Tentamen OOP   
Make IT Work

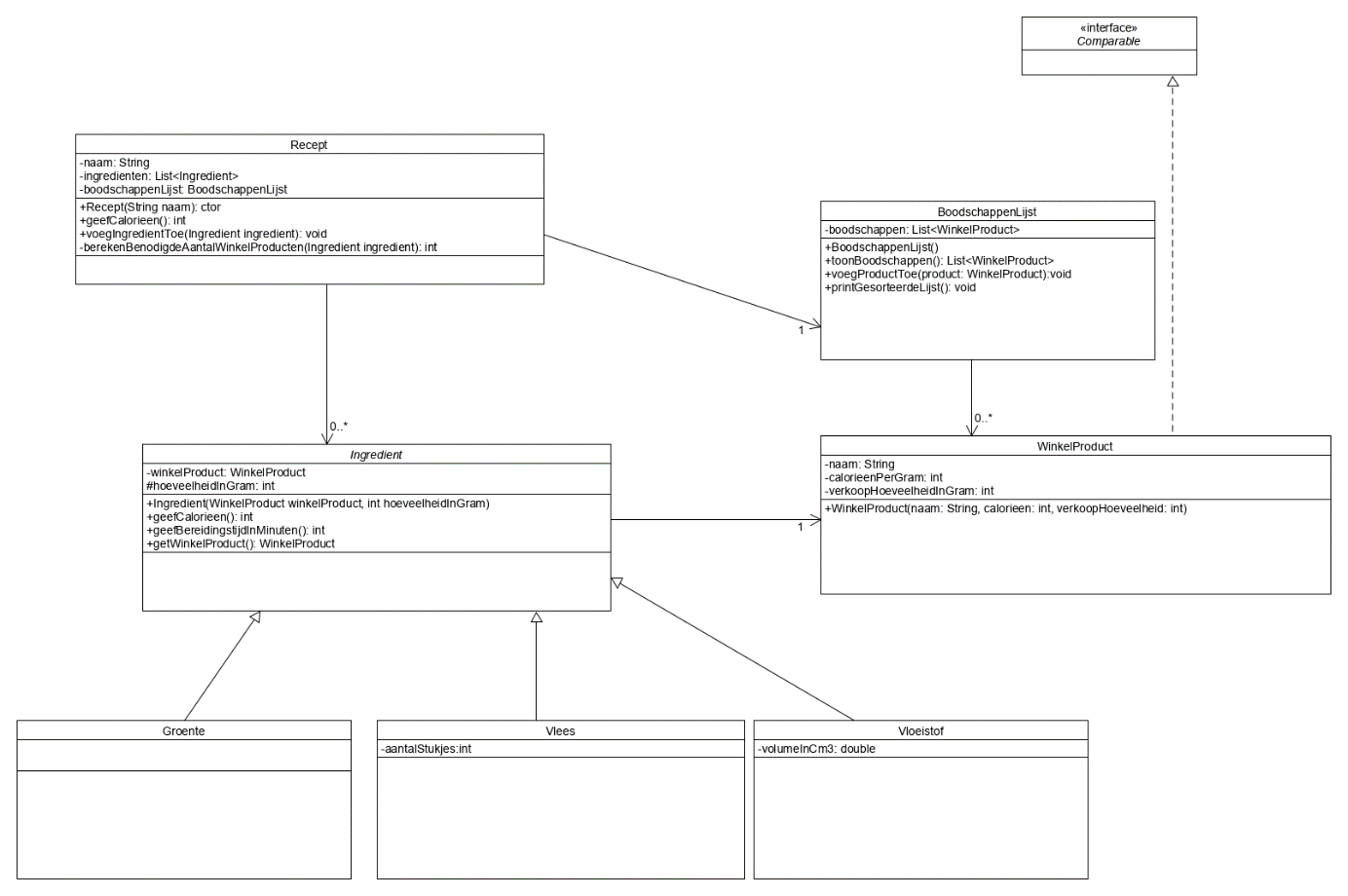
|  |  |
| --- | --- |
| Naam |  |
| Studentnummer |  |
| Tijdsduur | 2 uur |

# Opmerkingen

1. Je krijgt voor deze opgave een startproject genaamd **recepten - startproject inclusief tentamen.zip**.
2. In .zip zit een sql-script genaamd **recepten.sql** om een database te maken met een user.
3. Verander de naam van het project in TentamenOOPCohort03<naam><studentnr>, waarbij je je eigen naam en studentnummer op de juiste plek zet.
4. Upload na afloop een zip van je hele project naar de aangegeven locatie.
5. Het gebruik van het internet is toegestaan.
6. Daarnaast is toegestaan het gebruik van het boek, je eigen aantekeningen en al het materiaal op je eigen laptop (inclusief uitwerkingen van practicumopgaven).
7. Als je het tentamen inlevert checkt de docent of de upload is gelukt. Verlaat het lokaal niet voordat die check is uitgevoerd en je toestemming hebt gekregen!

# Klassendiagram Recepten

Het kookprogramma 'Make it Cook' heeft behoefte aan een recepten applicatie. Hiervoor is het volgende class diagram opgezet voor de klassen in 'model' package (zie **recepten.png** file).



