Informatik

2

Übungsaufgaben

Prof. Dr.-Ing. Holger Vogelsang

Sommersemester 2014

Don’t panic

Inhaltsverzeichnis

1

Einleitung

1.1 Organisation der Übungsaufgaben

Jeder von Ihnen muss seine eigene Lösung vorstellen. Zur Erlangung des Scheins sind wichtig:

- korrekte Funktionsweise der Implementierung
- vollständige Kommentierung mit Javadoc (außer bei einfachen Getter- und Setter-Methoden)
- Einhaltung der Code-Konventionen von Oracle: Diese müssen mittels des Plugins für Checkstyle überprüft werden (siehe Abschnitt 2.1.1).
- Prüfen Sie Ihr Programm mit dem Plugin „Findbugs“ auf mögliche Schwachstellen (siehe Abschnitt 2.2.1).

Die Plugins „Checkstyle“ und „Findbugs“ gibt es für die Entwicklungsumgebungen Eclipse, Netbeans und IntelliJ IDEA.

1.2 Anerkennung bereits abgegebener Lösungen

Sollte jemand von Ihnen schon in früheren Semestern Lösung abgegeben haben, so werden diese anerkannt.

Tabelle 1.1: Anrechnung

im WS 2012/2013	**im SS 2013**	**im WS 2013/2014**	**entspricht im SS 2014**
3.1	3.1	3.1	3.1
3.2	3.2	3.2	3.2
3.3 – 3.4	3.3 – 3.5	3.3 – 3.5	3.3 – 3.5
3.5	3.6	3.6	3.6
3.6	3.7	3.7	3.7
3.7 (basiert auf 3.1)	3.8	3.8	3.8

1.3 Vorbereitungen

Das Kapitel beschreibt die Installation der notwendigen Programme für das Labor.

1.3.1 Installation des JDK unter Windows

Zur reinen Ausführung von Java-Programmen genügt das JRE (Java Runtime Environment). Darin fehlen aber beispielsweise wichtige Werkzeuge wie der Compiler oder das Programm `javadoc`. Hier in der Vorlesung sollen eigene Programme erstellt werden. Daher muss das JDK (Java Development Kit, manchmal auch SDK genannt) mindestens in Version 7 verwendet werden. Installation:

- Download des JDK kostenlos unter
  http://www.oracle.com/technetwork/java/index.html
- Doppelklick auf das ausführbare JDK-Archiv, dann das Zielverzeichnis angeben
- Die dringend benötigte Dokumentation muss separat installiert werden. Sie sollte in das Verzeichnis der JDK-Software oder in ein eigenständiges Verzeichnis entpackt werden.
- Wenn das Verzeichnis der ausführbaren Programme nicht im Suchpfad liegt, sollte das Verzeichnis in die PATH-Variable aufgenommen werden. Die Programme können dann von der Kommandozeile aus aufgerufen werden (z.B. die Abfrage der JDK-Versionsnummer mit `java -version`). Das ist für die Übungen aber nicht erforderlich.

1.3.2 Integrierte Entwicklungsumgebungen

Die Anwendung des „nackten“ JDKs ist nicht sehr komfortabel. Daher bietet sich die Verwendung einer integrierten Entwicklungumgebung an. Auswahl:

- Eclipse: alle Plattformen, kostenlos unter http://www.eclipse.org.
- Netbeans: Oracle, alle Plattformen, kostenlos unter
  http://netbeans.org/index.html
- IntelliJ IDEA: Jetbrains, alle Plattformen, kostenlose Community-Edition unter
  http://www.jetbrains.com/idea/
- JDeveloper: Oracle, alle Plattformen, kostenlos unter
  http://www.oracle.com/technetwork/java/index.html

Auf den Poolrechnern sind Eclipse und Netbeans installiert. Als Projektverzeichnis sollte ein Verzeichnis auf dem Netzlaufwerk gewählt werden. Das erspart das manuelle Backup! Das später aus Eclipse heraus benötigte Progamm `javadoc` wird von Eclipse nicht immer automatisch gefunden. Es muss mit dem Java-Pfad eingebunden werden. Ihr Betreuer hilft Ihnen dabei. Am besten aber wäre es, wenn Sie Ihre Projekte direkt in Ihrem SVN-Repository ablegen. Eine Anleitung dazu finden Sie in den Vorlesungsunterlagen.

2

Hinweise zur Arbeit mit Eclipse

2.1 Checkstyle

Alle Lösungen müssen die Code-Konventionen von Oracle einhalten. Die Überprüfung übernimmt das Eclipse-Plugin für „Checkstyle“. Nur wenn keine Warnungen in Form kleiner gelber Ausrufezeichen ausgegeben werden, wird eine Aufgabe abgenommen. In den Pools ist „Checkstyle“ bereits installiert.

2.1.1 Checkstyle-Installation auf privaten Computern

Sie können das Plugin unter Eclipse mit `Help` → `Eclipse Marketplace` installieren. Dort geben Sie im Feld `Find` den Namen „Checkstyle“ ein und lassen den gefundenen Treffer installieren.

2.1.2 Globale Checkstyle-Konfiguration

Die Standardkonfiguration von Checkstyle ist relativ „scharf“ eingestellt. Damit Sie nicht komplett verzweifeln, können Sie aus dem Ilias von den Seiten zu den Übungen eine entschärfte Konfigurationsdatei herunterladen (`Checkstyle-Informatik2`) und irgendwo im lokalen Dateisystem abgelegen. Weitere Schritte in Eclipse:

- In Eclipse wird das Menü `Window` → `Preferences` ausgewählt.
- In der linken Baumansicht muss `Checkstyle` selektiert werden. Danach erfolgt ein Mausklick im Dialog auf den Button `New...`.
- Im Dialog `External Configuration File` auswählen.
- Die Informatik2-Konfiguration einlesen, indem `Browse` ausgewählt wird.
- Der Konfiguration einen Namen geben (z.B. `Informatik2`).
- Den Dialog mit `OK` schließen und die neue Konfiguration mit `Set as Default` zum Standard erklären.

2.1.3 Projekt-spezifische Checkstyle-Konfiguration

In jedem Projekt muss Checkstyle explizit eingeschaltet werden:

- mit der rechten Maustaste auf das Projekt klicken und `Properties` auswählen
- `Checkstyle` wählen
- im erscheinenden Dialog die gewünschte Konfiguration auswählen
- dann den Haken vor `Checkstyle active for this project` setzen und mit `OK` bestätigen
- Regelverstöße werden im Projekt durch gelbe Markierungen am Rand angezeigt: 

  Besitzt ein Projekt solche Verstöße, so werden die betroffenen Klassen auch im Package-Explorer markiert.

2.2 FindBugs

FindBugs analysierten Ihren Quelltext und versucht so, mögliche Fehlerquellen anzuzeigen. Nicht jede Markierung muss einen echten Fehler darstellen. Teilweise kann es sich auch nur um Hinweise handeln, dass hier in bestimmten Situationen Fehler aufteten können. Wenn Sie sicher sind, dass solche Situationen bei Ihnen nie auftreten können, dann ignorieren Sie diese Hinweise. Außerdem kann es vorkommen, dass FindBugs Stellen als fragwürdig markiert, obwohl sie absichtlich so erstellt wurden. Ein Beispiel dafür ist der Stringvergleich durch Vergleich der Referenzen auf die String-Objekte, anstatt die `equals`-Methode zu verwenden. Hier kann FindBugs nicht erkennen, ob Sie das absichtlich oder unabsichtlich gemacht haben. Sehen Sie die Meldungen von FindBugs einfach als Anregung, über den Code nachzudenken.

2.2.1 FindBugs-Installation

Sie können das Plugin unter Eclipse mit `Help` → `Eclipse Marketplace` installieren. Dort geben Sie im Feld `Find` den Namen „FindBugs“ ein und lassen den gefundenen Treffer installieren. Sollte mehr als ein Treffer sichtbar werden, dann nehmen Sie denjenigen, bei dem nicht einige Angaben in der Beschreibung `null` sind.

2.2.2 Verwendung

Die Anwendung könnte nicht einfacher sein: Nach einem Rechtklick auf das Projekt wählen Sie `Find Bugs` und dann den Eintrag `Find Bugs`. In der Perspektive `Find Bugs` sehen Sie nach dem Durchlauf die Treffer. Eventuell müssen Sie manuell auf diese Perspektive umschalten. Hinter dem Projektnamen erscheint im „Package Explorer“ in Klammern die Anzahl möglicher Probleme.

2.2.3 Konfiguration

FindBugs lässt sich ziemlich gut konfigurieren, so dass Sie bestimmte Fehlerarten ein- oder ausschalten können. Die Standard-Konfiguration ist für uns schon nicht schlecht. Sie können aber unter Umständen unter `Window` → `Preferences` → `FindBugs` die Option `Multithreaded Correctness` noch ausschalten, weil das Thema Multithreading erst in der Betriebssystemvorlesung behandelt wird und Sie hier mit möglichen Gefahrenmeldungen in diesem Zusammenhang nichts anfangen können.

2.2.4 FindBugs in den Pool-Installationen

Sollte FindBugs auf den PCs bzw. virtuellen Maschinen der Pools keine Fehler melden, dann ist unter Umständen ein kleiner manueller Eingriff in Ihr Projekt erforderlich:

1. Nach einem Rechtsklick auf Ihr zu untersuchendes Projekt wählen Sie `Properties` und dort im Dialog `FindBugs`.
2. Selektieren Sie `Enable Project Specific Settings`.
3. Selektieren Sie alle Einträge in dem eingerahmten Kästchen `Report Visible Bug Categories`.
4. Verschieben Sie den Schieberegler `Minium rank to report` auf den Wert 20.
5. Stellen Sie den Wert von `Minimum confidence to report` auf `Low`.
6. Prüfen Sie mit FindBugs Ihr Projekt erneut.

2.3 Kontextsensitive Hilfe

Um die kontextsensitive Hilfe unter Eclipse richtig nutzen zu können, sind einige kleinere Vorarbeiten erforderlich.

2.3.1 Im Pool-Raum

Normalerweise bietet Eclipse eine kontextsensitive Hilfe an, wenn Sie mit dem Mauszeiger auf eine Methode, eine Klasse usw. deuten. Im Poolraum werden Sie feststellen, dass das leider nicht immer funktioniert, weil der Proxy der Hochschule dieses so nicht zulässt. Es gibt mehrere Möglichkeiten, das zu umgehen. Beispielsweise kann in Eclipse die Verwendung des Proxys eingeschaltet werden. Dazu müssen die Anmeldedaten hinterlegt werden. Leider klappt auch das nicht immer zuverlässig. Sie können aber Eclipse anweisen, eine lokal installierte API-Dokumentation zu verwenden. Unter der URL `http://www.gnostice.com/nl_article.asp?id=209` finden Sie eine genaue Beschreibung der Vorgehensweise. Den Installationsort der lokalen API-Dokumentation im Pool erfahren Sie von Ihren Betreuern. Wenn Sie eine lokale Eclipse-Installation auf Ihren privaten Computern auch so anpassen wollen, dann müssen Sie neben dem JDK auch die Hilfe-Dokumente installieren. Sie finden diese unter der länglichen URL `http://www.oracle.com/technetwork/java/javase/downloads/index.html` (siehe „Java SE 7 Documentation“).

2.3.2 Für JavaFX

Die API-Dokumentation für JavaFX werden Sie ab Aufgabe 3.3 benötigen. Leider ist diese nicht Bestandteil des JDK und muss daher Eclipse, sofern Sie die Dokumentation in Eclipse verwenden wollen, bekannt gemacht werden. Zunächst einmal muss Ihr Projekt die JavaFX-Klassen überhaupt kennen. JavaFX wird zwar seit Java 7 mit dem Java SDK ausgeliefert, ist leider aber nicht im Klassenpfad vorhanden. Sie haben mehrere Möglichkeiten, Ihrem Projekt JavaFX bekannt zu machen (diese Scritte sind nicht erforderlich, wenn Sie das Grundgerüst für Aufgabe 3.3 aus dem Ilias einsetzen):

- Sie fügen die Datei `jfxrt.jar` zum „Build-Path“ Ihres Projektes hinzu. Dazu benötigen Sie im Projekt entweder einen Verweis auf die Datei im SDK-Verzeichnis, oder Sie kopieren die Datei zunächst in Ihr Projekt und stellen anschließend eine Abhängigkeit zu dieser Kopie her.

- Eleganter ist es, eine sogenannte „User Libary“ in Eclipse zu erstellen und diese in den „Build Path“ des Projektes aufzunehmen.

Welchen Weg Sie auch gegangen sind, es fehlt auf jeden Fall die Referenz auf die JavaDoc-Dateien. Um diese bekannt zu machen gehen Sie so vor:

- Klicken Sie mit der rechten Maustaste auf die Datei `jfxrt.jar` in Ihrem Projekt und wählen Sie `Properties`.
- Wählen Sie dann links im Dialog `Javadoc Location` aus.
- Rechts im Dialog fügen Sie unter `Javadoc URL` diesen Link ein: `http://docs.oracle.com/javafx/2/api/`. Die API-Dokumentation lässt sich auch mit einigen Umwegen herunterladen (`wget`, `WinHTTrack`, ...). Dann können Sie statt einer URL, die auf den Oracle-Server verweist, auch eine URL mit Verweis auf die lokale Kopie angeben.

2.4 Unit-Tests

Bisher wurden Methoden einer Lösung in einer `main`-Methode der Klasse selbst, in einer separaten Klasse oder manuell getestet. Daraus resultieren eine Anzahl von Probleme:

- Wird die Lösung erweitert oder verändert, dann besteht die Gefahr, dass Fehler eingebaut werden.
- Die manuelle Ausführung und Überprüfung ist sehr zeitaufwändig.

Der Lösungsansatz mit `JUnit` besteht darin, Testroutinen in einer separaten Testklasse zu erstellen. Dabei wird je Testfall eine Methode geschrieben. Diese Testmethode ruft die zu prüfende auf. Das Ergebnis des Aufrufs wird automatisch mit dem Sollwert verglichen.

2.4.1 Erstellung eines Testfalles

JUnit ist bereits in Eclipse integriert. Soll eine JUnit-Testklasse beispielsweise für die Klasse `Shop` erstellt werden, so genügt ein Rechtsklick mit der Maus auf die Klasse `Shop`. Im dann erscheinenden Menü werden `New` → `JUnit Test Case` ausgewählt.

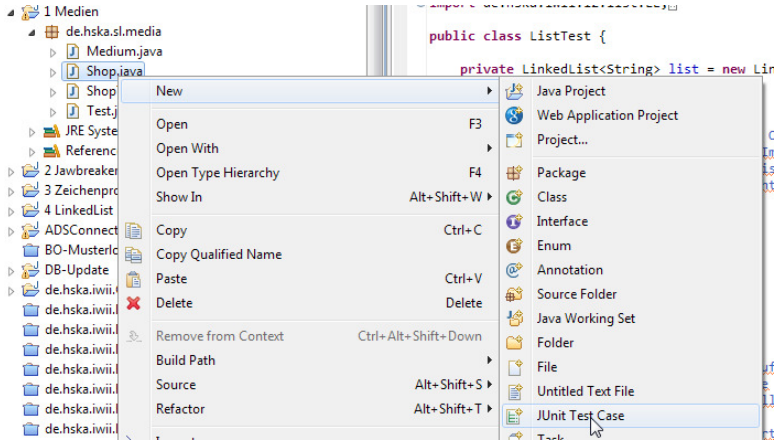

Abbildung 2.1: Testfall erstellen, Schritt 1

Es erscheint der folgende Dialog, in dem im Feld ein Name für die Testklasse vorgeschlagen wird. In der Regel sollten dieser beibehalten werden. Verwenden Sie JUnit 4 für Ihre Tests.

