


|   |  |
|---|--|
|  <b>INTEC BRUSSEL</b><br>MEER KANSEN OP WERK | <b>Datum:</b> 30/03/2020<br><b>Opleiding:</b> Java Developer<br><b>Lesmodule:</b> Java Basis<br><b>Test:</b> H1 – H9<br><b>Tijd:</b> |
| <b>Resultaat:</b> ..... / <b>21</b>   | <b>Naam:</b> .....   |

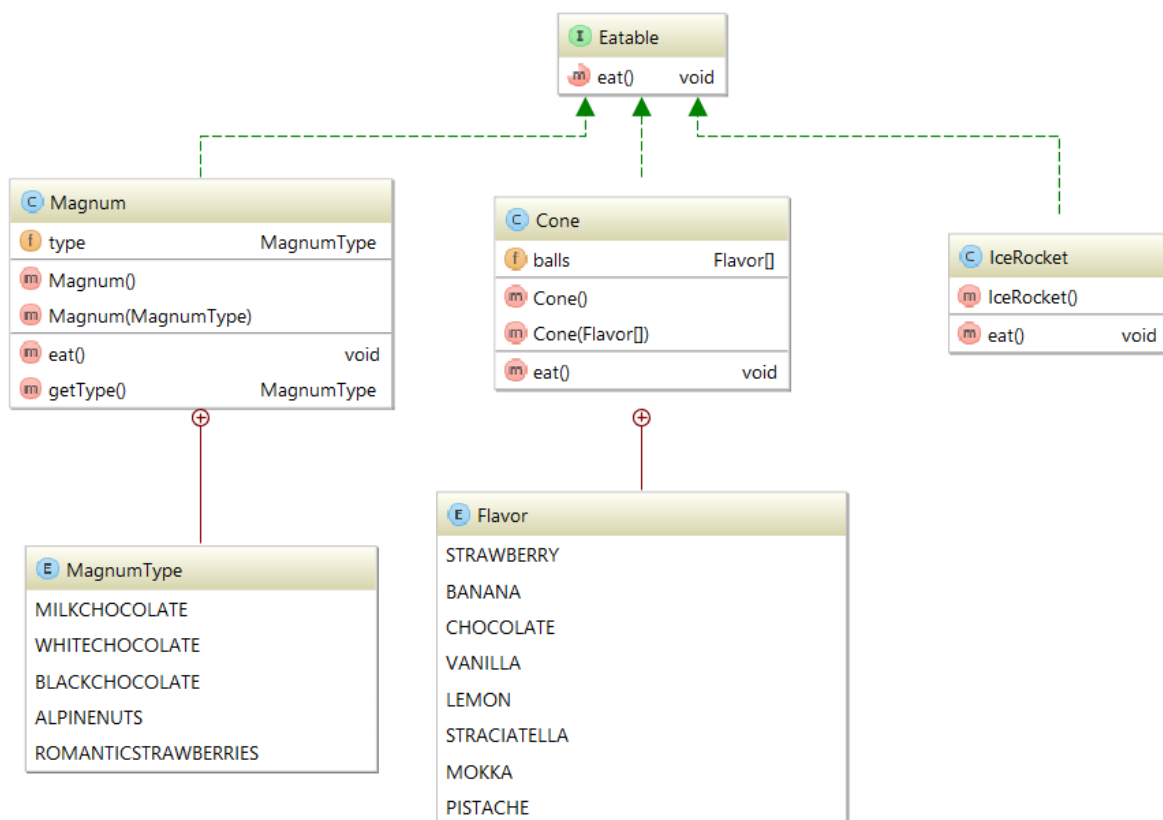
# 1. TAKE ME TO THE ICECREAM SHOP

We gaan in deze test een applicatie maken waarbij we de verkoop van en het eten van ijsjes gaan simuleren. De ijsjes zijn: magnumijsjes (verschillende types), hoorntjes (met verschillende smaken voor de ballen) en raketijsjes.

**Maak hiervoor een nieuw project aan met de packages: `be.intecbrussel.eatables`, `be.intecbrussel.sellers` en `be.intecbrussel.application`.**

## 1.1 Eatables

/5



Bovenstaand schema stelt de **eatables** package voor. We beschrijven hier de verschillende ijsjes die we later zullen verkopen. Belangrijke implementatiedetails:

- De eat methode print af wat je aan het eten bent. Bij een Magnum is dit dus afhankelijk van het magnumtype, bij een Cone zijn het alle verschillende soorten ballen, bij de IceRocket gaat het gewoon over het eten van een raketijsje.

