hogeschool Windesheim

Techniek – HBO-ICT

Naam: Studentnummer: Klas:

Practicum herk. JAVA Theorie (119893)

Vakcode : ICT.P.JAVA1.V21 (ICT.P.JAVA1.V20/V19/V18/V17) (t1)
Datum : dinsdag 14 juni 2022
Tijd : 11:30 - 13:30 uur

Klas:	Lokaal:	Aantal:
ICTM2a t/m p, ICTM2tt	…	226
	…	
	…	

Opgesteld door : Kevin de Korte; Wilco Moerman
Docenten : WPH01; KNJ24; RWM02; SSM36; LNR08; KEK01; MNC07; VEE02; CNW01; HZJ40; BHA40;
Gecontroleerd door : Jairo Hernandez; Aminah Balfaqih

Rekenmachine : alle rekenmachines toegestaan
Literatuur : alles (boeken, internet, aantekeningen)
Overige hulpmiddelen : laptop

Opgaven inleveren : ja

CONTROLEER VOORAF DE VOLGENDE GEGEVENS:

Dit tentamen bevat:

5 opgaves

18 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 theorie 14-06-2022"`** in **`"startcode_theorie_14-06-2022.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.

Opgave 1: koffers [25 punten]

In deze opgave maak je een `Koffer` met beveiliging.

```
public class Koffer {

	public int grootte;
	public String inhoud;
	public int code;

	// voor vraag (c)
	public ArrayList<Koffer> verdeel(int aantal, String naam) {
		int nieuweGrootte = 1 + grootte / aantal;
		ArrayList<Koffer> lijstje = new ArrayList<>();
		for (int i = 0; i < aantal; i++) {
			String nieuweInhoud = inhoud + "_"
					+ naam.toLowerCase() + "#" + (i + 1);
			lijstje.add(new Koffer(nieuweGrootte, nieuweInhoud));
		}
		return lijstje;
	}
}
```

a) [5 punten]
Maak twee constructors in de klasse Koffer met de volgende eisen:

- een constructor met de `grootte` en `inhoud` als inputs
- een andere constructor met alleen de `grootte` als input
- de default waarde voor attribuut `inhoud` is `"leeg"`
- de minimale `grootte` voor een `Koffer` is **`10.`** Indien de opgegeven `grootte` kleiner is dan **`10,`** dan wordt de `grootte` **`10`**
- bij het aanmaken wordt `"nieuw:"` geprint met de `inhoud` en `grootte`
- de constructor met maar 1 input, mag **maar één** Java-statement bevatten, dus maar één puntkomma. (Als dat niet lukt, zorg dan in ieder geval dat de code compileert.)

werking van Koffer-constructors
Output staat in comments, herkenbaar aan //>>

```
Koffer k1 = new Koffer(555, "JAVA-boek");  //>>  nieuw: JAVA-boek (555)
Koffer k2 = new Koffer(3);                 //>>  nieuw: leeg (10)
```

Let op: (b), (c) en **(d)** kunnen los van elkaar gemaakt worden.

b) [10 punten]
Pas ***encapsulatie*** toe om attributen te beschermen tegen wijzigingen van buitenaf. Maak daarom ook de setter **`setCode`** voor `code`, die een nieuwe `code` toekent aan de `Koffer`, als aan de volgende eisen wordt voldaan:

- de `code` mag maar **één** keer ingesteld worden
- de nieuwe `code` heeft **4** cijfers (dus **`1000`** t/m **`9999`** zijn geldig)

Print **`"mag niet"`** als niet aan de eisen voldaan wordt.

werking van setCode

```
Koffer k1 = new Koffer(77, "mobieltje");  //>>  nieuw: mobieltje (77)
// te klein of te groot
k1.setCode(1);                            //>>  mag niet
k1.setCode(1000000);                      //>>  mag niet

// goed
k1.setCode(5678);

// mag maar 1x ingesteld worden
k1.setCode(1234);                         //>>  mag niet
```

c) [5 punten]
De **`verdeel`**-methode maakt een aantal nieuwe koffers aan om de `inhoud` daarover te verdelen. De nieuwe koffers worden in een `ArrayList` verzameld.

