Methods in Software Engineering — Practice Exam

Prof. Dr. Gidon Ernst • Dr. Philipp Wendler • LMU Munich Summer Term 2023

Before the exam

- turn off your mobile phone and smartwatch now and store them away,
  if we catch you with a turned-on phone or similar we must regard it as cheating
- put any bags and jackets in the row in front of you

- write down your name and matriculation number

  Name:

  Matriculation Number:

When the exam time starts

- check that all 13 pages are included (the last page is spare)

Rules

- blue or black pen (not erasable, no pencil, no green/red),
- ok: mask, drinks, snack, ruler
- not ok: own paper, *any* other material, including:
  notes, books, calculators, dictionary

Time 90 minutes

Language

- you can answer both in English and/or German
- please ask if words or sentences are unclear

Grading ☐ please *do not* grade this exam ("entwerten")

Exercise	1	2	3	4	5	6	Σ
Points	of 10	of 17	of 8	of 16	of 14	of 15	of 80

Remarks

- the exam is subject to copyright and may not be distributed outside of this lecture
- this practice exam corresponds exactly to exam #1 held on July 26, 2023

1 Software Development and Maintenance (10 points)

a) Name the **three key tools** for modern development, as discussed in the lecture, and which are typically offered by code hosting platforms. (1.5 points)

b) Name **three features** of `git`, you can include its basic functionality. (1.5 points)

c) Briefly describe **two rules** (or good conventions) for commits. (2 points)

d) Name **one benefit** of using container systems (`docker`, `podman`) during testing. (1 point)

e) Name **two aspects** or activities that are relevant for software maintenance. For each, **briefly describe** why these aspects can be challenging and also **propose an idea** how to deal with these challenge. (4 points)

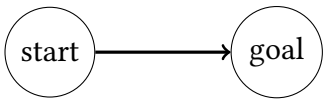
2 Domain-Specific Languages (DSLs) (17 points)

You are given the following grammar for a domain-specific language to describe simple graphic illustrations composed of nodes with a textual label and arrows between these nodes. Nodes are positioned at given `(x,y)`-coordinates, the label is simply a string , and arrows go from one node to another, identified by the respective labels.

```
number      ::= ( "0" ... "9" )+
label       ::= ( "a" ... "z" )+
coordinate  ::= "(" number "," number ")"
element     ::= node | arrow
node        ::= "node"  label coordinate
arrow       ::= "arrow" label label
description ::= element+
```

0) Here is an example program in this language, which describes the picture shown on the right:

```
node  start (0,0)
node  goal  (3,0)
arrow start goal
```



a) Does the grammar allow for an empty picture? (1 point)

☐ yes ☐ no

b) Define a class hierarchy as a UML diagram or in code to represent the abstract syntax of the grammar. Include just classes, attributes, and associations with multiplicities where appropriate, but do not include methods, You can assume that numbers are represented by integer values and labels by string values. (8 points)

c) Suppose you have the following class available to implement your interpreter

```
class Picture {
    drawNode(str: String, at: Coordinate)
    drawArrow(from: Coordinate, to: Coordinate)
}
```

How can you use this to "execute" programs in the given language? Your explanations can be in natural language. Clarify the aspects listed below. If needed, refer to your answer from **b)** as well as the methods of class `Picture`.

1. what is the *state* that you need to keep track of (1 point)
2. steps to draw an entire `description` (1 point)
3. steps to draw a `node` (2 points)
4. steps to draw an `arrow` (2 points)

Assume that the methods of class `Picture` take care of any layout concerns.[1]

d) According to *your* answer in **c)**, is the order in which `elements` are listed as part of a `description` important? (2 points)

☐ yes ☐ no

because

[1]For example, assume that coordinate `at` is the center of the node, similarly, `from` and `to` refer to centers of nodes and `drawArrow` positions the arrow correctly by reducing its length by the size of the circles around the nodes).

3 Refinement Types, Propositions as Types (8 points)

To express invariants over types, we have discussed the notion of "refinement" types of the form

```
t where x:  p(x),
```

where `t` is the base type and `p` is a predicate over variable `x` of that type.

a) Define a refinement type for numbers from `2` up and to including `7` in that way. (1 point)

b) *Implement* this type in a programming language of your choice (e.g. Java, Python) with a new class and a runtime check. (3 points)

Inductive data type can represent logical formulas and functions that implement these types show that the logical formula is true.

