Assignment 2: Software Testing

Course: IN3240/4240 Software Testing

Delivery format: Compressed file (.zip or .tar file) with the group's username(s)

Assignment deadline: **29/04/2019** kl. **23:59**

General information:

The assignment should be solved in groups of 2 – 4 students, unless you have been granted other arrangements. It consists of four parts. Which part you shall solve is depending on which course you are following:
Project groups consisting only of bachelor students: Part 2 and 3 are mandatory. In addition, part 1 or 4 must be solved.
Project groups consisting of at least one master student: Part 2, 3, and 4 are mandatory.

Otherwise it is important to read the assignment carefully and perform the tasks as described. Be precise when answering the questions. You may write in Norwegian.

If you find aspects of this assignment confusing or ambiguous, do not hesitate to contact any of the group teachers for guidance.

Part 1: JUnit – Unit Testing

This part of the assignment is about getting familiar with the Java unit testing framework tool. The following requires that you use Eclipse as an IDE. [1]

Background

A ***prime number*** is an integer greater than 1 that is divisible only by itself and by 1. The sequence of prime numbers thus starts with *2, 3, 5, 7, 11, 13, ...*

Prime number plays an important role in many contexts, even when it comes to data processing. It is therefore of interest to know effective algorithms to find prime numbers. Such an algorithm, dating back to ancient times, is called Erathostenes' sold [2]. This can be described as follows:

We think that we write on a paper all integers from 2 upwards to a chosen limit n. Then we proceed as follows: First, we iron over all multiples of *2 (ie 4, 6, 8, 10 , ...).* We then find first numbers that have not been deleted. It will be 3. Then we iron over all multiples of *3 (ie 9, 12, 15, 18, ...).* We continue in the same way: Now finds the first number that has not been deleted, it will be 5. Then we iron over all multiples of *5 (that is, 10, 15, 20, ...).* Here we go. When we finish this process, all numbers that are not exceeded will be prime numbers *(ie 2, 3, 5, 7, 11, ...).*

Exercise 1

Write a program which does the following:

1.1 **Decides** whether or not a given number is a prime number or not.

1.2 **Finds and return** all the prime numbers up to a given limit.

To find the prime numbers, proceed as follows:

Create an array of logical values with as many places as the set limit. Initialize it to true in all places.

The array indexes will correspond to the positive whole numbers. (We ignore 0 and 1.) The goal is now to set to false all places in the array that have an array index that is not prime so that when we are done we find the prime numbers as all the array indices where the array is still value threaten. To get this, you go through the index from index 2 and up. If it is true on an index space in your value, you go through the rest of the array by setting false all places that have an index that is a multiple of i.

You shall use Unit-testing to test the method(s) in the program. Create different test cases for each methods and combine all the test cases in a test suite

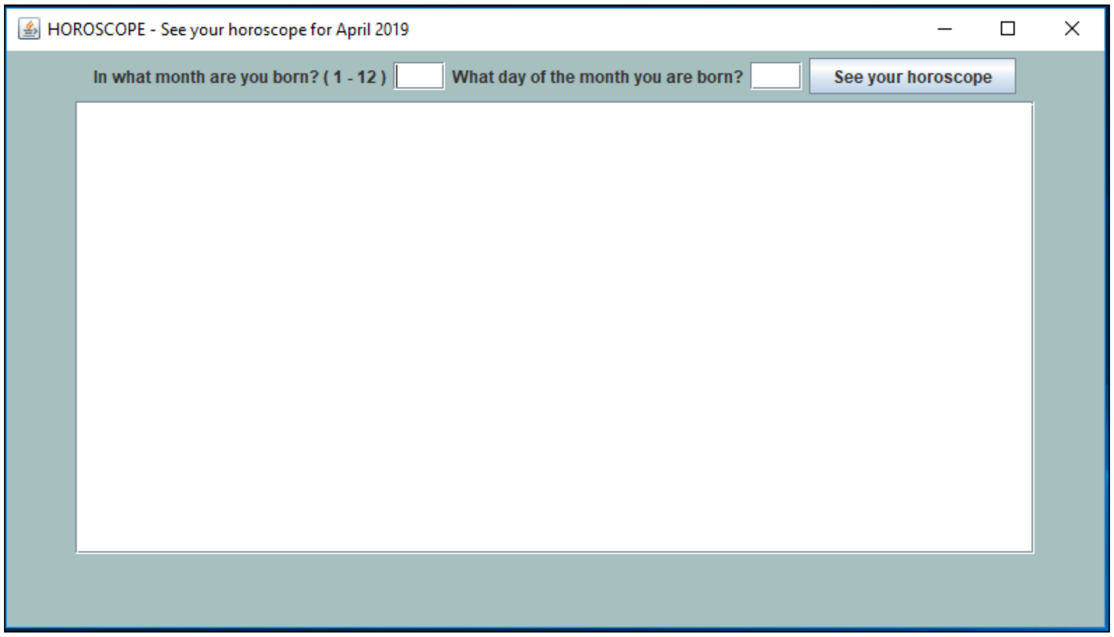
[1]Unit Testing with Eclipse Tutorial – `https://www.youtube.com/watch?v=v2F49zLLj-8`

