**Batch: B3 Roll No.: 121**

**Experiment / assignment / tutorial No.07**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE : User Defined Exception** |

**AIM:**

Create a user defined exception subclass TimeException with necessary constructors and overridden toString method. Write a program which accepts two integers with time in minutes and seconds and find the sum. It throws an object of the TimeException class if the value exceeds 60seconds otherwise it displays the total time. On printing, the exception object should display an exception name, appropriate message for exception.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

**CO1:** Understand the features of object oriented programming compared with procedural approach with C++ and Java

**CO4:**Explore the interface, exceptions, multithreading, packages **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1.Ralph Bravaco , Shai Simoson , “Java Programming From the Group Up” Tata McGraw-Hill.

2.Grady Booch, Object Oriented Analysis and Design.

3. <https://www.geeksforgeeks.org/user-defined-custom-exception-in-java/>

4. <https://www.javatpoint.com/finally-block-in-exception-handling>

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

**Exception handling** in java is a powerful mechanism or technique that allows us to handle runtime errors in a program so that the normal flow of the program can be maintained. All the exceptions occur only at runtime. A syntax error occurs at compile time.

**Exception in Java:**

In general, an exception means a problem or an abnormal condition that stops a computer program from processing information in a normal way.

An exception in java is an object representing an error or an abnormal condition that occurs at runtime execution and interrupts (disrupts) the normal execution flow of the program.

An exception can be identified only at runtime, not at compile time. Therefore, it is also called runtime errors that are thrown as exceptions in Java. They occur while a program is running.

For example:

* If we access an array using an index that is out of bounds, we will get a runtime error named ArrayIndexOutOfBoundsException.
* If we enter a double value while the program is expecting an integer value, we will get a runtime error called InputMismatchException.

When JVM faces these kinds of errors or dividing an integer by zero in a program, it creates an exception object and throws it to inform us that an error has occurred.If the exception object is not caught and handled properly, JVM will display an error message and will terminate the rest of the program abnormally.

If we want to continue the execution of remaining code in the program, we will have to handle exception objects thrown by error conditions and then display a user-friendly message for taking corrective actions. This task is known as exception handling in java.

**Types of Exceptions in Java**

Basically, there are two types of exceptions in java API. They are:

1. Predefined Exceptions (Built-in-Exceptions)

2. Custom (User defined)Exceptions

**Predefined Exceptions:**

Predefined exceptions are those exceptions that are already defined by the Java system. These exceptions are also called built-in-exceptions.Java API supports exception handling by providing the number of predefined exceptions. These predefined exceptions are represented by classes in java.

When a predefined exception occurs, JVM (Java runtime system) creates an object of predefined exception class. All exceptions are derived from java.lang.Throwable class but not all exception classes are defined in the same package. All the predefined exceptions supported by java are organized as subclasses in a hierarchy under the Throwable class.

All the predefined exceptions are further divided into two groups:

1. Checked Exceptions: Checked exceptions are those exceptions that are checked by the java compiler itself at compilation time and are not under runtime exception class hierarchy. If a method throws a checked exception in a program, the method must either handle the exception or pass it to a caller method.

2. Unchecked Exceptions: Unchecked exceptions in Java are those exceptions that are checked by JVM, not by java compiler. They occur during the runtime of a program. All exceptions under the runtime exception class are called unchecked exceptions or runtime exceptions in Java.

**Custom exceptions:**

Custom exceptions are those exceptions that are created by users or programmers according to their own needs. The custom exceptions are also called user-defined exceptions that are created by extending the exception class.

So, Java provides the liberty to programmers to throw and handle exceptions while dealing with functional requirements of problems they are solving.

**Exception Handling Mechanism using Try-Catch block:**

The general syntax of try-catch block (exception handling block) is as follows:

**Syntax:**

try

{

// A block of code; // generates an exception

}

catch(exception\_class var)

{

// Code to be executed when an exception is thrown.