De berekening en de nieuwe inhoud zijn al **goed** (en mogen niet veranderd worden).

De `verdeel`-methode kan ***crashen.*** Vul de methode aan, zodat de crashes **niet** meer kunnen plaatsvinden:

- je mag **geen** gebruik maken van een **`if`**
- er moet **`"dat is wiskundig onmogelijk!"`** uitgeprint worden als het om een **wiskundige** fout gaat
- bij alle **overige** fouten, print de methode **`"er is een fout opgetreden"`**
- bij alle fouten moet **`null`** gereturnd worden

werking van verdeel

```
String input = null;
Koffer k1 = new Koffer(33, "laptop");  //>>  nieuw: laptop (33)

ArrayList<Koffer> lijstje;
lijstje = k1.verdeel(2, "DEEL");       //>>  nieuw: laptop_deel#1 (17)
                                       //>>  nieuw: laptop_deel#2 (17)

// gevulde Arraylist (inhoud niet belangrijk voor deze vraag).
System.out.println(lijstje.size());    //>>  2

lijstje = k1.verdeel(0, "oeps!");      //>>  dat is wiskundig onmogelijk!
System.out.println(lijstje);           //>>  null

lijstje = k1.verdeel(1, input);        //>>  er is een fout opgetreden
System.out.println(lijstje);           //>>  null
```

d) [5 punten]
Maak de **`getInhoud`**-methode, die de `grootte` en `inhoud` samen als `String` returnt als je de juiste `code` hebt meegegeven.

Als de `code` onjuist is, gaat de `Koffer` in *"lockdown"*. Je kunt de inhoud dan **nooit** meer bekijken, ook niet met de juiste `code`.

Return `"---lockdown---"` als een verkeerde `code` gebruikt wordt.

De `code` veranderen kan uiteraard ook niet meer nadat de koffer in *"lockdown"* is.

werking van de lockdown functionaliteit

```
Koffer k1 = new Koffer(50, "telefoon");  //>>  nieuw: telefoon (50)
k1.setCode(7777);

System.out.println(k1.getInhoud(3400));  //>>  ---lockdown---
System.out.println(k1.getInhoud(5600));  //>>  ---lockdown---

// goede code werkt ook niet meer
System.out.println(k1.getInhoud(7777));  //>>  ---lockdown---
k1.setCode(7777);                        //>>  mag niet

Koffer k2 = new Koffer(33, "boek");      //>>  nieuw: boek (33)
k2.setCode(1234);
System.out.println(k2.getInhoud(1234));  //>>  33: boek
```

Opgave 2: stations en spoorlijnen [20 punten]

a) [10 punten]

`Spoorlijn`-objecten hebben een `ArrayList` met stations. De eerste is het beginpunt, de laatste het eindpunt. We willen `Spoorlijn`-objecten kunnen vergelijken. Ze zijn hetzelfde als als de begin- en eindstations hetzelfde zijn. De stations die er tussen liggen, en de volgorde, maken niet uit.

```
public class Spoorlijn {

	private String naam;
	private ArrayList<Station> stations;

	public Spoorlijn(String n) {
		naam = n;
		stations = new ArrayList<>();
	}

	public Station getLaatsteStation() {
		return stations.get(stations.size() - 1);
	}

	public int lengte() {
		return stations.size();
	}

	public String toString() {
		return naam + ": " + stations.get(0) + " --> "
				+ getLaatsteStation() + " (" + lengte() + ")";
	}

	public void voegStationToe(String station) {
		Station s = new Station(stations.size(), station);
		stations.add(s);
	}
}
```

```
public class Station {

	private int id;
	private String naam;

	public Station(int i, String n) {
		id = i;
		naam = n;
	}

	public String toString() { return naam; }
}
```

Maak de **equals**-methode voor `Spoorlijn`. Er geldt:

- de methode **overschrijft** *(override)* de `equals`-methode die je van `Object` overerft
- als beide `Spoorlijn`-objecten hetzelfde **begin- en eindstation** hebben, wordt `true` gereturned. Volgorde is niet belangrijk, en tussenstations ook niet:
  `Spoorlijn` **A** `-->` B `-->` **Z** is gelijk aan `Spoorlijn` **A** `-->` X `-->` Y `-->` **Z**
  `Spoorlijn` **A** `-->` ... `-->` **Z** is gelijk aan `Spoorlijn` **Z** `-->` ... `-->` **A**
- stations zijn gelijk als ze dezelfde **`naam`** hebben

Vul de code aan, zodat de `equals`-methode voldoet aan de bovenstaande eisen. Je mag beide klassen aanpassen. Hieronder volgt een voorbeeld van de werking.

werking van equals van Spoorlijn

```
Spoorlijn een = new Spoorlijn("IC");
een.voegStationToe("Zwolle");
een.voegStationToe("Deventer");

Spoorlijn twee = new Spoorlijn("Sprinter");
twee.voegStationToe("Zwolle");
twee.voegStationToe("Heino");
twee.voegStationToe("Raalte");
twee.voegStationToe("Wierden");
twee.voegStationToe("Deventer");

// begin- en eindpunt hetzelfde
System.out.println(een.equals(twee));  //>>  true

Spoorlijn drie = new Spoorlijn("IC");
drie.voegStationToe("Deventer");
drie.voegStationToe("Zwolle");

// eindpunt van ene is beginpunt van de ander, en andersom.
System.out.println(een.equals(drie));  //>>  true

Spoorlijn vier = new Spoorlijn("stoomtrein");
vier.voegStationToe("Zwolle");
vier.voegStationToe("Middle-of-Nowhere");

// ander begin- of eindpunt
System.out.println(een.equals(vier));  //>>  false
```

Let op: (a) en **(b)** kunnen los van elkaar gemaakt worden.

b) [10 punten]
De langste `Spoorlijn` die aangemaakt is, moet onthouden worden. Met de **`printLangste()`**-methode moet deze geprint worden.

Als twee spoorlijnen even lang zijn, moet de *laatst aangemaakte* onthouden worden.

Met de `lengte()`-methode krijg je de lengte van een `Spoorlijn`.

werking van printLangste()

```
Spoorlijn.printLangste();  //>>  null

Spoorlijn een = new Spoorlijn("IC");
een.voegStationToe("Zwolle");
een.voegStationToe("Deventer");

Spoorlijn.printLangste();  //>>  IC: Zwolle --> Deventer (2)

// Spoorlijn twee is langer dan Spoorlijn een, dus twee wordt onthouden
Spoorlijn twee = new Spoorlijn("Sprinter");
twee.voegStationToe("Zwolle");
twee.voegStationToe("Wezep");
twee.voegStationToe("Nunspeet");
twee.voegStationToe("Harderwijk");
twee.voegStationToe("Utrecht");

Spoorlijn.printLangste();  //>>  Sprinter: Zwolle --> Utrecht (5)

// Spoorlijn drie is korter dan de langste
Spoorlijn drie = new Spoorlijn("Sprinter");
drie.voegStationToe("Groningen");
drie.voegStationToe("Assen");
drie.voegStationToe("Zwolle");

Spoorlijn.printLangste();  //>>  Sprinter: Zwolle --> Utrecht (5)
```

Opgave 3: geheimschrift [25 punten]

In deze opgave ga je een **`array`** gebruiken voor geheime berichten.

```
public class Geheimschrift {

	public Geheimschrift(int lengte) { /*...todo...*/ }

	public void maakGereed() { /*...todo...*/ }

	public void verberg(int positie, int stap, String[] tekst) {
		/*...todo...*/
	}
}
```

Let op:
In deze hele opgave mag je er vanuit gaan, dat elke `String` in een `array` maar **1** symbool (letter, leesteken, etc) is. De arrays kun je zien als woorden en `Strings` die erin zitten als letters.

