Exercise 1

Given the following program with input variables \( x \), \( y \) and expressions \( e_1 \ldots e_8 \). The expressions \( e_1 \ldots e_8 \) will not change the program variables.

\[
e_1; \\
\text{while}(x > 0) \{ \\
\quad e_2; \\
\quad \text{if}(y > 0) \{ \\
\quad\quad e_3; \\
\quad\} \text{ else } \{ \\
\quad\quad e_4; \\
\quad\} \\
\quad \text{if}(x \mod y == 0) \{ \\
\quad\quad e_5; \\
\quad\} \text{ else } \{ \\
\quad\quad e_6; \\
\quad\} \\
\quad e_7; \\
\} \\
e_8;
\]

1. Create a set of Test Cases to achieve Full Line Coverage. How many Test Cases do you need?

2. Create a set of Test Cases to achieve Full Branch Coverage. How many Test Cases do you need?

3. Create a set of Test Cases to achieve Path Coverage. How many Test Cases do you need?

4. Assume that the expression \( e_2 \) will increase the value of \( y \). How many Test Cases do you need to achieve a Full Branch Coverage? Justify your answer.

5. Assume that the expression \( e_7 \) will decrease the value of \( x \). How many Test Cases do you need to achieve a Full Path Coverage? Justify your answer.

6. Which kind of software testing is this?
Exercise 2

Given the following function `sort` with input `X[] x`. The function specification requires a list with elements of type `X`.

```java
public X[] sort(X[] x) {...}
```

Provide Test Cases for Black-Box-Testing to specify if the program works correctly. How many test cases do you need? Justify your answer and describe the purpose behind each testcase.

Exercise 3

Consider the following method taking an object implementing the `IErrorLogger` interface and a number of other parameters that potentially logs an error depending on two complex predicates concerning the other parameters.

```java
public void logErrorIfNeeded(IErrorLogger logger, ...) {
    if (isCriticalErrorPredicate)
        logger.logError(true);
    else if (isNonCriticalErrorPredicate)
        logger.logError(false);
}
```

The only externally observable effect of the method is the call to the `logError` method. This makes writing a unit test for this method not exactly straightforward.

1. Rewrite this method in such a way that the complex predicates become testable in isolation.
2. Let’s assume rewriting the production code is not an option for you. Write a mock-up class for the `IErrorLogger` so that you can still unit test the method under test. Your mockup should be able to record the call to `logError` in order to enable your unit test to make sure the method under test had the expected effect.
3. Can you use a similar mockup class, if the type of `logger` parameter changes to `class ErrorLogger`, such that the call to `logError` becomes a normal virtual method call? Justify your answer.
4. In which way will the situation change if `ErrorLogger` is a final class? Or if `logErrorIfNeeded` uses a static method call to log the error (see below)?

```java
public void logErrorIfNeeded(...) {
    if (isCriticalErrorPredicate)
        ErrorLogger.logError(true);
    else if (isNonCriticalErrorPredicate)
        ErrorLogger.logError(false);
}
```
Or if logErrorIfNeeded creates the required logger instance itself?

```java
public void logErrorIfNeeded(...) {
    ErrorLogger logger = new ErrorLogger();
    if (isCriticalErrorPredicate)
        logger.logError(true);
    else if (isNonCriticalErrorPredicate)
        logger.logError(false);
}
```

Are you still able to write a mockup class for the ErrorLogger in the situations described above? Justify your answer.

5. Find out how testing frameworks like JMockit (http://code.google.com/p/jmockit/) or PowerMock (http://code.google.com/p/powermock/) are still able to “mock” static methods, final classes and constructors.