hogeschool Windesheim

Techniek – HBO-ICT

Naam: Studentnummer: Klas:

Practicum herk. JAVA Praktijk (119916)

Vakcode	: ICT.P.JAVA2.V21 (ICT.P.JAVA2.V20/V19/V18/V17) (t1)
Datum	: donderdag 16 juni 2022
Tijd	: 14:30 - 17:30 uur

Klas:	Lokaal:	Aantal:
ICTM2a t/m p, ICTM2tt	…	205
	…	
	…	
	…	
	…	

Opgesteld door	: Aminah Balfaqih; Wilco Moerman
Docenten	: WPH01; KNJ24; RWM02; SSM36; LNR08; KEK01; MNC07; VEE02; CNW01; HZJ40; BHA40;
Gecontroleerd door	: Kevin de Korte; Jairo Hernandez
Rekenmachine	: alle rekenmachines toegestaan
Literatuur	: alles (boeken, internet, aantekeningen)
Overige hulpmiddelen	: laptop
Opgaven inleveren	: ja

Belangrijk voor surveillanten:
Dit is een toets die 3 uur duurt. Daarom is met de Examencommissie afgesproken dat het voor studenten toegestaan is om tijdens het tentamen naar het toilet te gaan.

CONTROLEER VOORAF DE VOLGENDE GEGEVENS:

Dit tentamen bevat:

5 opgaves

20 genummerde pagina's

Waarschuw de surveillant als één van deze aantallen niet klopt!

Studentnummer	Naam	Klas	Cijfer
Tijd van inleveren:			

De regels en de punten

Het gebruik van telefoons/social media/forums/dropbox en alles wat je in contact brengt met anderen, 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 cijfer is het aantal behaalde punten gedeeld door 10. Het laagst te behalen cijfer is een 1, het hoogste een 10.

Voorbereiding

Alle in de toets getoonde code en het klassendiagram vind je op **ELO** in de folder **`"inleverpunt praktijk 16-06-2022"`** in **`"startcode_praktijk_16-06-2022.zip".`**

De codes met voorbeelden zijn ***geen volledige tests.*** Controleer zelf of je code *alles* doet wat de vraag staat. De voorbeelden staan in de klasse `Main` (in de `.zip`) in uitgecommente `main(...)`-methodes. Deze `Main`-klasse wordt *niet* nagekeken.

Als een regel code iets *print,* staat dat erachter *in commentaar,* na **`//>>`**

Voorbeeld: onderstaande System.out.println("Hoi") heeft dus output "Hoi"

```
// dit is gewoon commentaar
System.out.println("Hoi"); //>> Hoi
int x = 10; // ook gewoon commentaar
```

Een spelfoutje of een spatie teveel in een `toString()` of `print` is geen probleem.

Je mag op de papieren toets aantekeningen maken. Ook mag je het klassendiagram losmaken van de rest om het naast een vraag te leggen. **nakijkmodel:**
In deze pdf staat het nakijkmodel beschreven (typefouten voorbehouden, De tests die uitgevoerd zijn op de code, zijn bepalend).

Niet elke uitzondering en elk detail is hier opgeschreven, om deze tekst leesbaar te houden.

