Analysis, Design, and Software Architecture

(3rd Semester SWU)

Exam Simulation

November 4th, 2021

Instructions[1]

- Do not turn over until told to do so by the exam invigilator.
- This exam comprises of 10 pages (including this one). There are 5 questions containing exercises (sub-questions). Each exercise is worth the number of points specified at the beginning of the exercise, which sum up to a total of 28 points.
- You must achieve at least 40% of the marks to pass, i.e., 11 points.
- You have 1 hour to complete this exam.
- Neither written material nor electronic equipment may be brought into the examination room.
- Your handwritten code does not have to be 100% syntactically correct. It is better to make an educated guess than omit an answer for fear of not getting it right. Think of it as this: If *Intellisense* would have saved you in an IDE, you will probably get full points for your answer.
- You are strongly encouraged to write your final answers within the spaces provided (note: these provide you also with hints for the answers). If you really have to, you may use separate sheets of paper for these questions instead, but please, be clear about it: cross out everything and write "see separate paper, page 1" or something like that in the provided space and write the correct answer code in the extra sheet, e.g., "question 1, point a". Finally, as a wise professor once stated: "for the love of all that is Good and Holy, write legibly".
- Hand in all pages you received and any additional one you might have used for your final answers.
- Remember to fill out the header of all pages with ITU user and name.

[1] These are an adaptation for completeness. There is no requirement related to how well you perform the following exercises to pass the mandatory activity connected to this simulation.

Your ITU user:	....
Your name:	....

Question 1

An important device encountered time and again in a university is the coffee machine. Let us consider the "Moccamaster" filter coffee machine that can be seen around the ITU campus, and let us simplify its "core hardware and competencies". With regards to hardware, let us assume that the coffee machine consists of a water tank, a heating plate, a coffee pot, a water pipe that leads from the water container to the filter, and an on/off switch.

Inner workings: as soon as the coffee machine is switched on, it executes the action **`Heat coffee`**, which turns on the heating plate. When the coffee machine has been fully prepared for making coffee, that is, when ground coffee on a filter and water have been added, the keep warm function is switched off and coffee is made. The signal **`Switch off`** – sent when the on/off switch is set to off – ends the entire process.

Note: some elements have in the description above and the questions below been written using **`this format`**; they are elements of interest that are worth considering for your answers.

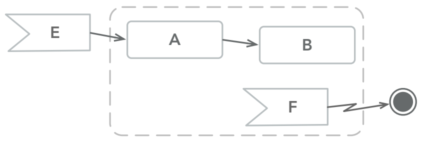
a. `(2 points)` Fill out the as-is scenario describing **`Martin`**, a regular user of the machine, preparing a coffee with the "Moccamaster" device detailed above. The machine does neither have water in the tank nor ground coffee in a clean filter.

Scenario name	brewCoffee
Participating actor instances	....
Flow of events	....

Your ITU user: ....

Your name: ....

b. (6 points) Draw a UML **activity** diagram that captures the inner workings described above. You might want to use the additional UML elements described below.
Important: An event can be modelled with an *accept event action* that waits for the occurrence of a specific (external) event, which can then trigger an action. The handling of exceptions can be modelled with an event inside an *interruptible activity region*, which allows to specify a group or actions which execution is terminated immediately if an event occurs. The notation element for an *accept event action* is a rectangle with a tip that points inwards from the left. The notation element for an *interruptible activity region* is a dashed rectangle with rounded corners that encloses the relevant actions. The edge exiting the region is in lightning bolt form.
In the example below: **E** and **F** are the external events; **E** triggers with a regular edge the action **A**; the set of actions **A** and **B** are in an interruptible activity region connected to the event **F** that, if triggered, leads to the activity final node via the lightning bolt edge.
Note: the diagram only depicts the necessary elements to showcase the use of these new elements.



Use this space between the horizontal lines to draw your final UML diagram.

Your ITU user: ....

Your name: ....

Question 2

On the **`repository`** architectural design pattern.

a. `(1 point)` Explain the problem the pattern provides a solution to.

....

....

....

b. `(1 point)` Name one severe weakness of the pattern.

....

....

c. `(2 points)` Provide in text an example showcasing a situation that demonstrates the effectiveness of the pattern.

....

....

....

....

d. `(2 points)` Draw a UML diagram depicting the example described above (the type of diagram is for you to choose).

Use this space between the horizontal lines to draw your final UML diagram.

Your ITU user: ....

Your name: ....

Question 3

Given the following code:

```
public class CoffeeDrink { ... }
public class GroundCoffee { ... }

public enum CoffeeSelection { FilterCoffee, Espresso }

public class BrewingUnit
{
    public CoffeeDrink Brew(CoffeeSelection coffee, GroundCoffee
        groundCoffee, int quantityWater)
    {
        ...
    }
}

public interface ICoffeeMachine
{
    CoffeeDrink BrewFilterCoffee();
    void AddGroundCoffee(GroundCoffee newCoffee);
    CoffeeDrink BrewEspresso();
}

public class BasicCoffeeMachine : ICoffeeMachine
{
    private GroundCoffee _groundCoffee;
    private BrewingUnit _brewingUnit;

    public BasicCoffeeMachine(GroundCoffee coffee)
    {
        _groundCoffee = coffee;
        _brewingUnit = new BrewingUnit();
    }

    public CoffeeDrink BrewFilterCoffee()
    {
        // brew a filter coffee
        return _brewingUnit.Brew(CoffeeSelection.FILTER_COFFEE,
            _groundCoffee, 118);
    }

    public void AddGroundCoffee(GroundCoffee newCoffee)
    {
        if (_groundCoffee != null)
        {
            if (_groundCoffee.Name == newCoffee.Name)
            {
```

Your ITU user: ....

Your name: ....

```
                _groundCoffee.Quantity = _groundCoffee.Quantity + newCoffee.Quantity;
            }
            else
            {
                throw new ArgumentException("Only one kind of coffee supported for each CoffeeSelection.");
            }
        }
        else
        {
            _groundCoffee = newCoffee;
        }
    }

    public CoffeeDrink BrewEspresso()
    {
        throw new NotImplementedException();
    }
}

public class EspressoMachine : ICoffeeMachine
{
    private GroundCoffee _groundCoffee;
    private BrewingUnit _brewingUnit;

    public EspressoMachine(GroundCoffee coffee)
    {
        _groundCoffee = coffee;
        _brewingUnit = new BrewingUnit();
    }

    public CoffeeDrink BrewFilterCoffee()
    {
        throw new NotSupportedException("This machine only brew espresso.");
    }

    public void AddGroundCoffee(GroundCoffee newCoffee)
    {
        if (_groundCoffee != null)
        {
            if (_groundCoffee.Name == newCoffee.Name)
            {
                _groundCoffee.Quantity = _groundCoffee.Quantity + newCoffee.Quantity;
            }
            else
            {
                throw new ArgumentException("Only one kind of coffee supported for each CoffeeSelection.");
```

Your ITU user: ....

Your name: ....

```
            }
        }
        else
        {
            _groundCoffee = newCoffee;
        }
    }

    public CoffeeDrink BrewEspresso()
    {
        return _brewingUnit.Brew(CoffeeSelection.ESPRESSO,
            _groundCoffee, 30);
    }
}
```

a (4 points) Which principle(s) are broken? Explain why.

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

b (4 points) Fix the code given your answer above. You do not need to implement everything, just write the code pieces which support your solution.

Your ITU user: ....

Your name: ....

Your ITU user: ....

Your name: ....

.

Question 4

Given the following class:

```
public static class Parser
{
    public static (string code, string locality) Parse(string input)
    {
        throw new NotImplementedException();
    }
}
```

a (3 points) Write a set of tests which validate the following:

Input	**Expected output**
"100 Tórshavn"	("100", "Tórshavn")
"1000 København K"	("1000", "København K")
"Jibberish"	(null, null)

Your ITU user: ....

Your name: ....

.

Question 5

`(3 points)` For each of the statements below, mark whether the statement is true or false. Each correct answer gains you 1 mark; each incorrect answer loses you 1 mark; a question left unanswered neither loses nor gains marks. The final points for this question is calculated by bounding the sum of marks between 0 and 3 points. In other words, each answer has a positive or negative impact of half a point.

Id	**True**	**False**	**Statement**
i.	☐	☐	UML scenarios include both a visual representation as well as a written description.
ii.	☐	☐	One of the challenges in software engineering is that the knowledge acquisition process is not linear.
iii.	☐	☐	UML scenarios can be of different types including: as-is, visionary, evaluation, and training.
iv.	☐	☐	Techniques to detect faults in software systems include: unit testing, integration testing, and system testing.
v.	☐	☐	Plan driven software development processes rely on quick development steps, feedback, and adaptation to clarify the requirements and design.
vi.	☐	☐	Design patterns are a fixed set of well documented solutions to known problems. Design patterns are grouped into three categories: structural, creational, and behavioral.