

Gr. 1, E. Pitzer

Name \_\_\_\_\_ Aufwand in h \_\_\_\_\_

Gr. 2, F. Gruber-Leitner

Punkte \_\_\_\_\_ Kurzzeichen Tutor / Übungsleiter \_\_\_\_\_ / \_\_\_\_\_

**Verschiebe-Puzzle – A\*-Algorithmus****(24 Punkte)**

Ein sehr bekanntes und beliebtes Rätsel ist das Verschiebe-Puzzle, das oft auch als 8- bzw. 15-Puzzle bezeichnet wird. Das Spiel besteht aus 8 (15) Kacheln, die von 1 bis 8 (15) durchnummeriert sind, die auf einem 3x3- (4x4-) Spielfeld angeordnet sind. Da ein Feld frei bleibt, können gewisse Kacheln verschoben werden. Die Aufgabe besteht nun darin, ausgehend von einer beliebigen Anordnung der Kacheln, diese ausschließlich durch Verschiebungen in die richtige Reihenfolge zu bringen (siehe nebenstehende Abbildung).



Eine Möglichkeit, dieses Problem zu lösen, ist Backtracking. Allerdings wird bei Anwendung dieses Verfahrens der Suchraum sehr groß, was zu nicht vertretbaren Rechenzeiten führt. Ein effizienter Algorithmus zur Lösung dieses Problems ist der sogenannte A\*-Algorithmus, der von Peter Hart, Nils Nilsson und Bertram Raphael bereits 1968 entwickelt wurde. Eine übersichtliche Darstellung des Algorithmus findet man beispielsweise auf der deutschen Wikipedia unter [http://de.wikipedia.org/wiki/A\\*-Algorithmus](http://de.wikipedia.org/wiki/A*-Algorithmus). Der A\*-Algorithmus wird oft zur Wegsuche bei Routenplanern eingesetzt. Er ist aber auch auf die hier angeführte Problemstellung anwendbar.

Der Grund A\*-Algorithmus enumeriert grundsätzlich auch alle möglichen Lösungsvarianten, allerdings versucht er, zunächst den erfolgsversprechendsten Weg zum Ziel zu verfolgen. Erst dann werden weitere Varianten untersucht. Findet der Algorithmus auf diese Weise bereits frühzeitig eine Lösung, müssen viele Lösungsvarianten erst gar nicht evaluiert werden. Damit der Algorithmus beim Durchwandern des Lösungsraums in die erfolgsversprechendste Richtung weitergehen kann, benötigt er eine Abschätzung der Kosten, die auf dem verbleibenden Weg zum Ziel anfallen werden. In unserer Problemstellung kann für diese Kostenfunktion  $h(x)$  die Summe der Manhattan-Distanzen (= Distanz in x-Richtung + Distanz in y-Richtung) aller Kacheln zu ihrer Zielposition herangezogen werden. Wenn  $g(x)$  die Kosten von der Ausgangskonfiguration bis zur Konfiguration  $x$  bezeichnet, stellt  $f(x) = g(x) + h(x)$  eine Abschätzung der Kosten von der Ausgangs- zur Zielkonfiguration dar, wobei der Weg zum Ziel über  $x$  verläuft.

Implementieren Sie die Lösung in folgenden Schritten:

- Gehen Sie bei der Implementierung testgetrieben vor. Implementieren Sie die nachfolgend angeführten Klassen Methode für Methode und geben Sie für jede Methode zumindest einen einfachen Testfall an. Erstellen Sie zunächst nur den Methodenrumpf mit einer Standardimplementierung, die nur syntaktisch korrekt sein muss. Implementieren Sie dann für diese Methode die Unittests, deren Ausführung zunächst fehlschlagen wird. Erweitern Sie anschließend die Implementierung der Methode so lange, bis alle Unittests durchlaufen. Erst wenn die Methoden-bezogenen Tests funktionieren, können Sie komplexere Tests erstellen.

Eine Testsuite mit einigen Tests wird Ihnen auf der E-Learning-Plattform zur Verfügung gestellt. Erweitern Sie diese Testsuite so wie beschrieben. Ihre Implementierung muss die vorgegebenen und die von Ihnen hinzugefügten bestehen.

- b) Implementieren Sie zunächst eine Klasse **Board**, die eine Board-Konfiguration repräsentieren kann und alle notwendigen Operationen auf einem Spielbrett unterstützt. **Board** soll folgende Schnittstelle aufweisen:

```
public class Board implements Comparable<Board> {
    // Board mit Zielkonfiguration initialisieren.
    public Board(int size);

    // Überprüfen, ob dieses Board und das Board other dieselbe Konfiguration aufweisen.
    public boolean equals(Object other);

    // <1, wenn dieses Board kleiner als other ist.
    // 0, wenn beide Boards gleich sind
    // >1, wenn dieses Board größer als other ist.
    public int compareTo(Board other);

    // Gibt die Nummer der Kachel an der Stelle (i,j) zurück, Indizes beginnen bei 1.
    // (1,1) ist somit die linke obere Ecke.
    // Wirft die Laufzeitausnahme InvalidBoardIndexException.
    public int getTile(int i, int j);

    // Setzt die Kachelnummer an der Stelle (i,j) zurück. Wirft die Laufzeitausnahmen
    // InvalidBoardIndexException und InvalidTileNumberException
    public void setTile(int i, int j, int number);

    // Setzt die Position der leeren Kachel auf (i,j)
    // Entsprechende Kachel wird auf 0 gesetzt.
    // Wirft InvalidBoardIndexException.
    public void setEmptyTile(int i, int j);

    // Zeilenindex der leeren Kachel
    public int getEmptyTileRow();

    // Gibt Spaltenindex der leeren Kachel zurück.
    public int getEmptyTileColumn();

    // Gibt Anzahl der Zeilen (= Anzahl der Spalten) des Boards zurück.
    public int size();

    // Überprüft, ob Position der Kacheln konsistent ist.
    public boolean isValid();

    // Macht eine tiefe Kopie des Boards.
    // Vorsicht: Referenztypen müssen neu allokiert und anschließend deren Inhalt kopiert werden.
    public Board copy();

    // Erzeugt eine zufällige lösbare Konfiguration des Boards, indem auf die bestehende
    // Konfiguration eine Reihe zufälliger Verschiebeoperationen angewandt wird.
    public void shuffle();

    // Verschiebt leere Kachel auf neue Position (row, col).
    // throws IllegalMoveException
    public void move(int row, int col);

    // Verschiebt leere Kachel nach links. Wirft Laufzeitausnahme IllegalMoveException.
    public void moveLeft();

    // Verschiebt leere Kachel nach rechts. Wirft IllegalMoveException.
    public void moveRight();

    // Verschiebt leere Kachel nach oben. Wirft IllegalMoveException.
    public void moveUp();
}
```

```

// Verschiebt leere Kachel nach unten. Wirft IllegalMoveException.
public void moveDown();

// Führt eine Sequenz an Verschiebeoperationen durch. Wirft IllegalMoveException.
public void makeMoves(List<Move> moves);
}

```

- c) Zur Implementierung des A\*-Algorithmus benötigt sie die Hilfsklasse **SearchNode**. Damit kann man den Weg von einem **SearchNode** zum Startknoten zurückverfolgen, da dieser mit seinem Vorgängerknoten verkettet ist. Ein **SearchNode** kennt die Kosten vom Startknoten bis zu ihm selbst. Ein **SearchNode** kann auch eine Schätzung für den Weg zum Zielknoten berechnen.

```

public class SearchNode implements Comparable<SearchNode> {
    // Suchknoten mit Board-Konfiguration initialisieren.
    public SearchNode(Board board);

    // Gibt Board-Konfiguration dieses Knotens zurück.
    public Board getBoard();

    // Gibt Referenz auf Vorgängerknoten zurück.
    public SearchNode getPredecessor();

    // Setzt den Verweis auf den Vorgängerknoten.
    public void setPredecessor(SearchNode predecessor);

    // Gibt Kosten (= Anzahl der Züge) vom Startknoten bis zu diesem Knoten zurück.
    public int costsFromStart();

    // Gibt geschätzte Kosten bis zum Zielknoten zurück. Die Abschätzung
    // kann mit der Summe der Manhattan-Distanzen aller Kacheln erfolgen.
    public int estimatedCostsToTarget();

    // Setzt die Kosten vom Startknoten bis zu diesem Knoten.
    public void setCostsFromStart(int costsFromStart);

    // Gibt Schätzung der Wegkosten vom Startknoten über diesen Knoten bis zum Zielknoten zurück.
    public int estimatedTotalCosts();

    // Gibt zurück, ob dieser Knoten und der Knoten other dieselbe Board-Konfiguration darstellen.
    // Vorsicht: Knotenkonfiguration vergleichen, nicht die Referenzen.
    public boolean equals(Object other);

    // Vergleicht zwei Knoten auf Basis der geschätzten Gesamtkosten.
    // <1: Kosten dieses Knotens sind geringer als Kosten von other.
    // 0: Kosten dieses Knotens und other sind gleich.
    // >1: Kosten dieses Knotens sind höher als Kosten von other.
    public int compareTo(SearchNode other);

    // Konvertiert die Knotenliste, die bei diesem Knoten ihren Ausgang hat, in eine Liste von Zügen.
    // Da der Weg in umgekehrter Reihenfolge gespeichert ist, muss die Zugliste invertiert werden.
    public List<Move> toMoves();
}

```

d) Implementieren Sie schließlich den A\*-Algorithmus in der Klasse `SlidingPuzzle`.

```
public class SlidingPuzzle {  
    // Berechnet die Zugfolge, welche die gegebene Board-Konfiguration in die Ziel-Konfiguration  
    // überführt. Wirft NoSolutionException (Checked Exception), falls es eine keine derartige  
    // Zugfolge gibt.  
    public List<Move> solve(Board board);  
  
    // Gibt die Folge von Board-Konfigurationen auf der Konsole aus, die sich durch  
    // Anwenden der Zugfolge moves auf die Ausgangskonfiguration board ergibt.  
    public void printMoves(Board board, List<Move> moves);  
}
```

Verwenden Sie bei Ihrer Lösung so weit wie möglich die Behälterklassen des JDK. Setzen Sie insbesondere bei der Implementierung des A\*-Algorithmus (`SlidingPuzzle.solve()`) eine Prioritätswarteschlange (`PriorityQueue`) für die Speicherung Liste der offenen Knoten und eine sortierte Menge (`Set`) für die Verwaltung der Liste der geschlossenen Knoten ein.

# 1 Verschiebe Puzzle A\*-Algorithmus

## 1.1 Lösungsidee

Folgend ist die Dokumentation für die Aufgabenstellung Verschiebe Puzzle mit dem A\*-Algorithmus lösen angeführt.

Es werden zwar Klassenspezifikationen vorgegeben, jedoch soll nicht stur diesen gefolgt werden, obwohl die Semantik beibehalten werden soll.

Bezüglich der konkreten Spezifikation der APIs und Implementierungen sei auf den Bereich *Source* verwiesen, wo einerseits beschrieben wird warum die API oder Implementierung so gewählt wurde und andererseits ist im angeführten Source die Javadoc enthalten. Ich finde es besser den Source über Javadoc zu kommentieren als in einem externen Dokument.

### 1.1.1 Maven Projekt

Dieses Mal soll das Projekt als Maven Projekt implementiert werden, da es sich hierbei um ein bekanntes und meist genutztes Framework für Build Konfiguration handelt und es sehr einfach erlaubt Dependencies einzubinden und auch automatisch die Sources dieser Dependencies zur Verfügung stellt. Dies sollte kein Problem darstellen, da Eclipse LUNA bereits mit Maven ausgeliefert wird. Es soll gewährleistet werden, dass es keine speziellen Konfigurationen erfordert um das Projekt zu importieren und es zum Laufen zu bringen.

Jedoch sei angemerkt das ein *Update Projekt* von Nöten sein wird um alle Dependencies von den Maven Repositories zu laden. Des Weiteren sollen keine Änderungen an der Standard *settings.xml* vorgenommen werden.

Für weitere Informationen bezüglich der Maven Konfiguration sei auf den Bereich *Projekt Konfiguration* verwiesen, in dem die *pom.xml* angeführt ist. Dort ist auch beschrieben mit welchen Maven Build command das Projekt gebuildet werden soll, da es auch Konfigurationen geben soll, die einen report für die Tests erstellen.

### 1.1.2 JUnit Tests

Wie verlangt soll testgetrieben vorgegangen werden wobei anstatt einer riesigen Testklasse je getesteter Methode eine eigene Testklasse implementiert werden soll.

So sind Tests übersichtlicher und es kann auch keinen Namenskonflikte geben. Des Weiteren wird es pro getesteter Methode sicherlich mehr als eine Testmethode geben.

Bei den JUnit Tests soll sich an folgende Konvention gehalten werden:

1. **Root Package:** Alle Test-Klassen sollen im package *at.fh.ooe.swe.test.\** zusammengefasst werden. Dadurch liegen alle Testklassen in einer eignen Domain und sind völlig unabhängig vom dem eigentlichen Source
2. **API-Package:** Alle Test Ressourcen wie z.B.: abstrakte Testklassen sollen im Package *at.fh.ooe.swe.test.api.\** liegen. Dadurch ist erkennbar das es sich hierbei um eine API für die Tests handelt des getesteten Projektes handelt.
3. **Test Packages:** Alle tatsächlichen Tests sollen in Packages liegen die wie folgt aufgebaut sein sollen '*at.fh.ooe.swe.test.<sourcePackage>.<testedClassName>(at.fh.ooe.swe.test.puzzle.impl.boardImpl.\*)*', wobei man sich an die Java Bean Convention zu halten hat.
4. **Test-Klassen:** Die Test-Klassen Name sollen wie folgt aufgebaut sein: '*<testedMethodName>Test (SolveTest, MakeMoveTest, usw.)*'. Ein Spezialfall könnten die Klassen sein, die zur Verfügung gestellt worden, wenn diese in Konflikt mit bestehenden Testklassen stehen.

## Übung 3

5. **Test-Methoden:** Die Testmethoden sollen so benannt werden, dass man leicht erkennen kann was getestet wird. Ein Präfix mit *test* ist zu vermeiden, da ja klar ist das es sich bei Methoden in Testklassen um Tests handelt. (*nullIterable*, *validAllFieldsRowPerRowLeftToRight*, *usw.*)

Für die JUnit Test soll JUnit4 verwendet werden wobei hierbei so viele Features wie möglich genutzt werden sollen. Da sicherlich bei den nächsten Übungen auch JUnit Test implementiert werden müssen, sollen alle Ressourcen, die bei anderen Tests wiederverwendet werden können in ein eigenes Projekt ausgelagert werden *junit-test-suite*, sodass diese Ressourcen einfach in anderen Projekten verwendet werden können. Es ist darauf zu achten, dass es absolut keine Abhängigkeiten zu dem eigentlichen Projekt gibt und das die Implementierungen so aufgebaut sind, dass sie für jedes anderes Projekt ohne Modifikation wiederverwendet werden können.

Da wir dazu gezwungen sind in dieser Dokumentation auch die Testresultate anzuführen und die Screenshots von JUnit im Eclipse zu wenig erscheinen, soll ein Logging implementiert werden, dass die Testresultate auf der Konsole ausgibt. Hierbei sollen *TestWatcher* implementiert werden, die einerseits die Invocation einer Testklasse sowie andererseits die Invocation einer Testmethode auf die Konsole loggen. Hierbei soll ein bekanntes Logging Framework namens Log4j verwendet werden.

### 1.2 A\*-Algorithmus

Da der A\*-Algorithmus bei dieser Problemstellung eine riesige Lösungsmenge produziert könnte, muss dieser optimiert werden.

Pro untersuchten Knoten können maximal 4 Nachfolgeknoten gefunden werden, was bedeutet, dass im schlimmsten Fall pro untersuchter Ebene der Lösungsbaum um den Faktor 4 anwächst.

Folgend ist ein Beispiel für den worst case angeführt wobei *M* die zu untersuchende Lösungsmenge darstellt:

1.  $M = \{N1\}$
2.  $M = \{N1, N2, N3, N4, N5\}$
3.  $M = \{N1, N2, N3, N4, N5, N6, N7, N8, N9\}$
4. ...

Es ist zu sehen dass bereits bei der dritten Ebene sich 9 Elemente in der zu untersuchenden Lösungsmenge befinden. Daher muss der Algorithmus so optimiert werden, wobei die noch nicht untersuchten Knoten (offen Knoten) und die bereits untersuchten Knoten (geschlossene Knoten) geprüft werden müssen ob bereits Knoten untersucht wurden, die geringere Kosten aufweisen. Ist dem so, dann soll der Nachfolgeknoten mit den höheren Kosten von der Untersuchung ausgeschlossen werden, was die Lösungsmenge so klein wie möglich hält.

Hierbei stellen aber die Implementierungen `node.hashCode();` `node.equals(Object other)` der Klasse `SearchNode<T>` ein Problem dar, da aufgrund der verwendeten `PriorityQueue<T>` diese Methoden die Kosten mit aufnehmen müssen.

Ansonsten würden bei dem Versuch Knoten aus der Queue zu löschen alle Knoten gelöscht werden, die dasselbe Board halten aber unterschiedliche Kosten.

Des Weiteren macht es dieser Umstand unmöglich einen Knoten mit einem bestimmten gesetzten Board zu erhalten, da ja auch die Kosten mit involviert sind.

Eine Verwendung des Containers `Set<T>`, `List<T>` scheidet ebenfalls aus da wir bei der untersuchung der offenen und geschlossenen Knoten Knoten erhalten müssen, die lediglich dasselbe `Board<T>` halten ohne Rücksicht auf deren Kosten.

Daher wurden folgende Container gewählt:

## Übung 3

---

1. **PriorityQueue<T>**: Die Queue, die die Knoten mit den geringsten Kosten vorne anreicht und diese zuerst zurückliefert. Erfordert bei `node.hashCode(); node.equals(Object other)` die Miteinbeziehung der Kosten.
2. **HashMap<Board<T>, SearchNode<T>>**: Für die bereits untersuchten und offenen Knoten. Hierbei haben wir konstante Laufzeit beim Hinzufügen und Erhalten eines Knoten aus der Map, da hier über die Hashfunktion `board.hashCode()` die Knoten in der Map adressiert werden. Ebenfalls wird so das Problem mit `node.hashCode(); node.equals(Object other)` umgangen. (Beinahe konstante Laufzeit aufgrund von Kollisionen)

Der Algorithmus soll versuchen die Nachfolgeknoten des aktuell untersuchten Knoten in den Container, offene wie auch geschlossenen Knoten, zu finden und zu entscheiden ob diese Knoten des Containers geringere Kosten besitzen. Wenn ja werden die aktuellen Nachfolgeknoten ignoriert und nicht weiter untersucht, da es schon einen besseren Weg zu diesem Board gibt.

## Übung 3

### 1.3 Project Konfigurationen

Folgend sind die Projektkonfigurationen angeführt wie *pom.xml* und *log4j.properties*.

Wenn das Projekt in Eclipse importiert wird, so muss zuerst ein *Update Project* über das Kontextmenu des Projekts *<maven\_project>*- *Maven - Update Project* gemacht werden, damit die Dependencies in das lokale Maven Repository geladen werden.

Danach kann das Projekt mit folgenden maven-build command gebuildet werden:

*clean test-compile surefire-report:report site*

1. **clean:** Löscht Inhalt des Verzeichnisses *\${basedir}/target*
2. **test-compile:** Kompiliert, testet und packaged das Projekt in das Verzeichnis *\${basedir}target*
3. **surefire-report:resport:** Erstellt den Surefire Report in dem Verzeichnis *\${basedir}/target/surefire-reports*
4. **site:** Erstellt einen Maven Report in dem Verzeichnis *\${basedir}/target/site* mit allen Projektinformationen und beinhaltet auch die Surefire Reports

Ebenfalls wird *log4j* als Logging Framework verwendet und diese Framework benötigt eine Konfiguration für die verwendeten Logger.

Da die Logger nur für die Tests verwendet werden, ist sie im Verzeichnis *\${basedir}/src/test/resources* enthalten und nicht in *\${basedir}/src/main/resources*.

#### 1.3.1 pom.xml

Folgend ist die *pom.xml* des Projekts angeführt.

Es wird hierbei das *maven-compiler-plugin* und *maven-surefire-plugin* verwendet. Letzteres wird dazu verwendet um den Test Report zu erstellen.

Listing 1: pom.xml

```

1 <project xmlns="http://maven.apache.org/POM/4.0.0"
  ↳ xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  ↳ http://maven.apache.org/xsd/maven-4.0.0.xsd">
3   <modelVersion>4.0.0</modelVersion>
4   <groupId>at.fh.ooe.swe4</groupId>
5   <artifactId>n-puzzle-problem</artifactId>
6   <version>0.0.1-SNAPSHOT</version>
7   <name>Puzzle</name>
8   <dependencies>
9     <dependency>
10      <groupId>junit</groupId>
11      <artifactId>junit</artifactId>
12      <version>4.12</version>
13    </dependency>
14    <dependency>
15      <groupId>commons-collections</groupId>
16      <artifactId>commons-collections</artifactId>
17      <version>3.2.1</version>
18    </dependency>
19    <dependency>
20      <groupId>commons-lang</groupId>
21      <artifactId>commons-lang</artifactId>
22      <version>2.6</version>
23    </dependency>
24    <dependency>
25      <groupId>at.fh.ooe.swe4</groupId>

```



## Übung 3

```

26     <artifactId>junit4-test-suite</artifactId>
27     <version>0.0.1-SNAPSHOT</version>
28 </dependency>
29 </dependencies>
30 <build>
31   <plugins>
32     <plugin>
33       <artifactId>maven-compiler-plugin</artifactId>
34       <version>3.3</version>
35       <configuration>
36         <source>1.8</source>
37         <target>1.8</target>
38         <!-- Disable annotation processing via compiler plugin. -->
39         <compilerArgument>-proc:none</compilerArgument>
40       </configuration>
41     </plugin>
42     <plugin>
43       <groupId>org.apache.maven.plugins</groupId>
44       <artifactId>maven-surefire-report-plugin</artifactId>
45       <version>2.9</version>
46       <configuration>
47         <redirectTestOutputToFile>true</redirectTestOutputToFile>
48       </configuration>
49     </plugin>
50   </plugins>
51 </build>
52 <reporting>
53   <plugins>
54     <plugin>
55       <groupId>org.apache.maven.plugins</groupId>
56       <artifactId>maven-surefire-report-plugin</artifactId>
57       <version>2.9</version>
58       <configuration>
59         <redirectTestOutputToFile>true</redirectTestOutputToFile>
60       </configuration>
61     </plugin>
62   </plugins>
63 </reporting>
64 </project>

```

### 1.3.2 log4j.properties

Folgend ist die *log4j.properties* des Projekts angeführt.

