EXPERIMENT 12
// Exception Handling in Java

Prerequisite Knowledge

Exception - An exception is a runtime error.
Exception Handling - It is the mechanism of handling exceptions that may occur for a program.

- ➔ Java provides the exception handling mechanism in an object-oriented manner.
- ➔ A **Java exception** is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code.
- ➔ When an exceptional condition arises, an object representing that exception is created and thrown in the method that caused the error.
- ➔ That method may choose to handle the exception itself, or pass it on. Either way, at some point, the exception is caught and processed.

Java's exception handling is managed via five keywords: **try, catch, throw, throws,** and **finally**.

try - Program statements that we want to monitor for exceptions are contained within a try block.

catch - If an exception occurs within the try block, it is thrown. This exception is further catched by a catch block and handled in some rational manner.

- ➔ System-generated exceptions are automatically thrown by the Java runtime system.

throw - To manually throw an exception, we use the keyword throw.

throws - Any exception that is thrown by a method but its handling is not specified by the method itself, is required to be listed by a throws clause, so that it can be handled by the caller of the method.

finally - Any code that must be executed after a try block completes is put in a finally block.

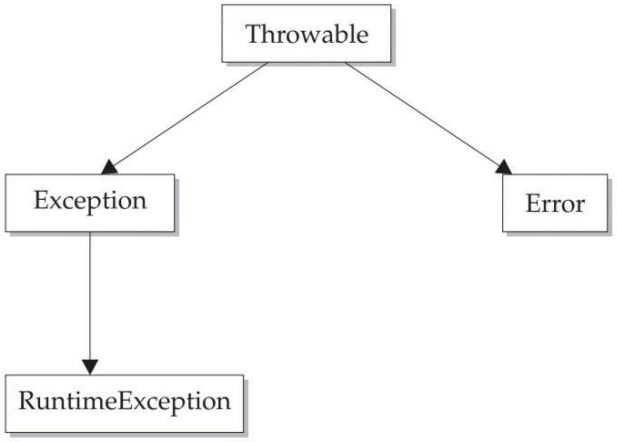
General form of an exception-handling block:

```
try
{  // block of code to monitor for errors  }
catch (ExceptionType1 exOb)
{  // exception handler for ExceptionType1 }
catch (ExceptionType2 exOb)
{ // exception handler for ExceptionType2  } …
finally
{  // block of code to be executed after try block ends  }
```

Here, ExceptionType is the type of exception that has occurred.

Types of Exceptions -

- ➔ All exception types are subclasses of the built-in class **Throwable**.
- ➔ Immediately below Throwable are two subclasses that partition exceptions into two distinct branches.
  - ◆ One branch is **Exception**. This class is used for exceptional conditions that user programs should catch.
    - There is an important subclass of Exception, called **RuntimeException**. Exceptions of this type are automatically defined for the programs that we write and include things such as division by zero and invalid array indexing.
  - ◆ The other branch is **Error**, which defines exceptions that are not expected to be caught under normal circumstances by our program. Exceptions of type Error are used by the Java run-time system to indicate errors having to do with the run-time environment, itself. Stack overflow is an example of such an error.



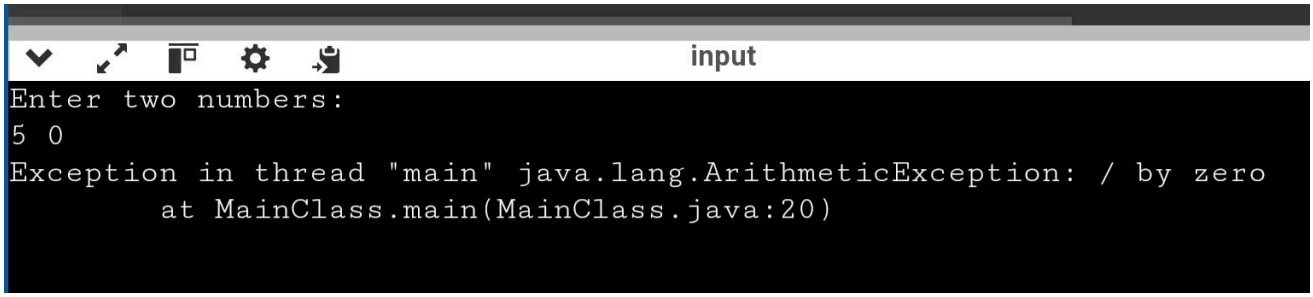
// Java's Default Exception Handler

Source Code -

```
import java.util.Scanner;
class MainClass
{
	public static void main (String args[])
	{
		Scanner sc = new Scanner(System.in);

		int num1,num2,res;
		System.out.println(“Enter two numbers: ”);
		num1 = sc.nextInt();
		num2 = sc.nextInt();
		res = num1/num2;
		System.out.println(“Result = ”+res);
	}
}
```

Output -



- ➔ When the Java run-time system detects the attempt to divide by zero, it constructs a new exception object and then throws this exception.
- ➔ This causes the execution of MainClass to stop, because once an exception has been thrown, it must be caught by an exception handler and dealt with immediately.
- ➔ In this example, we haven’t supplied any exception handlers of our own, so the exception is caught by the default handler provided by the Java run-time system.
- ➔ The default handler displays a string describing the exception, prints a stack trace from the point at which the exception occurred, and terminates the program.
- ➔ The type of exception thrown is a subclass of Exception called **ArithmeticException**, which more specifically describes what type of error happened.

