Exception Handling in Java

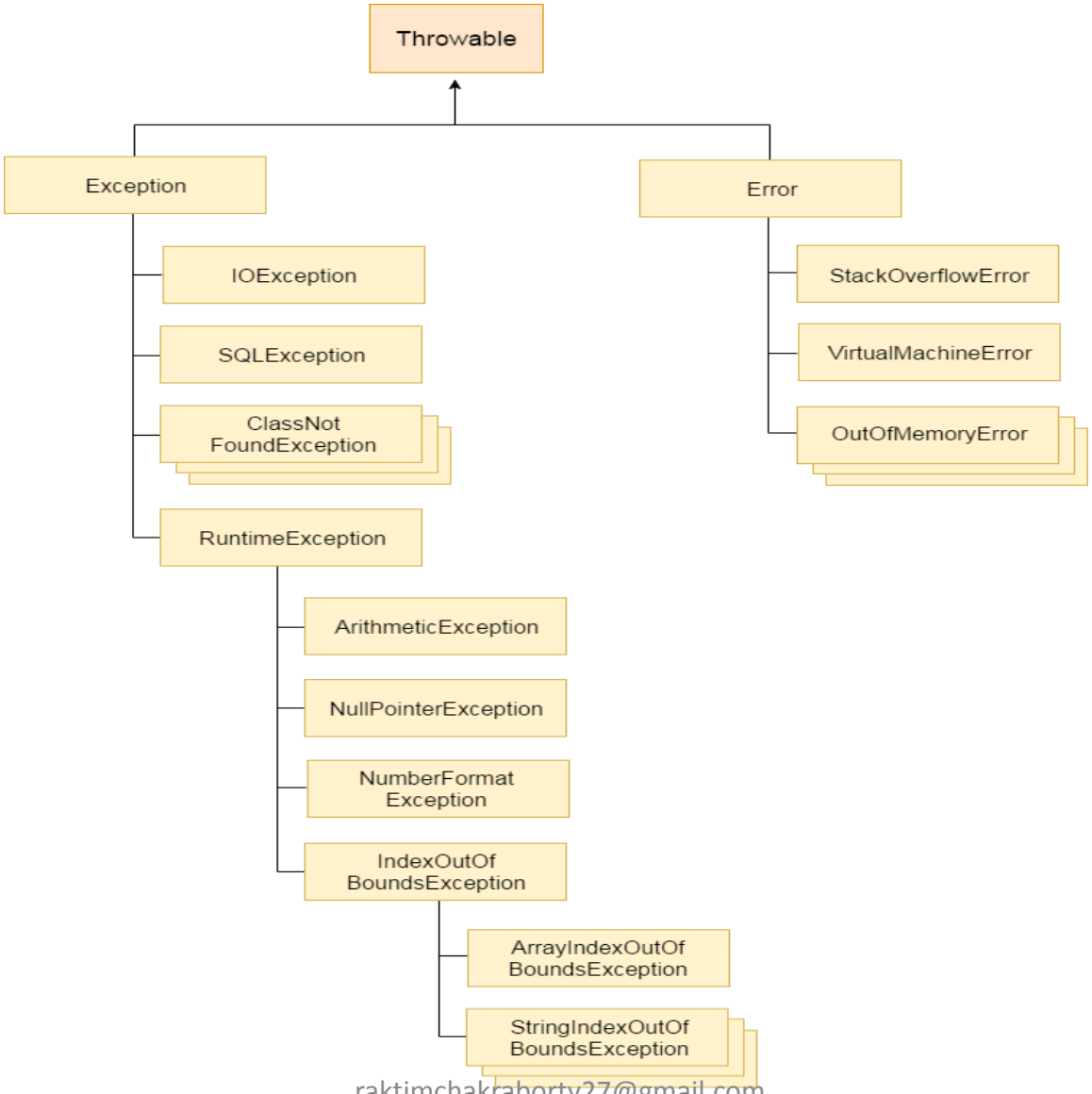
What is Exceptions

- The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.
- Exception is an abnormal condition.
- runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Advantage of Exception Handling

- The core advantage of exception handling is **to maintain the normal flow of the application**.
- An exception normally disrupts the normal flow of the application that is why we use exception handling.
- Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling

Hierarchy of Java Exception classes



Types of Java Exceptions

- According to Oracle, there are three types of exceptions:
- Checked Exception
- Unchecked Exception
- Error

Difference between Checked and Unchecked Exceptions

- 1) Checked Exception
- The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.
- 2) Unchecked Exception
- The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.
- 3) Error
- Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Keywords

- There are 5 keywords which are used in handling exceptions in Java.

try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
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 important 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 doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

final, finally and finalize in Java

- The final keyword can be used with class method and variable. A final class cannot be instantiated, a final method cannot be overridden and a final variable cannot be reassigned.
- The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred.
- The finalize() method is used just before object is destroyed and can be called just prior to object creation.

Example final

```
public class Tester {
  final int value = 10;

  // The following are examples of declaring constants:
  public static final int BOXWIDTH = 6;
  static final String TITLE = "Manager";
  public void changeValue() {
    value = 12;   // will give an error
  }
  public void displayValue(){
    System.out.println(value);
  }
  public static void main(String[] args) {
    Tester t = new Tester();
    t.changeValue();
    t.displayValue();
  }
}
```

Output
Compiler will throw an error during compilation.
Tester.java:9: error: cannot assign a value to final variable value value = 12; // will give an error ^ 1 error

Example finally

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

      try{
         int a = 10;
         int b = 0;
         int result = a/b;
      }
catch(Exception e){
         System.out.println("Error: "+ e.getMessage());
      }
      finally{
         System.out.println("Finished.");
      }
   }
}
```

Output
Error: / by zero Finished.

Example finalize

```
public class Tester {
   public void finalize() throws Throwable{
      System.out.println("Object garbage collected.");
   }
   public static void main(String[] args) {

      Tester t = new Tester();
      t = null;
      System.gc();
   }
}
```

Output
Object garbage collected.

Java Exception Handling Example

- Let's see an example of Java Exception Handling where we using a try-catch statement to handle the exception.

```
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:

Exception in thread main java.lang.ArithmeticException:/ by zero rest of the code...

Common Scenarios of Java Exceptions

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

The wrong formatting of any value may occur NumberFormatException. Suppose I have a string variable that has characters, converting this variable into digit will occur NumberFormatException.

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

4) A scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result in ArrayIndexOutOfBoundsException as shown below:

```
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

Java catch multiple exceptions

- 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.
- 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.

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

Example

```
public class MultipleCatchBlock2 {

   public static void main(String[] args) {

        try{
            int a[]=new int[5];

            System.out.println(a[10]);
           }
           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:
ArrayIndexOutOfBounds Exception occurs
rest of the code