SWT 1: Entwurfsmuster by example

Adrian E. Lehmann

9. Juli 2017

Inhaltsverzeichnis

1 Varianten Muster

1.1 Strategie Entwurfsmuster

engl. Strategy pattern

Definition 1.1
Das Strategieentwurfsmuster beschreibt eine Familie von Algorithmen, verkapselt diese und macht sie auswechselbar. Damit können Algorithmen unabhängig von Klienten variiert werden.

Beispiel 1.1 (Java)
Das folgende Beispiel wird nun eine untereinander austauschbare Familie von Sortieralgorithmen darstellen

```
package net.adrianlehmann.swt_revision.patterns.
    variation_patterns;

import java.util.Collection;

/**
 * Created by adrianlehmann on 09.07.17.
 */
public interface Sorter {
   <T extends Comparable<T>> void sort(Collection<T> toBeSorted);
}
```

Code Snippet 1: Definiton der Strategie Schnittstelle

```
package net.adrianlehmann.swt_revision.patterns.
    variation_patterns;

import java.util.Collection;

/**
 * Created by adrianlehmann on 09.07.17.
 */
public class BubbleSort implements Sorter{

    @Override
    public <T extends Comparable<T>> void sort(Collection<T>
        toBeSorted) {
        //Bubble Sort
    }
}
```

Code Snippet 2: Erste Implementierung

```
package net.adrianlehmann.swt_revision.patterns.
    variation_patterns;

import java.util.Collection;

/**
 * Created by adrianlehmann on 09.07.17.
 */
public class QuickSort implements Sorter{

    @Override
    public <T extends Comparable<T>> void sort(Collection<T>
        toBeSorted) {
```

```
        //QuickSort
    }
}
```

Code Snippet 3: Zweite Implementierung

```
package net.adrianlehmann.swt_revision.patterns.
    variation_patterns;

import java.util.Arrays;
import java.util.List;

/**
 * Created by adrianlehmann on 09.07.17.
 */
public class Main {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Test","abc");
        Sorter s = new BubbleSort();
        s.sort(list);
        s = new QuickSort();
        s.sort(list);
    }
}
```

Code Snippet 4: Implementierung

Anwendung 1.1
SSwtich-less"programming. Strategies erlauben es uns ohne switch case anweisungen zwischen verschiedenen Anwendungsfällen Algorithmen zu wechseln.

1.2 Dekorierer

engl. Decorator pattern

Definition 1.2
Der Dekorierer fügt Objekten dynamisch Funktionalität hinzu.

Beispiel 1.2 (Java)
Im folgenden Beispiel werden wird Eis mit Extras "dekorieren".

```
package net.adrianlehmann.swt_revision.patterns.
    variation_patterns.decorator;

/**
 * Created by adrianlehmann on 09.07.17.
 */
public interface IceCream {

    double getCost();
    String getFlavor();

}
```

Code Snippet 5: Erstellen userer zu dekorierenden Schnittstelle Schnittstelle

```
package net.adrianlehmann.swt_revision.patterns.
    variation_patterns.decorator;

/**
```

```
 * Created by adrianlehmann on 09.07.17.
 */
public class GenericIceCream implements IceCream{
    @Override
    public double getCost() {
        return 0.7;
    }

    @Override
    public String getFlavor() {
        return "Vanilla";
    }
}
```

Code Snippet 6: Schnittstellen Implementierung

```
package net.adrianlehmann.swt_revision.patterns.
    variation_patterns.decorator;

/**
 * Created by adrianlehmann on 09.07.17.
 */
public abstract class IceCreamDecorator implements IceCream{
    protected final IceCream iceCream;

    public IceCreamDecorator(IceCream iceCream) {
        this.iceCream = iceCream;
    }

    @Override
    public double getCost() {
        return iceCream.getCost();
    }

    @Override
    public String getFlavor() {
        return iceCream.getFlavor();
    }
}
```

Code Snippet 7: Dekorierer

```
package net.adrianlehmann.swt_revision.patterns.
    variation_patterns.decorator;

/**
 * Created by unknown on 09.07.17.
 */
public class WithChocolateChips extends IceCreamDecorater {
    public WithChocolateChips(IceCream iceCream) {
        super(iceCream);
    }

    @Override
    public double getCost() {
        return super.getCost() + 0.5;
    }

    @Override
    public String getFlavor() {
        return super.getFlavor() + ", Chocolate";
    }
```

```
}
```

Code Snippet 8: Konkrete Implementierung des Dekorierers

```
package net.adrianlehmann.swt_revision.patterns.
    variation_patterns.decorator;

/**
 * Created by adrianlehmann on 09.07.17.
 */
public class WithCaramel extends IceCreamDecorater {
    public WithCaramel(IceCream iceCream) {
        super(iceCream);
    }

    @Override
    public double getCost() {
        return super.getCost() + 0.5;
    }

    @Override
    public String getFlavor() {
        return super.getFlavor() + ", Caramel";
    }
}
```

Code Snippet 9: Weitere konkrete Implementierung des Dekorierers

```
package net.adrianlehmann.swt_revision.patterns.
    variation_patterns.decorator;

/**
 * Created by adrianlehmann on 09.07.17.
 */
public class Main {
    public static void main(String[] args) {
        IceCream iceCream = new GenericIceCream();
        IceCream decoratedIceCream = new WithCaramel(iceCream);

        //Print costs
        System.out.println(iceCream.getCost()); // 0.7
        System.out.println(decoratedIceCream.getCost()); // 1.2


        //Print flavors
        System.out.println(iceCream.getFlavor()); //Vanilla
        System.out.println(decoratedIceCream.getFlavor()); //
            Vanilla, Caramel
    }
}
```

Code Snippet 10: Verwendung

Anwendung 1.2
Wenn Zusatzoperationen welche nur teilweise auftreten auf weitere Objekte delegiert werden sollen, während alte Objekte Fortbestand haben sollen.