Opmerkingen:

* De ReceptenLauncher–class is al voor je gemaakt en bevat veel testcode. Maak hier gebruik van! De voorbeeldoutput is ook gebaseerd op deze testcode.
* In het class diagram zijn geen getters en setters opgenomen. Voeg zelf de getter(s) en setter(s) toe die je nodig denkt te hebben. Je mag niet standaard alle getters en setters opnemen, maar alleen degene die je nodig hebt.
* De all-args constructor voor de klassen 'Groente', 'Vlees' en 'Vloeistof' moet je zelf maken. Deze constructoren staat dus NIET in het class diagram.
* In het class diagram staan geen klassen die nodig zijn voor het maken van een connectie met een database en ook geen DAO-klassen. Een DBaccess klasse en een AbstractDAO zijn wel al aanwezig in je project. De database user uit de DBaccess klassie dien je zelf aan te maken in de database.
* De interface Comparable bestaat natuurlijk al in Java, die hoef je dus niet zelf te maken.
* Zorg dat je default methoden (bijvoorbeeld toString()) zelf implementeert.
* Als je zelf nog zaken wil toevoegen (omdat je denkt dat ze nodig zijn) die niet in het class diagram staan, geef dat dan duidelijk met commentaar aan in de code

# Deel I: Bouw klassenstructuur

## Stap 0: Verander de welkomstboodschap (5 punten)

Verander de boodschap zodat je eigen naam, je studentnummer en je klas verschijnt.

Run de applicatie. De output moet er als volgt uit zien.

|  |
| --- |
| Welkom bij de Recepten app van MiC, geschreven door: <Naam> <Klas> <Studentnr> |

## Stap 1: WinkelProduct (5 punten)

Maak de klasse WinkelProduct volgens het class diagram. Houd hierbij rekening met de volgende eisen:

* Een winkelproduct beschrijft een product dat in een winkel te koop is.  
  De verkoopHoeveelheidInGram, geeft de omvang aan van het winkelproduct. Ook wordt het aantal calorieen per gram bijgehouden.
* Je hoeft nog geen rekening met Comparable te houden: dit komt pas in een volgende stap.

|  |
| --- |
| stap 1  WinkelProduct{naam='bief', CalorieenPerGram=35, verkoopHoeveelheidInGram=200}  WinkelProduct{naam='kip', CalorieenPerGram=25, verkoopHoeveelheidInGram=100} |

## Stap 2: Ingredient (15 punten)

Maak de klasse Ingredient volgens het class diagram. Houd hierbij rekening met de volgende eisen:

* Per Ingredient wordt bijgehouden wat de hoeveelheid  
  is, die nodig is voor het recept. Een ingredient bevat ook een link naar een winkelproduct waar het ingredient in zit.
* de methode 'geefCalorieen()' berekend het aantal calorieen van dit ingredient op basis van het aantal gram, en het opgegeven winkelproduct.

*<tentamen gaat verder op volgende pagina>*

## Stap 3: klassen Groente, Vlees en Vloeistof (5+5+10 = 20 punten)

Maak de subklassen Groente , Vlees en Vloeistof volgens het class diagram. Houd hierbij rekening met de volgende eisen:

* Implementeer zelf een all-args constructor. De volgende regels gelden
  + voor 'Vloeistof' wordt de hoeveelheidInGram berekend als 1.1 \* volumeInCm3
* De volgende regels gelden voor het berekenen van de bereidingstijd:
  + Groente: bereidingstijd is 5 minuten per 100 gram
  + Vlees: bereidingstijd is ½ minuut per stukje + 1 minuut per 100 gram;
  + Vloeistof: bereidingstijd is 3 minuten per cm3 volume.

|  |
| --- |
| stap 2 en 3  Vlees{hoeveelheidInGram=400, aantalStukjes=8}  Vlees{hoeveelheidInGram=250, aantalStukjes=5}  Groente{hoeveelheidInGram=240}  Groente{hoeveelheidInGram=400}  Vloeistof{hoeveelheidInGram=330, volumeInCm3=300.0} |

## Stap 4: klasse BoodschappenLijst (5 punten)

Maak de klasse BoodschappenLijst volgens het class diagram. Houd hierbij rekening met de volgende eisen:

* Methode printGesorteerdeLijst() hoeft nog niet geimplementeerd te worden. Dit gebeurt in een latere stap.

|  |
| --- |
| BoodschappenLijst{boodschappen=[WinkelProduct{naam='bief', CalorieenPerGram=35, verkoopHoeveelheidInGram=200}, WinkelProduct{naam='kip', CalorieenPerGram=25, verkoopHoeveelheidInGram=100}, WinkelProduct{naam='zalm', CalorieenPerGram=50, verkoopHoeveelheidInGram=75}, WinkelProduct{naam='tomaat', CalorieenPerGram=5, verkoopHoeveelheidInGram=30}, WinkelProduct{naam='sla', CalorieenPerGram=0, verkoopHoeveelheidInGram=50}, WinkelProduct{naam='bloemkool', CalorieenPerGram=35, verkoopHoeveelheidInGram=120}, WinkelProduct{naam='wijn', CalorieenPerGram=70, verkoopHoeveelheidInGram=180}]} |

