# Übung Java Design Patterns - Abstract Factory

## Ausgangslage

Das Abstract Factory Pattern gehört zur Gruppe der Creational Patterns und ist Bestandteil der GoF.Mit dem Abstract Factory Pattern erzeugt man Familien verwandter oder voneinander abhängiger Objekte, ohne ihre konkreten Klassen zu benennen.Diese Übung soll anhand einer Vorlage aufzeigen, wie das Abstract Factory Pattern anzuwenden ist. Aus Ausgangslage dient uns eine bestehende Applikation VehiclePlace, welche das Factory Pattern implementiert, aber keine Abstract Factory. Die Vorlage implementiert das folgende Klassendiagramm: Die Applikation vermietet Fahrzeuge (Vehicles). Pro Art des Fahrzeuges ist eine Klasse (Car, Bike, Boat, ..) zugeordnet. Diese implementieren das Interface IVehicle. Das IVehicle Interface definiert Methoden für den Beginn (startRent()) und das Ende (stopRent()) einer Ausleihe des betroffenen Fahrzeuges. Die Methoden set/getMax() verwalten die maximale Anzahl der zu vermietenden Vehicles. Die Methode getAvailable() gibt die zur Zeit freien Vehicles und somit mietbaren Fahrzeuge zurück. Die Klasse VehiclePlace verwaltet alle vermietbaren Objekte (Vehicles).Diese Klasse hat nur eine Beziehung zum Interface IVehicle und kennt keine speziellen Vehicles. Diese Klasse kann auch keine neuen Vehicles instanziieren. Die Klasse VehicleFactory ist die einzige Klasse, die alle möglichen Vehicles kennt und instanziieren kann. Die Klasse stellt sicher, dass vom gleichen Typ nur genau eine Instanz vorhanden ist. Die Vorlage arbeitet mit der Klasse VehiclePlaceApp, welche eine in der Methode main(...) einen VehiclePlace installiert und über die VehicleFactory instanziiert. Das folgende Listing zeigt den Code der Methode main(...) auf: public class VehiclePlaceApp {
 public static void main(String[] args) {
  VehiclePlace vp = new VehiclePlace();

  // init vehicle place
  IVehicle car = VehicleFactory.instance("Car", 2);
  vp.add(car);
  IVehicle boat = VehicleFactory.instance("Boat", 5);
  vp.add(boat);
  IVehicle bike = VehicleFactory.instance("Bike", 10);
  vp.add(bike);

  // lets rent
  try {
  vp.startRent(bike);
  vp.startRent(boat);
  vp.startRent(car);
  vp.stopRent(bike);
  vp.stopRent(boat);
  vp.stopRent(car);
  } catch (Exception e) {
  System.out.println(e);
  }
  // lets rent a bike
  try {
  vp.startRent(car);
  vp.startRent(car);
  vp.startRent(car);
  } catch (Exception e) {
  System.out.println(e);
  }
 }
}Die Applikation ergibt den folgenden Output: start renting a bike
start renting a boat
start renting a car
stop renting a bike
stop renting a boat
stop renting a car
start renting a car
start renting a car
ch.std.jdpp.abstractfactory.VehicleException: no vehicle available for rent

## Template (Vorlage)

Diese Übung basiert auf einer Vorlage (Template). Die Java Dateien findet man hier.

## Vorgehen

Lösen Sie bitte die Aufgabe wie folgt:Studieren Sie das Klassendiagramm der Vorlage.Integrieren Sie die Klassen der Vorlage (Template) in Ihre Entwicklungsumgebung. Hierzu kann am einfachsten die ZIP-Datei verwendet werden.Kompilieren Sie die gesamte Applikation und führen Sie die Methode main(...) der Klasse VehiclePlaceApp aus. Studieren Sie das Resultat.Die Applikation soll nun so umgebaut werden, dass der VehiclePlace an verschiedenen Orten funktionieren kann. So soll z.B. ein amerikanischer VehiclePlace mit den Objekten USCar, USBoat und USBike oder ein französischer mit den Objekten FrenchCar, FrenchBoat, FrenchBike funktioneren können. Die Klasse VehiclePlace selber darf nicht angepasst werden. Das Interface IVehicle soll so belassen bleiben.Für jeden Ort (Land) kann hierzu eine Factory definiert werden. Die Applikation VehiclePlaceApp soll den Ort auswählen können und den VehiclePlace mit den richtigen Vehicle-Objekten initialisieren können. Diese Initialisierung soll über die richtige ConcreteFactory erfolgen.Planen und designen Sie jetzt das Refactoring, indem Sie die vorhandene Lösung mit dem Abstract Factory Pattern umbauen. Zeichnen Sie hierzu vielleicht auch ein UML-Diagramm. Sie können hierzu auch mit anderen Teilnehmern zusammenarbeiten.Setzen Sie alsdann Ihre Lösung in ein Java Programm um. Eventuell können Sie Teile der Vorlage wiederverwenden oder abändern.Testen Sie Ihre Lösung und verifizieren Sie das Resultat.

## Lösung

Eine mögliche Lösung finden Sie hier

## Kontakt

Simtech AG
Finkenweg 23
3110 Münsingen
Schweiz

## Impressum

Das Copyright für sämtliche Inhalte dieser Website liegt bei Simtech AG, Schweiz.
Beachten Sie auch unsere Hinweise zum Urheberrecht, Datenschutz und Haftungsauschluss.
Jeder Hinweis auf Fehler nehmen wir gerne entgegen.

## Copyright

2024 Simtech AG, All rights reserved, Powered by stack.ch written in Golang by Daniel Schmutz

## https://www.simtech-ag.ch/kurse-java-kurs-design-patterns-ressourcen