hogeschool Windesheim

Techniek – HBO-ICT

Naam: Studentnummer: Klas:

HP JAVA Praktijk (110246)

Vakcode	: ICT.P.JAVA2.V20 (ICT.P.JAVA.V19/18/17) (t1)
Datum	: donderdag 10 juni 2021
Tijd	: 14.30 - 17.30 uur

Klas:	Lokaal:	Aantal:
ICTM2a t/m i, ICTM2n t/m r, ICTM2tt	volgt	231,6

Opgesteld door	: Wilco Moerman
Docenten	: WPH01, LNR08, DSW01, DFG01, NMJ01, MNC07 VEE02, RWM02, FAP02, CNW01, SSW02, KGW01, DAS01, CSI01
Gecontroleerd door	: Wietske Doornbos, Wouter Keuning
Rekenmachine	: alle rekenmachines toegestaan
Literatuur	: alles: internet, boeken
Overige hulpmiddelen	: laptop
Opgaven inleveren	: ja

CONTROLEER VOORAF DE VOLGENDE GEGEVENS:

Dit tentamen bevat:

5 opgaves

14 genummerde pagina's

Waarschuw de surveillant als één van deze aantallen niet klopt!

Studentnummer	Naam	Klas	Cijfer
Tijd van inleveren:			

De toets, de punten, etc.

Het gebruik van telefoons, social media, forums, dropbox en alles wat je in contact met anderen kan brengen is tijdens de toets niet toegestaan.

Het gebruik van internet om informatie op te zoeken is wel toegestaan.
Je mag dus wel zoeken/googlen.
En je mag bv. wel iets lezen op een forum zoals *Stackoverflow*, maar je mag er geen vragen stellen.

In totaal zijn **100** punten te behalen. Het eindcijfer wordt verkregen door de behaalde punten te delen door **10**. Het laagst te behalen cijfer is een 1, het hoogste een 10.

Voorbereiding

Alle code uit deze toets kun je vinden op ELO in de folder **"inleverpunt praktijk 10 juni"** in het bestand **`"startsituatie_praktijk_10-juni-2021.zip"`**.

De gegeven voorbeeldcode in deze toets is geen volledige test, maar geeft een *voorbeeld* van de werking. Je zult zelf moeten testen/onderzoeken of je code alles doet wat het moet doen volgens de vraag.

De voorbeeldcode staat in de klasse `Main` per onderdeel in een losse `main(...)`-methode die je uit comments kunt halen als je 'm wilt gebruiken.

De prints in de toets zijn bedoeld om de *flow* van het programma goed te kunnen volgen. Ze moeten worden toegevoegd als dat in de opgave staat, maar een spelfoutje of een spatie teveel is dus geen probleem.

Als je wilt, kun je op de papieren toets aantekeningen maken, of het klassendiagram losmaken van de rest, zodat je het naast een vraag kunt leggen.

Inleiding

Vanwege Corona-maatregelen heeft de horeca maar beperkte capaciteit en is ze gedwongen om van elke bezoeker contactgegevens bij te houden.

In deze toets modelleer je een kroeg en de bijbehorende wachtrij die buiten staat.

Hieronder zie je een screenshot van het eindresultaat:

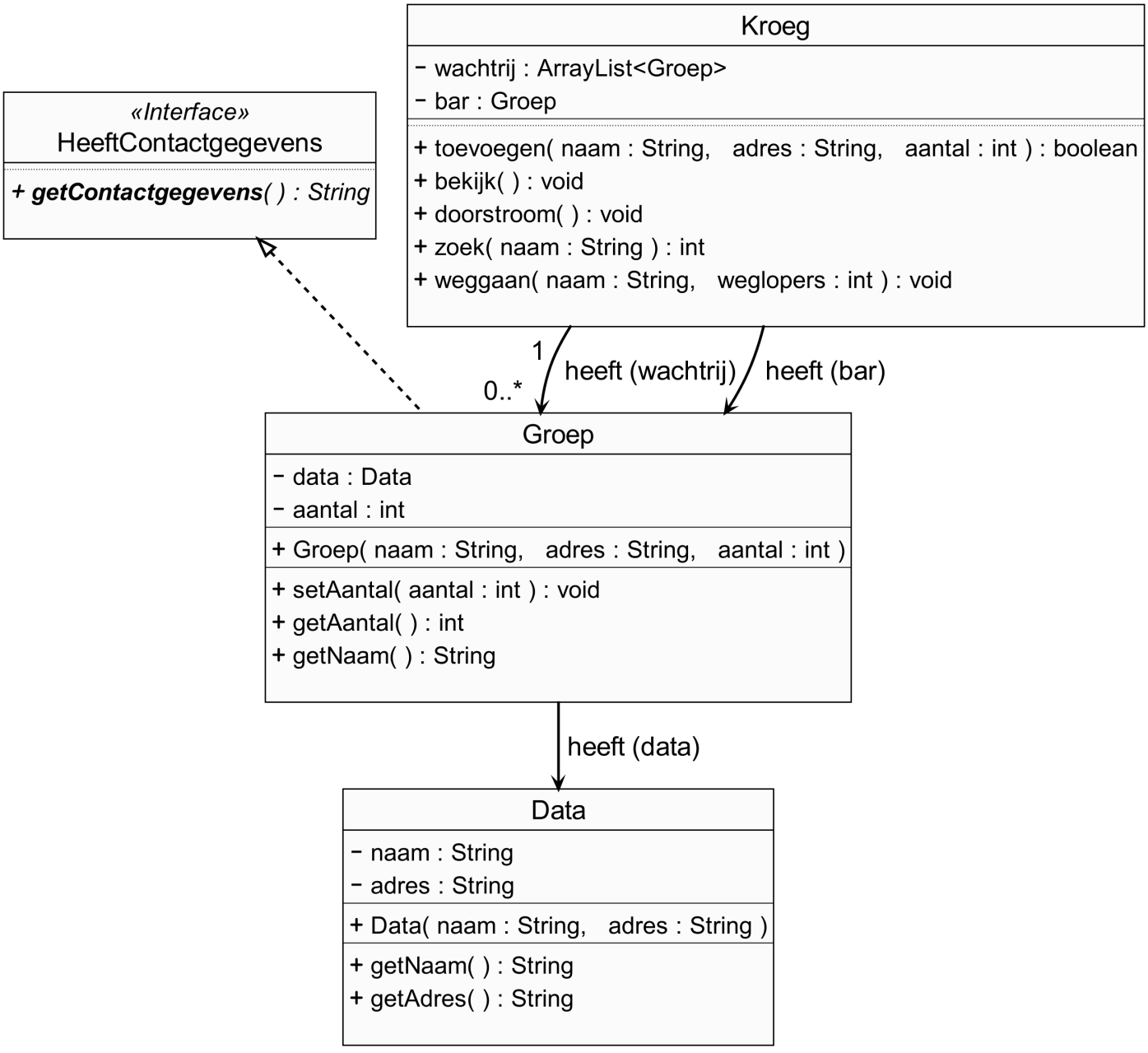
Klassendiagram

Hieronder volgt het klassendiagram voor opgave **1** en **2**. Overgeërfde methodes en methodes vanwege interfaces zijn hier **niet** in aangegeven (ook niet als in een opgave wel gevraagd wordt ze te implementeren). Abstracte methodes zijn ***vetgedrukt en cursief***. Interfaces zijn aangegeven met *<<Interface>>*.

Constructor-parameters die dezelfde naam hebben als attributen, worden in die attributen opgeslagen.

Voor elke methode, constructor en attribuut geldt dat ze **exact** zoals in het klassendiagram aangegeven, moeten worden gemaakt. Afwijken kan puntenaftrek opleveren.

Je mag **geen** andere *methodes* of *attributen* toevoegen dan dat er in het klassendiagram staan, behalve als ze `private` zijn. Als je niet weet hoe je iets moet oplossen volgens de regels, dan kun je er vanaf wijken, zodat je verder kunt met de volgende vraag.



Opgave 1: de wachtenden [20 punten]

a) [10 punten]
Maak klasse `Data` volgens het klassendiagram en zorg ervoor dat `Data`-objecten uitgeprint worden zoals in onderstaand voorbeeld.

Voor de `naam` gelden de volgende eisen:

- de lengte van de `String` is minimaal 2
- de `String` is niet gelijk aan de tekst `"??"`.
- de `String` is niet gelijk aan `null`.
- als default wordt bij ongeldige input `"onbekend"` gebruikt.