Hier sind die Logging Konfigurationen für *log4j* enthalten.

Listing 2: log4j.properties

```

1 log4j.rootLogger=INFO,STDOUT
2
3 log4j.appender.STDOUT=org.apache.log4j.ConsoleAppender
4 log4j.appender.STDOUT.layout=org.apache.log4j.PatternLayout
5 log4j.appender.STDOUT.layout.ConversionPattern=%p\t%d{HH:mm:ss,SSS}\t- %m%n

```

## Übung 3

## 1.4 JUnit Testsuite Source

Folgend sind die Sources der JUnit Testsuite angeführt.

### 1.4.1 AbstractLoggerWatcher.java

Diese Klasse dient als Basis für die LoggerWatcher und kapselt alle gemeinsamen Konfigurationen und Ressourcen.

Listing 3: AbstractLoggerWatcher.java

```
1 package at.fh.ooe.swe4.junit.test.suite.watcher.api;
2
3 import org.apache.log4j.Level;
4 import org.apache.log4j.Logger;
5 import org.junit.rules.TestWatcher;
6
7 /**
8  * This is the base class for the loggin watchers.
9  *
10  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
11  * @date May 2, 2015
12  */
13 public abstract class AbstractLoggerWatcher extends TestWatcher {
14
15     protected Logger log;
16     protected final Level level;
17     protected static final int SEPARATOR_REPEATIONS = 100;
18     protected static final String LOG_FORMAT = "%1$-12s";
19
20     public AbstractLoggerWatcher(Level level) {
21         super();
22         this.level = level;
23     }
24
25 }
```

## Übung 3

### 1.4.2 LoggingTestClassWatcher.java

Diese Klasse logt die Invocation einer Testklasse.

Listing 4: LoggingTestClassWatcher.java

```

1 package at.fh.ooe.swe4.junit.test.suite.watcher;
2
3 import org.apache.commons.lang.StringUtils;
4 import org.apache.log4j.Level;
5 import org.apache.log4j.Logger;
6 import org.junit.runner.Description;
7 import org.junit.runners.model.Statement;
8
9 import at.fh.ooe.swe4.junit.test.suite.watcher.api.AbstractLoggerWatcher;
10
11 /**
12  * This class is the test method invocation logger watcher.
13  *
14  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
15  * @date Apr 26, 2015
16  */
17 public class LoggingTestClassWatcher extends AbstractLoggerWatcher {
18
19     private String className;
20
21     public LoggingTestClassWatcher(final Level level) {
22         super(level);
23     }
24
25     @Override
26     public Statement apply(Statement base, Description description) {
27         log = Logger.getLogger(description.getTestClass());
28         return super.apply(base, description);
29     }
30
31     @Override
32     protected void starting(Description description) {
33         className = description.getTestClass()
34             .getName();
35         log.log(level, StringUtils.repeat("#", SEPARATOR_REPEATS));
36         log.log(level, new StringBuilder(String.format(LOG_FORMAT, "started:")).append(className)
37             .toString());
38         log.log(level, StringUtils.repeat("#", SEPARATOR_REPEATS));
39     }
40
41     @Override
42     protected void succeeded(Description description) {
43         log.log(level, StringUtils.repeat("-", SEPARATOR_REPEATS));
44         log.log(level, StringUtils.repeat("#", SEPARATOR_REPEATS));
45         log.log(level, new StringBuilder(String.format(LOG_FORMAT, "succeeded:")).append(className)
46             .toString());
47         log.log(level, StringUtils.repeat("#", SEPARATOR_REPEATS));
48     };
49
50     @Override
51     protected void failed(Throwable e, Description description) {
52         log.log(level, StringUtils.repeat("-", SEPARATOR_REPEATS));
53         log.log(level, StringUtils.repeat("#", SEPARATOR_REPEATS));
54         log.log(level, new StringBuilder(String.format(LOG_FORMAT, "failed:")).append(className)
55             .toString(), e);
56         log.log(level, StringUtils.repeat("#", SEPARATOR_REPEATS));
57     };

```

## Übung 3

---

58 | }

---

## Übung 3

### 1.4.3 LoggingTestInvocationWatcher.java

Diese Klasse logt die Invocation einer Testmethode.

Listing 5: LoggingTestInvocationWatcher.java

```

1 package at.fh.ooe.swe4.junit.test.suite.watcher;
2
3 import org.apache.commons.lang.StringUtils;
4 import org.apache.log4j.Level;
5 import org.apache.log4j.Logger;
6 import org.junit.runner.Description;
7 import org.junit.runners.model.Statement;
8
9 import at.fh.ooe.swe4.junit.test.suite.watcher.api.AbstractLoggerWatcher;
10
11 /**
12  * This class is the test method invocation logger watcher.
13  *
14  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
15  * @date Apr 26, 2015
16  */
17 public class LoggingTestInvocationWatcher extends AbstractLoggerWatcher {
18
19     public LoggingTestInvocationWatcher(final Level level) {
20         super(level);
21     }
22
23     @Override
24     public Statement apply(Statement base, Description description) {
25         log = Logger.getLogger(description.getTestClass());
26         return super.apply(base, description);
27     }
28
29     @Override
30     protected void starting(Description description) {
31         log.log(level, StringUtils.repeat("-", SEPARATOR_REPEATIONS));
32         log.log(level, String.format(LOG_FORMAT, "started:") + description.getMethodName());
33     }
34
35     @Override
36     protected void succeeded(Description description) {
37         log.log(level, String.format(LOG_FORMAT, "succeeded:") + description.getMethodName());
38     };
39
40     @Override
41     protected void failed(Throwable e, Description description) {
42         log.log(level, String.format(LOG_FORMAT, "failed:") + description.getMethodName(), e);
43     };
44
45 }

```

## Übung 3

### 1.5 N-Puzzle Source

Folgend sind die Sources des n-puzzle-problem Projekts angeführt.

#### 1.5.1 BoardListImpl.java

Folgend ist das Interface *Board<T>* *extends Comparable<T>* angeführt, welches die Spezifikation für die Board Implementierung darstellt. Es wurde beschlossen ein eigenes Interface zu spezifizieren, da es auch möglich sein soll den Container für die Verwaltung der Kacheln auszutauschen (*List<T>*, *T[]*, usw.) Des weiteren wurde beschlossen eine Typdefinition zu verlangen, da es auch möglich sein soll andere Datentypen außer Integer für die Kacheln zu verwenden. Die einzige Voraussetzung dafür ist, dass sich der Datentyp einer Rangordnung unterwerfen lässt, was über das Interface *Comparable<T>* gewährleistet wird. Es spielt hierbei keine Rolle um welchen Datentyp es sich handelt *Integer*, *BigInteger*, *Decimal*, *Character*, usw., das einzig wichtige ist das der Datentyp das Interface *Comparable<T>* korrekt implementiert.

Listing 6: BoardListImpl.java

```

1 package at.fh.ooe.swe4.puzzle.api;
2
3 import java.util.NoSuchElementException;
4 import java.util.function.IntConsumer;
5 import java.util.stream.IntStream;
6
7 import at.fh.ooe.swe4.puzzle.exception.InvalidBoardIndexException;
8 import at.fh.ooe.swe4.puzzle.exception.InvalidMoveException;
9 import at.fh.ooe.swe4.puzzle.model.Position;
10
11 /**
12  *
13  * This is the interface specification for the puzzle board.<br>
14  * The indices on a board start with 1 and not with 0.<br>
15  * E.g.:<br>
16  * A board with the dimension 4X4 are build as follows.
17  * <ul>
18  * <li><b>upper left corner:</b> (1,1)</li>
19  * <li><b>lower right corner:</b> (4,4)</li>
20  * </ul>
21  *
22  *
23  * @author Thomas Herzog <thomas.hertzog@students.fh-hagenberg.at>
24  * @date Apr 26, 2015
25  * @param <T>
26  *         the value type of the values on the board
27  */
28 public interface Board<T> extends Comparable<T> extends Comparable<Board<T>>, Cloneable {
29
30     /**
31      * Specifies the directions the empty tile can be moved.
32      *
33      * @author Thomas Herzog <thomas.hertzog@students.fh-hagenberg.at>
34      * @date Apr 26, 2015
35      */
36     public static enum Move {
37         UP, DOWN, LEFT, RIGHT;
38     }
39
40     /**
41      * Returns the tile at the position (rowIdx, colIdx).
42      *
43      * @param rowIdx

```

## Übung 3

```

44     *           the row index
45     * @param colIdx
46     *           the column index.
47     * @return the found tile
48     * @throws InvalidBoardIndexException
49     *           if the indices are invalid
50     */
51     public T getTile(int rowIdx, int colIdx);
52
53     /**
54     * Sets an tile on the given position (rowIdx, colIdx).
55     *
56     * @param rowIdx
57     *           the row index
58     * @param colIdx
59     *           the column index
60     * @param value
61     *           the value to be set on the defined position
62     * @throws InvalidBoardIndexException
63     *           if the indices are invalid
64     */
65     public void setTile(int rowIdx, int colIdx, T value);
66
67     /**
68     * Sets an empty tile on the given position (rowIdx, colIdx).
69     *
70     * @param rowIdx
71     *           the row index
72     * @param colIdx
73     *           the column index
74     * @throws InvalidBoardIndexException
75     *           if the indices are invalid
76     */
77     public void setEmptyTile(int rowIdx, int colIdx);
78
79     /**
80     * Gets position of the empty tile (rowIdx, colIdx).<br>
81     * If multiple empty tiles are present on this board then this method will
82     * return the first occurrence of an empty tile.<br>
83     * If no empty tile is present the returned position instance will contain
84     * invalid indices.
85     *
86     * @return the position model with the indices, it will have the indices set
87     *         to (-1, -1) if no empty tile could not be found
88     */
89     public Position getEmptyTilePosition();
90
91     /**
92     * Gets the Position of the tile with the given value.
93     *
94     * @param value
95     *           the value to be searched on this board.
96     * @return the found tile position
97     * @throws NoSuchElementException
98     *           if the value could not be found on the board
99     */
100    public Position getTilePosition(final T value);
101
102    /**
103    * Gets the size of the board N where the board will have the dimensions N x
104    * N
105    *
106    * @return the size of this board

```

# Übung 3

```

107     */
108     public int size();
109
110     /**
111      * Answers the question if this board is a valid board.<br>
112      * A board is invalid if one of the following conditions fit
113      * <ul>
114      * <li>no empty tile present</li>
115      * <li>tiles with duplicates values are present</li>
116      * </ul>
117      *
118      * @return true if this board is valid, false otherwise
119      */
120     public boolean isValid();
121
122     /**
123      * Shuffles the board tiles by performing random moves on the board.<br>
124      * It is ensured that the parity is kept (odd/even parity will be kept).
125      */
126     public void shuffle();
127
128     /**
129      * Moves the empty tile to the given position by switching the value on the
130      * given position with the empty tile position.<br>
131      * This method does not ensure that the move is a valid one, which means it
132      * is not ensured that the tile is only moved one position in any direction.
133      *
134      * @param rowIdx
135      *         the row index
136      * @param colIdx
137      *         the column index
138      * @throws InvalidBoardIndexException
139      *         if either the indices or the board is invalid
140      */
141     public void move(int rowIdx, int colIdx);
142
143     /**
144      * Moves the empty tile on step left.
145      *
146      * @throws InvalidMoveException
147      *         if the empty tile is already placed on the outer left column
148      */
149     public void moveLeft();
150
151     /**
152      * Moves the empty tile on step left.
153      *
154      * @throws InvalidMoveException
155      *         if the empty tile is already placed on the outer right column
156      */
157     public void moveRight();
158
159     /**
160      * Moves the empty tile on step right.
161      *
162      * @throws InvalidMoveException
163      *         if the empty tile is already placed on the top row
164      */
165     public void moveUp();
166
167     /**
168      * Moves the empty tile on step down.
169      *

```



## Übung 3

```

170     * @throws InvalidMoveException
171     *         if the empty tile is already placed on the bottom row
172     */
173     public void moveDown();
174
175     /**
176     * Performs all of the moves of the empty tile defined by the given Iterable
177     * instance.
178     *
179     * @param moves
180     *         the Iterable instance holding the move positions for the empty
181     *         tile
182     * @throws InvalidMoveException
183     *         if the empty tile is tried to be moved out of the board
184     */
185     public void makeMoves(Iterable<Move> moves);
186
187     /**
188     * Calculates the parity of this board.<br>
189     * The parity is build as follows: <br>
190     * <sum_of_ordered_pairs> + <row_idx_empty_tile>
191     *
192     * @return the parity of this board
193     * @throws IllegalStateException
194     *         if this board is invalid
195     */
196     public int calculateParity();
197
198     // Force overwrite of clone
199     /**
200     * Performs an deep copy of the current instance.
201     *
202     * @return the copied instance
203     */
204     public Board<T> clone();
205 }

```

## Übung 3

### 1.5.2 BoardImpl.java

Folgend ist die Implementierung für das Interface Board<T extends Comparable<T>> angeführt, welches als Container eine ArrayList<T> verwendet. Dieser Container wurde gewählt da es sich im Hintergrund um ein Object[] handelt, daher indexierte Zugriffe möglich sind und dieser Container Methoden zur Verfügung stellt, die den Zugriff bzw. die Handhabung dieses Object[] erleichtern. Der Container wird mit der zu erwartenden Größe initialisiert um ein dynamisches wachsen, welches ein Array.copy(...) verursacht, zu vermeiden.

Dies ist zwar nicht zwingend notwendig aber ist als Best Practise anzusehen.

Listing 7: Board.java

```

1 package at.fh.ooe.swe4.puzzle.impl;
2
3 import java.util.ArrayList;
4 import java.util.Arrays;
5 import java.util.Collections;
6 import java.util.Comparator;
7 import java.util.List;
8 import java.util.NoSuchElementException;
9 import java.util.function.Consumer;
10 import java.util.function.Function;
11 import java.util.stream.Collectors;
12 import java.util.stream.IntStream;
13
14 import org.apache.commons.lang.StringUtils;
15
16 import at.fh.ooe.swe4.puzzle.api.Board;
17 import at.fh.ooe.swe4.puzzle.exception.InvalidBoardIndexException;
18 import at.fh.ooe.swe4.puzzle.exception.InvalidMoveException;
19 import at.fh.ooe.swe4.puzzle.model.Position;
20
21 /**
22  * This is the implementation of the {@link Board} interface.<br>
23  * This class provides possibility to define the tile value type which needs to
24  * implement {@link Comparable} interface.
25  *
26  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
27  * @date May 2, 2015
28  * @param <T>
29  *         the value type of the tile values
30  */
31 public class BoardListImpl<T extends Comparable<T>> implements Board<T> {
32
33     private final int size;
34     private final List<T> container;
35
36     /**
37      * The default constructor for this class which needs an size to be given.
38      *
39      * @param size
40      *         the size of the board, for the rows and the columns
41      * @throws IllegalArgumentException
42      *         if size <=0
43      */
44     public BoardListImpl(final int size) {
45         super();
46         if (size <= 0) {
47             throw new IllegalArgumentException("Size must be greater than 0");
48         }
49         this.size = size;
50         final int containerSize = (int) Math.pow(size, 2);

```

# Übung 3

```

51     container = new ArrayList<T>(containerSize);
52     // Init with null values
53     for (int i = 0; i < containerSize; i++) {
54         container.add(null);
55     }
56 }
57
58 /**
59  * This constructor provides the size and container which contains the board
60  * values, where the container will be copied and cannot be modified from
61  * the outside.<br>
62  * The container must be a flat representation of the board.<br>
63  * It's size must be size*size which is the same as rows*columns.<br>
64  * E.g.: A value positioned at (3,2) will have the container index
65  * (3-1)*size + (2-1)
66  *
67  * @param size
68  *         the size of the board
69  * @param container
70  *         the container holding the board values
71  * @throws IllegalArgumentException
72  *         <ul>
73  *         <li>size <= 0</li>
74  *         <li>container is null</li>
75  *         <li>container.size() != (size*size)</li>
76  *         </ul>
77  */
78 public BoardListImpl(final int size, final List<T> container) {
79     super();
80     if (size <= 0) {
81         throw new IllegalArgumentException("Size must be greater than 0");
82     }
83     if (container == null) {
84         throw new IllegalArgumentException("Container must not be null");
85     }
86     if (((int) Math.pow(size, 2)) != container.size()) {
87         throw new IllegalArgumentException("Container size does not correspond to given board
88         ↪ size");
89     }
90     this.size = size;
91     this.container = new ArrayList<T>(container);
92 }
93
94 @Override
95 public T getTile(int rowIdx, int colIdx) {
96     return container.get(calculateContainerIdx(rowIdx, colIdx));
97 }
98
99 @Override
100 public void setTile(int rowIdx, int colIdx, T value) {
101     container.set(calculateContainerIdx(rowIdx, colIdx), value);
102 }
103
104 @Override
105 public void setEmptyTile(int rowIdx, int colIdx) {
106     setTile(rowIdx, colIdx, (T) null);
107 }
108
109 @Override
110 public Position getEmptyTilePosition() {
111     final int[] indices = IntStream.range(0, container.size())
112         .filter(i -> container.get(i) == null)
113         .toArray();

```

## Übung 3

```

113     if (indices.length != 0) {
114         final int rowIdx = ((indices[0] / size) + 1);
115         final int colIdx = (indices[0] - ((rowIdx - 1) * size) + 1);
116         return new Position(rowIdx, colIdx);
117     }
118     return new Position(-1, -1);
119 }
120
121 @Override
122 public Position getTilePosition(final T value) {
123     // row iteration
124     for (int i = 1; i <= size(); i++) {
125         // column iteration
126         for (int j = 1; j <= size(); j++) {
127             final T tile = getTile(i, j);
128             if (((tile == null) && (value == null)) || ((tile != null) && (tile.equals(value)))) {
129                 return new Position(i, j);
130             }
131         }
132     }
133     throw new NoSuchElementException("Tile with value '" + ((value == null) ? "null" :
134         ↪ value.toString()) + "' does not exist on the board");
135 }
136
137 @Override
138 public int size() {
139     return size;
140 }
141
142 @Override
143 public boolean isValid() {
144     return (container.stream()
145         .distinct()
146         .count() == (container.size())) && (container.stream()
147             .filter(element -> element == null)
148             .count() == 1);
149 }
150
151 @Override
152 public void shuffle() {
153     // We need to ensure the parity of the shuffled board
154     final boolean even = (calculateParity() % 2 == 0);
155     boolean shuffledEven = !even;
156
157     // fill list with the possible moves
158     final List<Move> moves = new ArrayList<Move>(Move.values().length * 4);
159     moves.addAll(Arrays.asList(Move.values()));
160     moves.addAll(Arrays.asList(Move.values()));
161     moves.addAll(Arrays.asList(Move.values()));
162     moves.addAll(Arrays.asList(Move.values()));
163
164     // as long as parity is not equal to original one
165     while (even != shuffledEven) {
166         // shuffle the possible moves
167         Collections.shuffle(moves);
168         moves.forEach(new Consumer<Move>() {
169             @Override
170             public void accept(Move direction) {
171                 try {
172                     // try to perform move
173                     move(direction);
174                 } catch (InvalidMoveException e) {
175                     // expected because random movements could try to move

```

## Übung 3

```

175         // empty tile out of the board
176     }
177 }
178 });
179 // recalculate the parity of the shuffled board
180 shuffledEven = (calculateParity() % 2 == 0);
181 }
182 }
183
184 @Override
185 public void move(int rowIdx, int colIdx) {
186     final Position position = getEmptyTilePosition();
187     setTile(position.rowIdx, position.colIdx, container.get(calculateContainerIdx(rowIdx,
188         ↪ colIdx)));
189     setTile(rowIdx, colIdx, null);
190 }
191
192 @Override
193 public void moveLeft() {
194     move(Move.LEFT);
195 }
196
197 @Override
198 public void moveRight() {
199     move(Move.RIGHT);
200 }
201
202 @Override
203 public void moveUp() {
204     move(Move.UP);
205 }
206
207 @Override
208 public void moveDown() {
209     move(Move.DOWN);
210 }
211
212 @Override
213 public void makeMoves(Iterable<Move> moves) {
214     if (moves == null) {
215         throw new InvalidMoveException("Cannot perform moves because iterable instance is null");
216     }
217     moves.iterator()
218         .forEachRemaining(new Consumer<Move>() {
219             @Override
220             public void accept(Move t) {
221                 move(t);
222             }
223         });
224 }
225
226 // Private helper
227 /**
228  * Check if the given indices are valid for this board.
229  *
230  * @param rowIdx
231  *         the row index
232  * @param colIdx
233  *         the column index
234  * @throws InvalidBoardIndexException
235  *         if at least one of the indices is invalid
236  */
237 private void checkForValidIndex(final int rowIdx, final int colIdx) {

