2021/2022 EXAM GUIDE

CONTENTS

ENTERING THE EXAMINATION

- Maximum number of seminar absences allowed: **4.**
- Maximum number of laboratory absences allowed: **2.**
- If you do not have the minimum number of attendances, you cannot enter examination during the **regular, or retake** session during 2022.
- Minimum laboratory grade to enter examination during regular session: **5.00** (no rounding).

STUDENTS WHO CANNOT TAKE PART IN EXAMINATION

- In case there are issues with your seminar situation we will contact you privately using MS Teams
- In case your lab grade is < **5.00** you may only attend the retake examination.
- In case you have not paid your school taxes, you are not allowed to take the exam (check the latest announcements on the faculty website).

- **Make sure to verify your situation and notify us of any issues as soon as possible!**

OBJECTIVE AND COURSE CONTENTS

OBJECTIVES

- Being familiar with some fundamental concepts in computer programming
- Introduction to basic concepts in software engineering
- Learn elements of software design, architecture, implementation and maintenance
- Familiarization with some of the software tools used in large-scale application development (unit testing, coverage)
- Basic knowledge of the Python 3.x programming language
- Using the Python language in software development, testing, running and debugging applications
- Learning/honing your own programming style ☺

COURSE CONTENT

PROGRAMMING IN THE LARGE

1. Introduction
2. Procedural programming. Compound Types
3. Test Driven Development
4. Modular Programming
5. Design guidelines in large scale programming
6. Exceptions
7. User Defined Types
8. UML. Design Principles.
9. Layered architecture. Inheritance.
10. Program Testing. Refactoring.

PROGRAMMING IN THE SMALL

11. Recursion. Computational Complexity
12. Searching. Sorting
13. Problem solving methods

EVALUATION

DURING THE SEMESTER

[L] 40% Laboratory – grade based on your activity during the semester

- 50% - Weighted average of 11 lab assignments, with 0 for the labs you were not graded for.
- 50% - Weighted average of grades obtained in two tests taken during the laboratory (20% first one, 30% second one).
- **NB!** The lab grade must be **>=5.00** (***2 decimals, no rounding***) in order to enter the written/practical examination

[L_B] 0 - ~1p Laboratory assignment bonus – optional bonus for extra laboratory work. This is added to the laboratory grade.

[S_B] 0 – ~0.5p Seminar bonus – optional bonus for your activity during the seminar. This is added to the laboratory grade.

DURING THE EXAM SESSION

[W] 20% Written exam – on the examination date.

- The written exam grade must be >=5.00 (***2 decimals, no rounding***) in order to pass the course.

[P] 40% Practical exam – on the examination date, after the written exam (with a break in between)

- The practical exam grade must be >=5.00 (***2 decimals, no rounding***) in order to pass the course.

Final grade: $0.4*\max(10, L + S_B + L_B) + 0.2*W + 0.4*P$

THE RETAKE SESSION

- During the retake session you can hand in laboratory work but are limited to a maximum laboratory grade [**L**] of **5.00**.
- You can choose to retake the written, the practical, or both examinations in case you have failed/not attended during the regular examination session.
- If you want to increase the grade obtained during the regular session, you may partake during the retake session. Your final grade will be the largest one between those obtained.

EXAMINATION DATES

- Examination dates are set in the Academic Info application
- Attending on a different date is possible only with the agreement of the course coordinator and for medical reasons proven with documentation.

Important!

- Make sure you've fulfilled your financial obligations towards the University, otherwise we are not allowed to grade you.
- Re-check the date/time of the exam beforehand (MS Teams, @General channel announcements)
- Be present on time and **have a photo ID ready**
- Have your computer ready with Python 3.x and your preferred IDE installed.
- It is possible that the problem statement during the practical exam will require drawing a board ☺. You are allowed to install the texttable Python library that provides this functionality. You can find the component here:
  - https://github.com/foutaise/texttable
  - **NB**! Read how to install it and make sure you know how to use it.
- During the practical exam, you are allowed use of the following libraries:
  - Those included in the default Python installation.
  - texttable or an equivalent library, used only to draw tables/boards.

After the exam!