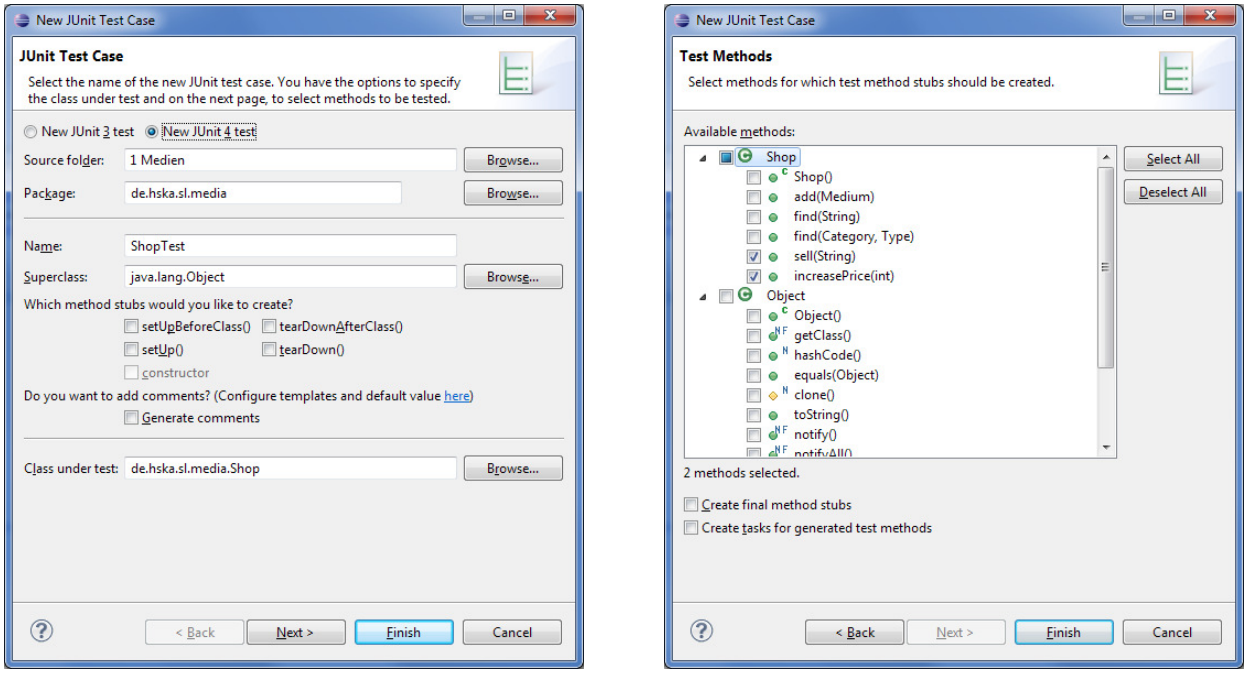

Abbildung 2.2: Testfall erstellen, Schritte 2 und 3

Mit `Next >>` erreicht man weitere Eingabemöglichkeiten. Hier werden die zu testenden Methoden ausgewählt (im Screenshot sind es `sell` und `increasePrice`.
Die Eingabe wird mit `Finish` abgeschlossen. Eventuell erscheint noch eine Rückfrage, die Sie mit `OK` beantworten sollten.

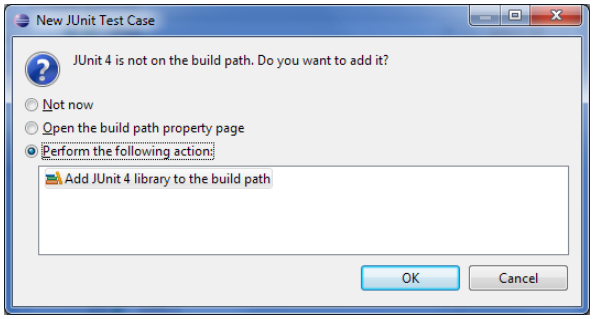

Abbildung 2.3: Testfall erstellen, Schritt 4

Die neue Klasse `ShopTest` enthält jetzt die zwei noch leeren Testmethoden `testSell` und `testIncreasePrice`, die mit den Annotationen `@Test` versehen sind. Annotationen werden im Laufe des Semesters in der Vorlesung genauer behandelt. Die Namen aller Testmethoden beginnen aus alter Tradition mit `test`, gefolgt vom Namen der zu testenden Methode, deren erster Buchstabe groß geschrieben wird. Dieses Namensschema müssen Sie mit JUnit 4 nicht mehr beibehalten. Es stammt noch von älteren Versionen. Wichtig ist lediglich die Annotation `@Test`. Sie müssen aber darauf achten, dass Testmethoden weder Übergabe- noch Rückgabeparameter besitzen und `public`

sind. Den Rumpf der Testmethode schreiben Sie natürlich selbst, weil nur Sie wissen, was Sie testen wollen. Er könnte so skizziert werden:

```
@Test
public void testSell() {
  Shop shop = new Shop();
  shop.add(new Medium("Delicatessen", Medium.Type.VIDEO_DVD,
                      Medium.Category.HUMOR,
                      2500, "Lager A2");
  shop.sell("Delicatessen");
  // Testcode fehlt hier noch

  shop.sell("Mars Attacks");
  // Testcode fehlt hier noch
}
```

Der Test des Ergebnisses erfolgt mit Methoden von JUnit:

- In der Testmethode wird `assertTrue(<Boole'scherAusdruck>)` aufgerufen.
- Ergibt der Ausdruck `false`, so bricht die Ausführung (später) hier ab. Der Testfall ist nicht erfüllt.
- Wenn der Ausdruck `true` wird, wird die Methode weiter ausgeführt, bis sie beendet ist. Der Testfall ist erfolgreich.

Neben `assertTrue` gibt es weitere Methoden, um Bedingungen zu prüfen. Sie finden Sie in der Dokumentation zu JUnit. So würde die Test-Methode aus dem Beispiel nach dem Einfügen der Prüfungen aussehen:

```
@Test
public void testSell() {
  Shop shop = new Shop();
  shop.add(new Medium("Delicatessen", Medium.Type.VIDEO_DVD,
                      Medium.Category.HUMOR,
                      2500, "Lager A2");
  shop.sell("Delicatessen");
  // DVD suchen
  Medium deli = shop.find("Delicatessen");
  // Test, ob Medium im Shop ist.
  assertTrue(deli != null);
  // Test, ob Anzahl um 1 reduziert wurde.
  assertTrue(deli.getCount() == 0);
}
```

Wenn Sie prüfen wollen, ob eine Methode immer eine bestimmte Ausnahme auslöst, dann können Sie die Annotation `@Test` in der folgenden Form verwenden. Dabei beinhaltet der Ausdruck hinter `expected =` die Klasse der erwarteten Ausnahme.

```
@Test(expected = ArithmeticException.class)
public void divisionWithException() {
  int i = 1/0;
}
```

Zum Ausführen der Tests wird die Testklasse im Package-Explorer durch Rechtsklick mit der Maus ausgewählt. Im erscheinenden Kontextmenü muss `Run` → `JUnit Test` angeklickt werden. Dadurch werden alle Testmethoden ausgeführt. Falls alle erfolgreich sind, wird im JUnit-Tab ein grüner Balken angezeigt. Erscheint hier ein roter Balken, so ist mindestens ein Test fehlgeschlagen. Dieser Test ist unter dem Balken rot markiert und kann direkt mit einem Doppelklick ausgewählt werden.

2.4.2 Automatische Initialisierung je Testfall

Prinzipiell lassen sich alle Tests wie oben beschrieben durchführen. Ein Problem wird aber schnell sichtbar: Viele Testmethoden benötigen am Anfang immer einen ähnlichen Code, in dem der Shop erstellt und mit Medien gefüllt wird. Als Lösung bietet sich an, die Initialisierung zu automatisieren. Damit wird von JUnit vor der Ausführung eines Testfalls immer eine neue Instanz von `Shop` erzeugt und gefüllt, dann die nächste Testmethode aufgerufen. Die Initialisierung wird von einer beliebigen Methode durchgeführt, die Sie mit der Annotation `@Before` versehen. Folgende Vorgehensweise empfiehlt sich für das Beispiel:

- Es wird ein privates Attribut `Shop shop` erstellt.
- Die hinzuzufügende Initialisierungsmethode erzeugt den Shop und trägt die Referenz in das private Attribut ein.

Dieses ist ein guter Stil und eine Arbeitserleichterung, da unnötiges, manuelles Kopieren von Code vermieden wird.
Sollten nach Ablauf der Testmethoden noch Aufräumarbeiten erforderlich sein, so können die in einer Methode erfolgen, die mit der Annotation `@After` versehen ist. Für die Shop-Aufgabe aber ist das nicht notwendig. Ergebnis (ohne den Testfall `increasePrice`):

```
public class ShopTest {

  private Shop shop;

  @Before
  public void setUp() throws Exception {
      shop = new Shop();
      shop.add(new Medium("Delicatessen", Medium.Type.VIDEO_DVD,
                          Medium.Category.HUMOR, 2500,
                          "Lager A2"));
    }

  /**
   * Test method for 'de.hska.iwii.i2.media.Shop.sell(String)'.
   */
  @Test
  public void testSell() {
    // Artikel verkaufen.
    shop.sell("Delicatessen");

    Medium deli = shop.find("Delicatessen");
    // Test, ob Medium im Shop ist.
    assertTrue(deli != null);
    // Test, ob Anzahl um 1 reduziert wurde.
    assertTrue(deli.getCount() == 0);
  }

  // ...
}
```

2.4.3 Automatische Initialisierung je Testklasse

Die Annotationen `@Before` und `@After` werden vor bzw. nach jedem Aufruf einer Testmethode (eines Testfalls) ausgeführt. In vielen Fällen genügt es aber, je Testklasse Me-

thoden vor dem Start und nach der Beendigung aller Testmethoden der Klasse aufzurufen. Hierzu dienen die Annotationen `@BeforeClass` und `@AfterClass`.

2.4.4 Weitere Hinweise zu JUnit

Mehr über JUnit ist unter `http://www.junit.org` zu finden:

- neueste Version
- Tutorial

Unter der URL `http://www.vogella.com/articles/JUnit/article.html` finden Sie eine kompakte Einführung in JUnit.
Ähnliche Frameworks für automatisierte Einzeltests existieren für C#, C++ und viele andere Programmiersprachen. JUnit ist „Standard“ in der Java-Softwareentwicklung und wird bei vielen Projekten eingesetzt. Unit-Tests erlauben eine komfortable Automatisierung der Tests. Dabei ist es für den Entwickler aber nicht erkennbar, welche Teile seines Codes wirklich in einem Testlauf überprüft werden. Dazu gibt es Tools, die die Testüberdeckung ermitteln. Das soll hier aber nicht näher betrachtet werden.

3

Pflichtaufgaben

3.1 Game of Life

Dieses Spiel wurde erstmalig Ende der 60er Jahre von J. H. Conway vorgeschlagen und detailliert untersucht. Eine genaue Beschreibung ist unter Anderem in Wikipedia zu finden: `http://de.wikipedia.org/wiki/Game_of_Life`. Das Spiel basiert auf einem zweidimensionales Feld aus n Zeilen und m Spalten. Jede Zelle des Feldes besitzt einen der zwei Zustände „lebend“ (im Bild unten blau dargestellt) oder „tot“. Die zu Beginn lebenden Zellen bilden die Anfangsgeneration.

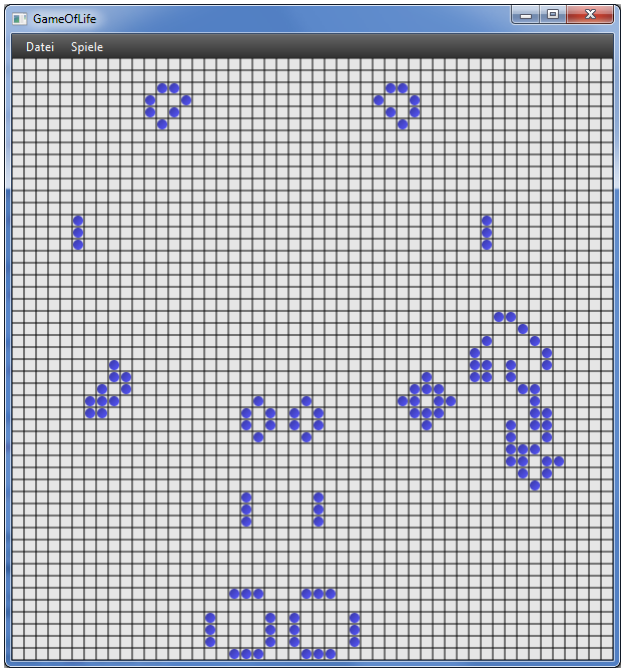

Abbildung 3.1: Szene aus dem GOL

In jedem Zeittakt wird eine neue Generation nach den folgenden Regeln ermittelt:

- Eine lebende Zelle lebt auch in der nächsten Generation, wenn zwei oder drei ihrer acht Nachbarn in der aktuellen Generation leben.

- Eine lebende Zelle stirbt an Überbevölkerung, wenn mehr als drei ihrer acht Nachbarn in der aktuellen Generation leben.
- Eine lebende Zelle stirbt an Vereinsamung, falls weniger als zwei ihrer acht Nachbarn in der aktuellen Generation leben.
- Eine tote Zelle wird in der nächsten Generation lebendig, wenn genau drei ihrer acht Nachbarn in der aktuellen Generation leben.

Das Feld ist nicht zyklisch aufgebaut. Das heißt, dass eine Zelle am Rand des Spielfeldes weniger Nachbarzellen als eine Zelle im Zentrum besitzt.

3.1.1 Initialisierung

Verwenden Sie die leere Klasse `GameOfLifeLogic` aus dem Grundgerüst als Basis für die Logik des Spiels. Die Visualisierung (Klassen `GameOfLifeApplication` und `GameOfLifeCanvas`) sowie einige vordefinierte Startsituationen (Klasse `Games`) sind bereits vorhanden. Das Vorgabeprojekt können Sie im Ilias auf den Übungsseiten herunterladen. Geben Sie Ihrer Logikklasse als Attribut ein dreidimensonales Array `population` mit Boole'schen Werten. Mit der dritten Dimension unterscheiden Sie zwischen neuer und aktueller Generation.

3.1.2 Betrieb

Ihre Klasse `GameOfLifeLogic` beinhaltet die Logik des Spiels. Damit Ihre Logik eingesetzt werden kann, müssen Sie die folgenden Methoden implementieren:

- `void setStartGeneration(boolean[][] generation)`: Durch den Aufruf dieser Methode durch die Visualisierung bekommen Sie die im Menü ausgewählte Startsituation als zweidimensionales Array übergeben.
- `void nextGeneration()`: Berechnet die Folgegeneration der aktuellen und trägt sie in das Array ein. Beachten Sie, dass während der Berechnung die neue Generation nicht die alte überschreiben darf. Das ist erst dann erlaubt, wenn die neue Generation vollständig berechnet wurde.
- `boolean isCellAlive(int x, int y)`: Testet, ob die Zelle an der Position (x, y) in der aktuellen Generation lebt. Dann ist der Rückgabewert `true`, ansonsten `false`.

3.2 Doppelt verkettete Liste

In dieser Aufgabe implementieren Sie eine doppelt verkettete Liste, wie sie bereits in der Vorlesung skizziert wurde.

3.2.1 Listenimplementierung

Implementieren Sie eine doppelt verkettete Liste für einen String als Datentyp. mit den folgenden öffentlichen Methoden und Konstruktoren. Einige davon könnten Ihnen aus der Vorlesung bekannt vorkommen.

- Der Standardkonstruktor erzeugt eine leere Liste.
- `void addFirst(String value)`: Trägt das Element vorne in die Liste ein.

- `void addLast(String value)`: Hängt das Element hinten an die Liste an.
- `void add(int index, String value)`: Fügt das Element am angegebenen Index in die Liste ein. Wenn der Index der Anzahl der Elemente in der Liste entspricht, wird der Wert also am Ende angehängt.
- `String get(int index)`: Liest den Wert am übergebenen Index aus.
- `String removeFirst()`: Löscht das erste Element.
- `String removeLast()`: Löscht das letzte Element.
- `String remove(int index)`: Löscht das Element am angegebenen Index und gibt den darin gespeicherten Wert zurück.
- `int getSize()`: Liest die Anzahl der Werte in der Liste aus.

Beachten Sie dabei, dass Sie keinen Code mit Copy und Paste duplizieren. Welche möglichen Fehler müssen Sie behandeln? Verwenden Sie dazu Zusicherungen („Assertions“). Die explizite Verwendung von Ausnahmen kennen Sie aus der Vorlesung ja noch nicht. Denken Sie daran, die Assertions auch einzuschalten. Dazu übergeben Sie beim Starten der virtuellen Maschine den Parameter `-ea`.

3.2.2 Test der Liste mit Unit-Tests

Erstellen Sie eine JUnit-Testklasse mit Testfällen, die alle Methoden Ihrer Liste überprüft. Untersuchen Sie auch alle möglichen Randfälle wie beispielsweise das Löschen aus einer leeren Liste, das Lesen am Anfang und Ende der Liste usw. Eine kurze Einführung in JUnit finden Sie in Abschnitt 2.4. Überprüfen Sie Ihre Lösung auch mit „Findbugs“.