```

## Übung 3

```

237     if ((rowIdx > size) || (rowIdx <= 0) || (colIdx > size) || (colIdx <= 0)) {
238         throw new InvalidBoardIndexException("One of the indicies over or underflows the border
        ↳ defined by size. rowIdx: " + rowIdx + " | colIdx: "
239             + colIdx);
240     }
241 }
242
243 /**
244  * Calculates the index for the backed container.
245  *
246  * @param rowIdx
247  *         the row index
248  * @param colIdx
249  *         the column index
250  * @return the calculated index
251  * @throws InvalidBoardIndexException
252  *         if at least one of the indexes is invalid
253  * @see BoardListImpl#checkForValidIndex(int, int)
254  */
255 private int calculateContainerIdx(final int rowIdx, final int colIdx) {
256     checkForValidIndex(rowIdx, colIdx);
257     return ((rowIdx - 1) * size) + (colIdx - 1);
258 }
259
260 /**
261  * Performs a move operation into the intended direction.<br>
262  * The move operation can be performed when the following conditions fit.
263  * <ul>
264  * <li>the board is valid</li>
265  * <li>the new position is valid</li>
266  * <li>the given {@link Move} enumeration is supported</li>
267  * </ul>
268  *
269  * @param direction
270  *         the direction to move to
271  * @throws InvalidMoveException
272  *         if the element cannot be moved to the intended direction
273  * @see BoardListImpl#calculateContainerIdx(int, int)
274  * @see BoardListImpl#getEmptyTilePosition()
275  */
276 private void move(final Move direction) {
277     if (direction == null) {
278         throw new InvalidMoveException("Cannot perform move operation with null direction");
279     }
280
281     int rowIdxDif = 0;
282     int colIdxDif = 0;
283     switch (direction) {
284     case UP:
285         rowIdxDif = -1;
286         break;
287     case DOWN:
288         rowIdxDif = 1;
289         break;
290     case LEFT:
291         colIdxDif = -1;
292         break;
293     case RIGHT:
294         colIdxDif = 1;
295         break;
296     default:
297         throw new IllegalArgumentException("Direction enum '" + direction.name() + "' is not
        ↳ handled");

```

## Übung 3

```

298     }
299
300     try {
301         final Position oldPosition = getEmptyTilePosition();
302         checkForValidIndex(oldPosition.rowIdx, oldPosition.colIdx);
303         move((oldPosition.rowIdx + rowIdxDif), (oldPosition.colIdx + colIdxDif));
304     } catch (InvalidBoardIndexException e) {
305         throw new InvalidMoveException("Cannot move to the intended direction. direction: " +
            ↪ direction.name(), e);
306     }
307 }
308
309 /**
310  * Compares this instance to another by comparing the size of the board.
311  */
312 public int compareTo(Board<T> o) {
313     return Integer.valueOf(size)
314         .compareTo(o.size());
315 }
316
317 @Override
318 public int calculateParity() {
319     final Position position = getEmptyTilePosition();
320     final int emptyTileIdx = calculateContainerIdx(position.rowIdx, position.colIdx);
321     int parity = position.rowIdx;
322     for (int i = 0; i < container.size(); i++) {
323         // ignore empty tile
324         if (i != emptyTileIdx) {
325             for (int j = 0; j < i; j++) {
326                 // ignore empty tile
327                 if (j != emptyTileIdx) {
328                     parity += (container.get(j)
329                         .compareTo(container.get(i)) > 0) ? 1 : 0;
330                 }
331             }
332         }
333     }
334     return parity;
335 }
336
337 @Override
338 public Board<T> clone() {
339     return new BoardListImpl<T>(size, container);
340 }
341
342 @Override
343 public int hashCode() {
344     final int prime = 31;
345     int result = 1;
346     result = prime * result + ((container == null) ? 0 : container.hashCode());
347     return result;
348 }
349
350 @Override
351 public boolean equals(Object obj) {
352     if (this == obj)
353         return true;
354     if (obj == null)
355         return false;
356     if (getClass() != obj.getClass())
357         return false;
358     BoardListImpl<T> other = (BoardListImpl<T>) obj;
359     if (container == null) {

```

## Übung 3

```

360         if (other.container != null)
361             return false;
362     } else if (!container.equals(other.container))
363         return false;
364     return true;
365 }
366
367 @Override
368 public String toString() {
369     final List<T> copy = new ArrayList<T>(container);
370     final int length = copy.stream()
371         .max(Comparator.comparing(item -> ((item != null) ? item.toString()
372             .length() : 0)))
373         .get()
374         .toString()
375         .length();
376     final StringBuilder sb = new StringBuilder(container.size() * length * 2);
377     sb.append(System.lineSeparator());
378     for (int i = 0; i < size(); i++) {
379         final List<T> result = copy.stream()
380             .limit(size())
381             .collect(Collectors.toList());
382         sb.append(result.stream()
383             .map(new Function<T, String>() {
384                 public String apply(T t) {
385                     final String value;
386                     int l = ((length > 1) && (length % 2 != 0)) ? (length + 1) : length;
387                     if (t != null) {
388                         value = t.toString();
389                     } else {
390                         value = " ";
391                     }
392                     return new StringBuilder("[").append(StringUtils.center(value, l))
393                         .append("]")
394                         .toString();
395                 }
396             })
397             .collect(Collectors.joining(" ")))
398         .append(System.lineSeparator());
399         copy.removeAll(result);
400     }
401     return sb.toString();
402 }
403 }

```



## Übung 3

### 1.5.3 SlidingPuzzle.java

Folgend ist die Implementierung des `SlidingPuzzle<T extends Comparable<T>>` angeführt, welches die Boards versucht in die Zielkonfiguration zu überführen. Da es möglich sein soll nicht nur eine Zielkonfiguration zu handhaben wurde die Implementierung erweitert um die Möglichkeit eine Zielkonfiguration zu übergeben.

Diese Klasse wurde mit dem Pattern *Fluent Interface* implementiert, da es dieser Pattern erlaubt hier eine gut verständliche Aufrufkette aufzubauen. Diese Aufrufkette liest sich wie Prosa wie in diesem Beispiel ersichtlich.

```
// Create new slide instance
SlidingPuzzle<Integer> slider = new SlidingPuzzle<>();
// start the slider with the initial board
slider.start(initialBoard)
    // Try to solve the board (returns SolutionHandler instance)
    .solve(goalBoard)
    // Starts the SolutionHandler instance
    .start()
    // Register custom logger with custom level
    .registerLogger(myLog, Level.DEBUG)
    // Fill resulting moves in given list
    .fillMoves(moves)
    // prints the resulting moves via the logger
    .printMoves()
    // performs the moves on the initial board and logs initial/resulting board state
    .performMoves()
    // ends the SolutionHandler, returns related SlidingPuzzle instance
    .end()
    // Ends the SlidingPuzzle instance (reset state)
    .end();
```

Es wurde eine interne Klasse `SolutionHandler<T extends Comparable<T>>` implementiert, welche das Handling der Lösung übernimmt. Auch diese Implementierung verwendet *Fluent Interface*.

Um durchgehend mit demselben Typ zu arbeiten zu können wird auch hier die Typdefinition verlangt, die der Typdefinition der Kacheln im `Board<T extends Comparable<T>>` entspricht. Damit ist gewährleistet, dass z.B.: eine `SlidingPuzzle<Integer>` Instanz nur auf `Board<Integer>` Instanzen angewendet werden kann und nicht z.B.: auf `Board<Character>` Instanzen.

Listing 8: BoardListImpl.java

```
1 package at.fh.ooe.swe4.puzzle.impl;
2
3 import java.util.ArrayList;
4 import java.util.HashMap;
5 import java.util.HashSet;
6 import java.util.Iterator;
7 import java.util.List;
8 import java.util.Map;
9 import java.util.PriorityQueue;
10 import java.util.Queue;
11 import java.util.Set;
12
13 import org.apache.commons.lang.StringUtils;
14 import org.apache.log4j.Level;
```

## Übung 3

```

15 import org.apache.log4j.Logger;
16
17 import at.fh.ooe.swe4.puzzle.api.Board;
18 import at.fh.ooe.swe4.puzzle.api.Board.Move;
19 import at.fh.ooe.swe4.puzzle.exception.InvalidMoveException;
20 import at.fh.ooe.swe4.puzzle.exception.NoSolutionException;
21 import at.fh.ooe.swe4.puzzle.model.SearchNode;
22
23 /**
24  * This class is the solver for the game.
25  *
26  * @author Thomas Herzog <thomas.hertzog@students.fh-hagenberg.at>
27  * @date Apr 26, 2015
28  * @param <T>
29  *         The value type of the board tiles
30  */
31 public class SlidingPuzzle<T> extends Comparable<T> {
32
33     private Board<T> board;
34     private Board<T> goal;
35     private Queue<SearchNode<T>> queue;
36     private Map<Board<T>, SearchNode<T>> open;
37     private Map<Board<T>, SearchNode<T>> closed;
38
39     private boolean started = Boolean.FALSE;
40
41     /**
42      * This is the solution handler which handles the solution result
43      *
44      * @author Thomas Herzog <thomas.hertzog@students.fh-hagenberg.at>
45      * @date May 1, 2015
46      * @param <T>
47      *         The value type of the board tiles
48      */
49     public static final class SolutionHandler<T> extends Comparable<T> {
50         private final SearchNode<T> node;
51         private final SlidingPuzzle<T> slider;
52         private SearchNode<T> start;
53         private List<Move> moves;
54         private Logger log;
55         private Level level;
56
57         private boolean started = Boolean.FALSE;
58
59         /**
60          * Constructor which gets the current {@link SlidingPuzzle} instance and
61          * resulting {@link SearchNode} instance, where no solution has been
62          * found if the node is null.
63          *
64          * @param node
65          *         the {@link SearchNode} instance representing the result.
66          *         If null no result has been found.
67          * @param slider
68          *         the {@link SlidingPuzzle} instance which gets returned on
69          *         method call {@link SolutionHandler#end()}
70          */
71         public SolutionHandler(SearchNode<T> node, SlidingPuzzle<T> slider) {
72             super();
73             this.node = node;
74             this.slider = slider;
75         }
76
77         /**

```

## Übung 3

```

78      * Starts the solution handling by creation the {@link Move} list.
79      *
80      * @return the current instance
81      * @throws IllegalStateException
82      *         if the solution handler is already started
83      * @throws NoSolutionException
84      *         if the set node is null
85      */
86      public SolutionHandler<T> start() throws NoSolutionException {
87          if (started) {
88              throw new IllegalStateException("The solution handler needs to be end before started
89                  ↪ again");
90          }
91          started = Boolean.TRUE;
92          if (node == null) {
93              throw new NoSolutionException("No solution found");
94          }
95          final Iterator<SearchNode<T>> it = node.iterator();
96          while (it.hasNext()) {
97              start = it.next();
98          }
99          moves = node.toMoves();
100         // init with default logging
101         this.log = Logger.getLogger(this.getClass());
102         this.level = Level.INFO;
103         return this;
104     }
105
106     /**
107      * Ability to provide a custom {@link Logger} instance and {@link Level}
108      *
109      * @param log
110      *         the {@link Logger} instance used for logging
111      * @param level
112      *         the {@link Level} instance defining the to use log level
113      * @return the current instance
114      * @throws IllegalStateException
115      *         if the solution handler hasn't been started yet
116      */
117     public SolutionHandler<T> registerLogger(final Logger log, final Level level) {
118         checkForStarted();
119         this.log = log;
120         this.level = level;
121         return this;
122     }
123
124     /**
125      * Prints the resulting moves via the {@link Logger}
126      *
127      * @return the current instance
128      * @throws IllegalStateException
129      *         if the solution handler hasn't been started yet
130      */
131     public SolutionHandler<T> printMoves() {
132         checkForStarted();
133         log.log(level, "Resulting moves:");
134         for (int i = 0; i < moves.size(); i++) {
135             log.log(level, new StringBuilder().append(String.format("%1$-" + ((moves.size() / 10) + 1)
136                 ↪ + "s", (i + 1)))
137                 .append(": ")
138                 .append(moves.get(i)
139                     .name())
140                 .toString());
141         }
142     }

```

# Übung 3

```

139     }
140     return this;
141 }
142
143 /**
144  * Performs the moves on the initial board which has been found via the
145  * given node.
146  *
147  * @return the current instance
148  * @throws IllegalStateException
149  *         if the solution handler hasn't been started yet
150  * @throws InvalidMoveException
151  *         if at least one of the resulting moves is invalid
152  * @see Board#makeMoves(Iterable)
153  */
154 public SolutionHandler<T> performMoves() {
155     checkForStarted();
156     final Board<T> board = start.getBoard()
157         .clone();
158     start.getBoard()
159         .makeMoves(moves);
160     log.log(level, "Initial state:");
161     // .append(board.toString())
162     String[] boardStrings = null;
163     boardStrings = StringUtils.split(board.toString(), System.lineSeparator());
164     for (String string : boardStrings) {
165         log.log(level, string);
166     }
167     log.log(level, "Initial state after movements:");
168     boardStrings = StringUtils.split(start.getBoard()
169         .toString(), System.lineSeparator());
170     for (String string : boardStrings) {
171         log.log(level, string);
172     }
173     return this;
174 }
175
176 @Deprecated
177 public List<Move> getMoves() {
178     return node.toMoves();
179 }
180
181 /**
182  * Fills the given list with the resulting moves.
183  *
184  * @param moves
185  *         the list to fill resulting moves in
186  * @return the current instance
187  * @throws IllegalArgumentException
188  *         if the given list is null
189  * @throws IllegalStateException
190  *         if the solution handler hasn't been started yet
191  */
192 public SolutionHandler<T> fillMoves(final List<Move> moves) {
193     checkForStarted();
194     if (moves == null) {
195         throw new IllegalArgumentException("Given moves list must not be null");
196     }
197     moves.addAll(this.moves);
198     return this;
199 }
200
201 /**

```

## Übung 3

```

202     * Ends the solution handling by resetting all members but the given node
203     * or slider.<br>
204     * This instance can be restarted by calling method
205     * {@link SolutionHandler#start}
206     *
207     * @return the related {@link SlidingPuzzle} instance
208     * @throws IllegalStateException
209     *         if the solution handler hasn't been started yet
210     */
211     public SlidingPuzzle<T> end() {
212         checkForStarted();
213         started = Boolean.FALSE;
214         this.level = null;
215         this.log = null;
216         this.moves = null;
217         return slider;
218     }
219
220     /**
221     * Checks if the {@link SolutionHandler} instance has been started
222     *
223     * @throws IllegalStateException
224     *         if the instance hasn't been started but it is tried to
225     *         perform an action on it.
226     */
227     private void checkForStarted() {
228         if (!started) {
229             throw new IllegalStateException("The solution handler needs to be started before the
230                 ↪ solution can be handled");
231         }
232     }
233
234     /**
235     * Default constructor which does not initializes this instance.<br>
236     * The {@link Board} instance to work on needs to be set via
237     * {@link SlidingPuzzleImpl#init(Board)}
238     *
239     * @see SlidingPuzzleImpl#init(Board)
240     */
241     public SlidingPuzzle() {
242         super();
243     }
244
245     /**
246     * Initializes the solver with the given board.<br>
247     * Keep in mind if the board is null or invalid the
248     *
249     * @param initial
250     *         the board to be resolved
251     * @return the current instance
252     * @throws IllegalStateException
253     *         if the slider has already been started
254     */
255
256     public SlidingPuzzle<T> start(final Board<T> initial) {
257         if (started) {
258             throw new IllegalStateException("The slider instance need to be end before restarted
259                 ↪ again.");
260         }
261         if ((initial == null) || (!initial.isValid())) {
262             throw new IllegalArgumentException("Cannot init this instance with an null or invalid
263                 ↪ board.");

```

## Übung 3

```

262     }
263     this.started = Boolean.TRUE;
264     this.board = initial;
265     this.queue = new PriorityQueue<SearchNode<T>>();
266     this.open = new HashMap<Board<T>, SearchNode<T>>();
267     this.closed = new HashMap<Board<T>, SearchNode<T>>();
268     return this;
269 }
270
271 /**
272  * Resets this instance by setting all container to null.
273  *
274  * @throws IllegalStateException
275  *         if the solution handler hasn't been started yet
276  */
277 public SlidingPuzzle<T> end() {
278     checkForStarted();
279     this.started = Boolean.FALSE;
280     this.board = null;
281     this.goal = null;
282     this.queue = null;
283     this.closed = null;
284     return this;
285 }
286
287 /**
288  * Solves the set initial board and tries to calculate the moves to be made
289  * to transform the initial board to the goal board.
290  *
291  * @param goal
292  *        the goal board to resolve the initial board to
293  * @return the {@link SolutionHandler} instance which is responsible for
294  *         handling the result
295  * @throws NoSolutionException
296  *        


297  *        <li>goal is null</li>
298  *        <li>goal is invalid</li>
299  *        <li>goal has different size</li>
300  *        </ul>
301  */
302 public SolutionHandler<T> solve(final Board<T> goal) throws NoSolutionException {
303     checkForStarted();
304     // goal must represent valid board
305     if ((goal == null) || (!goal.isValid()) || ((goal.size() != board.size()))) {
306         throw new NoSolutionException("Cannot solve the board if the goal board is either null,
307             ↳ invalid or of differen size");
308     }
309     this.goal = goal;
310
311     // validate parity of the two boards
312     if (!isResolvable()) {
313         return new SolutionHandler<T>(null, this);
314     }
315
316     // we found the solution right away
317     if (board.equals(goal)) {
318         return new SolutionHandler<T>(new SearchNode<>(0, null, board, goal, null), this);
319     }
320
321     SearchNode<T> current = new SearchNode<>(0, null, board, goal, null);
322     queue.add(current);
323     open.put(current.getBoard(), current);
324     // search as long nodes are left and solution hasn't been found

```

## Übung 3

```

324 while (!queue.isEmpty()) {
325     // poll from queue
326     current = queue.poll();
327     // remove from open as well
328     open.remove(current.getBoard());
329     // get successors of current node by performing moves on it
330     final List<SearchNode<T>> successors = performMoves(current);
331     // handle found successors
332     for (SearchNode<T> successor : successors) {
333
334         // check if already on open
335         SearchNode<T> tmpNode = open.get(successor.getBoard());
336         // skip investigation if open node has lower costs
337         if ((tmpNode != null) && (tmpNode.compareTo(successor) <= 0)) {
338             continue;
339         }
340
341         // check if already on closed
342         tmpNode = closed.get(successor.getBoard());
343         // skip investigation if closed has lower costs
344         if ((tmpNode != null) && (tmpNode.compareTo(successor) <= 0)) {
345             continue;
346         }
347
348         // check if we found the solution
349         if (successor.getBoard()
350             .equals(goal)) {
351             return new SolutionHandler<T>(successor, this);
352         }
353
354         // add successors on open and queue
355         open.put(successor.getBoard(), successor);
356         queue.add(successor);
357     }
358
359     // remember investigated node
360     closed.put(current.getBoard(), current);
361 }
362 // no solution found should never occur
363 throw new IllegalStateException("Solution should have been found but wasn't. Maybe parity
    ↳ check failed");
364 }
365
366 // Private helper
367 /**
368  * Tries to perform all possible moves on the current node board
369  *
370  * @param parent
371  *         the parent which is the predecessor of the successor
372  * @return the list of found successors
373  */
374 private List<SearchNode<T>> performMoves(final SearchNode<T> parent) {
375     final List<SearchNode<T>> successors = new ArrayList<SearchNode<T>>(Move.values().length);
376     for (Move direction : Move.values()) {
377         // Get board to move empty tile on
378         final Board<T> tmp = parent.getBoard()
379             .clone();
380         try {
381             // perform the moves
382             switch (direction) {
383                 case UP:
384                     tmp.moveUp();
385                     break;

```

## Übung 3

```

386         case DOWN:
387             tmp.moveDown();
388             break;
389         case LEFT:
390             tmp.moveLeft();
391             break;
392         case RIGHT:
393             tmp.moveRight();
394             break;
395         default:
396             throw new UnsupportedOperationException("Direction with name '" + direction.name() + "'
397                 ↳ cannot not handled");
398     }
399     // Add found successor in case of valid move
400     successors.add(new SearchNode<T>(parent.getCostsFormStart() + 1, parent, tmp, goal,
401         ↳ direction));
402     } catch (InvalidMoveException e) {
403         // do nothing on invalid move
404     }
405     }
406     return successors;
407 }
408 /**
409  * Answers the question if the set board is possible to be resolved to the
410  * set goal board.<br>
411  * It is if the parity is either even or odd on both boards.
412  *
413  * @return true if the set board is possible to be resolved to the set goal
414  *         board.
415  */
416 private boolean isResolvable() {
417     final int sourceParity = board.calculateParity() % 2;
418     final int targetParity = goal.calculateParity() % 2;
419     return (((sourceParity != 0) && (targetParity != 0)) || ((sourceParity == 0) && (targetParity
420         ↳ == 0)));
421 }
422 /**
423  * Checks if the {@link SlidingPuzzle} instance has been started
424  *
425  * @throws IllegalStateException
426  *         if the instance hasn't been started but it is tried to
427  *         perform an action on it.
428  */
429 private void checkForStarted() {
430     if (!started) {
431         throw new IllegalStateException("The slider needs to be started before the solution can be
432             ↳ handled");
433     }
434 }

```



## Übung 3

### 1.5.4 Position.java

Folgend ist der Source des Models `Position` angeführt, welches dafür verwendet wird um die Position der leeren Kacheln zu liefern, da es mir zu wenig erschien nur mit der Spalte der leeren Kacheln zu arbeiten.

