Klausurvorbereitungskurs zur FOP-Prüfung am 04.04.2023

Lösungsvorschläge zu allen Quizzen

Svana Esche

TECHNISCHE UNIVERSITÄT DARMSTADT

Version 6.2
23. März 2023

In diesem Dokument finden Sie alle Quizze, also von A bis Y, mit den jeweiligen Fragen/Aufgaben und passenden Lösungsvorschlägen.

Anmerkungen, Korrekturen und alternative Lösungen sind herzlich willkommen!
↪ Mail an `svana.esche@tu-darmstadt.de`

Änderungen zur vorherigen Version

Die Aufgabe 68 aus dem Quiz Q wurde entsprechend der Aufgabenstellung in Moodle angepasst mitsamt der Lösung.

Die erste Lösung zur Aufgabe 93 wurde in der zweiten Zeile angepasst.

Quiz A

Aufgabe 1: Zugriff auf Attribute

Der Zugriff auf ein Attribut, das im statischen Typ noch nicht definiert ist, ist nicht erlaubt, auch wenn der dynamische Typ dieses Attribut besitzt.

Warum ist das sinnvoll?

Lösungsvorschlag:

Abschnitt Begriffsbildung: Subtypen und statischer / dynamischer Typ aus dem Foliensatz 3b (Referenztypen)

Das ist sinnvoll, denn in unwesentlich komplexeren Situationen kann der Compiler schon nicht mehr bei der Übersetzung überprüfen, ob der dynamische Typ einer Referenz wirklich der gewünschte ist, also ob das Objekt, auf die Variable verweist, wirklich dieses Attribut hat.

Aufgabe 2: `doNothing`

Schreiben Sie eine `void`-Methode `doNothing`, die `public` ist und deren Implementation keine Anweisungen enthält. Die Methode `doNothing` besitzt einen Parameter `t` vom Typ `T`.

Lösungsvorschlag:

Abschnitt Die void-Methoden der Klasse aus dem Foliensatz 1e (Klassen und Methoden)

```
1 public void doNothing (T t){ }
```

Aufgabe 3: Array: Anzahl der Buchstaben

Schreiben Sie eine `public`-Klassenmethode `foobar`, die einen Parameter `a` vom Typ „Array von `char`“ und den Rückgabetyp `int` hat. Die Klasse, zu der `foobar` gehört, müssen Sie nicht schreiben. Ihre Methode `foobar` kann ohne Prüfung davon ausgehen, dass `a` tatsächlich auf ein Arrayobjekt verweist. Die Rückgabewert entspricht der Anzahl derjenigen Zeichen im Arrayobjekt, auf das `a` verweist, die Buchstaben sind.

Beispiel: $[?, a, =, B, \&, \%, c, D,] \rightarrow 4$

Lösungsvorschlag:

Abschnitt Begriffsbildung: Referenztypen aus dem Foliensatz 3b (Referenztypen)

```
public static int foobar ( char[] a ) {
        int numberOfLetters = 0;
        for ( int i = 0; i < a.length; i++ ){
                if ( Character.isLetter(a[i]) )
                        numberOfLetters++;
        }
        return numberOfLetters;
}
```

Aufgabe 4: Objekorientierung: Programmdesign

Nennen Sie die vier Kernelemente des objektorientierten Programmdesigns und die zwei Kernelemente der objektorientierten Abstraktion.

Lösungsvorschlag:

Abschnitt Rückschau: Objektorientierte Abstraktion aus dem Foliensatz 4a (Grundlagen)

Die Kernelemente des objektorientierten Programmdesigns sind:

1. Die zu erstellende Funktionalität wird in Klassen und Interfaces zerlegt.
2. Jede Klasse und jedes Interface repräsentiert ein Konzept, das typischerweise auch den Namen der Klasse beziehungsweise des Interface bildet. Umgekehrt wird jedes abstrakte Konzept als eine Klasse oder ein Interface realisiert.
3. Der Programmablauf besteht aus der Interaktion von Objekten.
4. Durch Vererbung kann ein Konzept ein anderes erweitern, verfeinern oder variieren. Objekte des abgeleiteten Konzeptes können an Stelle des Basiskonzepts verwendet werden.

Die zwei Kernelemente der objektorientierten Abstraktion sind:

1. Objekte/Klassen sind die zentralen Bausteine.
2. Ein Objekt hat einen momentanen Zustand.

Quiz B

Aufgabe 5: Interfaces: Sichtbarkeit von Methoden

Welche Sichtbarkeiten haben die Methoden eines Interfaces `X` und welche Auswirkungen hat das für die Klassen, die das Interface `X` implementieren?

Lösungsvorschlag:

Abschnitt Interfaces aus dem Foliensatz 3b (Referenztypen)

Alle Methoden im Interface `X` sind automatisch `public`, daher kann man das `public` bei den Methoden in einem Interface auch weglassen.

Daher müssen die Methoden bei der Implementation in den Klassen ebenfalls `public` sein.

Aufgabe 6: Finale Methoden

Betrachten Sie eine `public`-Klasse `X`, bei der nur eine `final`-Methode `foo` und keine anderen Methoden deklariert wurde. Die Klasse `Y` ist von Klasse `X` abgeleitet. Schreiben Sie die Klasse `Y` und eine Methode in `Y`, die nicht durch den Compiler gehen würde. Sie dürfen annehmen, dass der Rückgabetyp von `foo` `void` ist und die Implementierung von `foo` in Klasse `X` keine Anweisungen enthält.

Die Klasse `X` sowie die Methode foo in `X` müssen Sie nicht schreiben.

Lösungsvorschlag:

Abschnitt Finale Methoden aus dem Foliensatz 3c (Methoden)

```
public class Y extends X {
        void foo() {}
}
```

Aufgabe 7: Von Racket nach Java

Betrachten Sie folgende Funktion in Racket.

```
;; Type: integer -> integer
(define (a n)
        (cond
                [(= n 0) 1]
                [(= n 1) 1]
                [else (+ (* (a (- n 1)) n) (a (- n 2)))]
        )
)
```

Schreiben Sie eine Klassenmethode in Java, die die Racket-Funktion 1:1 realisiert.

Lösungsvorschlag:

Abschnitte Rekursion in Racket und Rekursion in Java aus dem Foliensatz 4a (Grundlagen)

Java:

```java
static int a (int n){
        if (n == 0 || n == 1)
                return 1;
        return a(n-1) * n + a(n-2);
}
```

Aufgabe 8: Packages: Namensprobleme

Wann treten Namensprobleme bei Packages auf und wie lassen sie sich lösen?

Lösungsvorschlag:

Abschnitt Packages und Standardbibliothek aus dem Foliensatz 3a (Grundlegendes)

Wenn verschiedene Inhalte mit dem gleichen Namen aus verschiedenen Packages importiert werden, kommt es zu einem Namenskonflikt. Es ist nicht klar, aus welchem Package die Funktionalität bspw. für die jeweilige Klasse benutzt werden soll.

Lösbar ist dies durch Qualifizierung des Namens: `package.inhalt` anstatt `inhalt`

Quiz C

Aufgabe 9: `super`

In den Subklassen steht der Konstruktor der Basisklasse nur mittels `super` zur Verfügung. Warum werden Konstruktoren einer Basisklasse nicht an die davon abgeleiteten Klassen vererbt?

Lösungsvorschlag:

Abschnitt Konstruktoren und Static Initializer aus dem Foliensatz 3c (Methoden)

In der Regel muss man davon ausgehen, dass für die abgeleitete Klasse eigene Initialisierungsschritte nötig sind, die aber im Konstruktor der Basisklasse fehlen. Fehlende Initialisierung ist eine Quelle für schwierig zu findende Fehler.

Aufgabe 10: Interfaces: Subtypen

Nennen Sie alle Subtypen eines Interfaces `I`.

Lösungsvorschlag:

Abschnitt Begriffsbildung: Subtypen und statischer / dynamischer Typ aus dem Foliensatz 3b (Referenztypen)

Alle direkt oder indirekt mit `extends` von `I` abgeleiteten Interfaces und alle `I` oder ein von `I` abgeleitetes Interface implementierenden Klassen und deren Subtypen

Aufgabe 11: Array: Median

Schreiben Sie eine `public`-Klassenmethode `foobar`, die einen Parameter `a` vom Typ „Array von `int`“ und den Rückgabetyp `double` hat. Die Klasse, zu der `foobar` gehört, müssen Sie nicht schreiben). Ihre Methode `foobar` kann ohne Prüfung davon ausgehen, dass `a` tatsächlich auf ein Arrayobjekt verweist und die Elemente in `a` sortiert sind.

Die Rückgabewert entspricht dem *Median*. Der *Median* kann auf folgende Weise bestimmt werden:

- Alle Werte werden (aufsteigend) geordnet.
- Wenn die Anzahl der Werte ungerade ist, ist die mittlere Zahl der Median.
- Wenn die Anzahl der Werte gerade ist, wird der Median als arithmetisches Mittel der beiden mittleren Zahlen definiert.

Beispiele:
$[1, 3, 3, 6, 7] \rightarrow 3$

$[1, 3, 3, 6, 6, 9] \rightarrow 0.5 \cdot (3 + 6)$ also 4.5

Lösungsvorschlag:

Abschnitt Begriffsbildung: Referenztypen aus dem Foliensatz 3b (Referenztypen)

```java
public static double foobar ( int[] a ) {

        if (a.length % 2 == 0 ){
                double sum = a[a.length/2] + a[a.length/2 - 1];
                return sum/2;
        }
        return a[a.length/2]; // oder a[(a.length - 1)/2];
}
```

Aufgabe 12: Scope

Beschreiben Sie, was unter dem Begriff Scope verstanden wird. Wo überall ist ein Attribut beziehungsweise eine Methode einer Klasse oder eines Interface sichtbar?

Lösungsvorschlag:

Abschnitt Scope von Definitionen von Identifiern aus dem Foliensatz 3a (Grundlegendes)

Der Scope einer Definition eines Identifiers ist der Bereich im Quelltext, in dem der Identifier gemäß dieser Definition angesprochen und verwendet werden kann.

Die Sichtbarkeit hängt von dem Modifier der Sichtbarkeit ab - also `public`, kein Modifier, `protected` oder `private`. Parameter und lokale Variablen einer Methode können genauso heißen wie Attribute der Klasse, zu der die Methode gehört. Die Attribute sind dann aber nicht mehr direkt sichtbar, sondern müssen mittels `this` angesprochen werden.

Quiz D

Aufgabe 13: `final`

Angenommen die Methode `foo` der Klasse `X` sei `final`. Was müssen Sie beim Ableiten von der Klasse `X` beachten?

Lösungsvorschlag:

Abschnitt Finale Methoden aus dem Foliensatz 3c (Java-Methoden)

Die Klasse darf abgeleitet werden, aber die Methode `foo` darf nicht überschrieben werden.

