Factory Patterns

Generelles Ziel der Factory Patterns ist es, den Programmcode unabhängig zu machen von konkreten Klassen. Eine Instanziierung mit dem `new`-Operator bedingt auch eine explizite Abhängigkeit von der Klasse des erzeugten Objektes. Da `new`-Operatoren im ganzen Code verstreut vorkommen können, wäre der Austausch einer Klasse mit grossem, fehleranfälligem Aufwand verbunden.

Factories helfen die Erzeugung von Objekten zentral vorzunehmen. Zudem können Factories mit relativ geringem Aufwand angewiesen werden Objekte von anderen Klassen zu erzeugen.

Factory Method

Ziel

Bereitstellen eines Interfaces bzw. einer abstrakten Methode zur Erzeugung von Objekten. Konkrete Subklassen entscheiden dann, welche konkreten Objekte erzeugt werden. *Factory Method* erlaubt es, die Instanzierung in konkrete Klassen auszulagern.

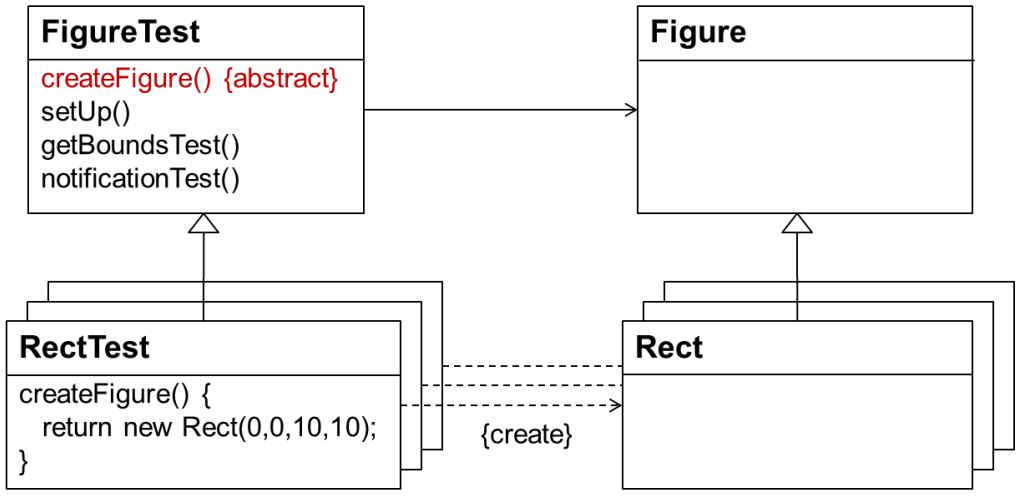
Motivation

JUnit Test für Figuren (z.B. Information Hiding Test). Es soll ein Test geschrieben werden, der für beliebige Figuren prüft, ob die Figure genügend gekapselt ist. Als Beispiel: wenn das Rectangle verändert wird, das man beim Aufruf von getBounds erhalten hat, dann darf sich die Lage der Figur *nicht* ändern.

