hogeschool Windesheim

Techniek – HBO-ICT

Naam: Studentnummer: Klas:

HP JAVA Theorie (110245)

Vakcode : ICT.P.JAVA1.V20 (ICT.P.JAVA.V19/18/17/16) (t1)
Datum : dinsdag 8 juni 2021
Tijd : 11.30 - 13.30 uur

Klas:	Lokaal:	Aantal:
ICTM2a t/m i, ICTM2n t/m r, ICTM2tt	volgt	234,6

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:

4 opgaves

16 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 andere personen kan brengen is tijdens de tentamentijd niet toegestaan.

Het gebruik van internet om informatie op te zoeken is wel toegestaan Je mag dus wel googlen en bv. 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 theorie 8 juni"** in het bestand **`"startsituatie_theorie_8-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.

In de toets is zoveel mogelijk (met **"let op"**) aangegeven of onderdelen van vragen los van elkaar gemaakt kunnen worden.

Als je wilt, kun je op de papieren toets aantekeningen maken.

Opgave 1: Wandelroutes [35 punten]

Gegeven is de klasse `Route`. Deze klasse bevat de gegevens van een wandelroute.

inhoud van klasse Route

```
public class Route
{
	public int afstand;
	public String start;

	public int moeilijkheidsgraad( int tijd, int conditie ) {
		// voor vraag (c)
		return conditie / ( tijd * ( afstand - 3 ) );
	}
}
```

a) [10 punten]
Maak twee constructors in de klasse `Route`:

- De ene heeft twee inputs, een `int` en een `String`. Deze worden opgeslagen in de twee attributen.
- De andere heeft alleen een `int` als parameter. Deze wordt uiteraard opgeslagen in het attribuut `afstand`. De default waarde voor `start` is **`"???"`**
- Van de twee constructors mag er eentje maar **1** Java-statement bevatten (dus maar 1 puntkomma).

Zorg dat onderstaande code werkt en de gewenste output geeft.

maken en printen van Route-objecten (in de comments staat wat uitgeprint wordt)

```
Route w1 = new Route( 11 );
System.out.println( w1 );              // Route van 11 km, start: ???

Route w2 = new Route( 7, "Zwolle" );
System.out.println( w2 );              // Route van 7 km, start: Zwolle
```

Let op: de rest van de onderdelen van **opgave 1** kunnen los van elkaar gemaakt

worden. Het maakt voor die onderdelen ook niet uit, of je constructors wel of niet aan de derde *bullet* in **1(a)** voldoen over maar 1x puntkomma.

b) [10 punten]
Gebruik ***encapsulatie*** om de attributen te beschermen tegen wijzigingen van *buitenaf.*

Voor **`afstand`** geldt:

- bij het aanmaken van een `Route` mag de `afstand` niet korter zijn dan **`3`** km.
- als de meegegeven waarde te kort is, zet `afstand` dan op **`-1`**.
  Print ook **`"te kort"`** als de meegegeven waarde te kort was.
- de constructors zijn de enige manier waarop `afstand` een waarde kan krijgen.

Voor **`start`** geldt:

- als de `start` nog **`"???"`** is, kan deze met een setter ingesteld worden.
- maar als de `start` eenmaal bekend is mag deze nooit meer veranderd kunnen worden van buitenaf.
- print **`"mag niet"`** als de aanpassing illegaal is (en dus niet uitgevoerd wordt).

werking van de constructor en setter (de comments tonen wat geprint wordt)

```
Route w = new Route( 12 );
System.out.println( w );              // Route van 12 km, start: ???

w.setStart( "Assen" );
System.out.println( w );              // Route van 12 km, start: Assen

w.setStart( "Wapenveld" );            // mag niet
System.out.println( w );              // Route van 12 km, start: Assen

Route w2 = new Route( 8, "Zwolle" );
System.out.println( w2 );             // Route van 8 km, start: Zwolle
w2.setStart( "Eindhoven" );           // mag niet

Route w3 = new Route( 1 );            // te kort
```

c) [5 punten]
Wandelaars willen graag weten hoe zwaar een bepaalde `Route` voor hen zal zijn. Om die reden is de methode `moeilijkheidsgraad(...)` toegevoegd. Deze berekening is uiteraard gebaseerd op jarenlang wetenschappelijk onderzoek en absoluut niet door de bedenker van de toets uit zijn duim gezogen, dus je mag deze berekening *niet* veranderen.