Aufgabe 14: Interfaces: Vererbung

Schreiben Sie ein `public`-Interface `A` mit einer Objektmethode `m1`, die Rückgabetyp `double`, einen `int`-Parameter `n` und einen `char`-Parameter `c` hat. Schreiben Sie ein `public`-Interface `B`, das von `A` erbt und zusätzlich eine Objektmethode `m2` hat, die keine Parameter hat und einen String zurückliefert.

Lösungsvorschlag:

Abschnitt Interfaces aus dem Foliensatz 3b (Referenztypen)

```
public interface A {
        double m1 ( int n, char c );
}

public interface B extends A {
        String m2 ();
}
```

Aufgabe 15: Packages

Packages bilden eine hierarchische Struktur in Java-Programmen ab. Nennen Sie mindestens drei Vorteile, die sich durch die Verwendung von Packages ergeben.

Lösungsvorschlag:

Abschnitt Packages und Standardbibliothek aus dem Foliensatz 3a (Grundlegendes)

1. Ein wichtiger Vorteil ist, dass man nützliche Funktionalität durch inhaltliche Gruppierung leichter auffinden kann. Durch die Gruppierung nach Sachthemen kann man sich auf der Suche nach Funktionalität auf ein oder wenige Packages beschränken, in denen die Funktionalität gemäß thematischer Aufteilung zu finden sein müsste.
2. Packages erlauben Zugriffsbeschränkungen - wie durch den Modifier `private` bei Klassen - auch auf der Ebene mehrerer inhaltlich aufeinander bezogener Klassen.
3. Durch Packages sind Namenskonflikte kontrollierbar, da durch die Ergänzung des Bezeichners des Packages die Entitäten eindeutig zuordbar sind.

Aufgabe 16: Interfaces: Vererbung

Schreiben Sie eine abstrakte `public`-Klasse `YZ`, die von `XY` erbt und sowohl das Interface `Y` als auch das Interface `java.util.Comparator<Integer>` implementiert.

Hierbei erbt das Interface `Y` vom Interface `X` und die `public`-Klasse `XY` implementiert `X`.

Lösungsvorschlag:

Abschnitte Abstrakte Klassen, Interfaces aus dem Foliensatz 3b (Referenztypen)

```
abstract public class YZ extends XY implements Y, Comparator<Integer> { }
// oder java.util.Comparator...
```

Quiz E

Aufgabe 17: Methode ohne Implementation

Betrachten Sie die folgenden Fälle: (1) eine Methode `doX` ohne Implementation und (2) eine Methode `doY` deren Implementation keine Anweisungen enthält. Beide Methoden sind parameterlos. Schreiben Sie eine Klasse `Class` und die darin enthaltenen Methoden `doX` und `doY`.

Lösungsvorschlag:

Abschnitt Abstrakte Klassen aus dem Foliensatz 3b (Referenztypen)

```
1 abstract public class Class {
2       abstract public void doX();
3       public void doY () { };
4 }
```

Eine Methode ohne Implementation muss als `abstract` deklariert werden. Eine Klasse, die eine `abstract`-Methode beinhaltet, muss als `abstract` deklariert werden.

Aufgabe 18: `this`

Wofür wird das Schlüsselwort `this` üblicherweise innerhalb eines Konstruktors verwendet, also zusätzlich zu der Ihnen bekannten allgemeinen Verwendung von `this` in Methoden? Warum ist nicht sinnvoll auf `this` in diesem Fall zu verzichten?

Lösungsvorschlag:

Abschnitt Anweisungen: Schlüsselwörter this und super aus dem Foliensatz 3c (Methoden)

Man kann in einem Konstruktor einer Klasse einen anderen Konstruktor derselben Klasse aufrufen: einfach Schlüsselwort `this` gefolgt von den Parametern des anderen Konstruktors.

Das ist beispielsweise sinnvoll, wenn zwei Konstruktoren eigentlich dasselbe tun sollen, aber der eine Konstruktor hat einen Parameter weniger als der andere, weil er den Parameterwert selbst festlegt und nicht durch den Nutzer festlegen lässt.

Aufgabe 19: Closure

Was wird unter dem Begriff Closure verstanden?

Lösungsvorschlag:

Abschnitt Functional Interfaces und Lambda-Ausdrücke in Java aus dem Foliensatz 4c (Funktionen als Daten)

Informationen aus dem Entstehungskontext des Lambda-Ausdrucks werden mitgespeichert und bei Verwendung des Lambda-Ausdrucks mitverwendet.

Aufgabe 20: Statischer, dynamischer Typ

Welche Konsequenzen hat die Wahl des statischen bzw. dynamischen Typs bei der Implementierung?

Lösungsvorschlag:

Abschnitt Begriffsbildung: Subtypen und statischer / dynamischer Typ aus dem Foliensatz 3b (Referenztypen)

Auf ein Attribut oder eine Methode kann genau dann über eine Referenz zugegriffen werden, wenn das Attribut beziehungsweise die Methode schon im statischen Typ vorhanden ist – dort definiert oder ererbt. Welche *Implementation* der Methode aufgerufen wird, das hängt von der Information im Objekt, also vom dynamischen Typ ab.

Quiz F

Aufgabe 21: Interface und Klassen: Anzahl

Wie viele Interfaces darf eine Klasse implementieren und von wie vielen Klassen darf eine Klasse abgeleitet sein? Beide Teile der Frage beziehen sich auf Java.

Lösungsvorschlag:

Abschnitt Interfaces aus dem Foliensatz 3b (Referenztypen)

Eine Klasse darf beliebig viele Interfaces implementieren, auch gar kein Interface. Eine Klasse darf von maximal einer anderen Klasse abgeleitet sein.

Aufgabe 22: Klassenattribute

Was ist der Unterschied zwischen Attributen von Klassen und Klassenattributen?

Lösungsvorschlag:

Abschnitt Klassenattribute aus dem Foliensatz 3b (Referenztypen)

Steht bei der Deklaration eines Attributs das Schlüsselwort static vor dem Typnamen, dann wird für dieses Attribut nicht in jedem Objekt eine Kopie des Attributs angelegt, sondern nur eine einzige Speicherstelle für die ganze Klasse. Der Inhalt der Speicherstelle kann geändert werden und ist nicht statisch.

Die normalen Attribute ohne static sind Attribute von Klassen, die mit static nennt man Klassenattribute.

Klassenattribute lassen sich ohne Deklaration einer Variable ansprechen, sondern über `Klassenname.Attributname`.

Aufgabe 23: Statischer, dynamischer Typ

Sind die folgenden Zeilen korrekt?

```
1 Object o = new String();
2
3 String s = new Object();
```

Gehen Sie bei Ihrer Beurteilung auf die Begriffe statischer und dynamischer Typ ein.

Lösungsvorschlag:

Abschnitt Begriffsbildung: Subtypen und statischer / dynamischer Typ aus dem Foliensatz 3b (Referenztypen)

Es gilt: der dynamische Typ muss immer entweder gleich dem statischen Typ oder ein Subtyp des statischen Typs sein.

Zeile 1 ist korrekt, da dort der dynamische Typ `String` ist, welches ein Subtyp von `Object` ist.

Zeile 3 ist nicht korrekt, da dort der dynamische Typ `Object` kein Subtyp von `String` ist.

Aufgabe 24: Funktionen höherer Ordnung

1. Was versteht man unter Funktionen höherer Ordnung?
2. Geben Sie zwei verschiedene Funktionen höherer Ordnung an, die bereits in Racket eingebaut sind.
3. Was macht in Java ein Interface zu einem Functional Interface?

Lösungsvorschlag:

Abschnitte Begriffsbildung: Funktionen höherer Ordnung, Fallbeispiel filter in Racket, Fallbeispiel map in Racket, Fallbeispiel fold in Racket, Functional Interfaces und Lambda-Ausdrücke in Java aus dem Foliensatz 4c (Funktionales Programmieren: Funktionen als Daten)

1. Funktionen, die Funktionen als Parameter oder Rückgabe haben, sind Funktionen höherer Ordnung.
2. Funktionen höherer Ordnung, die , die bereits in Racket eingebaut sind, sind: `map`, `filter` und `fold`
3. Ein Functional Interface ist ein Interface, bei dem genau eine Methode weder `static` noch `default` ist.

Quiz G

Aufgabe 25: Sichtbarkeit: `protected`

Erläutern Sie den Zusammenhang zwischen dem Schlüsselwort `protected` und dem Konzept der Vererbung der Objektorientierung

Lösungsvorschlag:

Abschnitt Vererbung von Attributen aus dem Foliensatz 3b (Referenztypen)

Auf Attribute, die `protected` deklariert sind, kann in einer abgeleiteten Klasse ebenfalls zugegriffen werden. Somit müssen die Klassen in einer Vererbungshierarchie zueinander stehen, genau gesagt, die Subklassen dürfen auf die Attribute der Superklassen zugreifen.

Erinnerung: Bei `protected` haben zusätzlich auch alle Klassen im selben Package Zugriff.

Aufgabe 26: Fehlerkorrektur

Korrigieren Sie alle Fehler im folgenden Code und erklären Sie, warum es sich um Fehler gehandelt hat.

```
1 public class X {
2         public void m1() { return 0;}
3         abstract public void m2();
4         abstract public void m3(int n){n++};
5 }
```

Lösungsvorschlag:

Abschnitt Abstrakte Klassen aus dem Foliensatz 3b (Referenztypen)

Wenn eine Klasse eine abstrakte Methode enthält, muss die Klasse selbst als `abstract` deklariert werden. Dieses Schlüsselwort fehlt in Zeile 1.

Zeile 2 ist nicht korrekt, da eine Methode mit Rückgabetyp `void` nichts zurückgeben darf. Mittels `return 0;` wird jedoch der Wert `0` zurück gegeben.

Wenn eine Methode als `abstract` deklariert wurde, dann darf die Methode keine Implementation besitzen. Dies ist Zeile 4 jedoch der Fall, da die Implementation von `m3` aus dem Statement `n++;` besteht.

Aufgabe 27: `main()`-Methode

Erklären Sie, warum die Bestandteile void, static und public in der Signatur der `main()`-Methode notwendig sind.

Erinnerung: Die Signatur ist `public static void main(String[] args)`

Lösungsvorschlag:

Abschnitt Kopf einer Methode aus dem Foliensatz 3c (Methoden)

Der Bestandteil `public` garantiert, dass die Methode `main` frei außerhalb der Klasse, zu der sie gehört, in einer beliebigen anderen Klasse aufgerufen werden kann.

Der Bestandteil `static` besagt, dass `main` eine Klassenmethode ist und somit kein Objekt der zugehörigen Klasse erstellt werden muss, um die Methode aufzurufen. Es existiert ja auch als Einstiegspunkt noch gar kein Objekt, auf welchem die main-Methode aufgerufen werden könnte.

Der Bestandteil `void` besagt, dass `main` nichts zurückgibt. Das ist sinnvoll, da es nichts gibt, an was ein eventuller Rückgabetyp zurückgegeben werden könnte.

Aufgabe 28: Überladen und Überschreiben

Kann eine Klasse zwei Methoden mit identischer Signatur haben? Gehen Sie bei Ihrer Antwort auf die Begriffe Überladen und Überschreiben ein.

