Modellierung und Programmierung 1

Übung 10

Stefan Preußner

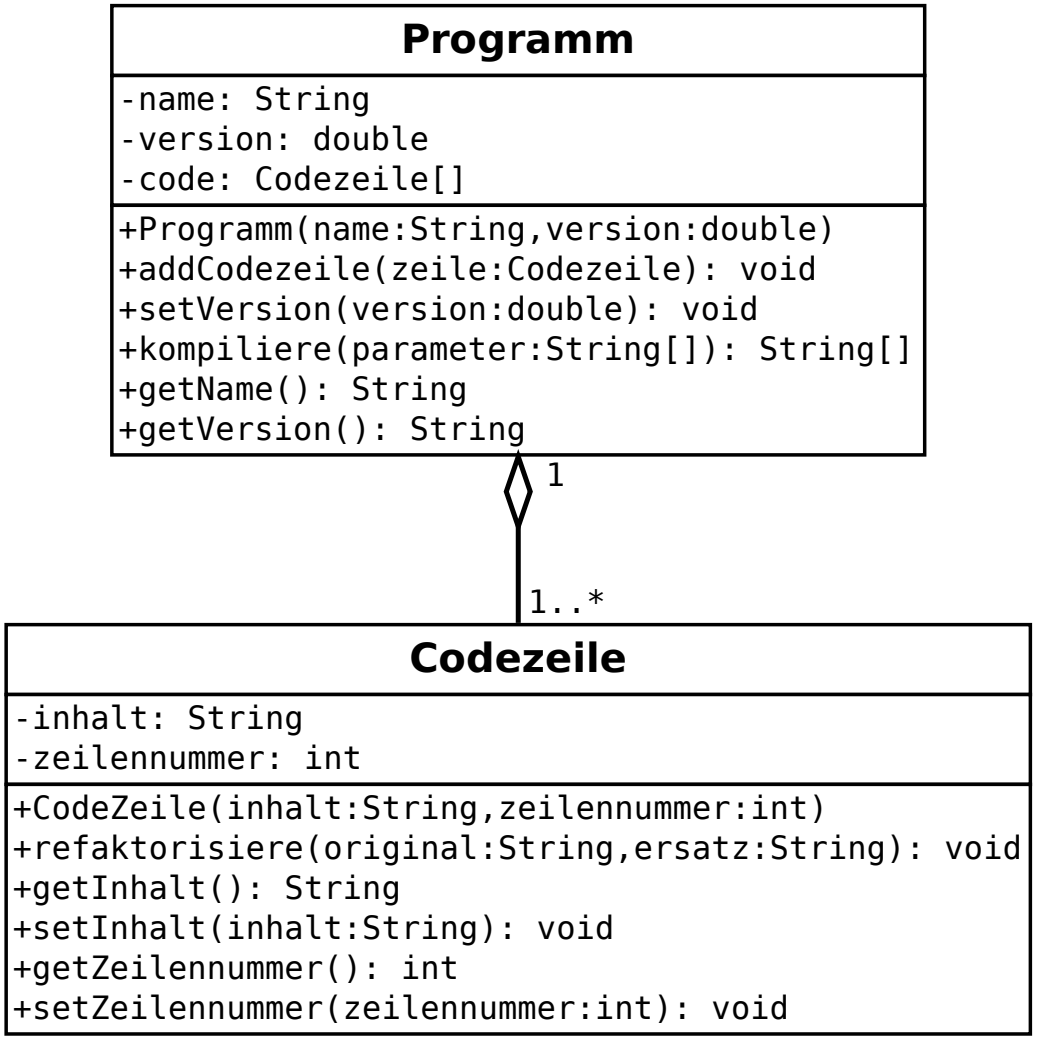
25./ 26. Januar 2021

Thema	Vorlesung	Übung
UML	Vorlesung 2	Übungen 2, 3
Datentypen, Ausdrücke & Kontrollstrukturen	Vorlesungen 3, 4, 5	—
Vererbung & Interfaces	(Vorlesung 2)	Übungen 4, 9
Ausnahmen	Vorlesung 6	Übung 8
Collections	Vorlesung 7	Übungen 6, 7
Strings	Vorlesung 8	Übung 7
I/O	Vorlesung 9	Übung 8
Pseudocode	—	Übung 9
Rekursion	Vorlesung 10	(Übung 9)
Threads	Vorlesung 11	—
Grafik	Vorlesung 13	—

