Shapes-Projekt – Grundlagen der Programmierung 2 SoSe 2018

Oliver Arnold, Wolfgang Golubski, Frank Grimm, Ina Klar

Das Projekt soll als Teil der Prüfungsvorleistung in den nächsten Wochen schrittweise durch jeweils zwei Studierende gemeinsam bearbeitet werden. Die Abgabe erfolgt über den Projektserver bis zum 11.04.2018 um 09:00 Uhr.

Folgende Rahmenbedingungen sind dabei zwingend einzuhalten:

- **Die Abgabe wird ausschließlich über das entsprechende Repository auf dem Projektserver bis zum angegebenen Abgabedatum akzeptiert. Machen Sie die finale Version mit einem entsprechenden Commit-Kommentar kenntlich. Ansonsten wird die letzte vor Ablauf der Frist hochgeladene Version zur Bewertung herangezogen.**
- **Aufgrund regelmäßiger Commits auf dem Projektserver muss nachvollziehbar sein, dass beide Studierende hinreichend an der Entwicklung beteiligt waren. Beispielsweise wird ein großer „Gesamt-Commit" des vollständigen Projekts am Ende der Projektbearbeitungszeit durch einen der Studierenden nicht akzeptiert.**

Projektaufgabe

Entwickeln Sie mit Eclipse ein neues Projekt namens Shapes, in dem eine Vererbungshierarchie von geometrischen Formen realisiert wird. Legen Sie dazu zunächst ein Paket namens `shapes` an.

Nutzung der Klasse WhiteBoard

Zum Zeichnen der geometrischen Objekte sollen Sie die Klasse `WhiteBoard` verwenden, die auf den grafischen Ausgabemöglichkeiten der Java-Bibliothek Swing aufsetzt. Diese wird als Bibliotheksdatei *whiteboard.jar* zur Verfügung gestellt. Legen Sie die JAR-Datei (am besten) im Projektverzeichnis Ihres Projekts in einem neu angelegten Ordner *lib* ab. Binden Sie die Datei dann als Library in Ihr Eclipse-Projekt ein. Rechtsklicken Sie dazu in Eclipse auf die JAR-Datei im *lib*-Verzeichnis Ihres Projekts und wählen Sie *Buildpath → Add to Build Path*. Nun können Sie in Ihren Klassen die Zeichenfunktionen des Whiteboards verwenden. Da die Klasse `WhiteBoard` im Paket `teaching` abgelegt ist, müssen Sie sie im Code entweder mit dem voll qualifizierten Klassennamen `teaching.WhiteBoard` ansprechen oder das Paket `teaching` mit der Anweisung `import teaching.*` in den Namensraum Ihrer Klassen einbinden.

Umsetzung der Vererbungshierarchie

Nun zur eigentlichen Aufgabe. Im Paket `shapes` sollen Sie eine Vererbungshierarchie umsetzen. An deren Spitze soll das Interface `Drawable` stehen. Es definiert zwei abstrakte Methoden, die alle darunterliegenden Klassen implementieren müssen.

Die weiteren Klassen entnehmen Sie bitte dem Bild und der API. Als Ausgangspunkt für die API bekommen Sie dazu eine Klassendokumentation, die alle Elemente beschreibt.

Bitte fragen Sie rechtzeitig nach, insbesondere wenn sich Fragen zum Klassendiagramm ergeben, die sich nicht aus der Klassendokumentation erklären lassen.

Hinweis zum Klassendiagramm: Neben den bereits aus GdP1 bekannten Vererbungs- und Interface-Realisierungspfeilen finden sich im Diagramm auch sogenannte Assoziationspfeile, beispielsweise der mit „center" bezeichnete Pfeil zwischen den Klassen `Circle` und `Point`. Dieser Pfeil bedeutet, dass jedes Objekt der Klasse `Circle` ein Attribut namens „center" vom Datentyp `Point` besitzt. Die

Angabe „0..1“ bedeutet, dass es nur ein einfaches Attribut ist, keine Liste oder ähnliches (was auch inhaltlich für den Mittelpunkt eines Kreises Sinn macht). Die Angabe „0..*“ (beispielsweise beim Pfeil zwischen `Polygon` und `Point`) zeigt hingegen an, dass auf mehrere Objekte eines bestimmten Typs verwiesen wird (in diesem Fall also auf alle Punkte, die zum Polygon gehören). In der Umsetzung müsste hier also in der Klasse `Polygon` beispielsweise eine Liste von `Point`-Objekten verwendet werden. Die Zeichen vor dem Namen der Assoziation zeigen die Sichtbarkeit des entsprechenden Attributs an. „-“ steht für `private`, „#“ für `protected` und „+“ für `public`.

Die Assoziationspfeile sind also eine alternative Darstellung der Attribute in den Klassen, durch die eine bessere Übersicht über die Zusammenhänge zwischen den Klassen gewonnen wird.

