# Exam Preparation

*Image: The Exam, Mr. Bean, youtube.com*

# Organizational Matters

- When:

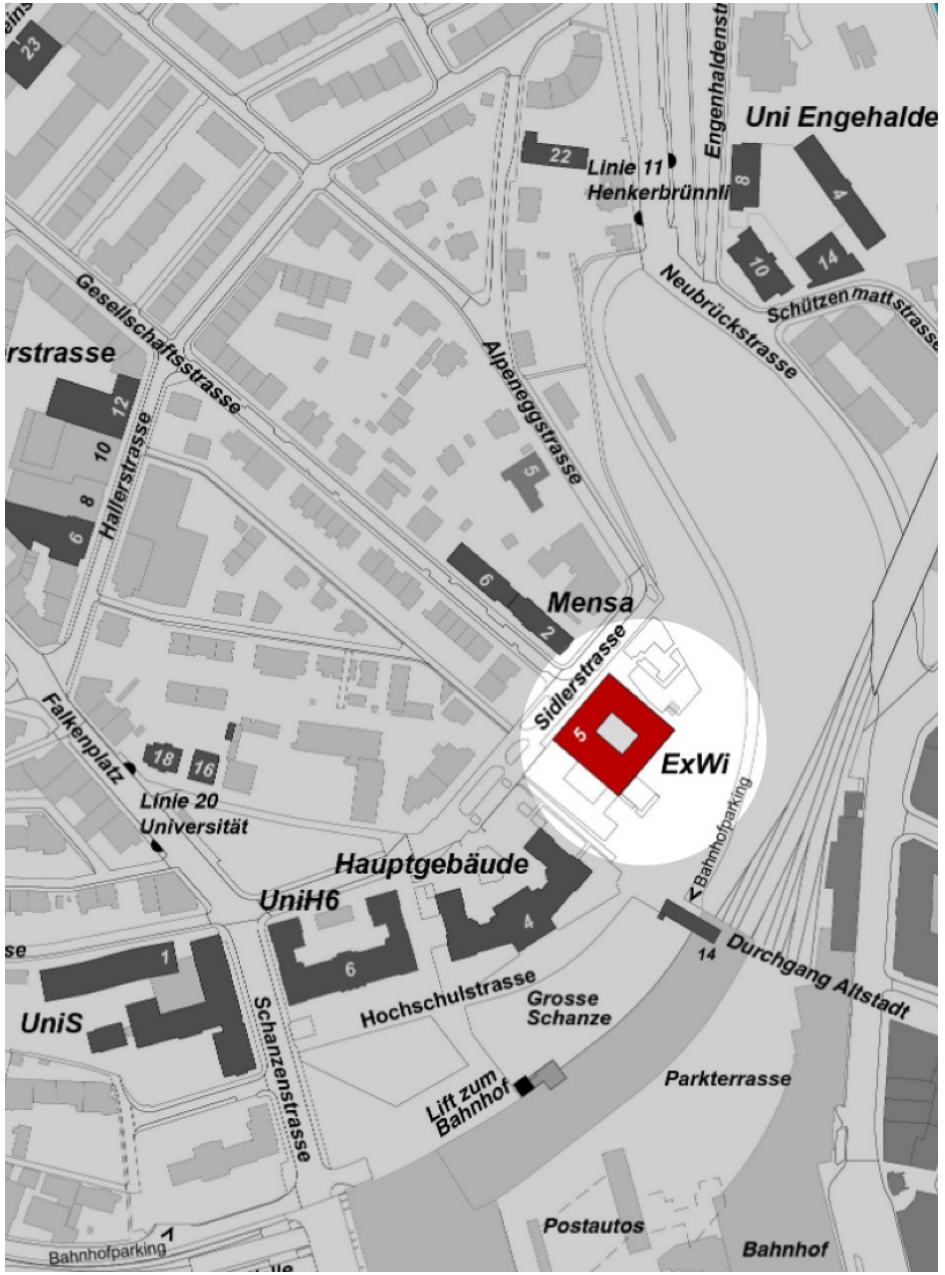
~~10:00 – 12:00~~

***Does not apply for the repetition exam.***
*See details for repetition exam on*
*https://seg.inf.unibe.ch/teaching/current/p2/*

