SE Besprechung

Übung 1 – Programmverständnis, Dokumentation

Mengia Zollinger

Teaching Assistant

Mengia Zollinger <mengia.zollinger@uzh.ch>

7. Semester Wirtschaftsinformatik

Fasttrack bei A. Bernstein

Abgabe

Erlaubte Datentypen:	Dokument als PDF, Source Code als .java keine Bilder
Beschriftung:	Namen der Gruppenmitglieder in Kopfzeile
Einteilung:	3er Gruppen, keine Einzelarbeiten
Abgabe an:	Nicolas Hoby <nhoby@access.uzh.ch> und Irina Todoran <todoran@ifi.uzh.ch>

Übung 1

Aufgabe 1:

Allgemeine Informationen

Aufgabe 2:

Einführung und Installationsanleitung der JGAP Bibliothek

Aufgabe 3 – Verständnis des Codes

Aufgabe 3.1 – Struktur

Ziel:

- Gesamtüberblick über Quellcode gewinnen
- UML Klassendiagramm auffrischen

Aufgabe:

- Klassendiagramm modellieren
- Alle Klassenvariabeln & mind. 5 Methoden modellieren
- Konstruktoren & Setter/Getter nicht modellieren

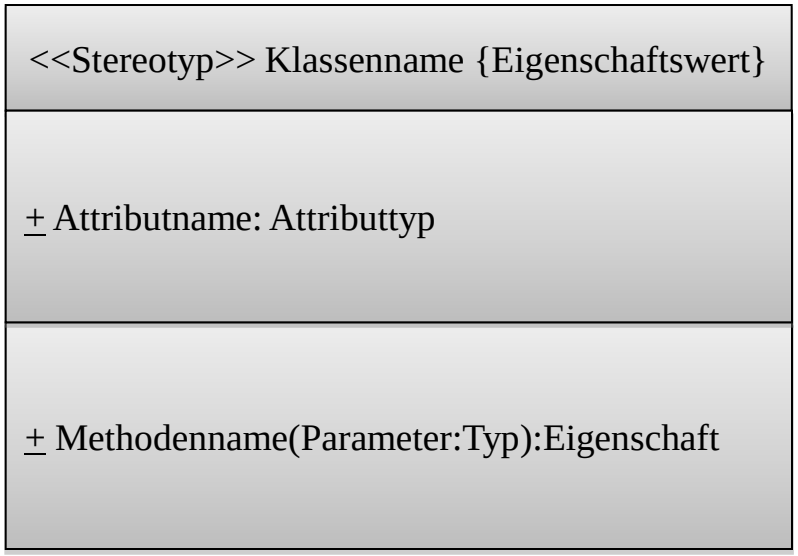
Aufgabe 3 – Verständnis des Codes

Kritik:

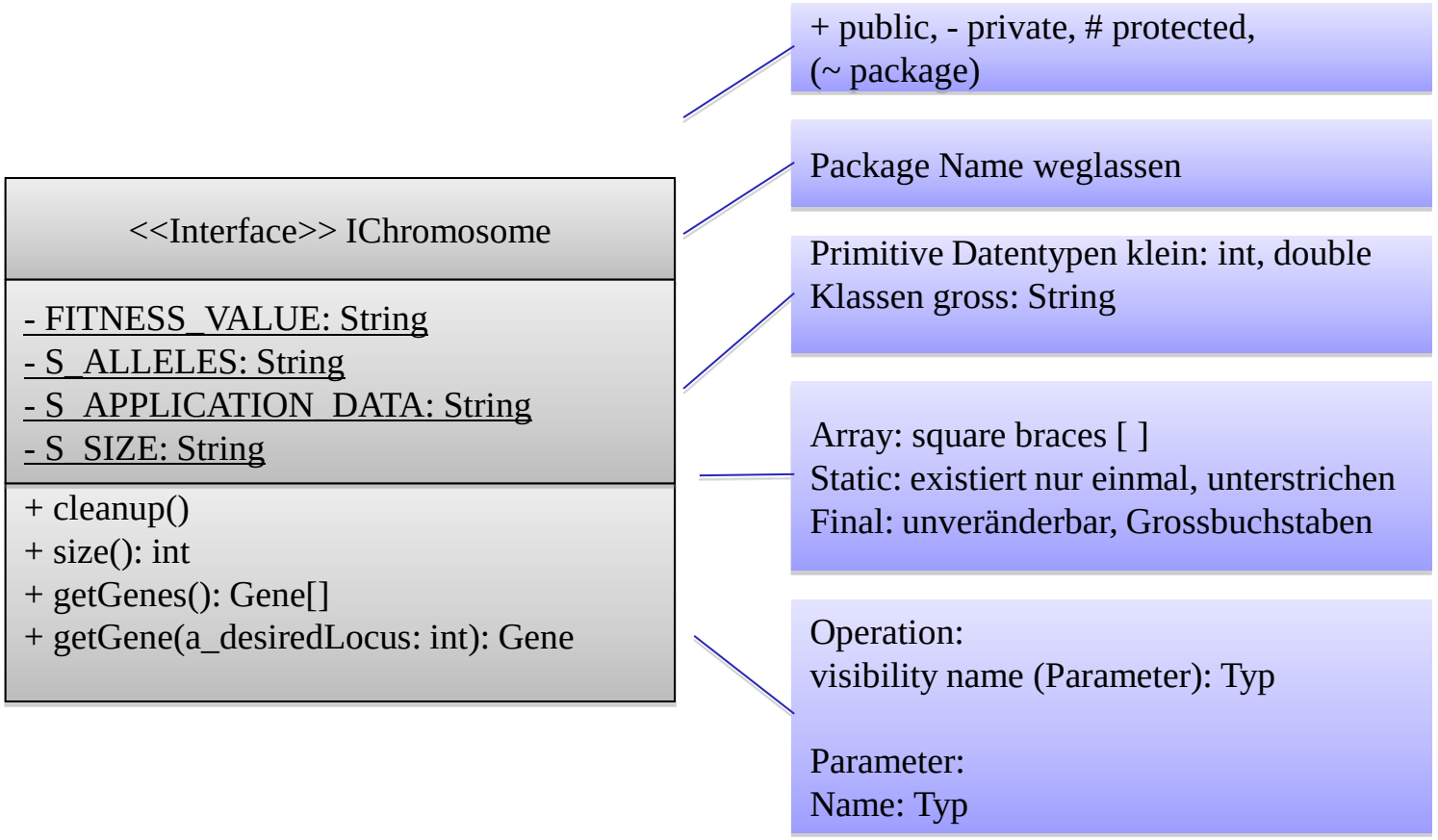
- Nur die angegebenen Klassen modellieren
- Alle Klassen auf eine A4 Seite
- Nicht von Hand zeichnen, z.B. mit Visio, UMLet, etc.
- Attribute, Methoden & Verbindungen vergessen
- Methoden in der falschen Klasse modelliert (Interface)