In de zip met de startcode vind je in klasse `Geheimschrift` ook drie methodes die je kunt gebruiken (maar niet aan mag passen) in **(b)** en **(c).**

a) [5 punten]
Geef de klasse `Geheimschrift` een attribuut dat een String-array is, genaamd **`code.`**

Maak ook de `Geheimschrift`-**constructor** af:

- de input is de gewenste lengte van het `code`-attribuut
- de constructor zorgt er dus voor dat het attribuut `code` die lengte krijgt.

b) [5 punten]
programmeer de **`maakGereed`**-methode:

- deze methode gebruikt de **`randomSymbool`**-methode om op elke plek in de array een willekeurig symbool te zetten
- de `maakGereed`-methode wordt aangeroepen in de **constructor**

Maak ook de **`print`**-methode die de inhoud van het `code`-attribuut uitprint.
Met `System.out.print` (en niet `println`) kun je tekst op dezelfde regel printen.

print en maakGereed-methodes

```
Geheimschrift g = new Geheimschrift(6);
g.print();  //>>  *-,-..
```

Let op: de *random* symbolen kunnen bij jou anders zijn dan in de voorbeelden.

c) [15 punten]
maak de **`verberg`**-methode. Deze krijgt als inputs een tweetal `ints` (`positie` en `stap`) en een `String-array` (`tekst`). De methode "verstopt" de `tekst` in het attribuut `code`. De letters worden met gelijke tussenafstanden in de array geplaatst:

- de eerste `String` in de `tekst`-array wordt op plek **`positie`** in het array-attribuut `code` gezet
- de tweede `String` in de `tekst`-array komt op plek **`positie + stap`** in `code`
- de derde op **`positie + stap + stap`**
- de vierde op **`positie + stap + stap + stap`**

  enz..

Let op: Ter controle kun je gebruik maken van de `cryptischeVerberg`-methode die de boodschap op dezelfde manier verbergt (maar op een veel te ingewikkelde manier). Hiermee kun je andere tests voor je `verberg`-methode maken.

Bij **foute inputs** kan `verberg(...)` ***crashen*** (`ArrayIndexOutOfBoundsException`):

- zorg ervoor dat dat **niet** meer kan gebeuren
- je mag hiervoor **geen** `try` en `catch` gebruiken
- als het niet gaat passen, wordt **`"fout: Array te klein"`** geprint
- het `code`-attribuut mag in zo'n geval **niet** veranderen

werking van verberg

```
Geheimschrift g = new Geheimschrift(12);
g.print();                          //>>  *-,-...+--.-

String[] java = {"J", "A", "V", "A"};
g.verberg(3, 2, java);
g.print();                          //>>  *-,J.A.V-A.-

// ------------ decoderen met de juiste start en stap --------------

String decoded = g.ontcijfer(3, 2, java.length);
System.out.println(decoded);        //>>  JAVA

// ------- decoderen met verkeerde start of stap werkt niet --------

String decodedFout = g.ontcijfer(4, 2, java.length);
System.out.println(decodedFout);    //>>  ..-.

decodedFout = g.ontcijfer(3, 4, java.length);
System.out.println(decodedFout);    //>>  JV-

// -------------------------- past nog net! ---------------------------

Geheimschrift g2 = new Geheimschrift(16);
g2.verberg(1, 4, java);
g2.print();                         //>>  ,J,.,A+.+V*..A--
decoded = g2.ontcijfer(1, 4, java.length);
System.out.println(decoded);        //>>  JAVA

// -------------------------- past niet! ----------------------------

Geheimschrift teKlein = new Geheimschrift(7);
String[] pastNiet = {"P", "A", "S", "T", " ", "N", "I", "E", "T"};
teKlein.verberg(1, 2, pastNiet);    //>>  fout: Array te klein

// de array is niet veranderd:
teKlein.print();                    //>>  ,,*.*+-
```

Opgave 4: getallen [10 punten]

Een behulpzame wiskundige heeft een wiskunde-gerelateerde klasse gemaakt (genaamd `Getal`). Er zitten helaas nog wat fouten in.

```
public class Getal {

	public int waarde;

	public Getal(int w) {
		waarde = w;
	}

	public String toString() {
		return "Getal: " + waarde;
	}

	// optellen met int
	public Getal plus(int x) {
		waarde = waarde + x;
		return new Getal(waarde);
	}

	// optellen met Getal
	public Getal plus(Getal getal) {
		getal.waarde = waarde + getal.waarde;
		return getal;
	}
}
```

