Programmieren in Java

http://proglang.informatik.uni-freiburg.de/teaching/java/2015/

Java-Übung Blatt 2 (Zusammengesetzte Klassen)

2015-04-30

Hinweise

- Ändern Sie nicht die Schreibweise von Bezeichnen, die auf dem Übungsblatt vorgegeben sind. Dies betrifft sämtliche global sichtbaren Namen von Eclipse-Projekten, Paketen, Klassen, etc.
- Bezeichner und Kommentare bitte auf *Englisch*!
- Schreiben Sie *sinnvolle* Kommentare.
- Laden Sie Ihre Lösungen mit Subversion (SVN) ins Übungssystem hoch. Den entsprechenden Pfad finden Sie online.
- Sollte das Übungssystem Ihre Einreichung nicht übersetzen können, dann wird sie nicht korrigiert und Sie erhalten keine Punkte.
- Die Korrektur Ihrer Abgabe wird unter dem Namen `Feedback-<login>-ex<NN>_<X>.txt` in das jeweilige Projektverzeichnis eingecheckt. Dabei ist `<login>` Ihr *myAccount*-Name und `<NN>_<X>` der Projektname.
- Sie können Ihre Gesamtpunktzahl im Übungsportal einsehen.
- Dokumentieren Sie Ihren Zeitaufwand, den gefühlten Schwierigkeitsgrad, sowie Probleme beim Lösen der Aufgabe oder auch Anregungen und Vorschläge zur Verbesserung in der Datei `<Projektverzeichnis>/erfahrungen.txt`

Hinweis 1: Dieses PDF stammt aus dem Archiv `ex02.zip`. Dieses Archiv enthält weiterhin noch Skelett-Projekte, die für die Bearbeitung der Aufgaben hilfreich sind.

Hinweis 2: Für dieses Blatt gibt es **zwei Wochen** Bearbeitungszeit.

Hinweis 3: Die Anforderungen in den Aufgaben `ex02_2` und `ex02_3` sind bewusst etwas freier formuliert um Ihnen Implementierungs- und Design-Freiheiten zu geben. Falls Ihnen die Anforderungen widersprüchlich oder unklar vorkommen, fragen Sie im Forum nach, wie diese gemeint sind.

Hinweis 4: Am Schluss des Blattes finden Sie noch zwei allgemeine Hinweise zur Java-API, die zum Lösen der Aufgaben hilfreich sein könnten.

Exercise 1 (Zusammengesetzte Klassen, 6 Punkte)

Projektverzeichnis: `ex02_1`. Package: `geometry`.

In Übungsblatt 1 haben Sie eine Ellipse implementiert. Dabei soll es nicht bleiben; wenn Sie aber weitere geometrische Figuren auf die gleiche Art entwickeln wie in Blatt 1, würden Sie wahrscheinlich viel Code duplizieren. Im Folgenden sollen Sie versuchen, diese Code-Duplizierung mittels *zusammengesetzer Klassen* und *Interfaces* zu vermeiden.

Richten Sie Ihr Projektverzeichnis wie gewohnt ein und übernehmen Sie außerdem Ihren Code aus Aufgabe 1.2 (in Eclipse: Copy-Paste im *Package Explorer*). Sie können hierzu auch die Musterlösung zu `ex01_2` verwenden, die Sie als Skelett im Archiv finden (z.B. falls Sie Aufgabe `ex01_2` nicht bearbeitet haben).

Zusammengesetzte Klassen Implementieren Sie eine Klasse Rectangle, die die selben Operationen wie Ellipse unterstützt. Ein Rechteck hat einen Ankerpunkt („links-oben“), sowie Höhe und Breite. Wie Ellipsen sollen auch Rechtecke unveränderlich sein.