Listing 9: Position.java

```

1 package at.fh.ooe.swe4.puzzle.model;
2
3 /**
4  * Holds the position information of an tile on a board.
5  *
6  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
7  * @date Apr 26, 2015
8  */
9 public class Position {
10
11     public final int rowIdx;
12     public final int colIdx;
13
14     public Position(int row, int column) {
15         super();
16         this.rowIdx = row;
17         this.colIdx = column;
18     }
19
20     @Override
21     public int hashCode() {
22         final int prime = 31;
23         int result = 1;
24         result = prime * result + colIdx;
25         result = prime * result + rowIdx;
26         return result;
27     }
28
29     @Override
30     public boolean equals(Object obj) {
31         if (this == obj)
32             return true;
33         if (obj == null)
34             return false;
35         if (getClass() != obj.getClass())
36             return false;
37         Position other = (Position) obj;
38         if (colIdx != other.colIdx)
39             return false;
40         if (rowIdx != other.rowIdx)
41             return false;
42         return true;
43     }
44
45     @Override
46     public String toString() {
47         return new StringBuilder("rowIdx: ").append(rowIdx)
48             .append(" | ")
49             .append("colIdx: ")
50             .append(colIdx)
51             .toString();
52     }
53 }

```

## Übung 3

### 1.5.5 SearchNode.java

Folgend ist der Source des Models `SearchNode<T extends Comparable<T>>` angeführt, welches den Knoten für den A\*-Algorithmus darstellt. Hierbei sei die Implementierung der Methoden `node.hashCode()`; `node.equals(other)` erwähnt, die nicht nur das referenzierte `Board<T>` verwendet sondern auch die Gesamtkosten.

Das Problem mit diesen Implementierungen ist, dass bei der Verwendung eines `Set<T>` Containers, welcher die geschlossenen Knoten enthält, Duplikate enthalten würde. Also Knoten mit demselben Board aber mit unterschiedlichen gesetzten Kosten.

Daher dürfen diese offenen und geschlossenen Knoten nicht in Containern verwendet werden, die deren `node.hashCode()`; `node.equals(Object other)` verwenden.

Listing 10: SearchNode.java

```

1  package at.fh.ooe.swe4.puzzle.model;
2
3  import java.util.ArrayList;
4  import java.util.Collections;
5  import java.util.Iterator;
6  import java.util.List;
7  import java.util.NoSuchElementException;
8  import java.util.function.BiFunction;
9  import java.util.function.Consumer;
10
11 import at.fh.ooe.swe4.puzzle.api.Board;
12 import at.fh.ooe.swe4.puzzle.api.Board.Move;
13
14 /**
15  * This class represents the search node for the solver algorithm.
16  *
17  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
18  * @date Apr 30, 2015
19  * @param <T>
20  *         the {@link Comparable<T>} type which represents the board value
21  */
22 public class SearchNode<T extends Comparable<T>> implements Comparable<SearchNode<T>>,
23     ↳ Iterable<SearchNode<T>> {
24
25     private int costsFormStart;
26     private int manhattanDistance;
27     private int fullCosts;
28     private SearchNode<T> predecessor;
29     private final Board<T> board;
30     private Move direction;
31
32     /**
33      * Labda expression for calculating
34      */
35     public static final BiFunction<Position, Position, Integer> CALC_MANHATTAN_DIST = (root, goal)
36     ↳ -> (Math.abs((root.rowIdx - goal.rowIdx)) + Math.abs((root.colIdx - goal.colIdx)));
37
38     /**
39      * Constructs this instance and calculates the Manhattan distance between
40      * given board an goal and also the full costs.
41      *
42      * @param costsFormStart
43      *         the level the node is in
44      * @param predecessor
45      *         the predecessor can be null
46      * @param board
47      *         the board this node represents
48      * @param goal

```

## Übung 3

```

47  *           the goal for calculating Manhattan distance
48  * @param direction
49  *           the direction performed to get to this state, can be null
50  * @throws IllegalArgumentException
51  *           if board or goal is null or they are not of the same size
52  */
53  public SearchNode(int costsFormStart, SearchNode<T> predecessor, Board<T> board, Board<T> goal,
    ↪ Move direction) {
54      super();
55      if ((board == null)) {
56          throw new IllegalArgumentException("Cannot configure this searchNode with a null board");
57      }
58      if ((goal == null)) {
59          throw new IllegalArgumentException("Cannot calculate manhattan distance on nul goal");
60      }
61      if (board.size() != goal.size()) {
62          throw new IllegalArgumentException("The given board and gola must be of the same size");
63      }
64      this.costsFormStart = costsFormStart;
65      this.predecessor = predecessor;
66      this.board = board;
67      this.direction = direction;
68      this.manhattanDistance = calculateMahanttenDistance(goal);
69      this.fullCosts = costsFormStart + manhattanDistance;
70  }
71
72  public int estimatedCostsToTarget() {
73      return manhattanDistance;
74  }
75
76  /**
77   * Calculates the Manhattan distance of all tiles on the given board to the
78   * goal board.
79   *
80   * @return the Manhattan distance of this board state compared to the goal
81   *         state.
82   * @throws NoSuchElementException
83   *         if the goal does not contain a value from the initial board
84   * @see SearchNode#CALC_MANHATTAN_DIST
85   * @see Board#getTile(int, int)
86   */
87  private int calculateMahanttenDistance(final Board<T> goal) {
88      int costs = 0;
89      for (int i = 1; i <= board.size(); i++) {
90          for (int j = 1; j <= board.size(); j++) {
91              final T tile = board.getTile(i, j);
92              // ignore empty tile
93              if (tile != null) {
94                  final Position goalPosition = goal.getTilePosition(tile);
95                  costs += CALC_MANHATTAN_DIST.apply(new Position(i, j), goalPosition);
96              }
97          }
98      }
99      return costs;
100  }
101
102  /**
103   * Gets the estimated total costs.
104   *
105   * @return the total cost of this node
106   */
107  public int estimatedTotalCosts() {
108      return getCostsFormStart() + estimatedCostsToTarget();

```

# Übung 3

```

109     }
110
111     /**
112      * Converts the node list started from this node to a list of {@link Move}
113      * which represent the moves to make to get this board state to the intended
114      * goal state.
115      *
116      * @return the list of moves to make to get to the goal state
117      */
118     public List<Move> toMoves() {
119         final List<Move> moves = new ArrayList<Move>();
120         iterator().forEachRemaining(new Consumer<SearchNode<T>>() {
121             public void accept(SearchNode<T> t) {
122                 // last node has no move set
123                 if (t.getMove() != null) {
124                     moves.add(t.getMove());
125                 }
126             }
127         });
128         // must be reversed because chain is in wrong order
129         Collections.reverse(moves);
130         return moves;
131     }
132
133     // Iterator methods
134     /**
135      * This is the Iterator implementation for iterating over the connected
136      * search node list of the given search node.
137      *
138      * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
139      * @date Apr 26, 2015
140      */
141     private static final class SearchNodeIterator<T extends Comparable<T>> implements
142         ↪ Iterator<SearchNode<T>> {
143
144         private SearchNode<T> node;
145
146         public SearchNodeIterator(SearchNode<T> node) {
147             super();
148             this.node = node;
149         }
150
151         public boolean hasNext() {
152             return node != null;
153         }
154
155         public SearchNode<T> next() {
156             if (!hasNext()) {
157                 throw new NoSuchElementException("No more search nodes are available");
158             }
159             final SearchNode<T> tmp = node;
160             node = node.getPredecessor();
161             return tmp;
162         }
163     }
164
165     // Getter and Setter
166     public Iterator<SearchNode<T>> iterator() {
167         return new SearchNodeIterator<T>(this);
168     };
169
170     public SearchNode<T> getPredecessor() {
171         return predecessor;

```

# Übung 3

```

171     }
172
173     public int getCostsFormStart() {
174         return costsFormStart;
175     }
176
177     public void setCostsFormStart(int costsFormStart) {
178         this.costsFormStart = costsFormStart;
179     }
180
181     public void setPredecessor(SearchNode<T> predecessor) {
182         this.predecessor = predecessor;
183     }
184
185     public Board<T> getBoard() {
186         return board;
187     }
188
189     public Move getMove() {
190         return direction;
191     }
192
193     public void setMove(Move direction) {
194         this.direction = direction;
195     }
196
197     /**
198      * Compares the two instance by their set full costs
199      */
200     public int compareTo(SearchNode<T> o) {
201         return Integer.valueOf(fullCosts)
202             .compareTo(o.fullCosts);
203     }
204
205     @Override
206     public int hashCode() {
207         final int prime = 31;
208         int result = 1;
209         result = prime * result + ((board == null) ? 0 : board.hashCode());
210         result = prime * result + fullCosts;
211         return result;
212     }
213
214     @Override
215     public boolean equals(Object obj) {
216         if (this == obj) {
217             return true;
218         }
219         if (obj == null) {
220             return false;
221         }
222         if (getClass() != obj.getClass()) {
223             return false;
224         }
225         SearchNode<T> other = (SearchNode<T>) obj;
226         if (board == null) {
227             if (other.board != null) {
228                 return false;
229             }
230         } else if (!board.equals(other.board)) {
231             return false;
232         }
233         if (fullCosts != other.fullCosts) {

```

## Übung 3

```

234     return false;
235 }
236     return true;
237 }
238
239 @Override
240 public String toString() {
241     final StringBuilder sb = new StringBuilder((int) (Math.pow(board.size(), 2) * 2));
242     sb.append(System.lineSeparator())
243         .append("-----")
244         .append(System.lineSeparator())
245         .append("SearchNode content")
246         .append(System.lineSeparator())
247         .append("-----")
248         .append(System.lineSeparator());
249     sb.append("costsFromStart: ")
250         .append(costsFormStart)
251         .append(System.lineSeparator())
252         .append("manhattanDistance: ")
253         .append(manhattanDistance)
254         .append(System.lineSeparator())
255         .append("fullCosts: ")
256         .append(fullCosts)
257         .append(board.toString());
258     return sb.toString();
259 }
260 }

```

## Übung 3

### 1.6 N-Puzzle Exception Source

Folgend ist der Source der implementierten Exceptions angeführt.

#### 1.6.1 InvalidBoardIndexException.java

Folgend ist die Implementierung der InvalidBoardIndexException **extends** RuntimeException angeführt, welche verwendet wird um anzuzeigen das ungültige Indizes gefunden wurden.

Listing 11: InvalidCoardIndexException.java

```

1  /**
2   *
3   */
4  package at.fh.ooe.swe4.puzzle.exception;
5
6  /**
7   * This exception indicates that an given index for a specific board is invalid.
8   *
9   * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
10  * @date Apr 26, 2015
11  */
12 public class InvalidBoardIndexException extends RuntimeException {
13
14     private static final long serialVersionUID = -6222371469322317929L;
15
16     /**
17      *
18      */
19     public InvalidBoardIndexException() {
20         // TODO Auto-generated constructor stub
21     }
22
23     /**
24      * @param message
25      */
26     public InvalidBoardIndexException(String message) {
27         super(message);
28         // TODO Auto-generated constructor stub
29     }
30
31     /**
32      * @param cause
33      */
34     public InvalidBoardIndexException(Throwable cause) {
35         super(cause);
36         // TODO Auto-generated constructor stub
37     }
38
39     /**
40      * @param message
41      * @param cause
42      */
43     public InvalidBoardIndexException(String message, Throwable cause) {
44         super(message, cause);
45         // TODO Auto-generated constructor stub
46     }
47
48     /**
49      * @param message
50      * @param cause
51      * @param enableSuppression
52      * @param writableStackTrace

```

## Übung 3

```

53  */
54  public InvalidBoardIndexException(String message, Throwable cause, boolean enableSuppression,
    ↳ boolean writableStackTrace) {
55      super(message, cause, enableSuppression, writableStackTrace);
56      // TODO Auto-generated constructor stub
57  }
58
59  }

```

### 1.6.2 InvalidMoveException.java

Folgend ist die Implementierung der InvalidMoveException `extends RuntimeException` angeführt, welche verwendet wird um anzuzeigen wenn versucht wurde die leere Kachel auf eine ungültige Position zu verschieben.

Listing 12: InvalidMoveException.java

```

1  /**
2   *
3   */
4  package at.fh.ooe.swe4.puzzle.exception;
5
6  /**
7   * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
8   * @date Apr 26, 2015
9   */
10 public class InvalidMoveException extends RuntimeException {
11
12     private static final long serialVersionUID = -1370866377658671474L;
13
14     /**
15      *
16      */
17     public InvalidMoveException() {
18         // TODO Auto-generated constructor stub
19     }
20
21     /**
22      * @param message
23      */
24     public InvalidMoveException(String message) {
25         super(message);
26         // TODO Auto-generated constructor stub
27     }
28
29     /**
30      * @param cause
31      */
32     public InvalidMoveException(Throwable cause) {
33         super(cause);
34         // TODO Auto-generated constructor stub
35     }
36
37     /**
38      * @param message
39      * @param cause
40      */
41     public InvalidMoveException(String message, Throwable cause) {
42         super(message, cause);
43         // TODO Auto-generated constructor stub
44     }
45

```



## Übung 3

```

46  /**
47   * @param message
48   * @param cause
49   * @param enableSuppression
50   * @param writableStackTrace
51   */
52  public InvalidMoveException(String message, Throwable cause, boolean enableSuppression, boolean
    ↳ writableStackTrace) {
53      super(message, cause, enableSuppression, writableStackTrace);
54      // TODO Auto-generated constructor stub
55  }
56
57  }

```

### 1.6.3 NoSolutionException.java

Folgend ist die Implementierung der NoSolutionException **extends** Exception angeführt, welche verwendet wird um anzuzeigen das keine Lösung gefunden wurde. Diese Ausnahme muss explizit abgefangen werden.

Listing 13: NoSolutionException.java

```

1  package at.fh.ooe.swe4.puzzle.exception;
2
3  public class NoSolutionException extends Exception {
4
5      private static final long serialVersionUID = 5795850985936976146L;
6
7      public NoSolutionException() {
8          // TODO Auto-generated constructor stub
9      }
10
11     public NoSolutionException(String message) {
12         super(message);
13         // TODO Auto-generated constructor stub
14     }
15
16     public NoSolutionException(Throwable cause) {
17         super(cause);
18         // TODO Auto-generated constructor stub
19     }
20
21     public NoSolutionException(String message, Throwable cause) {
22         super(message, cause);
23         // TODO Auto-generated constructor stub
24     }
25
26     public NoSolutionException(String message, Throwable cause, boolean enableSuppression, boolean
    ↳ writableStackTrace) {
27         super(message, cause, enableSuppression, writableStackTrace);
28         // TODO Auto-generated constructor stub
29     }
30
31  }

```

## Übung 3

### 1.7 N-Puzzle Test Source

Folgend ist der Source der N-Puzzle Tests angeführt.

#### 1.7.1 AbstractTest.java

Folgend ist die Implementierung der abstrakten Testklasse angeführt, welche die gemeinsamen Ressourcen für die konkreten Testklassen kapselt.

Hier sind `TestWatcher` deklariert, die das Logging der Tests realisieren.

Listing 14: AbstractTest.java

```

1 package at.fh.ooe.swe.test.api;
2
3 import java.util.ArrayList;
4 import java.util.List;
5 import java.util.function.IntConsumer;
6 import java.util.stream.IntStream;
7
8 import org.apache.log4j.Level;
9 import org.apache.log4j.Logger;
10 import org.junit.ClassRule;
11 import org.junit.Rule;
12 import org.junit.rules.TestWatcher;
13
14 import at.fh.ooe.swe4.junit.test.suite.watcher.LoggingTestClassWatcher;
15 import at.fh.ooe.swe4.junit.test.suite.watcher.LoggingTestInvocationWatcher;
16
17 /**
18  * This is the base test which provides common resources for the test
19  * implementation.
20  *
21  * @author Thomas Herzog <thomas.hertzog@students.fh-hagenberg.at>
22  * @date Apr 26, 2015
23  */
24 public abstract class AbstractTest {
25
26     protected final Logger log;
27     /**
28      * This watcher watches the invocation of an Test class.
29      */
30     @Rule
31     public final TestWatcher methodInvocationLogger = new LoggingTestInvocationWatcher(Level.INFO);
32
33     /**
34      * This watcher is used for logging the test execution.
35      */
36     @ClassRule
37     public static TestWatcher testClassInvocationLogger = new LoggingTestClassWatcher(Level.INFO);
38
39     /**
40      * Default constructor which initializes the logger with the current test
41      * class.
42      */
43     public AbstractTest() {
44         super();
45         log = Logger.getLogger(this.getClass());
46     }
47
48     /**
49      * Creates a container with the given size and sets integer values i in
50      * order from 1 to size.<br>

```

## Übung 3

```
51  * <b>This container will contain no null element.<b>
52  *
53  * @param size
54  *         the size of the container
55  * @return the created container
56  */
57  protected List<Integer> createContainer(final int size) {
58      final List<Integer> container = new ArrayList<Integer>(size);
59      IntStream.range(1, (size + 1))
60          .forEachOrdered(new IntConsumer() {
61              @Override
62              public void accept(int value) {
63                  container.add(value);
64              }
65          });
66
67      return container;
68  }
69 }
```

## Übung 3

### 1.7.2 CalculateParityTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `board.calculateParity()` angeführt.

Listing 15: CalculateParityTest.java

```

1 package at.fh.ooe.swe.test.puzzle.impl.boardImpl;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import static org.junit.Assert.*;
7
8 import org.junit.Test;
9 import org.junit.runner.RunWith;
10 import org.junit.runners.JUnit4;
11
12 import at.fh.ooe.swe.test.api.AbstractTest;
13 import at.fh.ooe.swe4.puzzle.api.Board;
14 import at.fh.ooe.swe4.puzzle.exception.InvalidBoardIndexException;
15 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
16
17 /**
18  * This test class tests the method {@link Board#calculateParity()}.
19  *
20  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
21  * @date Apr 30, 2015
22  */
23 @RunWith(JUnit4.class)
24 public class CalculateParityTest extends AbstractTest {
25
26     // -- Then --
27     @Test(expected = InvalidBoardIndexException.class)
28     public void invalidBoard() {
29         // -- Given --
30         final int size = 10;
31         final List<Integer> container = createContainer((int) Math.pow(size, 2));
32         final Board<Integer> board = new BoardListImpl<>(size, container);
33
34         // -- When --
35         board.calculateParity();
36     }
37
38     @Test
39     public void validEvenParity() {
40         // -- Given --
41         final int size = 2;
42         final List<Integer> container = new ArrayList<>((int) Math.pow(size, 2));
43         container.add(null);
44         container.add(2);
45         container.add(1);
46         container.add(3);
47         final Board<Integer> board = new BoardListImpl<>(size, container);
48
49         // -- When --
50         final int parity = board.calculateParity();
51
52         // -- Then --
53         assertTrue((parity % 2 == 0));
54         assertEquals(2, parity);
55     }
56

```

## Übung 3

```
57  @Test
58  public void validOddParity() {
59      // -- Given --
60      final int size = 2;
61      final List<Integer> container = new ArrayList<>((int) Math.pow(size, 2));
62      container.add(null);
63      container.add(3);
64      container.add(2);
65      container.add(1);
66      final Board<Integer> board = new BoardListImpl<>(size, container);
67
68      // -- When --
69      final int parity = board.calculateParity();
70
71      // -- Then --
72      assertTrue((parity % 2 == 0));
73      assertEquals(4, parity);
74  }
75 }
```

## Übung 3

### 1.7.3 ConstructorTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Konstruktoren der Klasse `BoardListImpl<T extends Comparable<T>>` angeführt.

Listing 16: ConstructorTest.java

```

1 package at.fh.ooe.swe.test.puzzle.impl.boardImpl;
2
3 import static org.junit.Assert.assertNotNull;
4
5 import java.util.ArrayList;
6 import java.util.List;
7
8 import org.junit.Test;
9 import org.junit.runner.RunWith;
10 import org.junit.runners.JUnit4;
11
12 import at.fh.ooe.swe.test.api.AbstractTest;
13 import at.fh.ooe.swe4.puzzle.api.Board;
14 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
15
16 /**
17  * This test class tests the provided constructor of the class {@link BoardListImpl}.
18  *
19  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
20  * @date Apr 26, 2015
21  */
22 @RunWith(JUnit4.class)
23 public class ConstructorTest extends AbstractTest {
24
25     // -- Then --
26     @Test(expected = IllegalArgumentException.class)
27     public void defaultNegativeSize() {
28         // -- Given --
29         final int size = -1;
30
31         // -- When --
32         new BoardListImpl<>(size);
33     }
34
35     // -- Then --
36     @Test(expected = IllegalArgumentException.class)
37     public void defaultZeroSize() {
38         // -- Given --
39         final int size = 0;
40
41         // -- Given / When --
42         new BoardListImpl<>(size);
43     }
44
45     // -- Then --
46     @Test(expected = IllegalArgumentException.class)
47     public void copyNotCorresponding() {
48         // -- Given --
49         final int size = 10;
50         final List<Integer> container = new ArrayList<Integer>((int) Math.pow((size - 1), 2));
51
52         // -- When --
53         new BoardListImpl<>(size, container);
54     }
55
56     // -- Then --

```

# Übung 3

```

57  @Test(expected = IllegalArgumentException.class)
58  public void copyNullContainer() {
59      // -- Given --
60      final int size = 9;
61      final List<Integer> container = null;
62
63      // -- Given --
64      new BoardListImpl<>(size, container);
65  }
66
67  @Test
68  public void defaultConstructor() {
69      // -- Given --
70      final int size = 1;
71
72      // -- When --
73      Board<Integer> board = new BoardListImpl<>(size);
74
75      // -- Then --
76      assertNotNull(board);
77  }
78
79  @Test
80  public void copyConstructor() {
81      // -- Given --
82      final int size = 10;
83      final List<Integer> container = createContainer((int) Math.pow(size, 2));
84
85      // -- When --
86      final Board<Integer> board = new BoardListImpl<>(size, container);
87
88      // -- Then --
89      assertNotNull(board);
90  }
91
92  }

```

## Übung 3

### 1.7.4 EqualsTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `board.equals(Object other)` angeführt.

Listing 17: EqualsTest.java

```

1 package at.fh.ooe.swe.test.puzzle.impl.boardImpl;
2
3 import java.util.ArrayList;
4 import java.util.Collections;
5 import java.util.List;
6
7 import static org.junit.Assert.*;
8
9 import org.junit.Before;
10 import org.junit.Test;
11 import org.junit.runner.RunWith;
12 import org.junit.runners.JUnit4;
13
14 import at.fh.ooe.swe.test.api.AbstractTest;
15 import at.fh.ooe.swe4.puzzle.api.Board;
16 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
17
18 /**
19  * This test class tests the method {@link Board#equals(Object)}.
20  *
21  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at> Apr 27, 2015
22  */
23 @RunWith(JUnit4.class)
24 public class EqualsTest extends AbstractTest {
25
26     @Test
27     public void notEqualNull() {
28         // -- Given --
29         final int size = 10;
30         final List<Integer> container = createContainer((int) Math.pow(size, 2));
31         final Board<Integer> board = new BoardListImpl<>(size, container);
32         final Board<Integer> other = null;
33
34         // -- When --
35         final boolean result = board.equals(other);
36
37         // -- Then --
38         assertFalse(result);
39     }
40
41     @Test
42     public void notEqualNotSameInstance() {
43         // -- Given --
44         final int size = 10;
45         final List<Integer> container = createContainer((int) Math.pow(size, 2));
46         final Board<Integer> board = new BoardListImpl<>(size, container);
47         final String other = null;
48
49         // -- When --
50         final boolean result = board.equals(other);
51
52         // -- Then --
53         assertFalse(result);
54     }
55
56     @Test

```



# Übung 3

```

57 public void notEqualDifferentSize() {
58     // -- Given --
59     final int size = 10;
60     final List<Integer> container = createContainer((int) Math.pow(size, 2));
61     final Board<Integer> board = new BoardListImpl<>(size, container);
62     final int otherSize = 5;
63     final List<Integer> otherContainer = createContainer((int) Math.pow(otherSize, 2));
64     final Board<Integer> otherBoard = new BoardListImpl<>(otherSize, otherContainer);
65
66     // -- When --
67     final boolean result = board.equals(otherBoard);
68
69     // -- Then --
70     assertFalse(result);
71 }
72
73 @Test
74 public void notEqualContainerElements() {
75     // -- Given --
76     final int size = 10;
77     final List<Integer> container = createContainer((int) Math.pow(size, 2));
78     final Board<Integer> board = new BoardListImpl<>(size, container);
79     final List<Integer> otherContainer = createContainer((int) Math.pow(size, 2));
80     Collections.shuffle(otherContainer);
81     final Board<Integer> otherBoard = new BoardListImpl<>(size, otherContainer);
82
83     // -- When --
84     final boolean result = board.equals(otherBoard);
85
86     // -- Then --
87     assertFalse(result);
88 }
89
90 @Test
91 public void isEqual() {
92     // -- Given --
93     final int size = 10;
94     final List<Integer> container = createContainer((int) Math.pow(size, 2));
95     final Board<Integer> board = new BoardListImpl<>(size, container);
96     final Board<Integer> otherBoard = new BoardListImpl<>(size, container);
97
98     // -- When --
99     final boolean result = board.equals(otherBoard);
100
101     // -- Then --
102     assertTrue(result);
103 }
104 }

```

## Übung 3

### 1.7.5 GetEmptyTilePositionTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `board.getEmptyTilePosition()` angeführt.

Listing 18: GetEmptyTilePositionTest.java

```

1  /**
2   *
3   */
4  package at.fh.ooe.swe.test.puzzle.impl.boardImpl;
5
6  import static org.junit.Assert.assertEquals;
7
8  import java.util.List;
9
10 import org.junit.Test;
11 import org.junit.runner.RunWith;
12 import org.junit.runners.JUnit4;
13
14 import at.fh.ooe.swe.test.api.AbstractTest;
15 import at.fh.ooe.swe4.puzzle.api.Board;
16 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
17 import at.fh.ooe.swe4.puzzle.model.Position;
18
19 /**
20  * This test class tests the method {@link Board#getEmptyTilePosition()}
21  *
22  * @author Thomas Herzog <thomas.hertzog@students.fh-hagenberg.at>
23  * @date Apr 26, 2015
24  */
25 @RunWith(JUnit4.class)
26 public class GetEmptyTilePositionTest extends AbstractTest {
27
28     @Test
29     public void multipleEmptyTiles() {
30         // -- Given --
31         final int size = 10;
32         final List<Integer> container = createContainer((int) Math.pow(size, 2));
33         container.set(0, null);
34         container.set(1, null);
35         final Board<Integer> board = new BoardListImpl<>(size, container);
36
37         // -- When --
38         final Position position = board.getEmptyTilePosition();
39
40         // -- Then --
41         assertEquals(new Position(1, 1), position);
42     }
43
44     @Test
45     public void noEmptyTiles() {
46         // -- Given --
47         final int size = 10;
48         final List<Integer> container = createContainer((int) Math.pow(size, 2));
49         final Board<Integer> board = new BoardListImpl<>(size, container);
50
51         // -- When --
52         final Position position = board.getEmptyTilePosition();
53
54         // -- Then --
55         assertEquals(new Position(-1, -1), position);
56     }

```

# Übung 3

```

57
58 /**
59  * Moves the empty tile over the whole board and checks if the position is
60  * properly determined from each set position.
61  */
62 @Test
63 public void validAllRowsAndColumns() {
64     // -- Given --
65     final int size = 10;
66     int oldIdx = 0;
67
68     for (int i = 0; i < size; i++) {
69         for (int j = 0; j < size; j++) {
70             final int idx = ((i * size) + j);
71             final List<Integer> container = createContainer((int) Math.pow(size, 2));
72             container.set(oldIdx, container.get(idx));
73             container.set(idx, null);
74             oldIdx = idx;
75             final Board<Integer> board = new BoardListImpl<>(size, container);
76
77             // -- When --
78             final Position position = board.getEmptyTilePosition();
79
80             // -- Then --
81             assertEquals(new Position((i + 1), (j + 1)), position);
82         }
83     }
84 }
85

```

## Übung 3

### 1.7.6 GetTilePositionTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `board.getTilePosition(T value)` angeführt.

Listing 19: GetTilePositionTest.java

```

1 package at.fh.ooe.swe.test.puzzle.impl.boardImpl;
2
3 import static org.junit.Assert.assertEquals;
4
5 import java.util.List;
6 import java.util.NoSuchElementException;
7
8 import org.junit.Test;
9 import org.junit.runner.RunWith;
10 import org.junit.runners.JUnit4;
11
12 import at.fh.ooe.swe.test.api.AbstractTest;
13 import at.fh.ooe.swe4.puzzle.api.Board;
14 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
15 import at.fh.ooe.swe4.puzzle.model.Position;
16
17 /**
18  * This test class test the method {@link Board#getTilePosition(Comparable)}
19  *
20  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at> Apr 27, 2015
21  */
22 @RunWith(JUnit4.class)
23 public class GetTilePositionTest extends AbstractTest {
24
25     // -- Then --
26     @Test(expected = NoSuchElementException.class)
27     public void noEmptyTile() {
28         // -- Given --
29         final int size = 10;
30         final List<Integer> container = createContainer((int) Math.pow(size, 2));
31         final Board<Integer> board = new BoardListImpl<>(size, container);
32
33         // -- When --
34         board.getTilePosition(null);
35     }
36
37     // -- Then --
38     @Test(expected = NoSuchElementException.class)
39     public void noSuchElement() {
40         // -- Given --
41         final int size = 10;
42         final List<Integer> container = createContainer((int) Math.pow(size, 2));
43         final Board<Integer> board = new BoardListImpl<>(size, container);
44
45         // -- When --
46         board.getTilePosition(Integer.MAX_VALUE);
47     }
48
49     /**
50      * Tests if each tile an be retrieved via tested method.
51      */
52     @Test
53     public void validAllTiles() {
54         // -- Given --
55         final int size = 10;
56         final List<Integer> container = createContainer((int) Math.pow(size, 2));

```

## Übung 3

```
57     final Board<Integer> board = new BoardListImpl<>(size, container);
58
59     for (int i = 1; i <= size; i++) {
60         for (int j = 1; j <= size; j++) {
61             final Integer value = container.get(((i - 1) * size) + (j - 1));
62
63             // -- When --
64             final Position position = board.getTilePosition(value);
65
66             // -- Then --
67             assertEquals(i, position.rowIdx);
68             assertEquals(j, position.colIdx);
69         }
70     }
71 }
72 }
```

## Übung 3

### 1.7.7 GetTileTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `board.getTile(int rowIdx, int colIdx)` angeführt.

Listing 20: GetTileTest.java

```

1 package at.fh.ooe.swe.test.puzzle.impl.boardImpl;
2
3 import static org.junit.Assert.assertEquals;
4
5 import java.util.List;
6
7 import org.junit.Test;
8 import org.junit.runner.RunWith;
9 import org.junit.runners.JUnit4;
10
11 import at.fh.ooe.swe.test.api.AbstractTest;
12 import at.fh.ooe.swe4.puzzle.api.Board;
13 import at.fh.ooe.swe4.puzzle.exception.InvalidBoardIndexException;
14 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
15
16 /**
17  * This class test the method {@link Board#getTile(int, int)}
18  *
19  * @author Thomas Herzog <thomas.hertzog@students.fh-hagenberg.at>
20  * @date Apr 26, 2015
21  */
22 @RunWith(JUnit4.class)
23 public class GetTileTest extends AbstractTest {
24
25     // -- Then --
26     @Test(expected = InvalidBoardIndexException.class)
27     public void invalidRowIndexUnderflow() {
28         // -- Given --
29         final int size = 10;
30         final int rowIdx = 0;
31         final int colIdx = 1;
32         final List<Integer> container = createContainer((int) Math.pow(size, 2));
33         final Board<Integer> board = new BoardListImpl<>(size, container);
34
35         // -- When --
36         board.getTile(rowIdx, colIdx);
37     }
38
39     @Test(expected = InvalidBoardIndexException.class)
40     public void invalidColumnIndexUnderflow() {
41         // -- Given --
42         final int size = 10;
43         final int rowIdx = 1;
44         final int colIdx = 0;
45         final List<Integer> container = createContainer((int) Math.pow(size, 2));
46         final Board<Integer> board = new BoardListImpl<>(size, container);
47
48         // -- When --
49         board.getTile(rowIdx, colIdx);
50     }
51
52     @Test(expected = InvalidBoardIndexException.class)
53     public void invalidRowIndexOverflow() {
54         // -- Given --
55         final int size = 10;
56         final int rowIdx = size + 1;

```

## Übung 3

```

57     final int colIdx = 1;
58     final List<Integer> container = createContainer((int) Math.pow(size, 2));
59     final Board<Integer> board = new BoardListImpl<>(size, container);
60
61     // -- When --
62     board.getTile(rowIdx, colIdx);
63 }
64
65 @Test(expected = InvalidBoardIndexException.class)
66 public void invalidColumnIndexOverflow() {
67     // -- Given --
68     final int size = 10;
69     final int rowIdx = 1;
70     final int colIdx = size + 1;
71     final List<Integer> container = createContainer((int) Math.pow(size, 2));
72     final Board<Integer> board = new BoardListImpl<>(size, container);
73
74     // -- When --
75     board.getTile(rowIdx, colIdx);
76 }
77
78 /**
79  * Tests if all retrieved tiles of the board can be retrieved validly.
80  */
81 @Test
82 public void validAllRowsAndColumns() {
83     // -- Given --
84     final int size = 10;
85     final List<Integer> container = createContainer((int) Math.pow(size, 2));
86     final Board<Integer> board = new BoardListImpl<>(size, container);
87
88     for (int i = 1; i <= size; i++) {
89         for (int j = 1; j <= size; j++) {
90             final int idx = ((i - 1) * size) + (j - 1);
91
92             // -- When --
93             final Integer value = board.getTile(i, j);
94
95             // -- Then --
96             assertEquals(container.get(idx), value);
97         }
98     }
99 }
100 }

```

## Übung 3

### 1.7.8 IsValidTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `board.isValid()` angeführt.

Listing 21: IsValidTest.java

```

1 package at.fh.ooe.swe.test.puzzle.impl.boardImpl;
2
3 import static org.junit.Assert.assertFalse;
4 import static org.junit.Assert.assertTrue;
5
6 import java.util.List;
7
8 import org.junit.Test;
9 import org.junit.runner.RunWith;
10 import org.junit.runners.JUnit4;
11
12 import at.fh.ooe.swe.test.api.AbstractTest;
13 import at.fh.ooe.swe4.puzzle.api.Board;
14 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
15
16 /**
17  * This test class tests the method {@link Board#isValid()}.
18  *
19  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
20  * @date Apr 26, 2015
21  */
22 @RunWith(JUnit4.class)
23 public class IsValidTest extends AbstractTest {
24
25     @Test
26     public void multipleNull() {
27         // -- Given --
28         final int size = 10;
29         final List<Integer> container = createContainer((int) Math.pow(size, 2));
30         container.set(0, null);
31         container.set(1, null);
32         final Board<Integer> board = new BoardListImpl<>(size, container);
33
34         // -- When --
35         final boolean result = board.isValid();
36
37         // -- Then --
38         assertFalse(result);
39     }
40
41     @Test
42     public void multipleNumber() {
43         // -- Given --
44         final int size = 10;
45         final List<Integer> container = createContainer((int) Math.pow(size, 2));
46         container.set(0, null);
47         container.set(1, 1);
48         container.set(2, 1);
49         final Board<Integer> board = new BoardListImpl<>(size, container);
50
51         // -- When --
52         final boolean result = board.isValid();
53
54         // -- Then --
55         assertFalse(result);
56     }
57

```



## Übung 3

```
58  @Test
59  public void noEmptyTile() {
60      // -- Given --
61      final int size = 10;
62      final List<Integer> container = createContainer((int) Math.pow(size, 2));
63      final Board<Integer> board = new BoardListImpl<>(size, container);
64
65      // -- When --
66      final boolean result = board.isValid();
67
68      // -- Then --
69      assertFalse(result);
70  }
71
72  @Test
73  public void valid() {
74      // -- Given --
75      final int size = 10;
76      final List<Integer> container = createContainer((int) Math.pow(size, 2));
77      container.set(0, null);
78      final Board<Integer> board = new BoardListImpl<>(size, container);
79
80      // -- When --
81      final boolean result = board.isValid();
82
83      // -- Then --
84      assertTrue(result);
85  }
86 }
```

## Übung 3

### 1.7.9 MakeMovesTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `board.makeMoves(List<Move> moves)` angeführt.

Listing 22: MakeMovesTest.java

```

1  /**
2   *
3   */
4  package at.fh.ooe.swe.test.puzzle.impl.boardImpl;
5
6  import java.util.ArrayList;
7  import java.util.Arrays;
8  import java.util.List;
9
10 import static org.junit.Assert.*;
11 import org.junit.Before;
12 import org.junit.Test;
13 import org.junit.runner.RunWith;
14 import org.junit.runners.JUnit4;
15
16 import at.fh.ooe.swe.test.api.AbstractTest;
17 import at.fh.ooe.swe4.puzzle.api.Board;
18 import at.fh.ooe.swe4.puzzle.api.Board.Move;
19 import at.fh.ooe.swe4.puzzle.exception.InvalidMoveException;
20 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
21 import at.fh.ooe.swe4.puzzle.model.Position;
22
23 /**
24  * This test class tests the method {@link Board#makeMoves(Iterable)}.<br>
25  * This test class depends on proper functionality of the method
26  * {@link Board#getEmptyTilePosition()}
27  *
28  * @author Thomas Herzog <thomas.hertzog@students.fh-hagenberg.at>
29  * @date Apr 26, 2015
30  */
31 @RunWith(JUnit4.class)
32 public class MakeMovesTest extends AbstractTest {
33
34     // -- Then --
35     @Test(expected = InvalidMoveException.class)
36     public void nullIterable() {
37         // -- Given --
38         final int size = 10;
39         final List<Integer> container = createContainer((int) Math.pow(size, 2));
40         Board<Integer> board = new BoardListImpl<>(size, container);
41
42         // -- When --
43         board.makeMoves((Iterable<Move>) null);
44     }
45
46     // -- Then --
47     @Test(expected = InvalidMoveException.class)
48     public void containsNull() {
49         // -- Given --
50         final int size = 10;
51         final List<Integer> container = createContainer((int) Math.pow(size, 2));
52         Board<Integer> board = new BoardListImpl<>(size, container);
53         final List<Move> moves = new ArrayList<Move>(Move.values().length + 1);
54         for (Move direction : Move.values()) {
55             moves.add(direction);
56         }

```

# Übung 3

```

57     moves.add(null);
58
59     // -- When --
60     board.makeMoves(moves);
61 }
62
63 // -- Then --
64 @Test(expected = InvalidMoveException.class)
65 public void invalidBoard() {
66     // -- Given --
67     final int size = 10;
68     final List<Integer> container = createContainer((int) Math.pow(size, 2));
69     Board<Integer> board = new BoardListImpl<>(size, container);
70
71     // -- When --
72     board.makeMoves(Arrays.asList(new Move[] { Move.DOWN }));
73 }
74
75 // -- Then --
76 @Test(expected = InvalidMoveException.class)
77 public void borderOverflow() {
78     // -- Given --
79     final int size = 10;
80     final List<Integer> container = createContainer((int) Math.pow(size, 2));
81     container.set(0, null);
82     final Board<Integer> board = new BoardListImpl<>(size, container);
83     final List<Move> moves = new ArrayList<Move>();
84     for (int i = 0; i < size; i++) {
85         moves.add(Move.RIGHT);
86     }
87
88     // -- When --
89     board.makeMoves(moves);
90 }
91
92 @Test
93 public void validAllFieldsRowPerRowLeftToRight() {
94     // -- Given --
95     final int size = 10;
96     final List<Integer> container = createContainer((int) Math.pow(size, 2));
97     container.set(0, null);
98     final Board<Integer> board = new BoardListImpl<>(size, container);
99     final List<Move> moves = new ArrayList<Move>();
100     boolean invert = Boolean.FALSE;
101     for (int i = 0; i <= size; i++) {
102         final Move direction;
103         if (invert) {
104             direction = Move.LEFT;
105             invert = Boolean.FALSE;
106         } else {
107             direction = Move.RIGHT;
108             invert = Boolean.TRUE;
109         }
110         for (int j = 0; j < (size - 1); j++) {
111             moves.add(direction);
112         }
113         if (i < (size - 1)) {
114             moves.add(Move.DOWN);
115         }
116     }
117
118     // -- When --
119     board.makeMoves(moves);

```

Übung 3

---

```
120 |  
121 |     // -- Then --  
122 |     final Position emptyTilePosition = board.getEmptyTilePosition();  
123 |     assertEquals(size, emptyTilePosition.rowIdx);  
124 |     assertEquals(size, emptyTilePosition.colIdx);  
125 | }  
126 | }
```

## Übung 3

### 1.7.10 MoveDownTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `board.moveDown()` angeführt.

Listing 23: MoveDownTest.java

```

1 package at.fh.ooe.swe.test.puzzle.impl.boardImpl;
2
3 import static org.junit.Assert.assertEquals;
4 import static org.junit.Assert.fail;
5
6 import java.util.List;
7
8 import org.junit.Test;
9
10 import at.fh.ooe.swe.test.api.AbstractTest;
11 import at.fh.ooe.swe4.puzzle.api.Board;
12 import at.fh.ooe.swe4.puzzle.exception.InvalidMoveException;
13 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
14 import at.fh.ooe.swe4.puzzle.model.Position;
15
16 /**
17  * This test class tests the method {@link Board#moveDown()}.<br>
18  * This test class depends on proper functionality of the method
19  * {@link Board#getEmptyTilePosition()}
20  *
21  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
22  * @date Apr 26, 2015
23  */
24 public class MoveDownTest extends AbstractTest {
25
26     // -- Then --
27     @Test(expected = InvalidMoveException.class)
28     public void invalidBoard() {
29         // -- Given --
30         final int size = 10;
31         final List<Integer> container = createContainer((int) Math.pow(size, 2));
32         final Board<Integer> board = new BoardListImpl<>(size, container);
33
34         board.moveDown();
35     }
36
37     @Test
38     public void alreadyOnBottomAllColumns() {
39         // -- Given --
40         final int size = 10;
41         int lastRow = (size * (size - 1));
42         for (int i = 1; i <= size; i++) {
43             try {
44                 int idx = lastRow + (i - 1);
45                 final List<Integer> container = createContainer((int) Math.pow(size, 2));
46                 container.set(idx, null);
47                 Board<Integer> board = new BoardListImpl<>(size, container);
48
49                 // -- When --
50                 board.moveDown();
51
52                 // -- Then --
53                 fail("Expected InvalidMoveException");
54             } catch (InvalidMoveException e) {
55                 // Should throw exception
56             }
57         }
58     }
59 }