De bedoeling van de **`plus`**-methodes is dat je `int` of een `Getal` bij een ander `Getal` kunt optellen. Maar als je **4** bij **1** optelt, is het natuulijk *niet* de bedoeling dat de 4 of de 1 daarna ineens van waarde veranderd zijn!
Een 4 is een 4 en dat moet zo blijven!

bugs: Getal-objecten veranderen van waarde

```
Getal vier = new Getal(4);
Getal twaalf = vier.plus(8);

System.out.println(vier);              //>>  Getal: 12
System.out.println(twaalf);            //>>  Getal: 12

Getal negenEnNegentig = new Getal(99);
Getal drie = new Getal(3);

Getal honderdTwee = negenEnNegentig.plus(drie);

System.out.println(drie);              //>>  Getal: 102
System.out.println(negenEnNegentig);   //>>  Getal: 99
System.out.println(honderdTwee);       //>>  Getal: 102
```

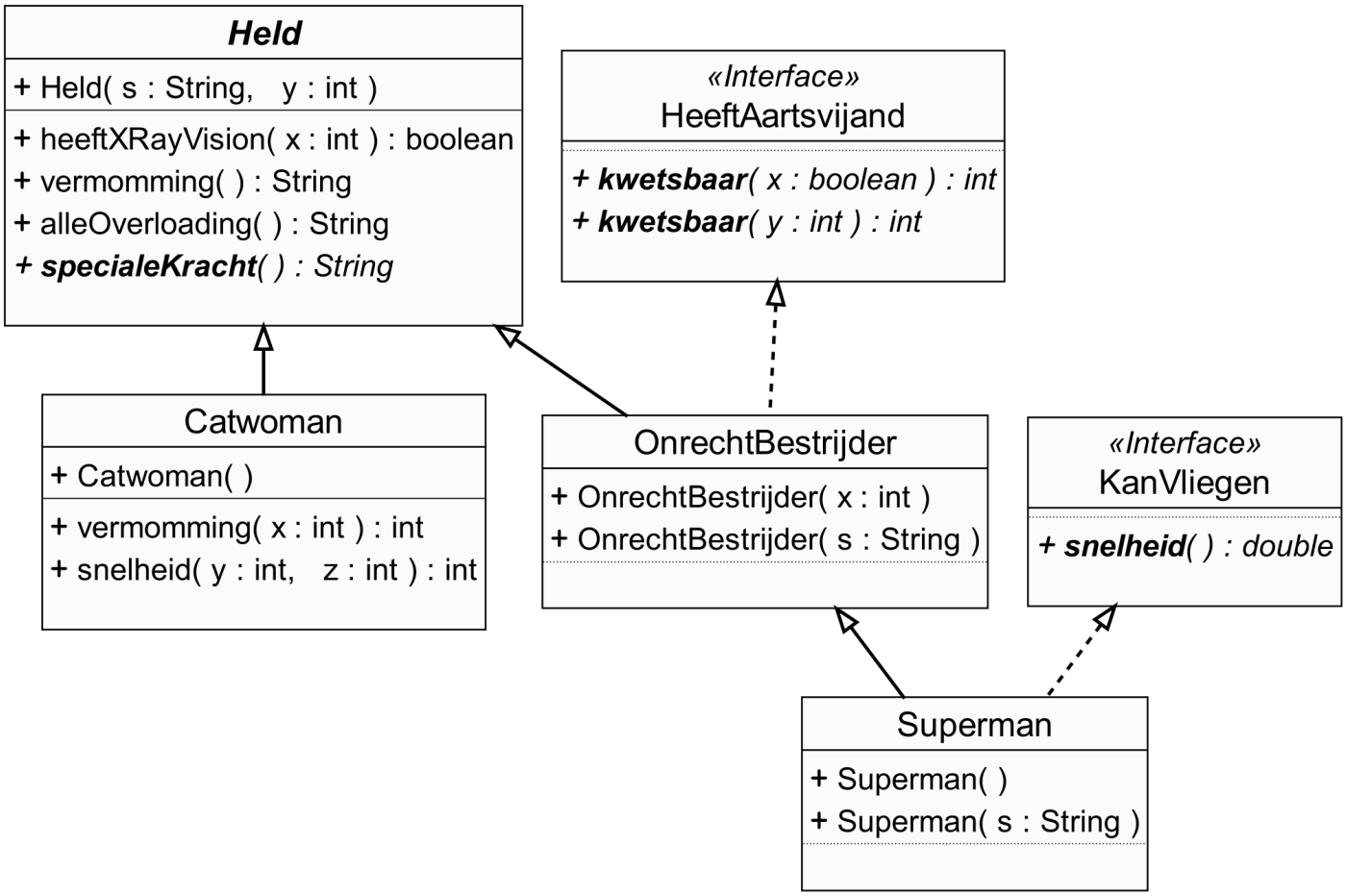
Repareer alle fouten in klasse `Getal`, zodanig dat de optellingen werken **en** de `Getal` -objecten niet veranderen door de methodes.

