Opdracht 2 – Design patterns

Bronvermelding

Het spreekt uiteraard voor zich, maar wordt toch expliciet vermeld. Zorg ervoor dat je bij deze opdracht doet aan bronvermelding. Gebruik hiervoor de APA of IEEE stijl.

IJsjes maken

Vorig jaar hebben we geleerd dat we inheritance kunnen gebruiken voor soortgelijke objecten. Als we volgens dat systeem nu software zouden schrijven voor een ijscoman met drie soorten ijs genaamd vanilla, chocolade en mint dan zou dat er in UML uitzien als in Figure 1:

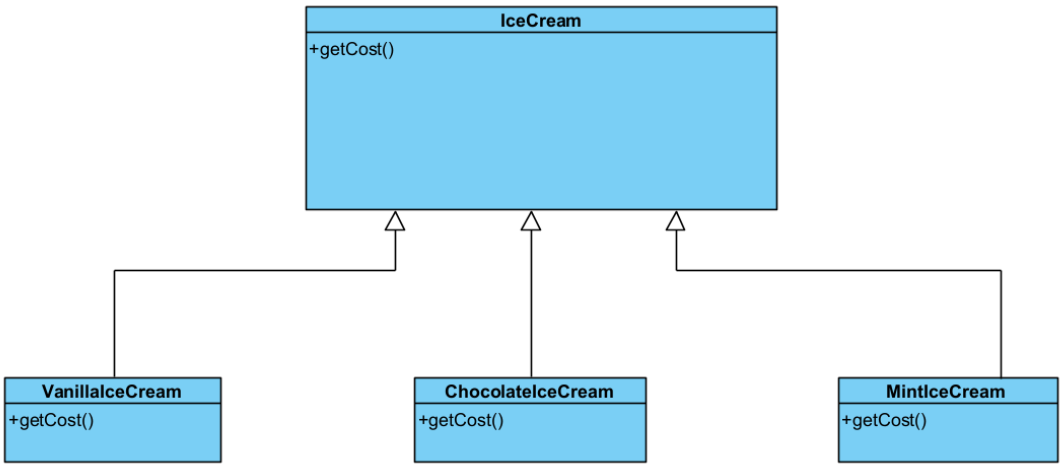

Figure 1: UML diagram van software voor de ijscoman

Op dit moment is dit nog te doen, maar zodra de ijscoman goede zaken draait en nieuwe soorten ijsjes wil toevoegen of zelfs bestaande smaken met elkaar wil combineren dan zal het al heel gauw onoverzichtelijk worden. Zie bijvoorbeeld Figure 2.

In dit gedeelte van de opdracht moet je dit probleem oplossen op de manier zoals getoond in Figure 3. Gebruik hiervoor onderstaande Main klasse. De rest van de van code dient helemaal zelf geïmplementeerd te worden. Er zijn dit keer dus geen testen in combinatie met een skelet aan code aanwezig.

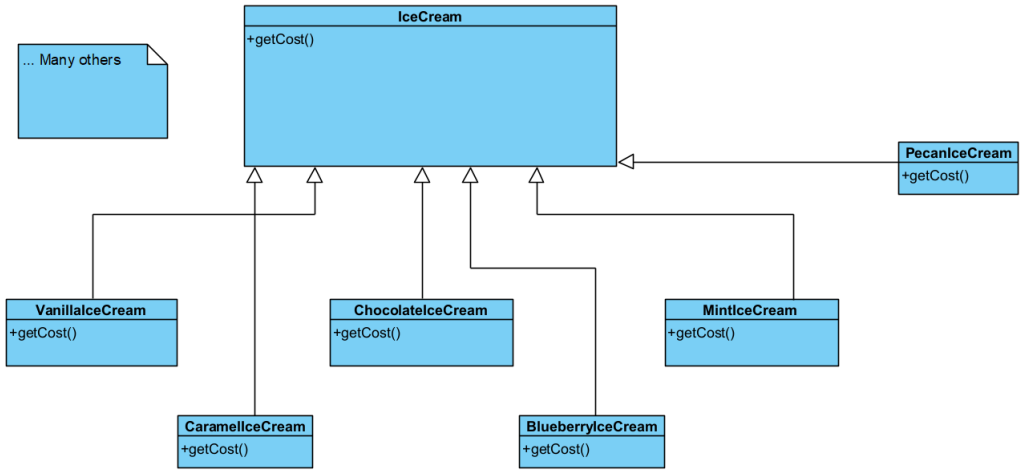

Figure 2: Software architectuur wordt onoverzichtelijk als er meer ijssmaken worden toegevoegd

De kosten voor een BasicIceCream zijn €0.50, VanillaIceCream €1.00, MintIceCream €1.50 en tot slot ChocolateIceCream €2.00. Zorg ervoor dat de geïmplementeerde code uiteindelijk werkt met de onderstaande main klasse. Als je het goed hebt gedaan dan volgt er een totaalprijs van € 5,-. Aangezien het echter mogelijk is om de juiste output te krijgen zonder gebruik te maken van de decorator Design Pattern, biedt de juiste uitkomst dan ook geen garantie voor correcte implementatie.

```
public class Main {

    public static void main(String[] args) {
        IceCream basicIceCream = new BasicIceCream();
        System.out.println("Basic ice cream cost €" + basicIceCream.cost());

        // Add Vanilla to the order
        IceCream vanilla = new VanillaIceCream(basicIceCream); // wrapping basic ice cream with vanilla
        System.out.println("Adding Vanilla - cost is: €" + vanilla.cost());

        // Add Mint to the order
        IceCream mintVanilla = new MintIceCream(vanilla); // wrapping
        System.out.println("Adding Mint - cost is: €" + mintVanilla.cost());

        // Add Chocolate to the order
        IceCream chocolateMintVanilla = new ChocolateIceCream(mintVanilla); // wrapping
        System.out.println("Adding Chocolate - cost is: €" + chocolateMintVanilla.cost());
    }

}
```