Lösungsvorschlag:

Abschnitt Signatur und Überschreiben / Überladen von Methoden aus dem Foliensatz 3c (Methoden)

Eine Klasse kann keine zwei Methoden mit derselben Signatur haben, denn beim Überschreiben geht die überschriebene Methode verloren, und beim Überladen müssen sich die Signaturen unterscheiden.

Quiz H

Aufgabe 29: Funktionales Programmdesign

Was sind die zentralen Bausteine im funktionalen Programmieren? Nennen Sie die zwei Kernelemente des funktionalen Programmdesigns.

Lösungsvorschlag:

Abschnitt Funktionale Abstraktion aus dem Foliensatz 4a (Grundlagen)

Funktionen sind die zentralen Bausteine.

Kernelemente:

1. Die zu erstellende Funktionalität wird in Funktionen zerlegt.
2. Die Ausführung eines Programms wird verstanden als Interaktion von Funktionen in dem Sinne, dass eine Funktion die andere aufruft.

Aufgabe 30: `assert`

Schreiben Sie folgenden Code-Abschnitt in eine entsprechende `assert`-Anweisung um:

```
1 if ( n < 1000 || n > 1000)  throw new AssertionError ( "Bad n!" );
```

Lösungsvorschlag:

Abschnitt Throwable, Error und Assert-Anweisung aus dem Foliensatz 5 (Fehlerbehandlung)

```
1 assert n == 1000 : “Bad n“!;
```

Aufgabe 31: Garbage Collector

Auf welches Problem reagiert der Garbage Collector und wie funktioniert er?

Lösungsvorschlag:

Abschnitt Garbage Collector aus dem Foliensatz 3b (Referenztypen)

Das Problem ist, dass für die Objekte, die vom Programm aus nicht mehr angesprochen werden können und daher absolut nutzlos geworden sind, immer noch deren Speicherplatz reserviert ist. Der Garbage Collector gibt alle Speicherplätze frei, die vom Programm aus nicht mehr erreichbar sind, und sorgt so dafür, dass aufgrund dieses Problems nicht zuwenig Speicherplatz zur Verfügung steht.

Aufgabe 32: Parameter: formal und aktual

Beschreiben Sie knapp und allgemein, was unter formalen und aktualen Parametern verstanden wird.

Lösungsvorschlag:

Abschnitt Begriffsbildung: formale vs. aktuale Parameter aus dem Foliensatz 3c (Methoden)

Was bei der Methodendefinition in der Parameterliste steht, das sind die formalen Parameter.

Was hingegen beim Methodenaufruf in der Parameterliste steht, das sind die aktualen Parameter.

Quiz I

Aufgabe 33: Runtime Exceptions

Inwiefern unterscheiden sich Runtime Exceptions von anderen Exceptions? Welche Konsequenzen würden sich ergeben, wenn Runtime Exceptions wie andere Exceptions behandelt werden würden?

Lösungsvorschlag:

Abschnitt Runtime Exceptions aus dem Foliensatz 5 (Fehlerbehandlung)

Exceptions vom Typ RuntimeException beziehungsweise von davon direkt oder indirekt abgeleiteten Klassen müssen nicht gefangen werden. Das ist eine Ausnahme zu allen anderen Exceptions.

Wenn Runtime Exceptions wie andere Exceptions behandelt werden würden, dann müsste jeder Zugriff auf ein Attribut oder ein Methodenaufruf in einen try-catch-Block, da diese eventuell eine NullPointerException werfen könnten. NullPointerException ist eine Subklasse von RuntimeException. Dies würde den Code aber völlig unleserlich machen.

Aufgabe 34: Racket: lambda

Betrachten Sie folgende lambda-Funktion in Racket:

```
(lambda (a b) (+ (* a a) (* b b) ) )
```

Schreiben Sie den äquivalenten lambda-Ausdruck in Java. Hierbei dürfen Sie annehmen, dass es sich bei den Eingabeparametern um ganze Zahlen handelt und der berechnete Wert zurück gegeben wird.

Lösungsvorschlag:

Abschnitte Spezialisierungen von generischen Funktionen in Racket und Functional Interfaces und Lambda-Ausdrücke in Java aus dem Foliensatz 4c (Funktionen als Daten)

```
(int a, int b) -> {return a*a + b*b;} // Langform

(a, b) -> a*a + b*b // Kurzform
```

Aufgabe 35: Comparator

Die Methode `compare` von `Comparator<Integer>` hat bekanntlich zwei Parameter vom Typ `Integer` und liefert `int` zurück. Sie sollen die Methode `compare` in der folgenden Version implementieren:

Die Methode soll

`+1` zurückliefern, wenn der `int`-Wert im ersten Parameter um mindestens 5 höher als der `int`-Wert im zweiten Parameter ist,

`-1` , wenn es genau umgekehrt ist, und

`0` sonst.

Lösungsvorschlag:

Abschnitt Comparator aus dem Foliensatz 6 (Generics)

```
public int compare ( Integer i1, Integer i2 ) {
        if ( i1 - i2 >= 5 )
                return 1; // oder: return +1;
        if ( i2 - i1 >= 5 )
                return -1;
        return 0; // oder: else return 0;
}
```

Aufgabe 36: Einschränkung von Typparametern

Was ist die Motivation dafür, dass Typparameter in Java eingeschränkt werden können?

Was gehört in das Klammerpaar `< >`, wenn Sie den Typ auf alle Klassen, die von Klasse `X` abgeleitet sind, vereinigt mit der Klasse `X` selbst, einschränken wollen?

Lösungsvorschlag:

Abschnitt Eingeschränkte Typparameter, Wildcards aus dem Foliensatz 6 (Generics)

Zur Motivation: Wenn der Typparameter `T` garantiert entweder gleich einer Klasse `X` oder abgeleitet von Klasse `X` ist, dann ist absolut sicher, dass `T` nur Typen annehmen kann, in denen bspw. die Methoden entsprechend definiert worden sind.

Um die Einschränkung bei einer Klasse, einem Interface oder einer Methode zu implementieren, sollte `<T extends X>` gewählt werden.

Um bei einer Methode eine solche Einschränkung zu formulieren, die aber gleichzeitig mit verschiedenen Instanziierungen von `X` umgehen kann, sollte `<? extends X>` gewählt werden. Wildcards schränken nur die Definition bei der Instanziierung von Typparametern, d.h. bei der Methodensignatur wäre `<? extends X>` nicht möglich. In Collections wiederum sind solche Wildcards möglich, bspw. `List<? extends X>`.

Quiz J

Aufgabe 37: Interface: Predicate

Schreiben Sie eine generische Klasse `Class`, die `public` ist und das Interface `Predicate` aus der Java-Standardbibliothek implementiert. Hierbei sollen `Class` und `Predicate` mit dem selben Typ `T` instanziiert werden können.

Die Methode `test` des Interface `Predicate` brauchen Sie nicht zu schreiben.

Lösungsvorschlag:

Abschnitt Generische Klassen aus dem Foliensatz 6 (Generics), Abschnitt Interface aus dem Foliensatz 3b (Referenztypen)

```
public class Class <T> implements Predicate <T> {
}
```

Aufgabe 38: Iterator: keine Buchstaben

Eine Liste `listOfChars` vom Typ `List<Character>` ist gegeben. Sie sollen mittels eines Iterators eine Liste `listOfNonLetters` vom Typ `LinkedList<Character>` erstellen. Die Liste `listOfNonLetters` soll keine Buchstaben von `listOfChars` enthalten. Die Reihenfolge der Elemente muss jedoch erhalten bleiben.

Beispiel: $(a, \$, D, E, 1, !) \rightarrow (\$, 1, !)$

Lösungsvorschlag:

Abschnitte Iterator, Collection aus dem Foliensatz 7 (Collections)

```
Iterator<Character> it = listOfChars.iterator();
LinkedList<Character> listOfNonLetters = new LinkedList<Character>();

while (it.hasNext()){
        Character c = it.next();
        if(! Character.isLetter(c)){
                listONonfLetters.add(c);
        }
}
```

Aufgabe 39: Java: lambda

Betrachten Sie folgenden Code-Abschnitt:

```
public class X {
        public static double applyFct( IntToDoubleFunction fct, int n )
        { … }
}
```

Ihre Aufgabe ist es nun eine Variable `area` vom Typ `double` zu erstellen und mit dem Wert zu initialisieren, der entsteht, wenn auf `X` die Methode `applyFct` aufgerufen wird. Hierbei soll `fct` die Berechnung einer Kreisfläche mit simulieren, also die Berechnung mittels der Formel $\pi \cdot r^2$ ausführen. Zudem wird `applyFct` mit dem aktualen Parameterwert 6 für `n` aufgerufen. Hierbei dürfen Sie die Variable `pi` als Wert von π verwenden.

Lösungsvorschlag:

Abschnitt Functional Interfaces und Lambda-Ausdrücke in Java aus dem Foliensatz 4c (Funktionen als Daten)

```
double area = X.applyFct(r -> r*r*pi, 6);
```

Aufgabe 40: Exceptions: `throws`

Die Klasse `X` besitzt eine Methode `m`, die Exceptions vom Typ `Ex1` wirft. Die Klasse `Y` erbt von `X` und überschreibt `m`. Welche Exception-Klassen dürfen in der `throws`-Klausel in `Y.m` stehen?

Lösungsvorschlag:

Abschnitt Methoden mit throws-Klauseln überschreiben aus dem Foliensatz 5 (Fehlerbehandlung)

In der überschreibenden Methode `m` in Klasse `Y` darf die Exception-Klasse `Ex1` ersetzt werden durch eine direkt oder indirekt abgeleitete Exception-Klasse.

Die überschreibenende Methode `m` in Klasse `Y` muss aber auch überhaupt keine Exception deklarieren.

Quiz K

Aufgabe 41: Interfaces vs. abstrakte Klassen

Nennen Sie die Unterschiede zwischen Interfaces und abstrakten Klassen.

Lösungsvorschlag:

Abschnitt Functional Interfaces und Lambda-Ausdrücke in Java aus dem Foliensatz 4c (Funktionen als Daten)

Interfaces können Mehrfachvererbung, abstrakte Klasse nicht. Abstrakte Klassen können Attribute und Methoden haben, die nicht public sind; Interfaces nicht. Genauer gesagt, Interfaces können nur Klassenkonstanten haben.

Aufgabe 42: Exceptions: Zweck

Nennen Sie den Zweck von Exceptions.

Lösungsvorschlag:

Abschnitt Exceptions aus dem Foliensatz 5 (Fehlerbehandlung)

Der Zweck von Exceptions ist dafür sorgen, dass die Abarbeitung eines Programms auch in unerwarteten, unerwünschten Fällen nicht vom Laufzeitsystem beendet wird, sondern stattdessen eine selbstgeschriebene Fehlerbehandlung durchgeführt wird und das Programm danach ordnungsgemäß weiterläuft.