verwachte werking (zonder bugs)

```
Getal vier = new Getal(4);
Getal twaalf = vier.plus(8);

System.out.println(vier);              //>>  Getal: 4
System.out.println(twaalf);            //>>  Getal: 12

Getal negenEnNegentig = new Getal(99);
Getal drie = new Getal(3);

Getal honderdTwee = negenEnNegentig.plus(drie);

System.out.println(drie);              //>>  Getal: 3
System.out.println(negenEnNegentig);   //>>  Getal: 99
System.out.println(honderdTwee);       //>>  Getal: 102
```

Opgave 5: Kryptonite [20 punten]

Een fan van *superhelden* heeft dit klassendiagram opgesteld: ***(vetgedrukt + cursief*** geeft "abstract" aan)



Methodes die door klassen moeten worden geimplementeerd vanwege een interface of abstracte klasse, staan **niet** in het diagram vermeld.

a) [10 punten]
Schrijf code op basis van dit diagram, dus met de correcte **klassen, overerving, interfaces** en **methodes** (constructors komen in **(c)):**

- de implementatie moet **zo weinig mogelijk** methodes bevatten
- voor elke overgeërfde methode zul je dus moeten bepalen of deze **overschreven** *(overriding)* moet worden, aan de hand van de regels hieronder

Regels voor wat methodes moeten doen:

- als een methode een *getal* returnt, return dan altijd **`42`**
- als een methode een **`String`** returnt, dan **moet** je de naam van de klasse en de methode gebruiken, dus bv.: `return` **`"Superman --> vermomming";`**

Let op: opgaves **(b)** en **(c)** kunnen los van elkaar gemaakt worden.

b) [5 punten]
Voeg de methode **`alleOverloading()`** toe aan klasse **`Held.`** Deze returnt een `String` met de **namen** van alle methodes waarbij sprake is van **overloading.**

Als je bv. denkt dat de methodes `heeftXRayVision`, `specialeKracht` en `vermomming` alle gevallen zijn van *overloading,* dan ziet `alleOverloading()` er zo uit (volgorde van de namen maakt niet uit):

```
public String alleOverloading() {
   return "specialeKracht + vermomming + heeftXRayVision";
 }
```

c) [5 punten]
Voeg de **constructors** uit klassendiagram toe aan je code:

- de constructors doen niks en slaan niks op. De inputs betekenen niks
- je mag zelf weten welke `super(...)` je aanroept, als er keuze is
- als je een *getal* nodig hebt als *input* voor een `super(..)`, gebruik dan 42
- als je een **`String`** nodig hebt als *input,* gebruik dan `"Captain Napalm"`

Let op: Als de code *vanwege de constructors* niet compileert, kun je voor **(c)** geen punten behalen.

Einde Tentamen

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

Upload je `zip/rar`-bestand op ELO in het inleverpunt in de folder genaamd: **`"inleverpunt theorie 14-06-2022"`**