UML

Es soll ein `Programm` modelliert werden. Ein `Programm` besitzt einen Namen (z.B. "MuPTest") und eine Versionsnummer (z.B. 1.23). Die Versionsnummer kann, im Gegensatz zum Namen, nachträglich geändert werden. Ein Programm kann unter Angabe einer Liste von Parametern (Strings) kompiliert werden, dabei wird eine Liste von Fehlermeldungen zurückgegeben. Ein Programm besteht aus mindestens einer `CodeZeile`, wobei jede `CodeZeile` nur zu genau einem Programm gehört. Codezeilen können nachträglich zu einem Programm hinzugefügt werden. Eine `CodeZeile` besteht aus einer Zeilennummer und einem Inhalt (z.B. "x++;"). Zeilennummer und Inhalt können über entsprechende Getter und Setter abgefragt und geändert werden. Eine `CodeZeile` kann außerdem refaktorisiert werden, dabei wird der zu ersetzende Text und der Ersatztext angegeben.

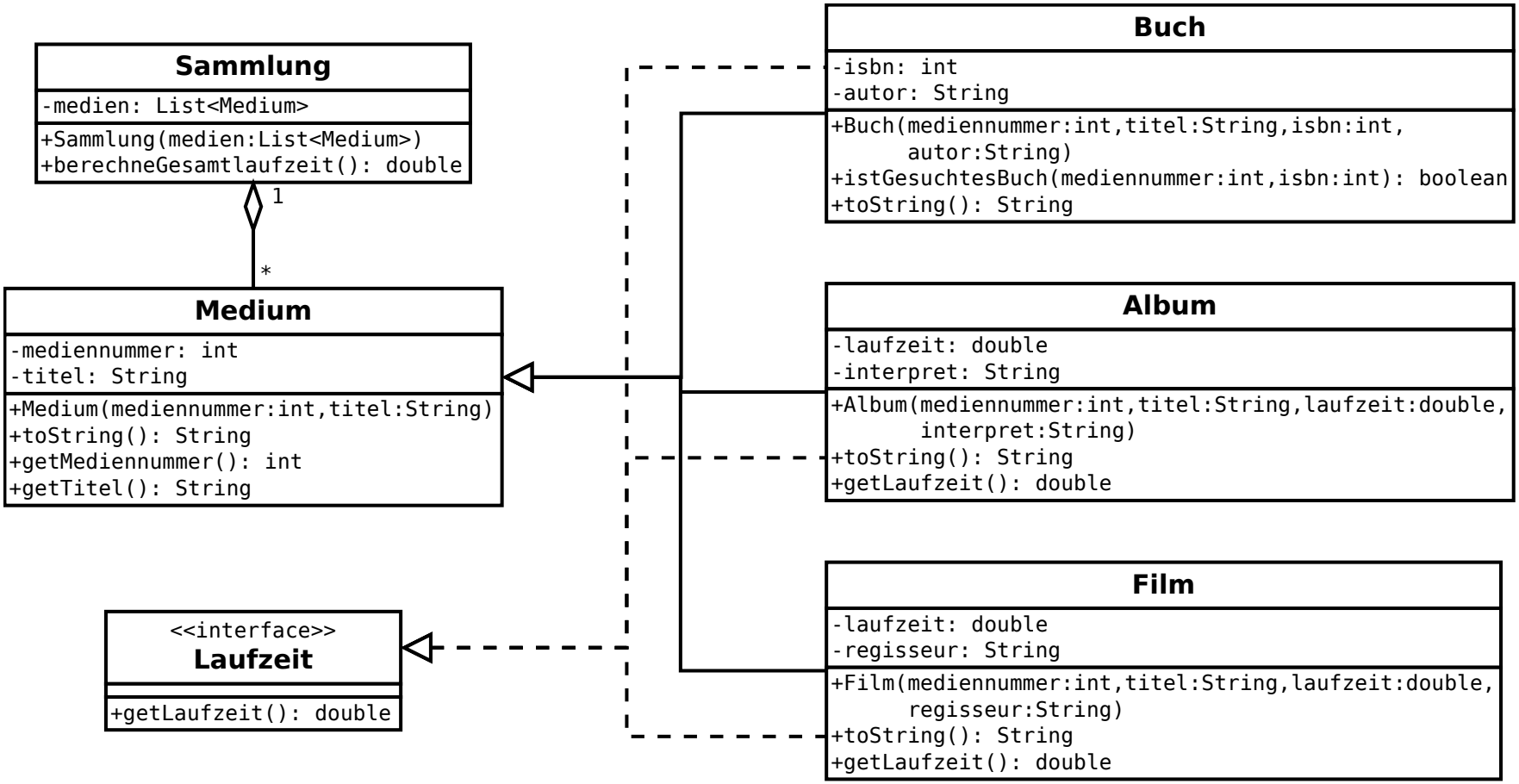
Es soll ein `Programm` modelliert werden. Ein `Programm` besitzt einen Namen (z.B. "MuPTest")und eine Versionsnummer (z.B. 1.23). Die Versionsnummer kann, im Gegensatz zum Namen, nachträglich geändert werden. Ein Programm kann unter Angabe einer Liste von Parametern (Strings) kompiliert werden, dabei wird eine Liste von Fehlermeldungen zurückgegeben. Ein Programm besteht aus mindestens einer `CodeZeile`, wobei jede `CodeZeile` nur zu genau einem Programm gehört. Codezeilen können nachträglich zu einem Programm hinzugefügt werden. Eine `CodeZeile` besteht aus einer Zeilennummer und einem Inhalt (z.B. "x++;"). Zeilennummer und Inhalt können über entsprechende Getter und Setter abgefragt und geändert werden. Eine `CodeZeile` kann außerdem refaktorisiert werden, dabei wird der zu ersetzende Text und der Ersatztext angegeben.