PROGRAM 16

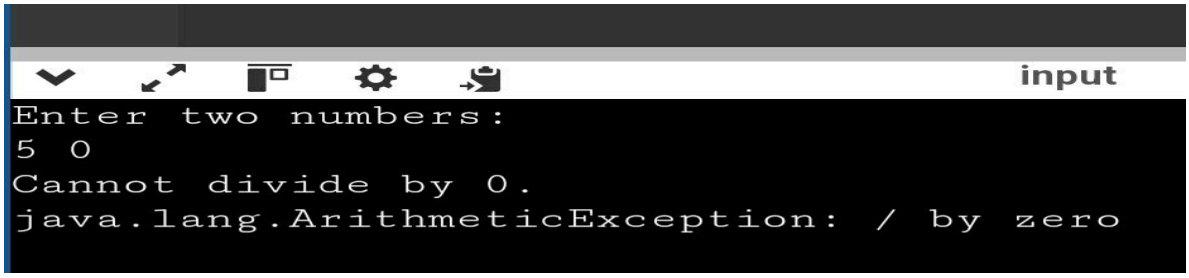
// Handling exceptions manually in Java.

Source Code -

```
import java.util.Scanner;
class MainClass
{
	public static void main (String args[])
	{
		Scanner sc = new Scanner(System.in);

		int num1,num2,res;
		System.out.println(“Enter two numbers: ”);
		num1 = sc.nextInt();
		num2 = sc.nextInt();
		try
		{
			res = num1/num2;
			System.out.println(“Result = ”+res);
		}
		catch(ArithmeticException exp)
		{
			System.out.println(“Cannot divide by 0.”);
			System.out.println(exp);
		}
	}
}
```

Output -



➔ Once an exception has been thrown from the try block, the code present in the try block after the thrown exception is not executed.

PROGRAM 17

// Nested try blocks and multiple catch blocks.

Source Code -

```
import java.util.Scanner;
class MainClass
{       public static void main (String args[])
        {       Scanner sc = new Scanner(System.in);
                int num1,num2,res;
                System.out.println("Enter two numbers: ");
                num1 = sc.nextInt();
                num2 = sc.nextInt();
                try
                {
                        try
                        {
                                res = num1/num2;
                                System.out.println("Result = "+res);
                        }
                        catch(NullPointerException exp)
                        {
                                System.out.println(exp);
                        }
                }
                catch(NullPointerException exp)
                {
                        System.out.println(exp);
                }
                catch(ArithmeticException exp)
                {
                        System.out.println("Cannot divide by 0.");
                        System.out.println(exp);
                }       }       }
```

Output -

// Throwing exceptions manually

- ➔ So far, we have only been catching exceptions that are thrown by the Java run-time system. However, we can also throw an exception explicitly from a try block, using the **throw** statement.

  Syntax:

  throw ThrowableInstance;

  Here, ThrowableInstance must be an object of type Throwable or a subclass of Throwable.

  - ◆ Primitive types, such as int or char, as well as non-Throwable classes, such as String and Object, cannot be used as exceptions.
- ➔ The flow of execution stops immediately after the throw statement and any subsequent statements in the try block are not executed.
- ➔ The nearest enclosing try block is inspected to see if it has a catch statement that matches the type of exception thrown. If it does find a match, control is transferred to that statement. If not, then the next enclosing try statement is inspected, and so on. If no matching catch is found, then the default exception handler halts the program and prints the stack trace for the exception.

PROGRAM 18

// Throwing exceptions manually.

Source Code -

```
import java.util.Scanner;
class MainClass
{
   public static void main (String args[])
   {
      Scanner sc = new Scanner(System.in);
      int num1,num2,res;
      System.out.println("Enter two numbers: ");
      num1 = sc.nextInt();
      num2 = sc.nextInt();
      try
      {
         res = num1/num2;
         throw new ArithmeticException();
      }
      catch(ArithmeticException exp)
      {
         System.out.println("Cannot divide by 0.");
         System.out.println(exp);     }        }        }
```

Output -

Enter two numbers:
5 0
Cannot divide by 0.
java.lang.ArithmeticException: / by zero

// Significance of 'throws' statement

- ➔ If a method is capable of causing an exception that it does not handle, it must specify this behavior so that callers of the method can guard themselves against that exception.
- ➔ We do this by including a **throws** clause in the method's declaration.
- ➔ A throws clause lists the types of exceptions that a method might throw.
  - ◆ This is necessary for all exceptions, except those of type Error or RuntimeException, or any of their subclasses. All other exceptions that a method can throw must be declared in the throws clause. If they are not, a compile-time error will result.

Syntax:

return_type method-name(parameter-list) throws (exception-list)
{
// body of method
}

Here, exception-list is a comma-separated list of the exceptions that a method can throw

PROGRAM 19

Source Code -

```
class ThrowsDemo
{
   static void throwOne() throws IllegalAccessException
   {
      System.out.println("Inside throwOne.");
      throw new IllegalAccessException("demo");
   }
   public static void main(String args[])
   {
      try
      {
         throwOne();
      }
```

```
        catch (IllegalAccessException e)
        {
            System.out.println("Caught " + e);
        }
}
}
```

Output -

```
Inside throwOne.
Caught java.lang.IllegalAccessException: demo
```

PROGRAM 20

// Significance of 'finally' block

Source Code -

```
import java.util.Scanner;
class MainClass
{
public static void main (String args[])
{
Scanner sc = new Scanner(System.in);

int num1,num2,res;
System.out.println("Enter two numbers: ");
num1 = sc.nextInt();
num2 = sc.nextInt();
try
{
res = num1/num2;
System.out.println("Result = "+res);
}

catch(ArithmeticException exp)
{
System.out.println("Cannot divide by 0.");
System.out.println(exp);
}
finally
{
   System.out.println("Exception Caught");        }        }        }
```

Output -

```
Enter two numbers:
5 0
Cannot divide by 0.
java.lang.ArithmeticException: / by zero
Exception Caught
```