- Where:

~~ExWi Building,~~
~~Lecture rooms A0 and/or B5~~
~~(we will be on site)~~

- Duration:
  - 105 Minutes (10:15 – 12:00)



# Rules & Preliminary Notes (1/2)

Programming 2 Exam

University of Bern

Software Engineering Group (SEG)

Prof. Dr. Timo Kehrer, Roman Bögli

Demo

Full Name: _____

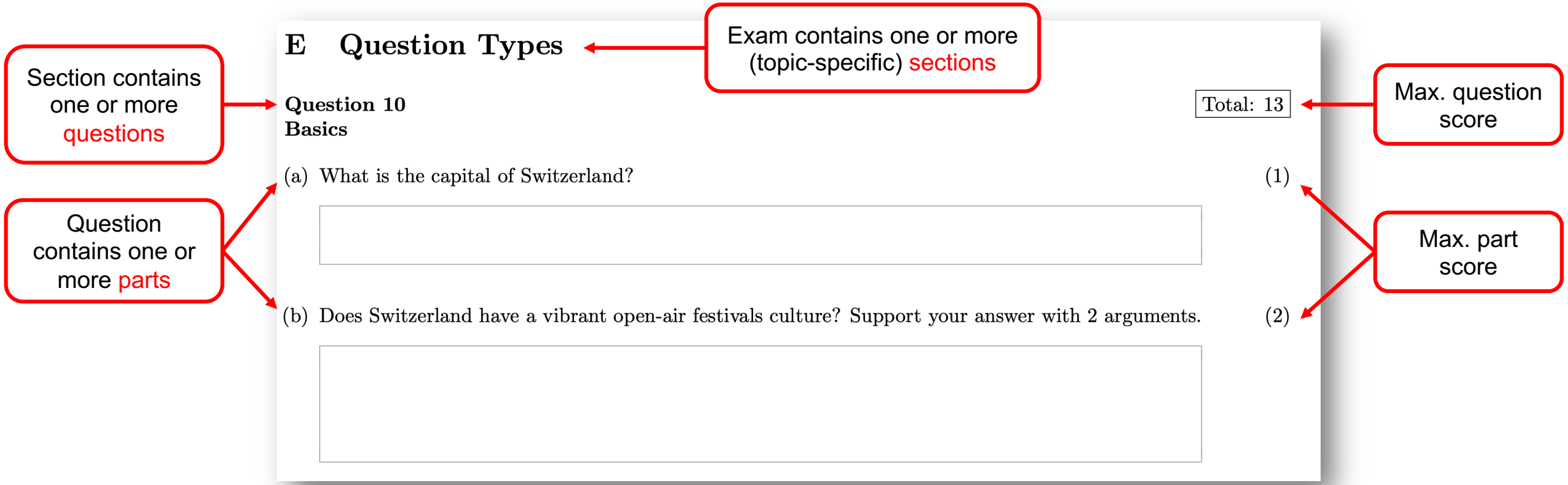
Matriculation No.: _____

Exam Preparation

# Rules & Preliminary Notes (2/2)

- Please have your student ID card ready for inspection.
- You must use a non-erasable pen. Unmistakably cross out anything you do not wish to be graded.
- No aids are permitted except a standard English dictionary.
- Answers should be as brief and concise as possible, unless the question explicitly demands detailed discussion or elaboration. Answers can be written in English or German.
- When you are unsure if you understood the question correctly, briefly state your interpretation.
- Questions asking for $n$ arguments or examples are subject to point deductions for wrong statements when more than $n$ arguments or examples are provided. Minimum score remains zero.
- True-False-Choice questions have decisive statements. Incorrect answers result in point deductions, so tick boxes only when confident. The minimum score is zero.
- When writing Java code, focus on overall syntax and structure. For example, a missing comma or parenthesis will not lead to point deductions.
- The provided answer space is generous and not indicative of required answer length or format.
- There is extra space from page 18 onwards, in case you need more space for your answers or other remarks. Please ensure clear referencing when using it. Additional paper can be provided upon request.