Es soll ein `Programm` modelliert werden. Ein `Programm` besitzt einen Namen (z.B. "MuPTest")und eine Versionsnummer (z.B. 1.23). Die Versionsnummer kann, im Gegensatz zum Namen, nachträglich geändert werden. Ein Programm kann unter Angabe einer Liste von Parametern (Strings) kompiliert werden, dabei wird eine Liste von Fehlermeldungen zurückgegeben. Ein Programm besteht aus mindestens einer `CodeZeile`, wobei jede `CodeZeile` nur zu genau einem Programm gehört. Codezeilen können nachträglich zu einem Programm hinzugefügt werden. Eine `CodeZeile` besteht aus einer Zeilennummer und einem Inhalt (z.B. "x++;"). Zeilennummer und Inhalt können über entsprechende Getter und Setter abgefragt und geändert werden. Eine `CodeZeile` kann außerdem refaktorisiert werden, dabei wird der zu ersetzende Text und der Ersatztext angegeben.

Vererbung & Interfaces

Implementieren Sie das nachfolgende UML-Diagramm:



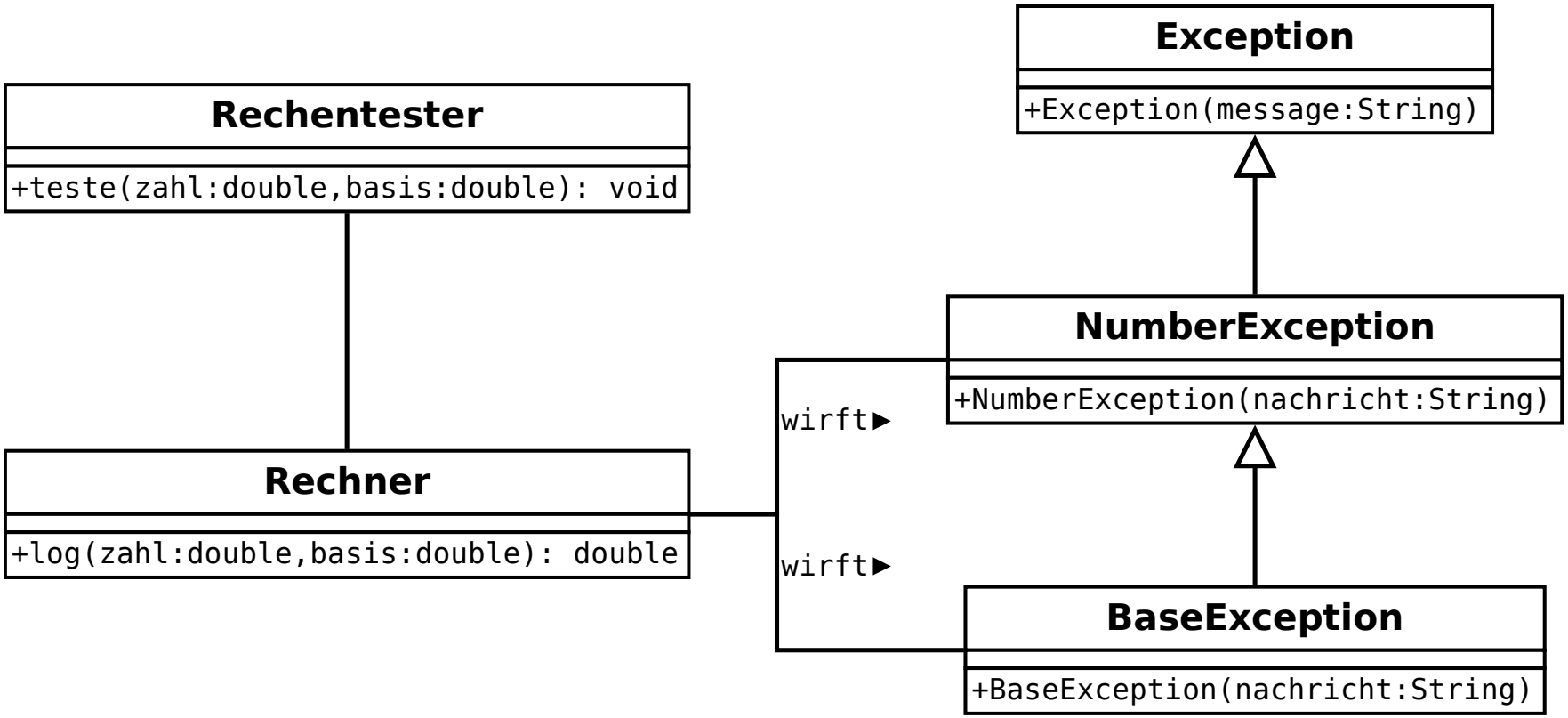
Beachten Sie bei der Umsetzung die folgenden Hinweise:

- Die Klasse `Medium` sowie das Interface `Laufzeit` sind bereits vorgegeben.
