Exercise Set: Developing Robust Classes

In this exercise set, we have marked questions we think are harder than others with a **[‡]**. We have also marked questions for which solutions are provided at the end of the set (**[SP]**). To check solutions for other questions than those marked with **[SP]**, ask one of the instructors or TAs or post a question to Piazza!

1. Answer the following questions with true or false and explain your choice in one sentence. **[SP]**

a) An operation specification is robust if it has an EFFECTS and a REQUIRES clause.

b) Exceptions in Java can help to make a method implementation more robust.

c) It is enough if all methods ensure the class invariant holds when they complete execution.

2. [SP] Given the following method specification:
a) Why is the specification not robust? (Explain briefly.)
b) Change the specification to make it more robust.

```
// Determines the first day of the month
// Requires: year is a valid year, month is a valid month
// Effects: returns the first day of the month
public static int getFirstDayOfaMonth(int month, int year)
```

3. Given the following code: how can you make it more robust? Discuss your solution.

```
// Determines the result of dividing 2000 by the integer n
// Requires: the integer n to be > 0
// Effects: returns 2000 divided by n
int divide2000(int n) {
      return 2000 / n;
}
```

4. What is the difference between a checked and an unchecked Exception?

SOLUTIONS:

1.
a) False.
An operation specification is robust when it covers all values that could be passed to the operation.

b) True.
You can use exceptions for illegal cases/values of the operation.

c) True.
If all methods ensure that the class invariant holds when they complete execution, the class invariant will also hold before all method of the class. (Note: this is only true if the fields of the class can only be accessed by other classes through the public methods of the class, which should usually be the case.)

2a) The method is not robust because the specification doesn't cover all possible values that could be passed as parameters. For example, it requires that we pass only those values of month that are valid (i.e, 1 through 12).

b)

```
// Determines the first day of the month
// Effects: if month is not a valid month or year is not a valid year
//          throws IllegalArgumentException,
//          otherwise returns the first day of the month
public static int getFirstDayOfaMonth(int month, int year)
      throws IllegalArgumentException
```