3.2.3 Ringpuffer mit einer verketteten Liste

Schreiben Sie eine Ringpuffer-Klasse, die intern statt eines Arrays fester Größe Ihre Listenimplementierung aus Aufgabenteil 3.2.1 ohne Änderungen einsetzt. Damit wäre der Ringpuffer theoretische in der Lage, beliebig viele Strings aufzunehmen. Die Größenbeschränkung entfällt also. Überlegen Sie sich, wie Sie die Schreib- und Lesemarken des Ringpuffers auf die Liste übertragen können.

3.2.4 Generische Liste

Schreiben Sie die Listenklasse aus Aufgabenteil 3.2.1 so um, dass Sie beliebige Datentypen in der Liste speichern können. Sie haben dann also eine generische Klasse erstellt.

3.2.5 Generischer Ringpuffer

Schreiben Sie die Ringpufferklasse aus Aufgabenteil 3.2.4 so um, dass Sie beliebige Datentypen im Puffer speichern können. Sie haben dann also eine generische Klasse erstellt.

3.3 Zeichenprogramm (Basis)

In dieser Aufgabe soll ein kleines Zeichenprogramm erstellt werden. Dieses ist in der Lage, Ellipsen, Rechtecke und Linien als Vektorelemente zu erstellen. Damit ist gemeint, dass das Programm nicht pixelorientierte Zeichnungen erstellt, sondern Vektorgrafiken. Diese Aufgabe verwendet JavaFX zur Darstellung. Eine sehr gute Einführung inklusive Referenz bietet Oracle online unter `http://docs.oracle.com/javafx/index.html`. Diese Aufgabe zum Zeichenprogramm lässt Sie auf zwei prinzipiell unterschiedliche Arten, zwischen denen Sie selbst auswählen können, lösen. Wenn Sie sich für eine Variante entschieden haben, dann müssen Sie diese auch für die Aufgaben 3.4 und 3.5 beibehalten. Der folgende Screenshot zeigt eine Übersicht über die Musterlösung beider Varianten.

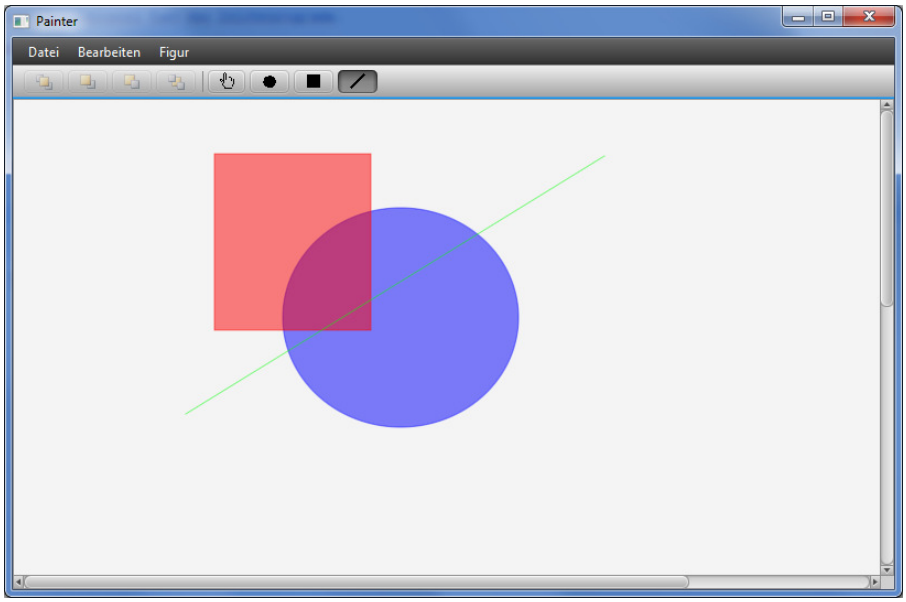

Abbildung 3.2: Screenshot des Malprogrammes

In den folgenden Abschnitten führen kleine Teilaufgaben zum Ergebnis. Dieses Vorgehen soll helfen, die Komplexität der Anwendung gering zu halten. Es ist empfehlenswert, sich an die Schritte zu halten, da sonst am Ende eventuell unnötig zusätzliche Arbeit entsteht. Das endgültige Programm muss die folgenden Funktionen unterstützen:

- Der Anwender kann verschiedene geometrische Figuren einfügen. Momentan sind nur Ellipsen, Rechtecke und Linien geplant. Das Programm ist aber so auszulegen, dass andere Figuren ohne großen Aufwand eingebaut werden können.
- Der Anwender kann existierende Figuren verschieben.
- Die Figuren liegen in der Zeichenfläche übereinander. Die Reihenfolge ihrer „Stapelung“ soll sich ändern lassen.
- Einzelne Figuren können zu Gruppen zusammengesetzt und wiederum wie eine einzelne Figur behandelt werden.
- Figuren lassen sich kopieren, einfügen und löschen.

Keine Panik, die folgenden Anleitungen sollen Ihnen helfen. Dort werden die Aufgaben auch genauer beschrieben. Sehen Sie sich auch die Dokumentation der JavaFX-Klassen

an, die Sie verwenden werden: http://docs.oracle.com/javafx/2/api/index.html. Dort finden Sie viele nützliche Methoden, die Sie nicht selbst schreiben müssen. Das Tutorial von Oracle unter http://docs.oracle.com/javafx/ zeigt Ihnen anhand von Beispielen den Einsatz vieler JavaFX-Klassen.

3.3.1 Variante 1: Zeichnen in einem Canvas-Objekt

Diese Variante dürfte für die meisten unter Ihnen leichter umzusetzen sein. Sie platzieren eine Zeichenfläche (Klasse `Canvas`) im Fenster und zeichnen die Figuren direkt darin. Der Vorteil dieses Verfahrens ist die leichtere Implementierbarkeit, wenn Sie im Umgang mit objekt-orientierter Programmierung noch etwas unsicher sind.

3.3.1.1 Klassendiagramm für die Figuren

Im ersten Schritt entsteht ein rudimentäres Klassendiagramm, das die Figuren verwaltet. Es müssen auf jeden Fall Ellipsen, Rechtecke und Linien auf eine einheitliche Art und Weise verwendet werden können. Darüber hinaus muss sich das Programm leicht um zusätzliche Figurtypen erweitern lassen. Um Fehler im Anfangsstadium zu vermeiden, sollte der Ansatz vor der Implementierung mit dem jeweiligen Betreuer besprochen werden.
Wichtige Fragestellungen sind:

- Welche Informationen muss eine Figur besitzen?
- Lassen sich schon jetzt notwendige Methoden erkennen?

In dieser Teilaufgabe sollten zunächst alle interaktiven Operationen des Anwenders ignoriert werden. Die Figuren werden erst einmal programmgesteuert erzeugt. Die anderen Punkte werden später berücksichtigt. Erstellen Sie das Klassendiagramm mit einem Modellierungswerkzeug Ihrer Wahl. Ihre Figuren erben nicht von einer der Klassen aus JavaFX.

3.3.1.2 Anwendungsstart

Erstellen Sie eine Klasse mit `main`-Methode. Die Methode erzeugt das vorgegebene Hauptfenster mit dem Aufruf: `Application.launch(MainWindow.class, args);` Die Klasse `Application` stammt von JavaFX, `MainWindow` ist die Fensterklasse des vorgegebenen Grundgerüstes. Sie sollten nach dem Start der Anwendung die Oberfläche sehen. Interaktionen darin sind noch nicht möglich. Wenn Sie lieber eine vertikale Toolbar-Leiste am linken Rand des Fensters haben möchten, dann können Sie vor dem `launch`-Aufruf die Methode `MainWindow.setHorizontalToolBar(false);` aufrufen.

3.3.1.3 Zeichenfläche

Auf die Zeichenfläche malen Sie später die einzelnen Figuren. Schreiben Sie eine eigene Klasse, die von `Canvas` aus JavaFX erbt. Geben Sie der Zeichenfläche eine feste Größe, damit der Layout-Manager der Oberfläche daraus die Fenstergröße ableiten kann. Abschließend erweitern Sie die Klasse aus Aufgabenteil 3.3.1.2 so, dass Sie vor dem `launch`-Aufruf dem Hauptfenster die Zeichenfläche übergeben:
`MainWindow.setCanvas(canvas);`
Fügen Sie jetzt testweise programmgesteuert einige JavaFX-Figuren in die Zeichenfläche ein und prüfen Sie, ob diese auch korrekt dargestellt werden.

3.3.1.4 Modell für alle Figuren

Um eine saubere Anwendungsstruktur zu schaffen, sollen Bildschirmausgabe und Datenmodell voneinander getrennt werden. Dazu ist in diesem Schritt ein sogenanntes Modell zu erstellen, das alle Figuren verwaltet. Da die Anzahl der Figuren unbekannt ist, scheidet ein Array natürlich aus. Die Klasse `Vector` darf nicht verwendet werden. Nehmen Sie statt dessen lieber eine `ArrayList` oder eine `LinkedList`. Um eine größtmögliche Flexibilität zu erhalten, wird das Zeichenprogramm nicht auf einer konkreten Implementierung des Modelles arbeiten, sondern stattdessen nur eine Schnittstelle kennen, die das Modell implementieren muss. Wichtige Anforderungen:

- Figuren lassen sich zum Modell hinzufügen.
- Alle Figuren des Modells können ausgelesen.
- Der Inhalt des Modells kann gelöscht werden.

3.3.1.5 Zeichnen des Modells

Damit sich jetzt auf dem Bildschirm „etwas ereignet“, soll in diesem Schritt die erste Fassung der Zeichenfläche mit einer Darstellung der Figuren entstehen. Vorgehensweise:

- Verwenden Sie Ihren Zeichenflächenklasse aus Aufgabenteil 3.3.1.3. Diese Klasse bekommt im Konstruktor die Größe der Fläche sowie eine Referenz auf das Modell mit den Figuren übergeben.
- Die Zeichenfläche durchläuft das Modell und zeichnet alle Figuren, wenn es notwendig ist. Das Zeichnen der Figuren erfolgt natürlich nicht direkt durch die Zeichenfläche. Stattdessen ruft diese die noch zu implementierende Zeichenmethode der einzelnen Figuren auf. Jede Figur zeichnet sich also selbst.

Wenn Sie verdeckte Objekte durch die darüberliegenden „durchschimmern“ lassen wollen, geben Sie als vierten Parameter der Füll-Farbe der Figuren einen Deckungsgrad von weniger als 100%. Beispiel für eine rote Farbe mit einem Deckungsgrad von 50%:

```
Color ellipseColor = new Color(1.0, 0.0, 0.0, 0.5);
```

3.3.1.6 Ereignisbehandlung

Jetzt soll die Anwendung interaktiv verwendet werden können. Dazu muss die sogenannte Ereignisbehandlung implementiert werden. Um die Lösung etwas zu vereinfachen, ist ein Großsteil der Ereignisbehandlung bereits in der Klasse `MainWindow` implementiert. Damit können Sie alle Ereignisse auf eine einheitliche Art und Weise behandeln, ohne die Details der JavaFX-Ereignisse kennen zu müssen. Der Ablauf der Benachrichtigung sieht immer so aus:

1. Ein JavaFX-Widget (hier die Zeichenfläche) stellt ein Ereignis fest.
2. Es benachrichtigt je nach Ereignistyp unterschiedliche Methoden der bereits vorgegebenen Klasse `MainWindow`.
3. Die Methoden in `MainWindow` analysieren das Ereignis und rufen in der zu erstellenden Ereignisbehandlung unterschiedliche Methoden auf. In diesen Methoden werden Sie später verschiedene Funktionen wie das Erstellen der Figuren umsetzen. Die Funktionen werden in weiteren Teilaufgaben schrittweise vorgestellt.

Damit die Benachrichtigung durch Ereignisse funktioniert, müssen einige Punkte beachtet werden:

- `MainWindow` erfordert, dass Sie die vorgegebene Schnittstelle `DrawingListener` implementieren. Dort sind die Signaturen aller Methoden vorgegeben, die im Falle von Ereignissen aufgerufen werden. Die Bedeutungen der einzelnen Methoden werden in späteren Aufgabenteilen vorgestellt.
- Es muss also eine Klasse geschrieben werden, die diese Schnittstelle implementiert. Im ersten Schritt sollten die Methodenrümpfe zunächst so implementiert werden, dass sie nur eine Meldung über die Art des Ereignisses ausgeben.
- Außerdem muss `MainWindow` natürlich noch Ihr Objekt, das die Ereignisbehandlung durchführt, kennen. Dazu wird ein Objekt der Beobachterklasse erzeugt und mit `MainWindow.setDrawingListener(listener);` am `MainWindow` registriert.
- Jetzt lässt sich testen, ob die Meldungen bei verschiedenen Ereignissen ausgegeben werden.

3.3.1.7 Erzeugen von Figuren

In diesem Abschnitt soll das interaktive Erzeugen einer Figur in der Ereignisklasse umgesetzt werden. Dazu ruft `MainWindow` drei öffentliche Methoden in der folgenden Reihenfolge auf.

1. `void startCreateFigure(String type, double x, double y)`:
   Betätigt der Anwender die Maustaste, wird die Methode aufgerufen. Der Mauszeiger befindet sich dabei in der Zeichenfläche an der Position (x, y). Jetzt wird an dieser Stelle eine Figur erzeugt. Die Größe der Figur wird in den folgenden Schritten festgelegt. Als Startwert sollte zunächst von einer Breite und Höhe von jeweils einem Pixel ausgegangen werden. Die Zeichenkette `type` beinhaltet den Namen der zu erstellenden Figur. Momentan sind implementiert:
   - `"ellipse"` für eine Ellipse
   - `"rectangle"` für ein Rechteck
   - `"line"` für eine Linie

   Erzeugen Sie die entsprechende Figur und fügen Sie sie in die Zeichenfläche ein.
2. `void workCreateFigure(double x, double y)`:
   Jetzt wird die Größe der Figur festgelegt. (x, y) beinhalten die aktuelle Position des Mauszeigers, während der Anwender die Maus mit gedrückter Maustaste bewegt. Die Größe der in Schritt 1 erzeugten Figur wird permanent angepasst. Diese Methode wird mehrfach aufgerufen, bis der Anwender die Maustaste losgelassen hat.
3. `void endCreateFigure(double x, double y)`:
   Die Maustaste wurde losgelassen. Die Erstellung der Figur ist somit beendet.

Bei jeder Änderung an Ihrem Modell, also wenn Sie z.B. Figuren hinzugefügt haben, müssen Sie die Zeichenfläche auffordern, sich selbst und alle darin enthaltenen Figuren neu zu zeichnen. Dieser Ansatz ist zunächst einmal sehr ineffizient. Als kleine Verbesserung kann das sogenannte „Clipping“ eingesetzt werden, mit dessen Hilfe nur noch der Teil des Bildschirmausschnitts neu gezeichnet wird, der sich auch verändert hat. Sie können diesen Punkt aber zunächst einmal ignorieren, er wird in Abschnitt 3.5.1.2 betrachtet.

3.3.1.8 Verschieben von Figuren

Erfolgt ein „Dragging“ auf einer bereits existierenden Figur, so wird diese verschoben. Der Ablauf sieht wie folgt aus:

1. `void startMoveFigure(double x, double y)`:
   Wird aufgerufen, wenn der Anwender mit der Maustaste an die Position (x, y) klickt und die Maustaste gedrückt hält. Damit wird die Figur an der Position (x, y) gewählt. Diese Figur müssen Sie selbst aus Ihrem Modell heraussuchen.
2. `void workMoveFigure(double x, double y)`:
   Solange der Anwender die Maus bei weiterhin gedrückter Taste bewegt, wird die Methode immer wieder aufgerufen.
3. `void endMoveFigure(double x, double y)`:
   Diese Methode wird nach dem Loslassen der Maustaste aufgerufen. Damit ist das Erzeugen beendet.

3.3.2 Variante 2: Knoten in einem Szene-Graphen

Hier verwenden Sie einen Knoten des Szenegraphen (Klasse `Pane`), in den Sie die Figuren als Knoten einfügen. Diese Variante hat den Vorteil, dass JavaFX Ihnen das komplette Zeichnen der Figuren abnimmt. Sie müssen sich etwas stärker in die API von JavaFX einarbeiten, kommen so aber zu einer eleganteren Lösung mit einem geringeren Programmieraufwand.