Verkeerde input kan `moeilijkheidsgraad(...)` echter laten crashen.

- Pas de methode aan, zodat deze niet meer kan crashen.
- Je mag hierbij **geen** gebruik maken van `if`
- Je mag **niet** van de klasse `Exception` gebruik maken
  (maar wel van klassen die daarvan afgeleid zijn/overerven/gebruik maken).
- Laat de methode op het moment dat je een fout afvangt **-1** returnen.
- Uiteraard mag je de berekening *niet* aanpassen.

d) [10 punten]
De wandelorganisatie wil graag de *kortste* en de *langste* afstand bijhouden van alle routes.

- Zorg ervoor dat `printMinMax()` de kortste en de langste `afstand` uitprint van alle aangemaakte `Route`-Objecten.
- Je mag er vanuit gaan dat routes nooit langer dan `1000` km zijn.
- Je hoeft geen rekening te houden met de te korte routes uit **1(b)**.

werking van printMinMax (in de comments staat wat geprint wordt)

```
Route w1 = new Route( 9 );
Route w2 = new Route( 5 );
Route w3 = new Route( 7 );
Route.printMinMax();
                              // kortste afstand is 5 km
                              // langste afstand is 9 km
Route w4 = new Route( 12 );
Route w5 = new Route( 4 );
Route.printMinMax();
                              // kortste afstand is 4 km
                              // langste afstand is 12 km
```

Opgave 2: Figuren [25 punten]

a) [10 punten]
Gegeven is de klasse `Rechthoek` waaraan je een `equals`-methode moet toegevoegen.

Hieronder is een start gemaakt, maar deze methode is nog niet goed.

klasse Rechthoek met foute equals

```
public class Rechthoek
{
	private int lengte;
	private int breedte;
	private int kleur;

	public Rechthoek( int lengte, int breedte, int kleur ) {
		this.lengte = lengte;
		this.breedte = breedte;
		this.kleur = kleur;
	}

	public boolean equals( Rechthoek obj ) {
		if ( this.kleur == r.kleur ) {
			return true;
		}
		return
			this.lengte == r.lengte || this.breedte == r.breedte
					&&
			this.lengte == r.breedte || this.breedte == r.lengte;
	}
}
```

De `equals`-methode zou aan de volgende eisen moeten voldoen:

- oveschrijft (*overriding*) de `equals`-methode die `Rechthoek` van Object overerft
- return't alleen `true` als de `kleur` gelijk is èn de rechthoeken overeenkomen wat `lengte` en `breedte` betreft
  (dit mag ook als ze gedraaid zijn, zie de gegeven voorbeeldcode)
- Als de input geen `Rechthoek` is, dan is de uitkomst uiteraard `false`.