# Exam Structure



# Question Types (1/7)

- Open questions

(a) What is the capital of Switzerland? (1)

(1) Bern

Exam Preparation

# Question Types (2/7)

*__Remember:__ Questions asking for $n$ arguments or examples are subject to __point deductions__ for wrong statements when more than $n$ arguments or examples are provided.*

- Open questions with arguments

(b) Does Switzerland have a vibrant open-air festivals culture? Support your answer with 2 arguments. (2)

(b) Does Switzerland have a vibrant open-air festivals culture? Support your answer with 2 arguments. (2)

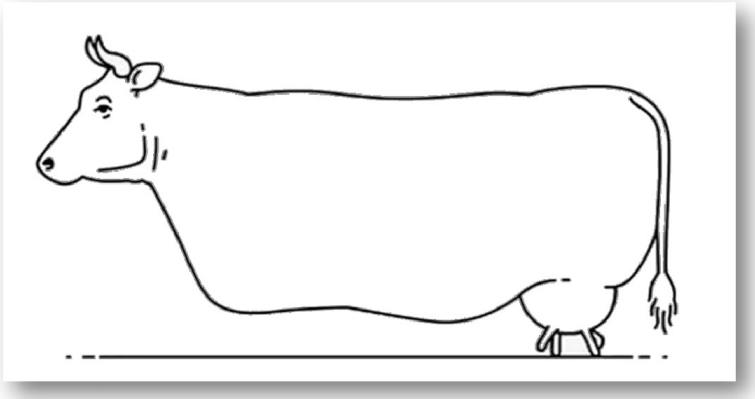
- (1) Yes, it does.
- (1) for any pair of two:
  - Switzerland hosts a wide array of festivals annually, attracting international audiences.
  - High purchasing power of Swiss attendees supports the thriving festival scene.
  - Breathtaking landscapes of the Alps and lakes provide ideal settings for outdoor festivals.
  - . . . (any other meaningful argument)
- (-1) for any unreasonable argument (e.g. because of the sea, different time zones, low gravity, . . . )

Taking (wild) guesses can lead to point deductions.

# Question Types (3/7)

- Open questions with supporting material (e.g. Image)

(e) What does the figure below analogically have to do with software engineering? Support your answer with 2 arguments. (2)



(1) for any of the following, but maximal 2. No point deduction for wrong answers here.

- Single Responsibility Principle (cow's udder has too many responsibilities)
- Separation of Concerns (different concerns should be managed separately and distinctly)
- Non-Agile Development (highly detailed but still incomplete product)
- . . . (any other reasonable answer)

Subjective / opinion-based questions are subject to more relaxed point allocation

# Question Types (4/7)

- Open questions with supporting material (e.g. Code)

(f) What is the output of the following Java `main` method? (2)

```
1  public class Example {
2      public static void main(String[] args) {
3          printNumbers();
4      }
5
6      public static void printNumbers() {
7          for (int i = 1; i <= 3; i++) {
8              System.out.println(i);
9          }
10     }
11 }
```

- (1) The output is: `1 2 3`

Exam Preparation

# Question Types (5/7)

*__Remember:__ When writing Java code, focus on overall syntax and structure. For example, a missing comma or parenthesis will not lead to point deductions.*

- Questions asking to write/complete Java code

(g) Given the Java code skeleton below, implement a version of the `printNumbers` method from above that does not use a `for-loop`. (2)

```
1  public static void printNumbersAlternative() {
2      // todo
3
4
5
6
7
8
9
10
11
12 }
```

- (1) compiles successfully
- (1) generated output: `1 2 3`
- (+1) required less than 3 lines of code
- (-1) disregards the provided code skeleton