}

(any number of catch blocks can be written)

(matching catch will be automatically taken)

(program does not stop abruptly when try catch is used and the remaining part of the program will be executed)

(the exception the system throws is usually unreadable, but with try catch it becomes more user-friendly)

(try catch allows for graceful termination or continuation in case of an error)

(throw and throws – an exception is thrown as an object and a catch block matching it will catch it and handle it;

**Example:**

public class TryCatchEx

{

public static void main(String[] args)

{

System.out.println("11");

System.out.println("Before divide");

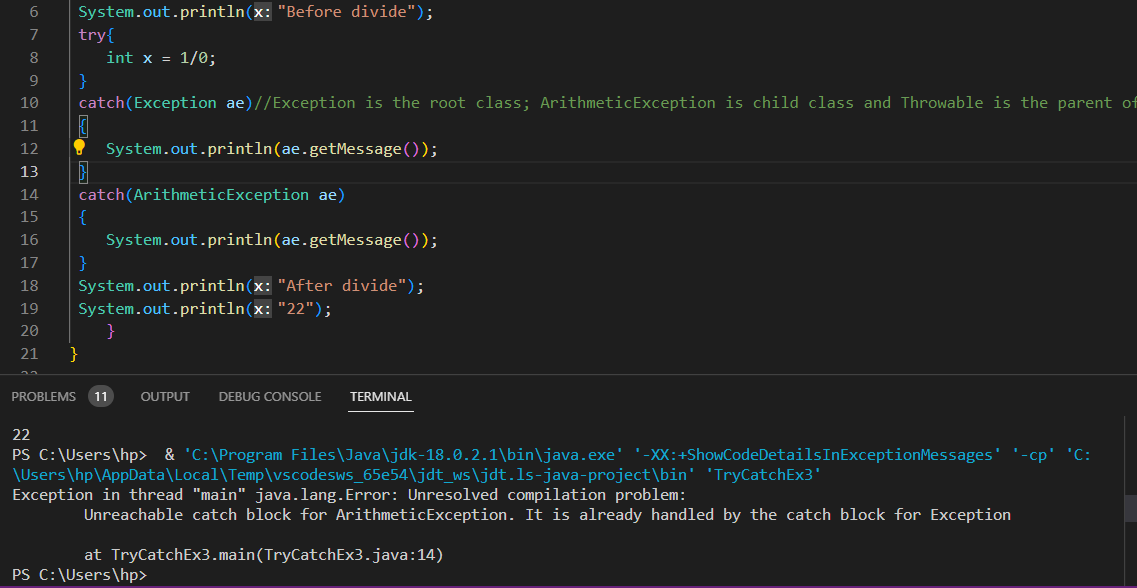
int x = 1/0;

System.out.println("After divide");

