CSE 215: Programming Language II Lab

Sec – 8, Faculty - MUO

Lab Officer: Tanzina Tazreen

Lab – 12

Exception Handling

Exceptions

Exceptions are circumstances where the program exits abruptly due to an erroneous code or input.

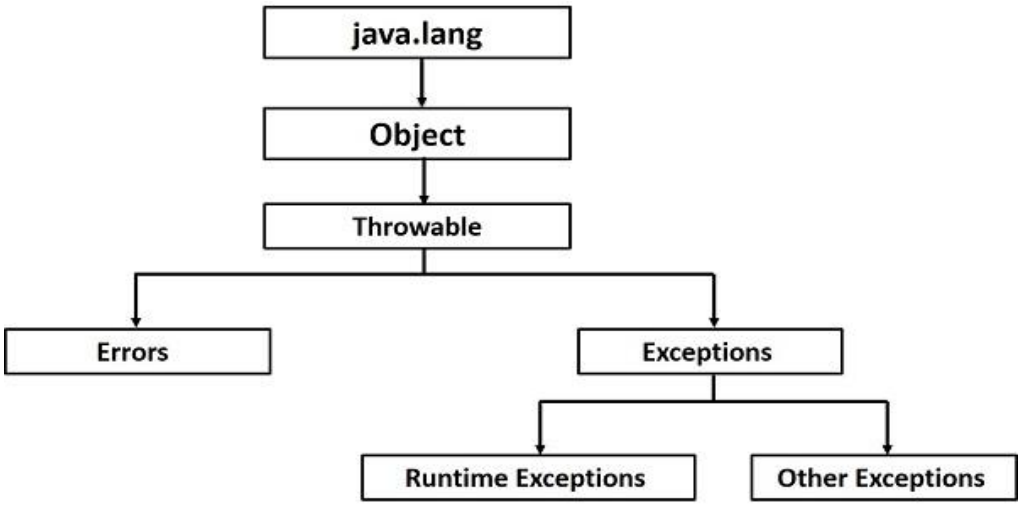
For example, if you have a Square class, if someone passes in a negative number for the length of the side, an error will occur. When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an **exception**.

Exception handling

Exception handling is the process of handling Exceptions so that the program exits in a more well-suited manner or simply continues on with other tasks instead of just crashing.

Mainly we have two categories of Exceptions:

- **Checked exceptions** − A checked exception is an exception that is checked (notified) by the compiler at compilation-time, these are also called as compile time exceptions. These exceptions cannot simply be ignored, the programmer should take care of (handle) these exceptions. *FileNotFoundException*
- **Unchecked exceptions** − An unchecked exception is an exception that occurs at the time of execution. These are also called as **Runtime Exceptions**. programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation. *ArrayIndexOutOfBoundsExceptionexception*



There are some built-in exceptions in Java. A select few among them are given below:

- **ArithmeticException**

It is thrown when an exceptional condition has occurred in an arithmetic operation.

- **ArrayIndexOutOfBoundsException**

It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

- **ClassNotFoundException**

This Exception is raised when we try to access a class whose definition is not found

- **FileNotFoundException**

This Exception is raised when a file is not accessible or does not open.

- **IOException**

It is thrown when an input-output operation failed or interrupted

- **NoSuchMethodException**

It is thrown when accessing a method which is not found.

- **NullPointerException**

This exception is raised when referring to the members of a null object. Null represents nothing

- **RuntimeException**

This represents any exception which occurs during runtime.

- **StringIndexOutOfBoundsException**

It is thrown by String class methods to indicate that an index is either negative or greater than the size of the string

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 an exception might occur. 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 custom or programmer defined 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.

try-catch-finally	Throwable (super class of all exceptions)

try{
// code which may throw any exception
}
catch(ExceptionType e){
// handles the exception e
}
finally{
// It'll always run
}
	Methods to get information about Exception
	
	+getMessage(): String
	+toString(): String
	+printStackTrace(): void
	+getStackTrace(): StackTraceElement[]

Multiple Catch Blocks

A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following −

Syntax:

```
try {
   // Protected code
} catch (ExceptionType1 e1) {
   // Catch block
} catch (ExceptionType2 e2) {
   // Catch block
} catch (ExceptionType3 e3) {
   // Catch block
}
```

For example:

```
public class Main {
  public static void main(String[] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    }
    catch (Exception e) {
      System.out.println("Something went wrong.");
    }
    finally {
```

```
      System.out.println("The 'try catch' is finished.");
    }
  }
}
```

throw and throws

```
try{
  if(condition){
  throw new ExceptionType();
  }
}
catch(ExceptionType e){
//handle here
}
```

throws is used in method signature

```
void M() throws ExceptionType{
  if(condition){
    throw new ExceptionType();
  }
}
/* The thrown exception must be
handled inside caller method */
```

```
void M() throws Exception1, Exception2, … ExceptionN{
  if(condition){
    throw new Exception1();
  }// else if statements
  else{
    throw new ExceptionN();
  }
}
```

Example

Throw an exception if **age** is below 18 (print "Access denied"). If age is 18 or older, print "Access granted":

```
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); // Set age to 15 (which is below 18...)
  }
}
```

Custom (User-defined) Exceptions

You can create your own exceptions in Java. Keep the following points in mind when writing your own exception classes −

- All exceptions must be a child of Throwable.
- If you want to create your own exception, you just need to extend the **Exception** class.

```
class IllegalAgeException extends Exception
{
    public IllegalAgeException(String s)
    {
        super(s);
    }
}
```

```


public class Main
{
   public static void checkAge(int age) throws IllegalAgeException
   {
    if (age < 18)
      throw new IllegalAgeException("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)
   {
       try
       {
         checkAge(21);
       }
       catch(IllegalAgeException e)
       {
           e.printStackTrace();
       }

    }
}
```

Task:

1. Write a program that takes 10 positive integers from user and prints the sum. If any negative value is entered, the program should catch it as an exception and display “Input positive integer only”. The program must continue taking input until it gets 10 positive integers.
2. Write a program that creates an integer array of size 100 and initialize it with random values:

```
int a = (int) (Math.random() * 10000);
```

The program then takes an integer from user, use it as an index and tries to print the corresponding element of that array. If index is out of array size, the program should catch it and display appropriate message.

3. Create a Triangle class. Now create IllegalTriangleException class that extends Exception. If the sum of any two sides is not greater than the third side, the Triangle class should throw IllegalTriangleException.