Softwareentwicklung 1 – Praktikum

Kontrollstrukturen und numerische Datentypen

Abgabe bis 20.12.2016, 23:00 Uhr

Ziele

- Erstellen von Klassen (Konstruktoren, Objektvariablen, Methoden)
- Erzeugen von Objekten und Aufrufen von Methoden
- Schriftliches Lösen von Programmieraufgaben

Hinweis

Sie finden für jede der Aufgaben ein vorbereitetes Greenfoot Szenario in meinem GitHub Projekt **karatest**. Für jedes Aufgabenblatt können Sie die jeweils aktuelle Version der Aufgaben in einem Archiv herunterladen:

https://github.com/uhafner/karatest/archive/assignment5.zip

Bitte nutzen Sie immer diese vorgegebenen Szenarien zur Lösung der Aufgaben. Verwenden Sie insbesondere immer die dort bereits enthaltenen Aufgabendateien für Ihre Lösung, nur so können wir Ihre Lösungen automatisiert auswerten. Ebenso dürfen Sie die Verzeichnisstruktur nicht umändern.

Erlaubte Elemente zur Umsetzung

Zur Lösung der Aufgaben dieses Blattes dürfen alle in der Vorlesung behandelten Java Sprachmittel genutzt werden.

Lösen der Aufgabe auf Papier

Das Übungsblatt ist so gestaltet, dass es sich vom Umfang und Schwierigkeit gut auf Papier (ohne Greenfoot) lösen lässt. Versuchen Sie daher, das Blatt möglichst komplett auf Papier zu entwerfen, bevor Sie den Quelltext schreiben.

Aufgabe 15 (ca. 20 Punkte)

Schreiben Sie eine kleine Grafikbibliothek, mit deren Hilfe Sie grafische Elemente in Karas Welt zeichnen können. Jedes grafische Element wird durch eine eigene Klasse dargestellt. Setzen Sie die folgenden Klassen um:

- **Pixel**: Ein Pixel (Picture Element) repräsentiert ein einzelnes Blatt in Karas Welt. Ein Pixel wird über die x- und y-Koordinate definiert (jeweils als **int**).
- **Line**: Eine Gerade in Karas Welt kann entweder horizontal oder vertikal sein. Eine Gerade wird über den Startpunkt (als **Pixel**) und die Länge (als **int**) definiert.
- **Box**: Ein Kasten kann entweder ausgefüllt sein, oder nicht. Ein Kasten wird über den Startpunkt (als **Pixel**) und die Breite und Höhe (jeweils als **int**) definiert.

Entwickeln Sie alle drei Klassen vollständig als **immutable Classes**. D.h. eine Instanz einer dieser Klassen ist konstant und nicht mehr änderbar. Jede Klasse besteht mindes-

tens aus den oben genannten Eigenschaften, ein oder mehreren Konstruktoren und einer vordefinierten Menge an Methoden. Die Anforderungen an die Klassen sind in den folgenden Abschnitten ausschnittsweise beschrieben. Nutzen Sie die beigelegten Tests, um die komplette Funktionalität Ihrer Klassen zu prüfen. Die Tests finden Sie im Ordner **src/test/java** des Projektes.

Hinweis: Eine Fehlerbehandlung für ungültige Aufrufe der Konstruktoren (negative Koordinaten, Längen <= 0, etc.) und Methoden ist nicht erforderlich. Setzen Sie voraus, dass alle Ihre Konstruktoren und Methoden immer korrekt aufgerufen werden.

Aufgabe 15a – Vorbereitung

Zum Testen Ihrer drei Klassen gibt es einige Demonstrationsaufrufe in der Klasse **Assignment15**. Im Verzeichnis **src/test/java** des Projektes gibt es zusätzlich für jede der drei Klassen eine zugeordnete Testklasse. Beim ersten Start von Greenfoot bzw. beim Compilieren der Testklassen wird Ihr Projekt mit vielen Compile-Fehlern angezeigt:

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
```

Beseitigen Sie diese im ersten Schritt, indem Sie alle erforderlichen Objektvariablen, Konstruktoren und Methoden leer (bzw. mit einem Default Rückgabewert) anlegen (siehe folgende Seiten).

Sie haben alle Compile-Fehler behoben, wenn der Aufruf **mvn compile** so endet:

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

Aufgabe 15b – Pixel

Ein Punkt hat einen 2-Parameter Konstruktor (x- und y-Koordinate). Einen neuen Punkt an der Stelle x=2 und y=4 erzeugt der Aufruf

new Pixel(2, 4)

Ein Punkt hat sowohl einen Getter für die x- als auch einen für die y.-Koordinate, das heißt die Aufrufe sind möglich:

new Pixel(2, 4).getX() ergibt 2
new Pixel(2, 4).getY() ergibt 4

Ein Punkt kann einen um ein x- und y-Offset verschobenen neuen Punkt erzeugen: ein Aufruf von

new Pixel(2, 4).move(-1, 2) liefert einen neuen Punkt an der Stelle (1, 6) zurück

Ein Punkt stellt eine **equals** Vergleichsmethode und **hashCode** Methode zur Verfügung, übernehmen Sie diese unverändert aus der Vorlage.

Aufgabe 15c – Line

Eine Gerade hat einen 2-Parameter Konstruktor. Der Aufruf

new Line(new Pixel(2, 4), 5)

erzeugt eine neue **horizontale** Gerade der Länge 5 am Startpunkt (2, 4).

Eine Gerade hat zusätzlich einen 3-Parameter Konstruktor. Der Aufruf

new Line(new Pixel(2, 4), 5, false)

erzeugt eine neue **vertikale** Gerade der Länge 5 am Startpunkt (2, 4). Für eine horizontale Gerade ist der dritte Parameter **true**.

Eine Gerade kann einen um ein x- und y-Offset verschobene neue Gerade erzeugen: ein Aufruf von

new Line(new Pixel(2, 4), 5).move(-1, 2) liefert eine neue Gerade der Länge 5 beginnend vom Punkt (1, 6) zurück

Eine Gerade bietet eine Methode **contains**, um zu überprüfen, ob ein übergebener Punkt auf der Geraden liegt oder nicht:

new Line(new Pixel(2, 4), 5).contains(new Pixel(3, 4)) ergibt **true**
new Line(new Pixel(2, 4), 5).contains(new Pixel(2, 5)) ergibt **false**

Eine Gerade bietet eine Methode **intersects**, um zu überprüfen, ob eine übergebene andere Gerade sich mit dieser Geraden schneidet:

new Line(new Pixel(2, 4), 5).intersects(new Line(new Pixel(3, 3), 2, false) ergibt **true**
new Line(new Pixel(2, 4), 5).intersects(new Line(new Pixel(1, 3), 2, false) ergibt **false**

Eine Gerade bietet eine Methode **getPixels**, die ein Array mit allen Punkten der Geraden zurückliefert. (Eine Sortierung der Punkte ist nicht erforderlich.)

Aufgabe 15d – Box

Ein Kasten hat einen 3-Parameter Konstruktor. Der Aufruf

new Box(new Pixel(2, 4), 5, 2)

erzeugt eine neue **Box** der Breite 5 und der Höhe 2 am Startpunkt (2, 4). Die Box ist nicht ausgefüllt.

Ein Kasten hat zusätzlich einen 4-Parameter Konstruktor. Der Aufruf

new Box(new Pixel(2, 4), 5, 2, true)

erzeugt eine neue ausgefüllte **Box** der Breite 5 und der Höhe 2 am Startpunkt (2, 4). Für eine nicht ausgefüllte Box ist der vierte Parameter **false**.

Ein Kasten kann einen um ein x- und y-Offset verschobenen neuen Kasten erzeugen: ein Aufruf von

new Box(new Pixel(2, 4), 5, 2).move(-1, 2) liefert einen neuen Kasten der gleichen Dimension und Art beginnend vom Punkt (1, 6) zurück

Ein Kasten bietet eine Methode **contains**, um zu überprüfen, ob ein übergebener Punkt auf oder im Kasten liegt oder nicht:

new Box(new Pixel(2, 4), 5, 3).contains(new Pixel(6, 6)) ergibt **true**
new Box(new Pixel(2, 4), 5, 3).contains(new Pixel(3, 5)) ergibt **false**
new Box(new Pixel(2, 4), 5, 3, true).contains(new Pixel(3, 5)) ergibt **true**

Ein Kasten bietet eine Methode **intersects**, um zu überprüfen, ob ein übergebener anderer Kasten sich mit diesem Kasten schneidet:

new Box(new Pixel(0, 0), 3, 5).intersects(new Box(new Pixel(2, 4), 5, 2) ergibt **true**
new Box(new Pixel(0, 0), 2, 4).intersects(new Box(new Pixel(2, 4), 5, 2) ergibt **false**
new Box(new Pixel(0, 0), 4, 4).intersects(new Box(new Pixel(1, 1), 2, 2) ergibt **false**
new Box(new Pixel(0, 0), 4, 4, false).intersects(new Box(new Pixel(1, 1), 2, 2, false) ergibt **true**

Ein Kasten bietet eine weitere Methode **intersects**, um zu überprüfen, ob ein übergebene Gerade sich mit diesem Kasten schneidet.

Ein Kasten bietet eine Methode **getPixels**, die ein Array mit allen Punkten des Kastens zurückliefert. (Eine Sortierung der Punkte ist nicht erforderlich.)