Heinrich, Falk-Niklas 1630123
Furkan Hidayet Rafet Aydin 1630039

Aufgabe 2.3: Abstract Factory Pattern (Pflichtaufgabe)
In einem System sollen Parks verwaltet werden. Es gibt verschiedene Arten von Parks: Zum einen Klosterparks, in denen Kräuter angepflanzt werden. Zum anderen Stadtparks, in denen Rosen wachsen. Die Umrandung unterscheidet sich: Der Klosterpark ist von einer Steinmauer, der Stadtpark von einer Hecke umgeben. Der Boden ist auch unterschiedlich: Im Klosterpark liegen Steinplatten, im Stadtpark wächst Gras. Kloster- und Stadtpark sind zwei unterschiedliche Produktfamilien. Jede Produktfamilie hat gleiche Merkmale (Pflanzen, Umrandung, Boden), die aber unterschiedlich ausgestaltet sind.
Ein erster Lösungsansatz zum Anlegen, Pflegen und Verwalten (u.a. Mitarbeiter*innen einsetzen, Bewässerung steuern, Besucher*innen zählen etc.) von Parks könnte wie folgt aussehen:

```
public class Park {
    private enum Parktyp { Klosterpark, Stadtpark };
    private Parktyp park;
    private Boden boden;
    private Pflanze pflanze;
    private Umrandung umrandung;

    public void bodenLegen() {
        switch (park) {
            case Klosterpark:
                boden = new Steinplatte();
            break;

            case Stadtpark:
                boden = new Gras();
            break;
        }
    }

    public void pflaenzeSetzen() {
        switch (park) {
            case KlosterPark:
                pflanze = new Kraeuter();
            break;

            case Stadtpark:
                pflanze = new Rose();
            break;
        }
    }

    public void umranden() {
    // ... wie die obigen Methoden

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

    public void steuernBewaesserung() {
    // ...
    }

    public void steuernBeleuchtung() {
    // ...
    }

    public void zaehlenBesucher() {
    // ...
    }

    public void verwaltenFinanzen) {
    // ...
    }

    // ...
    // ...
```

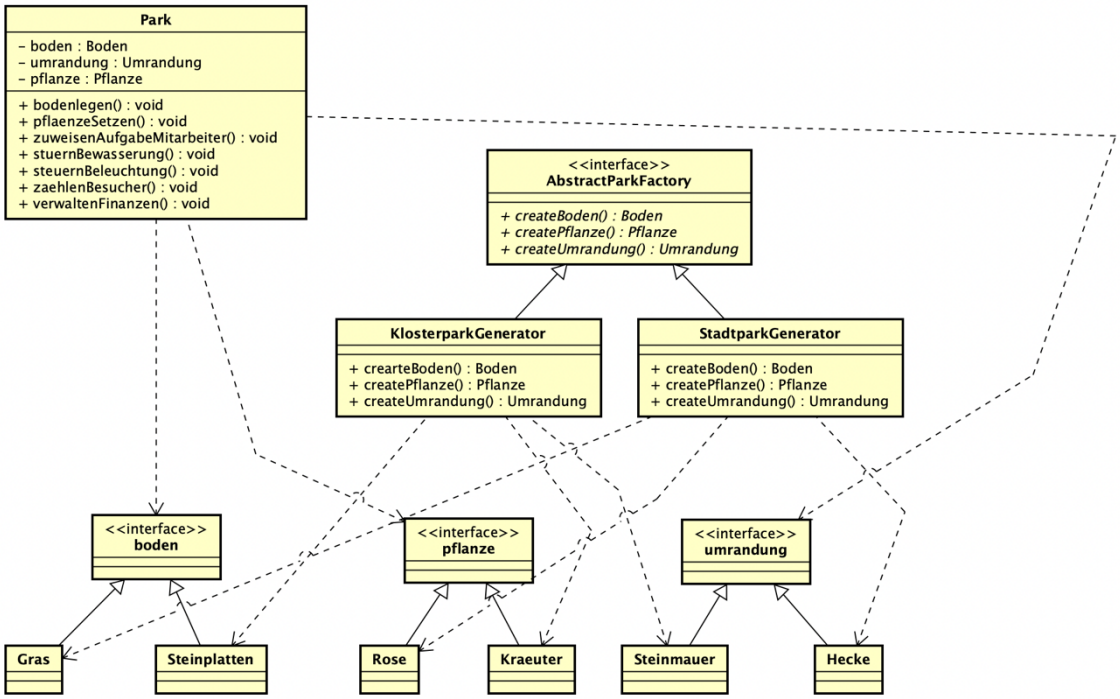
Welche Nachteile hat obige Implementierung? Verbessern Sie den Entwurf der Klasse `Park` durch Anwendung des **Abstract Factory Pattern**. Geben Sie hierzu das resultierende Klassendiagramm und die Implementierung der Java-Klassen an.
Welche Änderungen sind notwendig, um mit "Japanischer-Park" eine neue Parkart hinzuzufügen?

