Funktionale und objektorientierte Programmierkonzepte
Zusatzblatt B (mit Lösungen)

TECHNISCHE UNIVERSITÄT DARMSTADT

Prof. Karsten Weihe

Ansprechpartner:	Svana Esche
Wintersemester 23/24	v2.0
Themen:	Generics
Relevante Foliensätze:	Folien 11-103 von 06, Folien 56-63 von 07
Abgabe der Umfrage:	12.01.2024 bis 23:50 Uhr

Zusatzblatt B — Gesamt: 4 Bonus-Bonus-Punkte

Fehlvorstellungen bei Generics

Wir haben eine eigene Studie für Sie entworfen, bei der wir untersuchen, welche Vorstellungen Sie zu Generics haben und was typische Fehler zu Generics sind.

Durch die Teilnahme an dieser Studie erhalten Sie eine Selbsteinschätzung über Ihre Fertigkeiten im Bereich der Generics. Zusätzlich geben Sie uns die Möglichkeit zu untersuchen, welche Art von Unterstützung für Ihr Lernen in der FOP noch notwendig ist.

Achtung: Die 4 Punkte werden anteilig vergeben, je nachdem, wie weit Sie die Aufgaben bearbeiten. Also, je weniger Aufgaben Sie bearbeiten, desto weniger Punkte erhalten Sie. Die vergebenen Punkte hängen *nicht* von der Korrektheit Ihrer bearbeiteten Aufgaben ab.

Zugang: Der Link zur Studie ist: `https://survey.ise.tu-darmstadt.de/FOP_generics/`

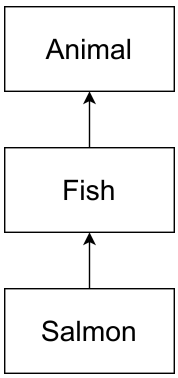
Lösungsvorschlag:

Vorstellung der Vererbungshierarchie

Angenommen, die Klassen Animal, Fish und Salmon[1] erben wie folgt voneinander:

```
1 public class Animal {...}
2 public class Fish extends Animal {...}
3 public class Salmon extends Fish {...}
```

Diese Vererbungshierarchie lässt sich graphisch darstellen als:



Sie finden diese graphische Darstellung zur Erinnerung auf den folgenden Seiten. Sie müssen sich diese nicht merken.

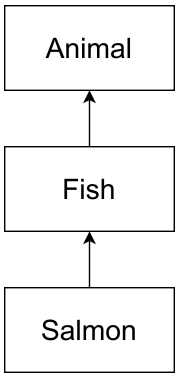
Zusätzlich gilt: Alle drei Klassen haben einen parameterlosen Konstruktor.

[1] In Deutsch sind das: Tier, Fisch und Lachs. Unter dem Link `https://de.wikipedia.org/wiki/Lachse` finden Sie eine Erklärung, was ein Lachs ist.

Aufgabe 0

Zur Erinnerung

Die Vererbungshierarchie sieht wie folgt aus:



Aufgabe 0

Betrachten Sie die folgende Code-Zeile:

```
1  Fish f = new __________ ();
```

Welche der genannten Typen können in der Lücke stehen, sodass der Code fehlerfrei kompiliert?

Sprich, welcher Code muss in der Lücke stehen, die durch _____ gekennzeichnet ist?

Schreiben Sie jeden Typ in eine eigene Zeile.

> *Fish*
> *Salmon*

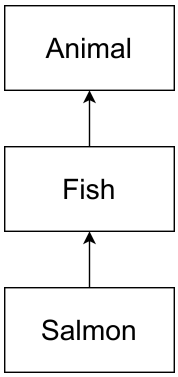
Warum genau diese und keinen anderen? Antworten Sie möglichst präzise.

