Assignment 2

Abgabe bis zum 5. Mai 2020 (Dienstag) um 23:59 Uhr

Geben Sie Ihr Assignment bis zum 5. Mai 2020 (Dienstag) um 23:59 Uhr auf folgender Webseite ab:
https://assignments.hci.uni-hannover.de
Ihre Abgabe muss aus einer einzelnen zip-Datei bestehen, die den Quellcode und ggf. ein PDF-Dokument bei Freitextaufgaben enthält. Beachten Sie, dass Dateien mit Umlauten im Dateinamen nicht hochgeladen werden können. Entfernen Sie die daher vor der Abgabe die Umlaute aus dem Dateinamen. Überprüfen Sie nach Ihrer Abgabe, ob der Upload erfolgreich war. Bei technischen Problemen, wenden Sie sich an Patric Plattner oder Dennis Stanke. Es wird eine Plagiatsüberprüfung durchgeführt. Gefundene Plagiate führen zum Ausschluss von der Veranstaltung.

Aufgabe 1: Bruchzahlen

a) Erstellen Sie zwei Dateien: `Fraction.java` und `Main.java`. Erstellen Sie in den Dateien die gleichnamigen Klassen. Die Klassen sollen den `public` modifier nutzen (z.B. `public class Test { ... }`). Erstellen Sie in der `Main` Klasse die `main` Methode. Die `Fraction` Klasse soll zwei `int` Attribute beinhalten: eins für den Zähler (numerator) und eins für den Nenner (denominator). Die `Main` Klasse soll in dem `model` package liegen. Legen Sie die Dateien in die jeweiligen Ordner, um die Packagestruktur umzusetzen. Die Packagestruktur soll wie folgt implementiert werden:

```
aufgabe1
├── model
│   └── Fraction
└── Main
```

Die `Fraction` Klasse soll eine `public String str()` Methode haben, die die Bruchzahl in folgendem Format als String ausgibt: `Zähler/Nenner`, also zum Beispiel: `5/6`. Um ihnen dies zu erleichtern ist in den Lösungshinweisen ein Hinweis auf eine hilfreiche Methode.

Implementieren Sie einen Konstruktor `public Fraction(int num, int denom)`, der die beiden Attribute entsprechend initialisiert.

Erstellen Sie in der `main` Methode zwei Bruchzahlen `f1` und `f2` und geben Sie diese auf die Kommandozeile aus.

b) Implementieren Sie eine Methode `private void reduce()`, die den Bruch kürzt. Also z.B. den Bruch $\frac{3}{6}$ auf $\frac{1}{2}$ kürzt.

Implementieren Sie Getter und Setter für Zähler und Nenner. Also folgende Methoden: `public int getNumerator()`, `public int getDenominator()`, `public void setNumerator(int num)` und `public void setDenominator(int denom)`. Die Setter sollen, nachdem Sie einen Wert verändert haben, den Bruch kürzen. Schreiben Sie darüber hinaus auch einen Setter `public void setValues(int num, int denom)`, der beide Werte gleichzeitig setzt und den Bruch danach kürzt.

Erweitern Sie den Konstruktor `public Fraction(int num, int denom)`, so dass er die Bruchzahl, nachdem die Attribute initialisiert wurden, kürzt.

Vorlesung: apl. Prof. Dr.-Ing. Matthias Becker xmb@hci.uni-hannover.de
Übung: Patric Plattner patric.plattner@hci.uni-hannover.de
Organisation: M.Sc. Dennis Stanke dennis.stanke@hci.uni-hannover.de

Beispiel:

```
Fraction f = new Fraction(3, 6);
System.out.println(f.str());
f.setNumerator(4);
System.out.println(f.str());
f.setValues(3, 9);
System.out.println(f.str());
```

Output:

```
1/2
2/1
1/3
```

c) Implementieren Sie eine Methode `private void extend(int amount)`, die einen Bruch um ein Vielfaches erweitert. So würde beispielsweise bei `amount = 3` der Bruch $\frac{1}{3}$ auf $\frac{3}{9}$ erweitert werden.