Aufgabe 3 – Verständnis des Codes

- Kritik:

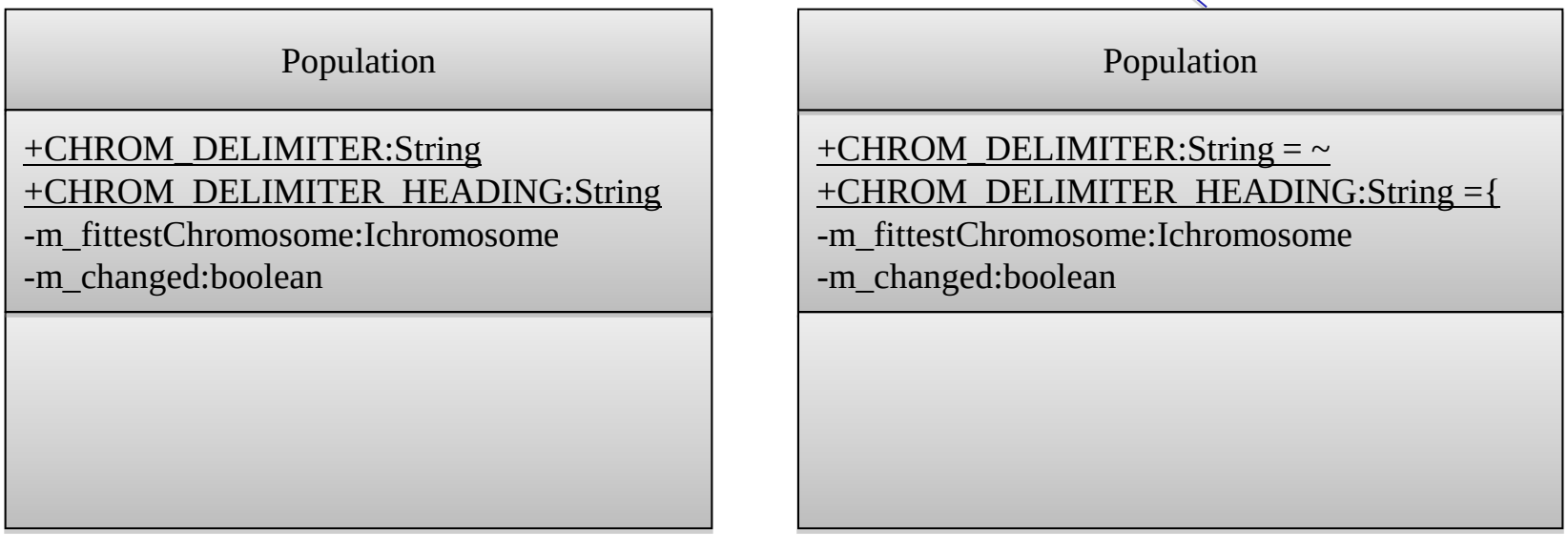


Aufgabe 3 – Verständnis des Codes

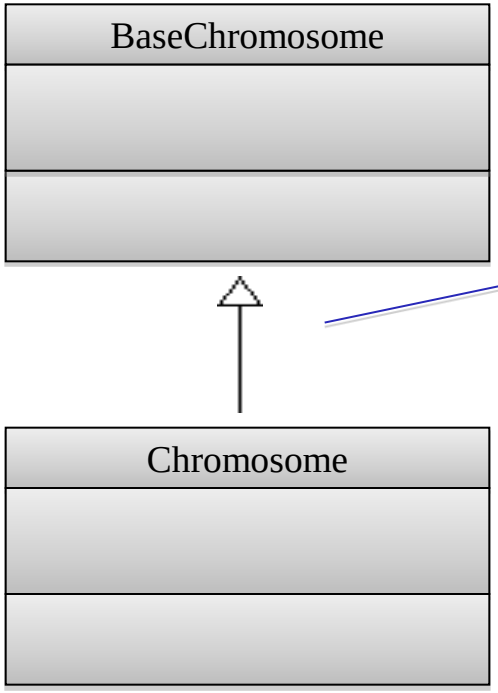


Aufgabe 3 – Verständnis des Codes

Initialwert weglassen!