3.3.2.1 Klassendiagramm für die Figuren

Im ersten Schritt entsteht ein rudimentäres Klassendiagramm, das die Figuren verwaltet. Es müssen auf jeden Fall Ellipsen, Rechtecke und Linien auf eine einheitliche Art und Weise verwendet werden können. Darüber hinaus muss sich das Programm leicht um zusätzliche Figurtypen erweitern lassen. Um Fehler im Anfangsstadium zu vermeiden, sollte der Ansatz vor der Implementierung mit dem jeweiligen Betreuer besprochen werden.
Wichtige Fragestellungen sind:

- Welche Informationen muss eine Figur besitzen?
- Lassen sich schon jetzt notwendige Methoden erkennen?

In dieser Teilaufgabe sollten zunächst alle interaktiven Operationen des Anwenders ignoriert werden. Die Figuren werden erst einmal programmgesteuert erzeugt. Die anderen Punkte werden später berücksichtigt. Erstellen Sie das Klassendiagramm mit einem Modellierungswerkzeug Ihrer Wahl. Berücksichten Sie dabei auch die JavaFX-Klassen, die Sie zur Darstellung verwenden. Vorgabe ist, dass Sie die Klasse aus dem Paket `javafx.scene.shape` verwenden und nicht direkt selbst auf einer Zeichenfläche malen. Jede Figur, die Sie erzeugen, erbt also von `Shape` und `Node`.

3.3.2.2 Anwendungsstart

Erstellen Sie eine Klasse mit `main`-Methode. Die Methode erzeugt das vorgegebene Hauptfenster mit dem Aufruf: `Application.launch(MainWindow.class, args);` Die Klasse `Application` stammt von JavaFX, `MainWindow` ist die Fensterklasse des vorgegebenen Grundgerüstes. Sie sollten nach dem Start der Anwendung die Oberfläche sehen. Interaktionen darin sind noch nicht möglich. Wenn Sie lieber eine vertikale Toolbar-Leiste am linken Rand des Fensters haben möchten, dann können Sie vor dem `launch`-Aufruf die Methode `MainWindow.setHorizontalToolBar(false);` aufrufen.

3.3.2.3 Zeichenfläche

In die Zeichenfläche platzieren Sie später die einzelnen Figuren. Schreiben Sie eine eigene Klasse, die von `Pane` aus JavaFX erbt. Geben Sie der Zeichenfläche eine bevorzugte Größe, damit der Layout-Manager der Oberfläche daraus die Fenstergröße ableiten kann. Abschließend erweitern Sie die Klasse aus Aufgabenteil 3.3.2.2 so, dass Sie vor dem `launch`-Aufruf dem Hauptfenster die Zeichenfläche übergeben:
`MainWindow.setMainPanel(panel);`
Fügen Sie jetzt testweise programmgesteuert einige JavaFX-Figuren in die Zeichenfläche ein und prüfen Sie, ob diese auch korrekt dargestellt werden.

3.3.2.4 Ereignisbehandlung

Jetzt soll die Anwendung interaktiv verwendet werden können. Dazu muss die sogenannte Ereignisbehandlung implementiert werden. Um die Lösung etwas zu vereinfachen, ist ein Großsteil der Ereignisbehandlung bereits in der Klasse `MainWindow` implementiert. Damit können Sie alle Ereignisse auf eine einheitliche Art und Weise behandeln, ohne die Details der JavaFX-Ereignisse kennen zu müssen. Der Ablauf der Benachrichtigung sieht immer so aus:

1. Ein JavaFX-Widget stellt ein Ereignis fest.
2. Es benachrichtigt je nach Ereignistyp unterschiedliche Methoden der bereits vorgegebenen Klasse `MainWindow`.
3. Die Methoden in `MainWindow` analysieren das Ereignis und rufen in der zu erstellenden Ereignisbehandlung unterschiedliche Methoden auf. In diesen Methoden werden Sie später verschiedene Funktionen wie das Erstellen der Figuren umsetzen. Die Funktionen werden in weiteren Teilaufgaben schrittweise vorgestellt.

Damit die Benachrichtigung durch Ereignisse funktioniert, müssen einige Punkte beachtet werden:

- `MainWindow` erfordert, dass Sie die vorgegebene Schnittstelle `DrawingListener` implementieren. Dort sind die Signaturen aller Methoden vorgegeben, die im Falle von Ereignissen aufgerufen werden. Die Bedeutungen der einzelnen Methoden werden in späteren Aufgabenteilen vorgestellt.
- Es muss also eine Klasse geschrieben werden, die diese Schnittstelle implementiert. Im ersten Schritt sollten die Methodenrümpfe zunächst so implementiert werden, dass sie nur eine Meldung über die Art des Ereignisses ausgeben.

- Außerdem muss `MainWindow` natürlich noch Ihr Objekt, das die Ereignisbehandlung durchführt, kennen. Dazu wird ein Objekt der Beobachterklasse erzeugt und mit `MainWindow.setDrawingListener(listener);` am `MainWindow` registriert.
- Jetzt lässt sich testen, ob die Meldungen bei verschiedenen Ereignissen ausgegeben werden.

Wenn Sie Lust haben, dann können Sie in der optionalen Aufgabe 4.12 neben den hier verwendeten Mausereignissen auch mit Gesten arbeiten, sofern Sie ein Gerät besitzen, das Gestenerkennung unterstützt. Ansonsten sehen Sie für die drei Methoden im Beobachter einfach leere Implementierungen vor.

3.3.2.5 Erzeugen von Figuren

In diesem Abschnitt soll das interaktive Erzeugen einer Figur in der Ereignisklasse umgesetzt werden. Dazu ruft `MainWindow` drei öffentliche Methoden in der folgenden Reihenfolge auf.

1. `void startCreateFigure(String type, double x, double y)`:
   Betätigt der Anwender die Maustaste, wird die Methode aufgerufen. Der Mauszeiger befindet sich dabei in der Zeichenfläche an der Position (x, y). Jetzt wird an dieser Stelle eine Figur erzeugt. Die Größe der Figur wird in den folgenden Schritten festgelegt. Als Startwert sollte zunächst von einer Breite und Höhe von jeweils einem Pixel ausgegangen werden. Die Zeichenkette `type` beinhaltet den Namen der zu erstellenden Figur. Momentan sind implementiert:
   - `"ellipse"` für eine Ellipse
   - `"rectangle"` für ein Rechteck
   - `"line"` für eine Linie

   Erzeugen Sie die entsprechende Figur und fügen Sie sie in die Zeichenfläche ein.
2. `void workCreateFigure(double x, double y)`:
   Jetzt wird die Größe der Figur festgelegt. (x, y) beinhalten die aktuelle Position des Mauszeigers, während der Anwender die Maus mit gedrückter Maustaste bewegt. Die Größe der in Schritt 1 erzeugten Figur wird permanent angepasst. Diese Methode wird mehrfach aufgerufen, bis der Anwender die Maustaste losgelassen hat.
3. `void endCreateFigure(double x, double y)`:
   Die Maustaste wurde losgelassen. Die Erstellung der Figur ist somit beendet.

3.3.2.6 Verschieben von Figuren

Erfolgt ein „Dragging“ auf einer bereits existierenden Figur, so wird diese verschoben. Der Ablauf sieht wie folgt aus:

1. `void startMoveFigure(Node node, double x, double y)`:
   Wird aufgerufen, wenn der Anwender mit der Maustaste auf die Figur `node` klickt und die Maustaste gedrückt hält. Damit wird die Figur an der Position (x, y) gewählt.
2. `void workMoveFigure(Node node, double x, double y)`:
   Solange der Anwender die Maus bei weiterhin gedrückter Taste bewegt, wird die Methode immer wieder aufgerufen.

3. `void endMoveFigure(Node node, double x, double y)`:
   Diese Methode wird nach dem Loslassen der Maustaste aufgerufen. Damit ist das Erzeugen beendet.

Die Klasse `Node` ist die Basisklasse der Klassen, die im Szene-Graphen verwendet werden können. Alle `Shape`-Objekte, also auch `Ecllipse`, `Rectangle` und `Line` erben indirekt von dieser Klasse.

3.4 Zeichenprogramm (Erweiterung)

Nachdem das Grundgerüst des Zeichenprogramms lauffähig ist, folgen die interessanten Erweiterungen wie Selektion, Gruppierung sowie Copy und Paste.

3.4.1 Variante 1: Zeichnen in einem Canvas-Objekt

Denken Sie bei allen Operationen, die Figuren verändern oder neue Figuren einfügen, daran, dass die Zeichenfläche anschließend neu gezeichnet werden muss.

3.4.1.1 Selektion von Figuren

Figuren lassen sich vom Anwender selektieren. Dazu wird im Beobachter die Methode
`void selectFigure(double x, double y, boolean shift)`
aufgerufen. (x, y) enthält die X- bzw. Y-Position des Mauszeigers während der Selektion und `shift` gibt an, ob während der Selektion durch die Maus auch die Shift-Taste an der Tastatur gedrückt war. Ist das der Fall, dann wird die neue Figur zusätzlich zu den bereits selektierten Figuren ausgewählt. Ist `shift` dagegen `false`, dann werden alle vorher selektierten Figuren wieder deselektiert. Zur grafischen Darstellung der Selektion bietet es sich an, der jeweiligen Figur eine andere Rahmenfarbe („Stroke“) zu geben oder z.B. die Füllfarbe dunkler werden zu lassen.

3.4.1.2 Gruppierung von Figuren

Jetzt erweitern Sie die Anwendung so, dass mehrere Figuren zu einer Gruppe zusammengefasst werden können. Eine Gruppe mit allen Figuren wird als eine Einheit betrachtet. Damit ist gemeint, dass die Selektion eines Gruppenelementes automatisch auch alle anderen Elemente der Gruppe auswählt. Die Gruppe wird also wie eine komplexe Figur behandelt, die selbst keine Darstellung besitzt. Eine Gruppe ist hier ein sogenanntes **Kompositum**. Dieses setzt sich aus mehreren Elementen zusammen und agiert wiederum wie ein einzelnes Element. Erweitern Sie Ihr Klassendiagramm aus Teilaufgabe 3.3.1.1 um die Gruppierungseigenschaft. Einige Hinweise zur Vorgehensweise:

- Erstellen Sie eine Klasse für Ihre Gruppe und lassen Sie diese von derselben Basisklasse erben wie die anderen Figuren.
- Es werden Methoden benötigt, um andere Figuren zur Gruppe hinzuzufügen oder aus der Gruppe zu entfernen.
- Wichtig ist auch, die Selektion einer Gruppe zu verwalten: Da die Gruppe keine eigene Darstellung besitzt, können Sie einfach alle in ihr vorhandenen Figuren selektiert darstellen.
- Wie kann sich eine Gruppe selbst zeichnen?

3.4.1.3 Erstellen einer Gruppe

Der Interaktionsablauf sieht wie folgt aus:

1. Der Anwender selektiert eine Menge von Figuren.
2. Er klickt auf die Taste zur Gruppierung. Dadurch wird in der Ereignisbehandlung die Methode `void groupFigures()` aufgerufen. Dort wird eine Gruppe erzeugt, die alle selektierten Figuren als Inhalt übergeben bekommt. Weiterhin muss die Gruppe natürlich auch zur Zeichenfläche hinzugefügt werden.
3. Außerdem müssen alle Figuren der Gruppe aus der Zeichenfläche entfernt werden, weil Sie jetzt in der Gruppe verwaltet werden. Ansonsten wären die Elemente in der Zeichenfläche doppelt vorhanden.

3.4.1.4 Auflösen einer Gruppe

Mit dieser Aktion soll eine Gruppe wieder in ihre Bestandteile zerfallen. Der Interaktionsablauf sieht wie folgt aus:

1. Der Anwender selektiert eine Gruppe.
2. Er klickt auf die Taste zur Degruppierung. Dudurch wird in der Ereignisbehandlung die Methode `void ungroupFigures()` aufgerufen. Sie liest alle Mitglieder der Gruppe aus und fügen sie wieder einzeln in die Zeichenfläche ein.
3. Danach wird die Gruppe aus der Zeichenfläche gelöscht.

Implementieren Sie in Ihrem Beobachter die Methode `boolean isGroupSelected()`. Die Methode liefert nur dann `true` zurück, wenn mindestens eine Gruppe ausgewählt wurde. Nur in diesem Fall werden die Menü- und Toolbar-Einträge zur Auflösung einer Gruppe aktiviert.

3.4.1.5 Eine Gruppe als Gruppenmitglied

Das ganze Spiel lässt sich jetzt natürlich weiter treiben: Was passiert, wenn eine Gruppe einer anderen Gruppe hinzugefügt wird? Die Antwort ist einfach: Da die Gruppe ihrerseits eine Figur ist, müssen keine zusätzlichen Fälle betrachtet werden. So schön verhält sich ein Kompositum.

3.4.1.6 Kopieren, Einfügen und Löschen von Figuren

Um die Arbeit mit dem Programm etwas komfortabler zu gestalten, soll jetzt das Kopieren, Einfügen und Löschen von Figuren unterstützt werden. Dazu werden in der Ereignisklasse verschiedene Methoden aufgerufen:

- `void copyFigures()`: Die Methode kopiert die momentan selektierten Figuren in einen internen Zwischenpuffer, das Clipboard. Zum Erstellen der Kopie kann der dafür in Java vorhandenen Standardmechanismus mittels der `clone`-Methode eingesetzt werden: Die oberste Basisklasse `Object` besitzt eine Methode `protected Object clone()`. Durch ein öffentliches Überschreiben dieser Methode in den Figur-Klassen implementiert jede Figur selbst, wie sie eine Kopie von sich anfertigen möchte. Als Ergebnis liefert die Methode eine Referenz auf die erzeugte Kopie zurück. Damit die Methode aufgerufen werden kann, muss die Figur-Klasse noch die Schnittstelle `Cloneable` implementieren. Während der Ereignisbehandlung werden dann nur noch die `clone`-Methoden der selektierten Objekte aufgerufen. Lesen Sie dazu unbedingt die genaue Diskussion zum Klonen von Objekten im Skript.

- `void pasteFigures()`: Fügt die kopierten Figuren in die Zeichenfläche ein. Damit diese nicht exakt auf den Originalen liegen, ist es empfehlenswert, die Kopien einfach um einige Pixel verschoben einzufügen oder aber die Position des Mauszeigers als neue Bildschirmkoordinate zu übernehmen.
- `void deleteFigures()`: Löscht alle selektierten Figuren.

Sie müssen die Methode `int getSelectedFiguresCount()` implementieren, damit die Menü- und Toolbareinträge zum Kopieren und Löschen nur dann aktiv sind, wenn diese Aktionen auch durchführbar sind. Die Methode gibt die Anzahl selektierter Figuren zurück. Weiterhin müssen Sie die Methode `int getFiguresInClipboardCount()` implementieren, die die Anzahl Figuren in Ihrem Clipboard zurückgibt. Damit werden die Menü- und Toolbareinträge für die Einfüge-Operation freigegeben oder gesperrt.

3.4.1.7 Reihenfolge der Modellelemente

Beim Arbeiten mit der Anwendung fällt schnell auf, dass sich einige Figuren nicht auswählen lassen, da andere darüber liegen. Sind die Figuren ausgefüllt dargestellt, dann spielte die Reihenfolge in der Zeichenfläche auch für die Sichtbarkeit eine große Rolle. In dieser Teilaufgabe soll daher die Reihenfolge der Figuren in der Zeichenfläche geändert werden können. Analog zu existierenden Zeichenprogrammen sind vier Methoden im Beobachter vorgesehen:

1. `moveSelectedFiguresUp()`: Alle ausgewählten Figuren wandern eine Ebene nach oben (in der Zeichenfläche eine Position nach vorne).
2. `moveSelectedFiguresDown()`: Alle ausgewählten Figuren wandern eine Ebene nach unten (in der Zeichenfläche eine Position nach hinten).
3. `moveSelectedFiguresToTop()`: Alle ausgewählten Figuren wandern ganz nach oben (in der Zeichenfläche ganz nach vorne).
4. `moveSelectedFiguresToBottom()`: Alle ausgewählten Figuren wandern ganz nach unten (in der Zeichenfläche ganz nach hinten).

3.4.2 Variante 2: Knoten in einem Szene-Graphen

Hier arbeiten Sie wieder direkt auf dem Szenegraphen und müssen sich nicht selbst um das Neuzeichnen der Figuren nach Änderungen kümmern.