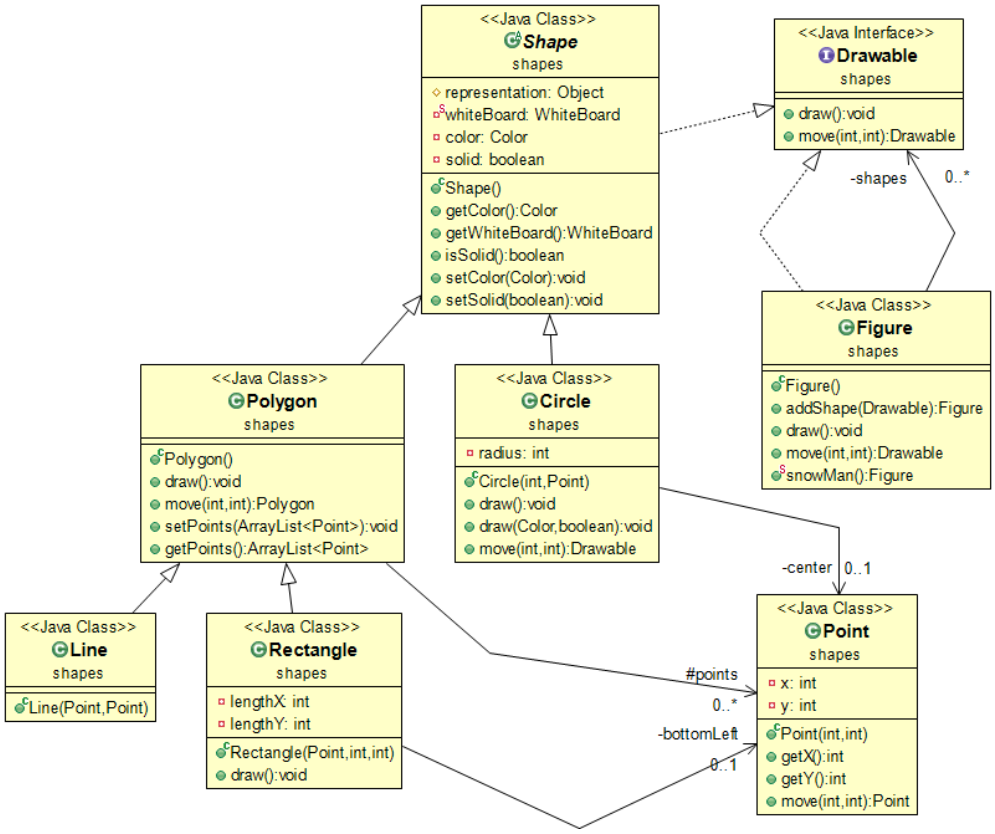

Abbildung 1: Vererbungshierarchie im Shapes-Projekt

Woche 1

Realisieren Sie im ersten Schritt das Interface `Drawable`, die abstrakte Klasse `Shape` sowie die Klassen `Point`, `Line`, `Rectangle`, `Circle` und `Polygon`.

Beachten Sie folgende Aspekte:

- Das `WhiteBoard` wird in der Klasse `Shape` verwaltet und es existiert nur ein Exemplar vom Typ `WhiteBoard`. Verwenden Sie hierzu in der Klasse `Shape` eine private statische Variable, die ein solches Exemplar instanziiert.
- Die `draw()`-Methoden sind nach folgendem Schema aufgebaut:

```
// Entfernen des Objekts an der alten Position,
// sofern vorhanden
getWhiteBoard().removeShape(representation);

// Aufbereiten der nötigen Informationen für das Zeichnen
// . . .
```

```
// Zeichnen des Objekts (hier: Polygon) an der neuen
// Position und „Aufbewahren“ des representation-Objekts
representation = getWhiteBoard().drawPolygon(x, y);
```

Realisieren Sie nun die Klasse `Figure` und ihre Methode `snowMan()`, durch die ein Schneemann auf dem Whiteboard gezeichnet wird. Ein Beispiel dazu findet sich in Abbildung 2. Weitere kreative Energie bei der Gestaltung des Schneemanns darf gerne ausgelebt werden. ☺

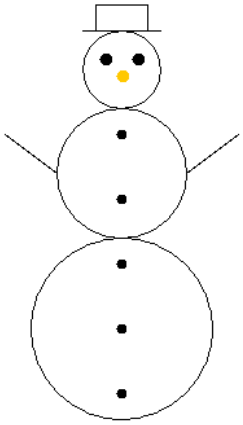

Abbildung 2: Ein Beispiel für einen mit den Shapes gezeichneten Schneemann

Ergänzen Sie Ihr Projekt um eine Klasse `Start`, in der Sie die `main`-Methode wie bereits in GdP1 erläutert anlegen. In diese Methode schreiben Sie Quellcode, mit dem Sie ihre entwickelten Klassen ausprobieren, d.h. Sie legen Instanzen der Klassen an, rufen deren Methoden auf und prüfen so, ob die entwickelten Klassen korrekt funktionieren.

Woche 2 (Exceptions)

Erweitern Sie mit Eclipse die Klasse `Circle` in der Art, dass eine `IllegalArgumentException` geworfen wird, wenn ein Konstruktor mit unsinnigen Werten (z.B. einem negativen Radius oder einer Null-Referenz als Mittelpunkt) aufgerufen wird. Rufen Sie den Konstruktor mit verschiedenen Werten auf, um die Exception auf alle möglichen (sinnvollen) Arten auszulösen und so die korrekte Funktionsweise zu überprüfen.

In der Klasse `Polygone` soll die Methode `setPoints(List<Point> p)` so modifiziert werden, dass eine Exception geworfen wird, wenn die `List` nicht mindestens zwei Punkte enthält. Schreiben Sie dazu eine eigene Exception-Klasse `PolygoneShapeException`, die von `Exception` (nicht von `RuntimeException`!) abgeleitet ist. Welche Änderungen müssen Sie dazu an verschiedenen Stellen des vorhandenen Codes durchführen, um das Projekt wieder fehlerfrei übersetzen und ablaufen lassen zu können? Überprüfen Sie auch die korrekte Funktion der Exception in `Polygone`.

Ergänzen Sie auch den Rest des Projekts um sinnvoll eingesetzte Exceptions.

Ergänzen Sie die `main`-Methode in Ihrer `Start`-Klasse um Überprüfungen der Exceptions.