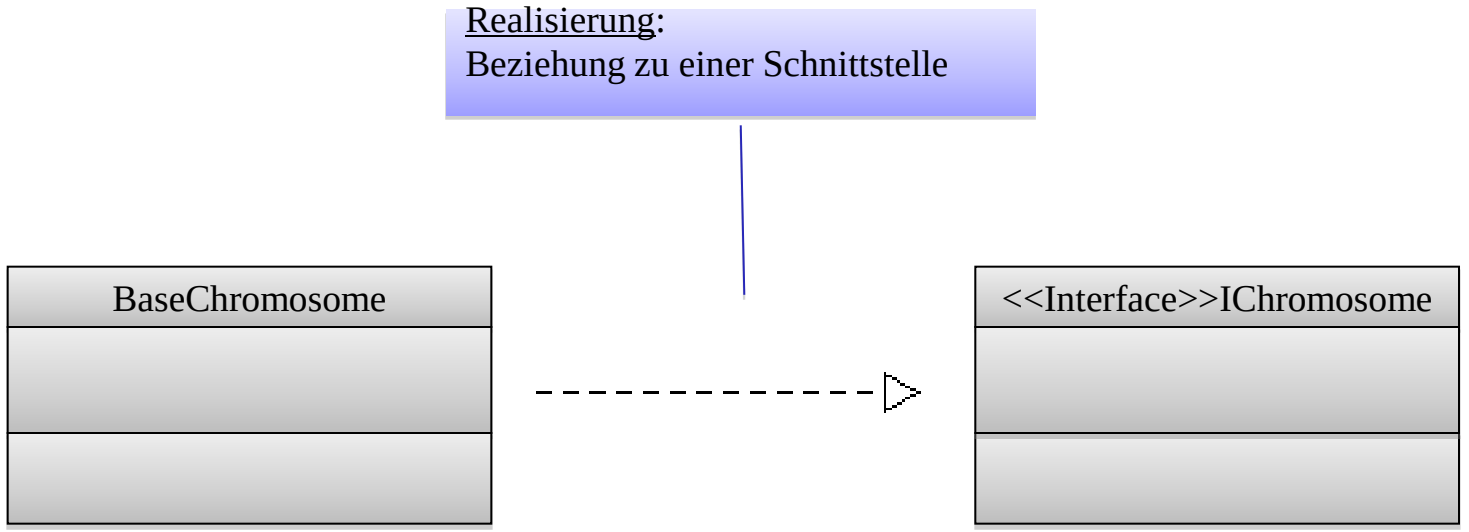
Aufgabe 3 – Verständnis des Codes



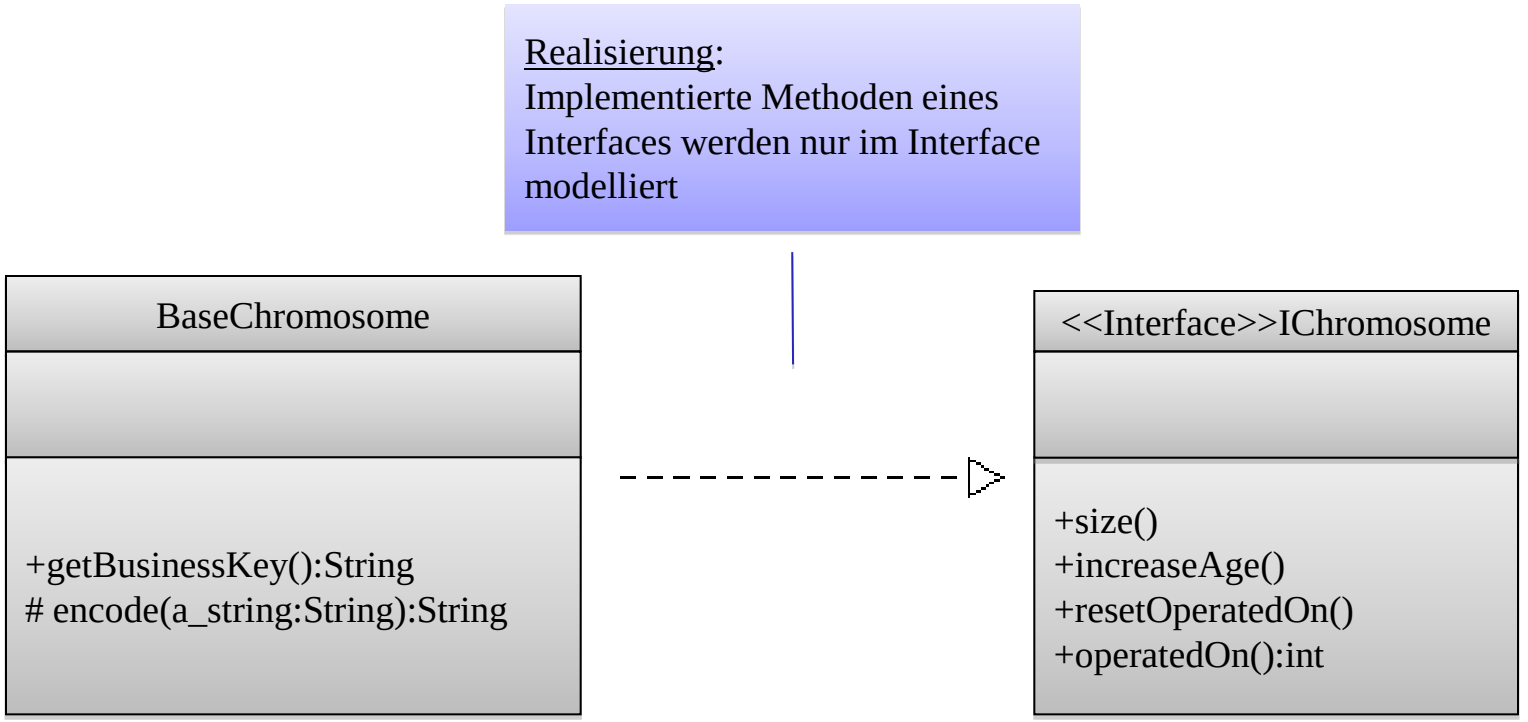
Vererbung:
Oberklasse ist eine Generalisierung der Unterklasse
„Is-a“ Beziehung