Data aanmaken en printen (verwachte output staat in comments):

```
Data d = new Data( "Wilco", "0800-JAVA" );
System.out.println( d );                    // Wilco (0800-JAVA)
Data d2 = new Data( "??", "ergens" );
System.out.println( d2 );                   // onbekend (ergens)
```

b) [10 punten]
Maak de interface `HeeftContactgegevens`

Maak alles van de klasse `Groep` overeenkomstig het klassendiagram:

- de aan de constructor meegegeven `naam` en het `adres` worden in het `Data`-attribuut genaamd `data` opgeslagen en het meegegeven `aantal` in het `aantal`-attribuut.
- de methode die aanwezig moet zijn vanwege de interface, returnt de `toString()` van `data`.
- de `getNaam`-methode returnt de `naam` die in `data` is opgeslagen.
- de `toString()` van `Groep` gebruikt de attributen `data` en `aantal`.

Groep aanmaken en printen (verwachte output staat in comments):

```
Groep g = new Groep( "Jan", "Javaweg 1", 5 );
System.out.println( g );                        // Jan (Javaweg 1): 5
System.out.println( g.getContactgegevens() ); // Jan (Javaweg 1)
```

Opgave 2: de Kroeg [40 punten]

In deze opgave maak je de `Kroeg`, inclusief `wachtrij` die buiten de `Kroeg` staat te wachten.

Als een `Groep` mensen aankomt, moeten `naam` en `adres` geregisteerd worden waarna de `Groep` toegevoegd wordt aan de `wachtrij`.

Je mag er in deze toets vanuit gaan, dat er nooit meerdere Groepen zijn met dezelfde `naam`.

De `Kroeg` heeft `1` `bar` met plek voor `5` personen.

a) [5 punten]
Maak de klasse `Kroeg` volgens het klassendiagram:

- maak de attributen
- je mag een constructor aanmaken, maar het is niet verplicht.

De rest van de methodes worden in de volgende onderdelen gemaakt.

b) [15 punten]
Maak de `toevoegen`-methode:

- als het `aantal` kleiner dan **`1`** of groter dan **`5`** is, wordt niks toegevoegd maar een bericht geprint (namelijk: `"sorry, kan niet"`)
- als de `bar` vrij (`null`) is, dan wordt de nieuwe `Groep` meteen opgeslagen in `bar` (de groep kan meteen binnenkomen)
- maar als de `bar` niet vrij is, dan wordt de nieuwe `Groep` toegevoegd aan de `wachtrij`.
- de toe te voegen `Groep` wordt uiteraard gemaakt op basis van de 3 inputs.
- als er toegevoegd is, wordt `true` gereturned, en anders `false`.

Maak ook de `bekijk`-methode. Deze print informatie over de `Kroeg`.

De werking van de `bekijk`- en `toevoegen`-methodes kun je zien in onderstaand voorbeeld.

werking van toevoegen en bekijk (verwachte outputs in de comments):

```
Kroeg k = new Kroeg();
k.bekijk();
                                                    // bar: leeg
                                                    // wachtrij: leeg
k.toevoegen( "Haantje", "de Voorste", 2 );
k.toevoegen( "Jan", "ergens", 3 );
k.toevoegen( "Anna", "thuis", 2 );
k.toevoegen( "P. Party", "0800-FEEST", 17 ); // sorry, kan niet
k.bekijk();
                                                    // bar: Haantje (de Voorste): 2
                                                    // wachtrij:
                                                    // * plek 0: Jan (ergens): 3
                                                    // * plek 1: Anna (thuis): 2
```

c) [5 punten]
Op een gegeven moment moet de `Groep` aan de `bar` weg, en kan een nieuwe `Groep` aanschuiven. Dit wordt gedaan door de methode `doorstroom`.

Maak de `doorstroom`-methode:

- de eerste `Groep` wordt uit de `wachtrij` gehaald in het `bar`-attribuut opgeslagen.
- je kunt een element uit `ArrayList lijst` halen met de methode `lijst.remove(int index)`

Hieronder zie je de werking van de `doorstroom`-methode:

werking van doorstroom (verwachte outputs in de comments):

```
Kroeg k = new Kroeg();
k.toevoegen( "Haantje", "de Voorste", 2 );
k.bekijk();
                                        // bar: Haantje (de Voorste): 2
                                        // wachtrij: leeg
k.doorstroom();
k.bekijk();
                                        // bar: leeg
                                        // wachtrij: leeg
k.toevoegen( "Jan", "ergens", 3 );
k.toevoegen( "Anna", "thuis", 4 );
k.toevoegen( "Rik", "r@ik.nl", 1 );
k.bekijk();
                                        // bar: Jan (ergens): 3
                                        // wachtrij:
                                        // * plek 0: Anna (thuis): 4
                                        // * plek 1: Rik (r@ik.nl): 1
k.doorstroom();
k.bekijk();
                                        // bar: Anna (thuis): 4
                                        // wachtrij:
                                        // * plek 0: Rik (r@ik.nl): 1
```

d) [15 punten]