3.4.2.1 Selektion von Figuren

Figuren lassen sich vom Anwender selektieren. Dazu wird im Beobachter die Methode `void selectFigure(Node node, double x, double y, boolean shift)` aufgerufen. `node` ist die zu selektierende Figur, (x, y) sind die X- bzw. Y-Position des Mauszeigers während der Selektion und `shift` gibt an, ob während der Selektion durch die Maus auch die Shift-Taste an der Tastatur gedrückt war. Ist das der Fall, dann wird die Figur `node` zusätzlich zu den bereits selektierten Figuren ausgewählt. Ist `shift` dagegen `false`, dann werden alle vorher selektierten Figuren wieder deselektiert. Zur grafischen Darstellung der Selektion bietet es sich an, der jeweiligen Figur eine andere Rahmenfarbe („Stroke“) zu geben oder z.B. die Füllfarbe dunkler werden zu lassen.

3.4.2.2 Gruppierung von Figuren

Jetzt erweitern Sie die Anwendung so, dass mehrere Figuren zu einer Gruppe zusammengefasst werden können. Eine Gruppe mit allen Figuren wird als eine Einheit betrachtet. Damit ist gemeint, dass die Selektion eines Gruppenelementes automatisch auch alle anderen Elemente der Gruppe auswählt. Die Gruppe wird also wie eine komplexe Figur behandelt, die selbst keine Darstellung besitzt. Eine Gruppe ist hier ein sogenanntes **Kompositum**. Dieses setzt sich aus mehreren Elementen zusammen und agiert wiederum wie ein einzelnes Element. Erweitern Sie Ihr Klassendiagramm aus Teilaufgabe 3.3.2.1 um die Gruppierungseigenschaft. Verwenden Sie die JavaFX-Klasse `Group`, die ihrerseits wie die anderen von Ihnen verwendeten Figuren auch von `Node` erbt. Einige Hinweise zur Vorgehensweise:

- Erstellen Sie eine Klasse für Ihre Gruppe und verwenden Sie dabei die `Group`-Klasse aus JavaFX.
- Es werden Methoden benötigt, um andere Figuren zur Gruppe hinzuzufügen oder aus der Gruppe zu entfernen. Oder bietet `Group` das vielleicht schon?
- Wichtig ist auch, die Selektion einer Gruppe zu verwalten: Da die Gruppe keine eigene Darstellung besitzt, können Sie einfach alle in ihr vorhandenen Figuren selektiert darstellen.

3.4.2.3 Erstellen einer Gruppe

Der Interaktionsablauf sieht wie folgt aus:

1. Der Anwender selektiert eine Menge von Figuren.
2. Er klickt auf die Taste zur Gruppierung. Dadurch wird in der Ereignisbehandlung die Methode `void groupFigures()` aufgerufen. Dort wird eine Gruppe erzeugt, die alle selektierten Figuren als Inhalt übergeben bekommt. Weiterhin muss die Gruppe natürlich auch zur Zeichenfläche hinzugefügt werden.
3. Außerdem müssen alle Figuren der Gruppe aus der Zeichenfläche entfernt werden, weil Sie jetzt in der Gruppe verwaltet werden. Ansonsten wären die Elemente in der Zeichenfläche doppelt vorhanden.

3.4.2.4 Auflösen einer Gruppe

Mit dieser Aktion soll eine Gruppe wieder in ihre Bestandteile zerfallen. Der Interaktionsablauf sieht wie folgt aus:

1. Der Anwender selektiert eine Gruppe.
2. Er klickt auf die Taste zur Degruppierung. Dudurch wird in der Ereignisbehandlung die Methode `void ungroupFigures()` aufgerufen. Sie liest alle Mitglieder der Gruppe aus und fügen sie wieder einzeln in die Zeichenfläche ein.
3. Danach wird die Gruppe aus der Zeichenfläche gelöscht.

Implementieren Sie in Ihrem Beobachter die Methode `boolean isGroupSelected()`. Die Methode liefert nur dann `true` zurück, wenn mindestens eine Gruppe ausgewählt wurde. Nur in diesem Fall werden die Menü- und Toolbar-Einträge zur Auflösung einer Gruppe aktiviert.

3.4.2.5 Eine Gruppe als Gruppenmitglied

Das ganze Spiel lässt sich jetzt natürlich weiter treiben: Was passiert, wenn eine Gruppe einer anderen Gruppe hinzugefügt wird? Die Antwort ist einfach: Da die Gruppe ihrerseits eine Figur ist, müssen keine zusätzlichen Fälle betrachtet werden. So schön verhält sich ein Kompositum.

3.4.2.6 Kopieren, Einfügen und Löschen von Figuren

Um die Arbeit mit dem Programm etwas komfortabler zu gestalten, soll jetzt das Kopieren, Einfügen und Löschen von Figuren unterstützt werden. Dazu werden in der Ereignisklasse verschiedene Methoden aufgerufen:

- `void copyFigures()`: Die Methode kopiert die momentan selektierten Figuren in einen internen Zwischenpuffer, das Clipboard. Zum Erstellen der Kopie kann leider der dafür in Java vorhandenen Standardmechanismus mittels der `clone`-Methode nicht eingesetzt werden, weil JavaFX-Klassen nicht die Schnittstelle `Cloneable` implementieren. Sehen Sie also eine eigene Methode vor, mit der jede Ihrer Figuren eine Kopie von sich selbst erzeugen kann. Verwenden Sie bitte aus Gründen der Einfachheit nicht die `Clipboard`-Klasse aus JavaFX. Eine normale `ArrayList` reicht als Clipboard völlig aus.
- `void pasteFigures()`: Fügt die kopierten Figuren in die Zeichenfläche ein. Damit diese nicht exakt auf den Originalen liegen, ist es empfehlenswert, die Kopien einfach um einige Pixel verschoben einzufügen oder aber die Position des Mauszeigers als neue Bildschirmkoordinate zu übernehmen.
- `void deleteFigures()`: Löscht alle selektierten Figuren.

Sie müssen die Methode `int getSelectedFiguresCount()` implementieren, damit die Menü- und Toolbareinträge zum Kopieren und Löschen nur dann aktiv sind, wenn diese Aktionen auch durchführbar sind. Die Methode gibt die Anzahl selektierter Figuren zurück. Weiterhin müssen Sie die Methode `int getFiguresInClipboardCount()` implementieren, die die Anzahl Figuren in Ihrem Clipboard zurückgibt. Damit werden die Menü- und Toolbareinträge für die Einfüge-Operation freigegeben oder gesperrt.

3.4.2.7 Reihenfolge der Modellelemente

Beim Arbeiten mit der Anwendung fällt schnell auf, dass sich einige Figuren nicht auswählen lassen, da andere darüber liegen. Sind die Figuren ausgefüllt dargestellt, dann spielte die Reihenfolge in der Zeichenfläche auch für die Sichtbarkeit eine große Rolle. In dieser Teilaufgabe soll daher die Reihenfolge der Figuren in der Zeichenfläche geändert werden können. Analog zu existierenden Zeichenprogrammen sind vier Methoden im Beobachter vorgesehen:

1. `moveSelectedFiguresUp()`: Alle ausgewählten Figuren wandern eine Ebene nach oben (in der Zeichenfläche eine Position nach vorne).
2. `moveSelectedFiguresDown()`: Alle ausgewählten Figuren wandern eine Ebene nach unten (in der Zeichenfläche eine Position nach hinten).
3. `moveSelectedFiguresToTop()`: Alle ausgewählten Figuren wandern ganz nach oben (in der Zeichenfläche ganz nach vorne).
4. `moveSelectedFiguresToBottom()`: Alle ausgewählten Figuren wandern ganz nach unten (in der Zeichenfläche ganz nach hinten).

3.5 Zeichenprogramm (Animationen und kleine Verbesserung)

Das Zeichenprogramm soll jetzt noch einige mehr oder wenige sinnvolle Animationen erhalten.

3.5.1 Variante 1: Zeichnen in einem Canvas-Objekt

In dieser Aufgabe nauen Sie Animationen ein und können auf freiwilliger Basis das Zeichnen optimieren.

3.5.1.1 Animationen

- Nach dem Einfügen eines neuen oder Verschieben eines existierenden Knotens soll die komplette Zeichenfläche kurz transparent und dann wieder sichtbar werden. Hierzu bietet sich eine `FadeTransition` an.
- Werden eine oder mehrere Figuren in den Ebenen verschoben, dann drehen Sie die Zeichenfläche einmal um 360 Grad.

3.5.1.2 Clipping (freiwillig Abgabe)

Das bisher erfolgte komplette Neuzeichnen der Zeichenfläche ist ineffizient. Ändern Sie Ihr Programm jetzt so ab, dass nur noch die geänderten Teile der Zeichenfläche neu gezeichnet werden. Werfen Sie dazu einen Blick auf die Methode `setClip` der Klasse `Canvas` in der API-Dokumentation von JavaFX.

3.5.2 Variante 2: Knoten in einem Szene-Graphen

- Nach dem Einfügen eines neuen oder Verschieben eines existierenden Knotens soll dieser kurz transparent und dann wieder sichtbar werden. Hierzu bietet sich eine `FadeTransition` an.
- Werden eine oder mehrere Figuren in den Ebenen verschoben, dann drehen Sie sie einmal um 360 Grad.

3.6 Game of Life mit Schnittstellen

Die Aufgabe zum „Game of Life“ hat einen ziemlichen Design-Fehler: Die Klasse des Grundgerüstes gibt die Klasse, in der Ihre Logik implementiert wurde, vor. Schöner wäre es, wenn das Grundgerüst lediglich eine Schnittstelle vorgibt, die die Logik implementieren muss. Leider waren Ihnen zum damaligen Zeitpunkt Schnittstellen zumindest aus der Vorlesung noch nicht bekannt. Erzeugen Sie eine Kopie Ihrer Lösung und bauen Sie diese so um, dass die Klasse `GameOfLifeApplication` eine Schnittstelle vorgibt, die die Logik implementiert. Übergeben Sie `GameOfLifeApplication` dann eine Referenz auf das Logik-Objekt. Leider geht das in der JavaFX-Lösung nur durch eine statische Methode in `GameOfLifeApplication`.

3.7 Verwendung der Collections-Klassen

Hier vergleichen Sie das Laufzeitverhalten einiger Datenstrukturen aus der Collections-API miteinander. So bekommen Sie ein Gefühl für die unterschiedlichen Zeitaufwände beim Lesen, Schreiben und Suchen. Verwenden Sie dazu immer Datenstrukturen mit `int`-Werten und nutzen Sie in Ihrem Code aus, dass diese Datenstrukturen gemeinsame Schnittstellen implementieren. Mit der Arbeit auf Schnittstellen sparen Sie sehr viel Code ein. Führen Sie die in der Tabelle 3.1 genannten Messungen durch.

Tabelle 3.1: Zeitmessungen mit Datenstrukturen

Operation	Datenstrukturen
Hängen Sie an die leere Datenstruktur nacheinander die `int`-Zahlen von `0` bis `99.999` an. Prüfen Sie, ob bei `Vector` und `ArrayList` Geschwindigkeitssteigerungen feststellbar sind, wenn Sie beiden beim Erzeugen im Konstruktor eine Größe vorgeben.	`Vector`, `ArrayList`, `LinkedList`, `HashSet`, `TreeSet`
Fügen Sie in die leere Datenstruktur nacheinander die `int`-Zahlen von `0` bis `99.999` immer am Anfang ein. Prüfen Sie, ob bei `Vector` und `ArrayList` Geschwindigkeitssteigerungen feststellbar sind, wenn Sie beiden beim Erzeugen im Konstruktor eine Größe vorgeben.	`Vector`, `ArrayList`, `LinkedList`
Nehmen Sie die oben gefüllten Datenstrukturen und suchen Sie mit Hilfe von Iteratoren den zuletzt eingefügten Wert.	`Vector`, `ArrayList`, `LinkedList`, `HashSet`, `TreeSet`
Nehmen Sie die oben gefüllten Datenstrukturen und suchen Sie mit Hilfe der Binärsuche den zuletzt eingefügten Wert. Die Implementierung der Binärsuche finden Sie in der Klasse `Collection`.	`Vector`, `ArrayList`
Nehmen Sie die oben gefüllten Datenstrukturen und suchen Sie mit Hilfe der in den Datenstrukturen vorhandenen Methoden den zuletzt eingefügten Wert.	`Vector`, `ArrayList`, `LinkedList`, `HashSet`, `TreeSet`

Nehmen der Messungen der Laufzeiten mit unterschiedlichen Datenstrukturen sollen Sie auch ermitteln, inwiefern sich die Zeitaufwände zwischen Iteratoren und Stream-Klassen unterscheiden. Führen Sie dazu die Messungen aus Tabelle 3.2 durch.

Tabelle 3.2: Zeitmessungen für Zugriffstechniken

Operation	Zugriffstechniken
Erzeugen Sie eine `ArrayList` mit `int`-Zufallszahlen im Wertebereich zwischen 0 und 9999.	`add`-Methode der `ArrayList`, `generate`-Methode der Stream-API
Addieren Sie alle geraden Zufallszahlen aus der `ArrayList` des ersten Tests.	`Iterator`-Durchlauf, sequentieller Stream (erzeugt mit der Methode `stream` der `ArrayList`), möglicherweise paralleler Stream[1] (Methode `parallelStream` der `ArrayList`)

Wenn Sie einen sehr schnellen oder sehr langsamen Computer haben, dann müssen Sie in den oben genannten Punkten die Anzahl der einzufügenden Daten eventuell anpassen, damit Sie einerseits aussagekräftige Zahlen bekommen und andererseits nicht ewig warten müssen. Lassen Sie Ihre Methoden im selben Programmlauf mehrfach nachein-

[1] Hier werden je nach Verfügbarkeit mehrere Kerne des Prozessors verwendet, so dass unter Umständen eine deutliche Geschwindigkeitssteigerung erkennbar sein kann.

ander laufen, um Schwankungen durch den Einsatz des JIT-Compilers auszugleichen. Implementierung der Zeitmessung:

```
public void methodeXY() {
  // Startzeit holen
  long start = System.currentTimeMillis();
  // Algorithmus laufen lassen
  // ...
  // Endzeit holen
  long end = System.currentTimeMillis();
  // Zeit in Millisekunden
  long durationInMsec = end - start;
}
```

In der optionalen Aufgabe 4.15 sehen Sie, wie Sie eine wesentlich elegantere Lösung erstellen können. Das dazu notwendige Wissen über aspekt-orientierte Programmierung wird allerdings erst später in der Vorlesung vermittelt.

3.8 Datenspeicherung

Erweitern Sie Ihre Lösung aus Aufgabe 3.7 und speichern Sie Ihre Messergebnisse in einer reinen Text-Datei (CSV-Format). Die Datei soll folgenden zeilenweisen Aufbau besitzen:

```
Art der Messung,Datenstrukturname,Zeit in Millisekunden
```

Beispiel:

```
Einfügen,java.util.HashSet,20
```

Jede Messung erhält eine eigene Zeile.

4

Aufgaben ohne Abgabe

Die folgenden Aufgaben dienen nur der weiteren Einarbeitung in Java. Ihre Bearbeitung ist freiwillig möglich, wobei Sie teilweise Musterlösungen im Ilias finden.

4.1 Sieb des Eratosthenes

Notwendige Vorkenntnisse zur Lösung der Aufgabe: Arrays, imperative Sprachelemente, Datentypen.
Berechnen Sie die Primzahlen, die kleiner als eine vorgebene konstante Zahl `MAX` sind. Verwenden Sie dazu den Algorithmus, der unter dem Namen „Sieb des Eratosthenes“ bekannt ist:

1. Schreiben Sie alle Zahlen von 2 bis MAX auf.
2. Streichen Sie dann alle Vielfachen von 2, dann von 3, dann von 5 (wieso nicht 4?), . . . usw.
3. Bis zu welcher Zahl müssen Sie den Algorithmus laufen lassen?
4. Alle jetzt nicht gestrichenen Zahlen sind Primzahlen. Geben Sie diese auf dem Bildschirm aus.

Hinweis zur Implementierung: Verwenden Sie ein Array von Wahrheitswerten (`boolean`). Der Index des Feldes entspricht der Zahl. Verwenden Sie die folgenden Methoden zur Zeitmessung.