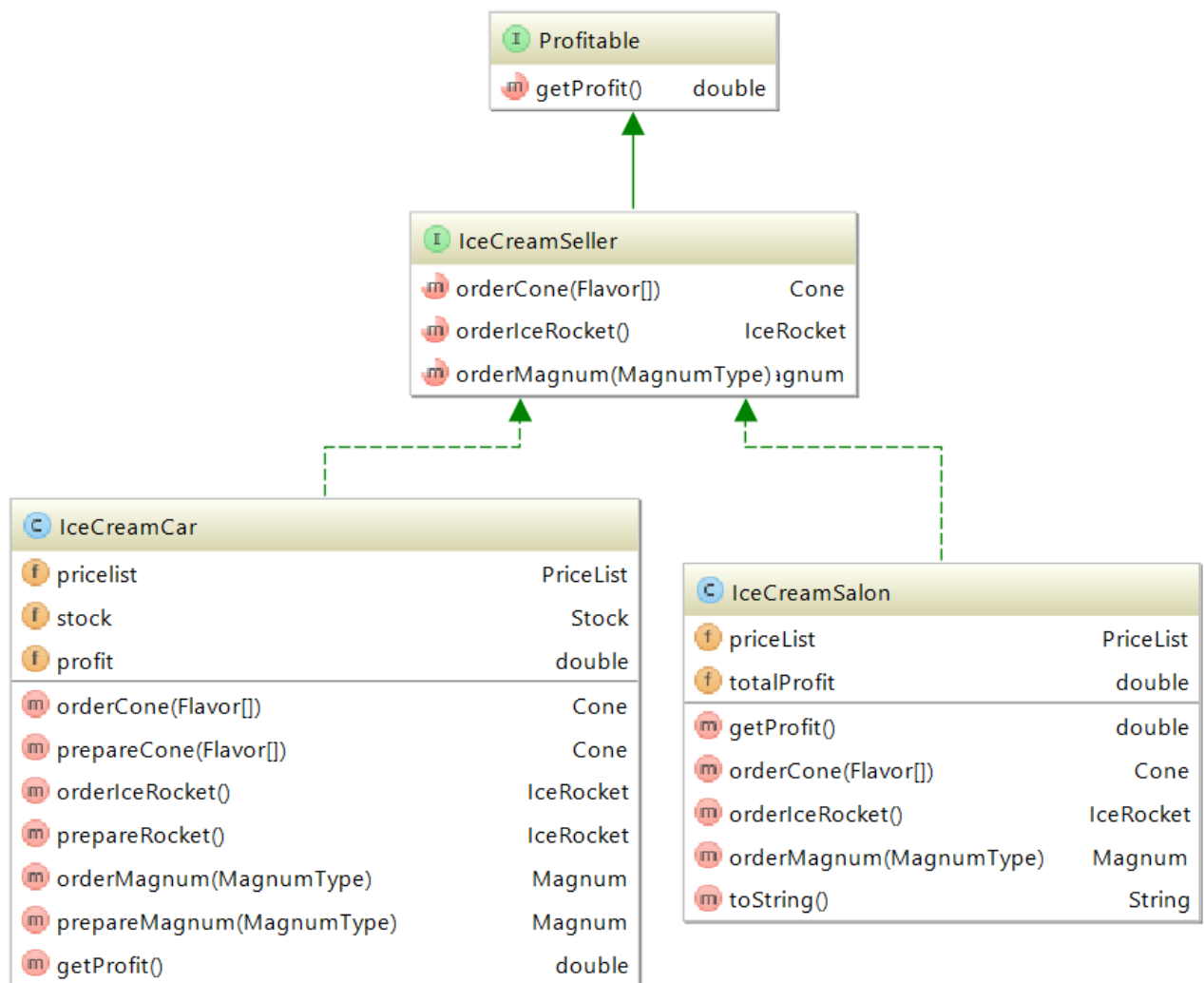
- De enums zijn genest binnen de Magnum en Cone klassen. Zij stellen de soorten magnumtypen en de smaken van de bolletjes ijs voor.