Aufgabe 43: Racket: mul

Betrachten Sie folgende rekursive Funktion in Racket, die eine Liste von Zahlen entgegennimmt.

```
(define (mul lst)
        (cond
                [(empty? lst) 1 ]
                [else (* (first lst) (mul (rest lst)))]
        )
)
```

Schreiben Sie in Java eine generische `public`-Objektmethode `foo` mit generischem Typparameter `T`, die einen Parameter vom formalen Typ `ListItem<T>` hat, `lst`, und einen Parameter `t` vom Typ `T`, und die nichts zurückliefert.

Die Funktionalität der Methode `foo` soll der Funktionalität der Funktion `mul` entsprechen.

Zur Erinnerung:

```
public ListItem<T> {
        T key;
        ListItem<T> next;
}
```

Sie dürfen annehmen, dass Sie mit dem Datentyp `T` multiplizieren und ganze Zahlen zuweisen können.

Ihre Methode `foo` darf davon ausgehen, dass der Parameter `lst` auf den Kopf einer korrekt gebildeteten Liste verweist (die auch leer sein darf, also aktualer Wert `null` ist möglich). Die Werte, die von `foo` aus `lst` übernommen werden sollen, werden mittels in `t` geschrieben.

Verbindliche Anforderung: Schleifen sind nicht erlaubt. Die Liste, auf die `lst` verweist, darf nicht verändert werden.

Lösungsvorschlag:

Abschnitt Rekursive Funktionen auf Listen aus dem Foliensatz 4b (Listen), und Abschnitt Eigene LinkedList-Klasse aus dem Foliensatz 7 (Collections)

```
public <T> void foo ( ListItem<T> lst, T t ) {
        if (lst == null)
                return ;
        if (lst != null){
                t = t * lst.key;
                lst = lst.next;
        }
        foo (lst, t);
}
```

Falls die Aufgabenstellung nicht fordert, dass die Methode nichts zurückliefert, dann könnte die Lösung so aussehen:

```
public <T> T foo (ListItem<T> lst) {
        if (lst == null)
                return 1;
        return lst.key * foo(lst.next);
}
```

Aufgabe 44: Map

Welche Eigenschaften müssen *Keys* und *Values* des Interface `Map` erfüllen?

Lösungsvorschlag:

Abschnitt Maps aus dem Foliensatz 7 (Collections)

Die Keys müssen alle unterschiedlich, also einzigartig sein und zu jedem Key wird genau ein Value zugeordnet.

Quiz L

Aufgabe 45: Referentielle Transparenz

Was ist mit referentieller Transparenz gemeint? Geben Sie zusätzlich, was referentielle Transparenz bei einem Methodenaufruf in Java bedeuten würde.

Lösungsvorschlag:

Abschnitte Was ist nun typisch funktional? und Aufweichung der reinen funktionalen Lehre in Racket aus dem Foliensatz 4d (Diskussion)

Referentielle Transparenz meint: Ein Ausdruck muss überall, wo er vorkommt, denselben Wert haben, denn ohne Zeit kann der Wert eines Ausdrucks auch keinen zeitabhängigen Wert haben.

Referentielle Transparenz in Java würde fordern, dass jeder Aufruf einer Methode mit demselben aktualen Parameter zum selben Ergebnis führen würde. Das ist in Java offenkundig nicht vorgesehen.

Aufgabe 46: Generics

Welche Unterschiede ergeben sich, wenn Sie Variante a) oder b) verwenden?

```
1 // Variante a)
2 public <T extends Character> double m ( X<T> n, T m ) { ... }
3
4 // Variante b)
5 public double m ( X<? extends Character> n, Character m ) { ... }
```

Lösungsvorschlag:

Abschnitt Wildcards aus dem Foliensatz 6 (Generics)

In Variante a) ist die gesamte Methode generisch und zwar auf `Character` und Suptypen von `Character`. In Variante b) ist nur der erste Parameter generisch, nicht die gesamte Methode.

Aufgabe 47: Allgemeiner Zugriff auf Elemente

Welches weitere, allgemeine Vorgehen gibt es neben dem Erzeugen eines Iterator, um den Zugriff auf Elemente von wahlweise Arrays bzw. Listen zu ermöglichen?

Erstellen Sie für die Liste `list` und das Array `a` das entsprechende Vorgehen und weisen Sie die entstehende Konstrukte jeweils der Referenz `b` zu.

Lösungsvorschlag:

Abschnitt Streams aus dem Foliensatz 8 (Streams und Files)

Streams erlauben einen allgemeinen Zugriff auf Elemente von sortierten Sequenzen und damit auch von Arrays bzw. Listen.

```
b = list.stream();

b = Arrays.stream(a);
```

Aufgabe 48: Array vs. Liste

Betrachten Sie die Konzepte Array und Liste in Java. Nennen Sie die wesentlichen Unterschiede.

Lösungsvorschlag:

Abschnitt Eigene LinkedList-Klasse aus dem Foliensatz 7 (Collections), Foliensatz 3b (Referenztypen), Foliensatz 1d (Arrays)

Arrays haben eine beliebige, aber feste Größe. Daher lässt sich die Größe eines Arrays nach dem Instanziieren nicht mehr ändern.

Listen hingegen sind von variabler Größe, die sich zur Laufzeit ändern kann.

Bei Arrays gibt es wahlfreien Zugriff. Jedes Element kann direkt über den zugehörigen Index angesprochen werden.

Beim Interface `List` kommt es auf die jeweiligen Instanzen der Listen an. Bei einer `LinkedList` muss sich durch die gesamte Liste „durchgehangelt“werden, um zum gesuchten Element zu gelangen. Bei einer `ArrayList` hingegen ist ebenfalls ein wahlfreier Zugriff möglich.

Quiz M

Aufgabe 49: Typisierung

Beschreiben Sie knapp und allgemein, inwiefern sich Racket und Java bezüglich der Typisierung unterscheiden.

Lösungsvorschlag:

Abschnitt Was ist nun typisch funktional? Aus dem Abschnitt 4d (Diskussion)

In Racket wird erst zur Laufzeit geprüft ob die Typen der Operanden zum Operator passen bzw. die Parameter einer vordefinierten Funktion zur Funktion passen. Dies wird unter dem Begriff *dynamische Typisierung* gefasst. Die Typen sind bei der Implementation nicht hinzuschreiben.

In Java prüft der Compiler die Passung von Typen - mit Ausnahme des Downcast. Dies wird unter dem Begriff *statische Typisierung* gefasst. Der Typ ist überall zu nennen.

Aufgabe 50: Interface und Generics

Schreiben Sie ein `public`-Interface `Inf1`, welches generisch mit einem Typparameter `T` ist, der auf `Number` und die von `Number` abgeleiteten Klassen beschränkt ist.

Lösungsvorschlag:

Abschnitt Eingeschränkte Typparameter aus dem Foliensatz 6 (Generics) und Abschnitt Interfaces aus dem Foliensatz 3b (Referenztypen)

```
1 public interface Inf1<T extends Number>{ }
```

Aufgabe 51: `ListItem<T>`

Wie seht der Schleifenkopf aus, wenn bei einer Liste vom Typ `ListItem<T>` alle Elemente der Liste durchlaufen werden sollen?

Lösungsvorschlag:

Abschnitt Eigene LinkedList-Klasse aus dem Foliensatz 7 (Collections)

```
1 for ( ListItem<T> p = head; p != null; p = p.next )
```

Aufgabe 52: Exceptions: abfangen

Die Methode `m` wirft Exceptions vom Typ `Exception1` und `Exception2`, hat einen Parameter `a` vom Typ „Array von `int`“ und einen Parameter `index` vom Typ `int` und den Rückgabetyp `int`.

Wo müssen die Exceptions der Methode `m` abgefangen werden?

Lösungsvorschlag:

Abschnitte Exceptions werfen und fangen, Exceptions weiterreichen statt fangen aus dem Foliensatz 5 (Fehlerbehandlung)

Alle Methoden, die Methode `m` aufrufen, müssen die Exceptions der Methode `m` entweder abfangen oder weiterreichen, und zwar an diejenigen Methoden, von denen sie selbst aufgerufen wurden.

In dieser Aufrufkette müssen an irgendeiner Stelle die Exceptions gefangen werden.

Quiz N

Aufgabe 53: Streams

Welchen Zweck haben Streams aus dem Package `java.util.stream`?

Lösungsvorschlag:

Abschnitt Streams aus dem Foliensatz 8 (Streams und Files)

Streams bilden in gewisser Weise eine einheitliche Schnittstelle für Listen, Arrays, Dateien sowie endlichen und unendlichen Sequenzen von Werten des Typparameters.

Aufgabe 54: Array: abrunden

Schreiben Sie eine `public`-Klassenmethode `foobar`, die einen Parameter `a` vom Typ „Array von `double`“ und den Rückgabetyp `b` vom Typ „Array von `int`“ hat. Die Klasse, zu der `foobar` gehört, müssen Sie nicht schreiben. Ihre Methode `foobar` kann ohne Prüfung davon ausgehen, dass `a` tatsächlich auf ein Arrayobjekt verweist. Die Länge des zurückgelieferten Arrays soll gleich der Länge des Arrays des aktualen Parameters sein. Im zurückgelieferten Array sollen alle Werte des Arrayobjekts des aktualen Parameters auf die nächste ganze Zahl abgerundet werden.

Beispiel: $[10.9, 10.0, 9.1, 9.5] \rightarrow [10, 10, 9, 9]$

Lösungsvorschlag:

Abschnitt Begriffsbildung: Referenztypen aus dem Foliensatz 3b (Referenztypen) und Abschnitt Konversionen aus dem Foliensatz 1b (Grundlagen)

```
public static int[] foobar ( double[] a ) {
        int[] b = new int[a.length];
        for ( int i = 0; i < a.length; i++ ){
                b[i] = (int) a[i];
        }
        return b;
}
```

Aufgabe 55: Exceptions: message

Schreiben Sie eine `public`-Klasse `Exc1`, die direkt von `Exception` abgeleitet ist. Klasse `Exc1` soll einen `public`-Konstruktor mit einem Parameter `obj` vom formalen Typ `Object` haben. Die Botschaft (message) des `Exc1`-Objektes soll eine `String`-Repräsentation des aktualen Parameters sein.

Lösungsvorschlag:

Abschnitt Exceptions aus dem Foliensatz 5 (Fehlerbehandlung)

```
public class Exc1 extends Exception {
        public Exc1 ( Object obj ) {
                super (obj.toString());
        }
}
```

Aufgabe 56: Funktionale Methode in Java

Beschreiben Sie in Worten, was unter einer funktionalen Methode in Java verstanden wird.

Schreiben Sie zusätzlich ein Stück Java-Code, bei dem eine funktionale Methode auftaucht und markieren Sie diese mittels einem Kommentar.

Lösungsvorschlag:

Abschnitt Functional Interfaces und Lambda-Ausdrücke in Java aus dem Foliensatz 4c (Funktionales Programmieren: Funktionen als Daten)

Eine funktionale Methode ist eine Methode, die Teil eines Interfaces ist und weder `static` noch `default` als Modifier hat.

```
public interface IntToDoubleFunction {
        double applyAsDouble ( int n ); // functional method
}
```

Quiz O

Aufgabe 57: `Predicate`

Welche Eigenschaften hat das Interface `Predicate` und warum ist es nicht sinnvoll, `Predicate` als Klasse zu definieren?

