Exceptions; try-Statements and Handlers

Weiss, section 2.5 (pages 47–51)
Exception

- Exception is an event that disrupts the normal flow of program’s instructions.
- When an error happens within a method, the method creates an object and hands it off to the runtime system.
- This object is called an exception object. It contains information about the error e.g., the type of error.
- Creating an exception object and handing it to the runtime system is called throwing an exception.
Exception

• The runtime system attempts to find a method to handle the exception.
• This method is one of methods that were called before the exception occurred.
• The list of methods is called call stack. The call stack is searched in the reverse order to find an exception handler.
• The handler needs to catch an appropriate exception.
Some code that throws an exception is inside a method. This method specifies that it can throw an exception.

```java
public class ListOfNumbers {
    private int[] array;
    private static int SIZE = 10;

    public ListOfNumbers() {
        array = new int[SIZE];
        for (int i = 0; i < SIZE; i++) {
            array[i] = i;
        }
    }

    public void printNumber(int i) {
        System.out.println(getNumber(i));
    }

    public int getNumber(int i) throws IndexOutOfBoundsException {
        return array[i];
    }
}
```
throw Statement

- Programmers can generate exceptions by using the **throw** clause.
- The throw statement requires a single argument: a throwable object.

```java
public Object pop( ) {
    Object obj;
    
    if (size == 0) {
        throw new EmptyStackException( );
    }
    // extracts an object, decreases stack size, and returns the object.
}
```

- Note that the **header** of the pop method **does not contain** a **throws** clause.
try and catch blocks

- The exception thrown by some illegal code is caught within a try block. That code is handled within a catch block.
- The body of catch block executes only when the type of exception it handles matches the thrown exception.

```java
public void printNumber(int i) {
    try {
        System.out.println(getNumber(i));
    } catch (IndexOutOfBoundsException e) {
        System.err.println(e.getMessage());
    }
}
```

If i = -1, getMessage() returns -1, which is stored inside the exception object.
Multiple Catch Blocks

```java
public void printNumber(int i) {
    try {
        if((i % 0) != 0) {
            System.out.println(getNumber(i));
        }
    } catch(OutOfRangeException e) {
        System.err.println(e.getMessage());
    } catch(ArithmeticException e) {
        System.err.println(e.getMessage());
    }
}
```

Output

```
/ by zero
```
finally block

• The finally block **always** executes when the try block exits w/o throwing an exception.
• Another use of finally block is to execute any code *cleanup* before program exits.

```java
public void printNumber(int i) {
    try {
        System.out.println(getNumber(i));
    } catch (IndexOutOfBoundsException e) {
        System.err.println(e.getMessage());
    } finally {
        System.out.println("finally block always executes.");
    }
}
```
NullPointerException

```java
public class NullPointerExceptionTest {

    public static void main(String[] args) {
        String str = null;
        try {
            if (str.equals("hello")) {
                System.out.println("Same");
            } else {
                System.out.println("Not same");
            }
        } catch (NullPointerException e) {
            e.printStackTrace();
        }
    }
}
```

Output

```
java.lang.NullPointerException
    at NullPointerExceptionTest.main(NullPointerExceptionTest.java:7)
```
NumberFormatException

- Occurs when converting a string to a numeric type.
- The value of the string is illegal for the conversion.

```java
public class NumberFormatExceptionTest {
    public static void main(String[] args) {
        String str = "123.33";
        try {
            int a = Integer.parseInt(str);
        } catch (NumberFormatException e) {
            e.printStackTrace();
        }
    }
}
```
Two Types of Exceptions

- Checked exceptions
  - They need to be caught otherwise program does not compile.
    - FileNotFoundException, EOFException

- Runtime (unchecked) exceptions
  - Programs does not expect these exceptions.
  - They occur due to programming bugs.
  - It is better to eliminate the bug that cause exceptions.
    - NullPointerException, ArithmeticException, IndexOutOfBoundsException
Propagate Exceptions

- When a **checked** exception object is thrown, it must also appear in the **throws** clause.
- Methods on the call stack may not handle the exception. In this case, they just need to **propagate** the exception using the **throws** clause.

```java
public void methodC( ) throws FileNotFoundException {
    methodD( );
}

public void methodD( ) throws FileNotFoundException {
    ...
    throw new FileNotFoundException( );
}
```