **static** Scanner *s*;

**public** **static** **void** main(String args[])

{

*s* = **new** Scanner(System.***in***);

Account acct = **new** Account();

**int** amt = 0;

System.***out***.println("Current balance: " + acct.balance());

System.***out***.println("Enter amount to be withdrawn: ");

amt = *s*.nextInt();

acct.withdraw(amt);

System.***out***.println("Current balance: " + acct.balance());

System.***out***.println("Enter amount to be withdrawn: ");

amt = *s*.nextInt();

acct.withdraw(amt);

}

}

**class** Account

{

**private** **int** balance = 1000;

**public** **int** balance()

{

**return** balance;

}

**public** **void** withdraw(**int** amount) **throws** NotSufficientFundException

{

**if** (amount > balance)

{

**throw** **new**

NotSufficientFundException( String.*format*( "Current balance:"+ balance +" is less than requested amount:" +amount ));

}

balance = balance - amount;

}

**public** **void** deposit(**int** amount)

{

**if** (amount <= 0)

{

**throw** **new** IllegalArgumentException(String.*format*("Invalid deposit amount:"+ amount));

}

}

}

**class** NotSufficientFundException **extends** RuntimeException

{

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** String message;

**public** NotSufficientFundException(String message)

{

**this**.message = message;

}

**public** NotSufficientFundException(Throwable cause, String message)

{

**super**(cause);

**this**.message = message;

}

**public** String getMessage()

{

**return** message;

}

}

**Output:**