## 1.2 Prijslijst

/2

| PriceList |                                   |        |
|-----------|-----------------------------------|--------|
| f         | ballPrice                         | double |
| f         | rocketPrice                       | double |
| f         | magnumStandardPrice               | double |
| m         | PriceList()                       |        |
| m         | PriceList(double, double, double) |        |
| m         | setBallPrice(double)              | void   |
| m         | setRocketPrice(double)            | void   |
| m         | setMagnumStandardPrice(double)    | void   |
| m         | getBallPrice()                    | double |
| m         | getRocketPrice()                  | double |
| m         | getMagnumPrice(MagnumType)        | double |

Dit schema hoort bij de **sellers** package en zal de prijzen van de ijsjes bijhouden. De getmagnumprice methode zal aan de hand van een gegeven MagnumType en een algemeen opgegeven magnumStandardPrice teruggeven wat de prijs van zo'n magnum zal zijn. Bv: Een alpinenuts magnum zal 1.5 keer de prijs zijn van een standaard magnum.



Dit schema past ook binnen de **sellers** package.

We starten met het maken van de interfaces en het IceCreamSalon. De IceCreamCar zullen we later behandelen.

Een IceCreamSalon stelt een ijsjeszaak voor die een ongelimiteerd aantal ijsjes heeft die ze zelf produceren. Je kan er ijsjes bestellen en je kan er de winst van opvragen. Ze implementeert immers de IceCreamSeller interface.

Uitleg bij de implementatie van het IceCreamSalon:

- `orderCone` zal op basis van een array van flavors een nieuwe cone aanmaken en teruggeven. Ook zal ze aan de hand van de pricelist de totalprofit waarde omhoog laten gaan.
- `orderIceRocket` zal een nieuwe icerocket aanmaken en teruggeven. Ook zal ze aan de hand van de pricelist de totalprofit waarde omhoog laten gaan.
- `orderMagnum` zal op basis van een MagnumType een nieuwe Magnum aanmaken en teruggeven. Ook zal ze aan de hand van de pricelist de totalprofit waarde omhoog laten gaan.

Vergeet ook niet een constructor aan te maken waar je een PriceList aan kan meegeven.

## 1.4 Eerste applicatie

/1

Maak in je application package een klasse aan met een main methode. (bv: IceCreamApp)

Maak een PriceList instantie aan.

Maak een IceCreamSalon instantie aan met behulp van de pricelist en steek deze in een IceCreamSeller variabele.

Bestel enkele ijsjes (order methoden), steek deze in een array van Eatable.

Roep van deze ijsjes de eat methode aan.

Aan het einde van je applicatie print je de profit af van de icecreamseller.

## 1.5 Keeping stock & NoMoreIceCreamException

/2

| Stock |                    |      |
|-------|--------------------|------|
| f     | iceRockets         | int  |
| f     | cones              | int  |
| f     | balls              | int  |
| f     | magni              | int  |
| m     | getIceRockets()    | int  |
| m     | setIceRockets(int) | void |
| m     | getCones()         | int  |
| m     | setCones(int)      | void |
| m     | getBalls()         | int  |
| m     | setBalls(int)      | void |
| m     | getMagni()         | int  |
| m     | setMagni(int)      | void |

De Stock klasse helpt ons in het bijhouden van een voorraad ijsjes. Deze stock zullen we nodig hebben wanneer we de iceCreamCar.

Tevens maken we een eigen exception klasse: NoMoreIceCreamException. Dit mag een runtime exception zijn maar het mag dus ook een gewone exception zijn.

## 1.6 IceCreamCar

/4

Zie het schema bij 1.3.

We gaan een 2<sup>e</sup> soort icecreamseller definiëren, de icecreamcar. Deze zal een eindige voorraad ijsjes hebben. Ze gaat dus gebruik maken van een stock.

Zorg dus ook dat je aan een constructor zowel een pricelist als een stock meegeven kan.

De orderMethoden zijn grotendeels hetzelfde als bij een IceCreamSalon, alleen dat je nu ook rekening moet houden met de stock. Indien er van een bepaald ijsje geen voorraad meer is, gooi je een NoMoreIceCreamException. Deze vang je dan eventueel later op in je programma. De ordermethoden geven in ieder geval normaal gezien nooit een null waarde terug.

De prepareCone, prepareRocket en prepareMagnum methoden zijn private en dienen eerder als hulpmethoden gebruikt te worden binnen je ordermethoden om het boeltje clean te houden.

## 1.7 AppV2

/1

Maak een nieuwe IceCreamApp aan in je application package. (vb:IceCreamAppV2)

Deze werkt exact als je eerste applicatie (kopieren) op enkele verschillen na:

- Je gebruikt een IceCreamCar instantie in plaats van een salon
- Je geeft een Stock en een PriceList mee aan je IceCreamCar constructor

Bestel nu meer raketijsjes dan er in je stock zitten. Wat gebeurt er? Vang deze eventuele uitzonderlijke gebeurtenis op en run het programma opnieuw.

*Veel Succes!*