Lösungsvorschlag:

Abschnitt Functional Interfaces und Lambda-Ausdrücke in Java aus dem Foliensatz 4c (Funktionen als Daten), Abschnitt Interfaces aus dem Foliensatz 3b (Referenztypen)

Es ist ein Functional Interface und kann demnach benutzt werden, um lambda-Ausdrücke in Java abzubilden. Allgemein ist ein Prädikat eine boolesche Funktion, also eine Funktion, die entweder true oder false zurückliefert.

Predicate wäre als Klasse nicht sinnvoll, da auf diese Weise die Möglichkeiten zur mehrfachen Implementation eingeschränkt werden. So ist es möglich, dass Klassen mehrere Interfaces implementieren, aber nicht mehrere Klassen.

Aufgabe 58: Exceptions: werfen

Die Klassen-Methode `m` wirft Exceptions vom Typ `Exception1` und `Exception2`, hat einen Parameter `a` vom Typ „Array von `int`“ und einen Parameter `index` vom Typ `int` und den Rückgabetyp `int`.

Die Methode `m` wirft eine Exception vom Typ `Exception1`, falls der Wert des letzten Elements von `a` mit dem Wert von `index` übereinstimmt. Zusätzlich wird eine Exception vom Typ `Exception2`, falls der Wert von `index` kleiner als 0 ist. Anschließend wird `index` zurückgegeben.

Schreiben Sie die Methode `m` mit den zuvor beschriebenen Eigenschaften.

Lösungsvorschlag:

Abschnitte Exceptions werfen und fangen, Mehrere Exception-Klassen aus dem Foliensatz 5 (Fehlerbehandlung)

```
public static int m (int[] a, int index) throws Exception1, Exception2 {
        if (index == a[a.length - 1]){
                throw new Exception1();
        }
        if (index < 0){
                throw new Exception2();
        }
        return index;
}
```

Aufgabe 59: `HashMap`

Schreiben Sie eine `public`-Methode `foo` mit formalen Parameter `map` vom Typ `HashMap`. Der Schlüsselwerttyp von `map` ist `T`, der Informationstyp ist generisch und beschränkt auf Klasse `Frame` sowie alle Referenztypen, von denen Frame ein `Subtyp` ist. Der Rückgabetyp der Methode `foo` ist `void`. Die Implementierung von `foo` hat keine Anweisungen.

Lösungsvorschlag:

Abschnitte Maps und Wildcards auf Collections und Listen aus dem Foliensatz 7 (Collections)

```
1 public void foo (HashMap <T, ? super Frame> map) {}
```

Aufgabe 60: Effectively Final

Was versteht man darunter, dass eine Variable *effectively final* ist und wie hängt dies mit lambda-Ausdrücken zusammen?

Lösungsvorschlag:

Abschnitt Functional Interfaces und Lambda-Ausdrücke in Java Foliensatz 4c (Funktionen als Daten)

Eine Variable, die in einem lambda-Ausdruck vorkommt, muss *effectively final* sein, also wird deren Wert ab diesem Vorkommen nicht noch einmal geändert. Dies ist notwendig, da durch den Closure der Entstehungskontext des lambda-Ausdrucks und damit auch der aktuale Wert der Variable gespeichert wird.

Quiz P

Aufgabe 61: Interface passend zu Predicate

Schreiben Sie ein Interface `DoublePredicate`, welches eine passende funtionale Methode `test` hat und die Funktionalität eines Prädikats entsprechend abbildet.

Lösungsvorschlag:

Abschnitt Functional Interfaces und Lambda-Ausdrücke in Java aus dem Foliensatz 4c (Funktionen als Daten)

Vorneweg, in Mathematik und Informatik ist ein Prädikat eine boolesche Funktion, also eine, die nur wahr oder falsch zurückgibt.

```
public interface DoublePredicate {
        boolean test (Double d);
}
```

Aufgabe 62: Generics: Vererbung

Schreiben Sie ein generisches `public`-Interface `A` mit Typparameter `T`, der auf `X` und die Subtypen von `X` beschränkt ist.

Schreiben Sie ein `public`-Interface `B`, das von `A` erbt, wobei `T` mit `Y` instanziiert ist.

Lösungsvorschlag:

Abschnitt Generische Klassen aus dem Foliensatz 6 (Generics), Abschnitt Interface aus dem Foliensatz 3b (Referenztypen)

```
public interface A <T extends X> {}

public interface B extends A<Y> {}
```

Aufgabe 63: Exceptions: fangen

Es gibt eine `public`-Klasse `A` mit einer `public`-Klassenmethode `km`, die einen `int`-Parameter `n` hat, `double` zurückliefert und potentiell `Exception` wirft. Diese müssen Sie nicht schreiben.

Schreiben Sie eine `public`-Klasse `B` mit einer `public`-Objektmethode `m`, die einen `int`-Parameter `n` hat, `double` zurückliefert. Diese Methode ruft `km` von `A` mit `n` auf, ohne ein Objekt von `A` dafür einzurichten, und liefert das Ergebnis von `km` zurück. Sollte `km` eine `Exception` werfen, dann soll stattdessen `0` zurückgeliefert werden.

Lösungsvorschlag:

Abschnitt Exceptions werfen und fangen aus dem Foliensatz 5 (Fehlerbehandlung)

```
public class B {
        public double m (int n) {
                try {
                        return A.km(n);
                }
                catch (Exception e){
                        return 0;
                }

        }
}
```

Aufgabe 64: Rekursion: Terminierung

Beschreiben Sie knapp und allgemein, was unter Rekursion verstanden wird. Gehen Sie dabei besonders darauf ein, wie sichergestellt wird, dass die Subroutine terminiert.

Lösungsvorschlag:

Thematische Zusammenfassung Rekursion, Abschnitte Rekursion in Java, Rekursion in Racket aus dem Foliensatz 4a (Grundlagen)

Rekursion bedeutet, dass die zugehörige Subroutine sich an irgendeiner Stelle in der Implementation mindestens einmal selbst aufruft.

Damit die Rekursion terminiert, muss die Subroutine eine Abbruchbedingung enthalten, und die Parameterwerte müssen bei jedem Rekursionsschritt näher an die Abbruchbedingung herankommen.

Quiz Q

Aufgabe 65: Interface IntPredicate

Initialisieren Sie eine Variable `pred` des Interface `IntPredicate` mit der Methode, die die Division durch 5 mit Rest 1 modelliert.

Lösungsvorschlag:

Abschnitt Functional Interfaces und Lambda-Ausdrücke in Java aus dem Foliensatz 4c (Funktionen als Daten)

```
IntPredicate pred = n -> n % 5 == 1;
```

Aufgabe 66: Java: lambda

Betrachten Sie die folgenden Code-Abschnitte, wobei unter `pi` eine Näherung der mathematischen Konstante π verstanden wird.

```
public class X {
        public static double applyFct( IntToDoubleFunction fct, int n )
                { … }
}
```

```
(define (make-circle-area r)
        (* pi r r)
)
```

Ihre Aufgabe ist es nun eine Variable `area` vom Typ `double` zu erstellen und mit dem Wert zu initialisieren, der entsteht, wenn auf `X` die Methode `applyFct` aufgerufen wird. Hierbei soll `fct` die Berechnung der Funktion `make-circle-area` simulieren. Zudem wird `applyFct` mit dem aktualen Parameterwert 6 für `n` aufgerufen. Hierbei dürfen Sie die Variable `pi` als Wert von π verwenden.

Lösungsvorschlag:

Abschnitte Functional Interfaces und Lambda-Ausdrücke in Java und Spezialisierungen von generischen Funktionen in Racket aus dem Foliensatz 4c (Funktionen als Daten)

```
double area = X.applyFct(r -> pi*r*r, 6);
```

Aufgabe 67: Iterator: Buchstaben

Eine Liste `listOfChars` vom Typ `List<Character>` ist gegeben. Sie sollen mittels eines Iterators eine Liste `listOfLetters` vom Typ `LinkedList<Character>` erstellen. Die Liste `listOfLetters` soll nur die Buchstaben von `listOfChars` enthalten. Die Reihenfolge der Elemente muss jedoch erhalten bleiben.

Beispiel: $(a, \$, D, E, 1, !) \rightarrow (a, D, E)$

Lösungsvorschlag:

Abschnitte Iterator, Collection aus dem Foliensatz 7 (Collections)

```
Iterator<Character> it = listOfChars.iterator();
LinkedList<Character> listOfLetters = new LinkedList<Character>();

while (it.hasNext()){
        Character possibleLetter = it.next();
        if(Character.isLetter(possibleLetter)){
                listOfLetters.add(possibleLetter);
        }
}
```

Aufgabe 68: Darstellungsinvariante, Implementationsinvariante

Beschreiben Sie knapp und allgemein, was unter den Begriffen Darstellungsinvariante und Implementationsinvariante verstanden wird.

Lösungsvorschlag:

Abschnitt Korrektheit von Klassen aus dem Foliensatz 12 (Korrekte Software)

Die Darstellungsinvariante beschreibt, wie Objekte der Klasse sich dem Nutzer der Klasse darstellen sollen. Konkreter formuliert heißt das: die Sicht, die die Attribute und Methoden vermitteln, die `public` definiert sind.

Die Implementationsinvariante ist analog zur Darstellungsinvariante, aber behandelt eben den Teil der Klassendefinition, der nicht `public` ist.

Quiz R

Aufgabe 69: Programmzeiger

Unter welchen Umständen springt der Programmzeiger und geht damit nicht zur nächsten auszuführenden Instruktion weiter?

Lösungsvorschlag:

Thematische Zusammenfassung Programme und Prozesse

Bei unbedingten Sprunginstruktionen springt der Programmzeiger auf die darin kodierte Zieladresse. Bei bedingten springt der Programmzeiger nut unter der Bedingung, dass ein boolescher Test zu wahr ausgewertet wurde. Diese Arten von Sprüngen lassen sich bei Verzweigungen (`if`) und Schleifen betrachten.

Zudem kommt es zu Sprüngen bei dem Aufruf von Subroutinen.

Aufgabe 70: Einlesen von Daten

Wann ist byteweises Einlesen und wann ist zeichenweises Einlesen von Daten sinnvoll?

Lösungsvorschlag:

Abschnitt Textdaten direkt (ohne Streams) aus dem Foliensatz 8 (Streams und Files)

Für Bytedaten wie etwa Bilddateien oder Audio- oder Video-Dateien ist byteweises Einlesen sinnvoll, für Textdateien hingegen zeichenweises Einlesen.

Aufgabe 71: Statischer, dynamischer Typ

Unter welche Bedingungen geht die folgende Code-Zeile durch den Compiler?

```
1 A a = new B();
```

Gehen Sie bei Ihrer Antwort auf die Begriffe statischer und dynamischer Typ ein.

