Methods in Software Engineering — Practice Exam

Prof. Dr. Gidon Ernst • Daniel Baier • LMU Munich

Summer Term 2024

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 14 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	Σ
Points	of 14	of 20	of 16	of 14	of 16	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 August 1, 2024

1 Development, Maintenance, Theory (14 points)

a) The lecture discussed the problems associated with object-oriented programming (OOP). Do you agree with the following statement? Add a brief explanation as to why or why not.

"OOP is outdated and should not be used anymore!" (1 point)

b) Name **one advantage** of Domain-Specific Languages (DSLs) over general-purpose programming languages. Given **one example** for such an advantage, i.e., name a DSL and explain how it achieves this advantage. (2 points)

c) Name **two features** of git **besides** branches, forks, commits and merge. (1 point)

d) Would you rate that the following commit message as good? Briefly explain why or why not.

"Fixed bug in production. Also refactor other code and some comments" (1 point)

e) Maintaining software is hard. Name **two challenges** typically associated with software maintenance and **for both a suggestion** to address these. (2 points)

f) Briefly describe the **criteria** when two classes satisfy Liskov and Wing's substitution principle. Name **scenario** in which this principle could be a useful guideline in practice. (2 points)

g) Name **one benefit** and **one potential challenge** of using data/class invariants. (2 points)

h) Implement a **refinement type** in an actual and reasonably well-known programming language of your choice. The type should represent strings that begin with the character `<` and end with the character `>`, i.e., which are delimited by angular brackets. (3 points)

This value should be accepted: `<hello world>`

This value should be rejected: `whoot`

actual solution

comments on the solution

2 Domain-Specific Languages (20 points)

You are given the following grammar for a domain-specific language to describe simple textual documents which can have placeholders in them, similarly to a template system used for example in web-development.

```
letter      ::= <any character except "$">
identifier  ::= ( "a" ... "z" )+
text        ::= letter+
placeholder ::= "${" identifier "}"
segment     ::= text | placeholder
paragraph   ::= segment* "\n\n"
document    ::= paragraph*
```

attributes

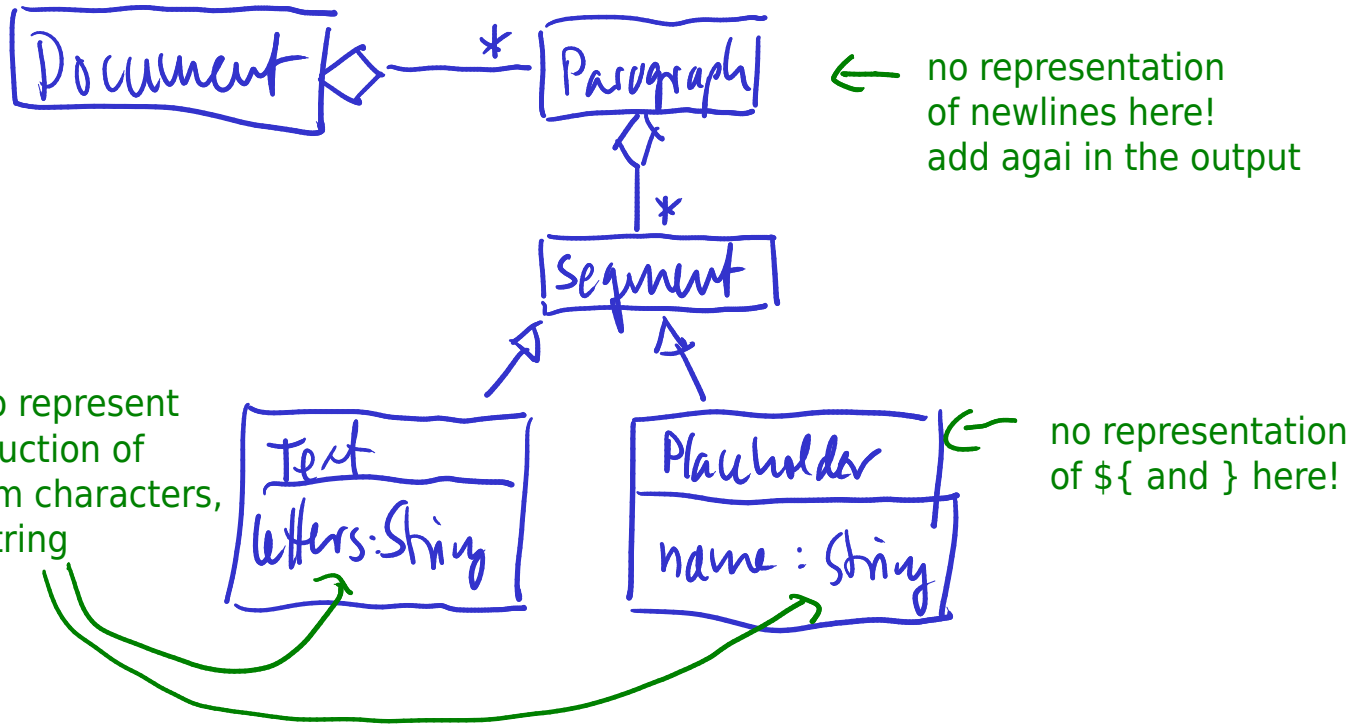
"alternatives":

