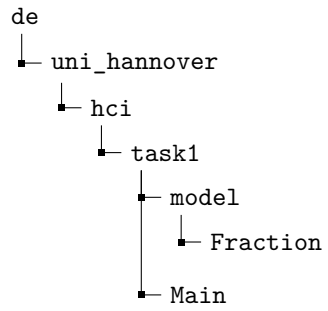
Assignment 2

Abgabe bis zum 27. April 2021 (Dienstag) um 23:59 Uhr

Geben Sie Ihr Assignment bis zum 27. April 2021 (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. In dieser Veranstaltung wird ausschließlich die Java/JavaFX Version 11 verwendet. Code und Kompilate anderer Versionen sind nicht zulässig. Ihre Abgaben sollen die vorgegebenen Dateinamen verwenden. Wiederholter Verstoß gegen diese Regel kann zu Punktabzügen führen.

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 `Fraction` 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:



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. `reduce()` soll den Bruch maximal kürzen, also so weit, bis er nicht mehr gekürzt werden kann.

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)`.

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

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.

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 static Fraction add(Fraction summandA, Fraction summandB)`, welche die beiden Brüche addiert und das Ergebnis als neuen gekürzten Bruch zurückgibt. `summandA` und `summandB` sollen nicht verändert werden, bzw. sollen ihre Werte nach der Addition dieselben sein, wie vor der Addition. 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 = Fraction.add(f1, f2);
System.out.println(f3.str());
```

Output:

```
(7/6)
```

d) Implementieren Sie eine Methode `public static Fraction multiply(Fraction multiplicandA, Fraction multiplicandB)`, die die beiden Brüche multipliziert und das Ergebnis als neuen gekürzten Bruch zurückgibt. `multiplicandA` und `multiplicandB` sollen nicht verändert werden, bzw. sollen ihre Werte nach der Addition dieselben sein, wie vor der Multiplikation. 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 = Fraction.multiply(f1, 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/en/java/javase/11/docs/api/java.base/java/lang/String.html#format(java.lang.String,java.lang.Object...)

Aufgabe 2: LadybugGame

Diese Aufgabe ist die erste in einer Reihe an aufeinander aufbauenden Aufgaben. In allen folgenden Aufgabenblättern wird eine der Programmier-Aufgaben auf die LadybugGame Aufgabe der vorherigen Woche aufbauen. Damit Ihnen dabei kein Nachteil entsteht, wenn Sie die LadybugGame Aufgabe der Vorwoche nicht bearbeitet haben, wird mit jedem Aufgabenblatt eine Musterlösung der LadybugGame Aufgabe der Vorwoche veröffentlicht.

In dieser Reihe an Aufgaben sollen Sie langsam ein Spiel erstellen, das an Kara (https://www.swisseduc.ch/informatik/karatojava/kara/) angelehnt ist. Sie sollen am Ende also ein Programm erstellen, in dem Sie interaktiv eine Spielfigur durch ein Labyrinth navigieren und dabei Gegenstände aufheben können. Dabei werden Sie das Programm Woche um Woche um weitere Funktionen erweitern und dabei neue Gegenstände der Vorlesung in ihr Programm einbauen.

a) Erstellen Sie die leeren Klassen `Application`, `Ladybug`, `Tree` und `Clover`. Alle Klassen sollen den `public` Modifier nutzen. Sie werden diese Klassen im Laufe der Assignments noch erweitern. Hier eine kurze Beschreibung, welche Klasse was machen soll, sobald diese fertig sind:

- `Application`: Diese Klasse enthält lediglich die `main` Methode.
- `Ladybug`: Diese Klasse soll die Spielfigur als Gegenstand auf dem Spielfeld darstellen.
- `Clover`: Diese Klasse soll ein Kleeblatt darstellen, was ein Sammelbarer Gegenstand auf dem Spielfeld ist.
- `Tree`: Diese Klasse soll einen Baum darstellen, welcher ein nicht begehbares Hindernis auf dem Spielfeld ist.

Diese Klassen sollen in eine Packagestruktur gelegt werden, welche dem Reverse Domain Namensschema aus der Vorlesung entsprcht (Informationen zu der Package Naming Convention von Oracle: https://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html). Nutzen Sie dabei die `hci.uni-hannover.de` Domain und benennen Sie ihr Package nach ihrer LUH-ID (Ihr WebSSO Login der Form `XXX-XXX`) oder ihrem Nachnamen.

Erstellen Sie gegebenenfalls Subpackages und legen Sie die Klassen, so wie Sie es für richtig halten, in diese Subpackages und schreiben sie über alle Klassen einen Kommentar, in welchem Sie begründen, weshalb Sie diese Klasse in das entsprechende Subpackage gelegt haben. Bei der Wahl der Subpackages gibt es keine falsche Antwort, allerdings sollen Sie sich bei der Wahl Gedanken machen und ihre Wahl (im Kommentar und bei der Vorstellung) begründen können.

b) Die Klassen `Ladybug`, `Tree` und `Clover` sollen bestimmte Eigenschaften haben. Implementieren Sie diese Eigenschaften und erstellen Sie zusätzlich pro Klasse einen sinnvollen Konstruktor und Setter/Getter für alle Member-Variablen der Klasse, die einen benötigen (Sie müssen ihre Entscheidungen bei ihrem Tutor begründen können). Hier ist die Liste der Eigenschaften, die die Klassen haben sollen:

`Ladybug`

- Ein `Ladybug` hat eine Blickrichtung (Nord/Ost/Süd/West).
- Ein `Ladybug` hat ein Array an `Clover`, welche die bereits aufgesammelten `Clover` darstellen soll.

`Tree`

- Ein Baum braucht hier noch keine Speziellen Eigenschaften.

`Clover`

- Ein `Clover` soll einen Punktewert (`int`) haben.

Da alle diese Klassen ein Gegenstand auf dem Spielfeld (welches auf der Kommandozeile ausgegeben wird) sind, sollen alle auch ein Textsymbol haben, welches diese Gegenstände darstellen soll. Alle drei Klassen sollen deshalb eine Methode haben, die ihr Symbol auf dem Spielfeld darstellt. Bedenken Sie dabei, dass `Ladybug` vier verschiedene Blickrichtungen hat.

c) Erstellen Sie nun die `main` Methode in der `Application` Klasse. Erstellen Sie in der `main` Methode je ein `Ladybug`, `Tree` und `Clover`(Wert: 10 Punkte) Objekt und machen Sie folgendes mit diesen Objekten:

- Geben Sie die Spielfeld-Symbole der drei Objekte aus.
- Drehen Sie den `Ladybug` einmal im Kreis und geben Sie nach jeder Teil-Drehung das Symbol der `Ladybug` aus.
- Geben Sie den Wert der `Clover` aus.

Bestehenskriterien

- Ihr Code ist in der Package-Struktur, die in **a)** aufgeführt wurde, organisiert.
- Ihr Code kompiliert ohne Fehler.
- Ihre Klassen weisen die in den Aufgaben beschriebenen Eigenschaften auf, wobei die genaue Umsetzung nicht vorgeschrieben ist.
- Sie geben die geforderten Informationen in der `main` Methode aus.

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;
    }

    public double distance(DebugData other) {
        return Math.pow(Math.pow(this.x - other.x, 2) + Math.pow(this.y - other.y, 2));
    }

    public double getX(){
        return this.x;
    }

    public double getY(){
        return this.x;
    }

    public void setX(double x){
        this.x = x;
    }

    public void setY(double y){
        this.y = y;
    }

    public String str() {
        return String.format("(%g, %g)", this.x, this.y);
    }
}
```

src/aufabe3/DebugMain.java

```java
package aufgabe3;

class DebugMain {
    public static void main(String[] args) {
```

```
        DebugData a = new DebugData(3.4, 5.6);
        DebugData b = new DebugData(1.0, 1.0);
        // Result should be approx. 5.18
        System.out.println(String.format("The distance between %s and %s is %g",
            a.str(), b.str(), a.distance(b)));
    }
}
```

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:

```
public class Debug { //K: class falsch geschrieben (ckass)

...

/*
...
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.