Check und Prepare

Vorbereitende Übungen

TECHNISCHE UNIVERSITÄT DARMSTADT

Autoren: Nhan Huynh und Darya Nikitina
Fachbereich: Informatik
Übungsblatt: 08
Version: 20. November 2021
Semesterübergreifend

V1 Theoriefragen

★★★

1. Welche Vorteile ergeben sich durch die Nutzung von Generics?
2. Diskutieren Sie Vor- und Nachteile von Collections gegenüber Arrays unter den folgenden Aspekten:
   (a) Finden von Elementen
   (b) Einfügen neuer Elemente
   (c) Löschen von Elementen
3. Auch in Racket haben Sie Listen kennengelernt. Ist das Java-Interface
4. `java.util.List` vergleichbar mit den Listen aus Racket?

V2 Collections und Exceptions

★★★

Gegeben sei folgender Codeausschnitt:

```
1  double foo(double[] numbers, double n) {
2    LinkedList<Double> list = new LinkedList<>();
3    for (double x : numbers) {
4      if (x > 0 && x <= n && x % 2 != 0) {
5        list.add(x);
6      }
7    }
8    Collections.sort(list);
9    return list.getLast();
10 }
```

(1) Beschreiben Sie kurz und bündig, aber präzise und unmissverständlich was der oben gegebene Code macht.

(2.1) An welcher Stelle kann im Code eine Exception geworfen werden? Durch welche Eingaben wird sie ausgelöst?

(2.2) Modifizieren Sie den Code mithilfe eines `try/catch`-Blockes so, dass in diesen Fällen die Nachricht der Exception auf der Konsole ausgegeben wird.

V3 A-well-a bird bird bird, bird is the word Part I

In dieser Aufgabe betrachten wir eine stark reduzierte Typhierarchie zur Modellierung von Vögeln. Dabei stellen die Pfeile die Erbbeziehungen zwischen Klassen dar. Dazu ist das folgende Typdiagramm in Abbildung 1 gegeben. Hier ist `Bird` also die Oberklasse und die drei anderen Klassen sind Erben dieser.

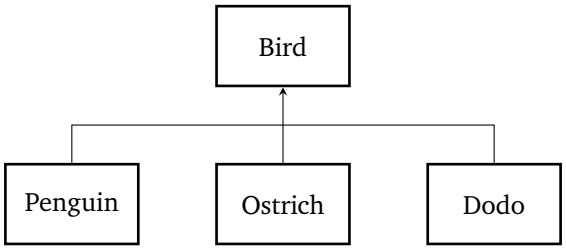


Abbildung 1: Typhierarchie mit drei Vogelarten

Wir nutzen die generische Datenstruktur `Vector<E>`. Dabei beschränken wir uns auf die Methode `void add(E entry)`, die ein Element vom Typ `E` in den Vector einfügt.

(1): Deklarieren und initialisieren Sie eine Variable `v` mit dem **statischen** Basistyp `List` und dem **dynamischen** Typ `Vector`, so dass darin genau Objekte der Typen `Birds`, `Penguin`, `Ostrich` und `Dodo` gespeichert werden können. Nutzen Sie generische Typparameter!

(2): Geben Sie Java-Code an, um in den obigen `Vector v` jeweils ein neues Element vom Typ `Penguin` und `Ostrich` einzufügen. Sie dürfen zur Vereinfachung die Parameter der Konstruktoren der Klassen `Penguin` und `Ostrich` durch ..... abkürzen.

(3): Die Methode `addAll(Birds)` existiert im Interface `List` und fügt eine gesamte `Collection` in eine gegebene `Collection` ein. Die Klasse `ArrayList` implementiert das Interface `List`. Ist die folgende Anweisung – nachdem Code der vorherigen Aufgaben – dann zulässig?

```
1 return new ArrayList<Dodo>().addAll(v);
```

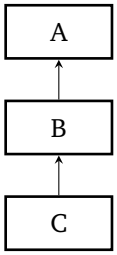
V4 Elemente tauschen

Schreiben Sie eine Methode

```
void switchElements(T[] a, int i, int j)throws IllegalArgumentException
```

Die Methode vertauscht die Elemente im übergebenen Array `a` an den zwei angegebenen Indizes `i` und `j`. Falls für `a` eine `null`-Referenz übergeben wird oder einer der Indizes nicht in dem Array liegt, soll eine `IllegalArgumentException` geworfen werden.

V5 Typhierarchie



Wir betrachten eine Typhierarchie (dargestellt in der Abbildung rechts) mit einer Klasse `A` und einem Erben `B`. Von `B` ist wiederum `C` abgeleitet. Markieren Sie im folgenden Java-Code jeweils hinter `//`, ob der Compiler die Zeile akzeptiert („Okay“) der ablehnt („Fehler“). Lösen Sie die Aufgabe zunächst durch eigene Überlegungen und überprüfen Sie erst später mittels Eclipse.

```java
class A {

}

class B extends A {

}

class C extends B {

}

public class G {

  public static void m(List<B> a, List<? extends B> b, List<? super B> c) {
    a.add(new C()); //
    b.add(new B()); //
    c.add(new C()); //
    a.add(new B()); //
    b.add(new A()); //
    c.add(new B()); //
    a.add(new A()); //
    b.add(null);    //
    c.add(new A()); //
    b.add(new C()); //
  }

  public static void main(String args[]) {
    m(new Vector<B>(), new Vector<C>(), new Vector<A>());
  }
}
```

V6 A-well-a bird bird bird, bird is the word Part II

Wir erweitern unsere Typhierarchie für Vögel aus Aufgabe nd betrachten neben den nicht-fliegen den Vögeln nun auch ihre flatternden Artgenossen.

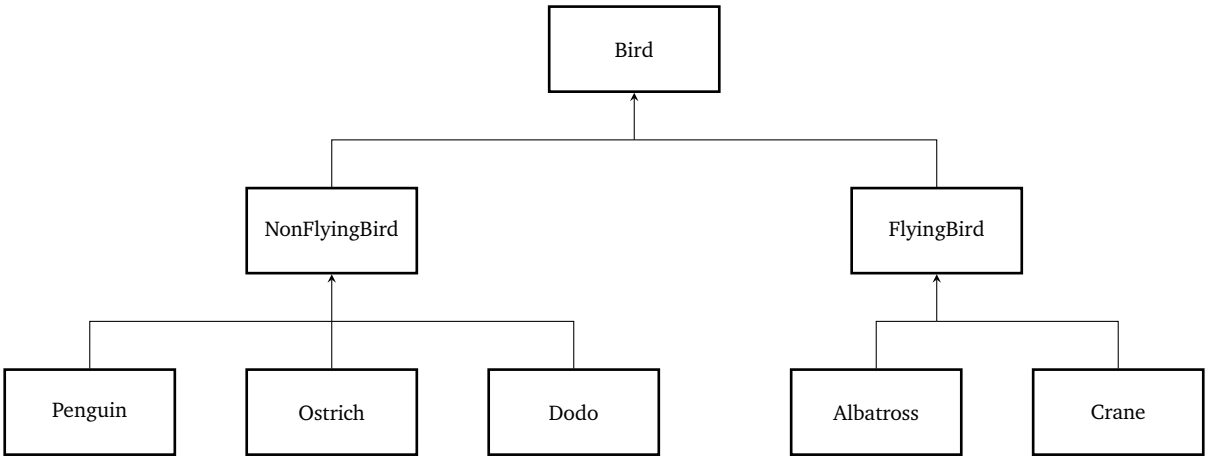


Abbildung 2: Erweiterte Typhierarchie

Vervollständigen Sie die untenstehenden Deklarationen der Methoden. Dabei sind nur die mit ..... markierten Stellen zu bearbeiten! Geben Sie zu jeder Typangabe eine kurze Erklärung, warum genau diese Typangabe die am besten passende oder korrekte ist.
Ihre Lösungen sollen möglichst weitgehend die Typsicherheit garantieren, aber gleichzeitig flexibel für möglichst viele konkrete Parametertypen sein.

Es sind immer generische **Sub**typen zu nutzen.

> **Hinweis:** Zur Vereinfachung wird in den Beispielen nicht auf `null` oder auf eine leere Liste getestet.