"many": List

Here is an example document in this DSL, applied to the use-case of E-Mails with placeholders, represented by identifiers delimited by `${` and `}` that are later to be filled from a database, and `\n` represents a newline character, so that paragraphs are deliminted by a fully empty line.

```
Dear ${firstname} ${lastname},

thank you for your registration to ${event}$!
We are very much looking forward to see you there.

Kind regards,
  ${organizer}
```

a) 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, Represent sequences of letters as well as identifiers as standard strings. (8 points)



In the next part of the exercise we are interested in rendering/formatting a document. This involves two main features

- Placeholders are replaced by concrete values given as input to the rendering method.
- The resulting text for each paragraph, which in turn consists of the concatenation of many segments, is wrapped into lines of a maximal length at word breaks, this length is also given as an input to the rendering method.

As an example, for a specified *text width* of 60 characters and giving `firstname` as "Barbara" and `lastname` as "Liskov", `event` as "Symposium on Principles of Programming Languages" and `organizer` as "Association for Computing Machinery", rendering the example from above produces:

```
Dear Barbara Liskov,

thank you for your registration to Symposium on Principles
of Programming Languages! We are very much looking forward
to see you there.

Kind regards,
  Association for Computing Machinery
```

b) Fill in *suitable types* of the parameters as well as the return value in the signature of the method `render`, as shown below. The type of `document` should match part **a)**. (4 points)

```
def render( document: Document (from a),
            textwidth: Int (e.g. 60),
            data: Map[String, String] *
          ): String or List[String]
```
(return type)

* the keys correspond to names of supplied placeholders
and the values correspond to the instantiation of these placeholders

e.g. Map("firstname" -> "Barbara", "lastname" -> "Liskov",
"organizer" -> "Association for Computing Machinery")

c) Describe how method `render` should be implemented. (8 points)

The description can be informal but precise. You may assume that a suitable method `wrap` is already implemented, which performs the line-wrapping correctly.

Take care to explain, clarify, or highlight in particular the following aspects:

- how the data structures from your answer in **a)** are used
- how the inputs to `render` from your answer in **b)** are used
- how placeholders are filled in
- how `segments` are assembled into a `paragraph`
- how `paragraphs` are assembled into a the final output of `render`
- in which step the helper method `wrap` is used

these are hints!
this does not mean you should just answer each point individually

for each paragraph in the document given as input to render:

how paragraphs are assembled

  for each segment in the paragraph:
    format the segment (below)
    append its result to current part

  wrap current part with given textwidth
      with the helper method

  output the result of wrap
  output two newlines
end

how segments are assembled into a paragraph

where wrap is used
(could also wrap final result instead each paragraph)

formatting a segment:
  if the segment is text:
    just return the segment's letters

  if the segment is a placeholder
    look up the placeholder's name in the given map, data
    return the corresponding value

how placeholders are filled in

3 Component Design (16 points)

Consider a international delivery service for parcels, similar to DHL, FedEx, UPS etc. 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 parcel tracking).

Specify the interface of the operations 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) `GetPriceListForTargetCountry`: Returns the price-list for different parcel sizes or weights that can be send for a given target country. (5 points)

input: target country a) string

b) enum de|us|uk|fr|...

type

also: a) should be a name of a proper country (lowercase is ok)

examples "Spain" "Spanien"

output price list type: list of prices (where price is a record with size/weight indication + amount float in EUR

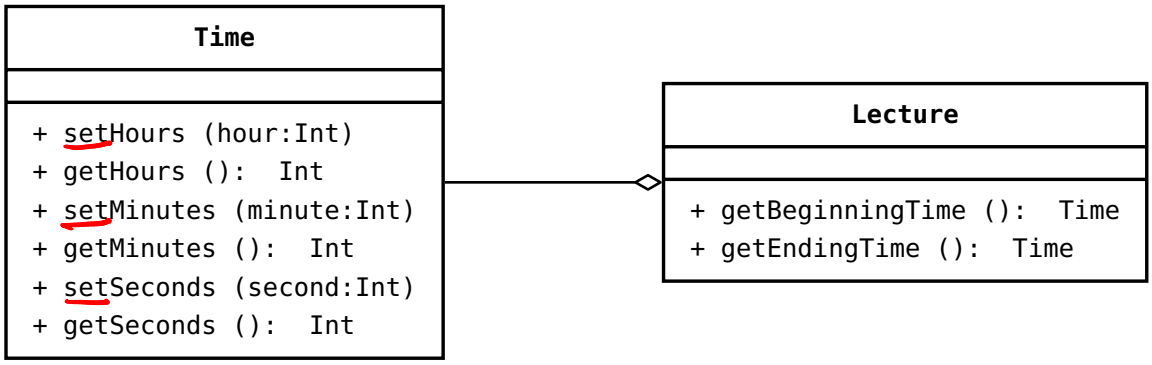
[{weight: 1000 g, price: 120, currency: "EUR"}, ...]

