Object Oriented Programming with Java
Practical-8

Introduction to Exception Handling

Java Exceptions

When executing Java code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an **exception** (throw an error).

Java try and catch

The `try` statement allows you to define a block of code to be tested for errors while it is being executed.

The `catch` statement allows you to define a block of code to be executed, if an error occurs in the try block.

The `try` and `catch` keywords come in pairs:

Syntax

```
try {
  //  Block of code to try
}
catch(Exception e) {
  //  Block of code to handle errors
}
```

Consider the following example:

This will generate an error, because **myNumbers[10]** does not exist.

```
public class Main {
  public static void main(String[ ] args) {
    int[] myNumbers = {1, 2, 3};
    System.out.println(myNumbers[10]); // error!
  }
}
```

The output will be something like this:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 10
        at Main.main(Main.java:4)
```

If an error occurs, we can use `try...catch` to catch the error and execute some code to handle it:

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.");
    }
  }
}
```

The output will be:

```
Something went wrong.
```

Finally

The `finally` statement lets you execute code, after `try...catch`, regardless of the result:

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.");
    }
  }
}
```

The output will be:

```
Something went wrong.
The 'try catch' is finished.
```

The throw keyword

The `throw` statement allows you to create a custom error.

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

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...)
  }
}
```

The output will be:

```
Exception in thread "main" java.lang.ArithmeticException: Access
denied - You must be at least 18 years old.
        at Main.checkAge(Main.java:4)
        at Main.main(Main.java:12)
```

If **age** was 20, you would **not** get an exception:

Example

```
checkAge(20);
```

The output will be:

```
Access granted - You are old enough!
```