Op basis van de opgegeven `naam` moet de `index` (`plek`) van de `Groep` met die `naam` in de `wachtrij` bepaald worden. Maak hiervoor de `zoek`-methode:

- als er een `Groep` met de juiste `naam` aanwezig is, wordt de `plek` van die `Groep` gereturned.
- als er geen `Groep` met de juiste `naam` gevonden wordt, returnt de methode **`-1`**.

Let op: Als het niet lukt om de `zoek`-methode werkend te krijgen, laat deze dan `"zoek"` printen en **0** returnen, zodat je verder kunt met de `weggaan`-methode. Je kunt daar dan doen alsof de `zoek`-methode de juiste index gevonden heeft.

Zo nu en dan verandert de `wachtrij` omdat er mensen geen zin meer hebben om in de rij te staan en weggaan.

Maak de `weggaan`-methode:

- deze heeft als inputs de `naam` van de gezochte `Groep` en het aantal `weglopers`.
- als er een `Groep` met de juiste `naam` aanwezig is en `weglopers` groter dan 0 is, dan wordt het `aantal` aangepast
- als het nieuwe `aantal` kleiner dan **1** is, moet de `Groep` uit de `wachtrij` verwijderd worden.
- in alle andere gevallen wordt niks gedaan.

werking van weggaan (verwachte outputs in de comments):

```
Kroeg k = new Kroeg();
k.toevoegen( "Haantje", "de Voorste", 2 );
k.toevoegen( "Anna", "thuis", 4 );
k.toevoegen( "Jan", "ergens", 3 );
k.toevoegen( "Rik", "r@ik.nl", 3 );
k.toevoegen( "L. Aatste", "achteraf", 1 );
k.bekijk();
                                        // bar: Haantje (de Voorste): 2
                                        // wachtrij:
                                        // * plek 0: Anna (thuis): 4
                                        // * plek 1: Jan (ergens): 3
                                        // * plek 2: Rik (r@ik.nl): 3
                                        // * plek 3: L. Aatste (achteraf): 1

System.out.println( k.zoek( "Anna" ) );     // 0
k.weggaan( "Anna", 1 );
k.weggaan( "Jan", 100 );
k.weggaan( "Rik", -20 );
k.bekijk();
                                        // bar: Haantje (de Voorste): 2
                                        // wachtrij:
                                        // * plek 0: Anna (thuis): 3
                                        // * plek 1: Rik (r@ik.nl): 3
                                        // * plek 2: L. Aatste (achteraf): 1
```

Opgave 3: GUI, het scherm [15 punten]

Vanaf deze opgave is er **geen** klassendiagram.
Je mag **getters** toevoegen aan klassen uit opgave 1 en 2 *voor gebruik in de GUI*.

Als je een methode uit opgave 1 of 2 nodig hebt die niet werkt, zorg er dan voor dat die methode wel compileert en de naam van de methode uitprint, zodat je 'm in de GUI kunt gebruiken en ook ziet dat er wat gebeurt.

a) [5 punten]
Maak een GUI zoals in onderstaand screenshot:

Voor de GUI geldt:

- heeft een `Kroeg`-object als attribuut dat in de constructor aangemaakt wordt.
- `FlowLayout`, ongeveer `650` bij `350` pixels.
- de applicatie wordt afgesloten door op het kruisje te klikken.
- de *titel* bevat je eigen naam en studentnummer
- de `"toevoegen"`-knop is in het begin groen.
- je kunt een `knop` een kleur geven met: `knop.setBackground(Color.green)`

b) [5 punten]
Maak de volgende knoppen werkend:

- als op de **"toevoegen"**-knop geklikt wordt, moet een `Groep`-object toegevoegd worden aan het `Kroeg`-attribuut m.b.v. de `toevoegen`-methode, op basis van de waardes in de tekstvelden (foutafhandeling komt bij **3(c)**).
- als op de **"doorstroom"**-knop geklikt wordt, moet de `doorstroom`-methode van het `Kroeg`-attribuut uitgevoerd worden.