good advice:
use decimal types for currency,
not float types

b) `BookShippingParcel`: Adds parcel to be delivered to the system. A booking to ship a parcel consists of the parcel size and a target address and a sender address. In order to allow proper processing for actual parcels, addresses are to be submitted as, zip-code, street, and country separately as structured data. Payment is not part of this operation. (5 points)

(similar)

c) Consider the following class `Lecture`, which uses a `Time` class for storing the beginning and ending times of a lecture:



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

- `Time` represents ☑ data ☐ an algorithm ☐ a component (1 point)
- `Lecture` represents ☑ data ☐ an algorithm ☐ a component (1 point)

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

Briefly explain (4 points)

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

Flaw: Time is mutable - if Lecture shares its internal objects,
some other code could mutate these, implictly changing lecture times

Workaround: Lecture can return copies/new instances
of class Time in getBeginningTime and getEndingTime

Better Design: Time should be immutable

this is an adaption from the 2022 lecture, which had class Date instead,
the Java community has discussed class date for this particular reason!

4 Liskov's Substitution Principle (14 points)

Consider two Scala classes, **Map1** and **Map2** that both implement a common interface **IntMap**, which provides methods for storing mappings of **Int** key values to a **Int** values. Please note that each key value is unique in a **Map** and only maps to a single **Int** value.

```
class Map1 extends IntMap {
  var keys:   Array[Int] = new Array(10)
  var values: Array[Int] = new Array(10)
  var used:   Int = 0

  def size(): Int = used

  def get(key: Int): Int = {
    var index = 0
    for (i <- 0 until used) {
      if(keys(i) == index)
          index = keys(i)
    }
    return values(index)
  }

  def put(key: Int, value: Int): Unit = {
    var i = 0
    while(i < used && keys(i) != key) {
      i += 1
    }
    keys(i) = key
    values(i) = value
    used += 1
  }
}
```

```
class Map2 extends IntMap {
  var entries: List[(Int, Int)] = List()


  def size(): Int = entries.length

  def get(key: Int): Int = {
    var result = 0
    for ((a, b) <- entries) {
      if(a == key){
          result = b
      } }
    return result
  }

  def put(key: Int, value: Int): Unit = {
    entries = (key, value) :: entries
  }
}
```

a) There is a problem concerning the **fixed sized arrays** in **Map1**, violating Liskov's substitution principle in relation to **Map2**. Briefly *describe this problem*, i.e., how it leads to behavior different from **Map2**, and *state the main idea how to solve this issue*. No need to provide the full history and no need to provide a concrete code fix. (3 points)

Hint: Assume from now on that the problem concerning the fixed sized arrays in Map1 is fixed.

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

Your histories should end in a mismatch in the result of **Map1** and **Map2**, where results are one of: return values of an operation, or an exception. You do not need to include a call to a constructor as the first event.

b) Provide a history, in which the mismatch is caused by a problem in class **Map1**. (3 points)

step	operation name	parameter value	result for **Map1**	result for **Map2**
1.	put	key 21, value 7	—	—
2.	get	42	7	≠ 0
3.				
⋮				

I) ✓

II) get 0 → 0 = 0

(= values(0) = default for integer arrays on JVM)

not 2 solution!

c) How would you fix this problem in class **Map1**? (1 point)

first: check against key, not index
also: check again after the loop,
maybe define what the result should be if key not present

d) Provide a history, in which the mismatch is caused by a problem in class **Map2**. (3 points)

step	operation name	parameter value	result for **Map1**	result for **Map2**
1.	put	key 3, value 5		
2.	put	3	6	
3.	get	3		
⋮				

