IS 2104 - Rapid Application Development

Java Exception Handling

M.V.P. Thilini Lakshika
University of Colombo School of Computing
tlv@ucsc.cmb.ac.lk

Lesson Outline

- What is an Exception?
- How to Handle Exception?
- Java Exception class Hierarchy
- Java Exception Keywords
- User Defined Exceptions

What is Exception?

- Exception is an object that represents an error or a condition that prevents execution from proceeding normally.
- If the exception is not handled, the program will terminate abnormally.
- There are two types of errors.

    1. Compile time errors

    Compile time errors can be again classified again into two types as

    - Syntax Errors
    - Semantic Errors

    2. Runtime errors

What is Exception?

Compile Time Errors	Runtime Errors
Referred to the error corresponding to syntax or semantics.	Referred to the error encountered during the execution of code at runtime.
Compile-time errors get detected by compiler at the time of code development.	Runtime time errors are not get detected by compiler and hence identified at the time of code execution.
Compile-time errors as already mentioned can get fixed at the time of code development.	Runtime time errors are getting to fixing state after once code get executed and errors get identified.
Ex: Missing Parenthesis (}) Printing the value of variable without declaring it Missing semicolon (terminator)	Ex: ClassNotFoundException, IOException, SQLException, RemoteException, Arithmetic exception, Divide by zero exception, Nullpointer exception, ArrayIndexOutOfBounds, Interrupted exceptions

Compile Time Errors

```
public class CompileTime{
   public static void main(String args[]){
      int x=10;
      int y=5;
      System.out.println(x/y)
   }
}
```

```
CompileTime.java:5: error: ';' expected
      System.out.println(x/y)
                         ^
1 error
```

```
public class CompileTime{
   public static void main(String args[]){
      in x=10;
      int y=5;
      System.out.println(x/y);
   }
}
```

```
CompileTime.java:3: error: cannot find symbol
      in x=10;
      ^
  symbol:   class in
  location: class CompileTime
1 error
```

```
public class CompileTime{
   public static void main(String args[]){
      int x=10;
      int y=5;
      int x=20;
      System.out.println(x/y);
   }
}
```

```
CompileTime.java:5: error: variable x is already defined in method main(String[])
      int x=20;
          ^
1 error
```

Runtime Errors

- A Runtime error is called an Exceptions error.
- An exception is a problem that arises during the execution of a program that interrupts the normal flow of program execution.
- Exceptions in Java are something that is out of developers control.
- When an Exception occurs, the program terminates abnormally, which is not recommended, therefore, these exceptions need to be handled.

Runtime Errors

```
public class RunTime{
   public static void main(String args[]){
      int x=10;
      int y=0;
      int z=x/y;
      System.out.println(z);
   }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at RunTime.main(RunTime.java:5)
```

```
public class RunTime{
    public static void main(String args[]){
        int a[]=new int[10];
        a[11] = 9;
       }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 11
        at RunTime.main(RunTime.java:4)
```

```
public class RunTime{
    public static void main(String args[]){
        String str=null;
        System.out.println (str.length());
    }
}
```

```
Exception in thread "main" java.lang.NullPointerException
        at RunTime.main(RunTime.java:4)
```

How to Handle Exception?

- Blaming your code failure on environmental issues is not a solution.
- You need a robust programming, which takes care of exceptional situations.
- Such code is known as Exception Handler.

Ex: Suppose you have written a program to access the server. Program worked fine while you were developing the code. During the execution time, the server suddenly shuts down. When your program tried to access it, an exception is raised.

How to Handle Exception?

- A good exception handling for earlier example would be, when the server is down, connect to the backup server.
- Check whether the server is down, If yes, write the code to connect to the backup server (Using traditional if and else conditions).
- But, using "if" and "else" loop is not effective when your code has multiple java exceptions to handle.

```
class connect{
if(Server Up){
// code to connect to server
} else{
// code to connect to backup server
}
}
```

How to Handle Exception?

Try Catch Block

- Java provides an inbuilt exception handling.

    1. The normal code goes into a TRY block.

    2. The exception handling code goes into the CATCH block.

- In our example, TRY block will contain the code to connect to the server.

- CATCH block will contain the code to connect to the backup server.

```
class connect{
  Try {
  // code to connect to server
  } catch Exception {
  // code to connect to backup server
  }
}
```

How to Handle Exception?

Syntax for try & catch

```
try{
  statement(s)
}
catch (exceptionType name){
  statement(s)
}
```

How to Handle Exception?