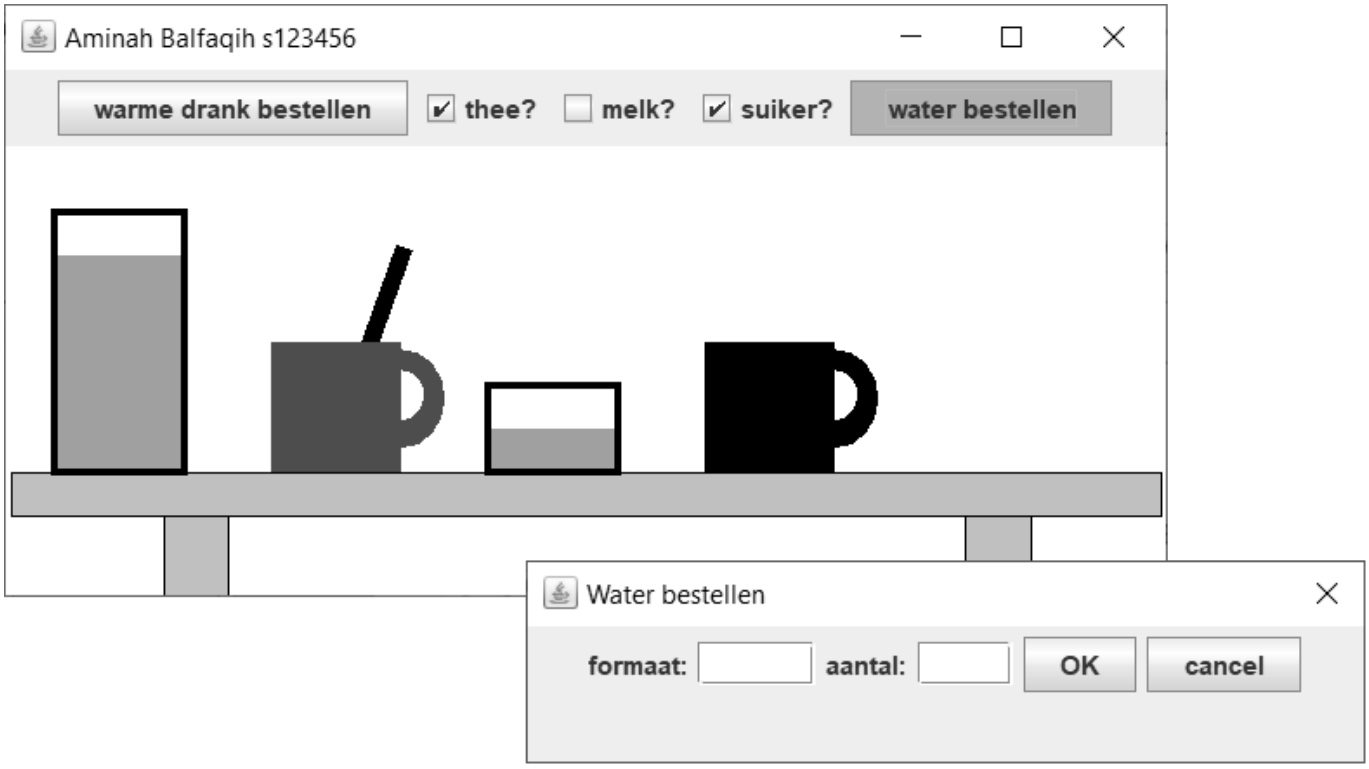
Binnen een vraag (bv. 1(a)) is meestal een **onderverdeling** gemaakt in losse onderdelen. Als voor een vraag bv. 15 punten te behalen waren, dan zijn die punten bv. opgesplitst naar 10 punten voor een bepaald onderdeel en 5 voor een ander onderdeel van wat gemaakt moest worden. Uiteraard kun je niet minder dan 0 punten op zo'n onderdeel krijgen ook al heb je volgens de nakijkregels meer aftrekpunten dan er punten in dat onderdeel zaten.

Controleer goed of een antwoord per ongeluk fout is gerekend vanwege een typo (bv. de kleur "white" in plaats van "wit"), want dat is uiteraard niet de bedoeling (de plugin die automatisch je cijfer voor Nederlands aanpast, komt volgend jaar pas **;-)**

Inleiding

Voor de locale horeca ga je een Starbucks-simulatie schrijven, waar je water, thee en koffie kunt bestellen.

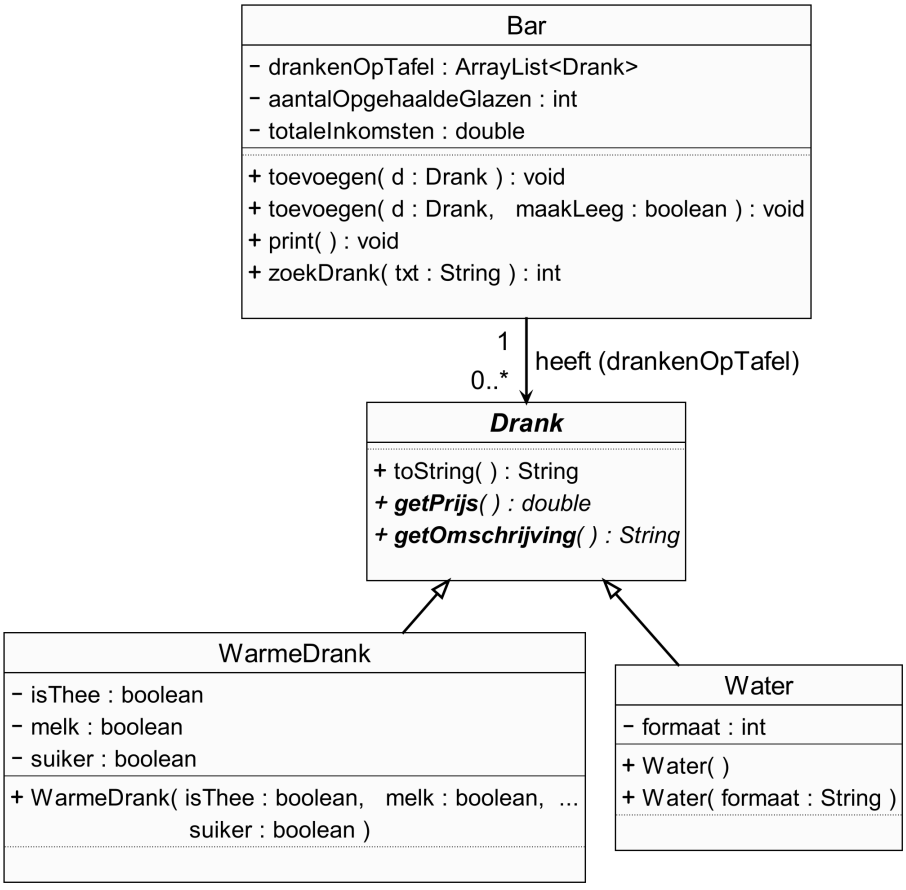
Het eindresultaat toont de dranken op tafel, en gaat er als volgt uitzien:
(kleurenversie zit in `"startcode_praktijk_16-06-2022.zip"`)



