Aufgabenblock 1

TECHNISCHE UNIVERSITÄT DARMSTADT

Softwarepraktikum, WS 17/18
Abgabetermin: 02.11.2017 23:59 Uhr
XML, Generics, Testen, Vererbung

Hinweis

Wir messen der Einhaltung der Grundregeln der wissenschaftlichen Ethik größten Wert bei. Mit der Abgabe einer Lösung bestätigen Sie, dass Sie/Ihre Gruppe der alleinige Autor/die alleinigen Autoren des gesamten Materials sind. Falls Ihnen die Verwendung von Fremdmaterial gestattet war, so müssen Sie dessen Quellen deutlich zitiert haben. Weiterführende Informationen finden Sie unter http://www.es.tu-darmstadt.de/lehre/plagiatshinweise.html.
Weiterhin dürfen **keine Klassen unterhalb** von org.sopra.internal.* referenziert werden, dies führt zu einer Bewertung mit **0 Punkten** für die betreffende Aufgabe. Jede Abgabe muss alle notwendigen Dateien zum Kompilieren enthalten. Das in der jeweiligen Teilaufgabe geforderte Schema der Bezeichnung der Dateien ist einzuhalten.
Zusätzlich soll in jeder Klasse das Interface org.sopra.api.exercises.ExerciseSubmission implementiert werden und die in der Dokumentation beschriebene Funktionalität besitzen.

Lerninhalte

- Kennenlernen der Spielanleitung und der wichtigsten Spielelemente.
- Umgang mit XML durch Erstellen einer Szenario-Datei üben.
- Kennenlernen des Frameworks durch das Erstellen nützlicher Hilfsmethoden für den weiteren Verlauf des Praktikums.
- Umgang mit der Collection-API von Java üben.
- Umgang mit JUnit durch Festlegen der Testkriterien und Testen der Hilfsmethoden üben.
- Umgang mit Vererbung am Beispiel eines Producers üben.

Aufgabe 1.0 - Entwicklungsumgebung einrichten

Verwenden Sie die unter Moodle bereitgestellte Anleitung zur Einrichtung der Entwicklungsumgebung. Lesen Sie sich außerdem die unter Moodle bereitgestellte Spielanleitung aufmerksam durch.

Aufgabe 1.1 - Szenario erstellen (9 Punkte)

In dieser Aufgabe sollen Sie ein Inselszenario erstellen, um die Spielanleitung und die wichtigsten Spielelemente kennenzulernen. Alle für ein Szenario relevanten Eigenschaften werden in der strukturierten Auszeichnungssprache *XML* in einer entsprechenden Datei beschrieben. Das Beispielszenario `scenario_bsp.xml` dient als Orientierungshilfe und enthält alle zulässigen Elemente und Attribute.
Erstellen Sie mit einem Texteditor wie z.B. Notepad++ oder einem Tabellenkalkulationstool eine Datei `scenario1.xml`. Die Datei soll folgende Knoten enthalten:

ID	ProducerType	Position
0	CoalFiredPowerPlant	(x,y) = (1,6)
1	GasFiredPowerplant	(5,4)
2	GasFiredPowerplant	(3,5)
3	HydroPowerPlant	(9,6)
4	NuclearPowerPlant	(3,3)
5	SolarPowerPlant	(2,8)
6	SolarPowerPlant	(3,8)
7	WindPowerPlant	(1,0)
8	WindPowerPlant	(2,0)
9	WindPowerPlant	(5,0)

ID	ConsumerType	Position	Name
10	City	(6,1)	Darmstadt
11	City	(0,2)	Offenbach
12	City	(8,2)	Mainz
13	City	(9,8)	Wiesbaden
14	IndustrialPark	(4,1)	Apel Automotive
15	IndustrialPark	(5,7)	WASF
16	IndustrialPark	(5,8)	Buyer Medical Care
17	CommercialPark	(5,6)	Hedge Services

Die Verbindungen (Kanten) zwischen den Knoten sind wie folgt gegeben:

ID1	ID2	Verbindungstyp
11	7	`LowVoltage`
7	8	`LowVoltage`
8	4	`LowVoltage`
4	14	`MediumVoltage`
14	9	`MediumVoltage`
9	10	`MediumVoltage`
10	1	`MediumVoltage`
1	12	`HighVoltage`
12	3	`HighVoltage`
3	13	`HighVoltage`
13	16	`HighVoltage`
16	15	`HighVoltage`
15	6	`HighVoltage`
5	6	`HighVoltage`
15	17	`HighVoltage`
17	0	`HighVoltage`
17	2	`HighVoltage`
17	4	`HighVoltage`

Zusätzlich soll das Szenario die in Abbildung 1 gezeigten Spielfeldelemente in der dargestellten Anordnung enthalten.

