ECS505U Software Engineering

Laboratory Exercise (Assessed, 5% of final mark)

Software Testing and Test-Driven Development (TDD)

Version 2.2, 01/12/2025

1. Background

In a brief overview of the assignment, you will be required to test a program by applying some of the techniques discusses during the Software Testing (Week 9) lecture.

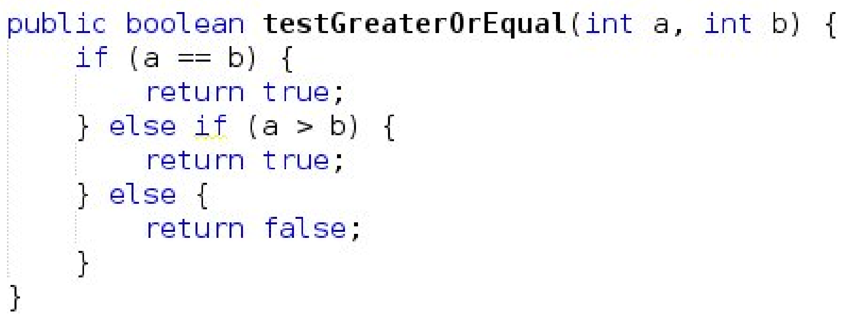
2. Prerequisites

- You should be able to run a recent version of IntelliJ on a computer. IntelliJ is installed on the ITL machines for campus students. Non-campus students install IntelliJ 2025.X (instructions available on QMPlus).
- You should have gone through the lecture slides of week 9.

3. Black-box Testing

In this exercise, you are given a class (MyMath with one function power) in a jar file without the access to its implementation details. You are expected to perform black box testing techniques discussed in the lecture to reveal any defects. The required techniques are equivalence portioning and boundary value analysis.

Equivalence partitioning (also called Equivalence Class Partitioning) is a software testing technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived.



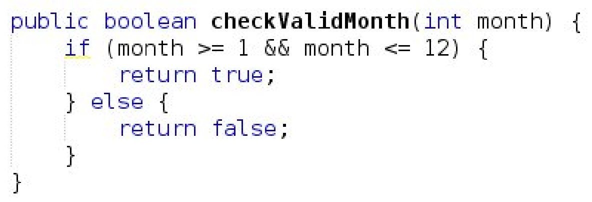
```java
public boolean testGreaterOrEqual(int a, int b) {
    if (a == b) {
        return true;
    } else if (a > b) {
        return true;
    } else {
        return false;
    }
}
```

Figure 1: Example Java function

For the code above equivalence partitions are the inputs that causes the software to exhibit the same behavior.

1. For the first if statement, all possible int values (-2^{31} to $2^{31}-1$) where *a* is equal to *b*
   For example, a is 5 and b is 5
2. For the second if statement, all possible int values where *a* is greater than *b*
   For example, a is 5 and b is 3
3. For the else statement, all possible int values where *a* is less than *b*
   For example, a is 4 and b is 6

Boundary testing or boundary value analysis is where test cases are generated using the extremes of the input domain, e.g. maximum, minimum, just inside/outside boundaries, typical values, and error values. It is like Equivalence Partitioning but focuses on "corner cases".



```java
public boolean checkValidMonth(int month) {
    if (month >= 1 && month <= 12) {
        return true;
    } else {
        return false;
    }
}
```

For the code above equivalence class are three:

1. Month is between 1 and 12. Boundary test for this class is
   a. Inside min input 1 and max input 12
   b. Outside min input 0 and max input 13
2. Month is less than 1(to -2^{31})
   a. Inside min input -2^{31}
   b. Outside min input not possible since it's less than min number capacity of integer
   c. Inside max input 1
   d. Outside max input 0
3. Month is greater than 12 (to 2^{31}-1)
   a. Inside min input 13
   b. Outside min input 12
   c. Inside max input 2^{31}-1
   d. Outside max input not possible since it's more than max number capacity of integer

In order to perform boundary analysis, we only need to test the code with the following inputs: -2^{31},0,1,12,13 and 2^{31}-1.

