# Week 4
# Exceptions

**Comment**

The purpose of this exercise is to give you practice handling exceptions and using enums, including where they are handled and how much of a method is affected when an exception is thrown. The code given for the exercise has no meaning or context apart from that purpose, and is not always style-guide compliant. You may need to make minor modifications in order to get the code to compile.

## 1 Exceptions

### 1.1 Setting up

Create a new class `UnknownException` inheriting from `Exception` in the appropriately named file. Then, create `PracticalFour.java` and `ExceptionEnum.java`, as follows:

```java
import java.io.FileNotFoundException;
import java.util.Random;

public class PracticalFour {

    private static void makeException(ExceptionEnum type) {
        // Your code here.
    }

    private static void f() {
        makeException(ExceptionEnum.NULL);
        makeException(ExceptionEnum.NONE);
        makeException(ExceptionEnum.MISSING);
        makeException(ExceptionEnum.NONE);
    }

    public static void main(String[] args) {
        f();
        /*
        Random random = new Random();
        for (int i = 0; i < 5; i++) {
            h(random);
        }
        */
    }
}
```

```java
public enum ExceptionEnum {
    NONE,
    NULL,
    BOUNDS,
    MISSING,
    UNKNOWN
}
```

### 1.2 makeException

The `makeException` method takes an `ExceptionEnum` as a parameter, and throws a different exception depending on the value of the parameter. The following table shows what exceptions should be thrown for different parameter values (i.e. the value of `type`).

| `type` | exception to throw |
|---|---|
| NULL | NullPointerException |
| BOUNDS | ArrayIndexOutOfBoundsException |
| NONE | - |
| MISSING | FileNotFoundException |
| UNKNOWN | UnknownException |

If the parameter `NONE` is given, print out “No problems".

### 1.3 Exception Handling

Modify the method `f()` provided above, so that exceptions thrown by calling `makeException` should be handled as follows:

1. `NullPointerException`s should be squashed (i.e. nothing is done), with the rest of the method continuing to execute as normal. Note that only an input of `ExceptionEnum.NULL` will cause `makeException` to throw a `NullPointerException`.
2. *Any other types of exception* should stop the rest of the method from executing and print out the exeception.

Running the `main` method should cause the following output to be printed:

```
No problems
java.io.FileNotFoundException
```

### 1.4 More Handling

Uncomment the code from `main` above and add the following additional methods to the `PracticalFour` class:

```java
private static void g(Random random) {
    int x = random.nextInt(ExceptionEnum.values().length);
    int y = random.nextInt(ExceptionEnum.values().length);

    makeException(ExceptionEnum.values()[x]);
    System.out.println("x = " + x);
    makeException(ExceptionEnum.values()[y]);
    System.out.println("y = " + y);
}

private static void h(Random random) {
    g(random);
    System.out.println("Reached here!");
}
```

Add exception handling code to the class to achieve the following behaviour:

g()

1. A `NullPointerException` or `ArrayIndexOutOfBoundsException` should stop the rest of the method from executing, and print out the exception.
2. All other execptions should propagate out of the method (i.e. be passed on to the calling method).

---

h()

1. `FileNotFoundException` should be printed if it occurs.
2. Any other execptions should propagate out of the method.
3. The print of "Reached here!" should execute regardless of what exceptions do or do not occur.

---

main()

1. Any exceptions which are thrown by calls to `h(random)` should be printed, but should not prevent further calls from occurring. *The catch you use here should be as specific as possible.*

### 1.5 Thinking

Is this a good way to do testing (using print statements throughout the code)?

- If so, what benefits are there?
- If not, how could it be improved?