Pfeile nicht beschriften (ausser mit Zahlen)

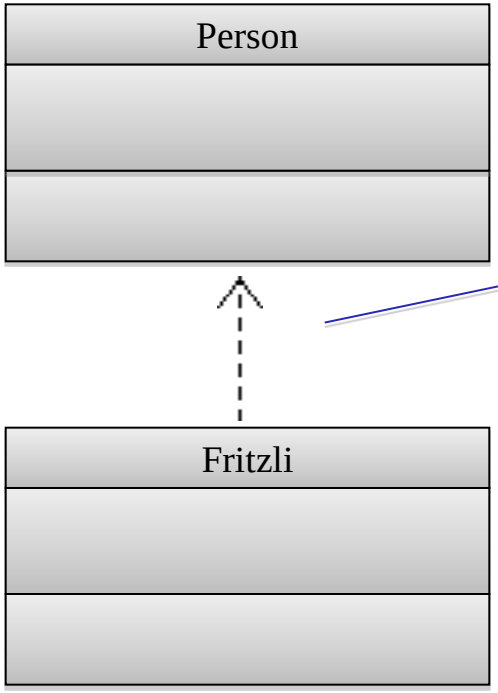
Aufgabe 3 – Verständnis des Codes



Aufgabe 3 – Verständnis des Codes

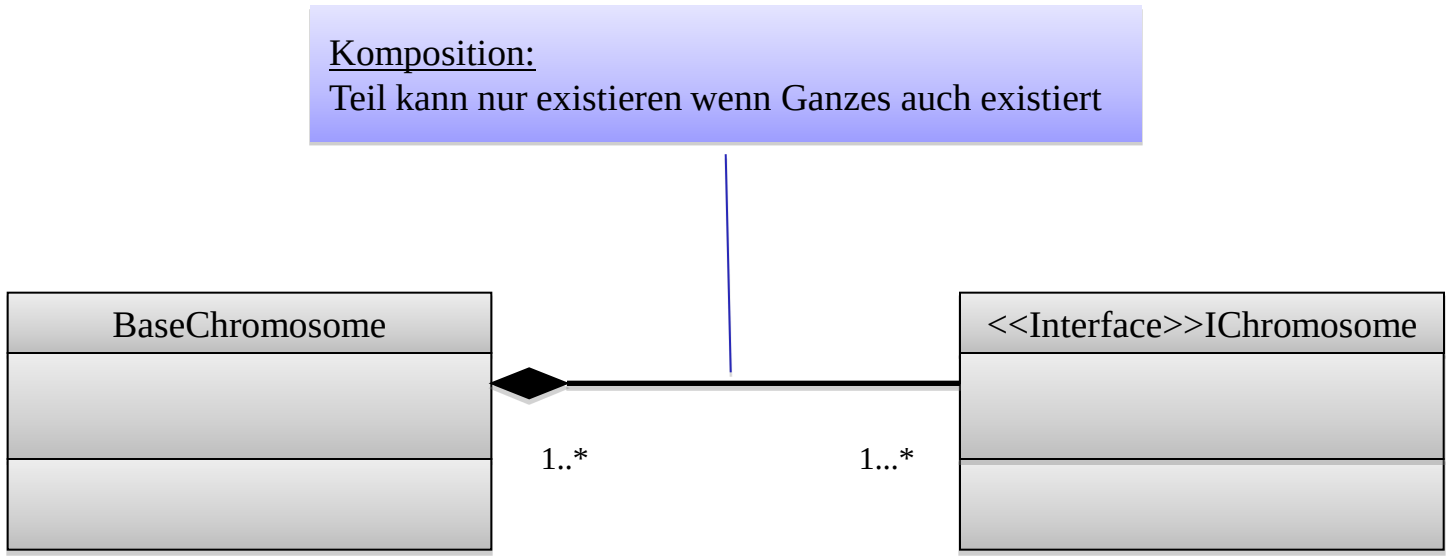


Aufgabe 3 – Verständnis des Codes



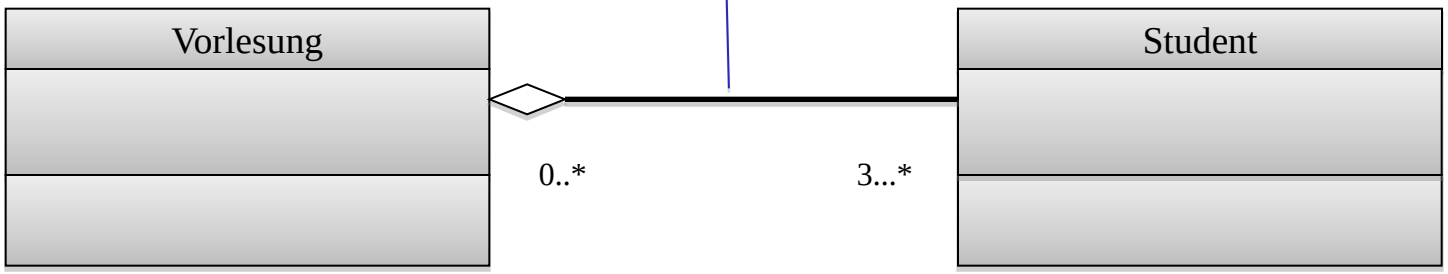
Instanzbeziehung:
Object „is instance of“ Class

