4. Programmieraufgabe

Objektorientierte Programmiertechniken

LVA-Nr. 185.A01
2017/2018 W
TU Wien

Themen:

Untertypbeziehungen, Zusicherungen

Ausgabe:

08. 11. 2017

Abgabe (Deadline):

22. 11. 2017, 12:00 Uhr

Abgabeverzeichnis:

`Aufgabe4`
nicht mehr `Aufgabe1-3`

Programmaufruf:

`java Test`

Grundlage:

Skriptum, Schwerpunkt auf Kapitel 2

Kontext

Im Bereich der erneuerbaren Energie werden zahlreiche Kleinanlagen angeboten, die im Hinblick auf Investitionskosten, laufende Kosten, Energieertrag und einige weitere Aspekte verglichen werden sollen. Folgendes Interface zur Beschreibung einer Anlage ist vorgegeben:

```
public interface Unit {
    int investmentCosts();  // in Euro
    int runningCosts();     // average in Euro per year
    int energyOutput();     // average in kWh per year
    int energyInput();      // average in kWh per year
}
```

Folgende Untertypen davon sind vorgesehen (alphabetisch sortiert):

`CogenerationUnit:` Eine Anlage, die bedarfsorientiert sowohl Elektrizität als auch für Raumheizung und Warmwasser nutzbare Wärme aus konventionellen Energieformen (auch Biogas, Pflanzenöl, Holz) gewinnt. Die Methode `energyOutput` gibt die durchschnittlich pro Jahr gelieferte Energie (Elektrizität und Wärme) zurück, aber nicht ungenutzte Abwärme. Das Ergebnis von `energyInput` enthält die gesamte durchschnittlich pro Jahr benötigte Energie.

`ElectricPowerUnit:` Eine Anlage, die elektrische Energie liefert. Diese Energie wird aus diversen Energieformen gewonnen (z. B. auch aus chemischen Prozessen zur Energiespeicherung in Batterien). Die Methode `energyOutput` gibt die durchschnittlich pro Jahr gelieferte elektrische Energie zurück. Das Ergebnis von `energyInput` berücksichtigt die durchschnittlich pro Jahr benötigte Energie in handelbaren Energieformen wie Elektrizität und verschiedenen Brennstoffen, jedoch nicht frei verfügbare Energieformen wie Licht, Wind und Umgebungswärme. Zusätzlich gibt es eine Methode namens `quality`, die im zurückgegebenen `int`-Wert die Qualität der gelieferten elektrischen Energie beschreibt: Ein Wert größer 0 besagt, dass die Anlage bei Bedarf zuverlässig über eine bestimmte Zeit eine bestimmte Leistung liefern kann – je höher, desto mehr. Bei einem Wert kleiner oder gleich 0 wird keine Mindestleistung zugesichert. Ein Wert kleiner 0 besagt, dass ein nachgeschalteter Abnehmer in der Lage sein muss, gelieferte Energie auch dann abzunehmen, wenn sie nicht benötigt wird – je kleiner, desto mehr Energie.

`EnergyGenerator:` Eine Anlage, die Energie in einer nutzbaren Form liefert. Diese Energie wird (bis auf die für den Betrieb der Anlage benötigte elektrische Energie) aus frei verfügbaren Energieformen gewonnen. Das Ergebnis von `energyOutput` beziffert die durchschnittlich pro Jahr gelieferte Energie ohne die beim Betrieb entstehende Abwärme. Das Ergebnis von `energyInput` enthält die zum Betrieb durchschnittlich pro Jahr benötigte elektrische Energie.

`HeatingSystem:` Eine Anlage, die sich ganzjährig und bedarfsorientiert zur Erzeugung von Wärme eignet (zur Warmwasserbereitung und Raumheizung). Das Ergebnis von `energyOutput` beziffert die durchschnittlich pro Jahr gelieferte nutzbare Wärmeenergie (ohne die beim Betrieb der Anlage entstehende Abwärme). Das Ergebnis von `energyInput` berücksichtigt wie für `ElectricPowerUnit` nur handelbare Energieformen, keine frei verfügbaren.

`HeatPump:` Mit Hilfe elektrischer Energie wird der Umgebung (Erde, Wasser oder Luft) jederzeit bei Bedarf nach dem Kühlschrankprinzip Wärme entzogen und zur Warmwasserbereitung und Raumheizung verwendet. Die Methode `energyOutput` gibt die durchschnittlich pro Jahr gelieferte nutzbare Wärmeenergie zurück (ohne Abwärme). Das Ergebnis von `energyInput` enthält die zum Betrieb der Anlage durchschnittlich pro Jahr benötigte elektrische Energie.

`PhotovoltaicSystem:` Solarzellen wandeln Licht direkt in Elektrizität um. Zur Anpassung der Energie an das Leitungsnetz enthält die Anlage einen Spannungswandler. In manche Anlagen ist ein Stromspeicher integriert. Die Methode `energyOutput` gibt die durchschnittlich pro Jahr gelieferte elektrische Energie zurück. Das Ergebnis von `energyInput` enthält, wie für `HeatPump`, die zum Betrieb der Anlage durchschnittlich pro Jahr benötigte elektrische Energie.

`SolarThermalSystem:` Direkte Sonnenstrahlung erwärmt Wasser. Solche Anlagen werden zur Warmwasserbereitung und teilweise als Zusatzheizung in der wärmeren Jahreszeit eingesetzt, sind im Winter jedoch nicht betriebstauglich. Die Methoden `energyOutput` und `energyInput` sind wie für `HeatPump` definiert.

`WindTurbine:` Wind treibt einen Generator zur Erzeugung elektrischer Energie an. Ein Spannungswandler und eventuell auch ein Stromspeicher kann integriert sein. Die Methoden `energyOutput` und `energyInput` sind wie für `PhotovoltaicSystem` definiert.

Die Beschreibungen dieser Typen enthalten nur unbedingt nötige Methoden. Zusätzliche Methoden können vorhanden sein.

Welche Aufgabe zu lösen ist

Schreiben Sie ein Programm mit (abstrakten) Klassen und Interfaces für alle unter *Kontext* angeführten Typen. Versehen Sie alle Typen mit den notwendigen Zusicherungen und stellen Sie sicher, dass Sie nur dort eine Vererbungsbeziehung (`extends` oder `implements`) verwenden, wo eine Untertypbeziehung auch hinsichtlich der Zusicherungen besteht. Ermöglichen Sie Untertypbeziehungen zwischen *allen* diesen Typen, außer wenn sie den Beschreibungen der Typen widersprechen würden.

alle Zusicherungen und Untertypbeziehungen

Besteht zwischen zwei Typen keine Untertypbeziehung, geben Sie in einem Kommentar in `Test.java` eine Begründung dafür an. Bitte geben Sie eine textuelle Begründung, auskommentierten Programmteile reichen nicht. Das Fehlen einer Methode in einer Typbeschreibung ist als Begründung ungeeignet, weil zusätzliche Methoden hinzugefügt werden dürfen.

Begründung wenn keine Untertypbeziehung

Sie können zusätzliche (abstrakte) Klassen und Interfaces einführen. Die Typstruktur soll trotzdem einfach und klein bleiben, wobei jedoch alle oben genannten Typen (mit den vorgegebenen Namen) vorkommen müssen, auch solche, die Sie vielleicht für nicht nötig erachten.

gegebene Typen mit gegebenen Namen

Schreiben Sie eine Klasse `Test` zum Testen Ihrer Lösung. Das Programm muss vom Abgabeverzeichnis (`Aufgabe4`) aus durch `java Test` ausführbar sein. Überprüfen Sie mittels Testfällen, ob dort, wo Sie eine Untertypbeziehung annehmen, Ersetzbarkeit gegeben ist.

Testklasse

Daneben soll die Datei `Test.java` als Kommentar eine kurze, aber verständliche Beschreibung der Aufteilung der Arbeiten auf die einzelnen Gruppenmitglieder enthalten – wer hat was gemacht.

Aufgabenaufteilung beschreiben

Was im Hinblick auf die Beurteilung wichtig ist

Die insgesamt 100 für diese Aufgabe erreichbaren Punkte sind folgendermaßen auf die zu erreichenden Ziele aufgeteilt:

- Untertypbeziehungen richtig erkannt und eingesetzt — 40 Punkte
- nicht bestehende Untertypbeziehungen gut begründet — 10 Punkte
- Zusicherungen konsistent und zweckentsprechend — 25 Punkte
- Lösung so getestet, dass Fehler aufgedeckt würden — 10 Punkte
- Lösung vollständig (entsprechend Aufgabenstellung) — 15 Punkte

Schwerpunkte berücksichtigen

Obwohl für das Testen der Lösung nur 10 Punkte veranschlagt sind, kann unzureichendes Testen doch zu größerem Punkteverlust führen, wenn dadurch bestehende Mängel nicht rechtzeitig erkannt werden. Auch Mängel in der Funktionalität können einen Verlust von deutlich mehr als 15 Punkten bedeuten, weil diese wahrscheinlich auch Fehler in Untertypbeziehungen und Zusicherungen nach sich ziehen.

Die größte Schwierigkeit liegt darin, alle Untertypbeziehungen zu finden und Ersetzbarkeit sicherzustellen. Vererbungsbeziehungen, die keine Untertypbeziehungen sind, führen zu sehr hohem Punkteverlust. Ebenso gibt es hohe Punkteabzüge für nicht wahrgenommene Gelegenheiten, Untertypbeziehungen zwischen vorgegebenen Typen herzustellen, sowie für fehlende oder falsche Begründungen für nicht bestehende Untertypbeziehungen. Geeignete Begründungen wären Gegenbeispiele.

alle Untertypbeziehungen

Eine Grundlage für das Auffinden der Untertypbeziehungen sind gute Zusicherungen. Wesentliche Zusicherungen kommen in obigen Beschreibungen vor. Untertypbeziehungen ergeben sich aus erlaubten Beziehungen zwischen Zusicherungen in Unter- und Obertypen. Es hat sich als günstig erwiesen, alle Zusicherungen, die in einem Obertyp gelten, im Untertyp nochmals hinzuschreiben, da sie sonst leicht übersehen werden. Nicht der Umfang der Kommentare ist entscheidend, sondern deren Qualität (Vollständigkeit, Aussagekraft, Unmissverständlichkeit, ...).

Testen Sie die Untertypbeziehungen. Es kommt nicht auf die Anzahl der Testfälle an, sondern auf deren Qualität: Sie sollen Verletzungen der Ersetzbarkeit aufdecken können. Gegenbeispiele stellen sicher, dass keine Untertypbeziehungen übersehen wurden.

Zusicherungen in Testklassen werden aus praktischen Überlegungen bei der Beurteilung nicht berücksichtigt.

Zur Lösung dieser Aufgabe müssen Sie Untertypbeziehungen und den Einfluss von Zusicherungen auf Untertypbeziehungen im Detail verstehen. Holen Sie sich entsprechende Informationen aus Kapitel 2 des Skriptums. Folgende zusätzlichen Informationen könnten hilfreich sein:

- Konstruktoren werden in einer konkreten Klasse aufgerufen und sind daher vom Ersetzbarkeitsprinzip nicht betroffen.
- Zur Lösung der Aufgabe sind keine Exceptions nötig. Wenn Sie dennoch Exceptions verwenden, darf eine Instanz eines Untertyps nur eine Exception werfen, wenn man auch von einer Instanz eines Obertyps in derselben Situation diese Exception erwarten würde.
- Mehrfachvererbung gibt es nur auf Interfaces. Sollte einer der verlangten Typen mehrere Obertypen haben, müssen alle Obertypen bis auf einen Interfaces sein. Die Obertypen sollen auch in diesem Fall so heißen wie in der Aufgabenstellung.

Lassen Sie sich von der Form der Beschreibung nicht täuschen. Daraus, dass die Beschreibung eines Typs die eines anderen referenziert oder teilweise wiederholt, folgt noch keine Ersetzbarkeit. Sie sind auf dem falschen Weg, wenn es den Anschein hat, A könne Untertyp von B und B gleichzeitig Untertyp von A sein, außer wenn A und B gleich sind.

Achten Sie auf die Sichtbarkeit. Alle oben beschriebenen Typen und Methoden sollen überall verwendbar sein. Die Sichtbarkeit von Implementierungsdetails soll so stark wie möglich eingeschränkt werden. Sichtbare Implementierungsdetails beeinflussen die Ersetzbarkeit.

Sichtbarkeit

Was man generell beachten sollte (alle Aufgaben)

Es werden keinerlei Ausnahmen bezüglich des Abgabetermins gemacht. Beurteilt wird nur, was rechtzeitig im Abgabeverzeichnis im Repository steht. Alle `.java`-Dateien im Abgabeverzeichnis (einschließlich Unterverzeichnissen) müssen auf der `g0` gemeinsam übersetzbar sein. Auf der `g0` sind nur Standardbibliotheken installiert; daher dürfen keine anderen Bibliotheken verwendet werden, auch nicht `junit`. Die Übersetzung schlägt häufig auch dann fehl, wenn falsche `.java`-Dateien im Abgabeverzeichnis stehen, etwa als Backup gedachte Dateien.

Abgabetermin einhalten

auf g0 testen

Schreiben Sie nicht mehr als eine Klasse in jede Datei (ausgenommen geschachtelte Klassen), halten Sie sich an übliche Namenskonventionen (Großschreibung für Namen von Klassen und Interfaces, kleine Anfangsbuchstaben für Variablen und Methoden, etc.), und verwenden Sie die Namen, die in der Aufgabenstellung vorgegeben sind.

eine Klasse pro Datei

vorgegebene Namen

Warum die Aufgabe diese Form hat

Die Beschreibungen der Typen bieten nur wenig Interpretationsspielraum bezüglich der Ersetzbarkeit. Die Aufgabe ist so formuliert, dass Untertypbeziehungen (abgesehen von Typen, die Sie vielleicht zusätzlich einführen) eindeutig sind. Über Testfälle und Gegenbeispiele sollten Sie in der Lage sein, schwerwiegende Fehler selbst zu finden.

eindeutig