- Startzeitpunkt: `long start = System.currentTimeMillis();`
- Endzeitpunkt: `long end = System.currentTimeMillis();`
- Zeitdifferenz in msec: `end - start;`

Messen Sie bitte nur die Zeit für die Berechnung (ohne Ausgabe). Um zu testen, ob Ihr Algorithmus korrekt ist, fügen Sie die folgende Methode in Ihren Code ein und rufen diese mit jeder Zahl sowie Ihrem Ergebnis (`true` = ist Primzahl, `false` = ist keine Primzahl) auf. Die Methode `testPrim` gibt nur im Fehlerfall eine Meldung aus („no news is good news“). Sie müssen die Funktionsweise der Methode nicht verstehen.

```
/**
  * Test auf Primzahl (Überprüfung der Implementierung).
  * @param numberToTest Die auf Primzahl zu testende Zahl.
  * @param testPrediction Vermutung des Studenten.
  *                       <ul>
  *                         <li> true: Ist eine Primzahl.
  *                         <li> false: Ist keine Primzahl.
  *                       </ul>
  */
  private static void testPrim(int numberToTest,
                               boolean testPrediction) {
    BigInteger bi = new BigInteger("" + numberToTest);
    if (bi.isProbablePrime(100) != testPrediction) {
      System.err.println("Falsch: Die Zahl " + numberToTest
                         + " ist "
                         + (testPrediction ? " keine "
                                           : " eine ")
                         + "Primzahl");
    }
  }
```

Fügen Sie die folgende Zeile als erste in Ihre Datei ein:

```
import java.math.BigInteger;
```

4.2 Ballspiel

Notwendige Vorkenntnisse zur Lösung der Aufgabe: Arrays, imperative Sprachelemente, Datentypen.
In dieser Aufgabe stellt ein Array ein kleines Spielfeld das, über das ein Ball bewegt wird. Das Spielfeld ist von einer Mauer umgeben.

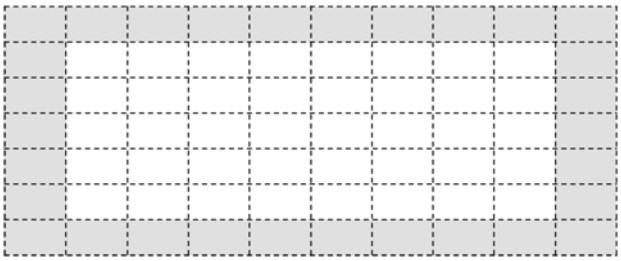

Abbildung 4.1: Aufbau des Spielfeldes

Dabei gelten die folgenden Regeln:

- Ein Element des Arrays ist entweder frei oder durch den Rand belegt.
- Die Größe des Spielfeldes soll beliebig veränderbar sein.

Gehen Sie bei der Entwicklung wie folgt vor:

1. Schreiben Sie eine Methode `init`, die zwei Parameter übergeben bekommt:
   - `int width`: Breite des Feldes
   - `int height`: Höhe des Feldes

2. Die Methode `init` erzeugt aus diesen Daten ein Spielfeld und trägt es als Attribut der Klasse ein. Die Koordinate (0,0) liegt in der linken oberen Ecke.
3. Jetzt wird der Ball an die linke, obere Ecke gesetzt und diagonal in Richtung rechte, untere Ecke bewegt.
4. Stößt der Ball auf den Rand, so wird er gemäß Einfallswinkel = Ausfallswinkel bewegt.
5. Schreiben Sie eine Methode `move`, die den Ball bewegt:
   - Fügen Sie zunächst die folgenden Attribute zur Klasse hinzu:
     - `ballX`: X-Position des Balles
     - `ballY`: Y-Position des Balles
     - `direction`: Richtung des Balles. Überlegen Sie sich, wie Sie die Richtung ausdrücken wollen und welche Möglichkeiten es hier gibt.
   - Die Methode `move` verwendet die Attribute, um die neue Ballposition zu bestimmen.
   - Anschließend trägt sie die neuen Werte in die drei Attribute ein.
6. Gehen Sie dabei von der Vereinfachung aus, dass in der Ecke des Spielfeldes der Ball immer direkt zurückbewegt wird.
7. Schreiben Sie eine weitere Methode `print`, die das Spielfeld in einer einfachen, grafischen Darstellung auf dem Bildschirm ausgibt:
   - Rand: `#`
   - Ball: `*`
   - Freies Feld: (Leerzeichen)
8. Schreiben Sie eine `main`-Methode, die ein Spielfeld erzeugt und den Ball 20 mal bewegt. Nach jeder Bewegung geben Sie das Spielfeld aus.
9. Kommentieren Sie Ihre Lösung!

4.3 Palindrom-Test

Notwendige Vorkenntnisse zur Lösung der Aufgabe: Arrays, Strings, imperative Sprachelemente, Datentypen.
In dieser Aufgaben sollen Sie prüfen, ob ein Wort ein Palindrom ist. Ein Palindrom ist ein Wort, das vorwärts und rückwärts gelesen dasselbe ist, wobei nicht zwischen Klein- und Großbuchstaben unterschieden wird. Beispiele: „Anna“, „Lagerregal“
Um die Aufgabe interessanter zu gestalten, sollen auch Palindromsätze erkannt werden. Damit sind Sätze gemeint, die nach Entfernung von Leerzeichen vorwärts und rückwärts gelesen identisch sind. Satzzeichen wie „.“ sollen nicht vokommen. Beispiel: „Erika feuert nur untreue Fakire“
Die folgenden Wörter und Sätze müssen Sie prüfen. Der Kommentar gibt an, ob es sich im ein Palindrom handelt oder nicht.

```
private static String[] palindromTests = {
   "Anna",                              // true
   "nie",                               // false
   "Reittier",                          // true
   "",                                  // false
   null,                                // false
   "Nasenloch",                         // false
   "Ada",                               // true
   "Rentner",                           // true
   "Anna hetzte Hanna",                 // true
   "Informatik 2",                      // false
   "Renate bittet Tibetaner",           // true
   "Erika feuert nur untreue Fakire" // true
};
```

Die Klasse `Character` erleichert Ihnen den Umgang mit Zeichen. Sie können mit der Methode `char toLowerCase(char)` ein Zeichen in einen Kleinbuchstaben umwandeln und mit `boolean isWhitespace(char)` prüfen, ob ein Zeichen ein unsichtbares Zeichen wie ein Leerzeichen, ein Tabulator usw. ist.

4.4 Bruchrechnung

Schreiben Sie eine Klasse `Bruch`, die einfache Operationen zur Arbeit mir Brüchen zur Verfügung stellt:

- erzeugen eines Bruches mit einem Standard-Konstruktor
- erzeugen eines Bruches mit einem übergebenen Zähler und Nenner
- Auslesen und Schreiben von Zähler sowie Nenner
- Addition und Substraktion zweier Bruchobjekte

Schreiben Sie einen Test, der alle Methoden und Konstruktoren zumindest einmal aufruft.

4.5 Komplexe Zahlen

Erstellen Sie eine Klasse `Complex` zur Verwaltung komplexer Zahlen. Die Klasse hat die privaten Attribute `real` und `imag` sowie die öffentlichen, arithmetischen Operationen `add` zur Addition einer weiteren komplexen Zahl und `absoluteValue` zur Berechnen des Betrages der komplexen Zahl.
Hinweise zu komplexen Zahlen:

- Mathematisches Modell mit zwei vorgegebenen komplexen Zahlen z_1 und z_2:
  $z_1 = real_1 + i * imag_1$
  $z_2 = real_2 + i * imag_2$
- Addition:
  $z_1 + z_2 := real_1 + real_2 + i * (imag_1 + imag_2)$
- Betrag:
  $|z1| = \sqrt{(real_1^2 + imag_1^2)}$

Definieren Sie die Addition als Methode mit nur einem Parameter! Definieren Sie Methoden zum Zugriff auf die privaten Daten und globale Funktionen, die diese Methoden aufrufen. Deklarieren Sie Konstruktoren mit und ohne Parameter.

4.6 Taschenrechner

Notwendige Vorkenntnisse zur Lösung der Aufgabe: Klassen, imperative Sprachelemente, Datentypen.
Schreiben Sie eine Klasse `Calculator`, die die Funktionalität eines Taschenrechners umsetzt. Der Rechner soll die folgenden Methoden besitzen:

- `public void add(double value)`: Addiert `value` zur Zwischensumme.
- `public void sub(double value)`: Subtrahiert `value` von der Zwischensumme.
- `public void neg()`: Negiert das Zwischenergebnis. Das Resultat wird das neue Zwischenergebnis.
- `public void clearResult()`: Löscht das intern gespeicherte Zwischenergebnis.
- `public double getResult()`: Liest das Zwischenergebnis aus.
- `public long getResultAsLong()`: Liest das Zwischenergebnis aus und liefert dieses als mathematisch korrekt gerundete `long`-Zahl zurück.
- `public void ln()`: Berechnet den natürlichen Logarithmus des Zwischenergebnisses und legt das Resultat als neues Zwischenergebnis ab.
- `public void log()`: Berechnet den Logarithmus zur Basis 10 des Zwischenergebnisses und legt das Resultat als neues Zwischenergebnis ab.
- `public void pow(double exp)`: Berechnet $result^{exp}$ und legt das Resultat als neues Zwischenergebnis ab.
- `public void sin()`: Berechnet den Sinus des Zwischenergebnisses und legt das Resultat als neues Zwischenergebnis ab.
- `public void round()`: Rundet das Zwischenergebnis in eine ganze Zahl.
- `public void or(double arg)`: Führt eine bitweise Oder-Verknüpfung von Argument und Zwischenergebnis durch. Das Resultat wird das neue Zwischenergebnis. Runden Sie beide Zahlen vorher!
- `public void and(double arg)`: Führt eine bitweise Und-Verknüpfung von Argument und Zwischenergebnis durch. Das Resultat wird das neue Zwischenergebnis. Runden Sie beide Zahlen vorher!
- `public void not()`: Invertiert alle Bits des Zwischenergebnisses. Das Resultat wird das neue Zwischenergebnis. Runden Sie dazu die Zahl vorher!

Suchen Sie für die Berechnung der mathematischen Ausdrücke passende Methoden aus der JDK-Dokumentation und schreiben Sie eine kleine `main`-Methode für die Klasse `Calculator`, um dessen einwandfreie Funktionsweise zu testen. Wenn Sie wollen, dann können Sie die Korrektheit noch besser mit JUnit-Tests sicherstellen. Denken Sie daran, die Methoden mit Javadoc-Kommentaren zu versehen.
Erweitern Sie Ihren Taschenrechner um einen Ergebnisspeicher:

- `public void memoryClear()`: Löscht den Speicher.
- `public void memoryAdd()`: Addiert die Zwischensumme zum Speicher.
- `public void memorySave()`: Legt die Zwischensumme im Speicher ab.
- `public double memoryRecall()`: Liest den Wert des Speichers aus.

Testen Sie auch diese neue Funktionalität, indem Sie die `main`-Methode erweitern oder neue JUnit-Tests erstellen.

4.7 Restaurantsimulation

Notwendige Vorkenntnisse zur Lösung der Aufgabe: Klassen, Pakete, imperative Sprachelemente, Datentypen, `ArrayList`-Klasse.
Die Lösung der Aufgabe soll die Sitzplatzbelegung in einem Restaurant simulieren. Dieses Restaurant wird von Besuchergruppen betreten. Das Restaurant weist jeder Gruppe einen Tisch zu. Nachdem alle Mitglieder der Gruppe gegessen haben, verlässt die Gruppe das Restaurant, und die Sitzplätze sind wieder frei. Modellierung:

- Ein Restaurant besteht aus mehreren Tischen. Alles andere ist für diese Simulation nicht relevant.
- Jeder Tisch hat eine bestimmte Anzahl von Stühlen.
- Ein Stuhl ist frei oder belegt.
- Eine Besuchergruppe besteht aus einer bestimmten Anzahl Personen.

Eine Besuchergruppe hat immer das folgende Verhalten:

- Restaurant betreten
- Platzsuche gemäß unten aufgeführter Strategie
- Aufenthalt im Restaurant
- Restaurant verlassen

Formal läuft die Platzsuche einer Gruppe mit n Personen so ab, wobei sich Gruppen niemals auftrennen:

1. Versuche, einen noch völlig leeren Tisch mit genau n Stühlen zu finden. Der Tisch muss also die exakt passende Stuhlzahl aufweisen.
2. Versuche, einen völlig leeren Tisch mit mindestens n Stühlen zu finden. Der Tisch darf somit mehr Stühle als Besucher in der Gruppe haben.
3. Suche einen Tisch, an dem noch mindestens n freien Stühle zu finden sind. An diesem Tisch dürfen bereits auch andere Gruppen sitzen. Wenn es mehrere solcher Tische gibt, so muss der genommen werden, an dem nach der Platzierung der Gruppe so wenig Stühle wie möglich frei bleiben.

Vorgehensweise zur Implementierung der Lösung:

1. Finden der Klassen, Implementierung des Verhaltens.
2. Test zur Verifikation der Funktionsweise anhand einer Testklasse. Aufgabe der Testklasse:
   - Erzeugen eines Restaurants mit zufälliger Tischanzahl und zufälliger Sitzplatzanzahl je Tisch.
   - Erzeugen einer Anzahl von Gruppen mit zufälligen Größen.
   - Platzieren der Gruppen. Beachten Sie Sonderfälle (z.B. kein Tisch mehr frei).
   - Gruppen verlassen das Restaurant.

   Nur die Testklasse darf Bildschirmausgaben vornehmen.

Erstellen Sie die Dokumentationskommentare für Ihre Klassen!

4.8 Iterator für die doppelt verkettete Liste

Implementieren Sie einen Iterator für Ihre doppelt verkettete Liste aus Aufgabe 3.2. Dieser Iterator kann im einfachsten Fall die existierende Schnittstelle `Iterator` implementieren. Da sich aber doppelt verkettete Listen sehr effizient vorwärts und rückwärts durchlaufen lassen, sollte Ihre Iterator am besten die vorhandene Schnittstelle `ListIterator` implementieren. Sehen Sie sich dazu diese API der Schnittstelle im Paket `java.util` an. Das Erzeugen des Iterators erfolgt durch Aufruf der Methode `listIterator()` Ihrer Listenklasse. Werfen Sie dazu einen Blick auf die Klasse `java.util.LinkedList`. Dort ist der `ListIterator` ebenfalls vorhanden.

4.9 Programmexport als lauffähige JAR-Datei

Notwendige Vorkenntnisse zur Lösung der Aufgabe: Eclipse-Bedienung, einfache Java-Kenntnisse aus „Informatik 1“.
JAR-Dateien sind Archive („**J**va **AR**chive“) im ZIP-Format, die neben Klassen als Bytecode auch andere Ressourcen wie Bilder usw. enthalten können. Interessant ist insbesondere die Möglichkeit, innerhalb einer sogenannten Manifest-Datei die Klasse im Archiv angeben zu können, die die `main`-Methode des Projektes enthält. Solch eine JAR-Datei kann im Betriebssystem direkt z.B. durch einen Doppelklick ausgewählt werden, um die `main`-Methode aufzurufen. Die Manifest-Datei muss `MANIFEST.MF` heißen und im Verzeichnis `META-INF` des Archivs liegen. Führen Sie die folgenden Schritte in Eclipse aus, um eine ausführbare JAR-Datei zu erstellen:

- Rechtsklick auf das Projekt, „Export“ auswählen
- im Dialog „Java“ und „Runnable JAR File“ auswählen
- Wenn das Projekt in Eclipse mindestens einmal gestartet wurde, lässt sich hier die Start-Konfiguration des Projektes auswählen. In dieser Konfiguration stehen sowohl Informationen wie der Name der Klasse, die die `main`-Methode besitzt, als auch eventuell abhängige Bibliotheken. Die Information zum Start wird verwendet, um die Manifest-Datei anzulegen.
- Unter „Export Destination“ wird angegeben, wie die JAR-Datei heißen und in welches Verzeichnis sie geschrieben werden soll.
- In der Auswahl zum „Library Handling“ wählen Sie für die Beispiele aus der Vorlesung am besten „Package required libraries into generated JAR“. Dabei werden JAR-Dateien, die das Projekt benötigt, in die erzeugte JAR-Datei kopiert und stehen beim Start direkt zur Verfügung.

Testen Sie, ob sich die erzeugte JAR-Datei ausführen lässt. Sehen Sie sich den Inhalt des erzeugten Archivs an (z.B. mit „7Zip“). Interessant ist hier insbesondere die Manifest-Datei.