Example

Step 1) Copy the following code into an editor and save the file.

```
class JavaException {
   public static void main(String args[]){
            int x = 10;
            int y = 0;
            int fraction = x/y;
            System.out.println("End Of Main");
   }
}
```

Step 2) Compile the code and run the program.

How to Handle Exception?

Step 3) An Arithmetic Exception - divide by zero is shown as below for line # 5 and line # 6 is never executed.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at JavaException.main(JavaException.java:5)
```

Step 4) Now let's examine how the try and catch will help us to handle this exception.

We will put the exception causing line of the code into a **try** block, followed by a **catch** block.

How to Handle Exception?

Step 5) Save, Compile & Run the code.

```
class JavaException {
   public static void main(String args[]) {
            int x = 10;
            int y = 0;
   try {
            int fraction = x / y;
            System.out.println("This line will not be Executed");
        } catch (ArithmeticException e) {
            System.out.println("In the catch Block due to Exception = " + e);
        }
            System.out.println("End Of Main");
  }
}
```

How to Handle Exception?

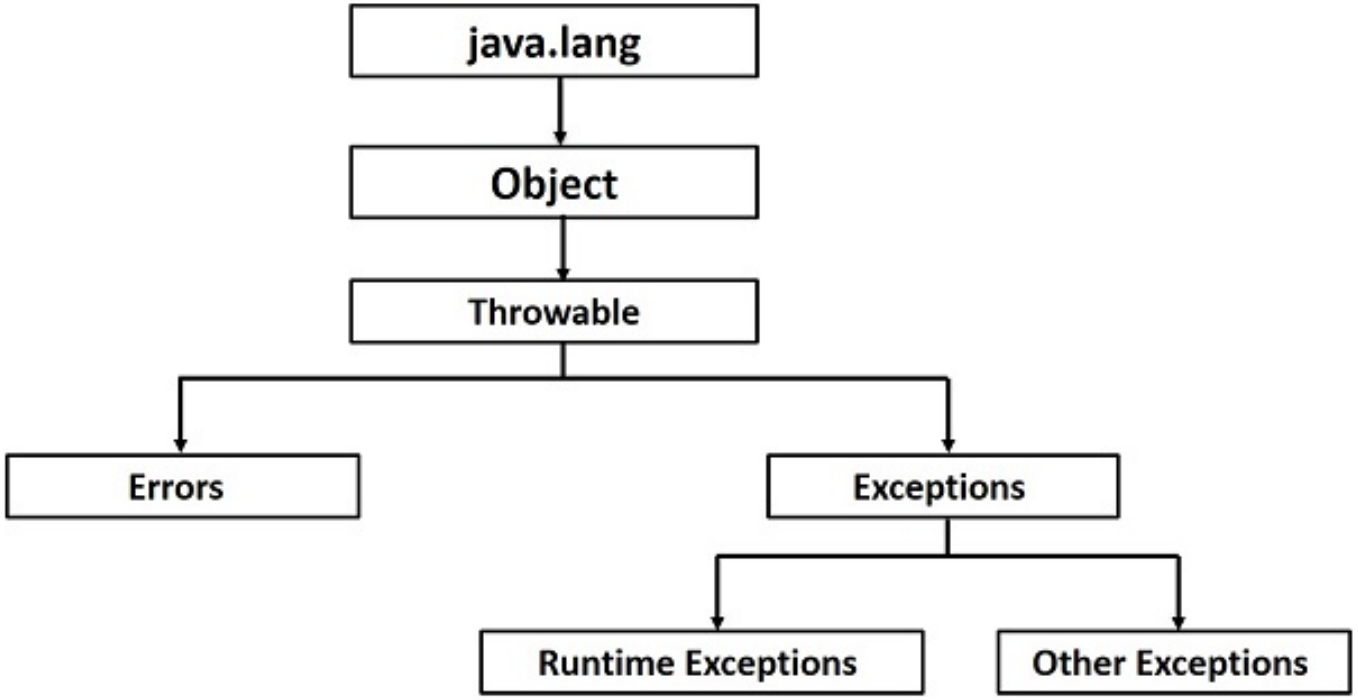
Expected output :

```
In the catch Block due to Exception = java.lang.ArithmeticException: / by zero
End Of Main
```

- As you observe, the exception is handled, and the last line of code is also executed.
- Line #7 will not be executed because as soon as an exception is raised the flow of control jumps to the catch block.
- The AritmeticException Object "e" carries information about the exception that has occurred which can be useful in taking recovery actions.

Java Exception class Hierarchy

- After one catch statement executes, the others are bypassed, and execution continues after the try/catch block.
- The nested catch blocks follow Exception hierarchy.



Java Exception class Hierarchy

- All exception classes in Java extend the class Throwable.
- Throwable has two subclasses, Error and Exception.
- The Error class defines the exception or the problems that are not expected to occur under normal circumstances by our program.

  Ex: Memory error, Hardware error, JVM error
- The Exception class represents the exceptions that can be handled by our program, and our program can be recovered from this exception using try and catch block.

Java Exception class Hierarchy

Example: To understand nesting of try and catch blocks

Step 1) Copy the following code into an editor and save the file.