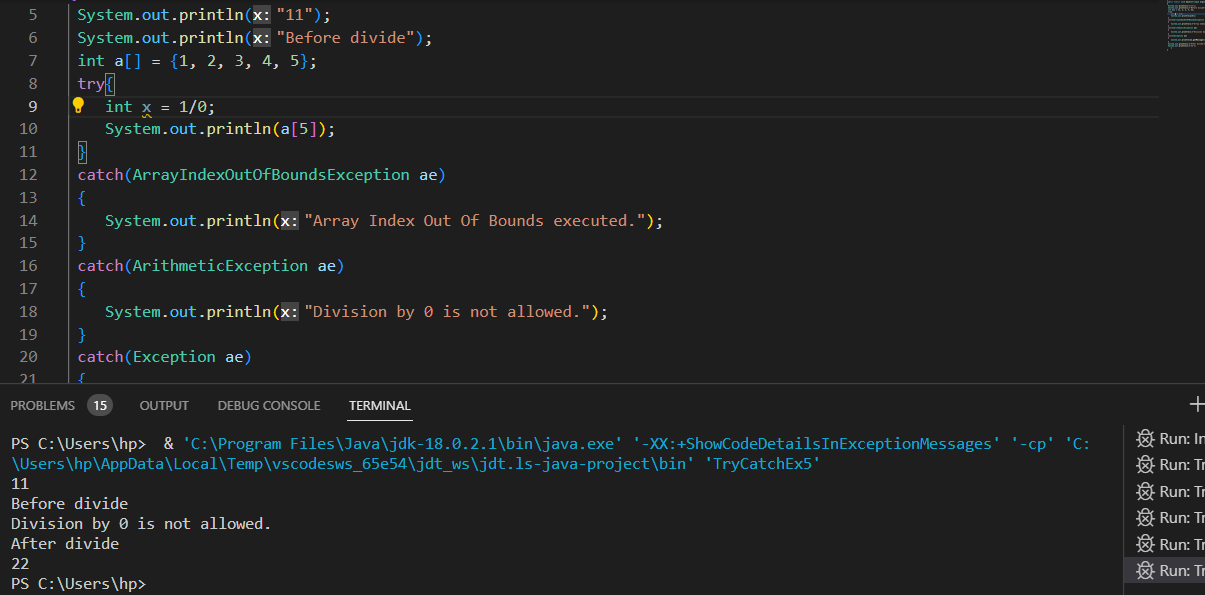
System.out.println("22");

}

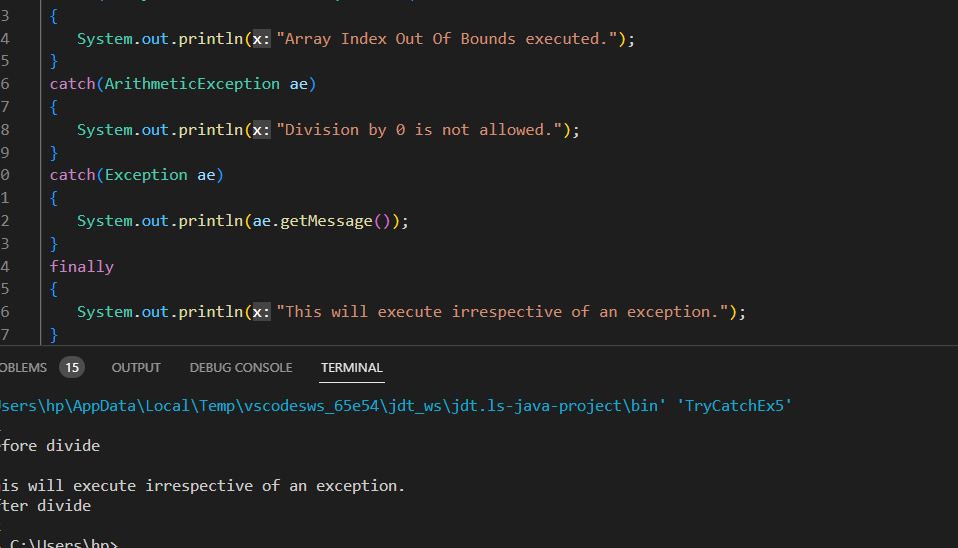
}



Child class exception handlers must be written at the beginning/at the top. Then, write the parent class exception handlers, towards the end. When parent is written first, there is no need to write the child ones because they are already covered (redundancy).



Only the first exception gets handled. Others do not even get displayed.



Finally is written only once.

