Tutorial 2: Exceptions

Exercise 1. Implement a standalone procedure to read in a file containing words and white space and produce a short version of the file in an output file. The short version should contain all of the words in the input file and none of the white space, except that it should preserve lines.

Exercise 2. Implement *search* as specified in Figure 4.1 in two ways: using *for* loops, and using *while (true)* loops that are terminated when accessing the array raises *IndexOutOfBoundsException*. Which implementation is better? Discuss.

Figure 4.1 Some specifications with exceptions

```
public static int fact (int n) throws NonPositiveException
   // EFFECTS: If n is non-positive, throws NonPositiveException, else
   //    returns the factorial of n.

public static int search (int[ ] a, int x)
      throws NullPointerException, NotFoundException
   // REQUIRES: a is sorted
   // EFFECTS: If a is null throws NullPointerException; else if x is not
   //    in a, throws NotFoundException; else returns i such that a[i] = x.
```

Exercise 3. A procedure that computes the sum of the elements in an array of integers might require a nonempty array. One way is to return 0 if the array is empty. Another way is to throw an exception if the array is empty. Provide the specification for both versions of the procedure and discuss which version is better.

Exercise 4. Consider a procedure

```
Public static void combine(int[] a, int[] b)
```

that multiplies each element of *a* by the sum of the elements of *b*; for example, if *a = [1, 2, 3]* and *b = [4, 5]*, then on return *a = [9, 18, 27]*. What should this procedure do if *a* or *b* is *null* or empty? Give a specification for *combine* that answers these questions and explain why your specification is a good one.

Exercise 5. Design and implement a procedure

```
Public static void divide(int[] a, int[] b)
```

that divides each element of array *a* by the element which has the same index from array *b*. If array *b* has less elements than array *a*, let *IndexOutOfBoundsException* be thrown and handle it by asking the user to either quit or provide the missing values to continue the calculation. Do similarly in the case of an *ArithmeticException* (division by error).

Exercise 6. Extend exercise 5 by creating 2 new Exception types to replace *IndexOutOfBoundsException* and *ArithmeticException*. Create new methods *printInfo* in these new exceptions to inform user about the situation (*) Hint: Catch *IndexOutOfBoundsException* and *ArithmeticException*, then delegate the handling by throwing a new Exception. Use multiple catch.

Submission

Submit a **zip** file containing all Java programs to this tutorial's submission box in the course website on FIT Portal.