> *Der dynamische Typ muss entweder gleich dem statischen Typ oder ein Subtyp des statischen Typs sein. Der dynamische Typ entspricht hier dem Code, der in der Lücke steht. Der statische Typ der Referenz `f` ist `Fish`.*
>
> *Damit kommen zwei Typen in Frage:*
> *1. `Fish`, in dem Fall, dass dynamischer und statischer Typ identisch sind.*
> *2. `Salmon`, da `Salmon` ein Subtyp von `Fish` ist.*
>
> *In der Vererbungshierarchie sind keine weiteren Subtypen von `Fish` aufgeführt. Daher gibt es keine weiteren Typen, die in Frage kommen.*

Aufgabe 1

Zur Erinnerung

Die Vererbungshierarchie sieht wie folgt aus:



Aufgabe 1

Betrachten Sie folgende Deklaration der Liste `s`.

```
1 List_____ s;
```

Wie muss der Typparameter der Liste `s` deklariert werden, damit die Liste `s` `Fish` als generischen Typparameter hat?

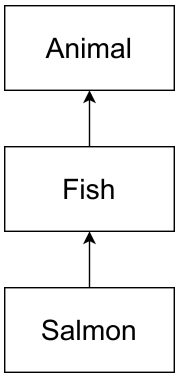
Sprich, welcher Code muss in der Lücke stehen, die durch _____ gekennzeichnet ist?

> `<Fish>`
>
> *Erklärung: Die Syntax um `Fish` als generischen Typparameter darzustellen ist `<Fish>`.*

Aufgabe 2

Zur Erinnerung

Die Vererbungshierarchie sieht wie folgt aus:



Aufgabe 2

Welche der genannten Typen können als Elemente zu einer Liste vom Typ `List<Fish>` hinzugefügt werden?

Schreiben Sie jeden Typ in eine eigene Zeile.

> *Fish*
> *Salmon*

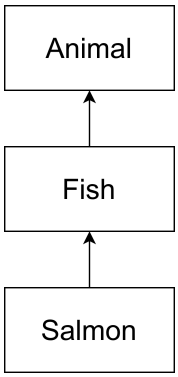
Warum genau diese und keinen anderen? Antworten Sie möglichst präzise.

> *In einer Liste mit generischem Typ `T` müssen die hinzugefügten Elemente von Typ `T` oder von einem Subtyp von `T` sein. Hier entspricht Typ `T` dem Typ `Fish`.*
>
> *Damit kommen zwei Typen in Frage:*
> *1. `Fish`, in dem Fall, dass das Element gleich dem generischen Typ der Liste ist.*
> *2. `Salmon`, da `Salmon` ein Subtyp von `Fish` ist.*
>
> *In der Vererbungshierarchie sind keine weiteren Subtypen von `Fish` aufgeführt. Daher gibt es keine weiteren Typen, die in Frage kommen.*

Aufgabe 3

Zur Erinnerung

Die Vererbungshierarchie sieht wie folgt aus:



Aufgabe 3

Betrachten Sie den folgenden Code:

```
1  public void f ( List<Fish> s ){
2      __________ value = s.get(0);
3  }
```

Das erste Element von `s` ist nicht null.

Welche der genannten Typen können in der Lücke stehen, sodass der Code fehlerfrei kompiliert?

Sprich, welcher Code muss in der Lücke stehen, die durch _____ gekennzeichnet ist?

Schreiben Sie jeden Typ in eine eigene Zeile.

> *Fish*
> *Animal*

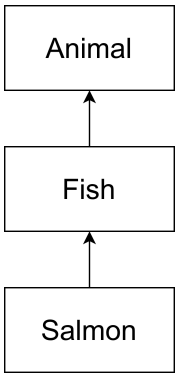
Warum genau diese und keinen anderen? Antworten Sie möglichst präzise.

> *Beim Auslesen der Liste `s` kann nur Typ `Fish` garantiert werden. Elemente vom Typ `Salmon` können zwar in der Liste `s` vorhanden sein. Es kann aber nicht garantiert werden, dass alle Elemente von Typ `Salmon` sind.*
>
> *Der Typ `Animal` ist auch möglich, das Elemente vom Typ `Fish` garantiert werden und jeder `Fish` auch ein `Animal` ist.*
>
> *Achtung: In dieser Aufgabe ist Auswahl der Typen auf die drei genannten Typen beschränkt. Wenn dies nicht der Fall ist, dann ergibt sich noch zusätzlich der Typ `Object`. Dies liegt daran, dass `Fish` ein Subtyp von `Object` ist, und daher als statischer Typ auch der Supertyp `Object` verwendet werden darf.*