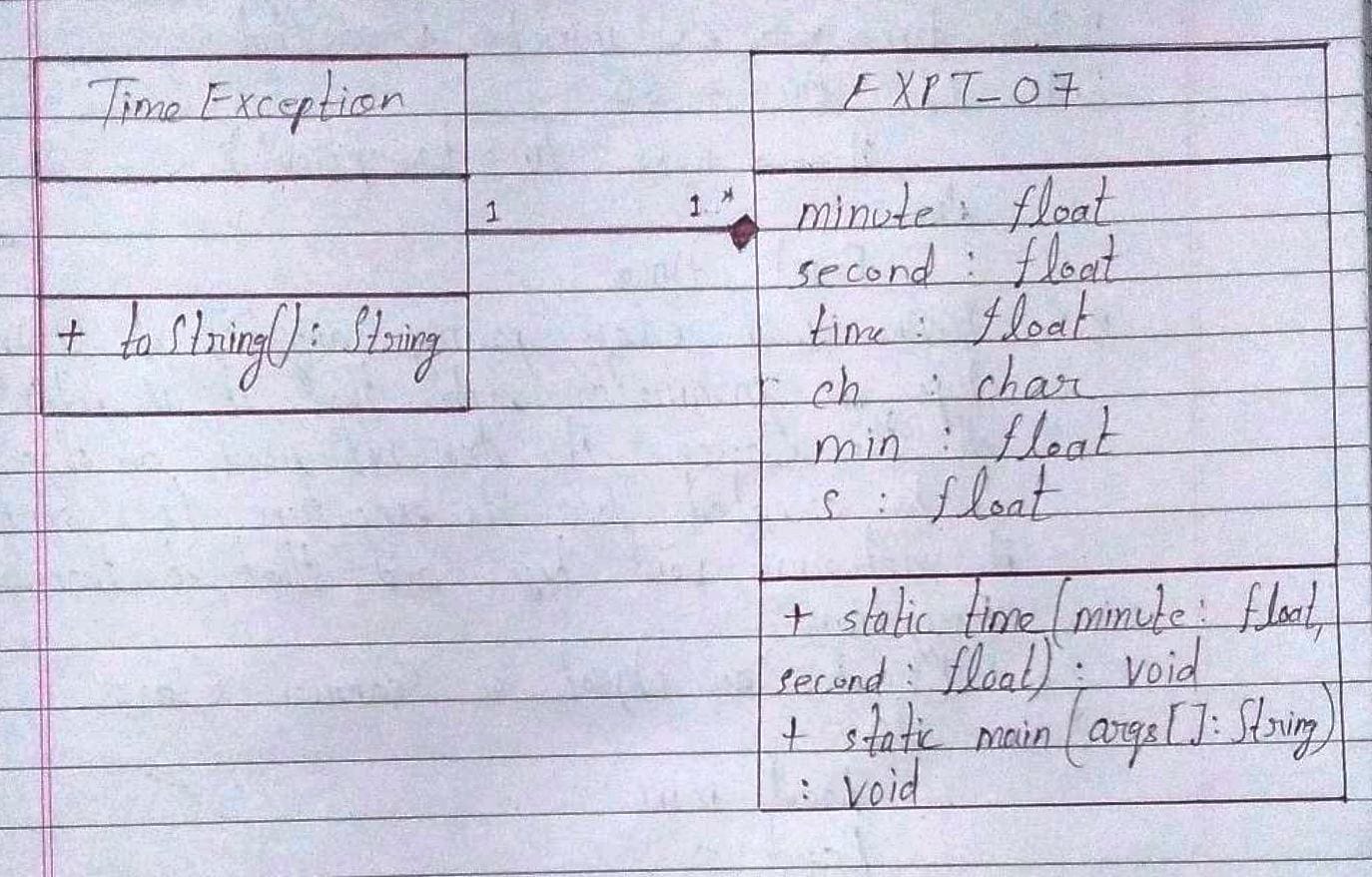
Output**:**

11

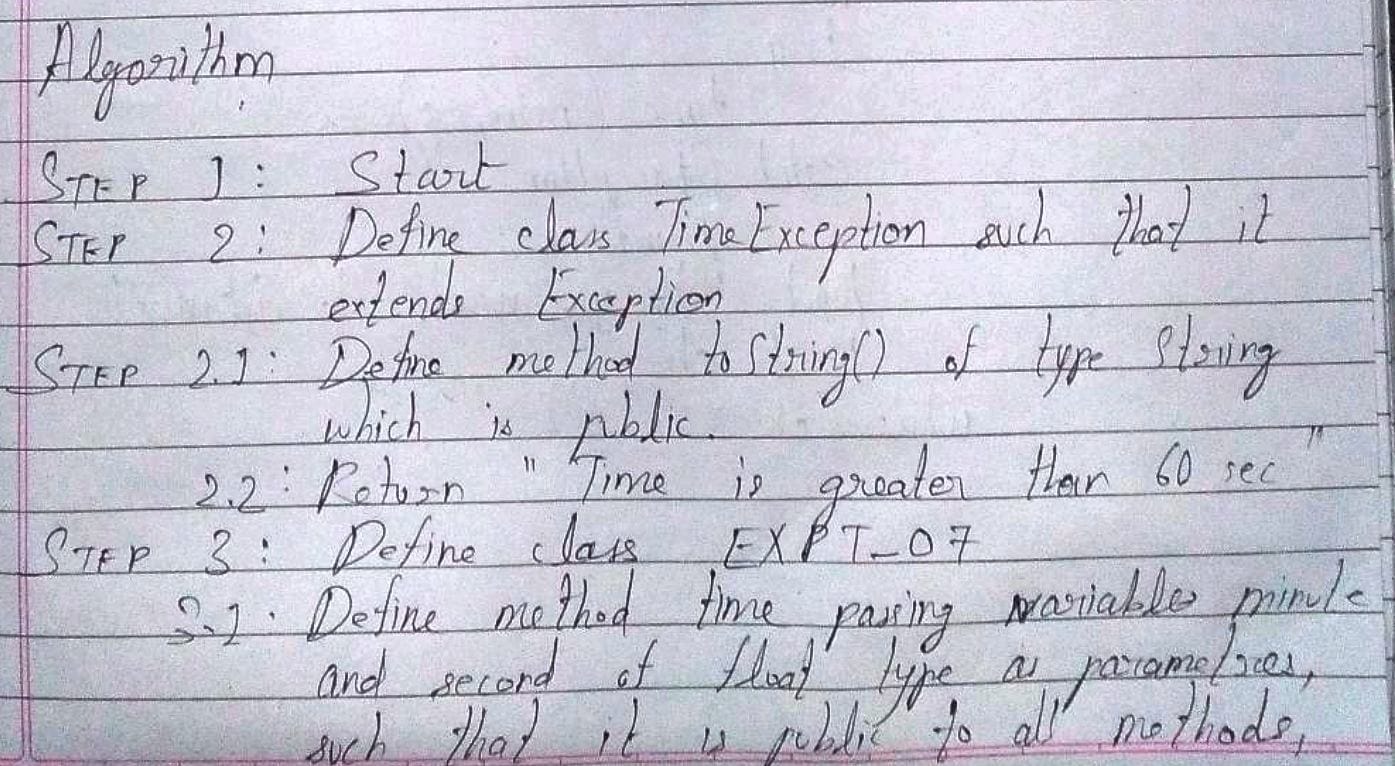
Before divide

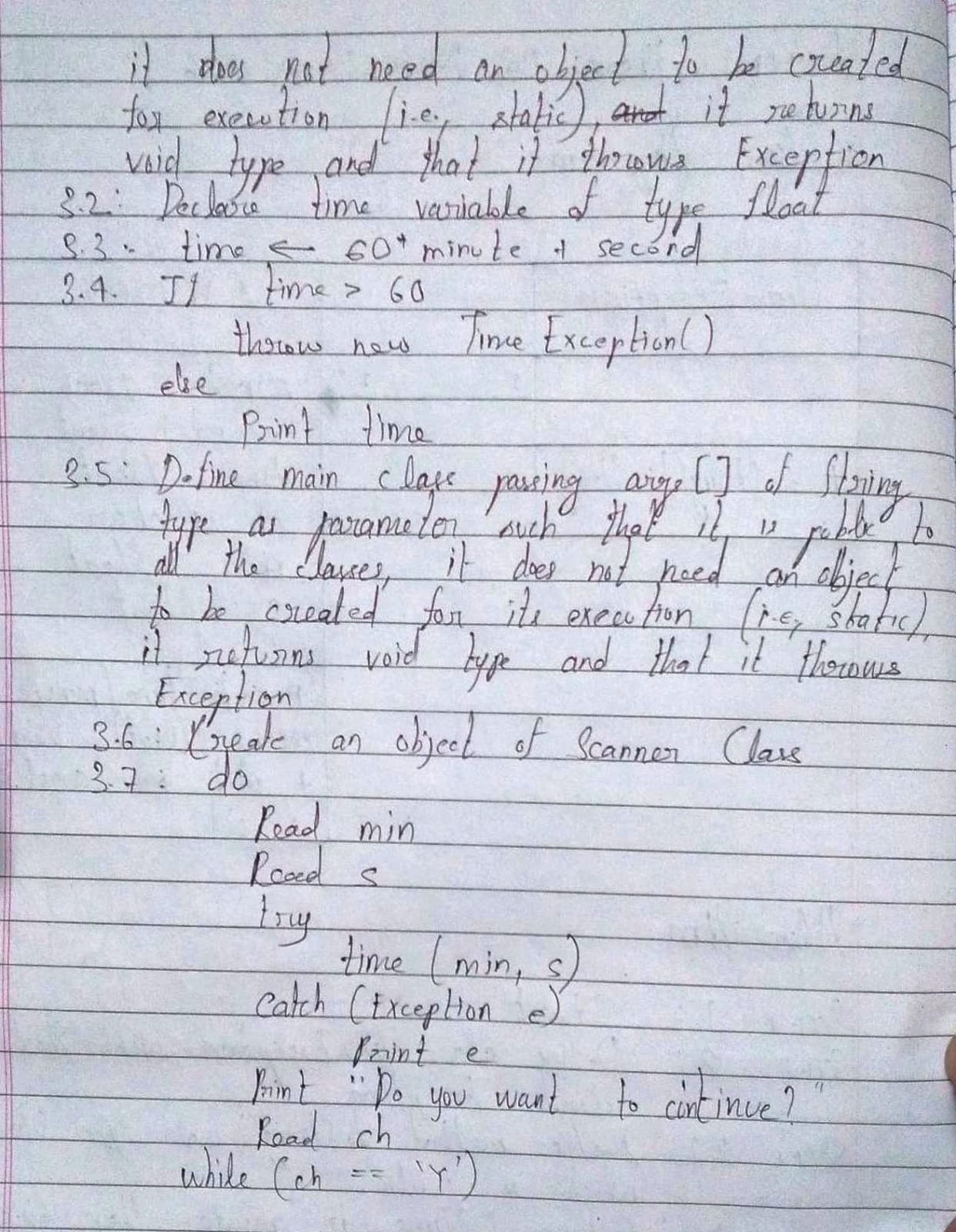
Exception in thread "main" java.lang.ArithmeticException: / by zero

**Class Diagram:**



**Algorithm:**





**Implementation details :**

Code:

import java.util.\*;

class TimeException extends Exception

{

    public String toString()

    {

        return "The time is greater than 60 seconds.";

    }

}

public class EXPT\_07

{

    public static void time(float minute, float second) throws Exception

    {

        float time;

        time = 60\*minute + second;

        if(time > 60)

        {

            throw new TimeException();

        }

        else

        {

            System.out.println("The time is: "+time);

        }

    }

    public static void main(String args[]) throws Exception

    {

        Scanner in = new Scanner(System.in);

        char ch;

        do

        {

            System.out.println("Enter the minutes: ");

            float min = in.nextFloat();

            System.out.println("Enter the seconds: ");

            float s = in.nextFloat();

            try {

                time(min, s);

            } catch (Exception e) {

                System.out.println(e);