Lösungen

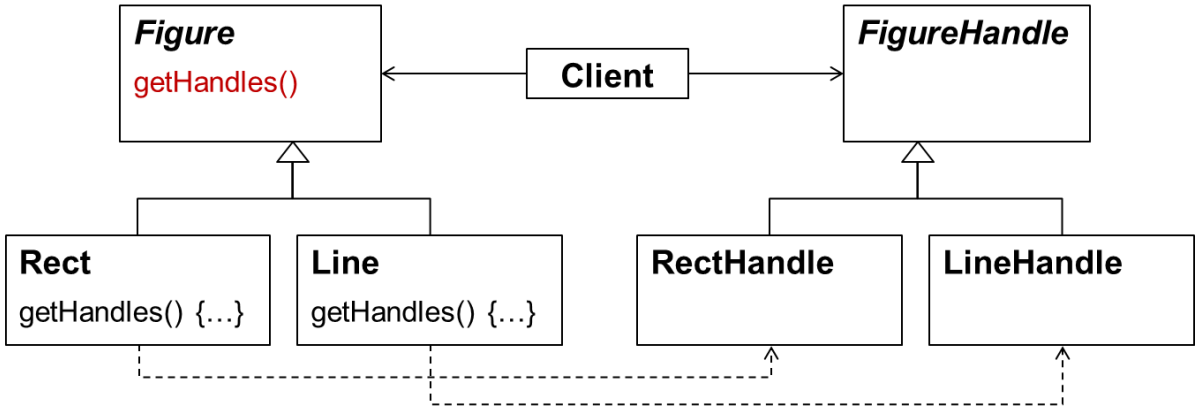
- Testklasse mit einem Konstruktor, dem die konkrete Figur übergeben wird.
  - ⇨ geht nicht für JUnit da JUnit-Klassen über einen parameterlosen Konstruktor erzeugt werden.
  - ⇨ JUnit unterstützt parametrisierte Unit-Tests, d.h. in der Testklasse wird ein Konstruktor definiert über welchen die zu testende Figur übergeben werden kann. Dies funktioniert im Kontext von JUnit, ist aber nicht eine generelle Lösung, denn so kann nur eine endliche Anzahl von Instanzen übergeben werden.

- Factory Method



Konsequenz

- Die Testklasse ist von konkreten Klassen unabhängig! Die Factory Methode stellt also einen Erweiterungspunkt dar, in den sich konkrete Subklassen einklinken können.
- Wird auch verwendet bei parallelen Klassenhierarchien.



Implementationsvarianten

- create-Methode abstrakt oder mit Defaultverhalten.
    - abstrakte Factory Methoden zwingen Subklassen eine eigene Implementation zu liefern. Werden dort eingesetzt wo es kein gemeinsames Default-Objekt gibt.
      Beispiel: `AbstractTool`, auf dieser Abstraktionsebene kann gar kein Default-Objekt zur Verfügung gestellt werden.

    - Defaultimplementationen werden meist in konkreten Basisklassen implementiert. Sie bieten eine Convenience-Implementation an, die von Subklassen bei nicht-gefallen überschrieben werden kann.
      Beispiel: `getHandles`

- Parameterized Factory Method
  Beispiel: `AbstractTool: createFigure(int figureType)`

```
public Figure createFigure(int figureType) {
  if (figureType == RECTANGLE) return new Rectangle();
  if (figureType == CIRCLE) return new Circle();
  //... add other class instatiations
  return null;
}
```

  Riechen Sie den Code Smell?