4.10 Visualisierung der doppelt verketteten Liste

Zeichen Sie die doppelt verkettete Liste aus Aufgabe 3.2 mit Hilfe von JavaFX in ein `Canvas`-Objekt oder verwenden Sie `Shape`-Objekte innerhalb einer `Pane`.

Um die Liste sequentiell durchlaufen zu können, gibt es mehrere Möglichkeiten:

- Verwenden Sie die Methode `E get(int index)`, um mit Hilfe des Indexes auf die einzelnen Elemente der Liste zuzugreifen. Dieser Ansatz ist wegen der linearen Laufzeit für jeden einzelnen Zugriff nicht empfehlenswert, bei kurzen Listen aber tolerierbar.
- Fügen Sie Methoden zur Liste hinzu, um Referenzen auf die Listenelemente „nach außen“ zu reichen. Dieser Ansatz hat den großen Nachteil, dass eigentlich interne Implementierungsdetails der Liste anderen Klassen bekannt gegeben werden.
- Die eleganteste Möglichkeit besteht darin, für die Liste einen Iterator zu implementieren. Wenn Sie Zeit und Lust haben, dann verwenden Sie diesen Ansatz. Vielleicht haben Sie den Iterator ja schon in Aufgabe 4.8 implementiert.

Die Ausgabe der Liste nehmen Sie so vor, wie Sie das in der Vorlesung an der Tafel gesehen haben. Damit die Darstellung hübsch wird, wäre es ganz gut, alle Listenelemente gleich breit werden zu lassen. Leider ist es in JavaFX (noch) nicht möglich, die Breite eines Textes bei Darstellung mit einem Zeichensatz vorab zu berechnen. Verwenden Sie einfach die Klasse `javafx.scene.text.Text` für die einzelnen Listenelemente und verzichten Sie auf identische Breiten. Man könnte diese auch mit den vorhandenen Mitteln von JavaFX erzwingen, allerdings soll die Aufgabe ja nicht allzu komplex werden.

4.11 Einarbeitung in JavaFX

In dieser Aufgabe üben Sie den Umgang mit einfachen Zeichenoperationen sowie den Grafik-Widgets unter JavaFX. Nehmen Sie die kleine Kinderzeichnung „Haus des Nikolaus“ (http://www.mathematische-basteleien.de/nikolaushaus.htm) als Basis.

4.11.1 Canvas

Zeichnen Sie das Haus mit Hilfe primitiver Zeichenoperationen auf einer `Canvas`. Dabei sollen alle Linien schwarz sein und die Fläche des Daches rot ausgefüllt werden. Beim Dach werden Sie feststellen, dass die API von JavaFX keine Dreiecke bereitstellt. Statt dessen können Sie einen Linienzug („Path“) erstellen und diesen ausfüllen. Erforderliche Aufrufe auf dem Grafikkontext:

- `beginPath` startet einen neuen Linienzug.
- `moveTo` setzt den Startpunkt des Linienzugs.
- `lineTo` zieht eine Linie vom vorherigen Punkt zum angegebenen neuen Punkt. Diese Aktion führen Sie mehrfach durch.
- `closePath` schließt den Linienzug. Wenn der letzte Punkt des Linienzugs nicht dem Startpunkt entspricht, dann wird eine zusätzliche Linie zum Startpunkt erzeugt.
- `fill` füllt den Inhalt des Linienzugs.
- `stroke` zeichnet die Linien des Linienzugs.

Neben Sie das Grundgerüst des Beispiels `java2d.PaintCanvas` als Basis für Ihre Lösung. Dann müssen Sie nicht selbst mit Layout-Managern arbeiten.

4.11.2 Pane

Erstellen Sie nun das Haus mit Hilfe der Grafik-Widgets in einer Gruppe (Klasse `Group`). Fügen Sie dann die Gruppe zur `Pane` hinzu. Anschließend registrieren Sie Beobachter an allen Figuren sowie an der `Pane` selbst und geben im Beobachter aus, welche der Figuren angeklickt wurde. Beachten Sie die Unterschiede, wenn Sie im Beobachter an den einzelnen Figuren die Ereignisse mit `consume` „verbrauchen“. Dann erhält die `Pane` die Ereignisse nicht mehr zugestellt.
Für diese Lösung nehmen Sie das Gerüst aus dem Beispiel `shapes.GroupTest` aus der Vorlesung.

4.12 Zeichenprogramm (Gesten)

Hier erweitern Sie das Zeichenprogramm aus Aufgabe 3.5 um Gestenbehandlung, wenn Sie ein Gerät besitzen, dass Gestenerkennung beherrscht. Es sind drei Arten von Gesten vorgesehen:

1. `void rotate(Node node, double angle)`:
   Dreht die übergebene Figur `node` um den Winkel `angle` weiter.
2. `void translate(Node node, double deltaX, double deltaY)`:
   Verschiebt die Figur `node` um die Abstände `deltaX` in X-Richtung bzw. um `deltaY` in Y-Richtung.
3. `void zoom(Node node, double zoomFactor)`:
   Vergrößert oder verkleinert die Figur `node` um den Faktor `zoomFactor`.

4.13 Visualisierung der Collections-Zeitmessungen

Geben Sie die Ergebnisse Ihrer Messungen aus der Aufgabe 3.7 mit Hilfe von JavaFX als Diagramm aus. Das Paket `javafx.scene.chart` besitzt bereits vordefinierte Diagrammtypen.

4.14 Layout-Management mit JavaFX

In dieser Aufgabe üben Sie den Einsatz von Layout-Managern anhand eines sehr einfachen Beispiels.

4.14.1 Einführung

Als Vorgabe erhalten Sie eine Zeichenfläche, die sogenannte Mandelbrot-Mengen zeichnen kann. `http://www.mathematische-basteleien.de/apfelmaennchen.htm` gibt Ihnen eine nähere Einführung, die aber für die Umsetzung der Lösung nicht erforderlich ist. Das Ergebnis der fertigen Anwendung sieht so aus:

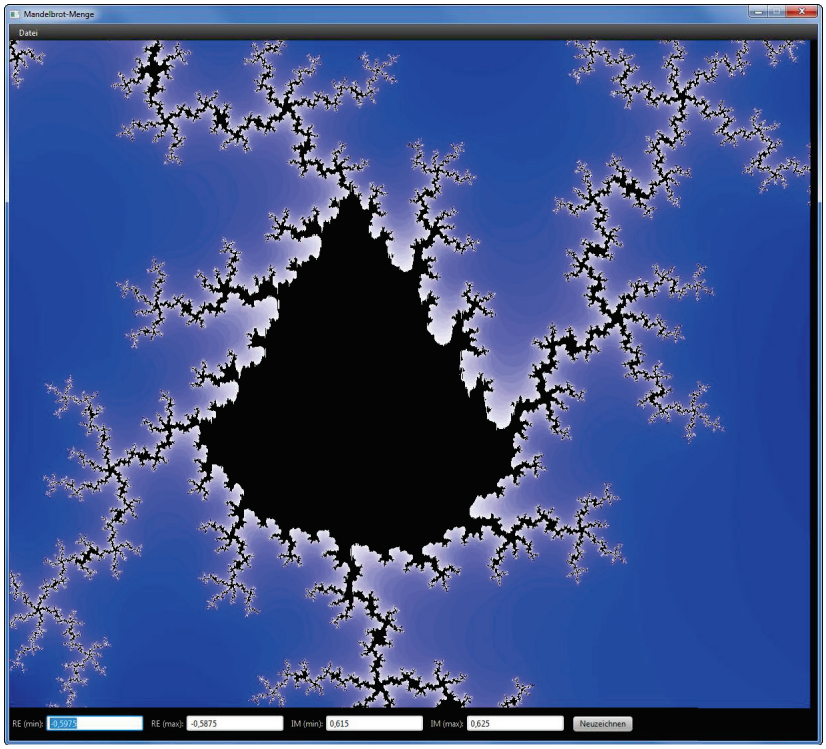

Abbildung 4.2: Musterlösung der Bonusaufgabe

Im vorgegebenen Programmgerüst[1] finden Sie eine Klasse `MandelbrotCanvas`, die von der JavaFX-Basisklasse `Canvas` erbt, vor. Diese Zeichenfläche betten Sie in Ihren grafische Oberfläche ein. Die Klasse bnötigt im Konstruktor die Breite und Höhe in Pixeln. Sie sollten hier möglichst ein 5:4-Verhältnis zwischen beiden Zahlen einhalten. Zusätzlich existiert noch eine Klasse `MandelbrotBean`, die die zur Berechnung erforderlichen Daten kapselt. Diese Daten sind:

- `convergenceSteps`: Anzahl Schritte, bis zu der eine Berechnungsfolge konvergieren muss. Sinnvoll ist hier der Wert 50, Sie können aber gerne etwas experimentieren.
- `reMin`: kleinere reelle Zahl.
- `reMax`: größere reelle Zahl
- `imMin`: kleinere imaginäre Zahl
- `imMax`: größere imaginäre Zahl

Die Zahlen `reMin` und `reMax` müssen im Bereich zwischen `-3` und `2`, `imMin` und `imMax` zwischen `-2` und `+2` liegen. Einige Zahlenkombinationen ergeben recht hübsche Ergebnisse, Auswahl:

[1] Das Gerüst basiert auf der Lösung unter `http://www.hameister.org/JavaFX_Fractal.html`, wurde aber gerade im Hinblick auf die Zeichengeschwindigkeit deutlich verbessert. Versuchen Sie bitte nicht, alle Details der Zeichenflächenimplementierung zu verstehen. Es wurde das sogannten Multithreading eingesetzt, um das Programm während der Ausgabe bedienbar zu halten und die Berechnung von der Ausgabe zu entkoppeln. Multithrading wird erst in einem späteren Semester behandelt.

Tabelle 4.1: Beispielzahlenbereiche

reMin	reMax	imMin	imMax	Hinweis
-3	2	-2	2	deckt den vollständigen Zahlenbereich der Mandelbrot-Menge ab
$-0,567709792$	$-0,557685031$	$0,638956191$	$0,646482313$	
$-0,5975$	$-0,5875$	$0,615$	$0,625$	
$-0,65$	$-0,45$	$0,51$	$0,71$	
$-1,80$	$-1,73$	$-0,035$	$0,035$	

4.14.2 Programmgerüst

Schreiben Sie ein JavaFX-Programmgerüst, das ein einfaches Menü (noch ohne Funktion) besitzt und die Zeichenfläche einbettet. Sie können sich dazu gerne die Grundgerüste des „Game of Life“ oder des Zeichenprogramms ansehen. Rufen Sie auf der Zeichenfläche die Methode `repaint` auf und übergeben Sie das ausgefüllte `MandelbrotBean`. Jetzt sollten Sie die erste Ausgabe sehen.

4.14.3 Formularelemente

Fügen Sie in Ihr Programm unterhalb der Zeichenfläche vier Textfelder ein, über die Sie die vier Bean-Parameter eingeben können. Die Eingabe für `convergenceSteps` ist nicht erforderlich. Setzen Sie vor alle vier Textfelder `Label` mit Bezeichnungen der Textfelder. Fügen Sie außerdem noch eine Taste „Neuzeichnen“ ein, die Ihnen erlaubt, die Methode `repaint` der Zeichenfläche mit den in die Textfelder eingegebenen Werten aufzurufen. Sehen Sie sich dazu die API der Klasse `TextField` an, und finden Sie heraus, wie Sie die eingegebenen Texte auslesen können. Zur Umwandlung der Texte in eine `Double`-Zahl können Sie eine Methode der Klasse `NumberFormat` verwenden.

```
private double convert(String text)
            throws NumberFormatException, ParseException {
  NumberFormat nf = NumberFormat.getInstance();
  // Anzahl Nachkommastellen auf 10 begrenzen
  nf.setMaximumFractionDigits(10);
  // Konvertieren
  return nf.parse(text).doubleValue();
}
```

Das Formular könnte so aussehen (unterer Rand des Fensters):

Abbildung 4.3: Musterlösung der Bonusaufgabe (Formular)

Verwenden Sie nur die aus der Vorlesung bekannten Layouts `FlowLayout`, `HBox`, `VBox` und `BorderPane`.

4.14.4 Beenden des Programms

Ergänzen Sie einen Menüpunkt „Ende“ im Datei-Menü, der es Ihnen erlaubt, das Programm zu beenden. Bei Auswahl des Menüpunktes rufen Sie einfach `System.exit(0)` auf.

4.14.5 Speichern eines Bildes

Ergänzen Sie einen Menüpunkt „Speichern“ im Datei-Menü, der es Ihnen erlaubt, das aktuelle Bild als Datei zu speichern. Sie können der Datei einen festen Namen sowie einen vordefinierten Dateityp geben. Hier finden Sie Tipps zum Speichern eines `Canvas`: http://docs.oracle.com/javafx/2/image_ops/jfxpub-image_ops.htm

4.14.6 Dateiauswahldialog

Es fehlt noch ein Dialog, mit dessen Hilfe Sie den Namen und Typ des zu speichernden Bildes interaktiv festlegen können. Der Dialog soll nach Auswahl des Menüpunktes „Speichern“ angezeigt werden. Sehen Sie sich dazu die Klassen `FileChooserBuilder` und `FileChooser` des Pakets `javafx.stage` an.

4.14.7 Undo- und Redo

Erweitern Sie Ihr Prgramm so, dass Sie durch Klicks auf eine „Zurück“-Taste jeweils die Ergebnisse der vorherigen Berechnungen ansehen können. Beim Klick auf eine „Vor“-Taste sehen Sie die nach einem Klick auf „Zurück“ durchgeführten Berechnungen. Sie bauen also das ein, was in Programmen unter „Undo“ und „Redo“ zu verstehen ist. Verwenden Sie dazu einfach eine `ArrayList` mit den zur Berechung verwendeten Daten (Tipp: Klasse `MandelbrotBean`). Merken Sie sich außerdem noch, welches Element aus der `ArrayList` Sie zur Zeit darstellen. Bei einem Undo-Aufruf stellen Sie die Daten des Beans davor, bei einem Redo die Daten dahinter dar. Haben Sie dagegen über das Formular am unteren Bildrand neue Daten eingegeben, dann löschen Sie alle Einträge in der `ArrayList` ab Ihrer aktuellen Merkposition und fügen das Bean mit den neuen Daten ein.
Wenn Sie wollen, dann können Sie die Tasten auch noch sperren (`setDisabled`), wenn ein Undo oder Redo nicht möglich ist.

4.15 Zeitmessung mit AspectJ

In Aufgabe 3.7 messen Sie die Zeitaufwände, die entstehen, wenn Sie mit unterschiedlichen Datenstrukturen arbeiten. Dabei sind vermutlich die Zeitmessungen in Ihren Lösungen direkt in den einzelnen Methoden vorhanden. Das hat den Nachteil, dass Sie verschiedene Aufgaben mischen. AspectJ hilft Ihnen, genau solche Probleme zu vermeiden. Dazu kopieren Sie Ihre Lösung in ein neues Projekt. Anschließend konvertieren Sie das Projekt in ein AspectJ-Projekt (Klick mit der rechten Maustaste auf das Projekt → „Configure“ → „Convert to AspectJ Project“). Fügen Sie dann die Messungen der Laufzeiten über Aspekte in die Methoden ein. Wenn Sie den Methoden einen einheitlichen Namensaufbau wie beispielsweise `workWith...` geben, dann können Sie die Pointcuts automatisch in alle Methoden mit diesem Namensmuster einfügen. Tipp: Die Startzeit beim Aufruf der Methode speichern Sie in einem Attribut innerhalb des Aspektes.

Das Ergebnis ist eine Lösung mit sauberer Trennung der Anliegen. Die Algorithmen beinhalten keine Fremdaufgaben wie die der Zeitmessungen mehr. Sie können jetzt sogar Zeitmessung in Methoden einfügen, für die Sie nicht einmal den Quelltext besitzen. Weiterhin können Sie Zeitmessungen sehr einfach ein- und ausschalten, indem Sie die Poincuts aktivieren oder deaktivieren.

4.16 Dynamisches Hashverfahren

In der Vorlesung „Informatik 2“ wurden zwei verschiedene Hashverfahren vorgestellt. Weitere sind aus Zeitgründen nicht besprochen worden. Diese Aufgabe stellt einen sehr einfachen Ansatz für dynamische Hashtabellen vor, der auch in ähnlicher Form in der Java-Klasse `HashMap` Einsatz findet. In der Praxis existieren weitere Verfahren, die aus Gründen des Aufwandes hier nicht betrachtet werden sollen.