Sowohl Ellipsen als auch Rechtecke sollten als zusammengesetzte Klassen implementiert werden. Beide Figuren werden typischerweise durch *Vektoren* und *Punkte* beschrieben; Beide haben jeweils einen Punkt, der ihre Position im Koordinatensystem beschreibt, sowie einen Vektor. Bei Rechtecken repräsentiert dieser die Seitenlängen, bei Ellipsen die Länge der Radien. Die Basisdatentypen von Java enthalten aber weder Punkte noch Vektoren.

Implementieren Sie also Klassen für Punkte und Vektoren. Punkte und Vektoren sind sich grundsätzlich zwar ähnlich, aber mathematisch doch verschieden – wir modellieren diese Unterscheidung durch die Verwendung unterschiedlicher Klassen.[1] Implementieren Sie auch Methoden

[1] Vektoren und Punkte sollten auch keine Sub- bzw. Superklassen voneinander sein.

```
public interface Shape {
  Shape offset(Vector v);
  boolean contains(Point p);
}
```

Abbildung 1: Das Interface Shape.

für die folgenden Operationen:

$+:$	$Punkt \times Vektor \mapsto Punkt$	Versetzt einen Punkt um einen Vektor.
$-:$	$Punkt \times Vektor \mapsto Punkt$	Versetzt einen Punkt um einen Vektor.
$-:$	$Punkt \times Punkt \mapsto Vektor$	Berechnet den Vektor zwischen zwei Punkten.
$+:$	$Vektor \times Vektor \mapsto Vektor$	Addiert zwei Vektoren.
$-:$	$Vektor \times Vektor \mapsto Vektor$	Subtrahiert zwei Vektoren.
$\|.\|:$	$Vektor \mapsto double$	Berechnet die Länge eines Vektors.

Benutzen Sie die Vektor Klassen in der Implementierung von Ellipse und Rectangle. Achten Sie darauf, Codeduplizierung zu vermeiden. Erweitern Sie auch die Testfälle entsprechend.

Wichtig: Lassen Sie die Signatur der aus `ex01_2` übernommenen Ellispe-Methoden unverändert. Nutzen Sie stattdessen Überladung.[2] Dadurch sind die alten Tests immer noch nützlich und sollten auch erfolgreich durchlaufen.[3]

Interfaces Die beiden Klassen haben Gemeinsamkeiten. Wie Sie sich vermutlich schon gedacht haben, können hier Interfaces verwendet werden, um die Figuren unter einem gemeinsamen Typen zusammenzufassen.

Implementieren Sie das in Abbildung 1 umrissene Interface in Ellipse und Rectangle. Auch hier gilt: Passen Sie weder existierende Methodensignaturen, noch Tests an.

Exercise 2 (Speisekarte, 8 Punkte)
Projektverzeichnis: `ex02_2`. Package: `menu`.

In manchen Restaurants gibt es keine herkömmliche Speisekarte auf Papier mehr. Stattdessen bieten diese Restaurants ihren Gästen Tablets an, auf denen sie die angebotenen Speisen und Getränke ansehen und bestellen können.

Ihre Aufgabe ist es nun, die Repräsentation der Speisekarte

1. zu designen und dann
2. dieses Design zu implementieren.

Nicht zu bearbeiten brauchen Sie den Bestellprozess (d.h. die Interaktion mit dem Gast zur Auswahl der Speisen und die Meldung an die Küche) und die grafische Darstellung. Um diese würden sich andere Entwickler kümmern und dabei ihre Repräsentation nutzen. Dabei erzeugen und manipulieren sie die Elemente der Speisekarte nur über Ihre Methoden und fragen auch Eigenschaften nur über Ihre Methoden ab.

1. Design

Die Anforderungen an Ihre API sind:

- Für jede Speise und jedes Getränk den Namen und den Einzelpreis zurückgeben
- Für jede Speise und jedes Getränk die enthaltenen Allergene zurückgeben (um Richtlinie 2003/89/EG einhalten zu können)

Ihre Aufgabe ist es nun, ein Datenmodell zu erstellen und für alle Methoden die Signatur anzugeben. Zeichnen Sie dazu die Klassendiagramme.