Extraordinary answers are sometimes rewarded with bonus points (given below max score)

```
1  public static void printNumbersAlternative_1() {
2      int i = 1;
3      while (i <= 3) {
4          System.out.println(i);
5          i++;
6      }
7  }
8
9  import java.util.stream.IntStream;
10 public static void printNumbersAlternative_2() {
11     IntStream.rangeClosed(1, 3).forEach(System.out::println);
12 }
```

*__Remember:__ The provided answer space is generous and not indicative of required answer length or format.*

Exam Preparation

# Question Types (6/7)

*__Remember:__ True-False-Choice questions have decisive statements. Incorrect answers result in point deductions, so tick boxes only when confident.*

- True-False-Choice questions

(c) Indicate whether each of the following statements is true or false: (2)

| True | False | |
|---|---|---|
| ○ | ✓ | Zurich is the capital of Switzerland. |
| ✓ | ○ | Switzerland has four official languages. |
| ○ | ✓ | The traditional food *Raclette* was adapted from Germany. |
| ✓ | ○ | Computers become increasingly important in today's societies. |

- (+0.5) for correct selection
- (-0.5) for incorrect selection
- no points for unselected

# Question Types (7/7)

*__Remember:__ True-False-Choice questions have decisive statements. Incorrect answers result in point deductions, so tick boxes only when confident.*

- True-False-Choice questions with given context

(d) Indicate whether each of the following statements is true or false: Stereotypically, Swiss people... (2)

| True | False | |
|---|---|---|
| ☑ | ☐ | ...are punctual. |
| ☐ | ☑ | ...prefer to speak High German over Swiss German. |
| ☐ | ☑ | ...are poor. |
| ☑ | ☐ | ...love chocolate and cheese. |

- (+0.5) for correct selection
- (-0.5) for incorrect selection
- no points for unselected

# Extra Space

- last section

## F Extra Space

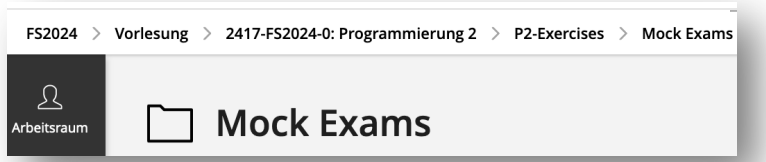
The subsequent blank pages may be utilized for further elaboration on previously provided answers or for appending additional remarks as desired by the student. Ensure clear referencing, e.g. "*continued answer to Question 2, part (d) from page 8*".

# How to Prepare

- Lecture slides
- Exercise slides

primarily

- Your exercise feedbacks
- Mock Exams
  - Available on ILIAS
  - Please note:
    - this year's exam is completely new
    - topics & question types in mock exam may not representative




secondary

Exam Preparation

# Comment to Mock Exams

## 3 Design patterns, 8 p

Consider the following code example.

```
1  // LegacyRectangle class
2  public class LegacyRectangle {
3      private int width;
4      private int height;
5
6      public LegacyRectangle(int width, int height) {
7          this.width = width;
8          this.height = height;
9      }
10
11     public int calculateArea() {
12         return width * height;
13     }
14 }
15
16 // Shape interface
17 public interface Shape {
18     int getArea();
19 }
20
21 // Adapter class
22 public class RectangleAdapter implements Shape {
23     private LegacyRectangle rectangle;
24
25     public RectangleAdapter(LegacyRectangle rectangle) {
26         this.rectangle = rectangle;
27     }
28
29     @Override
30     public int getArea() {
31         return rectangle.calculateArea();
32     }
33 }
34
35 // Client code
36 public class Client {
37     public static void main(String[] args) {
38         LegacyRectangle legacyRectangle = new LegacyRectangle(5, 10);
39         Shape rectangleShape = new RectangleAdapter(legacyRectangle);
40         int area = rectangleShape.getArea(); // returns 50
41         System.out.println("Area of rectangle: " + area);
42     }
43 }
```

(a) Describe the Adapter design pattern and its advantages and disadvantages.

(b) Provide an UML class diagram for the classes involved in the example and clearly point out the components of the pattern.