Nachteile:

- Zu viele Unterschiedliche Funktionen
- Geringe Kohäsion
- Schlecht wartbar/änderbar

Japanischen Park:

- Neuen Parktyp hinzufügen
- Neue Ausprägungen erstellen
- Neue Generator für den japanischen Park erstellen




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

    private AbstractParkFactory abstractFactory;
    private ParkGeneratorFactory parkgenFactory;

    public Park(String type) {} {
        this.type = type;
    }

    public void create() throws Exception {
        abstractFactory = parkgenFactory.createParkGenerator(this.type);
        boden = abstractFactory.createBoden();
        umrandung = abstractFactory.createUmrandung();
        pflanze = abstractFactory.createPflanze();
    }
}
```

```java
public interface Boden {

}
```

```java
public interface Pflanze {

}
```

```java
public interface Umrandung {

}
```

```java
public class Rose implements Pflanze {

}
```

```java
public class Gras implements Boden {

}
```

```java
public class Hecke implements Umrandung {

}
```

```java
public class Steinmauer implements Umrandung {

}
```

```java
public class Steinplatten implements Boden {

}
```

```java
public class Kraeuter implements Pflanze {

}
```

```java
public class ParkGeneratorFactory {
    public AbstractParkFactory createParkGenerator(String type) throws Exception {
        if (type.equals(anObject: "Klosterpark")) {
            return new KlosterparkGenerator();
        } else if (type.equals(anObject: "Stadtpark")) {
            return new StadtparkGenerator();
        }
        throw new Exception(message: "no correct generator type given");
    }
}
```

```java
public interface AbstractParkFactory {
    public Boden createBoden();
    public Pflanze createPflanze();
    public Umrandung createUmrandung();
}
```

```java
public class KlosterparkGenerator implements AbstractParkFactory {

    @Override public Boden createBoden() {
        return new Steinplatten();
    }

    @Override public Pflanze createPflanze() {
        return new Kraeuter();
    }

    @Override public Umrandung createUmrandung() {
        return new Steinmauer();
    }
}
```

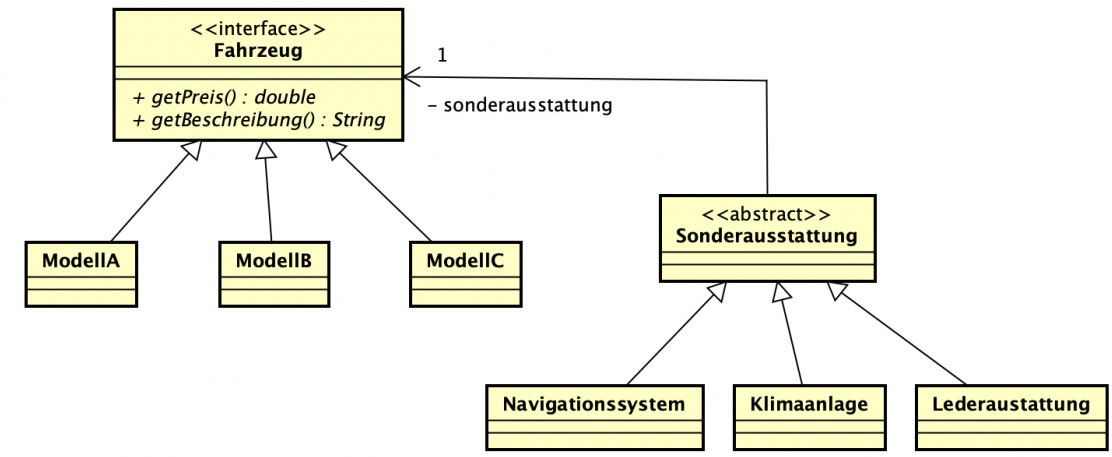
```java
public class StadtparkGenerator implements AbstractParkFactory {

    @Override public Boden createBoden() {
        return new Gras();
    }

    @Override public Pflanze createPflanze() {
        return new Rose();
    }

    @Override public Umrandung createUmrandung() {
        return new Hecke();
    }
}
```

Aufgabe 2.4: Decorator Pattern (Pflichtaufgabe)
Bei einem PKW-Hersteller gibt es wenige Grundmodelle (z.B. Modell_A, Modell_B, Modell_C) mit einer Fülle von optionalen Sonderausstattungen (z.B. Navigationssystem, Lederausstattung, Klimaanlage). Der Gesamtpreis für ein bestelltes Fahrzeug errechnet sich aus dem festen Grundpreis des Grundmodells plus der Preise der gewählten Sonderausstattung.

(a) Modellieren Sie obigen Sachverhalt mit dem **Decorator Pattern**.

(b) Implementieren Sie das Modell in Java. Neben der Preisberechnung mit der Methode `getPreis()` soll es auch eine Methode `getBeschreibung()` geben, die das bestellte Fahrzeug mit seiner Ausstattung beschreibt, z.B.

```
Bestelltes Fahrzeug: Ein Fahrzeug des Modell_B und eine
Klimaanlage und eine Lederausstattung
```


<<interface>>
Fahrzeug
+ getPreis() : double
+ getBeschreibung() : String
1
- sonderausstattung
<<abstract>>
Sonderausstattung
ModellA
ModellB
ModellC
Navigationssystem
Klimaanlage
Lederaustattung