Bisher haben Sie die absolut notwendige Basisfunktionalität designt. Das Restaurant möchte sich aber von der Konkurrenz absetzen. Überlegen Sie sich noch drei weitere Anforderungen an ihre Speisekarte, beschreiben Sie diese kurz in einer Text-Datei `ex02_2-Erweiterungen.txt`,

[2] In Java ist es möglich, mehrere Methoden mit gleichem Namen aber unterschiedlichen Argument-Typen in einer Klasse zu implementieren. Dieses Feature wir als "Überladung von Methoden" bezeichnet.

[3] Das heißt nicht, dass Sie auf weitere Tests für neue Funktionalität verzichten dürfen.

und arbeiten sie diese in das Klassendiagramm ein. Vergessen Sie nicht die Datei `ex02_2-Erweiterungen.txt` auch in ihr Repository zu committen.

Hinweise und Ideen:

- Die Speisekarte in verschiedene Kategorien (Vorspeisen, Hautpgerichte, Getränke, ...) aufteilen
- Empfehlung für ein Gericht (z.B. spezielle Tagesgerichte, usw)
- Weinempfehlung zu einem Gericht
- Beschreibungen der Gerichte, um sie dem Gast schmackhaft zu machen
- auf vegetarische, ovo-lakto-vegetarische, vegane, ... Gäste eingehen
- Die Preise der Speisekarte wahlweise in verschiedenen Währungen anzeigen lassen

2. Implementierung

Implementieren Sie Ihr Design. Achten Sie darauf, dass sich die Implementierung genau so verhält, wie Ihre Dokumentation es beschreibt. Ihr Tutor wird auch kontrollieren, ob Sie Ihr Design umgesetzt haben, oder ob Sie davon abgewichen sind.

Schreiben Sie, wie üblich, sinnvolle Javadoc-Kommentare; es soll möglich sein Ihre Speisekarten API[4] anhand der Kommentare und Methodensignaturen zu verwenden ohne die Implementierung der Methoden kennen zu müssen.

3. Testen

Testen Sie Ihre Implementierung bezüglich ihrer Anforderungen, unter anderem auch mit einer konkreten Speisekarte mit mindestens drei Vorspeisen, drei Hauptgänge und drei Desserts, sowie fünf Getränke.

Exercise 3 (Newsfeed, 6 Punkte)
Projektverzeichnis: `ex02_3`. Package: `newsfeed`.

In dieser Aufgabe sollen Sie einen Prototypen einer Newsfeed-Komponente, wie sie in soziales Netzwerken zum Einsatz kommt, implementieren. Ein Newsfeed ist eine Liste von Nachrichten, wie Sie beispielsweise auf dem Bildschirm erscheint wenn ein Benutzer sich in ein soziales Netzwerk einloggt. In unserem Newsfeed erscheinen drei Arten von Einsendungen: Text-, Foto- und Ereigniseinsendungen. Die Newsfeed-Komponente sollte mindestens die folgende Funktionalität unterstützen:

- Hinzufügen von Einsendungen beliebiger Art
- Anzeigen der bisher eingefügten Einsendungen mit allen Details
- Manipulieren der veränderlichen Eigenschaften von Einsendungen, wie *likes* und Kommentare (s.u.)

Es folgt nun die Spezifikation der drei verschiedenen Arten von Einsendungen:

Texteinsendungen enthalten

- den Benutzernamen des Autors
- den Text der Nachricht
- den Zeitpunkt ihrer Erstellung
- wie vielen Lesern die Einsendung gefällt (*likes*)
- eine Liste der Kommentare zu dieser Einsendung von anderen Benutzern

Fotoeinsengungen enthalten

- den Benutzernamen des Autors
- der Dateiname des anzuzeigenden Bildes
- die Überschrift für das Foto (eine Textzeile)
- den Zeitpunkt ihrer Erstellung
- wie vielen Lesern die Einsendung gefällt