(1): Die Methode `getFirst(List<.....> aListOfBirds)` liefert den ersten Vogel einer (nicht-leeren) Liste. Das Ergebnis muss kompatibel zum Typ `Bird` sein.

```
1 Bird getFirst(List<.....> aListOfBirds) {
2   return aListOfBirds.get(0);
3 }
```

(2): Die Methode `void add(Birds b, List<.....> aListOfBirds)` fügt einen neuen Vogel in die Liste ein. Es sollen dabei Vögel jedes bekannten Typs eingefügt werden können.

```
1 void add(Bird b, List<.....> aListOfBirds) {
2   aListOfBirds.add(b);
3 }
```

Übungsblatt 08: Vorbereitende Übungen von Nhan Huynh und Darya Nikitina

V7 Array Utility-Klasse

In dieser Aufgabe wollen wir eine bereits vorhandene Utility-Klasse, für Arrays vom Datentyp `int`, so umschreiben, dass diese für jeden beliebigen Datentyp verwendet werden kann. Die Klasse `ArrayUtils` implementiert folgende Methoden:

`void printArray (int[] array)` bekommt ein `int`-Array übergeben und gibt dessen Elemente auf der Konsole aus.

```
1 public static void printArray(int[] array) {
2   for (int i = 0; i < array.length; i++) {
3     System.out.print(array[i]);
4     if (i < array.length - 1) {
5       System.out.print(" ; ");
6     }
7   }
8   System.out.println();
9 }
```

`int getArrayIndex(int[] array, int value)` bekommt ein `int`-Array und einen Wert übergeben und durchsucht das Array nach dem übergebenen Wert. Wird der Wert gefunden, wird dessen Index im Array zurückgegeben, andernfalls `-1`.

```
1 public static int getArrayIndex(int[] array, int value) {
2   for (int i = 0; i < array.length; i++) {
3     if (array[i] == value) {
4       return i;
5     }
6   }
7   return -1;
8 }
```

`void simpleSort(int[] array)` sortiert das übergebene `int`-Array in aufsteigender Reihenfolge.

```
1 public static void simpleSort(int[] array) {
2   for (int i = 0; i < array.length; i++) {
3     for (int j = i + 1; j < array.length; j++) {
4       if (array[i] > array[j]) {
5         int backup = array[i];
6         array[i] = array[j];
7         array[j] = backup;
8       }
9     }
10  }
11 }
```

Schreiben Sie nun eine Klasse `GenericArrayUtils`, die alle drei oben genannten Methoden mit einem beliebigen Datentyp `T`, bzw. `T[]` für Arrays, implementiert.

> **Hinweis:** Überlegen Sie sich, wie Sie Elemente vom Typ `T` miteinander vergleichen können und welche Voraussetzung dieser Typ `T` mit sich bringen muss. Wie kann sich das im Klassenkopf der zu implementierenden Klasse widerspiegeln?

V8 XYZ

Gegeben seien die folgenden Klassen:

```
1  public class Triple<X, Y, Z> {
2
3    public X x;
4    public Y y;
5    public Z z;
6
7    public Triple(X x, Y y, Z z) {
8      this.x = x;
9      this.y = y;
10     this.z = z;
11   }
12 }
13
14 public class Utils {
15
16 }
```

Erweitern Sie nun die Klasse `Utils` um eine `public`-Klassenmethode `intoMap`, ohne dabei den Klassenkopf zu modifizieren. Die Methode bekommt eine `java.util.List` von Tripeln übergeben und gibt eine `java.util.Map` zurück. Jedes `Triple` in der Liste wird in die `Map` überführt, indem Sie die `x`-Variable des Triples als Schlüssel verwenden und ein neues Paar aus der `y`- und `z`-Variable des Triples erstellen, um dies als Wert des zugehörigen Schlüssels zu verwenden.

V9 Matrizenmultiplikation ★★★

In dieser Aufgabe wollen wir eine Matrix Klasse implementieren, die es uns erlaubt jeden beliebigen vergleichbaren Datentyp unter Anwendung von Java Generics mit ihr zu verwenden.