- Klasse `Album`:
  - Die Methode `toString()` soll folgenden String zurückgeben:
    `Mediennummer: <mediennummer>, Titel: <titel>, Laufzeit: <laufzeit>, Interpret: <interpret>`
- Klasse `Film`:
  - Die Methode `toString()` soll folgenden String zurückgeben:
    `Mediennummer: <mediennummer>, Titel: <titel>, Laufzeit: <laufzeit>, Regisseur: <regisseur>`

- Klasse `Buch`:
  - Die Methode `toString()` soll folgenden String zurückgeben:
    `Mediennummer: <mediennummer>, Titel: <titel>, ISBN: <isbn>, Autor: <autor>`
  - Die Methode `istGesuchtesBuch` soll eine Mediennummer und eine ISBN übernehmen und genau dann `true` zurückgeben, wenn wenigstens eine der beiden Nummern übereinstimmt.

Ausnahmen

Implementieren Sie das nachfolgende UML-Diagramm:



Beachten Sie bei der Umsetzung die folgenden Hinweise:

- Die Methode `log(zahl, basis)` der Klasse `Rechner` soll:
  - eine `NumberException` mit der entsprechenden Fehlermeldung werfen, wenn `zahl` negativ oder 0 ist
  - eine `BaseException` mit der entsprechenden Fehlermeldung werfen, wenn `basis` negativ, 0 oder 1 ist.