*<tentamen gaat verder op volgende pagina>*

## Stap 5: Klasse Recept (20 punten)

Maak de klasse Recept volgens het class diagram. Houd hierbij rekening met de volgende eisen:

* Recepten bestaan uit een lijst van ingredienten, en een boodschappenlijst die in de winkel gekocht moeten worden.
* methode *geefCalorieen()* berekend het totaal aantal calorieen van het recept. Dit totaal is de som van de calorieen van alle ingredienten.
* Bij het toevoegen van een ingredient (via *'voegIngredientToe'*), wordt gekeken naar de hoeveel gram die nodig is voor het ingredient. Met die benodigde hoeveelheid van het ingredient, wordt berekend hoeveel winkelproducten nodig zijn. Hiertoe wordt de methode *'berekenBenodigdeAantalWinkelProducten'* gebruikt. Er wordt (uiteraard) afgerond naar boven, omdat je alleen hele WinkelProducten kunt kopen.
* Deze berekende aantal winkelproducten worden in de boodschappenlijst geplaatst.
* Dus: bij het toevoegen van een ingredient, wordt ook altijd 1 of meer winkelproducten toegevoegd aan de boodschappenlijst.

Gebruik stap5 om je implementatie te testen. De output moet er als volgt uitzien.

|  |
| --- |
| Recept{naam='vleesdiner', aantal ingredienten=2, aantal in boodschappenLijst=5}  boodschappenlijst recept  bief  bief  kip  kip  kip  Recept{naam='alleengroente', aantal ingredienten=1, aantal in boodschappenLijst=8}  boodschappenlijst recept  bief  bief  kip  kip  kip |

*<tentamen gaat verder op volgende pagina>*

# Deel II: Gebruik klassen, bestand en database

## Stap 6: Sorteer en print alle WinkelProducten (10 punten)

De medewerkers

Voer de volgende stappen uit:

* Zorg dat de klasse WinkelProduct de interface Comparable implementeert (zie class diagram) en override vervolgens de methode compareTo().
  + Sorteer de WinkelProduct objecten in BoodschappenLijst op de attributen naam, en vervolgens op verkoopHoeveelheidInGram en aantalCalorieenPerGram.
* Override de methode toString() in de klasse BoodschappenLijst en voeg de sortering hieraan toe.

Gebruik stap6 om je implementatie te testen. De output moet er ongeveer als volgt uitzien.

|  |
| --- |
| bief  bloemkool  kip  sla  tomaat  wijn  zalm |

## Stap 7: Print de boodschappenlijst naar een bestand (10 punten)

De boodschappenlijst met alle winkelproducten dient weggeschreven te worden naar een bestand. Maak een klasse BoodschappenPrinter in package service. Voeg methode printBoodschappenNaarFile(BoodSchappenLijst lijst, String filenaam) toe, die de lijst naar een TXT bestand uitprint.

* Implementeer de methode printBoodschappenNaarFile(BoodSchappenLijst lijst, String filenaam) ()in klasse BoodschappenPrinter. Print alleen de naam van het winkelproduct.
* Zorg dat in de methode een tekst bestand gemaakt wordt met de opgegeven naam. Sla dit bestand op in een folder in het project (bijvoorbeeld de folder ‘resources’, of in de root van je project).

*<tentamen gaat verder op volgende pagina>*

## Stap 8: Schrijf een boodschappenlijst naar de database (10 punten)

Je hebt een sql-script gekregen (recepten.sql, in root van project) waarin een database genaamd ‘recepten' wordt gemaakt. Let wel: Voor ingredienten er is geen subtypering toegepast. Dat wil zeggen: alle attributen van de Java klasse ‘Ingredienten' en alle child klassen zijn in één tabel 'Ingredient' opgenomen.

In je project staat al een DBaccess klasse met de juiste properties om een connectie te maken met de database.

Maak een WinkelProductPerReceptDAO klasse die het mogelijk maakt om een winkelproduct aan een recept toe te voegen. Dit is hetzelfde wat er in JAVA gebeurt als je een WinkelProduct object aan een boodschappenlijst toevoegt.

* Voeg toe methode public void voegWinkelProductToeAanRecept(WinkelProduct product, Recept recept). In deze methode moeten de juiste records in de tabel 'WinkelProductenPerRecept' worden geplaatst.

|  |
| --- |
|  |

# Let op de code conventions (10 punten)

* Zorg dat je naam en het doel bij elke class bovenin staan (ICC #1).
* Gebruik de juiste inspringing (indentation) bij de lay-out (ICC #2).
* Let op juist gebruik hoofdletters en kleine letters (ICC #3).
* Gebruik goede namen (ICC #4).
* Vermijd magic numbers (ICC#5).
* Voeg waar nodig commentaar toe die inzicht geven in je code (ICC#7).
* Vermijd dode code (ICC #8).
* Denk aan encapsulation (ICC #9).