[4] API steht für *Application Programming Interface*. Hier meinen wir damit eine genaue Beschreibung der Klassen und Methoden, die eine Bibliothek einem Anwendungs-Programmierer zur Verfügung stellt. Siehe auch `http://en.wikipedia.org/wiki/Application_programming_interface`

- eine Liste der Kommentare zu dieser Einsendung von anderen Benutzern

Ereigniseinsendungen enthalten

- den Benutzernamen des Autors
- den Ereignistyp (Anmeldung, Abmeldung, ...)
- den Zeitpunkt ihres Auftretens

Vermeiden Sie bei der Implementierung duplizierten Code, indem Sie abstrakte Basisklassen und Vererbung benutzen. Achten Sie beim Aufbau einer Vererbungshierarchie darauf, dass keine Klasse Felder und Methoden enthält, die ihr nicht benötigt werden oder unsinnig sind.

Testen Sie Ihren Newsfeed in der main-Methode. Der Einfachheit halber können die Details einer Einsendung auf der Konsole (anstatt in einem Webbrowser) ausgegeben werden.

Eine mögliche Ausgabe könnte so aussehen:

```
vor 2 Stunde(n)
The globetrotter
hat sich angemeldet...

vor 10 Minuten
The globetrotter
  -  4 Person(en) gefällt das.
   3 Kommentar(e). Zum Einsehen hier klicken.
Hallo! Schöne Grüße aus Thailand. Die ersten Tage verbringe ich in Bangkok...

vor 35 Sekunden
The globetrotter
  -  3 Person(en) gefällt das.
   Keine Kommentare.
  [bangkok11.jpg]
  Statue im Wat Pho
```

Bezüglich der *likes* und Kommentarausgabe genügt es, die bloße Anzahl der vorhandenen *likes* bzw. Kommentare anzugeben. Weitere Informationen können angedeutet werden (wie z.B. mit dem Satz „Zum Einsehen hier klicken“). Die Ausgabe des Zeitraumes kann in Tagen, Stunden, Minuten oder Sekunden erfolgen. Verwenden Sie hierfür keine primitiven Datentypen, sondern z.B. die Klassen `java.time.Instant` bzw. `java.time.Duration`.

Die Formatierung der Ausgabe im Detail ist Ihnen überlassen. Achten Sie jedoch auf eine sinnvolle Anordnung der einzelnen Informationsbausteine.

Zwei Hinweise zur Java-API

Testen von `double`-Werten `double`- und `float`-Werte sind nur Näherungen. Wenn Sie in JUnit-Tests ein `double`-Ergebnis prüfen wollen, verwenden Sie Toleranzangaben wie hier:

```
1  double res = mySquareRoot(0.25);
2  //FALSCH: assertEquals(0.5, res); // scheitert an Rundungsfehlern
3  assertEquals( 0.5, res,
4  0.000001 ); // <-- akzeptiert Rundungsfehler bis 0.000001
```

Das Interface `Set<T>` In der Vorlesung wurde das `List`-Interface besprochen. Listen reichen im Prinzip auch für das Lösen der Aufgaben dieses Blattes aus.

Java stellt allerdings auch noch andere Collection-Interfaces zur Verfügung, die Sie gerne auch schon in diesem Blatt verwenden dürfen; Sie müssen die genaue Funktionalität aber selbstständig in der Java-API nachlesen.

Neben `List<T>` ist auch `Set<T>` besonders nützlich. `Set<T>` bezeichnet Mengen des Typs (d.h. der Klasse, oder des Interfaces) `T`. Wie Sie wissen unterscheiden sich Mengen von Listen dadurch, dass sie keine doppelten Elemente enthalten. Die wichtigsten Methoden des Interfaces sind `add`, `remove` und `contains`. Ob ein Element doppelt ist oder nicht wird mit der `equals`-Methode bestimmt, die in der Vorlesung besprochen wurde. Die Klasse `HashSet<T>` bietet Beispielsweise eine Implementierung von `Set<T>`.