4. Black Box Testing Exercise (Lab exercise begins here!)

MyMath minTwoToPower function specifications:

Returns the value of the minus two raised to the power of the argument (exponent)[1] up to 1.7976931348623157E308 (or $(2-2^{-52}) \times 2^{1023}$: double max[2]).

- If the exponent is 1024 or more, then returns
  - positive infinity if the exponent is even
  - negative infinity if the exponent is odd (inc. int max)
- If the exponent is
  - 1023 then returns negative double max
  - -1074 then double min
- If the exponent is -1075 or less, then returns
  - 0.0 if the exponent is even (inc. int min)
  - -0.0 if the exponent is odd
- If the exponent is 1 then returns -2
- If the exponent is 0 then returns 1.0
- If the exponent is -1 then returns -0.5

[1] Calculates $(-2)^{exponent}$ for the given exponent

[2] https://docs.oracle.com/javase/7/docs/api/constant-values.html#java.lang.Double.MAX_VALUE

- Identify the equivalence partitions based on the given specifications
- Write the minimum set of tests to cover all the identified partitions.
- Test and document test results in the test results document.

5. Branch and Path Coverage Exercise

Suppose you are asked to test the following C++ method.

```
int checkInputLength(char[] content) {//dynamic array container
      int length = content.length; //get number of chars in the container
         if(length > 20)
            return 1;
         if(length > 1)
            return 0;
         if(length == 0)
            return -1;
   }
```

- List the minimum amount of test cases that will cover all branches.
- List all paths in this code including the infeasible ones.
- List the minimum amount of test cases that will cover all feasible paths.
- Did any of test suites that executes of all paths, or the feasible paths reveal any defects?
- Enter your test cases and results in the test report document.

6. Challenge - Branch Coverage (**Optional unmarked – do it if you want to challenge yourself**)

In this part of the exercise, you are expected to find the test suite (with minimum number of test cases) that covers all branches in the code included in the BranchCoverageSourceCode.java file.

The file BranchCoverageSourceCode.java includes the source code for a small code with few branches. BlackBox.jar includes a class called “BranchCoverage” which is an instrumented version of the code in the BranchCoverageSourceCode.java. The instrumented code calculates the coverage of the provided test suite after test runs and provide the coverage score with getCoverage method.

You can use the example Junit test code in the BranchCoverageTest.java file to test your code. When finished, add your test cases and coverage result in the test report document.

HINT: I recommend performing mental execution (on paper) to find out the branches covered by your inputs. Integer inputs between -2 and 10 should be enough to cover all branches. You can also instrument the code under test (add instrumentation code to give you the branches covered) and find out the covered branches.

7. Test Driven Development Exercise

- Complete the code in TemperatureConverter class to pass all the unit tests in TemperatureConverterTest. The specifications of the class given below.
- Add the completed source code of TemperatureConverter class to the report document and submit the file when finished.

TemperatureConverter class is used for converting temperatures and it converts between Celsius, Fahrenheit and Kelvin temperatures. The class uses the formulas below for conversion.

From	To Fahrenheit	To Celsius	To Kelvin
Fahrenheit (F)	F	(F - 32) / 1.8	(F + 459.67) / 1.8
Celsius (C)	(C * 1.8) + 32	C	C + 273.15
Kelvin (K)	(K * 1.8) - 459.67	K - 273.15	K

For each temperature conversion the class has a method such as K2C which converts given Kelvin temperature to Celsius and F2K which converts given Fahrenheit temperature to Kelvin.

The class converts temperatures between min and max values (inclusive) given below. If a temperature outside these limits is provided the conversion method throws an illegal argument exception.

Temp	Max	Min
Fahrenheit (F)	9900000000032	- 459.67
Celsius (C)	5500000000000	- 273.15
Kelvin (K)	5500000000273.15	0

Class provides these max and min limits statically without needing to create an instance.