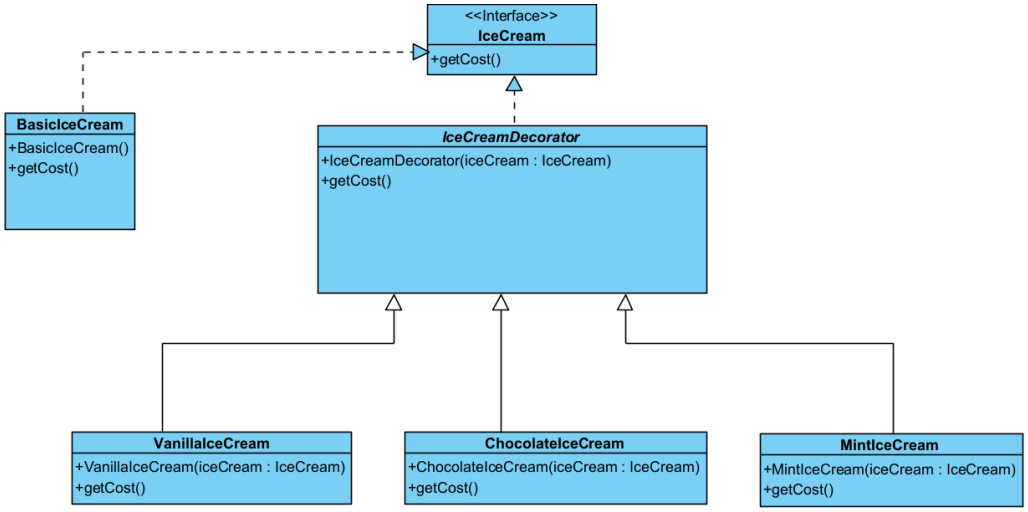

Figure 3: UML diagram voor het implementeren van de decorator pattern

Design Patterns herkennen

In dit gedeelte van de opdracht zal er van je gevraagd worden om uit elke categorie drie patterns te herkennen. Hiervoor krijg je steeds drie sets aan Java APIs waarbij elke set steeds gebruik maakt van één pattern. Het is de bedoeling dat je de drie vermelde patterns op de juiste manier weet te matchen met de sets.

Hierbij is het de bedoeling dat je eerst zelf onderzoek doet naar de drie gegeven design patterns met behulp van één van onderstaande bronnen:

- Design Patterns video series van Derek Banas
- Head First Design Patterns: A Brain-friendly Guide – Eric Freeman

Vervolgens kan je de betreffende Java APIs onderzoeken met behulp van de officiële documentatie docs.oracle.com of met behulp van verschillende studieboeken zoals:

- Big Java Late Objects - Cay S. Horstmann
- Core Java Volume I - Fundamentals – Cay S. Horstmann
- Head First Java - Kathy Sierra, Bert Bates

Na het onderzoek is het van belang dat je in je eigen woorden weet uit te leggen waarom je die specifieke set aan APIs hebt gematcht met dat design pattern. Gebruik hiervoor de template op pagina 5 en zorg ervoor dat je kort en bondig motiveert waarom deze match correct is en dat je je begrip van het gebruik van het design pattern aantoont. Voor het juist identificeren krijg je dan ook helemaal geen punten als de motivatie mist of niet klopt.

Creational Patterns herkennen

Match onderstaande sets met de volgende patterns: **builder, singleton, static factory method**

1. java.lang.Runtime
   java.lang.Desktop

2. com.google.common.collect.MapMaker

3. java.util.Calendar
   java.text.numberFormat
   java.nio.charset.Charset

Structural Patterns herkennen

Match onderstaande sets met de volgende patterns: **adapter, decorator, flyweight**

1. java.lang.Integer
   java.lang.Boolean

2. java.io.InputStreamReader
   java.io.OutputStreamWriter
   java.util.Arrays

3. java.io.BufferedInputStream
   java.io.DataInputStream
   java.io.BufferedOutputStream

Behavioral Patterns herkennen

Match onderstaande sets met de volgende patterns: **command, iterator, observer**

1. java.lang.Runnable
   java.util.concurrent.Callable

2. java.util.Iterator

3. java.util.EventListener

Template

Design Pattern	Classes	Onderbouwing
Builder		
Singleton		
Static Factory Method		
Adapter		
Decorator		
Flyweight		
Command		
Iterator		
Observer		

Design Patterns implementeren

Voor dit gedeelte van de opdracht willen we dat je een implementatie van design patterns maakt. Kies een eigen voorbeeld voor het ***singleton pattern*** en het ***adapter pattern***. Zorg dat de implementaties van deze voorbeelden simpel maar duidelijk zijn en jouw begrip van deze twee design patterns aantonen.