Practicum OO – lente 2016 **Opgave 8**

Patterns

Leerdoelen

Deze opgave bestaat uit twee onderdelen. Deze onderdelen zijn de pattern-opgaven uit het tentamen en hertentamen van 2014. Het *strategy pattern* en het *decorator pattern* uit deze opgave moet je ook in nieuwe situaties kunnen toepassen. Voor de volledigheid: ook de andere patterns uit het college behoren tot de tentamenstof.

1 Tijd voor een ijsje

In deze opdracht vragen we je een object-georiënteerde representatie te maken van verschillende soorten ijsjes die elk voorzien kunnen zijn van nul of meerdere *toppings*. Een topping is een ingrediënt waarmee je jouw ijsjes kunt bedekken. Bekende voorbeelden zijn slagroom of een chocodip. Voor een chocodip wordt het ijsje kort gedompeld in een vloeibare chocoladesaus. Deze saus vormt, eenmaal gestold, een dun laagje chocolade. In principe kun je zoveel toppings vragen als je wil, eventueel van één topping zelfs verschillende lagen. Wat afgeraden wordt is om het ijs eerst van slagroom te voorzien om het daarna te chocodippen. Er zijn vast meer onhandige combinaties, maar daar hoef je in deze opdracht allemaal geen rekening mee te houden.

We geven de volgende feiten over deze ijsjes:

1. Er zijn geen objecten van het basistype `IJsje`. Wel bestaan er objecten van de afgeleide types `VanilleIJs` en `YoghurtIJs`.

2. Ieder *IJsje* heeft een *beschrijving* en een *prijs*. Voor het modelleren van de eerste eigenschap gebruiken we een **protected** attribuut, genaamd `beschrijving`, van het type `String` terwijl de tweede eigenschap gemodelleerd wordt door een parameterloze (abstracte) methode met als naam `prijs` die een **int** (de prijs uitgedrukt in centen) als resultaat heeft. Voor het opvragen van de beschrijving introduceren we een methode `geefBeschrijving`. Deze methode dient voor de klasse `IJsje` zelf de string `"onbekend ijsje"` op te leveren. De klassen `VanilleIJs` en `YoghurtIJs` dienen een nieuwe, geschikte waarde aan `beschrijving` toe te kennen en een definitie te geven voor `prijs`. De prijs van een vanille-ijsje bedraagt 150 eurocent, terwijl een yoghurtijsje 200 cent kost.

3. Ieder ijsje kan door de klant naar keuze van een topping worden voorzien. Er zijn 3 soorten toppings, te weten `Slagroom`, `Chocodip` en `Spikkels`. De prijs van slagroom bedraagt 50 cent, van de chocodip 30 cent en de spikkels zijn gratis.

Voor het modelleren van de ijsjes ga je natuurlijk (en hier verplicht) gebruik maken van het zogenaamde *decorator* pattern. Hierbij representeer je het basistype `IJsje` en de concrete varianten hiervan op gebruikelijke wijze. Voor de topping introduceer je een geschikte abstracte klasse (bijvoorbeeld met naam `Topping`) die een uitbreiding vormt van `IJsje`, waarin je tevens een attribuut van het type `IJsje` opneemt. Zo werkt het decorator pattern immers. Het attribuut gebruik je om het ijsje dat van een topping wordt voorzien op te slaan. Verder zijn alle concrete toppings uitbreidingen van de klasse `Topping`.

1.a) Implementeer de Java klassen en eventuele interfaces die je hiervoor nodig hebt.

1.b) Geef enkele voorbeelden van concrete van toppings voorziene ijsjes waarvan je de beschrijving en de prijs afdrukt.

2 Webwinkel

In deze opdracht ga je een object-georiënteerd ontwerp te maken van een webwinkel. De klant van zo'n winkel krijgt bij binnenkomst een lege *WinkelWagen*. Tijdens het winkelen wordt de winkelwagen gevuld

met *Artikelen* en aan het einde van het bezoek aan de webwinkel worden de artikelen afgerekend waarna de webwinkel deze zal opsturen. De methode van verzending hangt in de praktijk af van de grootte en gewicht van het artikel. In deze applicatie zijn we enkel geïnteresseerd in de verzendkosten die bij het totaalbedrag zullen worden opgeteld. Ieder `Artikel` bevat een `beschrijving` (als `String` gerepresenteerd) en een `prijs` (waarvoor een **double** gebruikt wordt). Daarnaast kun je van elk artikel de `verzendkosten` opvragen. De eerste twee eigenschappen worden als attribuut vastgelegd; de `verzendkosten` door middel van een *abstracte methode*, die een **double** oplevert. Concrete artikelen geven een implementatie van deze methode. De artikelen worden verzameld in een `Winkelwagen`. Deze winkelwagen bevat methoden om een artikel toe te voegen, te verwijderen, de totaalprijs uit te rekenen (inclusief de verzendkosten) en de klant te laten betalen. Voor het doorgeven van de betaalwijze wordt gebruik gemaakt van het *strategy pattern* waarin middels een abstracte methode **boolean** `betaal(`**double** `bedrag)` de betaalwijze wordt aangeduid. Het **boolean** resultaat geeft aan of de betaling gelukt is. Er zijn verschillende concrete betaalwijzen: `iDeal`, `CreditCard` en `PayPal`. Voor `iDeal` heb je een bank, een rekeningnummer en een pincode nodig. Voor je `CreditCard` een kaartnummer, naam en verloopdatum en voor `PayPal` een emailadres en een wachtwoord. De default-betaalwijze voor het afrekenen is `iDeal`. Deze kan worden aangepast met de methode `veranderBetaalwijze`.

Natuurlijk wordt niet elk artikel afzonderlijk verzonden. Je mag voor de bepaling van de verzendkosten aannemen dat alle artikelen waarvan de verzendkosten gelijk zijn, gelijktijdig worden verzonden en dat hiervoor maar één keer verzendkosten in rekening worden gebracht. Om het geheel wat realistischer te maken veronderstellen we de volgende 3 concrete artikelen: watermeloenen (prijs €4,50 en verzendkosten €6,75), wijnglazen (prijs €8,50 en verzendkosten €6,75) en wasmachines (prijs €499 en verzendkosten €30,00).

2.a) Implementeer bovenstaande klassen. Voeg aan elke klasse een voor de hand liggende constructor toe. Van zowel de concrete artikelen als van de concrete betaalwijzen hoef je er maar één van ieder te implementeren. Maak de methoden zo simpel mogelijk. Het is bijvoorbeeld voldoende als de methode `betaal` alleen de gegevens van de betaalwijze en het bedrag afdrukt.

Inleveren

Vóór zondag 10 april, 23:59 uur, via Blackboard.