Lösungsvorschlag:

Abschnitt Begriffsbildung: Subtypen und statischer / dynamischer Typ aus dem Foliensatz 3b (Referenztypen)

Die Code-Zeile geht genau dann durch den Compiler, wenn `B` ein Subtyp von `A` ist. Dies gilt, weil hier die Referenz `a` den statischen Typ `A` und den dynamischen Typ `B` hat. Der dynamische Typ einer Referenz muss immer gleich dem statischen Typ oder ein Subtyp des statischen Typs sein.

Aufgabe 72: Iterator

Es gibt eine Referenz `c` vom Typ `ArrayList<Number>`, auf diese können Sie über den Identifizier `c` zugreifen.

Ihre Aufgabe ist es, einen Iterator `it` mit generischem Typ `Number` auf `c` anzulegen. Durchlaufen Sie mittels dem Iterator die Liste, die in `c` hinterlegt ist, und addieren Sie alle Werte auf, die in der Liste hinterlegt sind.

Erinnerung: Die Klasse `Number` besitzt die parameterlose Methode `doubleValue`, um den Wert des entsprechenden Objekts als `double` zurückzugeben.

Lösungsvorschlag:

Abschnitte Iterator, Collection aus dem Foliensatz 7 (Collections)

```
Iterator<Number> it = c.iterator();
double sum = 0;
while (it.hasNext()){
        sum += it.next().doubleValue();
}
```

Quiz S

Aufgabe 73: Downcast in Java

Inwiefern stellt der Downcast eine Ausnahme der statischen Typisierung in Java dar?

Geben Sie ein Beispiel an, wie Sie manuell zur Laufzeit den Typ der Variable `p` auf `Person` überprüfen können.

Lösungsvorschlag:

Abschnitt Was ist nun typisch funktional? aus dem Foliensatz 4d (Diskussion)

Bei der statischen Typisierung in Java prüft der Compiler die Passung der Typen. Bei einem Downcast kann der Typ jedoch nicht bereits zur Compilezeit überprüft werden, sondern erst zur Laufzeit. Dadurch stellt der Downcast eine Ausnahme der statischen Typisierung in Java dar.

```
if (p instanceof Person){...}
```

Aufgabe 74: Umkehrung bei Streams?

Wir können aus beliebigen, geordneten Sequenzen Streams erzeugen, wie bspw. bei Arrays.

Ist es möglich diesen Vorgang umzukehren? Wenn ja, formulieren Sie den zugehörigen Java-Code um einen Array `s` vom Typ `String` aus dem Stream `stream` zu erstellen.

Lösungsvorschlag:

Abschnitt Streams aus dem Foliensatz 8 (Streams und Files)

Ja, es ist möglich.

```
String [] s = stream.toArray ( String[]::new );
```

Aufgabe 75: Liste

In Java und in Racket existiert das Konzept der Liste. Beschreiben Sie knapp und allgemein, inwiefern sich das Konzept in den Programmiersprachen unterscheidet.

Lösungsvorschlag:

Thematische Zusammenfassung Listen

Konzeptionell sind Listen in Racket heterogen; die Datentypen der einzelnen Elemente werden nicht geprüft. Konzeptionell sind in Java Listen homogen; der generische Typparameter ist der Datentyp.

Das Durchlaufen einer Liste geschieht in Racket rekursiv, in Java meist iterativ mittels Iteratoren oder Schleifen.

Aufgabe 76: Beispiele für Fehler

Wann werden folgende Fehlerarten in Racket bzw. in Java entdeckt?

```
// A: Zugriff auf leere Elemente

(first empty)

String s = "abc";
s.charAt(3);

// B: Typfehler

Integer i = new Double(2.71);
```

Lösungsvorschlag:

Abschnitt (In)korrektheit auf den einzelnen Abstraktionsebenen aus dem Foliensatz 12 (Korrekte Software)

In Racket wird der Programmiercode interpretiert und damit direkt ausgeführt, somit werden die Fehler in Racket erst zum Programmablauf entdeckt.

In Java werden semantische Fehler, wie der Zugriff auf leere Elemente erst während des Programmablaufs entdeckt, indem sie eine RuntimeException werfen.

Syntaktische Fehler wie Typfehler werden bereits zur Compilezeit und damit vor dem eigentlichen Programmablauf entdeckt.

Quiz T

Aufgabe 77: Typinferenz

Was wird unter dem Begriff *Typinferenz* verstanden und wie taucht dieser Aspekt in Java auf?

Lösungsvorschlag:

Abschnitt Was ist nun typisch funktional? aus dem Foliensatz 4d (Diskussion)

Unter Typinferenz versteht man, dass die Programmiersprache automatisiert die Typen von Variablen und Konstanten aus dem Kontext heraus bestimmen kann. Falls die Typen nicht zueinander und zu dem Kontext passen, dann gibt es eine Fehlermeldung.

Dieser Aspekt taucht in Java bei den Lambda-Ausdrücken bei der Kurzform auf. Bei der Kurzform eines lamba-Ausdrucks mit nur einem formalen Parameter ist bspw. das Aufführen des Typs nicht notwendig und kann aus dem Kontext herausbestimmt werden. Dies entspricht gerade der Definition von Typinferenz.

Aufgabe 78: Exceptions: fangen und weiterreichen

Die `public`-Methode `w` ruft in ihrer Implementation die Methode `m` auf, die Exceptions vom Typ `Exception1` und `Exception2` wirft. Die Methode `w` fängt Exceptions vom Typ `Exception1` selbst und reicht alle anderen Exceptions weiter. Sie dürfen annehmen, dass Methode `w` einen Parameter `a` vom Typ „Array von `int`“ hat und `void` als Rückgabetyp.

Schreiben Sie die Methode `w`, wobei Sie deren Implementation mit `. . .` angeben.

Lösungsvorschlag:

Abschnitte Exceptions weiterreichen statt fangen aus dem Foliensatz 5 (Fehlerbehandlung)

```
public void w (int[] a) throws Exception2 {
        ...
}
```

Aufgabe 79: Generics: generische Parameter

Schreiben Sie eine `public`-Methode `n`, deren Implementierung leer und deren Rückgabetyp `int` ist. Die Methode `n` hat zwei Parameter `a` und `b`, wobei `b` vom Typ `int` ist. Die Methode `n` ist nicht generisch, aber deren Parameter `a` ist generisch. So hat `a` den Typ `X`, wobei hier `X` auf die Klasse `Character` und alle Subtypen von `Character` eingeschränkt ist.

Lösungsvorschlag:

Abschnitt Wildcards aus dem Foliensatz 6 (Generics)

```
public int n ( X<? extends Character> a, int b ) { }
```

Aufgabe 80: GUI

Nehmen Sie Stellung zu den folgenden Aussagen:

1. Für jede Art von `Listener` gibt es eine eigene Registrierungsmethode.
2. Die Klasse `Canvas` bietet eine unbegrenzte Zeichenfläche in einem Fenster.

Lösungsvorschlag:

Abschnitte Andere Typen von Komponenten in einem Fenster, Weitere Listener- und Event-Typen aus dem Foliensatz 10 (GUIs)

Beides Aussagen sind wahr. Zu 2. lässt sich ergänzen, dass dieses nur innerhalb des Fenstern angezeigt wird.

Quiz U

Aufgabe 81: Kontextfreiheit

Was versteht man in der Informatik unter dem Begriff Kontextfreiheit?

Lösungsvorschlag:

Abschnitt (In)korrektheit auf den einzelnen Abstraktionsebenen aus dem Foliensatz 12 (Korrekte Software)

Kontextfreiheit umfasst die Regeln, die rein lokal sind. Das heißt, Zusammenhänge zwischen verschiedenen Teilen des Quelltextes – die evtl. viele Zeilen auseinanderliegen könnten, mit vielen damit nicht in Zusammenhang stehenden anderen syntaktischen Konstrukten dazwischen – diese Zusammenhänge werden ignoriert.

Aufgabe 82: Klasse System in java.io

Nennen Sie die drei Klassenattribute der Klasse `System` in `java.lang` und erläutern Sie jeweils deren Aufgabe.

Lösungsvorschlag:

Abschnitt Bytedaten direkt (ohne Streams) aus dem Foliensatz 8 (Streams und Files)

Es gibt die Klassenattribute `out`, `err` und `in`. Die zwei zuerst genannten sind für das Schreiben zuständig und das letzte zum Lesen der getippten Zeichen von der Tastatur. Nach Konvention wird bei `out` die Ausgaben des Programms auf den Bildschirm geschrieben und bei `err` die Fehlermeldung, also bspw. die Messages von geworfenen Exceptions.

Aufgabe 83: Comparator

Betrachten Sie die Klasse `A` mit folgender Implementierung:

```
public class A {
        public String s;
        public int n;
}
```

Schreiben Sie nun eine Klasse `AComparator`, die das Interface `Comparator` mit generischem Typ `A` implementiert. Die Klasse `AComparator` besitzt eine Methode `compare`, die Objekte der Klasse `A` wie folgt vergleicht. Hierbei hat `compare` als Parameter `a1` und `a2`, beide vom Typ `A`.

- es wird `0` zurückgegeben, falls die Strings `s` von a`a1` und `a2` die gleiche Länge haben,
- es wird `+1` zurückgegeben, falls der String `s` von `a1` länger als der von `a2` ist,
- es wird `-1` zurückgegeben, falls der String `s` von `a1` kürzer als der von `a2` ist.

Lösungsvorschlag:

Abschnitt Comparator aus dem Foliensatz 6 (Generics)

```
public class AComparator implements Comparator <A> {
        public int compare ( A a1, A a2 ) {
                if ( a1.s.length() != a2.s.length() ){
                        if ( a1.s.length() < a2.s.length() )
                        return -1;
                        return +1; // auch möglich: else return +1;
                }
                return 0;
        }
}
```

Aufgabe 84: Java: lambda Part 2

Betrachten Sie die folgenden Code-Abschnitte, wobei unter `sqrt` die Quadratwurzel verstanden wird.

```
public class X {
        public static double applyFct( BinaryDoubleToDoubleFunction fct, double x,
           double y) { … }
}
```

```
(define (euclidean-distance-to-origin x y)
        (sqrt (+ (* x x) (* y y)))
)
```

Ihre Aufgabe ist es nun eine Variable `distance` vom Typ `double` zu erstellen und mit dem Wert zu initialisieren, der entsteht, wenn auf `X` die Methode `applyFct` aufgerufen wird. Hierbei soll `fct` die Berechnung der Funktion `euclidean-distance-to-origin` simulieren. Zudem wird `applyFct` mit den aktualen Parameterwerten 3.0 für `x` und 4.0 für `y` aufgerufen. Hierbei dürfen Sie `Math.sqrt()` für die Berechnung der Quadratwurzel in Java verwenden.

Lösungsvorschlag:

Abschnitte Functional Interfaces und Lambda-Ausdrücke in Java und Spezialisierungen von generischen Funktionen in Racket aus dem Foliensatz 4c (Funktionen als Daten)

```
double distance = X.applyFct( (x, y) -> Math.sqrt(x*x + y*y), 3.0 , 4.0);
```

Quiz V

Aufgabe 85: Fehlerbehandlung

Beschreiben Sie knapp und allgemein, wie die Klassen `Throwable`, `Error` und `Exception` zusammenhängen und was deren Zweck ist.