Hinweis: Prüfen Sie die Korrektheit der erstellten XML-Datei, indem Sie die erstellte Datei über die GUI (`Load Scenario`) oder über die API (`org.sopra.api.util.scenarioloader.ScenarioLoader`) laden. Speichern Sie dieses Szenario unter dem Namen `scenario1.xml` im Paket `solutions.exercise1`.

Weiterführende Informationen: XML

Aufgabe 1.2 - Hilfsmethoden implementieren (7 Punkte)

In dieser Aufgabe sollen Sie Hilfsmethoden implementieren, die im weiteren Verlauf des Praktikums nützlich sind und Sie mit der Collection-API vertraut machen.

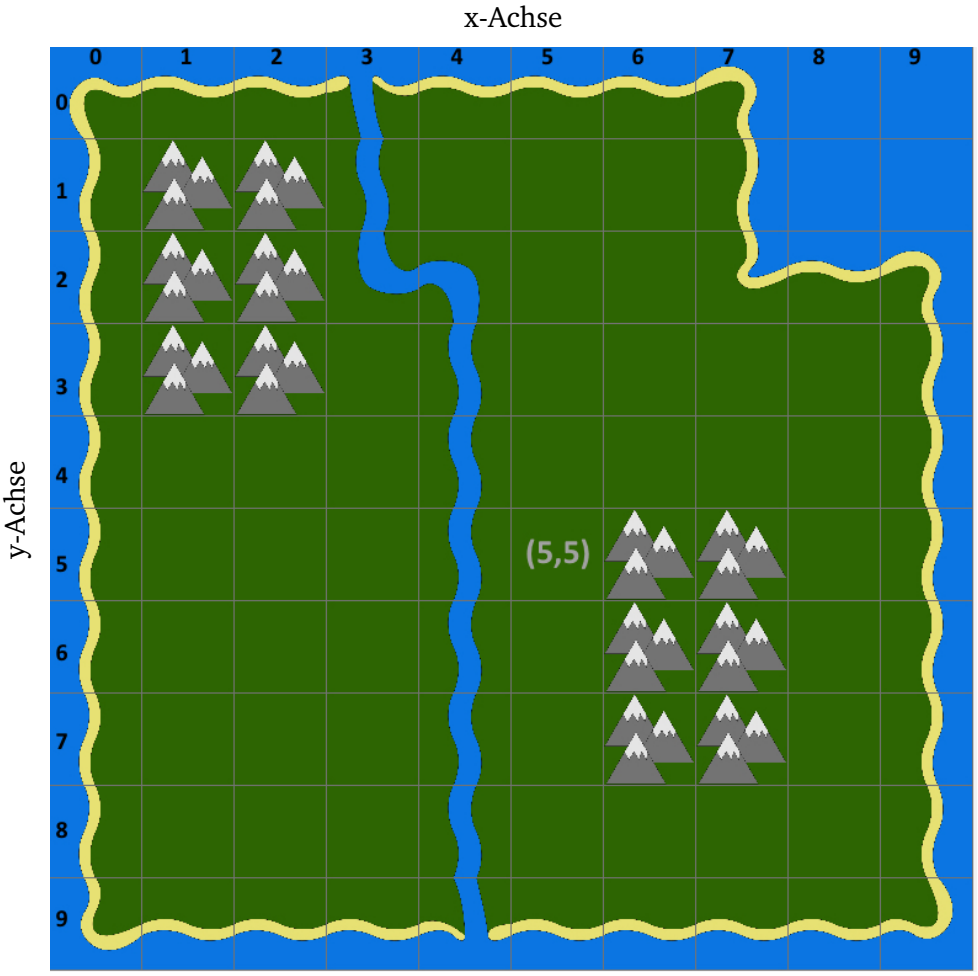

Abbildung 1: Spielfeld für `scenario1.xml`.