Elke keer dat op 1 van de 3 knoppen geklikt wordt, moet het `Kroeg`-attribuut geprint worden met de `bekijk`-methode.

De kleur van de **"toevoegen"**-knop geeft de status van het toevoegen aan:

- als het toevoegen van een `Groep` geslaagd is, dan wordt deze knop **groen**
- als het niet geslaagd is (`aantal` te groot of te klein), dan wordt deze knop **oranje**

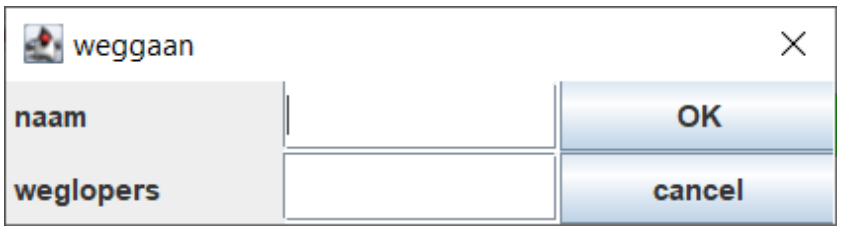
c) [5 punten]
Als je bij het tekstveld voor "aantal" *geen* getal invoert en toch wilt toevoegen, volgt een bepaald soort `Exception`:

- los dit op met een *zo specifiek mogelijke* `try ... catch`.
- er wordt in dit geval geen `Groep` toegevoegd aan het `Kroeg`-attribuut.
- de **"toevoegen"**-knop wordt in dit geval **rood**.

Opgave 4: GUI, dialoog [10 punten]

Soms gaan mensen uit de wachtrij weer weg. Via een dialoog ga je de `wachtrij` van het `Kroeg`-attribuut van de GUI veranderen.

a) [5 punten]
Maak dialoog zoals in het screenshot is weergegeven.



De dialoog verschijnt als de **"weggaan"**-knop wordt ingedrukt:

- is modaal en heeft als titel `"weggaan"`
- wordt afgesloten als op het kruisje of de knoppen geklikt wordt
- alle attributen zijn *private*

b) [5 punten]
Als er op de **"OK"**-knop geklikt wordt:

- dan wordt de wijziging doorgevoerd m.b.v. de `weggaan`-methode.
- als er geen `int` ingevuld wordt bij `aantal`, gebeurt er uiteraard niks
- de GUI/dialoog mag niet crashen.

Opgave 5: GUI, tekenen [15 punten]

Hierboven zie je een voorbeeld van het eindresultaat van de GUI, nadat er een paar Groepen toegevoegd zijn met de "`toevoegen`"-knop.

Hieronder staat wat getekend moet worden (dit hoeft niet op de pixel nauwkeurig):

- de linkerkant van het scherm is de `Kroeg`. Binnen in de `Kroeg` het licht (wit), maar de rest van het scherm is zwart, dat is 'buiten' waar het donker is.
- zet de woorden `Kroeg` en `wachtrij` op de juiste plek neer.
- als er een `Groep` aan de `bar` in de `Kroeg` zit, dan worden er evenveel groene cirkels getekend, als het `aantal` van de `Groep`.
- als de `bar` leeg is, wordt `"leeg hier!"` getekend in het `Kroeg`-gedeelte.
- elke `Groep` in de `wachtrij` wordt weergegeven met rode cirkels (evenveel als het `aantal` van die `Kroeg`)
- de cirkels hebben een diameter van 30 pixels.
- boven elke `Groep` wordt het resultaat van de `getContactgegevens`-methode getekend. (let op: `g.setColor(...)` werkt ook voor tekst)

Tips:

- met `setPreferredSize(new Dimension(...))` stel je de afmetingen van een `JPanel` in.
- met `g.setFont(new Font("default", Font.PLAIN, 14))` kun je het font wat groter/leesbaarder maken.

Einde Tentamen

Maak een archief (.zip of .rar) van je **java**-bestanden. Geef het archief de volgende naam: `"java-praktijk_Voornaam_Achternaam_studentnummer.zip"` (of `...rar`).

Upload je `zip/rar`-bestand op ELO in het inleverpunt in de folder genaamd: **`"inleverpunt praktijk 10 juni"`**

LET OP: Kies bij het uploaden op ELO voor de **inleveren** (of **submit**)-knop!