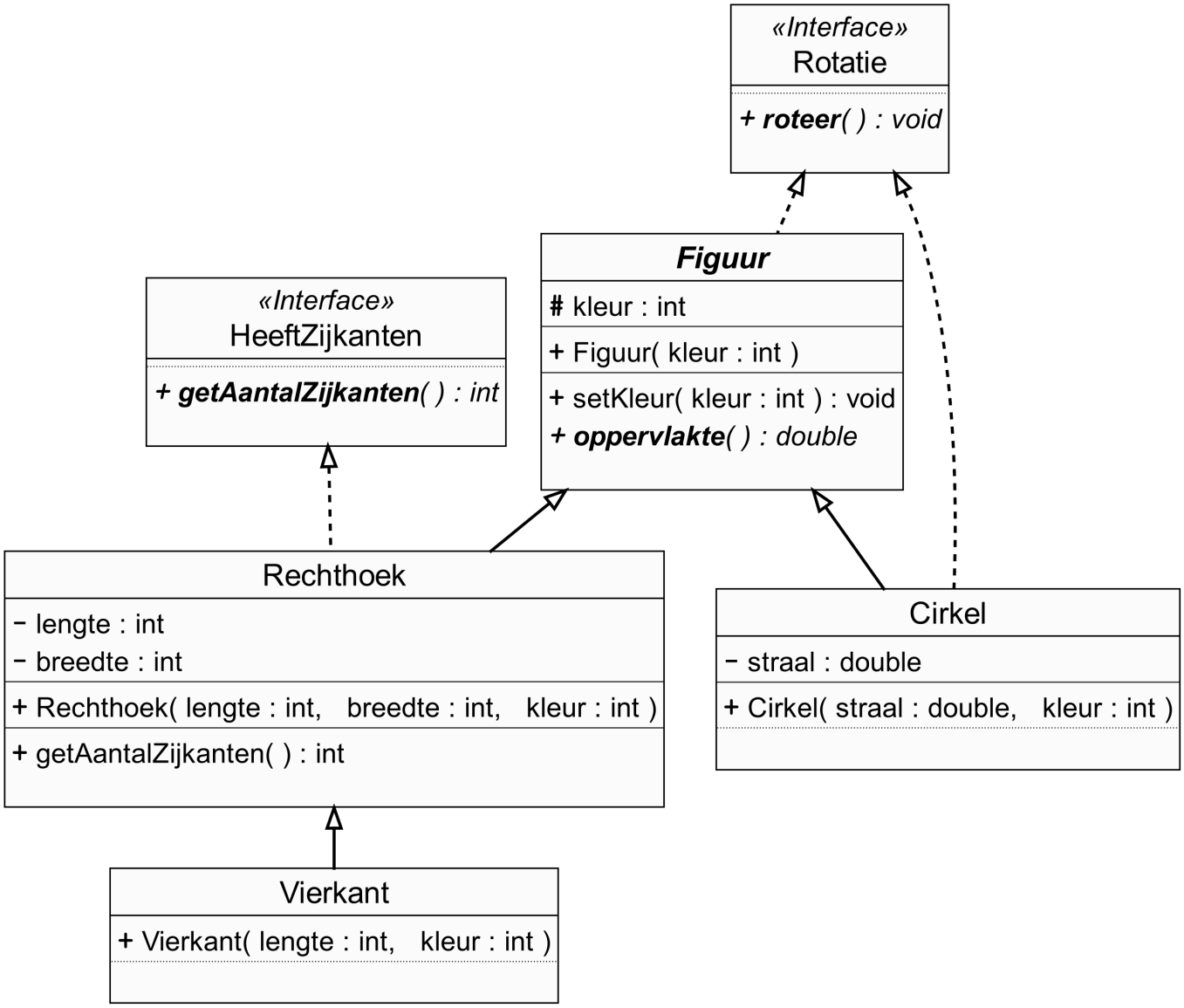
Repareer `equals(...)` in klasse `Rechthoek`. Hieronder staat een vb. van de werking.

verwachte werking van equals van Rechthoek (true/false output in comments)

```
Rechthoek re1 = new Rechthoek( 3, 4, 123 );
Rechthoek re2 = new Rechthoek( 5, 7, 123 );
Rechthoek re3 = new Rechthoek( 4, 3, 123 );
Rechthoek re4 = new Rechthoek( 3, 4, 7 );
Rechthoek re5 = new Rechthoek( 3, 4, 123 );

// vorm verschillend en kleur gelijk
System.out.println( re1.equals( re2 ) );     // false

// vorm gelijk maar gedraaid; kleur gelijk
System.out.println( re1.equals( re3 ) );     // true

// vorm gelijk; kleur verschillend
System.out.println( re1.equals( re4 ) );     // false

// vorm en kleur gelijk
System.out.println( re1.equals( re5 ) );     // true
```

Let op: Voor vraag **2(b)** is de `equals`-methode niet van belang.

b) [15 punten]
`Rechthoek` is deel van een groter geheel zoals je in dit klassendiagram kunt zien.



Maak een correcte implementatie van dit diagram, zodat de code werkt zoals in het voorbeeld verderop is weergegeven.

- Klasse `Rechthoek` heeft zelf geen attribuut `kleur`.
- Klasse `Vierkant` mag zelf geen `toString()` hebben.
- De methode `oppervlakte()` moet de oppervlakte van de `Figuur` berekenen. (de oppervlakte van een cirkel is `PI * straal * straal`)
- De methode `roteer()` roteert de `Figuur`

- de methode `getAantalZijkanten()` returnt het aantal zijkanten dat een `HeeftZijkanten`-Object heeft.