Aufgabe 3 – Verständnis des Codes



Aufgabe 3 – Verständnis des Codes

Aggregation:
Teil kann existieren auch wenn Ganzes nicht existiert



Aufgabe 3 – Verständnis des Codes

Aufgabe 3.2 – Anwendungsbereich

Ziel:

•Überblick über einzelnen Anwendungsbereich gewinnen

Aufgabe:

•Art von Genen identifizieren

•Klassennamen & Datentypen eruieren

•Supergenes, abstrakte Klassen & Interfaces ignorieren

Aufgabe 3 – Verständnis des Codes

Kritik:

- Nicht alle Klassen aufgezählt
- Nur Klassenname & Datentypen gefragt

Aufgabe 3 – Verständnis des Codes

Klassenname	Datentyp
BaseGeneImpl	Object
BooleanGene	Boolean
CompositeGene	Depends on subgenes
FixedBinaryGene	Numbers, only 0 and 1
MapGene	Map of alleles
MathGene	Mathematical operators
MyIntegerGene	Integer

Aufgabe 3 – Verständnis des Codes

DoubleGene	Double
IntegerGene	Integer
MultipleIntegerGene	Integer with restrictions
SetGene	Set of alleles
StringGene	String

Aufgabe 3 – Verständnis des Codes

CompositeGene ist kein eigener Datentyp, sondern besteht aus mehreren Subgenen

Unterstützt mehrere Datentypen

Aufgabe 3 – Verständnis des Codes

Aufgabe 3.3 – Verhalten

Ziel:

- Teilablauf der künstlichen Evolution nachvollziehen können
- Verhalten der Methode evolve() der Klasse GABreeder verstehen
- UML Sequenzdiagramm auffrischen

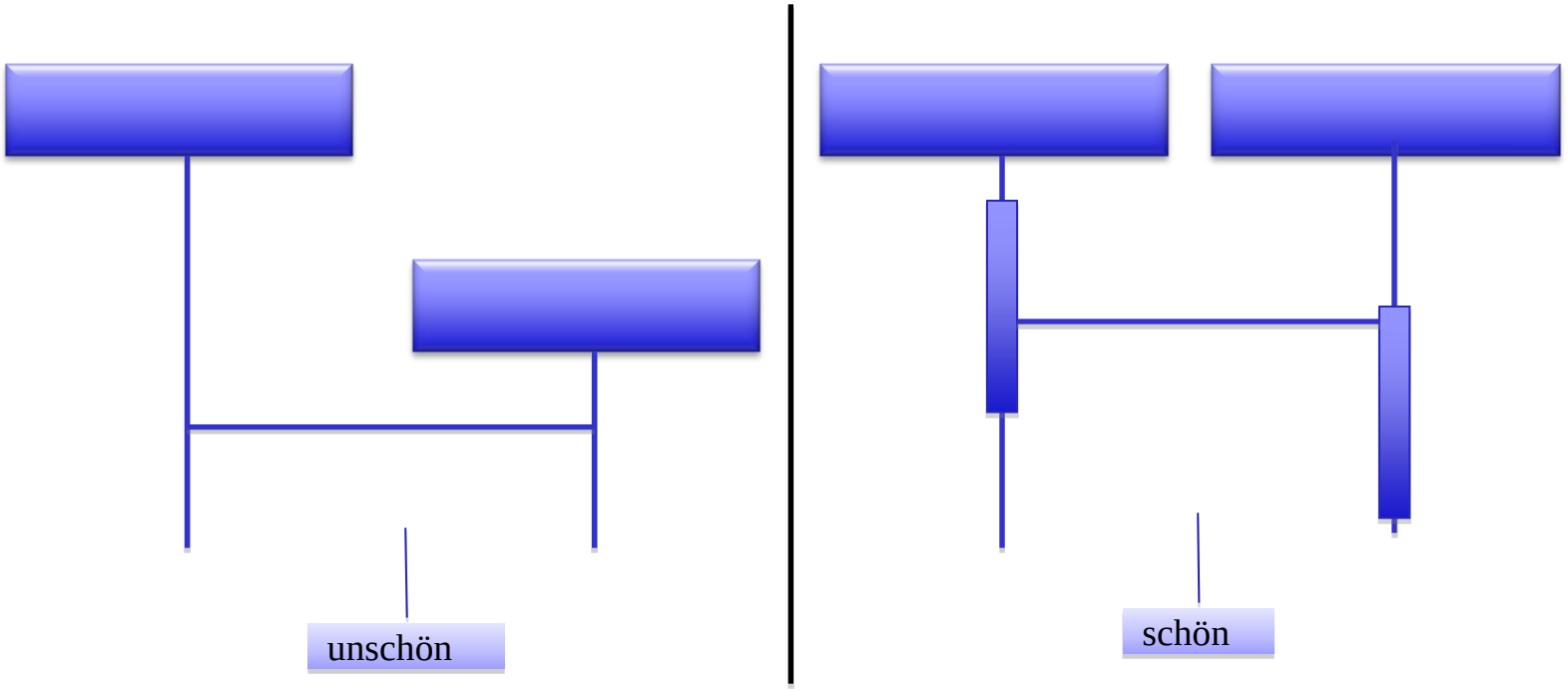
Aufgabe:

- Obersten 24 Zeilen der Methode evolve() modellieren

Aufgabe 3 – Verständnis des Codes

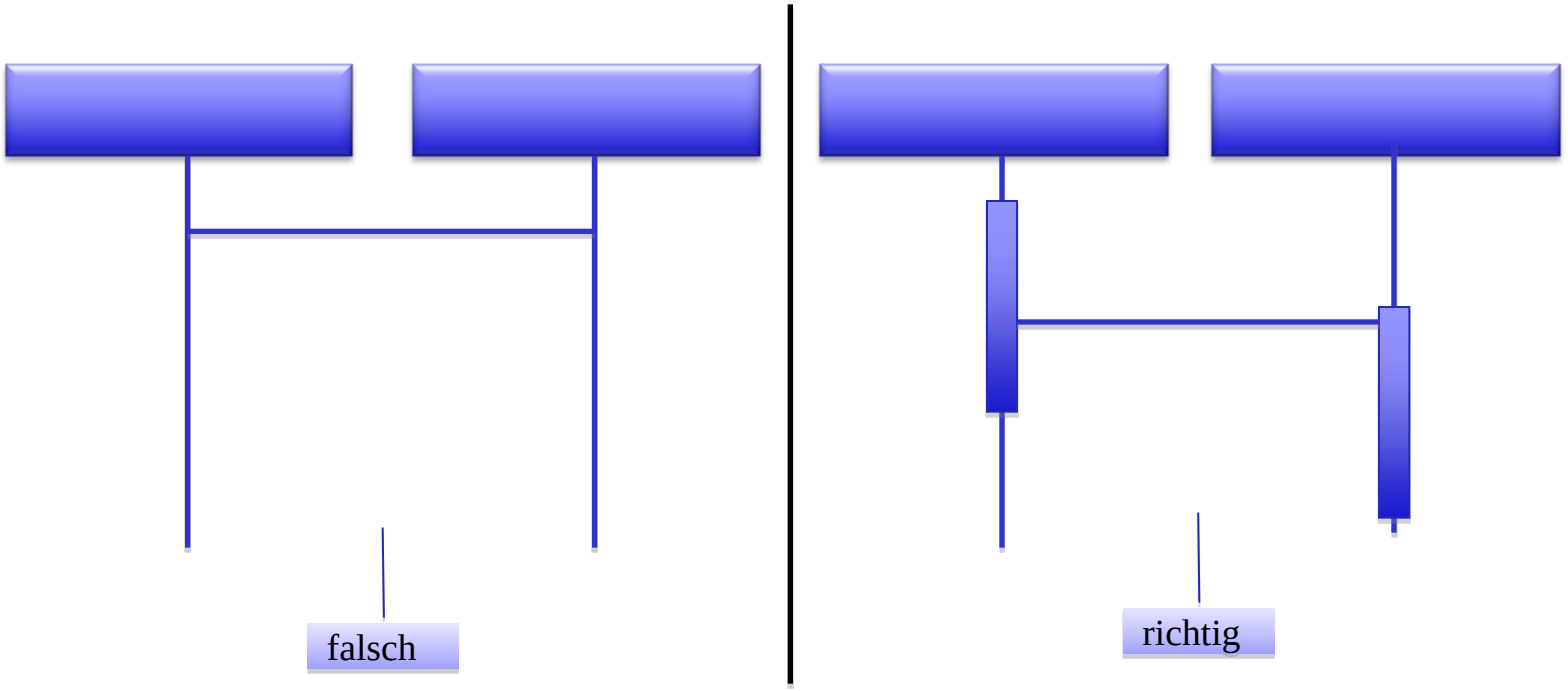
Kritik:

- Auf die Darstellung achten:



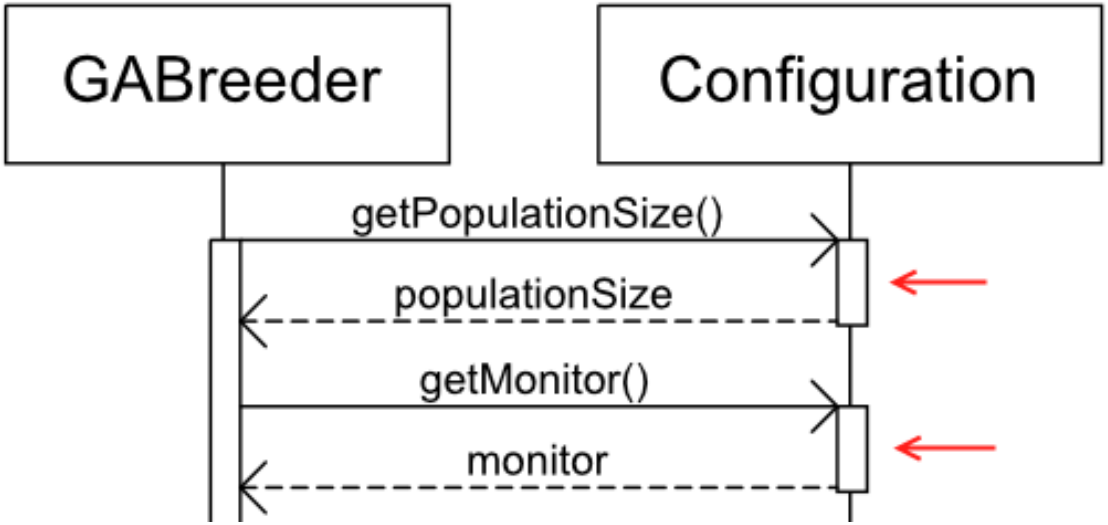
Aufgabe 3 – Verständnis des Codes

- Aktivierungsbereiche nicht eingezeichnet



Aufgabe 3 – Verständnis des Codes

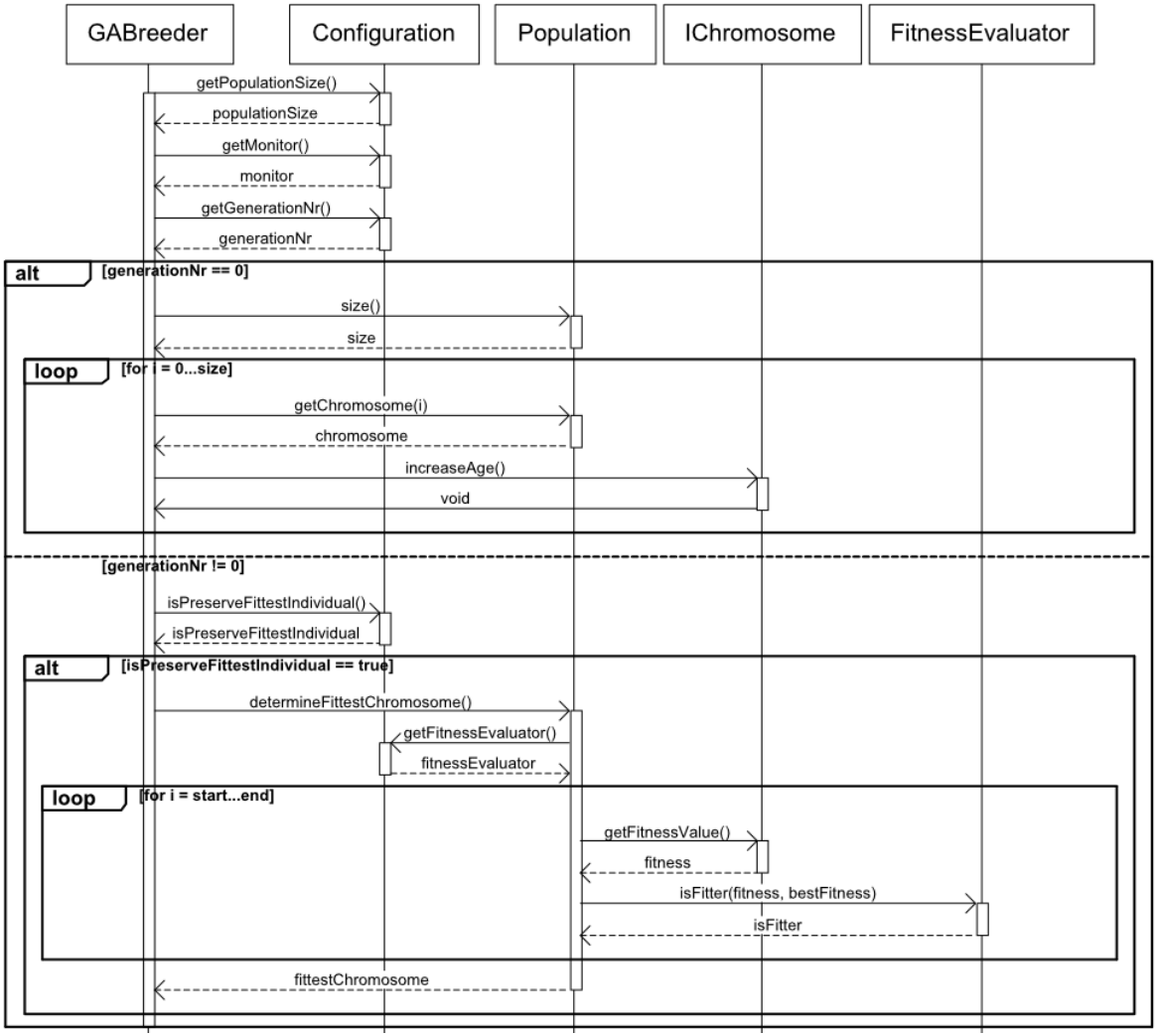
- Aktivierungsbereich nur, wo Klasse effektiv aktiv ist



Aufgabe 3 – Verständnis des Codes

- <create> nur, wenn Objekt neu erstellt wird (new Operator), nicht wenn es zum ersten Mal benutzt wird
- Nur Methodenaufrufe darstellen, keine Variablenzuweisungen

Aufgabe 3 – Verständnis des Codes



Aufgabe 4 – Verbessern des Codes

Aufgabe 4.1 – Dokumentation

Ziel:

- Javadoc Kommentare erarbeiten

Aufgabe:

- 3 Methoden dokumentieren (Javadoc Kommentar & innerhalb)

Aufgabe 4 – Verbessern des Codes

Bewertung

- Je 0.5 Punkte für Methodenkommentare (3)
- Je 0.25 Punkte für Kommentare im Code (6)
- 1 Bonuspunkt für allgemeinen Eindruck

Kritik:

- Nicht alle Kommentare ausgefüllt
- Nicht Code nacherzählen, sondern erklären warum etwas geschieht

Aufgabe 4 – Verbessern des Codes

Schlechtes Beispiel:

```
if (a_changeDifference == 0) {
        // Max Fitness wird zurückgegeben
        return a_maxFitness;
}
```