- Untersuchen Sie zuerst die in der Klasse `ScenarioUtilImpl` implementierte Methode, welche eine Implementierung der Methode `List<PlayfieldElement> getPlayfieldElementsByType(Scenario scenario, ElementType type)` mit einigen Fehlern darstellt und korrigieren Sie diese. Verwenden Sie die korrigierte Implementierung als Beispiel um die folgenden Methoden zu implementieren.
- Implementieren Sie die Methoden der in Moodle zur Verfügung gestellten Klasse `ScenarioUtilImpl`, die das Interface `ScenarioUtil` implementiert.

  Für jede Methode soll auf die Übergabe von `null` geprüft werden. Falls `null` übergeben wird, soll eine neue `RuntimeException` des Typs `java.lang.IllegalArgumentException` geworfen werden. Ergänzen Sie im Quellcode in Form von *vollständiger* Javadoc-Dokumentation knapp in *eigenen Worten* die Funktionalität der Klasse sowie jeder implementierten Methode.

  Beispiel für die Verwendung von Generics mit Listen:

```
List<SolarPowerPlant> plants = new ArrayList<SolarPowerPlant>();
// Ensures that only elements of the type SolarPowerPlant are included in the list
```

  Implementieren Sie folgende (nicht statische) Methoden:

a) `List<PowerLine> getPowerLinesByType(Graph<EnergyNode, PowerLine> graph, PowerLineType type)`: Gibt für einen gegebenen Graphen alle `PowerLine`-Objekte des gegebenen Typs in einer Liste zurück. Falls keine passenden Objekte vorhanden sind, soll eine leere Liste zurückgeliefert werden.

b) `List<ControllableProducer> getControllableProducers(Graph<EnergyNode, PowerLine> graph)`: Gibt für einen gegebenen Graphen alle Energieknoten, die eine Instanz vom Typ `ControllableProducer` sind, in einer Liste zurück. Falls keine passenden Objekte vorhanden sind, soll eine leere Liste zurückgeliefert werden.

c) `List<ControllableConsumer> getControllableConsumers(Graph<EnergyNode, PowerLine> graph)`: Gibt für einen gegebenen Graphen alle Energieknoten, die eine Instanz vom Typ `ControllableConsumer` sind, in einer Liste zurück. Falls keine passenden Objekte vorhanden sind, soll eine leere Liste zurückgeliefert werden.

d) `List<Producer> getProducers(Graph<EnergyNode, PowerLine> graph)`: Gibt für einen gegebenen Graphen alle Energieknoten, die eine Instanz vom Typ `Producer` sind, in einer Liste zurück. Falls keine passenden Objekte vorhanden sind, soll eine leere Liste zurückgeliefert werden.

e) `List<Consumer> getConsumers(Graph<EnergyNode, PowerLine> graph)`: Gibt für einen gegebenen Graphen alle Energieknoten, die eine Instanz vom Typ `Consumer` sind, in einer Liste zurück. Falls keine passenden Objekte vorhanden sind, soll eine leere Liste zurückgeliefert werden.

Weiterführende Informationen: Generics, Collection-API

Aufgabe 1.3 - Testen der Szenario Hilfsmethoden (10 Punkte)

In dieser Aufgabe sollen Sie eine Implementierung der Hilfsmethoden auf ihre Funktionsfähigkeit testen.

- Untersuchen Sie zuerst die in der Klasse `ScenarioUtilTest` bereitgestellte Implementierung der Methoden `testGetPowerLinesByType()` und `testGetPowerLinesByType_Parameters()` auf Fehler und korrigieren Sie diese. Verwenden Sie die korrigierte Implementierung als Beispiel um die folgenden Methoden zu implementieren.

- Ergänzen Sie dazu die in der bereitgestellten Klasse `ScenarioUtilTest` vorbereiteten Testfälle.
  Nutzen Sie das vererbte Attribut `scenario1` des Typs `Scenario`, um auf die bereits eingelesene `szenario1.xml`-Datei aus Aufgabe 1.2 zuzugreifen. Verwenden Sie das eingelesene Szenario als Testorakel, um erwartete Rückgabewerte der zu testenden Methoden zu ermitteln. Die zu testende Implementierung ist über das Attribut `sut` (*System Under Test*) des Typs `ScenarioUtil` bereitgestellt und wird vor jedem Testfallaufruf neu initialisiert.
  Speichern Sie die Klasse im Paket `solutions.exercise1` ab.

  Für jede Methode aus `ScenarioUtil` soll folgendes Verhalten getestet werden:

  - Soll-Verhalten bei korrektem Aufruf mit dem Szenario aus der Datei `szenario1.xml` und
  - Soll-Verhalten bei Aufruf mit fehlerhaften Übergabeparametern.

Hinweis: Beim Testen des Soll-Verhaltens bei korrektem Aufruf ist es ausreichend, zu überprüfen, ob die Länge der Liste den Erwartungen entspricht.

Beispiele für das Testen auf das Auftreten einer Exception mit Hilfe einer *Annotation*:

```
@Override
@Test(expected = IllegalArgumentException.class)
// the test succeeds if an Exception of the expected type is raised
public void testGetSolarPowerPlants_Parameters() throws Exception {
    sut.getSolarPowerPlants(null);
}
```

oder einem *try/catch - Block*:

```
@Override
@Test
public void testGetSolarPowerPlants_Parameters() throws Exception {
    try {
        sut.getSolarPowerPlants(null);
        fail(); // if the expected Exception is raised this part of the code is never reached
    } catch (IllegalArgumentException e) {
        // Test succeeded: null parameter value raised expected Exception
    }
}
```

Weiterführende Informationen: JUnit, Annotationen

Aufgabe 1.4 - Erstellen einer EnergyNode-Hierarchie (9 Punkte)

In dieser Aufgabe sollen Sie eine Vererbungshierarchie am Beispiel eines Producers erstellen.
Hinweis: Die Interfaces in dieser Aufgabe stimmen **nicht** mit den Interfaces `EnergyNode` und `Producer` überein, die Ihnen im weiteren Verlauf des Praktikums noch begegnen.

a) Erstellen Sie die abstrakte Klasse `SimpleEnergyNodeImpl`, die eine EnergyNode darstellt. Speichern Sie die Klasse im Paket `solutions.exercise1` ab.

   - Die Klasse soll die Interfaces `SimpleEnergyNode` und `ExerciseSubmission` implementieren und die in den Javadocs beschriebene Funktionalität besitzen.
   - Die Klasse besitzt die privaten und finalen Attribute `x` (x-Koordinate als ganze Zahl) und `y` (y-Koordinate als ganze Zahl). Das Attribut `energyLevel` (EnergyLevel als ganze Zahl) soll nur von erbenden Klassen zugreifbar sein. Der als protected gekennzeichnete Konstruktor nimmt zwei ganze Zahlen entgegen, um die Koordinaten zu initialisieren. Das `energyLevel` wird mit 0 initialisiert.

b) Erstellen sie die öffentliche Klasse `SimpleSolarPowerPlantImpl`, die von `SimpleEnergyNodeImpl` erbt und den Producer SolarPowerPlant darstellt. Speichern Sie die Klasse im Paket `solutions.exercise1` ab.

   - Die Klasse soll die Interfaces `SimpleProducer` und `ExerciseSubmission` implementieren und die in den Javadocs beschriebene Funktionalität besitzen.
   - Die Klasse besitzt keine zusätzlichen Attribute und ihr öffentlicher Konstruktor ruft den Konstruktor der Superklasse auf.

Hinweis: Ergänzen Sie im Quellcode in Form von *vollständiger* Javadoc-Dokumentation knapp in *eigenen Worten* die Funktionalität der Klasse sowie jeder implementierten Methode.

Weiterführende Informationen: Vererbung, Sichtbarkeiten

Kritik, Verbesserungsvorschläge und Bug-Report

Sollten Sie Kritik oder Verbesserungsvorschläge haben bzw. Bugs finden, dann nutzen Sie dafür bitte den Bug-Report Button im Moodle-Kurs.