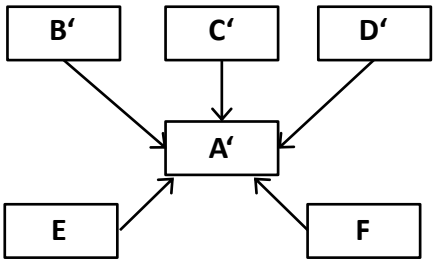
Software Engineering 2
Übungsblatt 2

Aufgabe: Interface versus Abstrakte Klasse

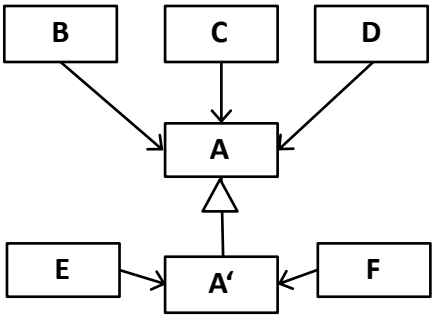
- Interface: Deklaration von Methoden, die in Unterklassen implementiert werden müssen
- Abstrakte Klasse: Deklaration von Methoden, aber auch Möglichkeit Methoden zu implementieren
- statt Interface kann auch abstrakte Klasse verwendet werden, aber Umkehrschluss gilt nicht immer

Aufgabe: Open-Closed Principle (Offen/Geschlossen-Prinzip)

[Ludewig, S.427]



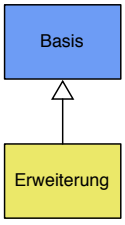
Variante ohne OCP

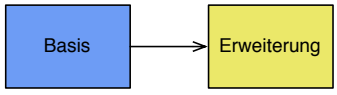


berücksichtigt OCP

Die Client-Klassen von A (also B, C, D) sind garantiert nicht von der Änderung betroffen.

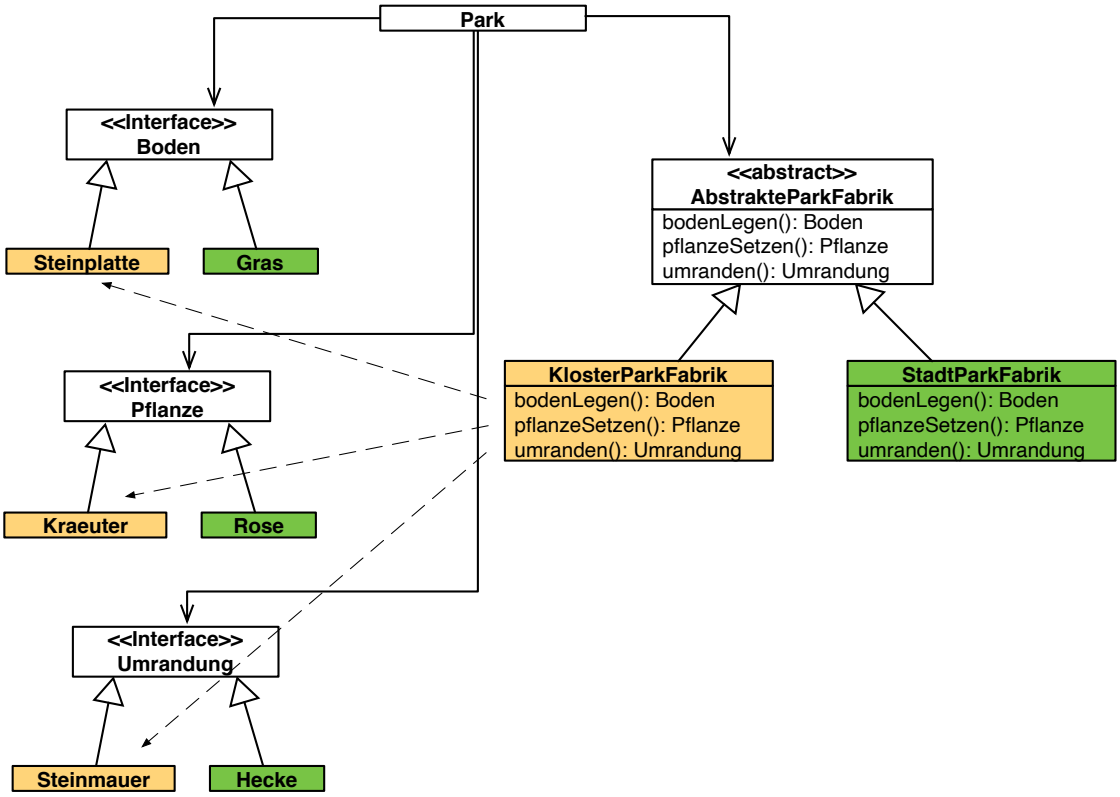
Umsetzungsbsp:

- **Vererbung**: Eine Erweiterung im Sinne des Open-Closed-Prinzips ist beispielsweise die Vererbung. Diese verändert das vorhandene Verhalten der Einheit nicht, erweitert aber die Einheit um zusätzliche Funktionen oder Daten. Überschriebene Methoden verändern auch nicht das Verhalten der Basisklasse, sondern nur das der abgeleiteten Klasse.
- Bsp: **Komponente** A (Basis) wird durch Komponente B (Erweiterung) erweitert (per Vererbung oder Delegation):
- 

oder


- **Interface** definieren
- Gegenbeispiel: switch für typ

Aufgabe Abstract Factory Pattern (siehe Siebler-Buch Kap. 11):



Nachteile erster Lösungsansatz:

- Objekterzeugung und eigentliche Funktionalität (operative Funktionen im Park) vermischt
- Offen/Geschlossen-Prinzip verletzt: Erweiterungen (neue Parkvariante, neue Park-Merkmale) erfordern Code-Änderungen; Änderungen (Tulpe statt Rose) erforern Code-Änderungen

Erweiterungen:

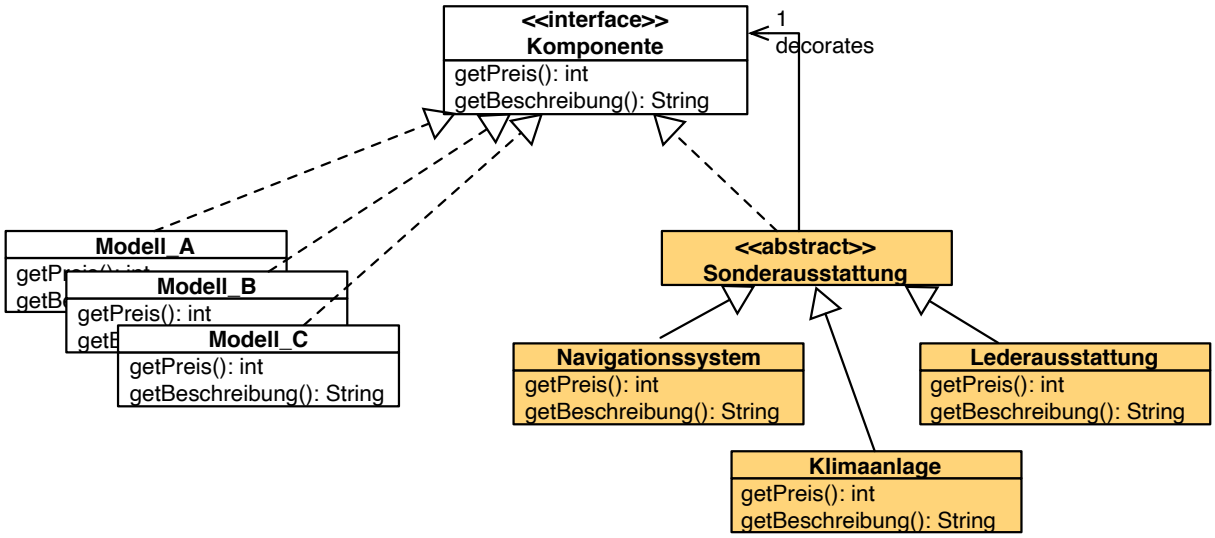
- neues Merkmal für Parks, z.B. Baum: zusätzliche Methode in AbstrakteParkFabrik
- neue Parkart, z.B. Japanischer Park: zusätzliche Fabrik "JapanParkFabrik" als Unterklassen von AbstrakteParkFabrik

```
public interface AbstrakteParkFabrik {
    Pflanze pflanzeSetzen();
    Boden bodenLegen();
    Umrandung umranden();
}
```

```
public class KlosterParkFabrik implements AbstrakteParkFabrik
{
    public Pflanze pflanzeSetzen() {
        return new Kraeuter();
    }

    public Boden bodenLegen() {
        return new Steinplatte();
    }

    public Umrandung umranden() {
        return new Steinmauer();
    }
}
```

```java
public class Park {
    private Boden boden;
    private Pflanze pflanze;
    private Umrandung umrandung;

    Park(AbstrakteParkFabrik fabrik) {
        Boden boden = fabrik.bodenLegen();
        Umrandung umrandung = fabrik.umranden();
        Pflanze pflanze = fabrik.pflanzeSetzen();
    }

    //  die eigentlichen operativen Funktionen im Park
    public void zuweisenAufgabeMitarbeiter() {
    // ...
    }
    ...
}
```

Aufgabe Decorator Auto-Bsp (siehe Siebler-Buch Kap. 22):



Erweiterungen:

- weitere Sonderausstattung, z.B. Schiebedach als Unterklasse von Sonderausstattung
- Preis der Klimaanlage ändert sich: Anpassung getPreis()-Methode in Klimaanlage

```
public interface Komponente
{
    public int getPreis();

    public String getBeschreibung();
}
```

```
public abstract class Sonderausstattung implements Komponente
{

    protected final Komponente BASIS_KOMPONENTE;

    protected Sonderausstattung(Komponente komponente)
    {
        this.BASIS_KOMPONENTE = komponente;
    }
}
```

```java
public class Navigationssystem extends Sonderausstattung
{
    public Navigationssystem(Komponente komponente)
    {
        super(komponente);
    }

    public int getPreis()
    {
        return BASIS_KOMPONENTE.getPreis() + 200;
    }

    public String getBeschreibung()
    {
        return BASIS_KOMPONENTE.getBeschreibung() + " und ein Navigtionssystem";
    }
}
```

```java
public class Model_A implements Komponente
{
    public int getPreis()
    {
        return 10000;
    }

    public String getBeschreibung()
    {
        return "Ein Kleinwagen";
    }
}
```

```java
public class ApplStart
{
    public static void main(String[] args)
    {
        Komponente grundmodell = new Model_A();
        Komponente navigation = new Navigationssystem(grundmodell);
        Komponente kaelte = new Klimaanlage(navigation);
        Komponente leder = new Lederausstattung(kaelte);


        System.out.println("Kundenwunsch: \n\t" + leder.getBeschreibung()
                + "\nPreis: \n\t" + leder.getPreis());

    }
}
```