Strategy Pattern

Zweck

«Identifiziere jene Aspekte, die sich ändern und trenne sie von jenen, die konstant bleiben.»

Das Ziel vom Strategy Pattern ist es den Code, der sich ändern kann vom Code der gleich bleibt zu trennen. Das heisst wenn man ein Objekt hat, wird alles was möglicherweise in Zukunft verändert wird, ausgelagert.

Beschreibung anhand eines Beispiels

Es besteht eine Klasse Hund von der die Klassen Husky und Pudel erben. Alle Hunde können bellen, deshalb wird diese Methode in der abstrakten Klasse Hund definiert. Da aber jede Hundeart anders bellt wird, die bellen Klasse in jeder erbenden Klasse (Pudel, Husky etc.) überschrieben.
Wenn jetzt eine weitere Hundeart Bulldogge hinzukommt, wird die bellen Methode wieder überschrieben, auch wenn das Bellen genau gleich ist wie beim Husky. So entsteht Code Redundanz.
Zudem ist es unmöglich ein Bellverhalten zu ändern während der Laufzeit. Genau hier setzt das Strategy Pattern an.

Nur was unverändert bleibt, bleibt auch in der Hundeklasse. Alles was möglicherweise verändert wird, wird in einer separaten Klasse definiert.

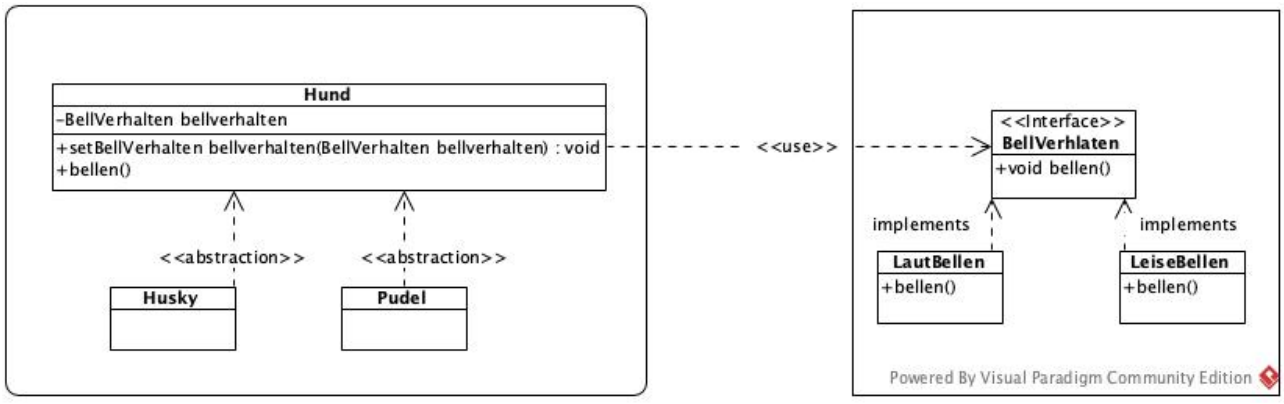
Damit das Bellverhalten auch während der Laufzeit geändert werden kann, wird eine Schnittstelle erstellt, also ein Interface mit der Funktion bellen().

Nun wird für jedes Bellverhalten eine eigene Klasse erstellt, zum Beispiel LautBellen und LeiseBellen, welche das Interface implementieren.

In der Hundeklasse kann nun einfach ein BellVerhalten instanziiert werden, die mit einem Setter geändert werden kann, auch während der Laufzeit. Code Redundanzen können so auch vermieden werden.

Für besseres Verständnis den Beispiel Code beachten.

UML



Beispiel-code

Main:

```
public class Main {

	public static void main(String[] args) {
		Husky husky = new Husky();
		Pudel pudel = new Pudel();

		pudel.bellen(); // GANZ LAUT BELLEN!!");

		husky.bellen(); // GANZ LAUT BELLEN!!");


		pudel.setBellVerhalten(new LeiseBellen());

		pudel.bellen(); // ganz leise bellen

	}

}
```

Hunde Klasse:

```
public abstract class Hund {

	  //Default Verhalten
    BellVerhalten bellVerhalten = new LautBellen();

    public void setBellVerhalten(BellVerhalten bellVerhalten) {
        this.bellVerhalten = bellVerhalten;
    }

    public void bellen(){
        bellVerhalten.bellen();
    }
}
```

Husky Klasse:

```
public class Husky extends Hund{

}
```

Pudel Klasse:

```
public class Pudel extends Hund{

}
```

Interface BellVerhalten:

```
interface BellVerhalten {
	public void bellen();
}
```

LautBellen Klasse:

```
public class LautBellen implements BellVerhalten{

	@Override
	public void bellen() {
		System.out.println("GANZ LAUT BELLEN!!");
	}

}
```

LeiseBellen Klasse:

```
public class LeiseBellen implements BellVerhalten{

	@Override
	public void bellen() {
		System.out.println("ganz leise bellen...");
	}

}
```