- We will try to grade your work as soon as possible.
- More details will be provided on MS Teams, @General channel ☺
- Check your grades in the AcademicInfo application and report any errors ASAP!
- **Errors cannot be corrected after the examination session is complete.**

WHAT THE EXAMINATION COVERS

BASIC ELEMENTS OF THE PYTHON LANGUAGE

- Instructions: ***=, ==, if, while, for.***
- Predefined data types: ***integer, real, string, list, dictionary, tuple.***
- Functions: ***defining, parameter transmission, specification.***
- User defined types – ***classes, objects, (static) methods, attributes, inheritance.***
- Exceptions – defining ***exception types, raising, catching***.
- Lambda expressions.

ALGORITHMS – SPECIFICATION/TESTS/IMPLEMENTATION

Possibilities:

- You are given the specification – implement and test
- You are given the implementation – specify and test it
- You are given a test function – implement and specify the function it tests

1. Implement and test the function having the following specification

```
"""
    Compute the sum of even elements in the given list
    input:
        l - the list of numbers

    output:
        The sum of the even elements in the list

    Raises TypeError if parameter l is not a Python list
    Raises ValueError if the list does not contain even numbers
"""
```

2. Specify and test the following function

```
def function(n):
    d = 2
    while (d < n - 1) and n % d > 0:
        d += 1
    return d >= n - 1
```

ALGORITHM COMPLEXITY

You are given a function – analyze its complexity (best case, average case, worst case) as well as the extra-space complexity

3. Analyze the runtime complexity for the following function

```
def complexity_1(x):
    m = len(x)
    found = False
    while m >= 1:
        c = m - m / 3 * 3
        if c == 1:
            found = True
        m = m / 3
```

4. Analyze the runtime complexity for the following function:

```
def complexity_2(x):
    found = False
    n = len(x) - 1
    while n != 0 and not found:
        if x[n] == 7:
            found = True
        else:
            n = n - 1
    return found
```

5. Analyze the runtime complexity for the following function:

```
def complexity_3(n, i):
    if n > 1:
        i *= 2
        m = n // 2
        complexity_3(m, i - 2)
        complexity_3(m, i - 1)
        complexity_3(m, i + 2)
        complexity_3(m, i + 1)
    else:
        print(i)
```

6. Create an iterable data structure and a ***Product*** class with attributes ***name***, ***type*** and ***price.*** Write a generic sort function having *n*log(n)* time complexity. Create an instance of the iterable data structure and add 10 products to it. Use your sort function implementation to sort the list:

- Alphabetically by product name
- Decreasing by price

PROGRAMMING TECHNIQUES

Studied during this course:

- Divide and conquer
- Backtracking
- Greedy
- Dynamic programming

Possibilities:

- You are given a problem statement with a solution within one of the given techniques.
- Select the adequate technique for solving a given problem statement
- What we will ask:
  - Indicate the solution in detail (with or without implementation).
  - ***Backtracking:***
    - ***Search space*** representation***, consistent***() and ***solution***() functions
  - ***Divide and conquer:***
    - ***Divide*** – description, explain why you choose to do it that way
    - ***Conquer –*** describe how it works
    - ***Combine –*** describe how it works
  - ***Greedy method:***
    - Describe the ***set of candidates***
    - Describe how you make each ***selection***
    - Describe how you ***update*** the set of candidates
  - ***Dynamic programming:***
    - How was the ***principal of optimality*** observed.
    - The recurrence describing the algorithm.
    - Overlapping ***sub-problems***.
    - How you used ***memoization***.

10. Determine the longest subsequence of decreasing numbers in a list using dynamic programming.

11. Select the most appropriate technique and describe the solution for calculating the sum of the even numbers in a given list.

PRACTICAL EXAMINATION

Below you will find problem statements similar to what you can expect as part of the practical exam. The problem statement will in general follow the requirements set out between Assignment 5 and Assignment 11, will require writing specification, tests and an implementation using layered architecture.

Observations:

1. Solving the following problem statement completely should be possible for you in a timespan between 3 and 4 hours, as it has a longer list of requirements.
2. In order to pass the practical exam, you must ***implement a working program!***
3. Aspects that were part of bonus points (e.g. GUI, SQL-backend, minimax) are not required for the exam.