We have the following types to represent logical conjunction $A \wedge B$ and disjunction $A \vee B$, as well as function types $A \to B$ that correspond to implication.

$$\texttt{data Pair}\ A\ B = (\mathit{fst} : A, \mathit{snd} : B)$$
$$\texttt{data Either}\ A\ B = \texttt{left}(a : A) \mid \texttt{right}(b : B)$$

Complete the missing parts below (type, function definition, formula):

c) formula $\big((A \vee B) \wedge (A \implies B)\big) \implies B$

type (2 points)

definition (2 points)

4 Component Design (16 points)

Consider a delivery service which picks up food from restaurants and delivers it to people's homes. We are concerned with the design of a web-service that is accessible to third-party web-sites or apps (such as an app for customers or an app for restaurants).

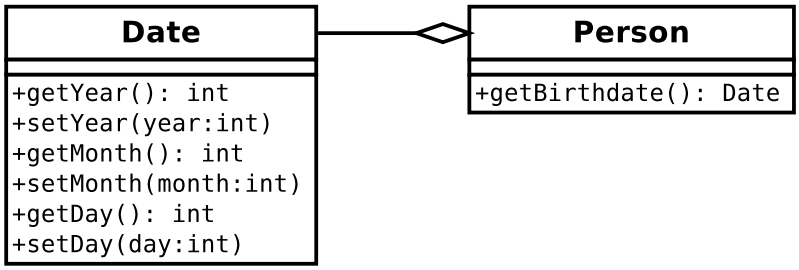
Specify the interface of the operation described below. Describe each **parameter** and **return value**. Explain how the underlined concepts below are represented in terms of their **type**, the respective **valid values**, and give an **example** for each.

The description can be informal (not tied to a particlar programming language) but should be precise (include all relevant details).

a) `GetMenuForRestaurant`: Returns the menu for a given restaurant. (5 points)

b) `AddCustomerRating`: Submits a rating and a comment by the customer about a restaurant. In order to avoid fake ratings and allow only ratings for actual orders, the customer needs to reference an order (by giving its order id and the total price), and the rating will be for the restaurant of that order. (5 points)

c) Consider the following class `Person`, which uses Java's `Date` class for storing the birthdate:



i. According to the classification of classes/objects in the lecture

- `Date` represents ☐ data ☐ an algorithm ☐ a component (1 point)
- `Person` represents ☐ data ☐ an algorithm ☐ a component (1 point)

ii. `Date` has a widely criticized design flaw, which causes problems for the class `Person`.
Note: We have discussed a *similar* example with an analogous problem in the lecture.

Briefly explain (4 points)

- What is this flaw in class `Date` and how does it affect `Person`?
- How can class `Person` work around this problem?
- How should class `Date` have been defined to avoid this problem?

5 Liskov's Substitution Principle (14 points)

Consider two Java classes, **A** and **B** that both implement a common interface `IntList`, which provides methods for storing a sequence of **`int`** values.

```
class A implements IntList {
  int[] array = new int[10];
  int cursor = -1;

  void add(int value) {
    cursor++;
    array[cursor] = value;
  }

  int get(int i) {
    checkArgument(i >= 0);
    checkArgument(i < size());
    return array[i];
  }

  int size() {
    return cursor;
  }
}
```

```
class B implements IntList {
  int[] array = new int[10];
  int cursor = 9;

  void add(int value) {
    array[cursor] = value;
    cursor--;
  }

  int get(int i) {
    checkArgument(i >= 0);
    checkArgument(i < size());
    return array[cursor + i + 1];
  }

  int size() {
    return array.length - cursor - 1;
  }
}
```

There are multiple distinct violations of Liskov's substitution principle in the code above. Provide *histories*, which uncover these. Write *one event per line*. Note: None of these violations would need changes to more than one part of the respective class in order to be fixed.