Klassendiagram

Hieronder volgt het klassendiagram voor **opgave 1** en **2.** Voor elke methode, constructor, variabelenaam en attribuut geldt dat ze **exact** moeten worden gemaakt zoals in het klassendiagram aangegeven. Hiervan afwijken kan puntenaftrek opleveren.

Abstracte klassen en methodes zijn ***vetgedrukt en cursief.*** Constructor-parameters die dezelfde naam hebben als attributen, worden in die attributen opgeslagen. Je mag **geen** andere *constructors, methodes* of *attributen* toevoegen voor **opgave 1 en 2** (behalve als ze `private` zijn, of *nodig* voor de GUI). Afwijken van deze regel levert puntenaftrek op.



Opgave 1: dranken [25 punten]

In deze opgave maak je de abstracte `Drank`-klasse en twee concrete klassen (`WarmeDrank` en `Water`).

a) [15 punten]
Maak klasse **`Water`** volgens het klassendiagram.

- er zijn 2 constructors. Bij de ene geef je input `formaat` (een `String`) mee. Deze input wordt dan vertaald wordt naar een `int` en opgeslagen in het *attribuut* `formaat`
- vertalen gebeurt als volgt:

String-input formaat is:	int-attribuut formaat wordt:
`"klein"`	1
`"normaal"`	2
`"groot"`	3

- als de input iets anders is of **`null`**, dan print de constructor **`"Hebben we niet!"`**. De waarde van attribuut `formaat` wordt dan **`1`**
- bij de constructor zonder inputs, wordt het attribuut `formaat` altijd **`2`**

`Water` erft de methodes **`getPrijs`** en **`getOmschrijving`** van `Drank`:

- `Water` is gratis, dus `getPrijs` returnt **`0`**
- `getOmschrijving()` returnt **`"een ... glas water"`** met op de `...` de tekst `"klein"`, `"normaal"` of `"groot"`
- klasse `Water` heeft zelf **geen** `toString()`

Klasse `Water` erft van de klasse **`Drank`**. Maak `Drank` volgens het klassendiagram. De **`toString()`** van `Drank` gebruikt `getOmschrijving()` en `getPrijs()`.

werking van Drank en Water.
Output staat in comments, herkenbaar aan //>>

```
Water w = new Water("groot");
System.out.println(w);                      //>>  een groot glas water (0.0)
System.out.println(w.getOmschrijving());    //>>  een groot glas water
System.out.println(w.getPrijs());           //>>  0.0

System.out.println(new Water());            //>>  een normaal glas water (0.0)

Water wXXL = new Water("XXL");              //>>  Hebben we niet!
System.out.println(wXXL);                   //>>  een klein glas water (0.0)
```

nakijkmodel:
3 punten: Drank is abstract class:

- 2 punten aftrek: Drank heeft een constructor
- 1 punt aftrek: Drank niet abstract
- 1 punt aftrek: getOmschrijving()-methode niet abstract
- 2 punten aftrek: getPrijs()-methode niet abstract

3 punten: toString() van Drank
6 punten: constructors van Water:

- 2 punten aftrek: attribuut formaat niet private
- 3 punten aftrek: aanwezigheid van constructors die niet in diagram stonden
- 2 punten aftrek: formaat krijgt niet waarde (2) bij Water()
- 3 punten aftrek (per keer): Water(String)-constructor vertaalt input (String) niet goed naar formaat (int)
- 4 punten aftrek: Water(String)-constructor gebruikt geen equals/switch maar == en gaat mis als er een nieuw String-object als input wordt gebruikt
- 1 punt aftrek: Water(String)-constructor print geen foutmelding bij ongeldige input
- 3 punten aftrek: Water(String)-constructor vertaalt ongeldige input niet naar 1
- 3 punten aftrek: Water(String)-constructor gaat fout bij input null

3 punten: de getters:

- 3 punten aftrek: getPrijs() returnt geen 0
- 3 punten aftrek (per keer): getOmschrijving() werkt niet

b) [10 punten]
Maak klasse **`WarmeDrank`** volgens het klassendiagram. De klasse heeft drie `booleans` om aan te geven of het om thee **`(isThee`** = `true`) of koffie **`(isThee`** = `false`) gaat en of er **`melk`** en/of **`suiker`** bij zitten.

De **`getPrijs`**-methode van `WarmeDrank` werkt als volgt:

- **Thee** is **`1`** euro
- **koffie** is **`2`** euro
- voor **suiker** wordt **`0.25`** euro extra gerekend
- voor **melk** wordt **`0.50`** euro extra gerekend

Let bij de **`getOmschrijving`**-methode op de **`"en"`** tussen melk en suiker!

Klasse `WarmeDrank` heeft zelf **geen** `toString()`.

werking van WarmeDrank

```
WarmeDrank thee = new WarmeDrank(true, false, true);
System.out.println(thee);              //>>  thee met suiker (1.25)
String bes = thee.getOmschrijving();
System.out.println(bes);               //>>  thee met suiker

WarmeDrank koffieZwart = new WarmeDrank(false, false, false);
System.out.println(koffieZwart);       //>>  koffie (2.0)

WarmeDrank koffieMetMelk = new WarmeDrank(false, true, false);
System.out.println(koffieMetMelk);     //>>  koffie met melk (2.5)

// let op de "en" tussen melk en suiker!
WarmeDrank koffieMetAlles = new WarmeDrank(false, true, true);
System.out.println(koffieMetAlles);    //>>  koffie met melk en suiker (2.75)
```

nakijkmodel:
2 punten: private attributen van WarmeDrank:

- 1 punt aftrek (per keer): attribuut niet private

3 punten: WarmeDrank(boolean,boolean,boolean)-constructor:

- 2 punten aftrek (per keer): WarmeDrank constructor slaat inputs niet op in attributen

5 punten: getPrijs() en getOmschrijving():

- 1 punt aftrek (per keer): getOmschrijving() werkt niet
- 1 punt aftrek: getOmschrijving() werkt niet
- 1 punt aftrek (per keer): getPrijs() werkt niet

Deze pagina is expres leeg gelaten

Opgave 2: Bar [30 punten]

In deze opgave gaat het over de `Bar`-klasse. Een `Bar` heeft een `ArrayList` (`drankenOpTafel`) waarin bijgehouden wordt welke dranken er op tafel staan.

Verder houdt `Bar` bij hoeveel lege glazen er van de tafel weggehaald zijn (`aantalOpgehaaldeGlazen`), en wat de totale inkomsten zijn (`totaleInkomsten`).

a) [5 punten]
Maak klasse `Bar` met de attributen.
Je mag een `Bar()`-constructor maken, maar het is niet verplicht.

nakijkmodel:
5 punten: Aanwezigheid van klasse Bar met de attributen:

- 4 punten aftrek: drankenOpTafel niet aanwezig of niet met new geinitialiseerd
- 1 punt aftrek: drankenOpTafel niet private
- 2 punten aftrek: als aantalOpgehaaldeGlazen niet aanwezig is
- 1 punt aftrek: aantalOpgehaaldeGlazen niet private
- 2 punten aftrek: totaleInkomsten niet aanwezig
- 1 punt aftrek: totaleInkomsten niet private

b) [15 punten]
Maak de **`print`**-methode. Deze print de gegevens zoals je hieronder kunt zien.

werking van print

```
Bar bar = new Bar();
bar.print();  //>>  aantalOpgehaaldeGlazen = 0
              //>>  totaleInkomsten = 0.0

bar.toevoegen(new Water("groot"));
bar.toevoegen(new Water());
bar.print();  //>>  aantalOpgehaaldeGlazen = 0
              //>>  totaleInkomsten = 0.0
              //>>    1e: een groot glas water (0.0)
              //>>    2e: een normaal glas water (0.0)
```