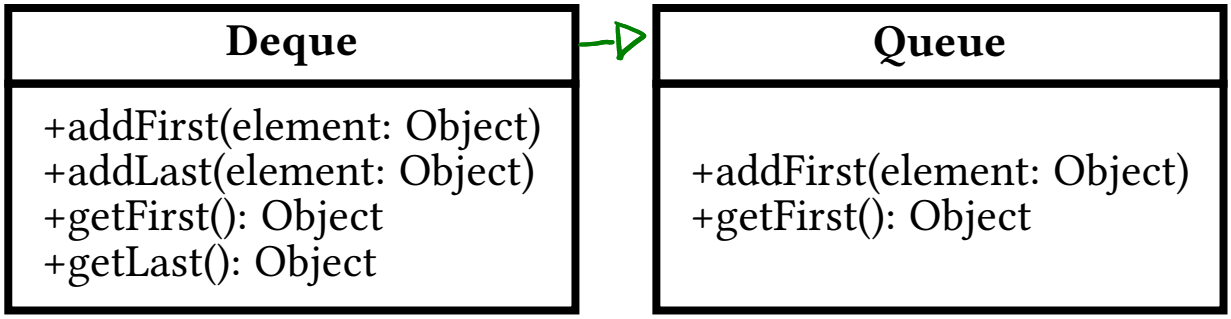
assuming b) is fixed

6 ≠ 5

e) How would you fix this problem in class **Map2**? (2 points)

break/return out of the loop once the entry is found

f) Consider the following two classes:



i. According to Liskov's Substitution Principle, which class is allowed to inherit from the other? Check *one* and briefly explain why. (2 points)

☒ Deque can be a subclass of Queue ☐ Queue can be a subclass of Deque

because

Class Deque it has the required methods to be a subclass of Queue

5 Invariants (16 points)

a) Define: A class invariant is (2 points)

- property/formula/boolean expression over the attributes
- initially established in the contstuctor and preserverd over method calls

(please learn this definition!)

Consider the following class, given in Scala. It implements a "deque", which is short for "double-ended queue". A deque is a data structure which provides efficient insertion and removal of elements both at the front and at the back.

Class **Deque** uses an array `data` which is initialized with some given positive capacity.
If `front == back` then the deque is empty. Otherwise, the array `data` is partially filled with the contents in the cells `data(front)` to `data(back-1)`, which are somewhere in the middle.

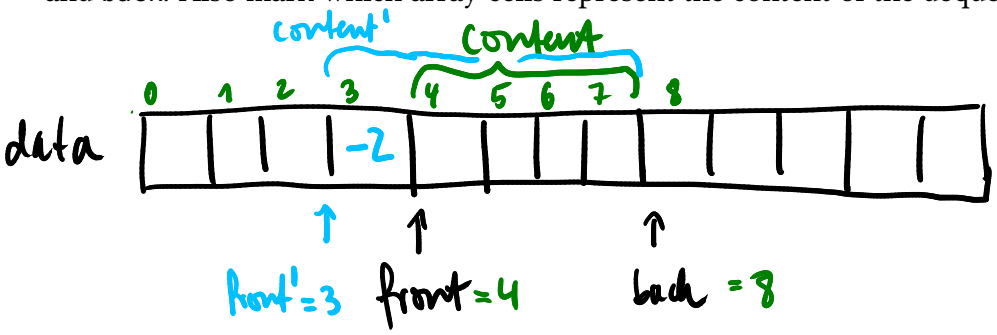
```
class Deque(capacity: Int) {
  var data: Array[Object] = new Array(capacity)
  var front = capacity / 2
  var back  = capacity / 2

  def size(): Int = back - front

  def addFront(value: Object): Unit = {
    require(0 < front)
    front -= 1
    data(front) = value
  }

  def removeFront(): Object = {
    val result = data(front)
    front += 1
    return result
  }

  def addBack(value: Object): Unit = {
    require(back < data.length)
    data(back) = value
    back += 1
  }

  def removeBack(): Object = {
    back -= 1
    return data(back)
  }
}
```

where the respective invariants need to be checked carefully, because statements could potentially invalidate these

if front == data.length
exception prevents the bad state

out of bounds exception in this case!
however, it is too late: the object is already in a bad state

b) Draw a schematic picture of a *non-empty* instance of **Deque**. It should include `data`, `front`, and `back`. Also mark which array cells represent the content of the deque. (3 points)



ex: addFront(-2)
front--
data(front) = -2

c) Which of the following properties corresponds to **class invariants** of **Deque**? (5 points)

Each correct answer gives 1 point, no negative points.

Note that some methods have *preconditions*, denoted by `require`, which throws an exception if the condition is not satisfied and therefore aborts the execution of the method (like `assert`).

note: all are established initially by the constructor

`0 <= front`	☑ Yes	☐ No
II) `0 <= back`	☐ Yes	☑ No
III) `front <= back`	☐ Yes	☑ No
`front <= data.length`	☑ Yes	☐ No
`back <= data.length`	☑ Yes	☐ No

because exception happens before the state change

d) At least one of the above is *not* an invariant. Pick **one** of those that are not invariants and **explain** how this property could possibly be violated. (2 points)

e) For your pick in **d)** provide a suitable precondition that ensures that this property **becomes** an invariant.

We have to add (2 points)

II) III)

`require(` 0 < back front < back `)`

to the following method of class **Deque**: (2 points)

removeBack removeBack

removeFront

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*