            }

            System.out.println("Do you want to continue?\nPress 'Y' otherwise press any character: ");

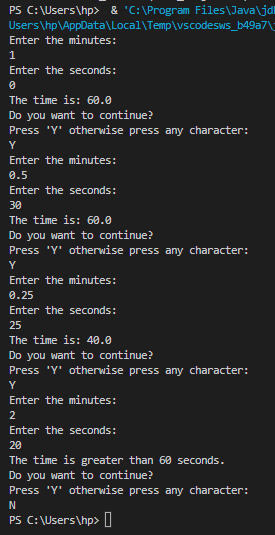
            ch = in.next().charAt(0);

        }while(ch=='Y');

    }

}

**Output:**



**Conclusion:**

Thus, in this experiment, the concept of in-built and user-defined packages has been learnt and implemented using Java Programming. The exceptions, especially the user defined ones (using try… catch and other methods), are useful because they enable a program to go beyond errors and execute statements after the erroneous statement if the related error has been handled. Further, using user-defined exceptions, the programmer can define custom conditions for a code to run, making Java versatile and efficient.

**Date: \_\_16-11-22\_\_ Signature of faculty in-charge**

**Post Lab Descriptive Questions**

1. Compare throw and throws.

Ans.

|  |  |  |
| --- | --- | --- |
| Sr. No. | throw | Throws |
| 1. | Java throw keyword is used to throw an exception explicitly in the code, inside the function or the block of code. | Java throws keyword is used in the method signature to declare an exception which might be thrown by the function during execution of the code. |
| 2. | The throw keyword is followed by an instance of Exception to be thrown. | The throws keyword is followed by class names of Exceptions to be thrown |
| 3. | throw is used within the method. | throws is used with the method signature. |
| 4. | Only one exception at a time is allowed to be thrown, i.e., multiple exceptions cannot be thrown. | Multiple exceptions can be declared using throws keyword that can be thrown by the method. Foe example, main() throws IOException, SQLException. |

1. Explain how to create a user defined exception and explicitly throw an exception in a program with a simple example.

Ans. Java provides the facility to create user-defined exceptions which are basically derived classes of Exception. Creating one’s own exception is known as a custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user needs.In simple words, it can be said that a User-Defined Exception or Custom Exception is to create one’s own exception class and throwing that exception using the ‘throw’ keyword.

For example, MyException in the below code extends the Exception class.

class MyException extends Exception{

public MyException(String s)

{

//Call constructor of parent Exception

super(s);

}

}

public class Main{

public static void main(String args[])

{

try{

throw new MyException(“GeeksGeeks”);

}

catch(MyException e)

{

System.out.println(“Caught”);

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

}

}

}

1. Suppose the statement2 causes an exception in following try-catch block:

try {

statement1;

statement2;

statement3;

}

catch(Exception1 e1) {

}

catch(Exception2 e2){

}

statement4;

Answer the following questions:

* Will statement3 be executed?
* If the exception is not caught, will statement4 be executed?
* If the exception is caught in the catch block, will statement4 be executed?
* If the exception is passed to the caller, will the statement4 be executed?

Ans. Yes, statement 3 will be executed.

No, if the exception in not caught, statement4 will not be executed.

Yes, if the exception is caught in the catch block, statement4 will be executed.

No, if the exception is passed to the caller, statement4 will not be executed.

1. Explain finally block with the help of an example.

Ans. The Java finally block is used to execute important code such as closing the connection, etc. The Java finally block is always executed whether an exception is handled or not. Therefore, it contains all the necessary statements that need to be printed regardless of whether the exception occurs or not.

Example:

class TestFinallyBlock

{

public static void main(String args[])

{

try

{

int data = 25/5;

System.out.println(data);

}

catch(NullPointerException e)

{

System.out.println(e);

}

finally

{

System.out.println(“Finally block is always executed.”);

}

System.out.println(“Rest of the code…”);

}

}