Maak ook beide **`toevoegen`**-methodes:

- bij de ene `toevoegen`-methode kun je met een `boolean (maakLeeg)` aangeven of `drankenOpTafel` eerst leeggemaakt moet worden *voordat* de nieuwe `Drank` wordt toegevoegd
- de andere `toevoegen`-methode bepaalt zelf of `drankenOpTafel` leeggemaakt moet worden. Als er **`5`** of meer dranken op tafel staan, wordt de tafel leeggemaakt. Daarna wordt de nieuwe `Drank` toegevoegd

Tip: Met de `clear()`-methode maak je een `ArrayList` leeg.

Beide `toevoegen`-methodes moeten ook bijhouden hoeveel dranken er van tafel zijn gehaald (in **`aantalOpgehaaldeGlazen).`**

Ook houden de methodes bij wat de totale inkomsten **`(totaleInkomsten)`** zijn: `totaleInkomsten` is de optelling van de *prijzen* van alle bestelde dranken.

voorbeeld van toevoegen(Drank d, boolean maakLeeg)

```
Bar bar = new Bar();
// de waarde van totaleInkomsten verandert door toevoegen(...)
bar.toevoegen(new WarmeDrank(false, true, false));
bar.toevoegen(new WarmeDrank(true, true, true));

// niet leegmaken (... , false)
bar.toevoegen(new Water(), false);
bar.print();  //>>  aantalOpgehaaldeGlazen = 0
              //>>  totaleInkomsten = 4.25
              //>>    1e: koffie met melk (2.5)
              //>>    2e: thee met melk en suiker (1.75)
              //>>    3e: een normaal glas water (0.0)

// leegmaken via de toevoegen(... , boolean) methode:
// er wordt true meegegeven, dus drankenOpTafel wordt eerst leeggemaakt.
// De waarde van aantalOpgehaaldeGlazen verandert door het leegmaken.
bar.toevoegen(new Water(), true);
bar.print();  //>>  aantalOpgehaaldeGlazen = 3
              //>>  totaleInkomsten = 4.25
              //>>    1e: een normaal glas water (0.0)
```

Hieronder volgt nog een voorbeeld van de andere `toevoegen`-methode.

voorbeeld van toevoegen(Drank d)

```
Bar bar = new Bar();
bar.toevoegen(new Water("groot"));
bar.toevoegen(new Water("groot"));
bar.toevoegen(new Water("normaal"));
bar.toevoegen(new Water("normaal"));
bar.toevoegen(new Water("klein"));

bar.print();  //>>  aantalOpgehaaldeGlazen = 0
              //>>  totaleInkomsten = 0.0
              //>>    1e: een groot glas water (0.0)
              //>>    2e: een groot glas water (0.0)
              //>>    3e: een normaal glas water (0.0)
              //>>    4e: een normaal glas water (0.0)
              //>>    5e: een klein glas water (0.0)

// Automatisch leegmaken omdat tafel te vol is: Dit is namelijk het 6e drankje.
// Dus de tafel wordt leegemaakt (en aantalOpgehaaldeGlazen aangepast).
// Daarna wordt de input toegevoegd.
bar.toevoegen(new WarmeDrank(true, true, true));

bar.print();  //>>  aantalOpgehaaldeGlazen = 5
              //>>  totaleInkomsten = 1.75
              //>>    1e: thee met melk en suiker (1.75)
```

nakijkmodel:

5 punten: de print()-methode:

- 2 punten aftrek: print() toont de statistieken niet
- 4 punten aftrek: print() print de drankenOpTafel niet
- 2 punten aftrek: volgnummers zijn niet juist (starten bij 1)

4 punten: toevoegen(Drank):

- 4 punten aftrek: toevoegen(Drank) voegt niks toe
- 3 punten aftrek: drankenOpTafel wordt niet leeggemaakt (bij ongeveer 5)
- 2 punten aftrek: drankenOpTafel wordt niet leeggemaakt bij size 5

2 punten: toevoegen(Drank, boolean):

- 2 punten aftrek: volgorde-probleem: toevoegen(Drank, true) maakt drankenOpTafel niet leeg of nadat Drank toegevoegd is

- 1 punt aftrek: toevoegen(Drank, false) maakt drankenOpTafel leeg

4 punten: bijhouden van de "statistieken":

- 2 punten aftrek: aantalOpgehaaldeGlazen wordt niet bijgehouden
- 2 punten aftrek: totaleInkomsten worden niet goed bijgehouden

c) [10 punten]

Maak de methode genaamd **`zoekDrank:`**

- deze zoekt in de `drankenOpTafel` naar de eerste `Drank` met een **`getOmschrijving()`** die hetzelfde is als de input
- als zo'n `Drank`-object aanwezig is, wordt de **plek** ervan gereturned
- als geen overeenkomende `Drank` wordt gevonden, wordt **`0`** gereturned
- plekken beginnen te tellen bij **`1.`**

voorbeelden van zoekDrank(...)

```
Bar bar = new Bar();
bar.toevoegen(new Water("groot"));
bar.print();                  //>>  aantalOpgehaaldeGlazen = 0
                              //>>  totaleInkomsten = 0.0
                              //>>    1e: een groot glas water (0.0)

// niet aanwezig:
int plek = bar.zoekDrank("thee met melk en suiker");
System.out.println(plek);  //>>  0

// ook niet aanwezig:
plek = bar.zoekDrank("water");
System.out.println(plek);  //>>  0

// wel aanwezig, op de eerste plek:
plek = bar.zoekDrank("een groot glas water");
System.out.println(plek);  //>>  1

bar.toevoegen(new WarmeDrank(true, true, true));
bar.print();                  //>>  aantalOpgehaaldeGlazen = 0
                              //>>  totaleInkomsten = 1.75
                              //>>    1e: een groot glas water (0.0)
                              //>>    2e: thee met melk en suiker (1.75)

// niet aanwezig:
plek = bar.zoekDrank("thee");
System.out.println(plek);  //>>  0

// wel aanwezig op de tweede plek:
plek = bar.zoekDrank("thee met melk en suiker");
System.out.println(plek);  //>>  2
```

nakijkmodel:

10 punten: werking van zoekDrank(String):

- 10 punten aftrek: zoeken gaat fout gaat bij iets wat aanwezig is (0-based of 1-based beide ok)
- 5 punten aftrek: zoeken fout gaat bij iets wat aanwezig is (tellen begint bij 1)

- 3 punten aftrek: zoeken gaat naar iets wat niet in de ArrayList zit, gaat fout (moet 0 returnen)
- 5 punten aftrek: zoeken gaat naar iets wat niet in de ArrayList zit, gaat fout (0 en -1)

Opgave 3: GUI, het scherm [10 punten]

Let op:
Voor de GUI-opgaves is er **geen** klassendiagram. Je kunt de namen van frames, dialogen, etc. zelf bepalen.

Als je een methode uit opgave 1 of 2 nodig hebt die **niet** (of niet goed) werkt, zorg er dan voor dat die methode **wel compileert** (kies indien nodig maar een geschikte return waarde). Op die manier kun je 'm vanuit de GUI gewoon aanroepen. Het is in de GUI-opgaves niet erg dat een methode uit opgave 1 of 2 niet goed werkt.

a) [5 punten]
Maak een GUI zoals in dit screenshot (hoeft niet op de pixel nauwkeurig).

- de GUI-titel bevat je **eigen naam en studentnummer**
- de GUI heeft een `Bar`-**attribuut**
- de constructor van je GUI heeft als enige input een `Bar`-object, dat dan in het `Bar`-attribuut wordt **opgeslagen**
- heeft een **`FlowLayout`** (ongeveer 550 bij 280 pixels)
- de applicatie wordt afgesloten d.m.v. het **kruisje**
- rechts naast de "`warme drank bestellen`"-knop staan **checkboxen**
- de knop "`water bestellen`" krijgt een **groene** achtergrond

Tips: Je maakt een checkbox met: **`new JCheckbox("thee?")`**
en met **`isSelected()`** vraag je op of een checkbox geselecteerd is of niet.

Je maakt een `JButton` **`b`** groen met:

```
b.setBackground(Color.green);
```

(Als je een *Macbook* hebt, moet je waarschijnlijk ook nog `b.setOpaque(true)` doen).

nakijkmodel:
in totaal **5 punten:**
1 punt: frame + goede layout
1 punt: Bar meegegeven en opgeslagen
1 punt: correcte titel
1 punt: sluit goed af (EXIT_ON_CLOSE)
1 punt: knop juiste kleur

(Tip voor evt. herkansing: houdt de uitwerking van een oefentoets bij de hand. Het maken van een JFrame is voor een groot deel copy-pasta)

b) [5 punten]
Maak de **`"warme drank bestellen"`**-knop werkend. Deze voegt een `WarmeDrank` toe aan het `Bar`-attribuut van de GUI. De checkboxen bepalen de inputs voor de `WarmeDrank`-constructor.