[2]https://no.wikipedia.org/wiki/Eratosthenes%27$_{s}il$

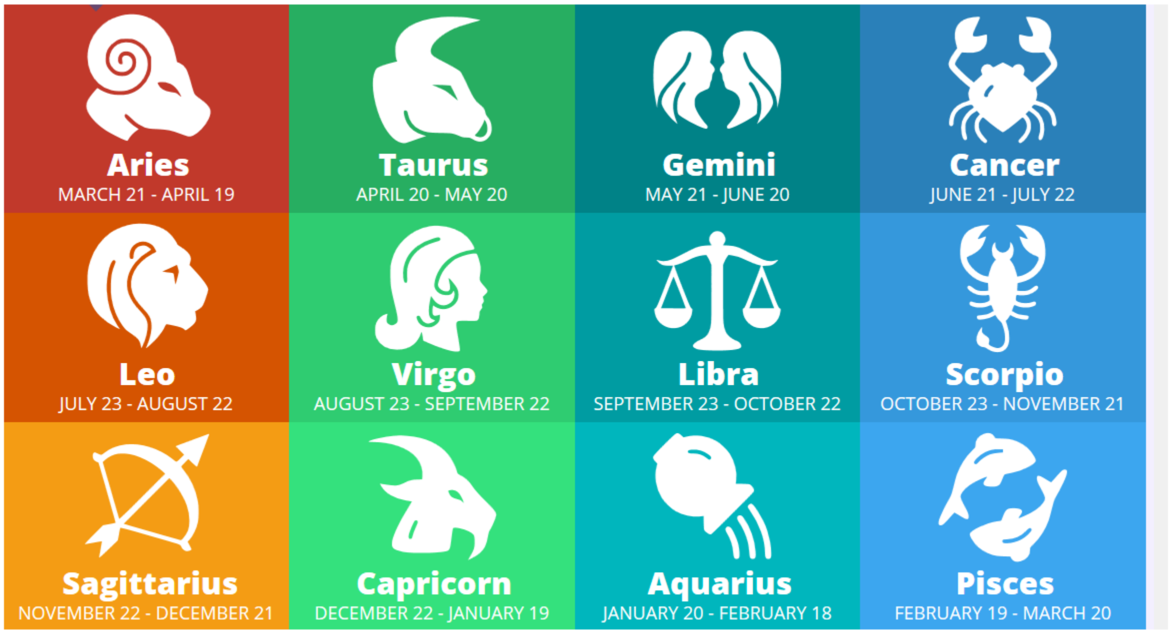
Part 2

Exercise 2

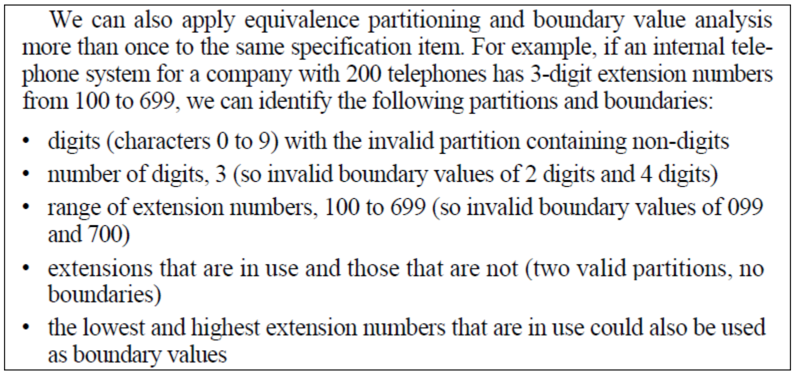
In this exercise you shall use *equivalence partitions* and *boundary value analysis* to detect failures in a little Java program that sets your horoscope based on your birthday:


HOROSCOPE - See your horoscope for April 2019
In what month are you born? (1 - 12)
What day of the month you are born?
See your horoscope

You will see the various dates for the different zodiac signs below:



The equivalence partitioning can be done in different ways on the same test object. Some of them will contain boundary values, others not. The following example from the textbook page 86 illustrates this:



We can also apply equivalence partitioning and boundary value analysis more than once to the same specification item. For example, if an internal telephone system for a company with 200 telephones has 3-digit extension numbers from 100 to 699, we can identify the following partitions and boundaries:

- digits (characters 0 to 9) with the invalid partition containing non-digits
- number of digits, 3 (so invalid boundary values of 2 digits and 4 digits)
- range of extension numbers, 100 to 699 (so invalid boundary values of 099 and 700)
- extensions that are in use and those that are not (two valid partitions, no boundaries)
- the lowest and highest extension numbers that are in use could also be used as boundary values

