Prof. Dr. Jasmin Blanchette
Dr. Martin Desharnais-Schäfer
Dr. Michael Kirsten

Ludwig-Maximilians-Universität München
Institut für Informatik
Discussion on 30.10.2026
Homework due on 06.11.2026 at 10:15

Possible solution for Exercise Sheet 3 in

Scientific and Technical English

WiSe 2026/27

The exercise sheets consist of in-class exercises and homework. The in-class exercises take place during the second half of the lecture time slots. The homework, which is optional and ungraded, can be submitted via the "Homework" section in Moodle. The homework is subject to peer review.

Unless indicated otherwise, generative artificial intelligence assistants such as ChatGPT may be used, as long as you acknowledge how you use them as specified by the Institute's policy on plagiarism.[1] However, you may not use such tools to generate peer reviews for you. In addition, we strongly recommend that you do not use them to generate entire solutions, since that would defeat the purpose of the exercises.

Homework 3-3 ***Utterly Unstylish*** As a part of a seminar paper, you are asked to write a brief introduction to inheritance in Java. For some reason, you have decided that it is your personal objective to hand in the most poorly written text possible for this assignment. Refer back to the style principles from the lecture, and choose any number of them to focus on. You can either try to break one of them as comprehensively as possible or combine multiple avenues of stylistic offense. Choose whichever option seems the more atrocious to you. Your text should be about 250 words long.

POSSIBLE SOLUTION:

The Hero with a Thousand Subclasses

As the literal bedrock of its class hierarchy system, the foundationalness of inheritance in Java for the wonderful world of object-orientated programming cannot be overstated. The `extends` keyword is the main character of the great epic of inheritance, because it is the one that facilitates the creation of an inheritance relationship from one class to another. We call the class that is being inherited from the *superclass*, while the class that performs the act of inheriting is the *subclass* on the other hand. Within this so-called subclass, the programmer can now access all the definitions he wants from the parent class (e.g., methods, attributes, etc.). The inverse, however, remains untrue; inheritance is a one-way mirror, and the

[1] `https://www.medien.ifi.lmu.de/lehre/Plagiate-IfI.pdf`

members of the subclass remain shrouded in mystery from the vantage point of the superclass. Even so, this functionality can provide a silver bullet asset in the grueling fight against code duplication. However, it must needs be remarked that there is one striking exception to this: As soon as a definition brandishes the `protected` keyword, it becomes invisible to subclasses. Another way to achieve an even more exhaustive prevention of inheritance access is the `final` keyword, which shields a class from anyone who might try to establish an inheritance connection. If one, however, was foolish enough to still attempt such a feat in the face of this adversity, the compiler would surely answer with a devastating error message.

Homework 3-4 ***From Casual to Formal*** Consider the following excerpts from two computer science texts. They are written in a rather casual register. Rewrite them to suit the level of formality appropriate for a university paper.

a) From Joshua Bloch and Neal Gafter, *Java Puzzlers: Traps, Pitfalls, and Corner Cases*, Addison-Wesley Professional, 2005:

> The behavior of this program is platform independent: It won't compile on any platform. If you tried to compile it, you got an error message that looks something like this:
>
> ```
> LinePrinter.java:3: ';' expected
> // Note: \u000A is Unicode representation of linefeed (LF)
> ^
> 1 error
> ```
>
> If you are like most people, this message did not help to clarify matters.

POSSIBLE SOLUTION:

The behavior of this program is platform-independent: It will not compile on any platform. Trying to compile it yields the following cryptic error message:

```
LinePrinter.java:3: ';' expected
// Note: \u000A is Unicode representation of linefeed (LF)
^
1 error
```

b) From Miran Lipovača, "Learn You a Haskell for Great Good!":[2]

> It would seem as though we've gained the ability to temporarily extract things from Maybe values without having to check if the Maybe values

[2] `https://learnyouahaskell.com/`

are Just values or Nothing values at every step. How cool! If any of the values that we try to extract from are Nothing, the whole do expression will result in a Nothing. We're yanking out their (possibly existing) values and letting >>= worry about the context that comes with those values. It's important to remember that do expressions are just different syntax for chaining monadic values.

POSSIBLE SOLUTION:

We can temporarily extract the contents from `Maybe` values without needing to check if the `Maybes` are `Just` or `Nothing` values at every step. If any of the `Maybes` that we try to extract from are `Nothing`, the entire `do` expression will evaluate to `Nothing`. We can retrieve their (possibly existing) values and let the `>>=` operator handle the context associated with those values. Recall that `do` expressions are simply different syntax for chaining monadic values.