In de bovenstaande screenshot wordt een kop thee met suiker toegevoegd aan het `Bar`-attribuut van de GUI (als op de knop geklikt wordt).

Tip: Je kunt de GUI iets laten printen, zodat je kunt zien of de knoppen werken. Als je `Exceptions` tegenkomt, stap dan eens met de *debugger* door de code.

nakijkmodel:
in totaal **5 punten:**

- 3 punten aftrek: button reageert (actionPerformed, etc) werkt niet
- 1 punt aftrek: checkbox thee/koffie werkt niet
- 1 punt aftrek: checkbox melk werkt niet
- 1 punt aftrek: checkbox suiker werkt niet

(n.b. het gaat in de GUI vragen niet om het correct werken van bv. de `toevoegen`-methode uit opgave 2. Het gaat om het werken van de GUI, dus om het correct *aanroepen* van een methode, dus `implements ActionListener,` etc. etc... werkend.).

Opgave 4: GUI, dialoog [20 punten]

a) [5 punten]
Als je op de **`"water bestellen"`**-knop drukt, moet een **modale** dialoog verschijnen. Alle attributen van de dialoog zijn **`private.`**

Maak de dialoog zoals in dit screenshot is weergegeven:

nakijkmodel:
in totaal **5 punten:**
2 punten: dialoog verschijnt
1 punt: modaal
1 punt: layout en titel
1 punt: private attributen

(n.b. als je een dialoog had die bij "ok" open bleef staan als de input geen getal was, dan was dat ook goed. Idem voor varianten waarbij een foutmelding werd getoond in zo'n geval).

b) [10 punten]
In de dialoog geef je aan wat voor **formaat** en **hoeveel** `Water` je wilt bestellen (`"formaat:"` en `"aantal:"`):

- als op de **"OK"**-knop gedrukt wordt, moet de bestelling toegevoegd worden aan het `Bar`-attribuut van de GUI
- als op **"cancel"** gedrukt wordt dan sluit de dialoog

nakijkmodel:
in totaal **10 punten:**
4 punten: aantal x toegevoegd
3 punten: gegevens uit dialoog verwerkt
3 punten: goede cancel werking

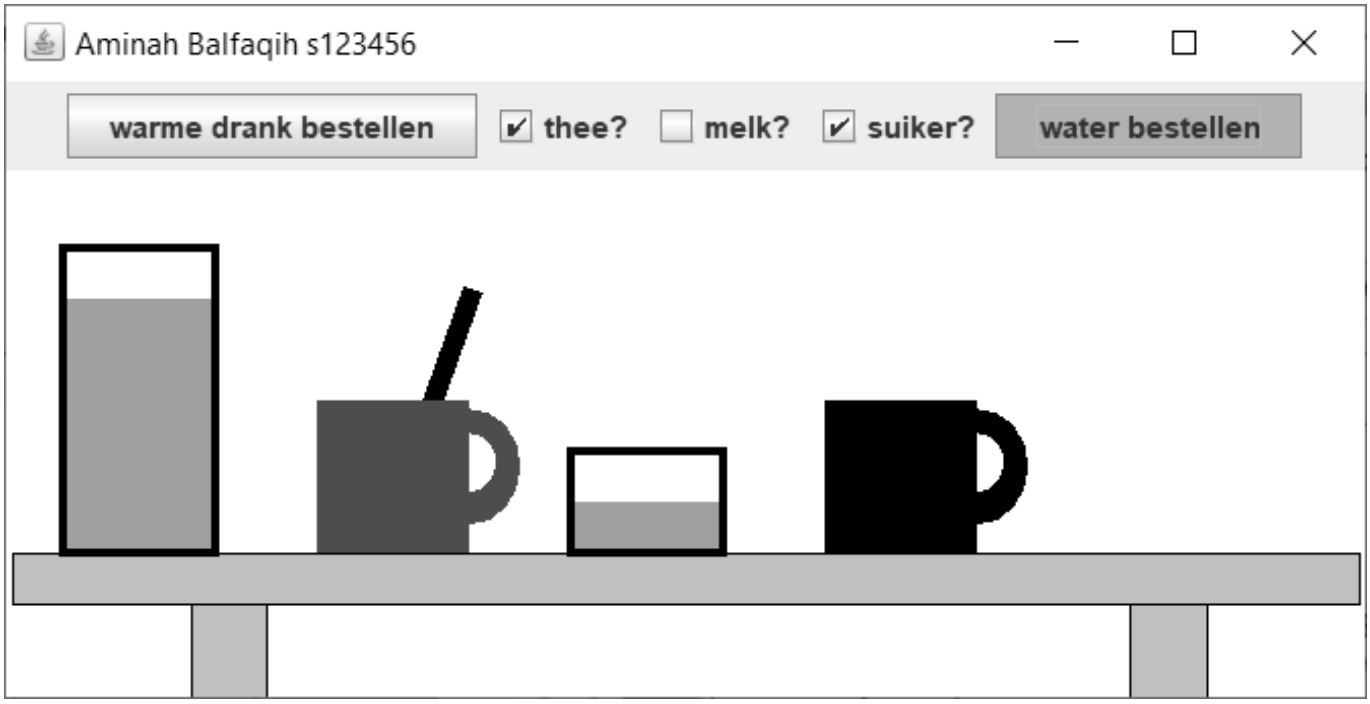
c) [5 punten]
Bij onderdeel **(b)** kan het gebeuren dat bij `"aantal:"` geen getal wordt ingevuld.