PROBLEM STATEMENT – STUDENT ASSIGNMENTS

Write an application to help with the management of laboratory activities for a faculty course such as FP. Students enrolled in the class can be assigned one of the **20 problem statements** (numbered from **1 to 20**) from each **laboratory**, and when they turn it in they are graded. The application will be used by the teacher and will provide the following functionalities:

- Add a student to the course. Each student has an ***id***, a ***name*** and a ***group***. You cannot have more than one student having the same ***id***, as well as students without a name or group.
- Remove a student from the course. A student can only be removed while they have not received any grades.
- Assign a laboratory problem statement to a student. You cannot assign more than one problem statements from the same laboratory to a student. If the student was already assigned a problem, the program must report the error.
- Assign a laboratory to a group. Each student in the group will be assigned a problem statement. Implement an algorithm to assign students with problem statements (e.g. each subsequent student is assigned the next problem in the list of problem statements). In case a student was already assigned a problem statement, this must not be changed!
- Grade a student for a laboratory, with an integer grade 1 to 10. Validate that the grade is valid. Grades cannot be changed!
- Best/worst students in a group. Given a group, list its students in decreasing order of their average grade.
- Students failing the class. Provide the list of all the students (regardless of group), for whom the average grade is less than 5.
- Undo/redo the last performed operation that changes the list of students or grade assignments.

NB! Data is **loaded** and **saved** to two text files: "***students.txt***", "***grades.txt***". When starting the program, make sure to have at least 10 students in the students file.

NB! Solutions without file-based repositories are acceptable, but the maximum grade for such solutions is ***7***.

NB! Your solution must adhere to the principles of layered architecture studied during the course. You will have domain, repository, controller, and UI packages/modules. The diagram below provides a guideline regarding the implementation, but it is not meant to be exhaustive.