(c) How does the Adapter pattern differ from the Proxy, Decorator (other relevant) patterns? Explain your answer in detail.

### 3.1 (Answer 3a)

The Adapter design pattern is a structural design pattern that allows objects with inco
together. It acts as a bridge between two incompatible interfaces, converting the interfa
interface that clients expect.

**The main components of the Adapter pattern are:**

1. Target: This defines the interface that the client expects to interact with.
2. Adaptee: This is the existing class or interface that needs to be adapted to the T
3. Adapter: This is the class that implements the Target interface and holds an i
translates the requests from the client into calls to the appropriate methods of th

**Advantages of the Adapter pattern:**

1. Increased reusability: Adapters allow the reuse of existing classes or componer
interfaces, without modifying their source code. This promotes code reusability ar
2. Seamless integration: Adapters enable the integration of different components or
interfaces, allowing them to work together smoothly.
3. Encapsulation: Adapters encapsulate the logic required to convert the interface of
interface. This keeps the complexity hidden from the client and promotes a clean

**Disadvantages of the Adapter pattern:**

1. Increased complexity: Introducing adapters can add an additional layer of complex
when dealing with multiple adapters. This can make the code harder to understa
2. Performance overhead: The Adapter pattern may introduce a slight performan
ditional indirection and translation of method calls. However, this overhead is u
system is highly performance-sensitive.
3. Dependency on specific implementations: Adapters are often tightly coupled to
the Adaptee, which can limit flexibility and make it harder to switch between dif

Overall, the Adapter pattern provides a flexible solution for integrating incompatil
code reuse. However, its usage should be carefully considered to ensure it doesn't introd
or dependencies.

missing predefined point allocation system

### 3.3 (Answer 3c)

While the Adapter, Proxy, and Decorator patterns all fall under the category of structural design patterns and share similarities, they have distinct purposes and differ in their implementation and usage.

**The key differences between the Adapter and Proxy patterns are:**

**Purpose**

1. Adapter: The Adapter pattern focuses on adapting the interface of one class to match the expected interface of another class.
2. Proxy: The Proxy pattern focuses on controlling access to an object and adding additional behavior around it.

**Interface**

1. Adapter: The Adapter pattern typically introduces a new interface that is compatible with the client's expectations.
2. Proxy: The Proxy pattern maintains the same interface as the original object, allowing clients to interact with it seamlessly.

**Adaptation vs. Indirection**

1. Adapter: The Adapter pattern typically introduces a new interface that is compatible with the client's expectations.
2. Proxy: The Proxy pattern introduces an additional layer of indirection and control over the object, enabling us to manage access to it or add extra behavior without modifying the original object's code.

**The key differences between the Adapter and Decorator patterns are:**

**Purpose**

1. Adapter: The Adapter pattern focuses on adapting the interface of one class to another, allowing them to work together.
2. Decorator: The Decorator pattern focuses on enhancing the behavior of an object by adding new responsibilities or functionalities dynamically.

**Object modification**

1. Adapter: The Adapter pattern does not modify the structure or behavior of the adapted class. It only provides a new interface to interact with it.
2. Decorator: The Decorator pattern modifies the behavior of the decorated object by adding new functionalities or responsibilities.

**Relationship with the original object:**

1. Adapter: The Adapter pattern provides a different interface to interact with the adapted object, making it compatible with the client's expectations.
2. Decorator: The Decorator pattern wraps the original object and adds new functionalities while maintaining the same interface.

In summary, the Adapter pattern is used to adapt the interface of one class to match another, the Proxy pattern provides an additional level of indirection and control over an object, and the Decorator pattern dynamically enhances the behavior of an object by adding new functionalities. Each pattern addresses different concerns and has distinct use cases.

# Out of Scope

- There will be no questions about:
  - Git (or VCS in general)
  - Agile project methodologies (e.g. Scrum, Kanban, etc.)
  - Topics from optional exercises (this year: Smalltalk, AspectJ)

- The reader *"Applying Design by Contract (Meyer 1992)"* is nice to read but not particularly exam relevant.
  - see on ILIAS