- vang de `Exception` op die dan ontstaat. De **`"water bestellen"`**-knop moet dan **rood** worden
- als er weer op een knop wordt geklikt, wordt de **`"water bestellen"`**-knop weer **groen** (tenzij er weer een foute input is uiteraard)

nakijkmodel:
in totaal **5 punten:**
3 punten: try-catch werkt
1 punt: foutmeling (kleur)
1 punt: foutmelding weg bij ok

Opgave 5: GUI, tekenen [15 punten]

In deze opgave maak je de grafische weergave van de `Bar` als volgt: (kleurenversie zit in `"startcode_praktijk_16-06-2022.zip"`)



Voor het tekenpaneel geldt:

- de achtergrond is **wit**
- de **lichtgrijze** balken onderaan zijn de tafel
- de tekening moet altijd ***up-to-date*** zijn

Tip: `setPreferredSize(new Dimension(.. , ..))` in plaats van `setSize(.., ..)`

Voor de **`drankenOpTafel`** van het `Bar`-attribuut van de GUI geldt:

- de afstand tussen de `Drank`-iconen is ongeveer **`40`** pixels

Voor **`Water`** geldt:

- een **blauwe** rechthoek met **breedte** van **`60`** pixels
- de **hoogte** van de rechthoek moet het `formaat` van het `Water`-object weergeven. Je kunt als hoogte **`40 * formaat`** gebruiken.
  (In de screenshot staan een grote en een kleine `Water`.)

Voor de **`WarmeDrank`** geldt:

- weergegeven als vierkant van **`60`** bij **`60`** pixels
- met overlappende cirkels maak je het oor van het kopje
- **thee** wordt met **rood** aangegeven
- **koffie** wordt met **zwart** weergegeven

- er moet een schuine zwarte lijn (een lepeltje) getekend worden
  als de `WarmeDrank` **melk en/of suiker** heeft

Je mag dingen toevoegen aan klassen uit opgave 1 en 2 **voor je GUI.** Je kunt bv. methodes toevoegen zodat objecten zichzelf tekenen, of *getters* om op te vragen of een `WarmeDrank` `suiker` heeft, ...

Tips:
Als je alles in het tekenpanel wilt tekenen, zul je moeten *"casten".* Stel dat je een **`Drank d`** hebt. Als je *weet* dat dit van het type `Water` is, dan kun je:

```
Water w = (Water) d;
```

doen om d (van type `Drank`) naar de nieuwe variabele w (van type `Water`) te *"casten"* (om te zetten).

Met

```
((Graphics2D) g).setStroke(new BasicStroke(3));
```

kun je **`g`** instellen om lijnen met dikte **`3`** tekenen.

nakijkmodel:
in totaal **15 punten:**
1 punt: er is een (werkend) tekenpanel
1 punt: witte achtergrond
1 punt: super.paintComponent(g)
1 punt: repaint werkt
1 punt: grijze tafel
2 punten: rechthoeken voor de iconen
2 punten: waterhoogte
2 punten: kleuren (water, koffie, thee)
1 punt: oor
1 punt: lepeltjes bij melk/suiker
2 punten: layout/overlap/etc

Einde Tentamen

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

Upload je `zip/rar`-bestand op ELO in het inleverpunt in de folder genaamd:
`"inleverpunt praktijk 16-06-2022"`