Schreiben Sie dann eine Methode `public Fraction add(Fraction summand)`, welche die beiden Brüche addiert und das Ergebnis als neuen gekürzten Bruch zurückgibt. Schreiben Sie einen Beispielaufruf in der `main` Methode mit den beiden Brüchen `f1` und `f2`.

Beispiel:

```
Fraction f1 = new Fraction(1, 2);
Fraction f2 = new Fraction(2, 3);
Fraction f3 = f1.add(f2);
System.out.println(f3.str());
```

Output:

```
7/6
```

d) Implementieren Sie eine Methode `public Fraction multiply(Fraction multiplicand)`, die die beiden Brüche multipliziert und das Ergebnis als neuen gekürzten Bruch zurückgibt. Schreiben Sie einen Beispielaufruf in der `main` Methode mit den beiden Brüchen `f1` und `f2`.

Beispiel:

```
Fraction f1 = new Fraction(1, 2);
Fraction f2 = new Fraction(2, 3);
Fraction f3 = f1.multiply(f2);
System.out.println(f3.str());
```

Output:

```
1/3
```

Bestehenskriterien

- Ihr Code ist in der Package-Struktur, die in **a)** aufgeführt wurde, organisiert.
- Ihr Code kompiliert ohne Fehler.
- Ihre Ausgaben entsprechen den Ausgaben der Beispiele. Die genaue Formatierung kann von den Beispielen abweichen.
- Die zu implementierenden Methoden funktionieren auch mit anderen Eingaben wie in der Aufgabe beschrieben.

Lösungshinweise

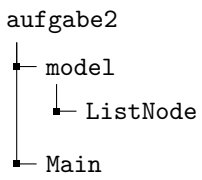
- Sie können die `String.format()` Methode nutzen um den String auszugeben. Mit dieser Methode können Sie Attribute in Strings einfügen. Ein Beispiel wäre `String.format("Your name is %s and you are %d years old.", name, age)`. Mehr infos dazu finden Sie hier im Java API: https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#format-java.lang.String-java.lang.Object...-

Aufgabe 2: Einfach verkettete Listen

Zu einfach verketteten Listen, finden Sie ein erklärendes Lernvideo im Stud.IP unter "Aufzeichnungen".

In dieser Aufgabe sollen explizit keine Methoden oder Klassen aus der Java Standardbibliothek, außer `System.out` und Methoden/Klassen, die explizit erwähnt werden, genutzt werden.

a) Erstellen Sie zwei Dateien: `ListNode.java` und `Main.java`. Erstellen Sie in den Dateien die gleichnamigen Klassen. Die Klassen sollen den `public` modifier nutzen (z.B. `public class Test { ... }`). Erstellen Sie in der `Main` Klasse die `main` Methode. Die Klassen sollen in folgender Package Struktur angeordnet sein:



Die `ListNode` Klasse soll ein `int` Attribut für den Wert haben und ein `ListNode` Attribut für das nachfolgende Kettenglied.

Implementieren Sie Getter und Setter für die beiden Attribute.

Erstellen Sie zur Initialisierung einen Konstruktor `public ListNode(int value, ListNode next)`.

b) Erstellen Sie eine `ListHead.java` Datei mit der gleichnamigen Klasse. Die Klasse soll den `public` modifier nutzen. Zudem soll die Klasse den Kopf einer Liste darstellen, sodass der Nutzer dieser Klasse nicht direkt mit der Implementierung Ihrer Liste in Kontakt treten muss. Die `ListHead` Klasse soll im `aufgabe2.model` Package liegen.

Erstellen Sie dazu auch einen Konstruktor `public ListHead(int[] data)`, welcher den Inhalt eines Integer Arrays in einer Liste speichert. Implementieren Sie auch einen `public ListHead(ListNode head)` Konstruktor, der den Listenkopf zu dem gegebenen Parameter setzt.

