National University of Computer & Emerging Sciences,
Karachi
Computer Science Department
Spring 2023, Lab Manual – 10

Course Code: CL-1004	Course : Object Oriented Programming Lab
Instructor(s) :	Abeer Gauher, Hajra Ahmed, Shafique

LAB - 10

Exception Handling

Exception Handling

The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions. Let's consider a scenario:

statement 1;

statement 2;

statement 3;

statement 4;

statement 5;//exception occurs

statement 6;

statement 7;

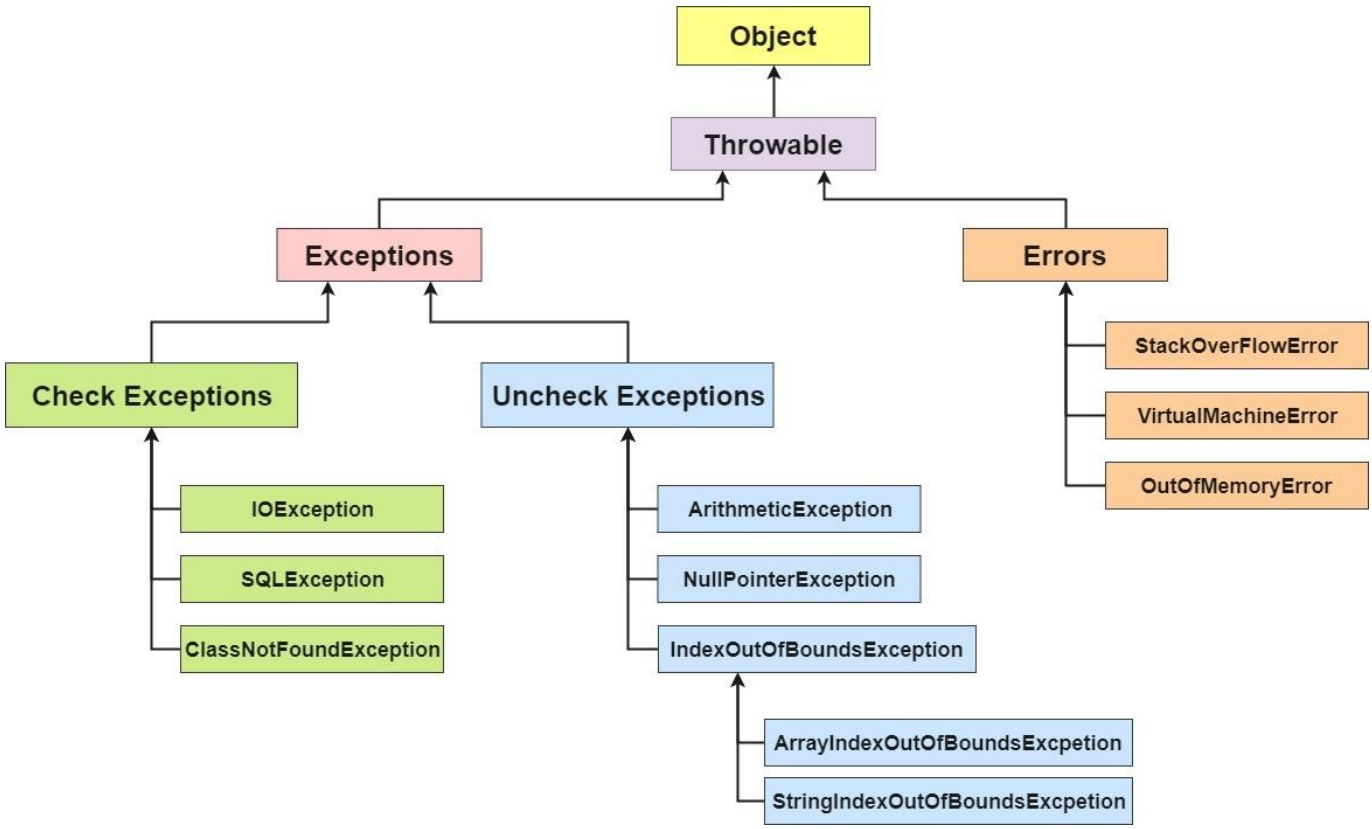
statement 8;

statement 9;

statement 10;

Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed.

Hierarchy of Java Exception classes



Java Exception Keywords

Java provides five keywords that are used to handle the exception. The following table describes each.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Simple Try Catch Example:

```
public class JavaExceptionExample{
   public static void main(String args[]){
    try{
        //code that may raise exception
        int data=100/0;
    }catch(ArithmeticException e){System.out.println(e);}
    //rest code of the program
    System.out.println("rest of the code...");
   }
}
```

Output:

```
java.lang.ArithmeticException: / by zerorest of the code...
```

Common Scenarios of Java Exceptions

There are given some scenarios where unchecked exceptions may occur. They are as follows:

1) A scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

int a=50/0;//ArithmeticException

2) A scenario where NullPointerException occurs

If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

String s=null;

System.out.println(s.length());//NullPointerException

3) A scenario where NumberFormatException occurs

If the formatting of any variable or number is mismatched, it may result into NumberFormatException. Suppose we have a string variable that has characters; converting this variable into digit will cause NumberFormatException.

```
String s="abc";

int i=Integer.parseInt(s);//NumberFormatException
```

4) A scenario where ArrayIndexOutOfBoundsException occurs

When an array exceeds to it's size, the ArrayIndexOutOfBoundsException occurs. there may be other reasons to occur ArrayIndexOutOfBoundsException. Consider the following statements.