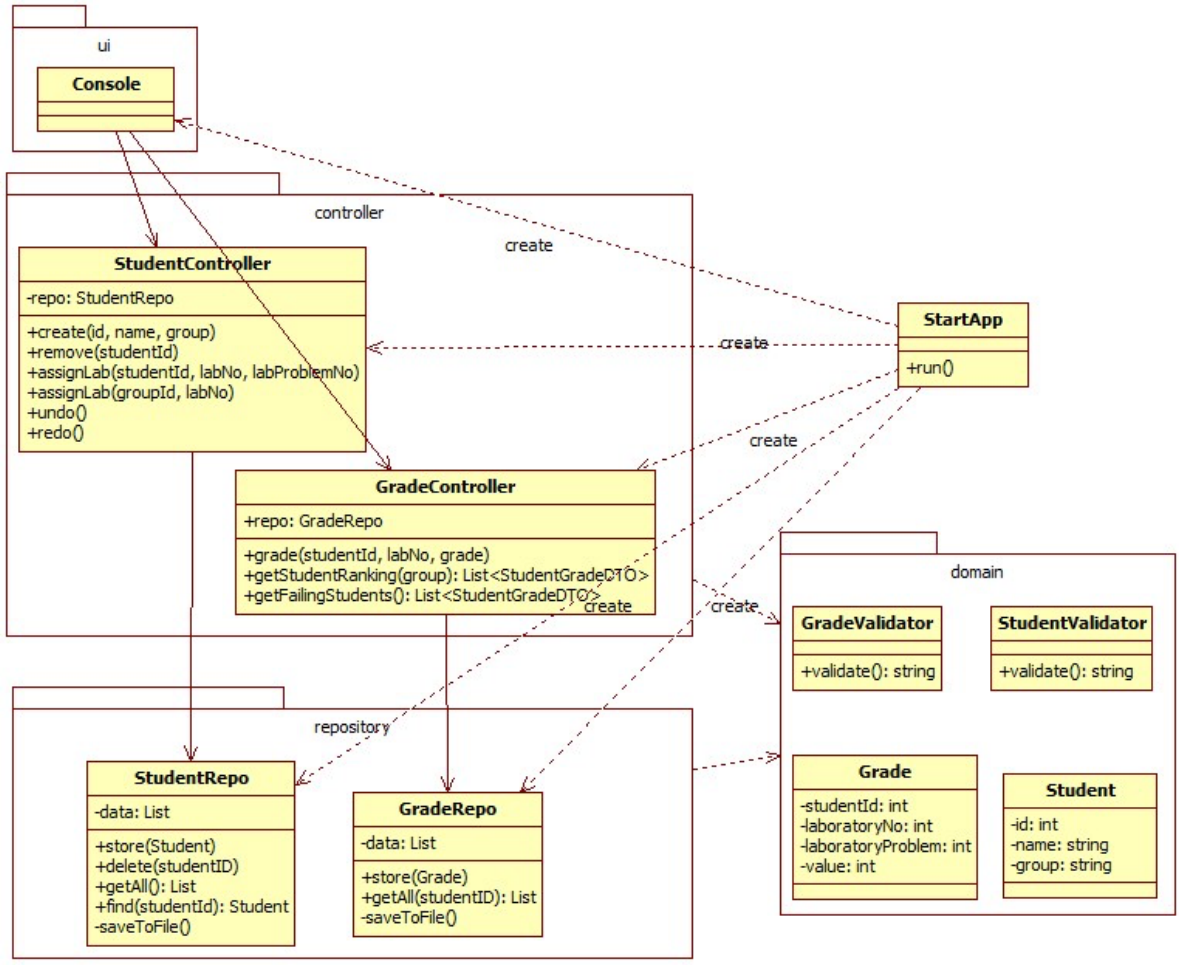

FIGURE 1 - CLASS DIAGRAM

Babeș-Bolyai University
Cluj Napoca
Faculty of Mathematics and Computer Science
Fundamentals of Programming

PROBLEM STATEMENT - BATTLESHIP[1]

Implement a console-based variation of the classical board game that you can play against the computer. To keep things simple, the game is played on a 6x6 grid as shown in the figure below. Before the game starts, both players place two battleships on the board, so that no part of the ships is outside the board and they do not overlap. Once the game starts, players take turns attacking squares, with hits tracked on the player and targeting boards. The game ends when one player hits all the squares occupied by the enemy ships. Program functionality is broken down as follows:

1. Place your battleships on the board using the following command: **ship <$C_1L_1C_2L_2C_3L_3$>**
   [E.g. commands **ship C3D3E3** and **ship A0A1A2** gives the ship position in the figure below]
   In case an invalid square is provided, a part of the ship falls outside the grid, or ships overlap (have at least one common square) the program will provide an error message and will not place the ship **[2p]**.

2. You can repeat the **ship** command as many times as you wish, until you are pleased with your ships' position on the grid. If you already placed two ships on the board, entering a valid command will replace the ship that was added first with the current one. **[1p]**

3. Each time the command results in valid placement of a ship on the player's board, the player board will be displayed as illustrated on the left hand side of the figure below. **[1p]**.

4. Start the game using the following command: **start**
   The start command can only be provided once two battleships have been placed on the player board. Once the game starts, the program will <u>randomly</u> place two battleships on the computer's board, using the same rules. **[2p]**

```
  A B C D E F        A B C D E F
0 + . . . . .      0 . . . . . .
1 + . . . . .      1 . . . . . .
2 + . . . . .      2 . . . . . .
3 . . + + + .      3 . . . . . .
4 . . . . . .      4 . . . . . .
5 . . . . . .      5 . . . . . .
  Player board       Targeting board
```

5. Play the game **[2p]**. The player and computer will attack squares in turn, with the player having the first attack. Attacks are made using the command: **attack <*square*>**. [E.g. **attack E4**]. When all the squares containing a player's ships are hit, the player loses the game. The program will provide a message in this regard.

   After each attack, the player and targeting boards are updated. Given the player and targeting boards above, the following series of attacks will result in the following possible board configuration:

```
attack E4                A B C D E F        A B C D E F
Player misses!         0 + . . . . .      0 . . . . . .
computer attack C2     1 + . . . . .      1 . . . . . .
Computer misses!       2 + . o . . .      2 . . X . . .
attack C3              3 . . + X + .      3 . . X . . .
Player hits!           4 . . . . . .      4 . . . . o .
computer attack D3     5 . . . . . .      5 . . . . . .
Computer hits!           Player board       Targeting board
attack C2
Player hits!
```

6. Cheats. We want to know where the computer's battleships are. Using the **cheat** command, the program will reveal the placement of the computer's ships (it is up to you to decide how) **[1p]**.

Non-functional requirements:

- Implement a layered architecture solution.
- Provide specification and tests for the methods involved in functionalities 1 and 2. In case no specifications or tests are provided, these functionalities are graded at 50% value.

[1] Actual subject during the 2016/2017 retake exam ☺