Let op:

- Methodes die een klasse moet implementeren vanwege overerving of interfaces, staan in het klassendiagram *niet* vermeld in de klasse zelf maar alleen in de "super" klasse of interface.
- Je moet dus zelf bepalen welke methodes in welke klassen **nodig** zijn en dus geimplementeerd moeten worden.
- **Overbodige** methodes en/of attributen leveren puntenaftrek op.
- Abstracte onderdelen zijn ***vetgedrukt en cursief*** weergegeven in het klassendiagram.

verwachte werking van Figuur, Cirkel, Rechthoek en Vierkant

```
Figuur figuur = new Rechthoek( 5, 10, 1 );
System.out.println( figuur );                // Rechthoek: 5 x 10 en kleur 1
System.out.println( figuur.oppervlakte() ); // 50.0
figuur.roteer();
System.out.println( figuur );                // Rechthoek: 10 x 5 en kleur 1

Vierkant vierkant = new Vierkant( 4, 13 );

HeeftZijkanten zijkanter = vierkant;
int n = zijkanter.getAantalZijkanten();
System.out.println( n );                     // 4

figuur = vierkant;
System.out.println( figuur );                // Vierkant: 4 x 4 en kleur 13
System.out.println( figuur.oppervlakte() ); // 16.0
figuur.roteer();
System.out.println( figuur );                // Vierkant: 4 x 4 en kleur 13

figuur = new Cirkel( 2, 77 );
System.out.println( figuur );                // Cirkel: straal = 2.0, kleur = 77
System.out.println( figuur.oppervlakte() ); // 12.566370614359172
figuur.roteer();
figuur.setKleur( 808 );
System.out.println( figuur );                // Cirkel: straal = 2.0, kleur = 808
```

Opgave 3: Vliegtuig en upgrade [10 punten]

In deze opgave modelleer je een `Vliegtuig` en een `Garage`.
Klasse `Vliegtuig` mag niet veranderd worden.

klasse Vliegtuig:

```
// Aan klasse Vliegtuig mag niks veranderd worden
public class Vliegtuig
{
	private String naam;

	public Vliegtuig( String naam ) { this.naam = naam; }

	public void setNaam( String update ) { this.naam = update; }

	public String getNaam() { return naam; }

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

Startsituatie voor klasse Garage (met bug erin):

```
public class Garage
{
	private Vliegtuig vliegtuig;

	public void zetInGarage( Vliegtuig v ) { vliegtuig = v; }

	public void upgrade( String upgradeNaam ) {
		// de nieuwe naam voor het Vliegtuig
		upgradeNaam = vliegtuig.getNaam() + upgradeNaam;

		// upgrade het Vliegtuig
		Vliegtuig upgrade = new Vliegtuig( upgradeNaam );

		// en update de Garage
		zetInGarage( upgrade );
	}
}
```

`Vliegtuig`-Objecten kunnen een *upgrade* krijgen. Dat gebeurt in methode `upgrade(...)` van klasse `Garage`.

Upgraden is het veranderen van de `naam`: **`"F-16"`** moet bv. **`"F-16.upgrade"`** worden.

In de `upgrade`-methode zit een bug. Hieronder volgt een voorbeeld:

Voorbeeld van het optreden van de bug in methode upgrade

```
Garage garage = new Garage();
Vliegtuig f16 = new Vliegtuig( "F-16" );
garage.zetInGarage( f16 );
garage.upgrade( ".upgrade" );

// Waarom is de naam van f16 niet veranderd in "F-16.upgrade" na upgrade(...) ?
System.out.println( f16 );                  // F-16
```

In bovenstaand voorbeeld is na de *upgrade* de `naam` van het meegegeven `Vliegtuig`-Object **niet** veranderd. De `naam` van `f16` is nog steeds `"F-16"` terwijl de bedoeling van de methode is, dat het **`"F-16.upgrade"`** wordt.

Repareer deze bug in de methode `upgrade` van klasse `Garage`.

Opgave 4: Vliegtuig en Vliegveld [30 punten]

Nu ga je een `Vliegveld` modelleren. Hiervoor gebruik je ook klasse `Vliegtuig` die je al in de vorige opgave zag (en nog steeds niet mag veranderen). