Programmieren in Java

Vorlesung 05: Testen und Pattern

Peter Thiemann

Albert-Ludwigs-Universität Freiburg, Germany

SS 2013

Inhalt

Testen

- Testen von abstrakten Klassen
- Testen mit Mock-Objekten

Pattern

- Template Method
- Rekursion
- Command Pattern
- Composite Pattern

Testen von abstrakten Klassen

- Problem: abstrakte Klassen können nicht instanziert werden
- Lösung: Verwende anonyme Klassen zum Testen
- Beispiel: Testen von `AProperty.buy()`

Testen von AProperty.buy()

```
@Test
public void testBuy() {
  int price = 100;
  AProperty prop = new AProperty("my property", price) {
    public IAction action(Player current, Collection<Player> others, IDice dice) {
      return null;
    }
    public boolean isMortgaged() { return false; }
    public int calculateRent(ReadDice dice) { return 0; }
  };
  Player p = new Player("my player");
  p.setCash(price);

  assertTrue(prop.buy(p));
  assertEquals("Owner must be p", p, prop.owner);
  assertTrue(prop.isOwned());
  assertFalse(prop.isMortgaged());
  assertTrue(p.ownsProperty(prop));
  assertEquals("Owner has not paid", 0, p.getCash());
}
```

Testen mit Mock-Objekten

- Problem: Die Miete von Utilities hängt von der zufällig gewürfelten Zahl ab.
- Lösung: Verwende anstelle des "echten" Würfels ein **Mock-Objekt**, das das gleiche Interface wie der Würfel implementiert, aber eine durch den Konstruktor festgelegte Zahl liefert.
- Allgemeines Problem: Die Anwendung hängt von (oft externen) Objekten ab, die noch nicht vorhanden sind, noch nicht implementiert sind, oder sich nicht-deterministisch verhalten.
- Lösung: Abstrahiere diese Komponenten durch Interfaces und teste sie mit deterministischen Mock-Objekten, die die gleichen Interfaces implementieren.

Der Mock-Würfel

```
public class TwoD6Mock implements IDice {
  private int w1, w2;

  /**
   * Create rolled pair of dice, projected into range 1-6
   * @param w1 first dice, between 1-6
   * @param w2 second dice, between 1-6
   */
  public TwoD6Mock(int w1, int w2) {
    this.w1 = adjust(w1);
    this.w2 = adjust(w2);
  }

  private int adjust(int w0) {
    int w = w0 % 6;
    return (w <= 0) ? (w + 6) : w;
  }
```

Der Mock-Würfel/2

```
  @Override
  public int getValue() {
    return this.w1 + this.w2;
  }

  @Override
  public boolean isDoubles() {
   return this.w1 == this.w2;
  }

  @Override
  public void roll() {
   // do nothing
   // a variation of the dice might prescribe a sequence of rolls
  }
}
```

Utility Test

```
@Test
public void test() {
  Player p = new Player("Test Player");
  Utility water = Utility.makeWater();
  Utility electric = Utility.makeElectric();
  IDice d = new TwoD6Mock(1, 1);

  assertEquals("Water: Not yet bought", 0, water.calculateRent(d));
  assertEquals("Electric: Not yet bought", 0, electric.calculateRent(d));
  assertTrue(water.buy(p));
  assertEquals("Water utility owned, rolled 1+1", 8, water.calculateRent(d));
  assertEquals("Electric: Not yet bought", 0, electric.calculateRent(d));
  assertTrue(electric.buy(p));
  assertEquals("Water: Both utilities owned, rolled 1+1", 20, water.calculateRent(d));
  assertEquals("Electric: Both utilities owned, rolled 1+1", 20, electric.calculateRent(d)
  d = new TwoD6Mock(5, 6);
  assertEquals("Water: Both utilities owned, rolled 5+6", 110, water.calculateRent(d));
  assertEquals("Electric: Both utilities owned, rolled 5+6", 110, electric.calculateRent(d
}
```

Pattern

- Pattern sind Muster für Standardlösungen von Entwurfsproblemen
- Bereits vorgekommen:
  - Singleton - von einer Klasse soll höchstens eine Instanz erzeugt werden.
  - Factory Method
    - Konstruktor einer Klasse soll nicht beliebig aufgerufen werden
    - Zu eine öffentlichen abstrakten Klasse gibt es unterschiedliche private Implementierungen, die austauschbar bleiben sollen und daher nicht im Programm auftauchen sollen.

Template Method

- Methode in einer abstrakten Klasse, die abstrakte Methoden der gleichen Klasse aufruft.
- Implementiert allgemeines Verfahren, das in Subklassen spezialisiert werden kann.
- Die abstrakten Methoden heissen auch **Hook-Methoden**.
- Beispiele: `IProperty.rentAmount()`, `IProperty.obtainMortgage()`

Rekursion

- Betrachte die Methode `turn()`, die einen Zug eines Spielers ausführt.
- Beim Erreichen eines Feldes kann eine Aktion ausgelöst werden, durch die der Spieler auf weitere Felder transportiert wird. Dadurch wird wieder eine Aktion ausgelöst.

Command Pattern

- Objekte dienen als Container für Methoden und ihre Argumente
- Beispiel: `IAction`, `IActionCard`
- Anwendung: Ereigniskarten; Aufräumaktionen zum Ende eines Spielzugs

Composite Pattern

- Beispiel: im Interface `IAction` wird eine Reihe von Aktionen zusammengefasst (Vereinigung von Klassen).
- Manche Aktionen betreffen mehrere Mitspieler und können daher nur teilweise gelingen.
- Modelliert durch Aktion mit Unterobjekten, die selbst wieder Aktionen sind.