Your histories should end in a mismatch in the result of **A** and **B**, where results are one of: return values of an operation, or nothing (—) in case of **`void`**, or an exception. You do not need to include a call to a constructor as the first event.

a) Provide a history, in which the problem is in **A**. (3 points)

step	operation name	parameter value	result for **A**	result for **B**
1.				
2.				
3.				
⋮				

b) How would you fix this problem in class **A**? (1 point)

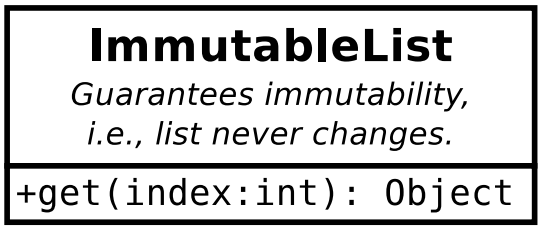
c) Provide a history, in which the problem is in class **B**. (3 points)

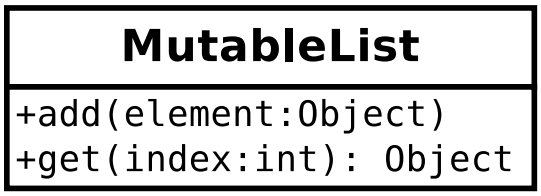
step	operation name	parameter value	result for **A**	result for **B**
1.				
2.				
3.				
⋮				

d) How would you fix this problem in class **B**? (2 points)

e) Explain briefly how Liskov's Substitution Principle can be useful for *code refactoring*. (1 point)

f) Consider the following two types:





i. According to Liskov's Substitution Principle, would it be valid to let **MutableList** inherit from **ImmutableList**? (2 points)

☐ yes ☐ no

because

ii. According to Liskov's Substitution Principle, would it be valid to let **ImmutableList** inherit from **MutableList**? (2 points)

☐ yes ☐ no

because

6 Component and System Invariants (15 points)

a) Consider the following class, given equivalently in Java and in Python:

```java
class BiMap {
  Map forward = new HashMap();
  Map backward = new HashMap();

  void put(Object a, Object b) {
    if (forward.containsKey(a))
        throw new RuntimeException();
    if (backward.containsKey(b))
        throw new RuntimeException();
    forward.put(a, b);
    backward.put(b, a);
  }

  Object getForward(Object a) {
    return forward.get(a);
  }

  Object getBackward(Object b) {
    return backward.get(b);
  }
}
```

```python
class BiMap:
  forward = dict()
  backward = dict()

  def put(self, a, b):
    assert a not in forward

    assert b not in backward

    self.forward[a] = b
    self.backward[b] = a


  def get_forward(self, a):
    return self.forward[a]


  def get_backward(self, b):
    return self.backward[b]
```

Recall for yourself the definition of a **class invariant**.

Select exactly the statements that correspond to **class invariants** of **BiMap**. (10 points)

Each correctly check box gives +1 point, each wrongly checked box gives -1 point.

- ☐ The sizes of `forward` and `backward` are the same.
- ☐ The runtime type of all objects stored in `forward` must be the same.
- ☐ The runtime type of all objects stored in `backward` must be the same.
- ☐ The keys of `forward` are exactly the values of `backward`.
- ☐ The values of `forward` are exactly the keys of `backward`.
- ☐ Parameter `a` of method `put` must not be in the mapping already.
- ☐ If an object a is mapped to b in `forward`, then `backward` maps b to a.
- ☐ For every object a holds: a cannot be in the set of keys of `forward` and the set of keys of `backward` at the same time.
- ☐ An object a must not be mapped to itself in `forward`.
- ☐ The set of keys in `forward` is exactly the same as the set of objects that were previously passed as parameter `a` to method `put`.

b) Name **one advantage** of explicitly checking inside the methods of a class whether the class invariants hold at runtime, e.g. using assertions. (1 point)

c) Is it more useful to add such runtime checks for class invariants at the beginning or the end of the methods of a class? (2 points)

It is more useful to check ☐ at the beginning ☐ at the end of methods

because

d) Is it possible to check at runtime whether the *last* property from **a)** is an invariant?

- The set of keys in `forward` is exactly the same as the set of objects that were previously passed as parameter a to method `put`.

Select and answer one of the two possibilities: (2 points)

☐ Yes, with the following code: ☐ No, because:

Extra Page

If you use this page, please

- strike through those parts and solution attempts that should not be graded
- place a short note on the exercise sheet: “see extra page” or similar
- more paper is available on request, *please always return all sheets*