Factory Design Pattern**S** opdracht

Informatielinks

- https://en.wikipedia.org/wiki/Factory_(object-oriented_programming)
- https://stackoverflow.com/questions/929021/what-are-static-factory-methods
- https://stackify.com/static-factory-methods/
- https://en.wikipedia.org/wiki/Factory_method_pattern
- https://sourcemaking.com/design_patterns/factory_method
- https://www.tutorialspoint.com/design_pattern/factory_pattern.htm

Opmerking

Er zijn verschillende “Factory” patterns:

- Static Factory
- Simple Factory
- Factory Method
- Abstract Factory

Indien je dus ergens ziet staan “Factory Pattern”, dan kan het nog onduidelijk zijn om welke het gaat.

Van deze patterns hoef je de Abstract Factory niet te kennen.

Let op, de naamgeving van deze patterns kan verwarrend zijn, zo wordt er in de Factory Method vaak een abstracte class gebruikt, waardoor het gemakkelijk te verwarren is met Abstract Factory.

Case

Een nieuwe pizzeria in Rotterdam is van plan pizza’s te maken voor klanten. Huidige code:

```
class RdamPizzaStore {
    public Pizza orderPizzaInShop(String pizzaType) {
        Pizza pizza;
        if (pizzaType.equals("hawai")) {
            pizza = new RdamHawaiPizza();
        } else if (pizzaType.equals("pepperoni")) {
            pizza = new RdamPepperoniPizza();
        } else if (pizzaType.equals("cheese")) {
            pizza = new RdamCheesePizza();
        } else {
            throw new RuntimeException("Unknown pizza type");
        }

        pizza.prepare();
        pizza.bake();
        pizza.cut();
        pizza.box();

        return pizza;
    }
}

abstract class Pizza {
    public void prepare() {
        System.out.println("Adding ingredients to the pizza");
```

```
    }

    public void bake() {
        System.out.println("Putting pizza in the oven");
    }

    public void cut() {
        System.out.println("Cutting the pizza in peaces");
    }

    public void box() {
        System.out.println("Putting the pizza in a square box");
    }
}

class RdamHawaiPizza extends Pizza {}
class RdamPepperoniPizza extends Pizza {}
class RdamCheesePizza extends Pizza {}
```

Omdat de pizzeria net begonnen is, weten ze nog niet waar ze tegen aan gaan lopen, en kunnen er nog allerlei nieuwe pizza's bij komen.

Opdracht 1 – Static Factory

Een streven binnen object georiënteerd programmeren is o.a. het open for extension, closed for modification principe: een class of systeem zou uitgebreid moeten kunnen worden zonder dat je de bestaande code/class/systeem hoeft te wijzigen. (Stel je zou een class van een mede student veel gebruiken binnen je eigen systeem, en deze student loopt zijn/haar class/code continue te wijzigen, dan kan dat continue onzekerheid geven of je eigen systeem dan nog wel goed blijft werken. Bij voorkeur moeten classen daarom ongewijzigd blijven. Code die vaak wijzigt wordt, wordt daarom meestal geïsoleerd van de rest van de code.)

Een deel van de RDamPizzaStore code voor het maken van pizza's wijzigt vaak t.o.v. de andere code. Isoleer deze code door een static factory patroon toe te passen. Test of de code werkt.

Opdracht 2 – Simple Factory

(Code kan je het beste copy/pasten, zodat je ook opdracht1 nog hebt. Gemakkelijkste is om de code in een aparte package te plaatsen.)

Het nadeel van de static factory is dat de RdamPizzaStore altijd vast zit aan dezelfde factory, want er is immers een harde verwijzing naar via een static method.

In DenHaag besluiten ze ook een pizzeria te starten: DHPizzaStore. In deze pizzeria maken ze dezelfde pizza's, alleen met een iets andere stijl. In Rdam houden ze van dikke pizzabodems, en in DenHaag van dunne bodems, maar ze hebben wel dezelfde pizza soorten: hawai, pepperoni etc.

Maak een DHPizzaStore en bijbehorende pizza's, en pas de Simple Factory design pattern toe op beide pizzeria's. Test ook de code.

Opdracht 3 – Factory Method

I.p.v. de Simple Factory kan ook een Factory Method design pattern worden gebruikt. Het verschil tussen deze twee is dat de Simple Factory composition gebruikt (namelijk een PizzaStore die een

factory heeft en gebruikt), terwijl de Factory Method een abstract class gebruikt en creatie uitbesteedt aan zijn concrete subclassen.

Kopieer de code naar een nieuwe package, en wijzig van Simple Factory naar Factory Method. Test ook de code.