4.16.1 Einfügen

Überschreitet der Füllgrad der Hashtabelle einen vorgegebenen Wert, dann wird das Array der Hashtabelle doppelt so groß. Dadurch sinkt der Füllgrad wieder. In Abbildung 4.4 ist eine Situation einer dynamischen Hashtabelle zu sehen.

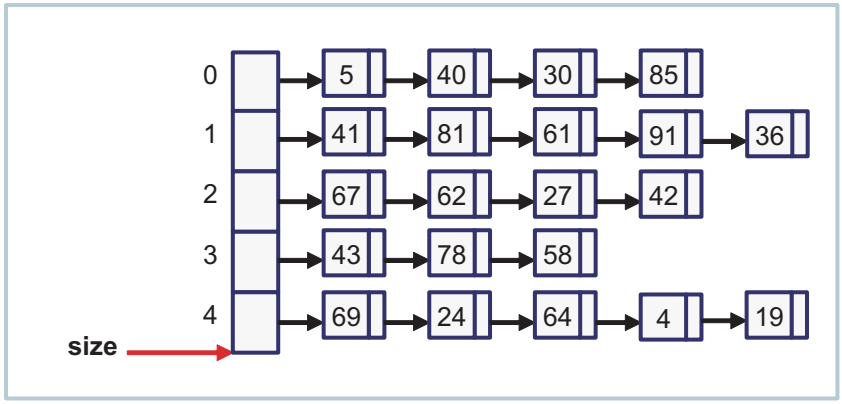

Abbildung 4.4: Aktuelle Situation einer dynamischen Hashtabelle

Es werden hier im Bild nur `Integer`-Werte abelegt, um die Grafik übersichtlich zu halten. `size` beinhaltet die Größe der Hashtabelle. Soweit unterscheidet sich dieser Ansatz nicht von dem, der in der Vorlesung vorgestellt wurde: Neue Einträge werden einfach an die entsprechenden Listen angehängt. Bei einem hohen Füllgrad würden sich also sehr lange Listen bilden. Um dieses Problem zu umgehen, wird ein maximaler Füllgrad `max` der Tabelle festgelegt. In diesem Beispiel hier beträgt er 400%. Im Schnitt sollen also nicht mehr als vier Werte je Tabellenfeld abgelegt werden. Realistischer ist ein Füllgrad von 50–75%.
Wird ein weiterer Wert eingefügt, dann befinden sich 21 Einträge in der Tabelle, sie ist überfüllt (Füllgrad 420%). Somit ist es Zeit für eine Reorganisation mit dem folgenden prinzipiellen Ablauf:

1. Der Speicherplatz der Hashtabelle wird verdoppelt.

2. Die Einträge aller Listen der alten Hashtabelle werden neu einsortiert. Dazu wird ihr Hashwert wie bisher verwendet. Allerdings erfolgt jetzt die Modulo-Berechnung so, dass die doppelt so große Tabelle berücksichtigt wird. Anhand dieses Wertes erfolgt die Einsortierung in die neue Tabelle.
3. Die alte Tabelle wird nicht mehr benötigt und durch die neue ersetzt.
4. `size` wird verdoppelt.

Die Abbildung 4.5 zeigt die Tabelle nach der Reorganisation, wobei die verschobenen Einträge durchgestrichen dargestellt sind.

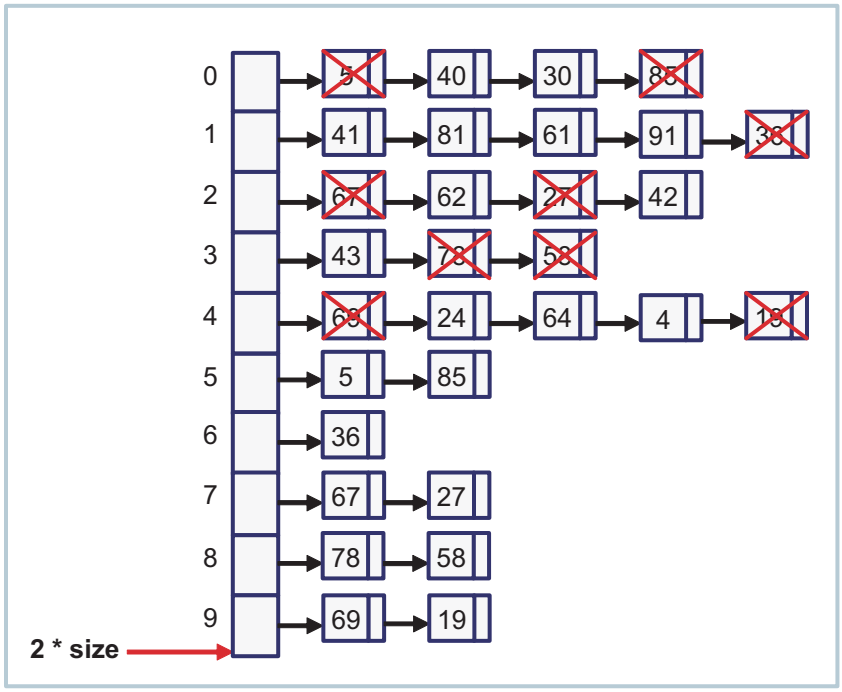

Abbildung 4.5: Dynamische Hashtabelle nach der Reorganisation

4.16.2 Löschen

Beim Löschen dagegen soll nicht auf Einhalten einer Untergrenze untersucht werden. Hier wird nicht mehr benötigter Platz nicht wieder freigegeben.

4.16.3 Aufgabenstellung

Sie können für die Implementierung Hashtabelle gerne das Fragment aus der Vorlesung „Informatik 2“ verwenden und erweitern.

Struktur

Jedes Feld der Hashtabelle besitzt eine verkettete Liste zur Aufnahme der Daten.

Konstruktoren

Die Klasse erhält zwei Konstruktoren:

1. Ein Konstruktor erhält lediglich die Initialgröße der Tabelle als Parameter und verwendet als Schwellwert die Konstante 0.75.
2. Ein weiterer Konstruktor besitzt zwei Parameter. Es handelt sich dabei um die Initialgröße sowie den Schwellwert. Prüfen Sie diese Parameter mit Assertions auf sinnvolle Werte.

Methoden

Die Klasse besitzt die folgenden Methoden:

- `put`: Fügt einen Wert mit dem angegebenen Schlüssel ein. Duplikate sind nicht erlaubt.
- `get`: Liest den Wert zum übergebenen Schlüssel wieder aus. Diese Methode existiert bereits.
- `remove`: Löscht das Paar aus Schlüssel und Wert aus der Tabelle. Als Übergabeparameter erhält die Methode den Schlüssel der zu löschenden Daten. Die Methode gibt als Ergebnis den zum Schlüssel gehörigen Wert zurück. Existiert der Schlüssel nicht in der Tabelle, dann ist der Rückgabewert `null`.

Aus Effizienzgründen darf der Hashwert bei einer Reorganisation nicht neu berechnet werden. Statt dessen wird er zusammen mit Schlüssel und Wert in der Liste abgelegt. Nur die Modulo-Berechnung erfolgt stets erneut. Bauen Sie das Lösungsgerüst aus der Vorlesung so um, dass Sie statt eines Paars, bestehend aus Schlüssel und Wert, ein Tripel aus Schlüssel, Wert und Hashwert speichern. Die Listenelemente nehmen solche Tripel auf.

Testprogramm

Das Testprogramm überprüft alle Methoden eingehend. Schreiben Sie weiterhin eine Methode, die den Inhalt der Hashtabelle ausgibt. Diese Methode wird in einem Test vor und nach dem Einfügen bzw. Löschen aufgerufen.

4.17 Erweiterung des binären Suchbaums

In der Vorlesung wurde Ihnen bereits eine sehr einfache Teilimplementierung eines binären Suchbaumes gezeigt, dessen Implementierung Sie in den Beispielen zur Vorlesung unter dem Namen `SimpleMapMap` finden. Die Implementierung besitzt Methoden zum Einfügen und Suchen.

4.17.1 Löschen

Erweitern Sie die Klasse so, dass Sie auch Elemente aus dem Baum löschen können, indem Sie den zu entfernenden Schlüssel angeben.

4.17.2 Ausgabe als XML-Dokument

Schreiben Sie eine Methode, die den Baum in Form eines XML-Dokumentes ausgegeben kann. Dabei sollen auch die optischen Einrückungen stimmen. Der linke Nachfolger eines Knotens wird immer vor dessen rechtem Nachfolger ausgegeben. Verwenden Sie den Preorder-Durchlauf. Beispielausgabe:

```
<treemap>
  <node key="A" value="42">
    <node key="D" value="567"/>
    <node key="F" value="234">
      <node key="E" value="456"/>
      <node key="H" value="345"/>
    </node>
  </node>
</treemap>
```

Um die Einrückungen sehr einfach zu implementieren, übergeben Sie beim rekursiven Aufruf für die Ausgabe der Kindknoten die aktuelle Einrücktiefe. Diese nehmen die Kindknoten und erhöhen sie für ihre eigenen Ausgaben.

4.17.3 Iterator

Achtung: Diese Aufgabe ist wirklich sehr schwierig und nicht klausurrelevant. Erweitern Sie Ihre Baum-Klasse aus Aufgabe 4.17 so, dass Sie Iteratoren unterstützt. Hinweise:

- Sie müssen mit Hilfe der Iteratoren den Baum komplett sortiert (inorder) durchlaufen können.
- Eine rekursive Lösung ist nicht möglich, da der Iterator nach jedem gefundenen Knoten diesen zurückgibt.
- Verwenden Sie daher die in der Vorlesung aufgezeigte iterative Variante, die ein Stack-Objekt zum Zwischenspeichern der aktuellen Position verwendet.
- Jeder Iterator muss ein eigenes Stack-Objekt haben, da ja eventuell mehrere Iteratoren auf einem Baum operieren. Daher verwenden Sie den Stack als Attribut der Iterator-Klasse.

4.18 Ringpuffer mit Spring

Nehmen Sie sich die Aufgabe 3.2.3 als Vorlage und kopieren Sie sie. Die Lösung soll so umgebaut werden, dass die Abhängigkeiten zwischen der Liste und dem Ringpuffer mit Spring aufgelöst werden. Da Sie jetzt eine Liste statt eines Arrays fester Größe verwenden, kann der Rinpuffer natürlich auch „beliebig“ viele Elemente aufnehmen. Sie müssen also nicht mehr prüfen, ob er voll ist.

4.18.1 Vorbereitungen

Führen Sie diese Schritte durch:

1. Erstellen Sie eine Kopie der Aufgabe 3.2.3.
2. Legen Sie Im Projekt einen Ordner mit dem Namen `lib` an.

3. Kopieren Sie in diesen Ordner alle Bibliotheken, die Spring benötigt. Sie können einfach die Dateien aus dem Spring-Beispielprojekt der Vorlesung nehmen. Dieses Vorgehen mit lokalen Kopien ist nicht optimal. Besser wäre es, die Bibliotheken zentral abzulegen und im Projekt darauf zu verweisen. Aber wir wollen es nicht zu kompliziert machen.
4. Jetzt muss Ihr Projekt die Bibliotheken auch verwenden. Dazu müssen Sie mit der rechten Maustaste auf das Projekt klicken und dann `Properties` auswählen. Im daraufhin erscheinenden Dialog wählen Sie auf der linken Seite im Baum `Java Build Path` aus. Dieser beinhaltet alle Bibliotheken und Projekte, die dieses Projekt für seine Arbeit benötigt. Klicken Sie im Dialog auf `Add Jars...` und selektieren Sie alle Jar-Dateien aus dem `lib`-Verzeichnis Ihres Projektes.

Nun ist Ihr Projekt für den Einsatz von Spring vorbereitet.

4.18.2 Umbau

Erstellen Sie zunächst eine Abstraktion Ihrer Liste, indem Sie ein Interface mit allen öffentlichen Methoden der Liste schreiben. Diese Schnittstelle wird von Ihrer Liste implementiert. Deklarieren Sie die beteiligten Klassen (`LinkedList` und den Ringpuffer) als Spring-Komponenten. Markieren Sie die Referenzen, so dass Spring die Abhängigkeiten auflösen kann. Der Ringpuffer soll dabei nur die Schnittstelle kennen, die von der Liste implementiert wird. Es bleibt dabei Ihnen überlassen, ob Sie sich für die XML-Deklarationen, Annotationen oder die Java-basierte Konfiguration entscheiden. Stellen Sie abschließend die `main`-Methode so um, dass Sie sich eine Referenz auf den Ringpuffer holt und diesen für einige Tests verwendet. Dazu verschieben Sie `main` in eine neue Klasse.

5

Bonusaufgabe

Diese Aufgabe soll Sie motivieren, sich noch etwas weiter mit Java zu beschäftigen. Als Belohnung erhalten Sie für die Abgabe einer korrekten Lösung 5 Bonuspunkte in der Klausur. Somit können Sie sich schon einmal vorab um eine Notenstufe verbessern (außer, wenn Sie ohnehin eine 1,0 schreiben werden).

5.1 Rechner für UPN-Ausdrücke

Sie sollen Postfix-Ausdrücke (auch als UPN = umgekehrte polnische Notation bekannt) berechnen. Beispiel (das Ergebnis ist 34365):

```
15 42 18 + 61 24 - * 71 + *
```

Die Berechnung des Ausdruckes erfolgt so:

- Der Postfix-Ausdruck wird von links nach rechts ausgewertet.
- Zahlen werden auf einem Stack zwischengespeichert.
- Bei einem zweistelligen Operator werden die oberen beiden Argumente vom Stack genommen, mit dem Operator verknüpft und wieder auf dem Stack abgelegt.
- Zum Schluss liegt das Ergebnis als einzige Zahl auf dem Stack.

Der Vorteil der Postfix-Notation gegenüber der bekannteren Infix-Notation solcher mathematischen Ausrücke besteht darin, dass keine Klammern benötigt werden.

5.1.1 Aufbau des Ausdrucks

Die UPN-Ausdrücke liegen als Zeichenketten (Strings) vor. Es sind nur natürliche Zahlen sowie die zweistelligen Operatoren `+`, `-`, `*` und `/` zulässig. Zahlen und Operatoren sind durch eine oder mehrere Leerzeichen bzw. Tabulatoren voneinander getrennt.

5.1.2 Berechnung

Berechnen Sie jetzt den Wert solch eines Ausdrucks. Den Algorithmus dazu haben Sie bereits in Abschnitt 5.1 kennengelernt. Beachten Sie bitte diese Hinweise und verwenden Sie unbedingt die erwähnten Klassen und Methoden:

- Die Zeichenkette muss in einzelne „Stücke“ (Operatoren oder Zahlen) zerlegt werden. Verwenden Sie dazu die existierende Klasse `StringTokenizer` aus dem Paket `java.util`. Die Klasse zerlegt Zeichenketten in einzelne „Stücke“, die sogenannten Tokens. Trennzeichen wie Leerzeichen werden automatisch ignoriert.
- Zur Umwandlung der Tokens in Zahlen verwenden Sie die Methode `parseInt` der Klasse `Integer`.

Sie können bei der Aufgabenstellung immer davon ausgehen, dass der UPN-Ausdruck korrekt geformt ist und keine Fehler enthält.

6

Literaturverzeichnis

[Ull02] Ullenboom C.: Java ist auch eine Insel, Galileo Computing (kostenlos als HTML-Dokumente unter http://www.tutego.de/javabuch/)

[Ull03] Ullenboom C.: Java ist nicht nur eine Insel, Galileo Computing

[RaScWi11] Ratz D., Scheffler J., Seese D., Wiesenberger J.: Grundkurs Programmieren in Java. Hanser Fachbuchverlag, 2011

[Cetus] Cetus-Links: http://www.cetus-links.org

[ORA01] Oracle: Java-Tutorial, http://download.oracle.com/javase/tutorial/

[ORA02] Oracle: JDK-Referenz http://www.oracle.com/technetwork/java/javase/documentation/index.html

[ORA03] Oracle: Code Conventions for the Java™ Programming Language http://www.oracle.com/technetwork/java/codeconv-138413.html

[ORA04] Oracle: JavaFX-Referenz und Tutorium http://docs.oracle.com/javafx/index.html

7

Stichwortverzeichnis

Symbole

C

E

F

I

J

N

O

T

U