Abstract Factory

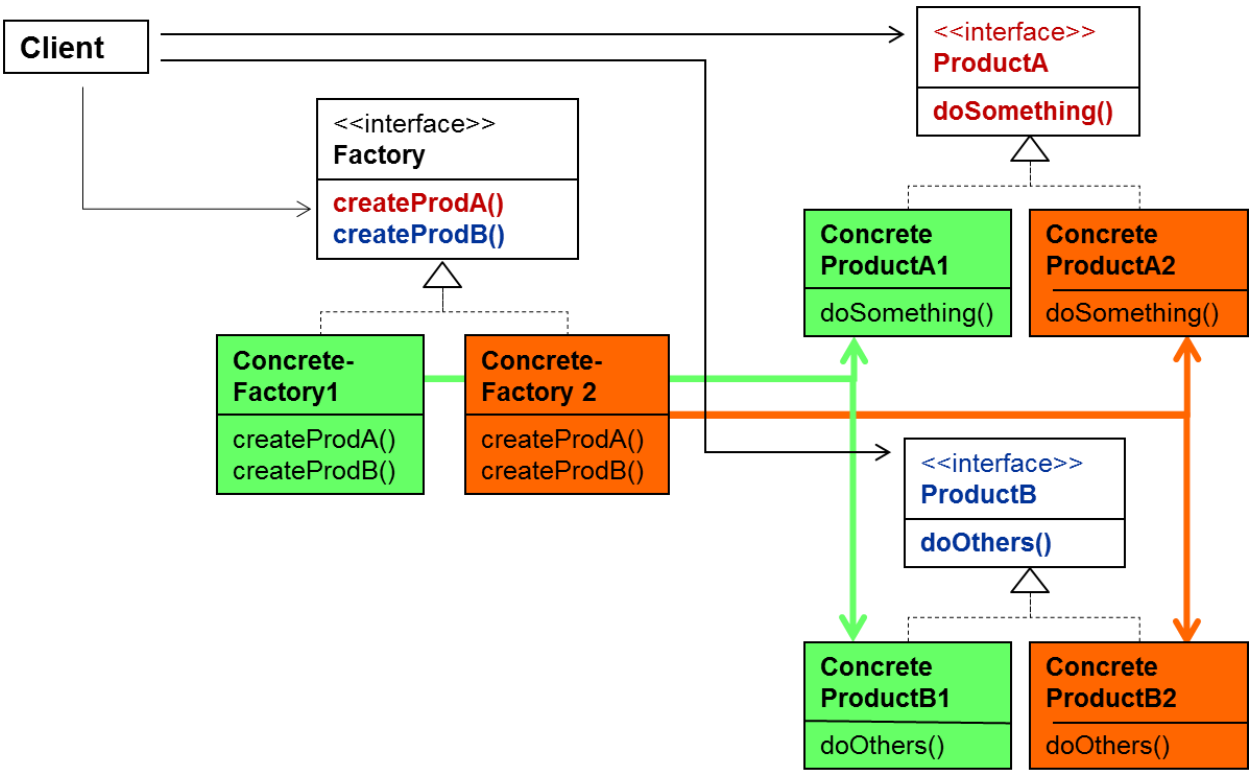
Ziel

Ein Interface anbieten zur Erzeugung von Familien zusammenhängender Objekte.

Motivation

- System soll von der Erzeugung von Objekten unabhängig sein, d.h. keine konkreten Implementierungen referenzieren.
- System soll mit einem bestimmten Satz von Klassen konfiguriert werden können.
- Erweiterbarkeit durch neue Implementierungen.

Struktur



Implementation

```
interface Factory {
    A createA();
    B createB();
}
```

```
interface A {
    // …
}
```

```
interface B {
    // …
}
```

1) Wo ist die konkrete Fabrik?
Die konkrete Fabrik muss für alle zugreifbar sein, d.h. sie muss in einer Klassenvariablen abgelegt werden.

- Falls die Factory eine abstrakte Klasse ist, dann kann diese Variable gerade in der Factory sein.
- Sonst in einer eigenen Klasse:

```
public class CurrentFactory {
   private CurrentFactory() { }; // prevents instantiation
   private static Factory fac = null;
   public static Factory getFactory() { return fac; }
   public static void setFactory(Factory f) {
      if (f == null) throw new NullPointerException();
      fac = f;
   }
}
```

Varianten:

- Mit Defaultimplementierung
- Nicht änderbar nach erstmaligem Setzen (frozen)

```
public static void setFactory(Factory f) {
   if (f == null) throw new NullPointerException();
   if (fac != null && f != fac) throw new IllegalStateException();
   fac = f;
}
```

2) Wie wird konkrete Fabrik registriert?
Jemand muss `setFactory` mit einer konkreten Factory aufrufen!

- Extern: `CurrentFactory.setFactory(new Factory1());`

  Intern: `Factory1.register();`
  Registriermethode in der Factory, welche eine neue Instanz registriert

```
class Factory1 implements Factory {
   public A createA() { /*...*/ }
   public B createB() { /*...*/ }
   static void register() { CurrentFactory.setFactory(new Factory1()); }
   private Factory1() { } // prevents instantiation
}
```

- Automatisches Registrieren beim Laden der Fabrik:

```
class Factory1 implements Factory {
  public A createA() { /* ... */ }
  public B createB() { /* ... */ }
  static { CurrentFactory.setFactory(new Factory1()); }
  private Factory1(){} // prevents instantiation
}
```

Mit `Class.forName("Factory1")` kann diese Klasse geladen und initialisiert werden.