Gutes Beispiel:

```
if (a_changeDifference == 0) {
        // Bester Fall: Wenn Wechseldifferenz minimal
        // (=0) ist, ist der Fitnesswert maximal.
        return a_maxFitness;
}
```

Aufgabe 4 – Verbessern des Codes

Aufgabe 4.2 – Verbesserung des Codes

Ziel:

- Code analysieren
- Verbesserungsvorschläge ausarbeiten

Aufgabe:

- Analyse der Methode checkMinFitness()
- Stichworte zur Verbesserung der Methode notieren

Aufgabe 4 – Verbessern des Codes

Auswahl an Schwachstellen:

- Methodenname checkMinFitness(): Methode checkt nicht nur, sie verändert auch den Zustand des Systems
- Feedback fehlt: Methode gibt keine Auskunft darüber, ob und was gemacht wurde
- Kommentare: erzählen den Code (..is doubled), anstatt ihn zu erklären
- Methodenkommentare erwähnen Variablen, ohne dass User darauf Zugriff hat

Aufgabe 4 – Verbessern des Codes

Bewertung:

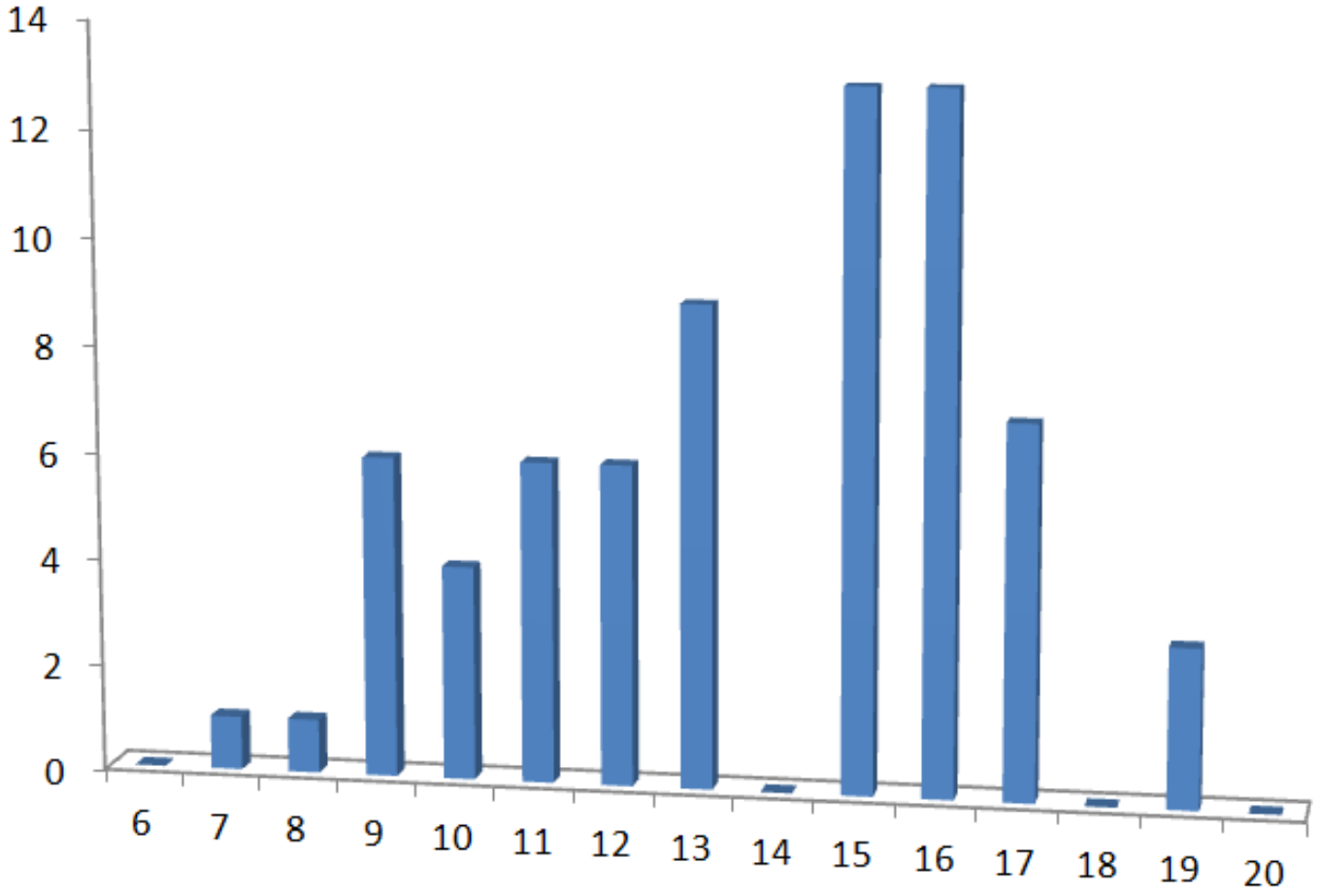
•0.5 Punkte pro gültigem Punkt, max. 2 Punkte

Kritik:

•private static int MAX_ALLOWED_EVOLUTIONS

- Ist keine Konstante, auch wenn sie wie eine aussieht (nicht final)
- Der Fehler ist nicht, dass im Code eine Konstante verändert wird (was nicht geht), sondern dass eine Nicht-Konstante wie eine Konstante deklariert wird

Übung 1 Resultate



Allgemeines

- Fragen/Feedback zu Übung 1?
- Fragen zur Gruppeneinteilung?