Lösungsvorschlag:

Abschnitt Throwable, Error und Assert-Anweisung aus dem Foliensatz 5 (Fehlerbehandlung)

Die Klasse `Exception` ist direkt abgeleitet von einer Klasse `Throwable`, und von `Throwable` ist noch eine Klasse `Error` abgeleitet. Alle drei sind im Package `java.lang` .

Klassen, die von `Error` direkt oder indirekt abgeleitet sind, werden typischerweise vom Laufzeitsystem geworfen in Fällen, in denen der Fehler nach menschlichem Ermessen so gewichtig ist, dass wohl keine sinnvolle Fehlerbehandlung mehr möglich ist, sondern Programmabbruch die beste Lösung zu sein scheint.

Die Funktionalität des Fangen und Werfen werden in Klasse `Throwable` allgemein behandelt.

Aufgabe 86: Generischer Parameter

Schreiben Sie eine `public`-Methode `n`, deren Implementierung leer und deren Rückgabetyp `int` ist. Die Methode `n` hat zwei Parameter `a` und `b`, wobei **`b`** vom Typ `int` ist.

Die Methode `n` ist nicht generisch, aber deren Parameter `a` ist generisch. So hat `a` den Typ `X`, wobei hier `X` auf die Klasse `Character` und alle (in)direkten Basisklassen und alle (in)direkt implementierten Interfaces eingeschränkt ist.

Lösungsvorschlag:

Abschnitt Wildcards aus dem Foliensatz 6 (Generics)

```
1 public int n ( X<? super Character> a, int b ) { }
```

Aufgabe 87: Array: abrunden (rekursiv)

Schreiben Sie zwei `public`-Klassen-Methoden in Java mit Namen `fillArray` und `foo`. Die Methode `fillArray` soll rein rekursiv sein, das heißt es dürfen keine Schleifen vorkommen. Die Methode `foo` hat einen Parameter `a` vom Typ „Array von `double`“ und den Rückgabetyp `b` vom Typ „Array von `int`“. Die Methode `foo` soll alle Werte des Arrayobjekts des aktualen Parameters `a` auf die nächste ganze Zahl abrunden und diese mittels `b` zurückgeben.

Ihre Methode `foo` kann ohne Prüfung davon ausgehen, dass `a` tatsächlich auf ein Arrayobjekt verweist; dieses kann aber Länge 0 haben.

Die Methode `fillArray` hat Eingabeparameter `a` vom Typ „Array von `double`“ und `b` vom Typ „Array von `int`“ und einen Eingabeparameter `index` vom Typ `int`. Die Methode `fillArray` soll für die Setzung der Komponenten des Arrays zuständig sein.

Beispiel: $[10.9, 10.0, 9.1, 9.5] \rightarrow [10, 10, 9, 9]$

Lösungsvorschlag:

Abschnitt Begriffsbildung: Referenztypen aus dem Foliensatz 3b (Referenztypen) und Abschnitt Konversionen aus dem Foliensatz 1b (Grundlagen)

```java
public static void fillArray (double[] a, int[] b, int index){
        if (a.length == index)
                return;
        b[index] = (int) a[index];
        fillArray(a, b, index+1);
}

public static int[] foo (double[] a){
        int[] b = new int [a.length];
        fillArray(a, b, 0);
        return b;
}
```

Aufgabe 88: Beispiele für Fehler

Geben Sie für ein konkretes Beispiel in Java an, bei dem ein Fehler auf der lexikalischen Ebene auftritt.

Geben Sie zusätzlich zwei konkrete verschiedene Beispiele von syntaktischen Fehlern in Java an.

Lösungsvorschlag:

Abschnitt (In)korrektheit auf den einzelnen Abstraktionsebenen aus dem Foliensatz 12 (Korrekte Software)

```java
whil ( t>0) { // lexikalischer Fehler durch falsche Schreibweise von while
        ...
}

for (int i = 0, i < 100; i++){ // syntaktischer Fehler durch , statt ;
        ...
}

if (m != null] // syntaktischer Fehler durch falsche Klammersetzung
        ...
```

Quiz W

Aufgabe 89: `List`

Was ist der Unterschied zwischen den Methoden `add` und `set` der Klasse `List`?

Lösungsvorschlag:

Erster Abschnitt aus dem Foliensatz 7 (Collections)

Diese Methode `add` ist identisch zur Methode `set`, mit einer Ausnahme: Methode `set` überschreibt das Element am gegebenen Index mit dem neuen Wert, wohin gegen Methode `add` das neue Element an diesem Index einfügt, ohne ein anderes Element zu entfernen. Das neue Element wird also unmittelbar vor dem Element eingefügt, das bisher am angegebenen Index stand. Dieses und alle weiteren Elemente mit höherem Index rutschen entsprechend um 1 in ihrem Index weiter.

Aufgabe 90: Darstellungsinvariante und Implementationsinvariante

Beschreiben Sie knapp und allgemein, was unter den Begriffen Darstellungsinvariante und Implementationsinvariante verstanden wird.

Lösungsvorschlag:

Abschnitt Korrektheit von Klassen aus dem Foliensatz 12 (Korrekte Software)

Die Darstellungsinvariante beschreibt, wie Objekte der Klasse sich dem Nutzer der Klasse darstellen sollen. Konkreter formuliert heißt das: die Sicht, die die Attribute und Methoden vermitteln, die `public` definiert sind.

Die Implementationsinvariante ist analog zur Darstellungsinvariante, aber behandelt eben den Teil der Klassendefinition, der nicht `public` ist.

Aufgabe 91: lambda: Java

Betrachten Sie eine `public`-Klassenmethode namens `foo` aus der Klasse `X`. Die Methode `foo` hat den Rückgabetyp `int` und drei Parameter:

(1) `bar` als ein Functional Interface `BinaryIntFct` mit funktionaler Methode `apply`, die zwei Parameter vom Typ `int` und Rückgabetyp `int` hat;

(2) `m` vom Typ `int`,

(3) `n` vom Typ `int`.

Rufen Sie die Methode `foo` mit einem lambda-Ausdruck für `bar` auf, der die Multiplikation zweier Zahlen realisiert, sowie mit den Werten 135 und 246 für `m` und `n`.

Lösungsvorschlag:

Abschnitt Functional Interfaces und Lambda-Ausdrücke in Java aus dem Foliensatz 4c (Funktionen als Daten)

```
1 X. foo ( ( int a, int b ) -> ( a * b ), 135, 246 );
```

Aufgabe 92: Comparator

Eigentlich alle Klassen in der Standardbibliothek, bei denen Vergleiche Sinn ergeben, implementieren ein Interface `Comparable` und haben dadurch eine Methode `compareTo`. Warum soll daneben noch ein separates Interface `Comparator` implementiert werden?

Lösungsvorschlag:

Abschnitt Comparator aus dem Foliensatz 6 (Generics)

Interface `Comparator` erlaubt es, den Vergleich der Elemente eines Referenztyps auf verschiedene Art zu definieren, so dass in jedem Kontext die passende Definition der Reihenfolge in Form einer spezifischen `Comparator` implementierenden Klasse verwendet werden kann.

Quiz X

Aufgabe 93: Array: invertieren

Schreiben Sie eine `public`-Klassenmethode `foobar`, die einen Parameter `a` vom Typ ”Array von int“ und den Rückgabetyp `b` vom Typ „Array von `int`“ hat. Die Klasse, zu der `foobar` gehört, müssen Sie nicht schreiben. Ihre Methode `foobar` kann ohne Prüfung davon ausgehen, dass `a` tatsächlich auf ein Arrayobjekt verweist.

Die Länge des zurückgelieferten Arrays soll gleich der Länge des Arrays des aktualen Parameters sein. Im zurückgelieferten Array sollen alle Werte des Arrayobjekts des aktualen Parameters in umgekehrter Reihenfolge wie zuvor erscheinen.

Beispiel: $[10, 4, 6, 7, 3, 11] \rightarrow [11, 3, 7, 6, 4, 10]$

Achtung: Sie dürfen die Methode `ArrayUtils.reverse()` **nicht** benutzen.

Lösungsvorschlag:

Abschnitt Begriffsbildung: Referenztypen aus dem Foliensatz 3b (Referenztypen)

```
public static int[] foobar ( int[] a ) {
        int[] b = a;
        for ( int i = 0; i < a.length/2; i++ ){
                int currentValueOfB = b[i];
                b[i] = a[a.length - i - 1];
                a[a.length - i - 1] = currentValueOfB;
        }
        return b;
}
```

Alternative Lösung:

```
public static int[] foobar ( int[] a ) {
        int[] b = new int[a.length];
        for ( int i = 0; i < a.length; i++ ){
                b[i] = a[a.length - i - 1];
        }
        return b;
}
```

Aufgabe 94: HashMap

Initialisieren Sie einen Zusammenhang mit Identifier `studi` vom Typ `HashMap`, der einer Matrikelnummer in Form einer natürlichen Zahl den vollständigen Namen einer Person zuordnet. Hierbei müssen Sie sich die geeigneten Datentypen für Matrikelnummer und Namen selbst überlegen.

Fügen Sie anschließend sich selbst als Person in dieses Objekt ein.

Lösungsvorschlag:

Abschnitt Maps aus dem Foliensatz 7 (Collections)

```
HashMap <Integer, String> studi = new HashMap <Integer, String>();

studi.put(1234567, "Vorname Nachname");
```

Alternative Lösung:

```
HashMap <Integer, String> studi = new HashMap <Integer, String>();

studi.put(new Integer(1234567), new String("Vorname Nachname"));
```

Aufgabe 95: Bytebasierter Zugriff

Das Ziel eines Outputstreams bei den bytebasierten Zugriffsmöglichkeiten heißt Datensenke. Wann ist ein solcher bytebasierter Zugriff sinnvoll?

Lösungsvorschlag:

Abschnitt Bytedaten direkt (ohne Streams) aus dem Foliensatz 8 (Streams und Files)

Byteweise Zugriff ist sinnvoll, wenn eine Datei nicht einen Text, sondern binäre Daten enthält, zum Beispiel ein Bild, eine Audio- oder eine Video-Datei.