Erstellen Sie eine `public String str()` Methode in der `ListHead` Klasse, welche die Liste als String zurückgibt. Der String soll wie folgt formatiert sein: `"(zahl1)->(zahl2)->null"`. Beispiele dafür finden Sie in den Beispielaufrufen dieser Aufgabe. Zum vereinfachen der Ausgabe finden Sie einen Hinweis in den Lösungshinweisen.

Alle Zugriffe auf Daten zwischen Klassen (also z.B. wenn Sie den Wert eines `ListNode` in einer Methode des `ListHead` auslesen wollen) sollen per Getter/Setter erfolgen und nicht über direkte Zugriffe auf die Attribute.

Beispiel:

```
int[] dataArr = {1, 2, 3};
ListHead list1 = new ListHead(dataArr); // list1 = (1)->(2)->(3)->null
System.out.println(list1.str());

ListHead list2 = new ListHead(
    new ListNode(4,
        new ListNode(5,
            null
        )
    )
); // list2 = (4)->(5)->null
System.out.println(list2.str());
```

Output:

```
(1)->(2)->(3)->null
(4)->(5)->null
```

Erstellen Sie eine Liste, die von folgendem Array erstellt wurde: `{1, 2, 16, 4, 9}`. Geben Sie die Liste auf die Konsole aus. Die Ausgabe sollte demnach wie folgt aussehen: `(1)->(2)->(16)->(4)->(9)->null`.

c) Implementieren Sie eine Methode `void sort()` auf der `ListHead` Klasse, die die Liste **aufsteigend** sortiert. Wir empfehlen dabei den Bubblesort Algorithmus, da Sie mit diesem Algorithmus bereits aus Assignment 1 vertraut sind.

Ihnen bleibt aber selbst überlassen welchen Algorithmus Sie wählen, solange Sie die Funktionsweise des jeweiligen Algorithmus Ihrem Tutor erklären können. Sie dürfen dabei beliebig viele Hilfsmethoden implementieren, dies ist aber optional.

Beispiel:

```
int[] dataArr = {5, 3, 1, 4, 2};

ListHead list = new ListHead(dataArr);

list.sort();

System.out.println(list.str());
```

Ausgabe:

```
(1)->(2)->(3)->(4)->(5)->null
```

Sortieren Sie die Liste, die Sie in **b)** erstellt haben und geben Sie diese erneut aus.

Bestehenskriterien

- Ihr Code ist in der Package-Struktur, die in **a)** aufgeführt wurde, organisiert.
- Ihr Code kompiliert ohne Fehler.
- Ihre Ausgaben entsprechen den Ausgaben der Beispiele. Hier soll das Format genau der Vorgabe entsprechen (`(z1)->(z2)->null`
- Die zu implementierenden Methoden funktionieren auch mit anderen Eingaben wie in der Aufgabe beschrieben.

Lösungshinweise

- Sie können die `String.format()` Methode nutzen um den String auszugeben. Mit dieser Methode können Sie Attribute in Strings einfügen. Ein Beispiel wäre `String.format("Your name is %s and you are %d years old.", name, age)`. Mehr Informationen dazu finden Sie hier im Java API:
  https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#format-java.lang.String-java.lang.Object...-
- Wenn Sie über eine Liste iterieren um Knoten zu finden, die sie z.B. vertauschen wollen, merken Sie sich die Referenzen von dafür relevanten anderen Knoten (z.B. Vorgängern).
- Bedenken Sie, dass Sie unter Umständen auch das `head` Attribut der `ListHead` Klasse beim Vertauschen von Knoten aktualisieren müssen.
- In dem Lernvideo zu einfach verketteten Listen finden sie Infos dazu, wie Sie simple Listenoperationen durchführen (Vertauschen, Einfügen, Löschen).