```
1 class JavaException {
2 public static void main(String args[]) {
3       try {
4          int x = 10;
5          int y = 2;
6          int fraction = x / y;
7          int g[] = { 1 };
8          g[20] = 100;
9       }
10      /*catch(Exception e){
11         System.out.println("In the catch block due to Exception = "+e);
12      }*/
13      catch (ArithmeticException e) {
14         System.out.println("In the catch block due to Exception = " + e);
15      } catch (ArrayIndexOutOfBoundsException e) {
16         System.out.println("In the catch block due to Exception = " + e);
17      }
18      System.out.println("End Of Main");
19      }
20 }
```

Java Exception class Hierarchy

Step 2) Compile the code and run the program.

Step 3) An ArrayIndexOutOfBoundsException is generated. Change the value of int y to 0. Save, Compile & Run the code.

Step 4) An ArithmeticException must be generated.

Step 5) Uncomment line #10 to line #12. Save, Compile & Run the code.

You will get a compilation error. This is because Exception is the base class of ArithmeticException Exception. Any Exception that is raised by ArithmeticException can be handled by Exception class as well. So the catch block of ArithmeticException will never get a chance to be executed which makes it redundant. Hence the compilation error.

Java Exception Keywords

- try
  - Used to specify a block where we should place exception code.
  - The try block must be followed by either catch or finally.
  - It means, you can't use try block alone.
- catch
  - 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.

Java Exception Keywords

- finally
  - The "finally" block is used to execute the important code of the program.
  - It is executed whether an exception is handled or not.
- throw
  - The "throw" keyword is used in method body to throw an exception.
- throws
  - The "throws" keyword is used to declare exceptions. It doesn't throw an exception.
  - It specifies that there may occur an exception in the method.
  - It is always used with method signature.

Java Exception Keywords

Java Finally Block

- The finally block is executed irrespective of an exception being raised in the try block. It is optional to use with a try block.

```
try {
    statement(s)
} catch (ExceptiontType name) {
    statement(s)
} finally {
    statement(s)
}
```

- In case, an exception is raised in the try block, finally block is executed after the catch block is executed.

Java Exception Keywords

Example : Finally Block

Step 1) Copy the following code into an editor and save the file.

```
class JavaException {
public static void main(String args[]) {
        try {
           int x = 10;
           int y = 0;
           int fraction = x / y;
        }
        catch (ArithmeticException e) {
           System.out.println("In the catch block due to Exception = " + e);
        }finally{
           System.out.println("Inside the finally block");
        }
    }
}
```

Java Exception Keywords

Step 2) Save, Compile & Run the Code.

Output :

```
In the catch block due to Exception = java.lang.ArithmeticException: / by zero
Inside the finally block
```

Step 3) Change the value of variable y = 2.

Save, Compile and Run the code and observe the output.

User Defined/Custom Exceptions Java

- Creating your own Exception is known as custom exception or user-defined exception.
- By the help of custom exception, you can have your own exception and message.
- Custom exception is creating your own exception class and throws that exception using throw keyword.
- This can be done by extending the class Exception.

User Defined Exceptions

- There is no need to override any of the methods available in the Exception class, in your derived class.
- But practically, you will require some amount of customizing as per your programming needs.

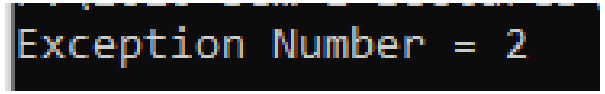
Example: To created a User-Defined Exception Class

Step 1) Copy the following code into an editor and save the file.

```
class MyException extends Exception{
   int a;
   MyException(int b) {
      a=b;
   }
   public String toString(){
    return ("Exception Number = "+a) ;
   }
 }
class JavaException{
  public static void main(String args[]){
     try{
       throw new MyException(2);
       // throw is used to create a new exception
            and throw it.
     } catch(MyException e){
      System.out.println(e) ;
     }
  }
}
```

User Defined Exceptions

Step 2) Compile & Run the code.

Expected output : 

- The keyword **throw** is used to create a new Exception and throw it to the catch block.

END