# Throwing Exceptions

## *Objective*

In this lab, we will experiment with some of the basic coding involved with creating new exceptions and throwing and catching them.

## *Overview*

In this lab you will:

- Create a new exception.
- Write some code that throws the new exception
- Write some code that tries to execute the new code and handles the new exception.

## *Step by Step Instructions*

### Exercise 1: Creating an Exception

An exception is typically just a marker class which means that we tell different exceptions apart simply by their name.

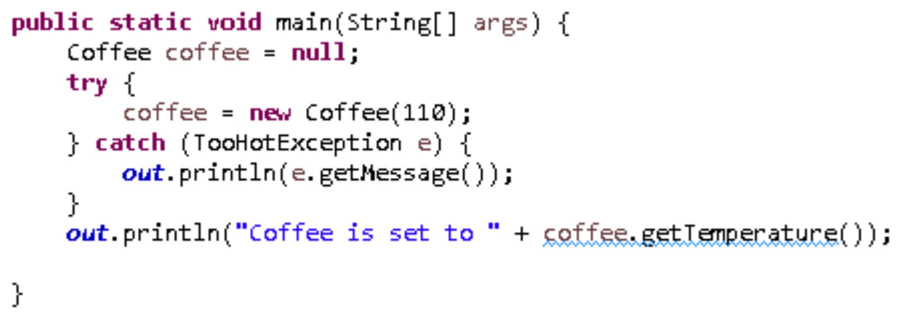
1. For this lab, we will work in the **ClassExercises** project and the **com.lq.exercises** package. Create a new exception class named **TooHotException**. Ensure that this new class extends **Exception**. Your class should have two constructors (Default and accepts String message. Delete any others as we will not use them. Make sure you have the appropriate super calls in each.

2. Create another new class named **Coffee**. Add a method named `setTemperature()` that accepts one parameter of type `int` and returns a `void`. Also, add an attribute to the **Coffee** class of type `int` named `temperature`. Add a `getTemperature()` method that returns the contents of `temperature`.

3. In the body of the `setTemperature()` method, check to see if the `temperature` that was passed in is greater than 120 degrees. If it is, `throw` a new **TooHotException** back to the caller of the method with a message that the coffee is too hot.. Otherwise, set the `temperature`. Remember to add the `throws` declaration to your `setTemperature()` method.

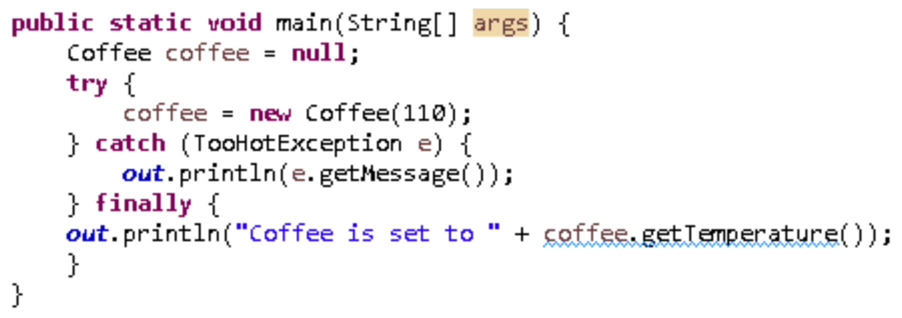
4. Add a constructor to accept a temperature.

5. Create a new class named **CoffeeExerciser**. Ensure that it has a `main()` method. In the body of the main method, create a **Coffee** object with a value of 110. You will have to do this inside of a `try` block and will also need a `catch` block to deal with the possibility of the exception getting thrown. At the very end of the `main()` method, call `getTemperature()` and print out the value that it contains to ensure things are working well. Your `main()` method should look similar to the following. Run your program and observe the output.



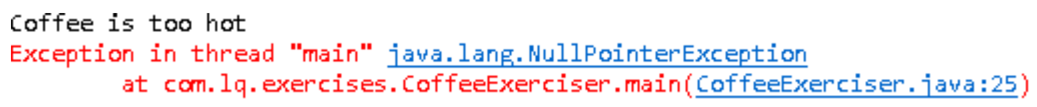
```
public static void main(String[] args) {
    Coffee coffee = null;
    try {
        coffee = new Coffee(110);
    } catch (TooHotException e) {
        out.println(e.getMessage());
    }
    out.println("Coffee is set to " + coffee.getTemperature());

}
```

6. Let's make a change so that we can see how `finally` blocks get processed. Move the print statement which is currently at the end of the `main()` method into a `finally` block. Re-run your program. Does the print statement still execute?



```
public static void main(String[] args) {
    Coffee coffee = null;
    try {
        coffee = new Coffee(110);
    } catch (TooHotException e) {
        out.println(e.getMessage());
    } finally {
    out.println("Coffee is set to " + coffee.getTemperature());
    }
}
```

7. Now, change the temperature from 110 to 125. Re-run your program. What prints this time? It should look as follows:



```
Coffee is too hot
Exception in thread "main" java.lang.NullPointerException
        at com.lq.exercises.CoffeeExerciser.main(CoffeeExerciser.java:25)
```

Because an exception was thrown, we never actually created the Coffee object and set `temperature` to a new value. But we did execute the `catch` block and the `finally` block. And since the `finally` block used the coffee reference, which was never set up an instance due to the **TooHotException**, a **NullPointerException** was thrown.

This kind of logic error is very common. In fact, it has been latent in the code since Step 5 introduced the original version of `main()`. Tracking down the root causes of exceptions is important, as is proper exception handling to prevent exceptions from catastrophically cascading.

8. Challenge Exercise: Can you make it so that the **CoffeeExerciser** does not fail with an exception, but rather reports that the temperature is 0, *i.e.*,

```
Coffee is too hot
Coffee is set to 0
```

   Hint: there are at least four different ways to make that happen.