Aufgabe 3: Debugging

In diesen Aufgaben werden Sie einen fehlerhaften Java Codeabschnitt bekommen. Diese Fehler können **syntaktisch** oder **semantisch** sein. Es kann sich dabei um Compiler- oder Laufzeitfehler handeln. Sie dürfen den Code kompilieren und ausführen um die Fehler zu finden. Arbeiten Sie in den Template-Dateien `DebugMain.java` und `DebugData.java`, welche in der Debug.zip bereits in einer Ordner-/Packagestruktur organisiert sind. Diese finden Sie im Stud.IP. Nutzen Sie Kommentare, um die Fehler zu Kennzeichnen. Gegeben ist der folgende Codeabschnitt:

src/aufabe3/data/DebugData.java

```java
package aufgabe3.data;

import java.lang.Math;

// This class is representing a cartesian coordinate in a 2d space with double coordinates.
class DebugData {
    private double x;
    private double y;

    public DebugData(int x, int y) {
        this.x = x;
        this.y = y;
    }

```

```
15      public double distance(DebugData other) {
16          // = sqrt((this.x - other.y)^2 + (this.y - other.y)^2)
17          return Math.pow(Math.pow(this.x - other.x, 2) + Math.pow(this.y - other.y,
                ↪ 2));
18      }
19
20      public double getX(){
21          return this.x;
22      }
23
24      public double getY(){
25          return this.x;
26      }
27
28      public void setX(double x){
29          this.x = x;
30      }
31
32      public void setY(double y){
33          this.y = y;
34      }
35
36      public String str() {
37          return String.format("(%g, %g)", this.x, this.y);
38      }
39  }
```

src/aufabe3/DebugMain.java

```
1   package aufgabe3;
2
3   class DebugMain {
4       public static void main(String[] args) {
5           DebugData a = new DebugData(3.4, 5.6);
6           DebugData b = new DebugData(1.0, 1.0);
7           // Result should be approx. 5.18
8           System.out.println(String.format("The distance between %s and %s is %g",
9               a.str(), b.str(), a.distance(b)));
10      }
11  }
```

Der Codeabschnitt enthält zwischen 3-5 Fehler. Kompilieren und führen Sie den Code aus. Suchen Sie anhand der Fehlermeldungen des Compilers oder der Laufzeitfehlermeldungen Fehler im Code und korrigieren Sie diese. Kommentieren Sie am Ende der jeweiligen Zeile, was Sie korrigiert haben.

Erstellen Sie während der Fehlerkorrektur eine Errors.txt im aufgabe3 Ordner, in der Sie die Fehler dokumentieren. Schreiben Sie die Zeile(n) des Fehlers auf und beschreiben Sie den Fehler. Kopieren Sie die Fehlermeldung, sofern es eine gab, anhand welcher Sie den Fehler erkannt haben. Die Dokumentation soll wie in folgendem Beispiel aussehen:

```
1   public class Debug { //K: class falsch geschrieben (ckass)
2
3   ...
4
5   /*
6   ...
7   Zeile 1: class Keyword falsch geschrieben (ckass):
```

```
Fehlermeldung:
**************
Debug.java:1: error: class, interface, or enum expected
public ckass Debug {
       ^
**************
Der Compiler erwartet eines der drei oben angegebenen Keywords, hat aber nur das falsch geschriebene ckass bekommen.
...
*/
```

Bestehenskriterien

- Ihr korrigierter Code funktioniert so wie in den Kommentaren im Code beschrieben.
- Alle korrigierten Fehler haben einen kurzen Kommentar, der beschreibt, warum die Stelle im alten Code nicht funktioniert hat.
- Alle Fehlermeldungen (Compiler und Laufzeit) sind in einem Blockkommentar `/* ... */` unter dem Code in der `Debug.java`.

Lösungshinweise

- Im Code vorgegebene Kommentare beschreiben immer das **korrekte** Verhalten des Programms.