Um Ihnen die Aufgabe zu erleichtern, wird ihnen ein Interface bereitgestellt, welches benutzt werden soll um arithmetische Operationen mit den Matrizen durchzuführen. Bevor man einen Datentyp mit unserer `Matrix` Klasse benutzen kann, muss man zuvor das arithmetische Interface für diesen konkreten Datentyp implementieren. Machen Sie sich mit den Beispielen auf der folgenden Seite vertraut.

Generisches Interface:

```java
/**
 * Defines generic arithmetic operations.
 *
 * @param <T> the type of the arithmetic elements
 */
public interface Arithmetic<T> {

  /**
   * @returns the generic arithmetic representation of zero.
   */
  T zero();

  /**
   * Returns the result of the addition of a and b.
   *
   * @param a the first summand
   * @param b the second summand
   * @return the result of the addition of a and b
   */
  T add(T a, T b);

  /**
   * Returns the result of the multiplication of a and b.
   *
   * @param a the first multiplicand
   * @param b the second multiplicand
   * @return the result of the multiplication of a and b
   */
  T mul(T a, T b);
}
```

Konkrete Implementierung für Gleitkommazahlen mit dem Datentyp Float:

```java
/**
 * Defines float arithmetics.
 */
public class FloatArithmetic implements Arithmetic<Float> {

  @Override
  public Float zero() {
    return 0f;
  }

  @Override
  public Float add(Float a, Float b) {
    return a + b;
  }

  @Override
  public Float mul(Float a, Float b) {
    return a * b;
  }
}
```

Bearbeiten Sie ausgehend davon die Aufgaben auf der nächsten Seite.

V9.1 Erstellen der Matrix-Klasse

Erstellen Sie zunächst die Klasse `public class Matrix<T extends Comparable<T>>`, ie die folgenden aufgezählten Variablen besitzt:

- `private Arithmetic<T> arithmetic` ist zuständig für das Durchführen von arithmetischen Operationen.
- `private LinkedList<LinkedList<T>> data` enthält die Daten der Matrix. Hierbei repräsentiert die äußere `LinkedList` die Reihen und die innere `LinkedList` die Spalten der Matrix.
- `private int rows` wird zum speichern der aktuellen Anzahl der Reihen der Matrix verwendet.
- `private int columns` wird zum speichern der aktuellen Anzahl der Spalten der Matrix verwendet.

Implementieren Sie nun folgende Methoden:

- `public Matrix(int rows, int columns, Arithmetic<T> arithmetic)` erhält die gewünschte Größe der Matrix in Form von Reihen und Spalten und ein entsprechendes Objekt dessen Klasse das arithmetische Interface implementiert. Alle übergebenen Variablen dieser Methode sollen in den entsprechenden Objektvariablen gespeichert werden. Zusätzlich soll jeder Zellenwert mit `arithmetic.zero()` initialisiert werden.
- `public int getRows()` gibt die aktuelle Anzahl der Reihen der Matrix zurück.
- `public int getColumns()` gibt die aktuelle Anzahl der Spalten der Matrix zurück.
- `public T getCell(int row, int column)` erhält den Index einer Reihe sowie den Index einer Spalte und gibt den Wert der Zelle zurück.
- `public void setCell(int row, int column, T value)` erhält den Index einer Reihe sowie den Index einer Spalte und einen gewünschten Wert und setzt diesen an der entsprechenden Stelle in der Matrix ein.

V9.2 Addition und Multiplikation

Implementieren...

...Sie nun die Methode `public Matrix<T> add(Matrix<T> other)`. Diese erhält eine andere Matrix, addiert die übergebene Matrix und die Matrix, auf der die Methode aufgerufen wurde, miteinander und gibt das Ergebnis der Addition zurück. Sollten die Reihen- und/oder Spaltenanzahl der beiden Matrizen nicht übereinstimmen, soll `null` zurückgegeben werden.

...Sie nun die Methode `public Matrix<T> mul(Matrix<T> other)`. Diese erhält eine andere Matrix, multipliziert die übergebene Matrix und die Matrix, auf der die Methode aufgerufen wurde, miteinander und gibt das Ergebnis der Multiplikation zurück. Sollten die Spaltenanzahl der aktuellen Matrix sich von der Reihenanzahl der übergebenen Matrix unterscheiden, soll `null` zurückgegeben werden.