SEW – S06

Mustergültige Zusammenarbeit

René Hollander, Paul Kalauner 4AHIT

11.12.2014

Inhaltsverzeichnis

Aufgabenstellung

Implementieren Sie die Quakologie in Java.
- Erkennen Sie die verwendeten Muster!
- Wann und wo wurden die Muster eingesetzt?
- Erkläre die verwendeten Muster (kleiner Tipp, es sind deren sechs!)

Geben Sie den entsprechenden Beispiel-Code (Java -> jar) und die gewünschte Dokumentation der Patterns (PDF) ab.

Es sind keine Test-Cases verlangt.

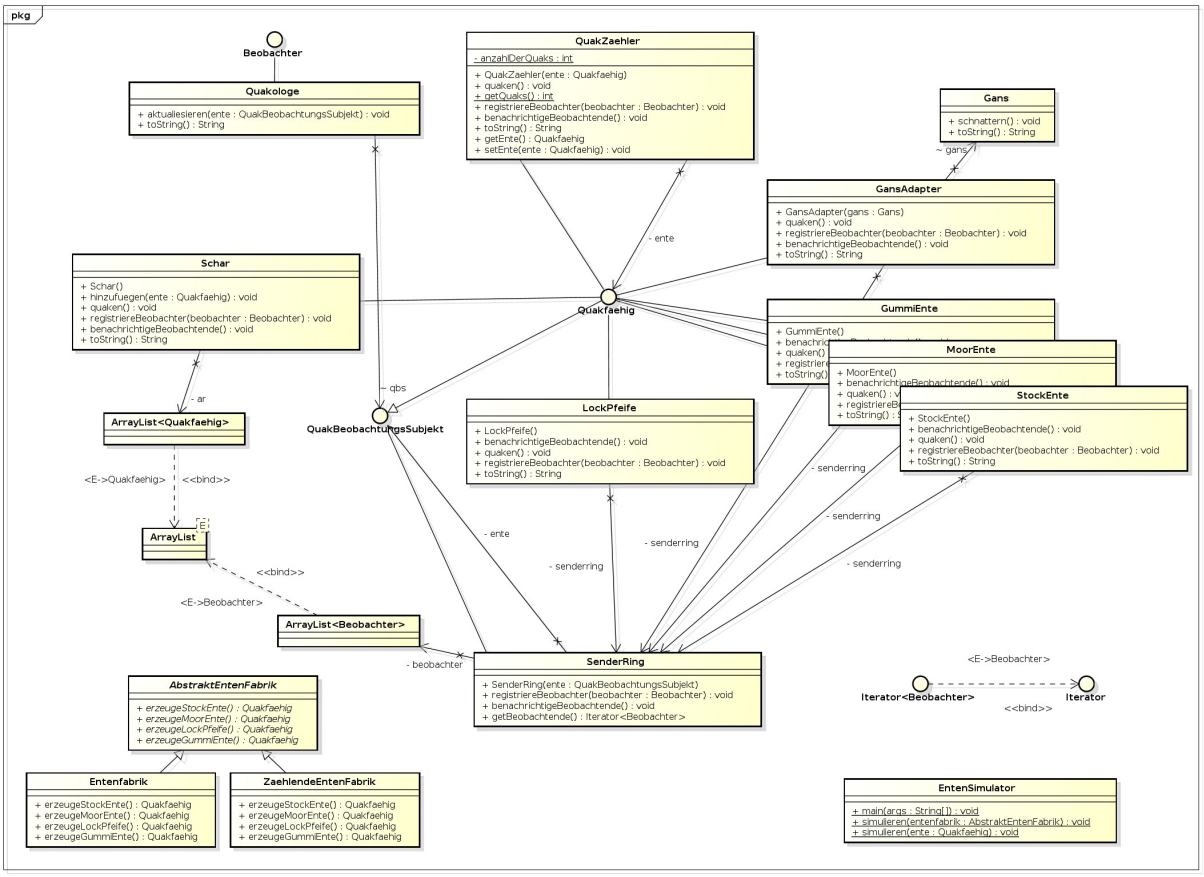

Abbildung 1, UML-Diagramm, Quelle: [1, Seite 5]

Implementierung

Die Implementierung haben wir von [2] übernommen.

Verwendete Patterns

- Observer Pattern
- Abstract Factory Pattern
- Adapter Pattern
- Iterator Pattern
- Strategy Pattern
- Composite Pattern

Observer Pattern

Das Observer Pattern ist für das Benachrichtigen des Quakologen (der einzige Beobachter) zuständig.

Code-Ausschnitt:

```
public class Quakologe implements Beobachter {

    public void aktualisieren(QuakBeobachtungsSubjekt ente) {
        System.out.println("Quakologe: " + ente + " hat gerade gequakt.");
    }

    public String toString() {
        return "Quakologe";
    }
}
```

```
public class SenderRing implements QuakBeobachtungsSubjekt {
    ArrayList<Beobachter> beobachtende = new ArrayList<Beobachter>();
    QuakBeobachtungsSubjekt ente;

    public SenderRing(QuakBeobachtungsSubjekt ente) {
        this.ente = ente;
    }

    public void registriereBeobachter(Beobachter beobachter) {
        beobachtende.add(beobachter);
    }

    public void benachrichtigeBeobachtende() {
        for (Beobachter beobachter : beobachtende) {
            beobachter.aktualisieren(ente);
        }
    }

    public Iterator<Beobachter> getBeobachtende() {
        return beobachtende.iterator();
    }
}
```

Abstract Factory Pattern

Für die Erzeugung der Enten wird eine Abstract Factory verwendet. Die Fabrik "EntenFabrik" erzeugt normale Enten, die "ZaehlendeEntenFabrik" erzeugt Enten die mitzählen wie oft sie quaken.