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

a[10]=50; //ArrayIndexOutOfBoundsException
```

Multi - Catch:

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

Points to remember:

- At a time only one exception occurs and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception or a compile – time error occurs.

Multi - Catch Example:

```
public class MultipleCatchBlock1 {

    public static void main(String[] args) {

        try{
            int a[]=new int[5];
            a[5]=30/0;
           }
           catch(ArithmeticException e)
              {
               System.out.println("Arithmetic Exception occurs");
              }
           catch(ArrayIndexOutOfBoundsException e)
              {
               System.out.println("ArrayIndexOutOfBounds Exception occurs");
              }
           catch(Exception e)
              {
               System.out.println("Parent Exception occurs");
              }
           System.out.println("rest of the code");
    }
}
```

Output:

```
Arithmetic Exception occurs
rest of the code
```

Java finally block

Java finally block is a block used to execute important code such as closing the connection, etc.

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 the exception occurs or not.

The finally block follows the try-catch block.

Finally Example: the code throws an exception however the catch block cannot handle it. Despite this, the finally block is executed after the try block and then the program terminates abnormally.

```
public class TestFinallyBlock1{
      public static void main(String args[]){

      try {

        System.out.println("Inside the try block");

        //below code throws divide by zero exception
       int data=25/0;
       System.out.println(data);
      }
      //cannot handle Arithmetic type exception
      //can only accept Null Pointer type exception
      catch(NullPointerException e){
        System.out.println(e);
      }

      //executes regardless of exception occured or not
      finally {
        System.out.println("finally block is always executed \n");
      }

      System.out.println("rest of the code...");
      }
}
```

Output:

```
Inside the try block
finally block is always executed

Exception in thread "main" java.lang.ArithmeticException: / by zero
at TestFinallyBlock1.main(TestFinallyBlock1.java:12)
```

Throw Keyword:

The throw keyword is used to create a custom error.

The throw statement is used together with an exception type. There are many exception types available in Java: ArithmeticException, ClassNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc.

```
public class Main {
  static void checkAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18 years old.");
    } else {
      System.out.println("Access granted - You are old enough!");
    }
 }

 public static void main(String[] args) {
   checkAge(15);
 }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: Access denied - You must be at least 18 years old.
    at Main.checkAge(Main.java:7)
    at Main.main(Main.java:14)
```

Java throws keyword

The Java throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception.

Syntax of Java throws

return_type method_name() throws exception_class_name{

//method code

}

The throws keyword is used in a method signature and declares which exceptions can be thrown from a method. The throws keyword can be useful for propagating exceptions in the call stack and allows exceptions to not necessarily be handled within the method that declares these exceptions. Only checked exceptions are required to be thrown using the throws keyword. Unchecked exceptions don't need to be thrown or handled explicitly in code.

Sr. no.	Basis of Differences	throw	throws
1.	Definition	Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code.	Java throws keyword is used in the methoc signature to declare an exception whicl might be thrown by the function while the execution of the code.
2.	Type of exception Using throw keyword, we can only propagate unchecked exception i.e., the checked exception cannot be propagated using throw only.	Using throws keyword, we can declare both checked and unchecked exceptions. However, the throws keyword can be used to propagate checked exceptions only.	
3.	Syntax	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.
4.	Declaration	throw is used within the method.	throws is used with the method signature.
5.	Internal implementation	We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions.	We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException.

Java throws Example

```
public class TestThrows {
   //defining a method
   public static int divideNum(int m, int n) throws ArithmeticException {
      int div = m / n;
      return div;
   }
   //main method
   public static void main(String[] args) {
      TestThrows obj = new TestThrows();
      try {
         System.out.println(obj.divideNum(45, 0));
      }
      catch (ArithmeticException e){
         System.out.println("\nNumber cannot be divided by 0");
      }

      System.out.println("Rest of the code..");
   }
}
```

Lab Tasks

1. You are building a program that calculates the average of a list of integers. Write a Java program that handles the following exceptions:
    - If the list is null or empty, the program should display an appropriate error message.
    - If any of the integers in the list are not valid integers (i.e., they contain non-numeric characters), the program should skip that integer and display only numeric data's average.
    - If the list contains only invalid integers (non – numeric), the program should display an appropriate message.

    Your program should use try-catch blocks to handle the exceptions. You can assume that the list is provided as an argument.

2. Calculate the factorial of a given integer n. The program should take input from the user in the form of a string and convert it to an integer (Integer.parseInt()). Write a Java program that handles the following exceptions:

    - If the input string is null or empty, the program should display an appropriate error message.
    - If the input string does not represent a valid integer (i.e., it contains non-numeric characters), the program should display an appropriate error message.
    - If the input integer is negative, the program should display an appropriate error message.
    - If the input integer is too large to calculate the factorial (i.e., it causes an integer overflow), the program should display an appropriate error message.

    Your program should use try-catch blocks to handle the exceptions. If the input integer is valid and can be used to calculate the factorial, the program should calculate the factorial and display it to the user.

3. Ask the user to enter two numbers and perform a mathematical operation on them. The program should handle the following exceptions:

    - If the input for either number is null or empty, the program should display an appropriate error message.
    - If the input for either number does not represent a valid integer or double (i.e., it contains non-numeric characters), the program should display an appropriate error message.
    - If the user attempts to divide by zero, the program should display an appropriate error message.

    Your program should use try-catch blocks to handle the exceptions. If the input is valid, the program should perform the mathematical operation and display the result to the user.

4. Prompt the user to enter a sentence and then performs a series of operations on the sentence. The program should handle the following exceptions:
    - If the input string is null or empty, the program should display an appropriate error message.
    - If the input string contains less than two words, the program should display an appropriate error message.

    Your program should use try-catch blocks to handle the exceptions. If the input and operation are valid, display the modified sentence to the user.