Aufgabe 4

Zur Erinnerung

Die Vererbungshierarchie sieht wie folgt aus:



Aufgabe 4

Betrachten Sie den folgenden Methodenkopf der Methode `f`.

```
1 public void f ( List_____ s ){...}
```

Wie muss der Typparameter der Liste `s` deklariert werden, damit die Methode `f` sowohl `List<Fish>` als auch `List<Salmon>` akzeptiert?

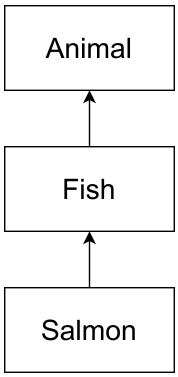
Sprich, welcher Code muss in der Lücke stehen, die durch _____ gekennzeichnet ist?

> *<? extends Fish>*
>
> *Erklärung: Eine Liste mit Aufbau `List<Fish>` ist genau zu `List<Fish>` kompatibel. Solch eine Liste ist jedoch nicht zu einer Liste mit Aufbau `List<Salmon>` kompatibel, auch wenn `Salmon` ein Subtyp von `Fish` ist.*
>
> *Daher wird hier eine Wildcard benötigt, die sowohl `Fish` als auch dessen Subtyp `Salmon` umfasst. Dies ist `<? extends Fish>`.*

Aufgabe 5

Zur Erinnerung

Die Vererbungshierarchie sieht wie folgt aus:



Aufgabe 5

Betrachten Sie den folgenden Code:

```
1  public void f ( List<? super Fish> s ){
2      __________ value = s.get(0);
3  }
```

Das erste Element von `s` ist nicht null. Welcher Typ oder welche Typen dürfen in der Lücke stehen, sodass der Code fehlerfrei kompiliert?

Sprich, welcher Code muss in der Lücke stehen, die durch _____ gekennzeichnet ist?

> *Object*

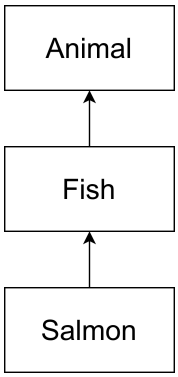
Warum genau diese und keinen anderen? Antworten Sie möglichst präzise.

> *Die Liste `s` mit generischem Typ `<? super Fish>` ist kompatibel zu jeder `List<T>`, solange `T` dem Typ `Fish` entspricht oder ein Supertyp von `Fish` ist. Im Fall unserer Vererbungshierarchie könnte also Liste `s` den Typ `List<Animal>` haben.*
>
> *Wir wissen aber nicht, welche konkrete Liste in der Liste `s` gespeichert ist. Damit können wir auch keinen Typ außer `Object` garantieren.*

Aufgabe 6

Zur Erinnerung

Die Vererbungshierarchie sieht wie folgt aus:



Aufgabe 6

Betrachten Sie den folgenden Code:

```
1 ArrayList<? extends Fish> a;
```

Welche Werte können wir `a` hinzufügen?

null

Warum genau diese und keinen anderen? Antworten Sie möglichst präzise.

Die `ArrayList a` mit generischem Typ `<? extends Fish>` ist kompatibel zu jeder `ArrayList<T>`, solange `T` dem Typ `Fish` entspricht oder ein Subtyp von `Fish` ist. Im Fall unserer Vererbungshierarchie könnte also `ArrayList a` den Typ `List<Salmon>` haben.

Wir wissen aber nicht, welche konkrete Liste in der `ArrayList a` gespeichert ist. Zudem garantiert der Compiler, dass die Funktionalität auch mit allen möglichen weiteren Subtypen von `Fish` kompatibel ist. So könnte es einen neuen Subtyp `Tuna`, also Thunfisch, von `Fish` geben.

Insgesamt kann nur `null` eingefügt werden, weil diese der kleinste gemeinsame Nenner alle Subklassen von `Fish` ist.