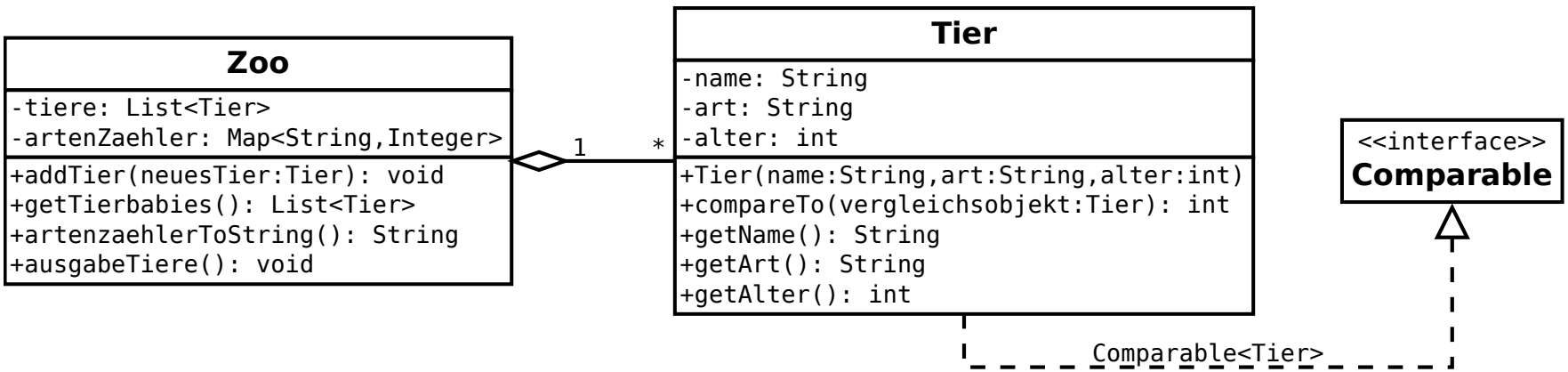
  Die Ausnahmen sollen nicht weiter behandelt werden.

Beachten Sie bei der Umsetzung die folgenden Hinweise:

- Die Methode `teste(zahl, basis)` der Klasse `Rechentester` soll das Ergebnis der Berechnung `Rechner.log(zahl, basis)` ausgeben. Alle auftretenden Ausnahmen sollen so behandelt werden, dass eine entsprechende Fehlermeldung auf der Fehlerkonsole `System.err` ausgegeben wird. Unabhöngig davon, ob eine Ausnahme behandelt wurde oder nicht, sollen zum Schluss die Parameter der Methode ausgegeben werden.

Collections

Implementieren Sie das nachfolgende UML-Diagramm:



Beachten Sie bei der Umsetzung die folgenden Hinweise:

- Klasse `Tier`:
  - `name` ist der Name des Tieres. `art` gibt die Tierart und `alter` das Alter in Tagen an.
  - `compareTo` soll zuerst die Art lexikographisch vergleichen. Ist `art` bei beiden Tieren gleich, dann soll der Name lexikographisch verglichen werden.
  - `toString` soll die Informationen über ein Tier in folgendem Format zurückgeben:
    `<Art> <Name>, <Alter> Tage alt`

Beachten Sie bei der Umsetzung die folgenden Hinweise:

- Klasse `Zoo`:
  - `tiere` ist die Liste aller Tiere im Zoo. `artenZaehler` soll für jede Tierart zählen, wie häufig sie im Zoo vorkommt.
  - `addTier` soll das übergebene Tier zur Liste aller Tiere hinzufügen und den Artenzählen aktualisieren.
  - `getTierbabies` soll eine Liste aller Tiere, die jünger sind als 100 Tage, zurückgeben.
  - `artenzaehlerToString` soll einen String zurückgeben, der tabellarisch die Häufigkeit jeder Tierart angibt.
  - `ausgabeTiere` soll alle Tiere im Zoo ausgeben. Die Ausgabe soll dabei in der Reihenfolge erfolgen, wie sie durch die Sortierfunktion der Klasse Tier vorgegeben ist. tiere soll nicht verändert werden.

Strings

(siehe Paket `strings`)

Rekursion und Pseudocode

(siehe Paket `rekursion` sowie
die Übungen 5 und 6 in der Klasse `StringUebung`)