Aufgabe 96: Generics: Laufzeit vs. Compile-Zeit

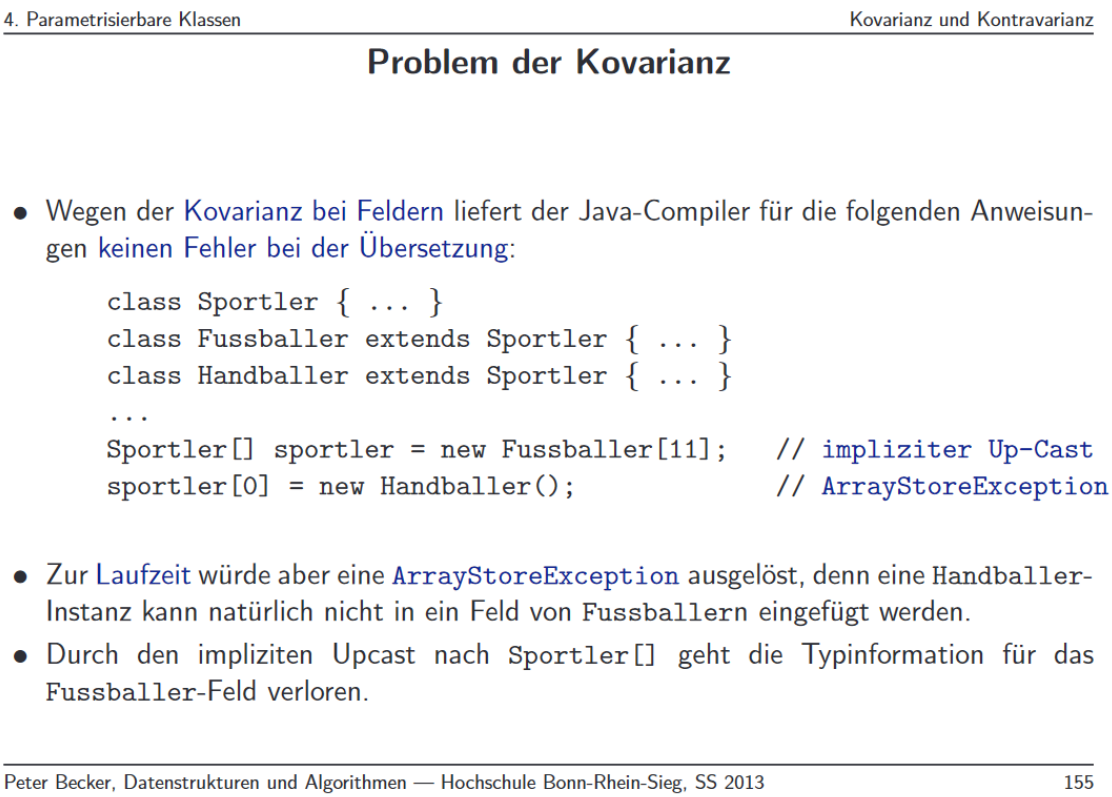
Werden die Typparameter bei Generics zur Laufzeitzeit oder zur Compile-Zeit festgelegt? Können dennoch Laufzeitfehler auftreten?

Lösungsvorschlag:

Abschnitt Generische Klassen aus dem Foliensatz 6 (Generics)

Die Typparameter werden bereits zur Compile-Zeit festgelegt. Fall es zur Fehlern kommen sollte mit Generics, dann werden diese bereits vom Compiler festgestellt. Also kann es keine Fehler zur Laufzeit geben.

Ausführliches Beispiel:



4. Parametrisierbare Klassen — Kovarianz und Kontravarianz

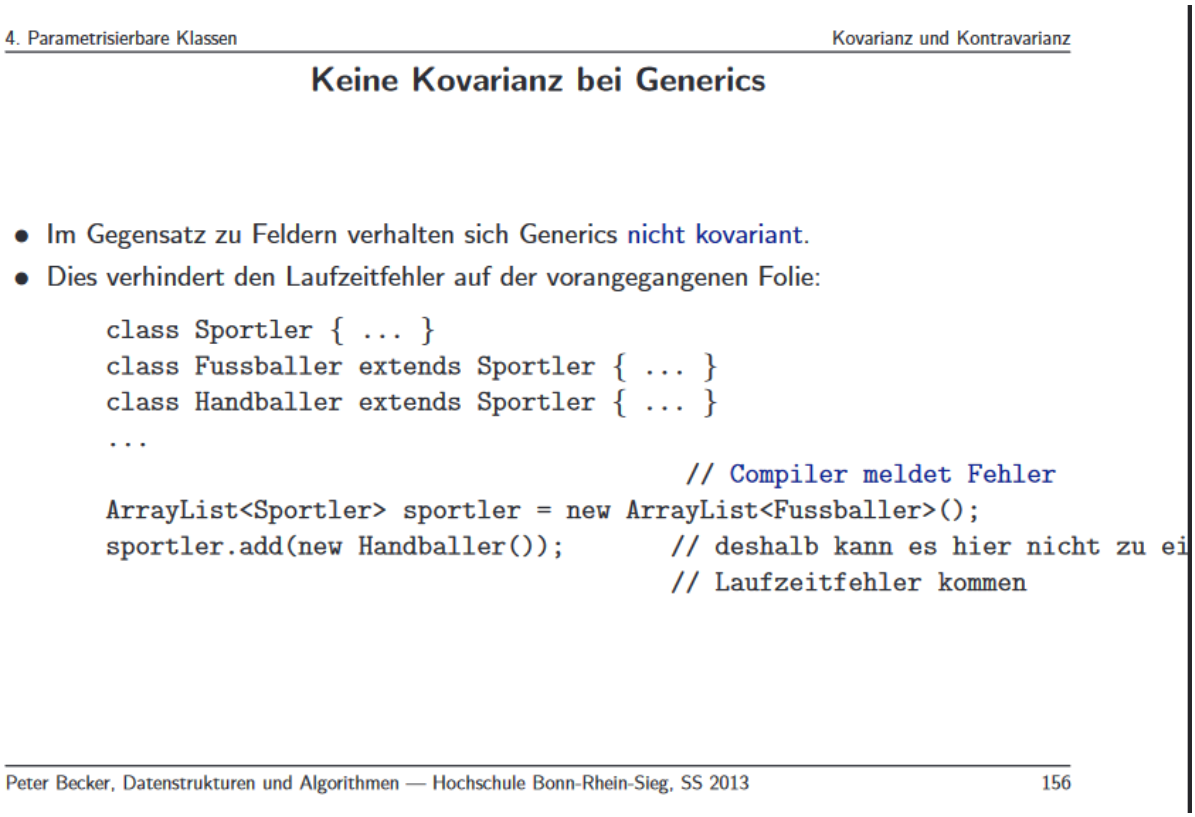
Problem der Kovarianz

- Wegen der Kovarianz bei Feldern liefert der Java-Compiler für die folgenden Anweisungen keinen Fehler bei der Übersetzung:

```
class Sportler { ... }
class Fussballer extends Sportler { ... }
class Handballer extends Sportler { ... }
...
Sportler[] sportler = new Fussballer[11];   // impliziter Up-Cast
sportler[0] = new Handballer();             // ArrayStoreException
```

- Zur Laufzeit würde aber eine `ArrayStoreException` ausgelöst, denn eine `Handballer`-Instanz kann natürlich nicht in ein Feld von `Fussballern` eingefügt werden.
- Durch den impliziten Upcast nach `Sportler[]` geht die Typinformation für das `Fussballer`-Feld verloren.

Peter Becker, Datenstrukturen und Algorithmen — Hochschule Bonn-Rhein-Sieg, SS 2013 — 155



4. Parametrisierbare Klassen — Kovarianz und Kontravarianz

Keine Kovarianz bei Generics

- Im Gegensatz zu Feldern verhalten sich Generics nicht kovariant.
- Dies verhindert den Laufzeitfehler auf der vorangegangenen Folie:

```
class Sportler { ... }
class Fussballer extends Sportler { ... }
class Handballer extends Sportler { ... }
...
                                        // Compiler meldet Fehler
ArrayList<Sportler> sportler = new ArrayList<Fussballer>();
sportler.add(new Handballer());       // deshalb kann es hier nicht zu ei
                                      // Laufzeitfehler kommen
```

Peter Becker, Datenstrukturen und Algorithmen — Hochschule Bonn-Rhein-Sieg, SS 2013 — 156

Abbildung 1: Zitat: Peter Becker, Datenstrukturen und Algorithmen — Hochschule Bonn-Rhein-Sieg, SS 2013, Folien 155-156

1 Quiz Y

Aufgabe 97: Exception: Konstruktor

Schreiben Sie eine `public`-Klasse `MyException`, die von `java.lang.Exception` erbt. Der Konstruktor dieser Klasse hat einen Parameter `n` vom Typ `Double` und ruft den Konstruktor von `Exception` auf, der einen formalen Parameter vom Typ `String` hat, und zwar mit folgendem aktualem Parameter:

Falls `n` auf ein Objekt verweist, den in diesem Objekt gespeicherten Wert als `String`, ansonsten String "`null`".

Erinnerung: Die Klassenmethode `toString` von `Double` liefert eine `String`-Repräsentation ihres Parameters zurück; Klasse `String` hat einen Konstruktor mit einem Parameter vom formalen Typ `String`.

Lösungsvorschlag:

Abschnitt Exceptions aus dem Foliensatz 5 (Fehlerbehandlung)

```java
public class MyException extends Exception { // alternativ: java.lang.Exception

        public MyException ( Double n ) {
                super ( n == null ? "null" : new String ( Double.toString(n));
        }
}
```

Aufgabe 98: Textbasierter Zugriff

Welche Klassen stehen in `java.io` für den textbasierten Zugriff auf Daten zur Verfügung? Zu welchen Basisklasse aus dem bytebasierten Zugriff sind diese Klasse analog?

Lösungsvorschlag:

Abschnitt Textdaten direkt (ohne Streams) aus dem Foliensatz 8 (Streams und Files)

Die Klasse `Reader` und `Writer` und davon abgeleitete Klassen stehen zur Verfügung. Die Klasse `Reader` ist analog zur Klasse `Inputstream`, und `Writer` zur Klasse `OutputStream`.

Aufgabe 99: Beispiele für Fehler

Was unterscheidet lexikalische und syntaktische Fehler von semantischen Fehlern in Java? Gehen Sie dabei auf die Rolle des Programmablaufs ein.

Lösungsvorschlag:

Abschnitt (In)korrektheit auf den einzelnen Abstraktionsebenen aus dem Foliensatz 12 (Korrekte Software)

Semantische Fehler werden erst zur Laufzeit des Programms entdeckt, indem eine RuntimeException geworfen wird. Die anderen beiden Fehlerarten werden bereits vor der Laufzeit, und zwar zur Compilezeit vom Compiler entdeckt.

Aufgabe 100: Darstellungs- und Implementationsinvariante

Beschreiben Sie kurz und bündig, aber präzise und unmissverständlich die Darstellungsinvariante der Klasse `String`, überlegen Sie sich eine möglichst einfach gehaltene Implementation (also der `private`-Teil) der Klasse `String` und formulieren Sie dazu die Implementationsinvariante.

Lösungsvorschlag:

Abschnitt Korrektheit von Klassen aus dem Foliensatz 12 (Korrekte Software)

Darstellungsinvariante: Ein Objekt der Klasse `String` repräsentiert eine Sequenz von Zeichen (alternativ: von `char`).

Implementationsinvariante: Ein Arrayobjekt von `char`, insbesondere nicht `null`.

Das Verhältnis von Darstellungs- und Implementationsinvariante ist, dass im Array ist die Sequenz gespeichert.