```

## Übung 3

```
57     }
58 }
59
60 @Test
61 public void validAllColsFromTop() {
62     // -- Given --
63     final int size = 10;
64     for (int i = 1; i < size; i++) {
65         final List<Integer> container = createContainer((int) Math.pow(size, 2));
66         int idx = (i - 1);
67         container.set(idx, null);
68         final Board<Integer> board = new BoardListImpl<>(size, container);
69
70         // -- When --
71         for (int j = 1; j < size; j++) {
72             board.moveDown();
73         }
74
75         // -- Then --
76         final Position emptyTilePosition = board.getEmptyTilePosition();
77         assertEquals(size, emptyTilePosition.rowIdx);
78         assertEquals(i, emptyTilePosition.colIdx);
79     }
80 }
81 }
```

## Übung 3

### 1.7.11 MoveUpTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `board.moveUp()` angeführt.

Listing 24: MoveUpTest.java

```

1 package at.fh.ooe.swe.test.puzzle.impl.boardImpl;
2
3 import static org.junit.Assert.assertEquals;
4 import static org.junit.Assert.fail;
5
6 import java.util.List;
7
8 import org.junit.Test;
9
10 import at.fh.ooe.swe.test.api.AbstractTest;
11 import at.fh.ooe.swe4.puzzle.api.Board;
12 import at.fh.ooe.swe4.puzzle.exception.InvalidMoveException;
13 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
14 import at.fh.ooe.swe4.puzzle.model.Position;
15
16 /**
17  * This test class tests the method {@link Board#moveUp()}.
18  *
19  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
20  * @date Apr 26, 2015
21  */
22 public class MoveUpTest extends AbstractTest {
23
24     // -- Then --
25     @Test(expected = InvalidMoveException.class)
26     public void invalidBoard() {
27         // -- Given --
28         final int size = 10;
29         final List<Integer> container = createContainer((int) Math.pow(size, 2));
30         final Board<Integer> board = new BoardListImpl<>(size, container);
31
32         // -- When --
33         board.moveUp();
34     }
35
36     @Test
37     public void alreadyOnTopAllColumns() {
38         final int size = 10;
39         int oldIdx = 0;
40         for (int i = 0; i < size; i++) {
41             try {
42                 // -- Given --
43                 final List<Integer> container = createContainer((int) Math.pow(size, 2));
44                 container.set(oldIdx, container.get(i));
45                 container.set(i, null);
46                 oldIdx = i;
47                 final Board<Integer> board = new BoardListImpl<>(size, container);
48
49                 // -- When --
50                 board.moveUp();
51
52                 // -- Then --
53                 fail("Expected InvalidMoveException");
54             } catch (InvalidMoveException e) {
55                 // Should throw exception
56             }
57         }
58     }
59 }

```

# Übung 3

```

58     }
59
60     @Test
61     public void validAllColsFormBottom() {
62         // -- Given --
63         final int size = 10;
64         int lastRow = (size * (size - 1));
65         for (int i = 1; i <= size; i++) {
66             int idx = lastRow + (i - 1);
67             List<Integer> container = createContainer((int) Math.pow(size, 2));
68             container.set(idx, null);
69             Board<Integer> board = new BoardListImpl<>(size, container);
70
71             // -- When --
72             for (int j = 1; j < size; j++) {
73                 board.moveUp();
74             }
75
76             // -- Then --
77             final Position emptyTilePosition = board.getEmptyTilePosition();
78             assertEquals(1, emptyTilePosition.rowIdx);
79             assertEquals(i, emptyTilePosition.colIdx);
80         }
81     }
82 }

```



## Übung 3

### 1.7.12 MoveLeftTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `board.moveLeft()` angeführt.

Listing 25: MoveLeftTest.java

```

1 package at.fh.ooe.swe.test.puzzle.impl.boardImpl;
2
3 import static org.junit.Assert.*;
4
5 import java.util.List;
6
7 import org.junit.Before;
8 import org.junit.Test;
9
10 import at.fh.ooe.swe.test.api.AbstractTest;
11 import at.fh.ooe.swe4.puzzle.api.Board;
12 import at.fh.ooe.swe4.puzzle.exception.InvalidMoveException;
13 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
14 import at.fh.ooe.swe4.puzzle.model.Position;
15
16 /**
17  * This test class tests the method {@link Board#moveLeft()}.<br>
18  * This test class depends on proper functionality of method
19  * {@link Board#getEmptyTilePosition()}
20  *
21  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
22  * @date Apr 26, 2015
23  */
24 public class MoveLeftTest extends AbstractTest {
25
26     // -- Then --
27     @Test(expected = InvalidMoveException.class)
28     public void invalidBoard() {
29         // -- Given --
30         final int size = 10;
31         final List<Integer> container = createContainer((int) Math.pow(size, 2));
32         final Board<Integer> board = new BoardListImpl<>(size, container);
33
34         // -- When --
35         board.moveLeft();
36     }
37
38     @Test
39     public void alreadyLeftAllRows() {
40         // -- Given --
41         final int size = 10;
42         for (int i = 1; i <= size; i++) {
43             try {
44                 int idx = (size * (i - 1));
45                 final List<Integer> container = createContainer((int) Math.pow(size, 2));
46                 container.set(idx, null);
47                 Board<Integer> board = new BoardListImpl<>(size, container);
48
49                 // -- When --
50                 board.moveLeft();
51
52                 // -- Then --
53                 fail("Expected InvalidMoveException");
54             } catch (InvalidMoveException e) {
55                 // should throw exception
56             }
57         }
58     }
59 }

```

## Übung 3

```
57     }
58 }
59
60 @Test
61 public void validAllRowsFromRight() {
62     // -- Given --
63     final int size = 10;
64     int oldIdx = ((size * size) - 1);
65     for (int i = size; i > 0; i--) {
66         int idx = (size * i) - 1;
67         final List<Integer> container = createContainer((int) Math.pow(size, 2));
68         container.set(idx, null);
69         final Board<Integer> board = new BoardListImpl<>(size, container);
70
71         // -- When --
72         for (int j = 0; j < (size - 1); j++) {
73             board.moveLeft();
74         }
75
76         // -- Then --
77         final Position emptytilePosition = board.getEmptyTilePosition();
78         assertEquals(i, emptytilePosition.rowIdx);
79         assertEquals(1, emptytilePosition.colIdx);
80     }
81 }
82 }
```

## Übung 3

### 1.7.13 MoveRightTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `board.moveRight()` angeführt.

Listing 26: MoveRightTest.java

```

1 package at.fh.ooe.swe.test.puzzle.impl.boardImpl;
2
3 import static org.junit.Assert.assertEquals;
4 import static org.junit.Assert.fail;
5
6 import java.util.List;
7
8 import org.junit.Test;
9
10 import at.fh.ooe.swe.test.api.AbstractTest;
11 import at.fh.ooe.swe4.puzzle.api.Board;
12 import at.fh.ooe.swe4.puzzle.exception.InvalidMoveException;
13 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
14 import at.fh.ooe.swe4.puzzle.model.Position;
15
16 /**
17  * This test class tests the method {@link Board#moveRight()}.<br>
18  * This test class depends on proper functionality of method
19  * {@link Board#getEmptyTilePosition()}
20  *
21  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
22  * @date Apr 26, 2015
23  */
24 public class MoveRightTest extends AbstractTest {
25
26     // -- Then --
27     @Test(expected = InvalidMoveException.class)
28     public void invalidBoard() {
29         // -- Given --
30         final int size = 10;
31         final List<Integer> container = createContainer((int) Math.pow(size, 2));
32         final Board<Integer> board = new BoardListImpl<>(size, container);
33
34         // -- When --
35         board.moveRight();
36     }
37
38     @Test
39     public void alreadyRightAllRows() {
40         // -- Given --
41         final int size = 10;
42         for (int i = 1; i <= size; i++) {
43             try {
44                 int idx = ((size * i) - 1);
45                 final List<Integer> container = createContainer((int) Math.pow(size, 2));
46                 container.set(idx, null);
47                 final Board<Integer> board = new BoardListImpl<>(size, container);
48
49                 // -- When --
50                 board.moveRight();
51
52                 // -- Then --
53                 fail("Expected InvalidMoveException");
54             } catch (InvalidMoveException e) {
55                 // Should throw exception
56             }
57         }
58     }
59 }

```

## Übung 3

```
57     }
58 }
59
60 @Test
61 public void validAllRowsFromLeft() {
62     // -- Given --
63     final int size = 10;
64     for (int i = 1; i <= size; i++) {
65         int idx = (size * (i - 1));
66         final List<Integer> container = createContainer((int) Math.pow(size, 2));
67         container.set(idx, null);
68         final Board<Integer> board = new BoardListImpl<>(size, container);
69
70         // -- When --
71         for (int j = 1; j < size; j++) {
72             board.moveRight();
73         }
74
75         // -- Then --
76         final Position emptytilePosition = board.getEmptyTilePosition();
77         assertEquals(i, emptytilePosition.rowIdx);
78         assertEquals(size, emptytilePosition.colIdx);
79     }
80 }
81 }
```

## Übung 3

### 1.7.14 SetTileTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `board.setTile(int rowIdx, int colIdx, T value)` angeführt.

Listing 27: SetTileTest.java

```

1 package at.fh.ooe.swe.test.puzzle.impl.boardImpl;
2
3 import static org.junit.Assert.assertEquals;
4
5 import java.util.List;
6
7 import org.junit.Test;
8 import org.junit.runner.RunWith;
9 import org.junit.runners.JUnit4;
10
11 import at.fh.ooe.swe.test.api.AbstractTest;
12 import at.fh.ooe.swe4.puzzle.api.Board;
13 import at.fh.ooe.swe4.puzzle.exception.InvalidBoardIndexException;
14 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
15
16 /**
17  * This test class tests the method {@link Board#setTile(int, int, Number)} This
18  * test depends on proper function of the method<br>
19  * {@link Board#getTile(int, int)} method which is used to get the added value
20  * from the board
21  *
22  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
23  * @date Apr 26, 2015
24  */
25 @RunWith(JUnit4.class)
26 public class SetTileTest extends AbstractTest {
27
28     // -- Then --
29     @Test(expected = InvalidBoardIndexException.class)
30     public void invalidRowIndexUnderflow() {
31         // -- Given --
32         final int size = 10;
33         final int rowIdx = 0;
34         final int colIdx = 1;
35         final Board<Integer> board = new BoardListImpl<>(size);
36
37         // When
38         board.setTile(rowIdx, colIdx, 1);
39     }
40
41     @Test(expected = InvalidBoardIndexException.class)
42     public void invalidColumnIndexUnderflow() {
43         // -- Given --
44         final int size = 10;
45         final int rowIdx = 1;
46         final int colIdx = 0;
47         final Board<Integer> board = new BoardListImpl<>(size);
48
49         // When
50         board.setTile(rowIdx, colIdx, 1);
51     }
52
53     @Test(expected = InvalidBoardIndexException.class)
54     public void invalidRowIndexOverflow() {
55         // -- Given --
56         final int size = 10;

```

# Übung 3

```

57     final int rowIdx = size + 1;
58     final int colIdx = 1;
59     final Board<Integer> board = new BoardListImpl<>(size);
60
61     // When
62     board.setTile(rowIdx, colIdx, 1);
63 }
64
65 @Test(expected = InvalidBoardIndexException.class)
66 public void invalidColumnIndexOverflow() {
67     // -- Given --
68     final int size = 10;
69     final int rowIdx = 1;
70     final int colIdx = size + 1;
71     final Board<Integer> board = new BoardListImpl<>(size);
72
73     // When
74     board.setTile(rowIdx, colIdx, 1);
75 }
76
77 /**
78  * Tests if all elements can be set on the board.
79  */
80 public void valid() {
81     // -- Given --
82     final int size = 10;
83     final List<Integer> container = createContainer(size);
84     container.set(0, null);
85     final Board<Integer> board = new BoardListImpl<>(size);
86
87     for (int i = 1; i <= size; i++) {
88         for (int j = 1; j <= size; j++) {
89             final int idx = (((i - 1) * size) + (j - 1));
90             final Integer value = container.get(idx);
91
92             // -- When --
93             board.setTile(i, j, value);
94
95             // -- Then --
96             assertEquals(value, board.getTile(i, j));
97         }
98     }
99 }
100 }

```

## Übung 3

### 1.7.15 ConstructorTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Konstruktoren der Klasse `SearchNode<T extends Comparable<T>>` angeführt.

Listing 28: ConstructorTest.java

```

1 package at.fh.ooe.swe.test.puzzle.model.searchNode;
2
3 import java.util.List;
4
5 import org.junit.Test;
6 import org.junit.runner.RunWith;
7 import org.junit.runners.JUnit4;
8
9 import static org.junit.Assert.*;
10 import at.fh.ooe.swe.test.api.AbstractTest;
11 import at.fh.ooe.swe4.puzzle.api.Board;
12 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
13 import at.fh.ooe.swe4.puzzle.model.SearchNode;
14
15 /**
16  * This test class tests the provided constructors of the {@link SearchNode}
17  * class.
18  *
19  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
20  * @date Apr 27, 2015
21  */
22 @RunWith(JUnit4.class)
23 public class ConstructorTest extends AbstractTest {
24
25     // -- Then --
26     @Test(expected = IllegalArgumentException.class)
27     public void boardNull() {
28         // -- Given
29         final Board<Integer> board = null;
30
31         // -- When --
32         new SearchNode<>(0, null, null, board, null);
33     }
34
35     // -- Then --
36     @Test(expected = IllegalArgumentException.class)
37     public void goalNull() {
38         // -- Given
39         final Board<Integer> board = null;
40
41         // -- When --
42         new SearchNode<>(0, null, board, null, null);
43     }
44
45     @Test
46     public void valid() {
47         // -- Given --
48         final int size = 10;
49         final List<Integer> container = createContainer((int) Math.pow(size, 2));
50         container.set(0, null);
51         final Board<Integer> board = new BoardListImpl<>(size, container);
52         final List<Integer> goalContainer = createContainer((int) Math.pow(size, 2));
53         goalContainer.set(0, null);
54         final Board<Integer> goal = new BoardListImpl<>(size, goalContainer);
55
56         // -- When --

```

## Übung 3

```
57     final SearchNode<Integer> node = new SearchNode<>(0, null, board, goal, null);
58
59     // -- Then --
60     assertNotNull(node);
61     assertTrue(board.equals(node.getBoard()));
62     assertEquals(0, node.getCostsFormStart());
63 }
64 }
```



## Übung 3

### 1.7.16 ComparableTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `node.compare(SearchNode<T> other)` angeführt.

Listing 29: ComparableTest.java

```

1 package at.fh.ooe.swe.test.puzzle.model.searchNode;
2
3 import static org.junit.Assert.assertEquals;
4
5 import java.util.ArrayList;
6 import java.util.Collections;
7 import java.util.List;
8 import java.util.function.IntConsumer;
9 import java.util.stream.IntStream;
10
11 import org.junit.Test;
12 import org.junit.runner.RunWith;
13 import org.junit.runners.JUnit4;
14
15 import at.fh.ooe.swe.test.api.AbstractTest;
16 import at.fh.ooe.swe4.puzzle.api.Board;
17 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
18 import at.fh.ooe.swe4.puzzle.model.SearchNode;
19
20 /**
21  * This test class tests the method {@link SearchNode#compareTo(SearchNode)}.
22  *
23  * @author Thomas Herzog <thomas.hertzog@students.fh-hagenberg.at>
24  * @date Apr 28, 2015
25  */
26 @RunWith(JUnit4.class)
27 public class ComparableTest extends AbstractTest {
28
29     @Test
30     public void validAllDifferentCosts() {
31         // -- Given --
32         final int size = 10;
33         final List<Integer> container = createContainer((int) Math.pow(size, 2));
34         container.set(0, null);
35         final Board<Integer> board = new BoardListImpl<>(size, container);
36         final List<SearchNode<Integer>> orderedNodes = new ArrayList<SearchNode<Integer>>();
37         IntStream.range(0, 10)
38             .forEachOrdered(new IntConsumer() {
39
40                 @Override
41                 public void accept(int value) {
42                     final SearchNode<Integer> node = new SearchNode<>(value, null, board, board, null);
43                     orderedNodes.add(node);
44                 }
45             });
46         final List<SearchNode<Integer>> shuffledNodes = new
47             ↳ ArrayList<SearchNode<Integer>>(orderedNodes);
48         Collections.shuffle(shuffledNodes);
49         Collections.sort(shuffledNodes, SearchNode::compareTo);
50
51         // -- When --
52         IntStream.range(0, shuffledNodes.size())
53             .forEachOrdered(new IntConsumer() {
54
55                 @Override
56                 public void accept(int i) {

```

## Übung 3

```
56         // - Then --
57         assertEquals(shuffledNodes.get(i)
58                     .getCostsFormStart(), i);
59     }
60 }
61 }
62
63 }
```

## Übung 3

### 1.7.17 EqualsTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `node.equals(Object other)` angeführt.

Listing 30: EqualsTest.java

```

1 package at.fh.ooe.swe.test.puzzle.model.searchNode;
2
3 import static org.junit.Assert.assertFalse;
4 import static org.junit.Assert.assertTrue;
5
6 import java.util.Collections;
7 import java.util.List;
8
9 import org.junit.Before;
10 import org.junit.Test;
11 import org.junit.runner.RunWith;
12 import org.junit.runners.JUnit4;
13
14 import at.fh.ooe.swe.test.api.AbstractTest;
15 import at.fh.ooe.swe4.puzzle.api.Board;
16 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
17 import at.fh.ooe.swe4.puzzle.model.SearchNode;
18
19 @RunWith(JUnit4.class)
20 public class EqualsTest extends AbstractTest {
21
22     @Before
23     public void init() {
24     }
25
26     @Test
27     public void notEqualNull() {
28         // -- Given --
29         final int size = 10;
30         final List<Integer> container = createContainer((int) Math.pow(size, 2));
31         container.set(0, null);
32         final Board<Integer> board = new BoardListImpl<>(size, container);
33         final SearchNode<Integer> node = new SearchNode<>(0, null, board, board, null);
34         // -- When --
35         final boolean result = node.equals(null);
36
37         // -- Then --
38         assertFalse(result);
39     }
40
41     @Test
42     public void notEqualDifferentInstance() {
43         // -- Given --
44         final int size = 10;
45         final List<Integer> container = createContainer((int) Math.pow(size, 2));
46         container.set(0, null);
47         final Board<Integer> board = new BoardListImpl<>(size, container);
48         final SearchNode<Integer> node = new SearchNode<>(0, null, board, board, null);
49
50         // -- When --
51         final boolean result = node.equals(new Integer(0));
52
53         // -- Then --
54         assertFalse(result);
55     }
56

```

## Übung 3

```

57  @Test
58  public void notEqualNotSameBoard() {
59      // -- Given --
60      final int size = 10;
61      final List<Integer> container = createContainer((int) Math.pow(size, 2));
62      container.set(0, null);
63      final Board<Integer> board = new BoardListImpl<>(size, container);
64      final SearchNode<Integer> node = new SearchNode<>(0, null, board, board, null);
65      final List<Integer> otherContainer = createContainer((int) Math.pow(size, 2));
66      otherContainer.set(size - 1, null);
67      final Board<Integer> other = new BoardListImpl<>(size, otherContainer);
68
69      // -- When --
70      final boolean result = node.equals(new SearchNode<Integer>(0, null, other, other, null));
71
72      // -- Then --
73      assertFalse(result);
74  }
75
76  @Test
77  public void equal() {
78      // -- Given --
79      final int size = 10;
80      final List<Integer> container = createContainer((int) Math.pow(size, 2));
81      container.set(0, null);
82      final Board<Integer> board = new BoardListImpl<>(size, container);
83      final SearchNode<Integer> node = new SearchNode<>(0, null, board, board, null);
84      final Board<Integer> other = new BoardListImpl<>(size, container);
85
86      // -- When --
87      final boolean result = node.equals(new SearchNode<>(0, null, other, other, null));
88
89      // -- Then --
90      assertTrue(result);
91  }
92  }

```

## Übung 3

### 1.7.18 EstimatedCostsToTargetTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `node.estimatedCostsToTarget()` angeführt.

