Aufgabe 2 Abstrakte Klassen / Interfaces / Generics

Die Aufgabe muss zu Beginn des Praktikums vollständig bearbeitet sein!
Das Kopieren des Quelltextes aus den Java-Collection-Klassen ist nicht erlaubt.

A2-1 Leichtgewichtiges Framework für Collections

Sie sollen eine leichtgewichtiges Framework für Teile der Collection-Klassen implementieren. Zunächst als Hierarchie von nicht generischen Typen, die dann in ein Framework von generischen Typen überführt werden soll.

Wir beginnen mit den Interfaces `LWCollection, LWList, LWSet, LWQueue`.

LWCollection

Alles `LWCollection's` sind `Iterable<Object>`

Methoden:

```
boolean add(Object elem)
boolean contains(Object elem)
int size()
boolean isEmpty()
boolean remove(Object elem)
boolean addAll(Iterable<Object> col)
boolean removeAll(Iterable<Object> col)
```

LWList Methoden

```
Object get(int index) throws IndexOutOfBoundsException
Object set(int index, Object elem) throws IndexOutOfBoundsException
void add(int index, Object elem) throws IndexOutOfBoundsException
int indexOf(Object elem)
Object remove(int elem) throws IndexOutOfBoundsException
```

LWSet = Markerinterface

LWQueue

```
void enqueue(Object elem);
Object dequeue() throws NoSuchElementException;
boolean isEmpty();
int size();
```

Definieren Sie die Interfaces und implementieren Sie zunächst zwei abstrakte Klassen `LWAbstractCollection` und `LWAbstractList`, die möglichst sinnvoll Methoden der Interfaces `LWCollection` und `LWList` implementieren.

A2-2

Implementieren Sie zwei konkrete Klassen `LWArrayList`, die das `LWList` Interface mit Hilfe eines Resizing Arrays implementiert, und `LWLinkedList`, die das `LWList` und das `LWQueue` Interface als doppelt verkettete Liste mit Wächterknoten implementiert. Recyclen Sie für die `LWLinkedList` die `DoublyLinkedList` aus dem ersten Semester. Sie müssen für den Iterator der `LWLinkedList` kein `remove` schreiben.

A2-3

Implementieren Sie auch die Methoden `equals` und `hashCode` für die Listen.

A2-4

Erzeugen Sie Listen für die beiden Implementierungen.

1. Je eine Liste mit Integer Werten und je eine Liste mit Integer und Double Werten und berechnen Sie für diese Liste die Summe der Elemente.
2. Je eine Liste mit gemischten Werten, Zahlen, Zeichenketten und andere Objekte und berechnen Sie auch für diese Listen die Summe der enthaltenen Zahlen.

A2-5

Überführen Sie das Modell aus A2-1 / A2-3 in ein generisches Modell.

A2-6

Erzeugen Sie Listen für die beiden Implementierungen.

1. Je eine Liste mit Integer Werten und je eine Liste mit Integer und Double Werten und berechnen Sie für diese Liste die Summe der Elemente.
2. Je eine Liste mit gemischten Werten, Zahlen, Zeichenketten und andere Objekte und berechnen Sie auch für diese Listen die Summe der enthaltenen Zahlen.
3. Schreiben Sie eine statische generische Methode, die zwei generische Listen als Parameter hat und kopieren Sie die Elemente der ersten Liste in die zweite. Achten Sie darauf, dass der generische Typ der zweiten Liste möglichst flexibel ist.
4. Ergänzen Sie die Implementierungen der Listen um einen verallgemeinerten Kopierkonstruktor, der möglichst flexibel ist.

A2-7

Welche generischen Typen sind zu den folgenden statischen Listentypen kompatibel? Gefragt sind hier nur die Typargumente.

```
LWList<Number> lwln;                 Number
LWList<? extends Object> lwlo;       Alles
LWList<Object> lwlo;                 Alles unter Iterable
LWList<? super Integer> lwlsi;       Iterable>= ?>=Integer
LWList<?> lwl;                       Alles
```

A2-8 Einzelfragen Interfaces / abstrakte Klassen

1. Wenn eine abstrakte Klasse, die ein Interface implementiert, Teile der Methoden nicht realisiert, was gilt dann für weitere abstrakte Klassen? Was gilt für die konkreten Klassen?
2. Ist es erlaubt, dass eine Klasse zwei Interfaces implementiert, wenn diese abstrakte Methoden gleicher Signatur enthalten (Ergebnistypen sind identisch)?
3. Dürfen abstrakte Klassen Konstruktoren und Instanzvariablen enthalten?
4. Was meint Vererbung von Spezifikation? Welche Art von Vererbung kennen Sie darüber hinaus?

A2-9 Einzelfragen Generics

1. Warum sollten Sie keine bivarianten Ergebnistypen verwenden?
2. In welchen Situationen können Sie mit bivarianten Parametern arbeiten?
3. Gegen Sie ein Beispiel, in dem der Cast auf eine Typvariable zur Laufzeit einen Fehler verursacht. An welcher Stelle tritt der Fehler auf?
4. Wenn Ihre Implementierung mit einem generischen Array arbeitet, was müssen Sie dann sicher stellen?
5. Was ist der Unterschied zwischen Typvariablen und Typargumenten? Was muss für Typargumente gelten?

A2-10 Factory Pattern

1. Entwickeln Sie für die generischen Listen aus A2-5 eine Factory, die gesteuert über einen Parameter eine der beiden Implementierungen erzeugt.
2. Entwickeln Sie einen Client, dem beim Start der Parameter für die Auswahl übergeben wird, und der die Factory nutzt, um die zugehörige Implementierung zu ermitteln. Führen Sie in dem Client ein Experiment durch, in dem Sie die Zeit für das Einfügen von N-Elementen am Anfang der Liste messen. (`System.nanoTime()` liefert die Zeit in Nano-Sekunden). Starten Sie mit N=1000 und verdoppeln das N bis Sie „ungeduldig“ werden und messen in jedem Durchlauf die Zeit für das Einfügen der N Elemente. Führen Sie den Test für die Implementierung mit einer `LWLinkedList` und einer `LWArrayList` durch. Was beobachten Sie?
3. Entwickeln Sie eine Factory für die Scanner-Implementierung aus Aufgabe 1. Nutzen Sie die Factory, um die Zeiten und die Ergebnisse ihrer Implementierung mit denen einer anderen Gruppe zu vergleichen. Führen Sie dazu entweder unterschiedliche Package- oder Klassennamen pro Gruppe ein.