2.1 In which ways can you apply equivalence partitioning to the input of the horoscope program? For each way, specify

   i) the equivalence partitions, both valid and invalid

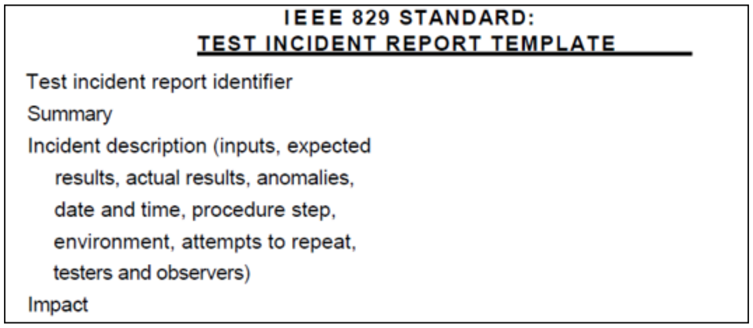
   ii) any boundary values

   Hint: You don't need to list ***every*** equivalence partition and its boundary values. It is sufficient to describe them uniquely as sets, intervals or in words.

   You shall now test the program by using the boundary values and values from the equivalence partitions. Click here[3] and choose 'open' to run the program.

2.2 There are at least three defects/incidents in this program (there may be more). Can you find them? Consider the degree of severity of the failures. In which order will you prioritize correcting the defects?

2.3 Use the template below to write an incident report about the defects/incidents you have discovered.



IEEE 829 STANDARD:
TEST INCIDENT REPORT TEMPLATE

Test incident report identifier
Summary
Incident description (inputs, expected
  results, actual results, anomalies,
  date and time, procedure step,
  environment, attempts to repeat,
  testers and observers)
Impact

[3]http://www.cs.hioa.no/ evav/hioa-dev/Horoscope19.jar

Exercise 3

In order to reduce the amount of air pollution in the city, a City Council has decided to introduce a new system for calculating road charges for passage of the toll ring around the center.

You shall now use decision tables to determine all combinations of conditions to explore the business rules that should be tested according to this system.

• The "rush hour" is defined as the period from 07:30 until 09:30, and later from 15:30 until 17:30.

• Electric vehicles are not charged outside the "rush hour" period, but are charged a fee of 30 NOK during peak times.

• Petrol- and diesel vehicles are required to pay 50 NOK outside the "rush hour" period, and get an additional 100 percent increase in the price during peak times.

3.1 Set up a decision table with the different business rules, including the conditions and related price. There should be one rule for each combination of conditions.

If there are any combinations (rules) that should not occur i in practice, you can mark these combinations (rules) with an 'X' in the action part of the table.

3.2 Reduce the number of rules by simplifying (rationalizing) the decision table, without losing any of the system functionality.

(In this exercise, we have specifically chosen to disregard buses and trailers.)

Part 3

Scenario: A check-in machine is installed at an airport. A detailed description of how to use this machine is as follows:

To use the check-in machine adhere to the following procedure:

1 *You can choose to check in with your booking number or with your credit card:*

- a) *You have chosen to check in with your booking number. Enter your number and click "Continue".*
  - i) *The machine validates your booking numbers*
  - ii) *If your number is correct, the machine goes to 2*
  - iii) *If the number is not correct, the machine goes back to 1.a., where you are able to try again*
- b) *You have chosen to check in with your credit card. Put your card into the machine.*
  - i) *The machine validates the card*
  - ii) *If the card is accepted, the machine goes to 2*
  - iii) *If the card is rejected, the machine goes to 1*

2 *You will now be prompted for the number of luggage items you want to check in. When you have entered the number, you click "Continue" and the machine goes to 3*

3 *You will now be asked if you want to reserve a seat or change an existing seat reservation (if you have already reserved a seat)*

- a) *If you confirm, the machine goes to 4*
- b) *If you do not confirm, the machine goes to 5*

4 *You choose an available seat and click "Continue". The machine goes to 5.*

5 *Your boarding card will be printed out along with the luggage tags. When you have taken the boarding card and luggage tags, the machine goes to 6.*

6 *The machine wishes you an enjoyable flight, and goes back to 1.*

During interaction with the machine, you always have the possibility to click "Cancel". If you do so, the machine will immediately go back to 1.

Exercise 4

In this exercise you shall use *State Transition diagrams* and *State Transition tables* to describe the use of the check-in machine.

4.1 Draw a state transition diagram that shows how the machine works.

4.2 How do you measure

i) State coverage?
ii) Transition coverage?

4.3 Write down a sequence of states to achieve 100 percent transition coverage. What is the state coverage of the same sequence? Is there any relationship between transition coverage and state coverage?

Exercise 5

In this exercise, you shall *use case testing* to test the check-in machine.

5.1 What kind of scenarios will be the main focus in your use cases?

5.2 Write a testing procedure to test the check-in machine.

Part 4

Essay

Agile software development is an approach under which requirements and solutions evolve through the collaboration between the development team and the customer/product owner. How will agile software development affect the testing activities? Discuss issues and challenges in adapting the testing activities to an agile development process related to the following:

- Fundamental test process
- Testing principles
- V-model
- Test levels and test types
- Testing techniques
- Test management
- Team coordination
- Exploratory testing
- Test-Driven Development

The essay should be between 1000 and 1500 word.