Listing 31: EstimatedCostsToTargetTest.java

```

1 package at.fh.ooe.swe.test.puzzle.model.searchNode;
2
3 import static org.junit.Assert.assertEquals;
4
5 import java.util.ArrayList;
6 import java.util.Collections;
7 import java.util.List;
8 import java.util.function.IntConsumer;
9 import java.util.stream.IntStream;
10
11 import org.junit.Test;
12 import org.junit.runner.RunWith;
13 import org.junit.runners.JUnit4;
14
15 import at.fh.ooe.swe.test.api.AbstractTest;
16 import at.fh.ooe.swe4.puzzle.api.Board;
17 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
18 import at.fh.ooe.swe4.puzzle.model.Position;
19 import at.fh.ooe.swe4.puzzle.model.SearchNode;
20
21 /**
22  * This test class tests the method {@link SearchNode#estimatedCostsToTarget()}.<br>
23  * This test class depends on proper functionality of the method
24  * {@link Board#getTilePosition(Comparable)}
25  *
26  * @author Thomas Herzog <thomas.hertzog@students.fh-hagenberg.at>
27  * @date Apr 27, 2015
28  */
29 @RunWith(JUnit4.class)
30 public class EstimatedCostsToTargetTest extends AbstractTest {
31
32     // -- Then --
33     @Test(expected = IllegalArgumentException.class)
34     public void nullGoalBaord() {
35         // -- Given --
36         final int size = 10;
37         final List<Integer> container = createContainer((int) Math.pow(size, 2));
38         container.set(0, null);
39         final Board<Integer> board = new BoardListImpl<>(size, container);
40         final SearchNode<Integer> node = new SearchNode<>(0, null, board, null, null);
41
42         // -- When --
43         node.estimatedCostsToTarget();
44     }
45
46     // -- Then --
47     @Test(expected = IllegalArgumentException.class)
48     public void goalOfDifferentSize() {
49         // -- Given --
50         final int size = 10;
51         final List<Integer> container = createContainer((int) Math.pow(size, 2));
52         container.set(0, null);
53         final Board<Integer> board = new BoardListImpl<>(size, container);
54         final int goalSize = size + 1;
55         final List<Integer> goalContainer = createContainer((int) Math.pow(goalSize, 2));
56         goalContainer.set(0, null);

```

## Übung 3

```

57     final Board<Integer> goal = new BoardListImpl<>(goalSize, goalContainer);
58     final SearchNode<Integer> node = new SearchNode<>(0, null, board, goal, null);
59
60     // -- When --
61     node.estimatedCostsToTarget();
62 }
63
64 @Test
65 public void validZeroDistance() {
66     // -- Given --
67     final int size = 10;
68     final List<Integer> container = createContainer((int) Math.pow(size, 2));
69     container.set(0, null);
70     final Board<Integer> board = new BoardListImpl<>(size, container);
71     final Board<Integer> goal = new BoardListImpl<>(size, container);
72     final SearchNode<Integer> node = new SearchNode<>(0, null, board, goal, null);
73
74     // -- When --
75     final int cost = node.estimatedCostsToTarget();
76
77     // -- Then --
78     assertEquals(0, cost);
79 }
80
81 @Test
82 public void validShuffledGoal() {
83     IntStream.range(0, 10)
84         .forEach(new IntConsumer() {
85             @Override
86             public void accept(int iterationCount) {
87                 // -- Given --
88                 final int size = 10;
89                 final List<Integer> container = createContainer((int) Math.pow(size, 2));
90                 container.set(0, null);
91                 final Board<Integer> board = new BoardListImpl<>(size, container);
92                 final List<Integer> goalContainer = new ArrayList<Integer>(container);
93                 Collections.shuffle(goalContainer);
94                 final Board<Integer> goal = new BoardListImpl<>(board.size(), goalContainer);
95                 final SearchNode<Integer> node = new SearchNode<>(0, null, board, goal, null);
96                 int costs = 0;
97                 for (int i = 1; i <= board.size(); i++) {
98                     for (int j = 1; j <= board.size(); j++) {
99                         final Integer value = board.getTile(i, j);
100                         if (value != null) {
101                             final Position position = goal.getTilePosition(value);
102                             // uses same algorithm as SearchNode via
103                             // static
104                             // final lambda member variable
105                             costs += SearchNode.CALC_MANHATTAN_DIST.apply(new Position(i, j), position);
106                         }
107                     }
108                 }
109
110                 // -- When --
111                 final int cost = node.estimatedCostsToTarget();
112
113                 // -- Then --
114                 assertEquals(costs, cost);
115             }
116         });
117 }
118 }

```

## Übung 3

### 1.7.19 IteratorTest.java

Folgend ist die Implementierung der Testklasse für die Tests der `Iterator<T>` Implementierung für die Klasse `SearchNode<T extends Comparable<T>>` angeführt.

Listing 32: IteratorTest.java

```

1 package at.fh.ooe.swe.test.puzzle.model.searchNode;
2
3 import static org.junit.Assert.assertEquals;
4
5 import java.util.ArrayList;
6 import java.util.Iterator;
7 import java.util.List;
8 import java.util.NoSuchElementException;
9
10 import org.junit.Test;
11 import org.junit.runner.RunWith;
12 import org.junit.runners.JUnit4;
13
14 import at.fh.ooe.swe.test.api.AbstractTest;
15 import at.fh.ooe.swe4.puzzle.api.Board;
16 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
17 import at.fh.ooe.swe4.puzzle.model.SearchNode;
18
19 /**
20  * This test class tests the method {@link SearchNode#iterator()} which returns
21  * an iterator instance.<br>
22  * So this test class test the iterator implementation which has been
23  * implemented as inner class.
24  *
25  * @author Thomas Herzog <thomas.hertzog@students.fh-hagenberg.at>
26  * @date Apr 28, 2015
27  */
28 @RunWith(JUnit4.class)
29 public class IteratorTest extends AbstractTest {
30
31     // -- Then --
32     @Test(expected = NoSuchElementException.class)
33     public void overflow() {
34         // -- Given --
35         final int size = 10;
36         final List<Integer> container = createContainer((int) Math.pow(size, 2));
37         container.set(0, null);
38         final Board<Integer> board = new BoardListImpl<>(size, container);
39         final SearchNode<Integer> node = new SearchNode<>(0, null, board, board, null);
40
41         // -- When --
42         final Iterator<SearchNode<Integer>> it = node.iterator();
43         while (true) {
44             it.next();
45         }
46     }
47
48     @Test
49     public void noPredecessors() {
50         // -- Given --
51         final int size = 10;
52         final List<Integer> container = createContainer((int) Math.pow(size, 2));
53         container.set(0, null);
54         final Board<Integer> board = new BoardListImpl<>(size, container);
55         final SearchNode<Integer> node = new SearchNode<>(0, null, board, board, null);
56     }

```

## Übung 3

```

57     // -- When --
58     final Iterator<SearchNode<Integer>> it = node.iterator();
59     int count = 0;
60     while (it.hasNext()) {
61         it.next();
62         count++;
63     }
64
65     // -- Then --
66     assertEquals(1, count);
67 }
68
69 @Test
70 public void valid() {
71     // -- Given --
72     final int size = 10;
73     final List<Integer> container = createContainer((int) Math.pow(size, 2));
74     container.set(0, null);
75     final Board<Integer> board = new BoardListImpl<>(size, container);
76     final SearchNode<Integer> node = new SearchNode<>(0, null, board, board, null);
77     final List<SearchNode<Integer>> nodes = new ArrayList<SearchNode<Integer>>();
78     nodes.add(node);
79     SearchNode<Integer> tmp = node;
80     int expectedCount = 1;
81     for (int i = 0; i < 10; i++) {
82         final SearchNode<Integer> predecessor = new SearchNode<>(0, null, board, board, null);
83         tmp.setPredecessor(predecessor);
84         tmp = predecessor;
85         nodes.add(predecessor);
86         expectedCount++;
87     }
88
89     // -- When --
90     final Iterator<SearchNode<Integer>> it = node.iterator();
91     int count = 0;
92     while (it.hasNext()) {
93         final SearchNode<Integer> itNode = it.next();
94
95         // -- Then --
96         assertEquals(itNode, nodes.get(count));
97         count++;
98     }
99
100    // -- Then --
101    assertEquals(expectedCount, count);
102 }
103 }

```



## Übung 3

### 1.7.20 FHProvidedTest.java

Folgend ist die Implementierung der Testklasse für die Tests, welche von der FH zur Verfügung gestellt wurden angeführt. Um diese Tests unabhängig von den anderen Tests zu halten, wurden diese in dieser Testklasse gekapselt, obwohl diese die Namenskonvention bricht.

Listing 33: FHProvidedTest.java

```

1 package at.fh.ooe.swe.test.puzzle.model.searchNode;
2
3 import static org.junit.Assert.*;
4
5 import org.junit.Before;
6 import org.junit.Test;
7 import org.junit.runner.RunWith;
8 import org.junit.runners.JUnit4;
9
10 import at.fh.ooe.swe.test.api.AbstractTest;
11 import at.fh.ooe.swe4.puzzle.api.Board;
12 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
13 import at.fh.ooe.swe4.puzzle.model.SearchNode;
14
15 /**
16  * This test class contains the provided test which have been modified to fit
17  * actual implementation but the semantics has been kept.
18  *
19  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
20  * @date May 1, 2015
21  */
22 @RunWith(JUnit4.class)
23 public class FHProvidedTest extends AbstractTest {
24
25     private Board<Integer> goal;
26
27     @Before
28     public void init() {
29         goal = new BoardListImpl<>(3);
30         goal.setTile(1, 1, 1);
31         goal.setTile(1, 2, 2);
32         goal.setTile(1, 3, 3);
33         goal.setTile(2, 1, 4);
34         goal.setTile(2, 2, 5);
35         goal.setTile(2, 3, 6);
36         goal.setTile(3, 1, 7);
37         goal.setTile(3, 2, 8);
38         goal.setTile(3, 3, null);
39     }
40
41     @Test
42     public void test1() {
43         // -- Given --
44         final Board<Integer> initial = new BoardListImpl<>(3);
45         initial.setTile(1, 1, 1);
46         initial.setTile(1, 2, 2);
47         initial.setTile(1, 3, 3);
48         initial.setTile(2, 1, 4);
49         initial.setTile(2, 2, 5);
50         initial.setTile(2, 3, 6);
51         initial.setTile(3, 1, 7);
52         initial.setTile(3, 2, 8);
53         initial.setTile(3, 3, null);
54         SearchNode<Integer> node = new SearchNode<>(0, null, initial, goal, null);
55     }

```

# Übung 3

```

56     // -- When --
57     final int costs = node.estimatedCostsToTarget();
58
59     // -- Then --
60     assertEquals(0, costs);
61 }
62
63 @Test
64 public void test2() {
65     // -- Given --
66     final Board<Integer> initial = new BoardListImpl<>(3);
67     initial.setTile(1, 1, 1);
68     initial.setTile(1, 2, 2);
69     initial.setTile(1, 3, 3);
70     initial.setTile(2, 1, 4);
71     initial.setTile(2, 2, null);
72     initial.setTile(2, 3, 6);
73     initial.setTile(3, 1, 7);
74     initial.setTile(3, 2, 8);
75     initial.setTile(3, 3, 5);
76     SearchNode<Integer> node = new SearchNode<>(0, null, initial, goal, null);
77
78     // -- When --
79     final int costs = node.estimatedCostsToTarget();
80
81     // -- Then --
82     assertEquals(2, costs);
83 }
84
85 @Test
86 public void test3() {
87     // -- Given --
88     final Board<Integer> initial = new BoardListImpl<>(3);
89     initial.setTile(1, 1, 1);
90     initial.setTile(1, 2, null);
91     initial.setTile(1, 3, 3);
92     initial.setTile(2, 1, 4);
93     initial.setTile(2, 2, 5);
94     initial.setTile(2, 3, 6);
95     initial.setTile(3, 1, 7);
96     initial.setTile(3, 2, 8);
97     initial.setTile(3, 3, 2);
98     final SearchNode<Integer> node = new SearchNode<>(0, null, initial, goal, null);
99
100    // -- When --
101    final int costs = node.estimatedCostsToTarget();
102
103    // -- Then --
104    assertEquals(3, costs);
105 }
106 }

```

## Übung 3

### 1.7.21 SolveIntegerTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `solver.solve(Board<T> gaol)` angeführt, welche von der FH zur Verfügung gestellt wurden. Diese Tests testen den A\*-Algorithmus mit Integer Kacheln. Sie wurden angepasst um mit der Implementierten API arbeiten zu können, sind aber semantisch gleich den zu den Verfügung gestellten Tests.

Listing 34: SolveIntegerTest.java

```

1 package at.fh.ooe.swe.test.puzzle.impl.slidingPuzzle;
2
3 import static org.junit.Assert.assertEquals;
4
5 import java.util.ArrayList;
6 import java.util.List;
7
8 import org.junit.Before;
9 import org.junit.Test;
10 import org.junit.runner.RunWith;
11 import org.junit.runners.JUnit4;
12
13 import at.fh.ooe.swe.test.api.AbstractTest;
14 import at.fh.ooe.swe4.puzzle.api.Board;
15 import at.fh.ooe.swe4.puzzle.api.Board.Move;
16 import at.fh.ooe.swe4.puzzle.exception.NoSolutionException;
17 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
18 import at.fh.ooe.swe4.puzzle.impl.SlidingPuzzle;
19
20 /**
21  * This test class contains the provided tests for testing the method
22  * {@link SlidingPuzzle#solve(at.fh.ooe.swe4.puzzle.api.Board)}.<br>
23  * Therefore that theses test resolving enough no further tests this method need
24  * to implemented.
25  *
26  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
27  * @date May 1, 2015
28  */
29 @RunWith(JUnit4.class)
30 public class SolveIntegerTest extends AbstractTest {
31
32     private Board<Integer> goal3;
33     private Board<Integer> goal4;
34
35     @Before
36     public void init() {
37         // 3 x 3 board
38         final List<Integer> container3 = createContainer((int) Math.pow(3, 2));
39         container3.set(container3.size() - 1, null);
40         goal3 = new BoardListImpl<>(3, container3);
41
42         // 4 x 4 board
43         final List<Integer> container4 = createContainer((int) Math.pow(4, 2));
44         container4.set(container4.size() - 1, null);
45         goal4 = new BoardListImpl<>(4, container4);
46     }
47
48     @Test
49     public void solveSimplePuzzleTest1() throws NoSolutionException {
50         // -- Given --
51         final int size = 3;
52         final Board<Integer> initial = new BoardListImpl<>(size);
53         initial.setTile(1, 1, 1);

```

# Übung 3

```

54     initial.setTile(1, 2, 2);
55     initial.setTile(1, 3, 3);
56     initial.setTile(2, 1, 4);
57     initial.setTile(2, 2, 5);
58     initial.setTile(2, 3, 6);
59     initial.setTile(3, 1, 7);
60     initial.setTile(3, 2, null);
61     initial.setTile(3, 3, 8);
62     SlidingPuzzle<Integer> solver = new SlidingPuzzle<>();
63
64     // -- When --
65     final List<Move> moves = new ArrayList<>();
66     solver.start(initial)
67         .solve(goal3)
68         .start()
69         .fillMoves(moves)
70         .printMoves()
71         .performMoves()
72         .end()
73         .end();
74
75     // -- Then --
76     assertEquals(1, moves.size());
77     assertEquals(goal3, initial);
78 }
79
80 @Test
81 public void solveSimplePuzzleTest2() throws NoSolutionException {
82     // -- Given --
83     final int size = 3;
84     final Board<Integer> initial = new BoardListImpl<>(size);
85     initial.setTile(1, 1, 1);
86     initial.setTile(1, 2, 2);
87     initial.setTile(1, 3, 3);
88     initial.setTile(2, 1, 4);
89     initial.setTile(2, 2, 5);
90     initial.setTile(2, 3, 6);
91     initial.setTile(3, 1, null);
92     initial.setTile(3, 2, 7);
93     initial.setTile(3, 3, 8);
94     SlidingPuzzle<Integer> solver = new SlidingPuzzle<>();
95
96     // -- When --
97     final List<Move> moves = new ArrayList<>();
98     solver.start(initial)
99         .solve(goal3)
100        .start()
101        .fillMoves(moves)
102        .printMoves()
103        .performMoves()
104        .end()
105        .end();
106
107     // -- Then --
108     assertEquals(2, moves.size());
109     assertEquals(goal3, initial);
110 }
111
112 @Test
113 public void solveComplexPuzzleTest1() throws NoSolutionException {
114     // -- Given --
115     final int size = 3;
116     // 8 2 7

```

## Übung 3

```

117 // 1 4 6
118 // 3 5 X
119 final Board<Integer> initial = new BoardListImpl<>(size);
120 initial.setTile(1, 1, 8);
121 initial.setTile(1, 2, 2);
122 initial.setTile(1, 3, 7);
123 initial.setTile(2, 1, 1);
124 initial.setTile(2, 2, 4);
125 initial.setTile(2, 3, 6);
126 initial.setTile(3, 1, 3);
127 initial.setTile(3, 2, 5);
128 initial.setTile(3, 3, null);
129 SlidingPuzzle<Integer> solver = new SlidingPuzzle<>();
130
131 // -- When --
132 solver.start(initial)
133     .solve(goal3)
134     .start()
135     .printMoves()
136     .performMoves()
137     .end()
138     .end();
139
140 // -- Then --
141 assertEquals(goal3, initial);
142 }
143
144 @Test
145 public void solveRandomPuzzlesTest2() throws NoSolutionException {
146     // -- Given --
147     final int size = 3;
148     final List<Integer> container = createContainer((int) Math.pow(size, 2));
149     container.set(container.size() - 1, null);
150     final Board<Integer> initial = new BoardListImpl<>(size, container);
151     final SlidingPuzzle<Integer> solver = new SlidingPuzzle<Integer>();
152     for (int k = 0; k < 50; k++) {
153         initial.shuffle();
154
155         // -- When --
156         final List<Move> moves = new ArrayList<>();
157         solver.start(initial)
158             .solve(goal3)
159             .start()
160             .fillMoves(moves)
161             .printMoves()
162             .performMoves()
163             .end()
164             .end();
165
166         // -- Then --
167         assertEquals(goal3, initial);
168     }
169 }
170
171 @Test
172 public void solveSimplePuzzleTest_4x4() throws NoSolutionException {
173     // -- Given --
174     final Board<Integer> initial = goal4.clone();
175     final SlidingPuzzle<Integer> solver = new SlidingPuzzle<Integer>();
176
177     // -- When --
178     initial.moveLeft();
179     final List<Move> moves = new ArrayList<>();

```

# Übung 3

```

180     solver.start(initial)
181         .solve(goal4)
182         .start()
183         .fillMoves(moves)
184         .printMoves()
185         .performMoves()
186         .end()
187         .end();
188
189     // -- Then --
190     assertEquals(1, moves.size());
191     assertEquals(goal4, initial);
192 }
193
194 @Test
195 public void solveComplexPuzzleTest_4x4() throws NoSolutionException {
196     // -- Given --
197     final Board<Integer> initial = goal4.clone();
198     final SlidingPuzzle<Integer> solver = new SlidingPuzzle<>();
199
200     // -- When --
201     initial.moveLeft();
202     initial.moveLeft();
203     initial.moveUp();
204     initial.moveLeft();
205     initial.moveUp();
206     initial.moveUp();
207     initial.moveRight();
208     initial.moveDown();
209     initial.moveLeft();
210
211     final List<Move> moves = new ArrayList<>();
212     solver.start(initial)
213         .solve(goal4)
214         .start()
215         .fillMoves(moves)
216         .printMoves()
217         .performMoves()
218         .end()
219         .end();
220
221     // -- Then --
222     assertEquals(9, moves.size());
223     assertEquals(goal4, initial);
224 }
225 }

```

## Übung 3

### 1.7.22 SolveCharacterTest.java

Folgend ist die Implementierung der Testklasse für die Tests der Methode `solver.solve(Board<T> goal)` angeführt, welche von der FH zur Verfügung gestellt wurden. Diese Tests testen den A\*-Algorithmus mit `Character` Kacheln. Diese Tests sind eine Kopie der Integer Tests, nur mit dem `Character` Datentyp für die Kacheln. Diese Tests beweisen dass es keine Rolle spielt, welchen Datentyp man verwendet unter der Voraussetzung dass diese das Interface `Comparable<T>` korrekt implementieren.