Code-Ausschnitt:

```
public abstract class AbstrakteEntenFabrik {

    public abstract Quakfaehig erzeugeStockEnte();

    public abstract Quakfaehig erzeugeMoorEnte();

    public abstract Quakfaehig erzeugeLockPfeife();

    public abstract Quakfaehig erzeugeGummiEnte();
}
```

```
public class Entenfabrik extends AbstrakteEntenFabrik {

    public Quakfaehig erzeugeStockEnte() {
        return new StockEnte();
    }

    public Quakfaehig erzeugeMoorEnte() {
        return new MoorEnte();
    }

    public Quakfaehig erzeugeLockPfeife() {
        return new LockPfeife();
    }

    public Quakfaehig erzeugeGummiEnte() {
        return new GummiEnte();
    }
}
```

```
public class ZaehlendeEntenFabrik extends AbstrakteEntenFabrik {

    public Quakfaehig erzeugeStockEnte() {
        return new QuakZaehler(new StockEnte());
    }

    public Quakfaehig erzeugeMoorEnte() {
        return new QuakZaehler(new MoorEnte());
    }

    public Quakfaehig erzeugeLockPfeife() {
        return new QuakZaehler(new LockPfeife());
    }

    public Quakfaehig erzeugeGummiEnte() {
        return new QuakZaehler(new GummiEnte());
    }
}
```

Adapter-Pattern

Das Adapter-Pattern wird eingesetzt damit die Gans als Ente ausgeben kann. Anderes ausgedrückt: Der Gansadapter, implementiert die `quaken()` Methode von `Quakfaehig` und ruft `gans.schnattern()` auf.

Code-Ausschnitt:

```
public class GansAdapter implements Quakfaehig {
    Gans gans;
    SenderRing senderRing;

    public GansAdapter(Gans gans) {
        this.gans = gans;
        senderRing = new SenderRing(this);
    }

    public void quaken() {
        gans.schnattern();
        benachrichtigeBeobachtende();
    }

    public void registriereBeobachter(Beobachter beobachter) {
        senderRing.registriereBeobachter(beobachter);
    }

    public void benachrichtigeBeobachtende() {
        senderRing.benachrichtigeBeobachtende();
    }

    public String toString() {
        return "sich als Ente ausgebende Gans";
    }
}
```

```
public class Gans {

    public void schnattern() {
        System.out.println("Schnatter");
    }

    public String toString() {
        return "Gans";
    }
}
```

Iterator-Pattern

Das Iterator-Pattern wird nur indirekt verwendet. Es handelt sich nämlich um die Java-Implementierung. Die Methode `getBeobachtende()` liefert den Iterator der `ArrayList beobachtende` zurück:

```
public Iterator<Beobachter> getBeobachtende() {
    return beobachtende.iterator();
}
```

Strategy-Pattern

Das Strategy-Pattern wird für den `QuakZaehler` verwendet. Die „Behaviours“ sind in diesem Fall die verschiedenen Enten. Der `QuackZaehler` delegiert dann die `quack()` Methode an die Enten und erhöht den QuakZaehler.

Code-Ausschnitt:

```
public class QuakZaehler implements Quakfaehig {
    Quakfaehig ente;
    static int anzahlDerQuaks;

    public QuakZaehler(Quakfaehig ente) {
        this.ente = ente;
    }

    public void quaken() {
        ente.quaken();
        anzahlDerQuaks++;
    }

    public static int getQuaks() {
        return anzahlDerQuaks;
    }

    public void registriereBeobachter(Beobachter beobachter) {
        ente.registriereBeobachter(beobachter);
    }

    public void benachrichtigeBeobachtende() {
        ente.benachrichtigeBeobachtende();
    }

    public String toString() {
        return ente.toString();
    }
}
```

Composite-Pattern

Das Composite Pattern wird bei der Klasse Schar verwendet. Die Klasse Schar, welche selber `Quakfaehig` implementiert, hat eine `ArrayList<Quackfaehig>`. Wenn nun die `quack()` Methode von `Schar` aufgerufen wird, wird von jedem `Quackfaehig` Objekt in der Liste die `quack()` Methode aufgerufen.

Code-Beispiel:

```
public class Schar implements Quakfaehig {
    ArrayList<Quakfaehig> quakende = new ArrayList<Quakfaehig>();

    public void hinzufuegen(Quakfaehig quaker) {
        quakende.add(quaker);
    }

    public void quaken() {
        for (Quakfaehig quaker : quakende) {
            quaker.quaken();
        }
    }

    public void registriereBeobachter(Beobachter beobachter) {
        for (Quakfaehig quaker : quakende) {
            quaker.registriereBeobachter(beobachter);
        }
    }

    public void benachrichtigeBeobachtende() {
    }

    public String toString() {
        return "Entenschar";
    }
}
```

Aufruf:

```
Schar entenSchar = new Schar();

entenSchar.hinzufuegen(moorEnte);
entenSchar.hinzufuegen(lockPfeife);
entenSchar.hinzufuegen(gummiEnte);
entenSchar.hinzufuegen(gansEnte);
simulieren(entenSchar);
```

```
void simulieren(Quakfaehig ente) {
    ente.quaken();
}
```

Quellen

[1]: PDF Mustergültige Zusammenarbeit. Abrufbar unter: https://elearning.tgm.ac.at/mod/resource/view.php?id=30831 [abgerufen am 11.12.2014]

[2]: tlins GitHub. Abrufbar unter: https://github.com/tlins/Quakologie/tree/master/quack_java [abgerufen am 11.12.2014]

Abbildungsverzeichnis