Listing 35: SolveCharacterTest.java

```

1 package at.fh.ooe.swe.test.puzzle.impl.slidingPuzzle;
2
3 import static org.junit.Assert.assertEquals;
4
5 import java.util.ArrayList;
6 import java.util.List;
7 import java.util.function.IntConsumer;
8 import java.util.stream.IntStream;
9
10 import org.junit.Before;
11 import org.junit.Test;
12 import org.junit.runner.RunWith;
13 import org.junit.runners.JUnit4;
14
15 import at.fh.ooe.swe.test.api.AbstractTest;
16 import at.fh.ooe.swe4.puzzle.api.Board;
17 import at.fh.ooe.swe4.puzzle.api.Board.Move;
18 import at.fh.ooe.swe4.puzzle.exception.NoSolutionException;
19 import at.fh.ooe.swe4.puzzle.impl.BoardListImpl;
20 import at.fh.ooe.swe4.puzzle.impl.SlidingPuzzle;
21
22 /**
23  * This test class contains the provided tests for testing the method
24  * {@link SlidingPuzzle#solve(at.fh.ooe.swe4.puzzle.api.Board)}.<br>
25  * Therefore that these tests resolving enough no further tests this method need
26  * to implemented.
27  *
28  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
29  * @date May 1, 2015
30  */
31 @RunWith(JUnit4.class)
32 public class SolveCharacterTest extends AbstractTest {
33
34     private Board<Character> goal3;
35     private Board<Character> goal4;
36
37     @Before
38     public void init() {
39         // 3 x 3 board
40         final List<Character> container3 = createContainerWithChars((int) Math.pow(3, 2));
41         container3.set(container3.size() - 1, null);
42         goal3 = new BoardListImpl<>(3, container3);
43
44         // 4 x 4 board
45         final List<Character> container4 = createContainerWithChars((int) Math.pow(4, 2));
46         container4.set(container4.size() - 1, null);
47         goal4 = new BoardListImpl<>(4, container4);
48     }
49
50     @Test
51     public void solveSimplePuzzleTest1() throws NoSolutionException {

```

## Übung 3

```

52 // -- Given --
53 final int size = 3;
54 final Board<Character> initial = new BoardListImpl<>(size);
55 initial.setTile(1, 1, 'a');
56 initial.setTile(1, 2, 'b');
57 initial.setTile(1, 3, 'c');
58 initial.setTile(2, 1, 'd');
59 initial.setTile(2, 2, 'e');
60 initial.setTile(2, 3, 'f');
61 initial.setTile(3, 1, 'g');
62 initial.setTile(3, 2, null);
63 initial.setTile(3, 3, 'h');
64 SlidingPuzzle<Character> solver = new SlidingPuzzle<>();
65
66 // -- When --
67 final List<Move> moves = new ArrayList<>();
68 solver.start(initial)
69     .solve(goal3)
70     .start()
71     .fillMoves(moves)
72     .printMoves()
73     .performMoves()
74     .end()
75     .end();
76
77 // -- Then --
78 assertEquals(1, moves.size());
79 assertEquals(goal3, initial);
80 }
81
82 @Test
83 public void solveSimplePuzzleTest2() throws NoSolutionException {
84 // -- Given --
85 final int size = 3;
86 final Board<Character> initial = new BoardListImpl<>(size);
87 initial.setTile(1, 1, 'a');
88 initial.setTile(1, 2, 'b');
89 initial.setTile(1, 3, 'c');
90 initial.setTile(2, 1, 'd');
91 initial.setTile(2, 2, 'e');
92 initial.setTile(2, 3, 'f');
93 initial.setTile(3, 1, null);
94 initial.setTile(3, 2, 'g');
95 initial.setTile(3, 3, 'h');
96 SlidingPuzzle<Character> solver = new SlidingPuzzle<>();
97
98 // -- When --
99 final List<Move> moves = new ArrayList<>();
100 solver.start(initial)
101     .solve(goal3)
102     .start()
103     .fillMoves(moves)
104     .printMoves()
105     .performMoves()
106     .end()
107     .end();
108
109 // -- Then --
110 assertEquals(2, moves.size());
111 assertEquals(goal3, initial);
112 }
113
114 @Test

```



## Übung 3

```

115 public void solveComplexPuzzleTest1() throws NoSolutionException {
116     // -- Given --
117     final int size = 3;
118     // 8 2 7
119     // 1 4 6
120     // 3 5 X
121     final Board<Character> initial = new BoardListImpl<>(size);
122     initial.setTile(1, 1, 'h');
123     initial.setTile(1, 2, 'b');
124     initial.setTile(1, 3, 'g');
125     initial.setTile(2, 1, 'a');
126     initial.setTile(2, 2, 'd');
127     initial.setTile(2, 3, 'f');
128     initial.setTile(3, 1, 'c');
129     initial.setTile(3, 2, 'e');
130     initial.setTile(3, 3, null);
131     SlidingPuzzle<Character> solver = new SlidingPuzzle<>();
132
133     // -- When --
134     solver.start(initial)
135         .solve(goal3)
136         .start()
137         .printMoves()
138         .performMoves()
139         .end()
140         .end();
141
142     // -- Then --
143     assertEquals(goal3, initial);
144 }
145
146 @Test
147 public void solveRandomPuzzlesTest2() throws NoSolutionException {
148     // -- Given --
149     final int size = 3;
150     final List<Character> container = createContainerWithChars((int) Math.pow(size, 2));
151     container.set(container.size() - 1, null);
152     final Board<Character> initial = new BoardListImpl<>(size, container);
153     final SlidingPuzzle<Character> solver = new SlidingPuzzle<>();
154     for (int k = 0; k < 50; k++) {
155         initial.shuffle();
156
157         // -- When --
158         final List<Move> moves = new ArrayList<>();
159         solver.start(initial)
160             .solve(goal3)
161             .start()
162             .fillMoves(moves)
163             .printMoves()
164             .performMoves()
165             .end()
166             .end();
167
168         // -- Then --
169         assertEquals(goal3, initial);
170     }
171 }
172
173 @Test
174 public void solveSimplePuzzleTest_4x4() throws NoSolutionException {
175     // -- Given --
176     final Board<Character> initial = goal4.clone();
177     final SlidingPuzzle<Character> solver = new SlidingPuzzle<>();

```

## Übung 3

```
178
179 // -- When --
180 initial.moveLeft();
181 final List<Move> moves = new ArrayList<>();
182 solver.start(initial)
183     .solve(goal4)
184     .start()
185     .fillMoves(moves)
186     .printMoves()
187     .performMoves()
188     .end()
189     .end();
190
191 // -- Then --
192 assertEquals(1, moves.size());
193 assertEquals(goal4, initial);
194 }
195
196 @Test
197 public void solveComplexPuzzleTest_4x4() throws NoSolutionException {
198     // -- Given --
199     final Board<Character> initial = goal4.clone();
200     final SlidingPuzzle<Character> solver = new SlidingPuzzle<>();
201
202     // -- When --
203     initial.moveLeft();
204     initial.moveLeft();
205     initial.moveUp();
206     initial.moveLeft();
207     initial.moveUp();
208     initial.moveUp();
209     initial.moveRight();
210     initial.moveDown();
211     initial.moveLeft();
212
213     final List<Move> moves = new ArrayList<>();
214     solver.start(initial)
215         .solve(goal4)
216         .start()
217         .fillMoves(moves)
218         .printMoves()
219         .performMoves()
220         .end()
221         .end();
222
223     // -- Then --
224     assertEquals(9, moves.size());
225     assertEquals(goal4, initial);
226 }
227
228 /**
229  * Creates a container with the given size and sets char values c in order
230  * from ordinal(c) to ordinal(size).<br>
231  * <b>This container will contain no null element.<b>
232  *
233  * @param size
234  *         the size of the container
235  * @return the created container
236  */
237 protected List<Character> createContainerWithChars(final int size) {
238     final List<Character> container = new ArrayList<>(size);
239     IntStream.range(0, size)
240         .forEachOrdered(new IntConsumer() {
```

## Übung 3

```
241         @Override
242         public void accept(int value) {
243             container.add(new Character((char) (97 + value)));
244         }
245     });
246
247     return container;
248 }
249 }
```

## Übung 3

## 1.8 Tests

Folgend sind die Tests der N-Puzzle Implementierungen angeführt.

Da der Testoutput extrem lang ist und das Dokument ebenso, werden hier nur die wichtigsten Tests angeführt. Die Tests können jederzeit im Eclipse über den beschriebenen maven-build command ausgeführt und reproduziert werden. Ebenso können die Tests im Eclipse Klasse für Klasse ausgeführt werden.

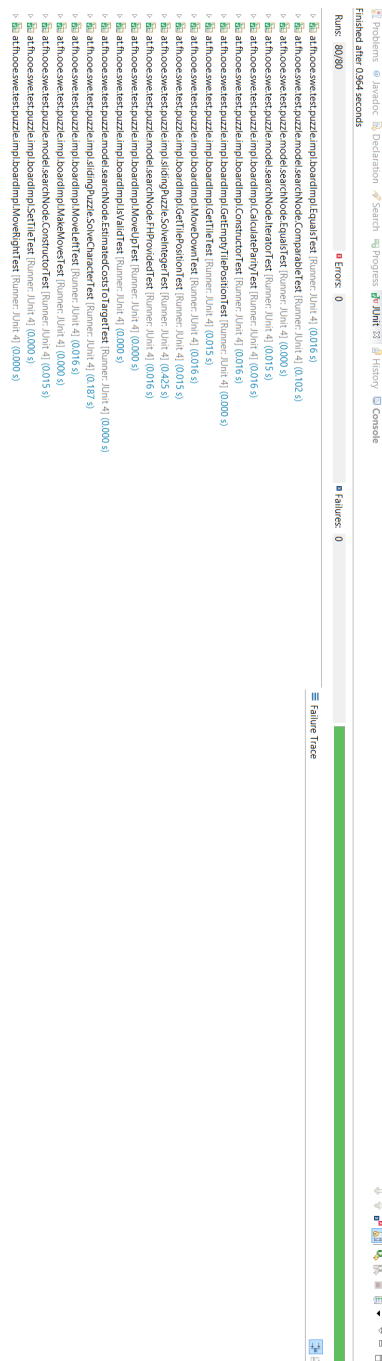


Abbildung 1: Diese Abbildung zeigt die JUnit Testresultate im Eclipse

## Übung 3

```

INFO 21:27:09,820 - started: at.fh.ooe.swe.test.puzzle.impl.slidingPuzzle.SolveIntegerTest
INFO 21:27:09,820 - #####
INFO 21:27:09,820 - -----
INFO 21:27:09,820 - started: solveComplexPuzzleTest_4x4
INFO 21:27:09,883 - Resulting moves:
INFO 21:27:09,898 - 1: RIGHT
INFO 21:27:09,898 - 2: UP
INFO 21:27:09,898 - 3: LEFT
INFO 21:27:09,898 - 4: DOWN
INFO 21:27:09,898 - 5: DOWN
INFO 21:27:09,898 - 6: RIGHT
INFO 21:27:09,898 - 7: DOWN
INFO 21:27:09,898 - 8: RIGHT
INFO 21:27:09,898 - 9: RIGHT
INFO 21:27:09,898 - Initial state:
INFO 21:27:09,898 - [ 2 ] [ 6 ] [ 3 ] [ 4 ]
INFO 21:27:09,898 - [   ] [ 1 ] [ 7 ] [ 8 ]
INFO 21:27:09,898 - [ 5 ] [ 9 ] [11] [12]
INFO 21:27:09,898 - [13] [10] [14] [15]
INFO 21:27:09,898 - Initial state after movements:
INFO 21:27:09,898 - [ 1 ] [ 2 ] [ 3 ] [ 4 ]
INFO 21:27:09,898 - [ 5 ] [ 6 ] [ 7 ] [ 8 ]
INFO 21:27:09,898 - [ 9 ] [10] [11] [12]
INFO 21:27:09,898 - [13] [14] [15] [   ]
INFO 21:27:09,898 - succeeded: solveComplexPuzzleTest_4x4
INFO 21:27:09,898 - -----
INFO 21:27:09,898 - started: solveSimplePuzzleTest1
INFO 21:27:09,914 - Resulting moves:
INFO 21:27:09,914 - 1: RIGHT
INFO 21:27:09,914 - Initial state:
INFO 21:27:09,914 - [1] [2] [3]
INFO 21:27:09,914 - [4] [5] [6]
INFO 21:27:09,914 - [7] [ ] [8]
INFO 21:27:09,914 - Initial state after movements:
INFO 21:27:09,914 - [1] [2] [3]
INFO 21:27:09,914 - [4] [5] [6]
INFO 21:27:09,914 - [7] [8] [ ]
INFO 21:27:09,914 - succeeded: solveSimplePuzzleTest1
INFO 21:27:09,914 - -----
INFO 21:27:09,914 - started: solveSimplePuzzleTest2
INFO 21:27:09,914 - Resulting moves:
INFO 21:27:09,914 - 1: RIGHT
INFO 21:27:09,914 - 2: RIGHT
INFO 21:27:09,914 - Initial state:
INFO 21:27:09,914 - [1] [2] [3]
INFO 21:27:09,914 - [4] [5] [6]
INFO 21:27:09,914 - [ ] [7] [8]
INFO 21:27:09,914 - Initial state after movements:
INFO 21:27:09,914 - [1] [2] [3]
INFO 21:27:09,914 - [4] [5] [6]
INFO 21:27:09,914 - [7] [8] [ ]
INFO 21:27:09,914 - succeeded: solveSimplePuzzleTest2
INFO 21:27:09,914 - -----

```

Abbildung 2: Diese Abbildung zeigt die Resultate zur Verfügung gestellten Tests mit dem *Integer* Datentyp

# Übung 3

```

INFO 21:27:10,226 - started: solveComplexPuzzleTest1
INFO 21:27:10,273 - Resulting moves:
INFO 21:27:10,273 - 1 : UP
INFO 21:27:10,273 - 2 : UP
INFO 21:27:10,273 - 3 : LEFT
INFO 21:27:10,273 - 4 : DOWN
INFO 21:27:10,273 - 5 : LEFT
INFO 21:27:10,273 - 6 : UP
INFO 21:27:10,273 - 7 : RIGHT
INFO 21:27:10,273 - 8 : DOWN
INFO 21:27:10,273 - 9 : RIGHT
INFO 21:27:10,273 - 10 : DOWN
INFO 21:27:10,273 - 11 : LEFT
INFO 21:27:10,273 - 12 : LEFT
INFO 21:27:10,273 - 13 : UP
INFO 21:27:10,273 - 14 : RIGHT
INFO 21:27:10,273 - 15 : DOWN
INFO 21:27:10,273 - 16 : RIGHT
INFO 21:27:10,273 - 17 : UP
INFO 21:27:10,273 - 18 : LEFT
INFO 21:27:10,273 - 19 : DOWN
INFO 21:27:10,273 - 20 : LEFT
INFO 21:27:10,273 - 21 : UP
INFO 21:27:10,273 - 22 : UP
INFO 21:27:10,289 - 23 : RIGHT
INFO 21:27:10,289 - 24 : RIGHT
INFO 21:27:10,289 - 25 : DOWN
INFO 21:27:10,289 - 26 : DOWN
INFO 21:27:10,289 - Initial state:
INFO 21:27:10,289 - [8] [2] [7]
INFO 21:27:10,289 - [1] [4] [6]
INFO 21:27:10,289 - [3] [5] [ ]
INFO 21:27:10,289 - Initial state after movements:
INFO 21:27:10,289 - [1] [2] [3]
INFO 21:27:10,289 - [4] [5] [6]
INFO 21:27:10,289 - [7] [8] [ ]
INFO 21:27:10,289 - succeeded: solveComplexPuzzleTest1
INFO 21:27:10,289 - -----
INFO 21:27:10,289 - started: solveSimplePuzzleTest_4x4
INFO 21:27:10,289 - Resulting moves:
INFO 21:27:10,289 - 1: RIGHT
INFO 21:27:10,289 - Initial state:
INFO 21:27:10,289 - [1] [2] [3] [4]
INFO 21:27:10,289 - [5] [6] [7] [8]
INFO 21:27:10,289 - [9] [10] [11] [12]
INFO 21:27:10,289 - [13] [14] [ ] [15]
INFO 21:27:10,289 - Initial state after movements:
INFO 21:27:10,289 - [1] [2] [3] [4]
INFO 21:27:10,289 - [5] [6] [7] [8]
INFO 21:27:10,289 - [9] [10] [11] [12]
INFO 21:27:10,289 - [13] [14] [15] [ ]
INFO 21:27:10,289 - succeeded: solveSimplePuzzleTest_4x4
INFO 21:27:10,289 - -----
INFO 21:27:10,289 - #####
INFO 21:27:10,289 - succeeded: at.fh.ooe.swe.test.puzzle.impl.slidingPuzzle.SolveIntegerTest
INFO 21:27:10,289 - #####

```

Abbildung 3: Diese Abbildung zeigt die Resultate zur Verfügung gestellten Tests mit dem *Integer* Datentyp

# Übung 3

```

INFO 21:29:01,847 - #####
INFO 21:29:01,847 - started: at.fh.ooe.swe.test.puzzle.impl.slidingPuzzle.SolveCharacterTest
INFO 21:29:01,847 - #####
INFO 21:29:01,863 - -----
INFO 21:29:01,863 - started: solveComplexPuzzleTest_4x4
INFO 21:29:01,957 - Resulting moves:
INFO 21:29:01,957 - 1: RIGHT
INFO 21:29:01,957 - 2: UP
INFO 21:29:01,957 - 3: LEFT
INFO 21:29:01,957 - 4: DOWN
INFO 21:29:01,957 - 5: DOWN
INFO 21:29:01,957 - 6: RIGHT
INFO 21:29:01,957 - 7: DOWN
INFO 21:29:01,957 - 8: RIGHT
INFO 21:29:01,957 - 9: RIGHT
INFO 21:29:01,957 - Initial state:
INFO 21:29:01,972 - [b] [f] [c] [d]
INFO 21:29:01,972 - [ ] [a] [g] [h]
INFO 21:29:01,972 - [e] [i] [k] [l]
INFO 21:29:01,972 - [m] [j] [n] [o]
INFO 21:29:01,972 - Initial state after movements:
INFO 21:29:01,972 - [a] [b] [c] [d]
INFO 21:29:01,972 - [e] [f] [g] [h]
INFO 21:29:01,972 - [i] [j] [k] [l]
INFO 21:29:01,972 - [m] [n] [o] [ ]
INFO 21:29:01,972 - succeeded: solveComplexPuzzleTest_4x4
INFO 21:29:01,972 - -----
INFO 21:29:01,972 - started: solveSimplePuzzleTest1
INFO 21:29:01,972 - Resulting moves:
INFO 21:29:01,972 - 1: RIGHT
INFO 21:29:01,972 - Initial state:
INFO 21:29:01,972 - [a] [b] [c]
INFO 21:29:01,972 - [d] [e] [f]
INFO 21:29:01,972 - [g] [ ] [h]
INFO 21:29:01,972 - Initial state after movements:
INFO 21:29:01,972 - [a] [b] [c]
INFO 21:29:01,972 - [d] [e] [f]
INFO 21:29:01,972 - [g] [h] [ ]
INFO 21:29:01,972 - succeeded: solveSimplePuzzleTest1
INFO 21:29:01,972 - -----
INFO 21:29:01,972 - started: solveSimplePuzzleTest2
INFO 21:29:01,972 - Resulting moves:
INFO 21:29:01,972 - 1: RIGHT
INFO 21:29:01,972 - 2: RIGHT
INFO 21:29:01,972 - Initial state:
INFO 21:29:01,972 - [a] [b] [c]
INFO 21:29:01,972 - [d] [e] [f]
INFO 21:29:01,972 - [ ] [g] [h]
INFO 21:29:01,972 - Initial state after movements:
INFO 21:29:01,972 - [a] [b] [c]
INFO 21:29:01,972 - [d] [e] [f]
INFO 21:29:01,972 - [g] [h] [ ]
INFO 21:29:01,972 - succeeded: solveSimplePuzzleTest2
INFO 21:29:01,972 - -----

```

Abbildung 4: Diese Abbildung zeigt die Resultate zur Verfügung gestellten Tests mit dem *Character* Datentyp

# Übung 3

```

INFO 21:29:02,254 - started: solveComplexPuzzleTest1
INFO 21:29:02,285 - Resulting moves:
INFO 21:29:02,285 - 1 : UP
INFO 21:29:02,285 - 2 : UP
INFO 21:29:02,285 - 3 : LEFT
INFO 21:29:02,285 - 4 : DOWN
INFO 21:29:02,285 - 5 : LEFT
INFO 21:29:02,285 - 6 : UP
INFO 21:29:02,285 - 7 : RIGHT
INFO 21:29:02,285 - 8 : DOWN
INFO 21:29:02,285 - 9 : RIGHT
INFO 21:29:02,285 - 10 : DOWN
INFO 21:29:02,285 - 11 : LEFT
INFO 21:29:02,285 - 12 : LEFT
INFO 21:29:02,285 - 13 : UP
INFO 21:29:02,285 - 14 : RIGHT
INFO 21:29:02,285 - 15 : DOWN
INFO 21:29:02,285 - 16 : RIGHT
INFO 21:29:02,285 - 17 : UP
INFO 21:29:02,285 - 18 : LEFT
INFO 21:29:02,285 - 19 : DOWN
INFO 21:29:02,285 - 20 : LEFT
INFO 21:29:02,285 - 21 : UP
INFO 21:29:02,285 - 22 : UP
INFO 21:29:02,285 - 23 : RIGHT
INFO 21:29:02,285 - 24 : RIGHT
INFO 21:29:02,285 - 25 : DOWN
INFO 21:29:02,285 - 26 : DOWN
INFO 21:29:02,285 - Initial state:
INFO 21:29:02,285 - [h] [b] [g]
INFO 21:29:02,285 - [a] [d] [f]
INFO 21:29:02,285 - [c] [e] [ ]
INFO 21:29:02,285 - Initial state after movements:
INFO 21:29:02,285 - [a] [b] [c]
INFO 21:29:02,285 - [d] [e] [f]
INFO 21:29:02,285 - [g] [h] [ ]
INFO 21:29:02,285 - succeeded: solveComplexPuzzleTest1
INFO 21:29:02,285 -----
INFO 21:29:02,285 - started: solveSimplePuzzleTest_4x4
INFO 21:29:02,285 - Resulting moves:
INFO 21:29:02,285 - 1: RIGHT
INFO 21:29:02,285 - Initial state:
INFO 21:29:02,285 - [a] [b] [c] [d]
INFO 21:29:02,285 - [e] [f] [g] [h]
INFO 21:29:02,285 - [i] [j] [k] [l]
INFO 21:29:02,285 - [m] [n] [ ] [o]
INFO 21:29:02,285 - Initial state after movements:
INFO 21:29:02,285 - [a] [b] [c] [d]
INFO 21:29:02,285 - [e] [f] [g] [h]
INFO 21:29:02,285 - [i] [j] [k] [l]
INFO 21:29:02,285 - [m] [n] [o] [ ]
INFO 21:29:02,285 - succeeded: solveSimplePuzzleTest_4x4
INFO 21:29:02,285 -----
INFO 21:29:02,285 - #####
INFO 21:29:02,285 - succeeded: at.fh.ooe.swe.test.puzzle.impl.slidingPuzzle.SolveCharacterTest
INFO 21:29:02,285 - #####

```

Abbildung 5: Diese Abbildung zeigt die Resultate zur Verfügung gestellten Tests mit dem *Character* Datentyp



## Übung 3

```

INFO 21:30:09,833 - #####
INFO 21:30:09,833 - started: at.fh.ooe.swe.test.puzzle.model.searchNode.FHProvidedTest
INFO 21:30:09,833 - #####
INFO 21:30:09,833 - -----
INFO 21:30:09,833 - started: test1
INFO 21:30:09,895 - succeeded: test1
INFO 21:30:09,895 - -----
INFO 21:30:09,895 - started: test2
INFO 21:30:09,895 - succeeded: test2
INFO 21:30:09,895 - -----
INFO 21:30:09,895 - started: test3
INFO 21:30:09,895 - succeeded: test3
INFO 21:30:09,895 - -----
INFO 21:30:09,895 - #####
INFO 21:30:09,911 - succeeded: at.fh.ooe.swe.test.puzzle.model.searchNode.FHProvidedTest
INFO 21:30:09,911 - #####

```

Abbildung 6: Diese Abbildung zeigt die Resultate zur Verfügung gestellten Tests für die `SearchNode<T>` angeführt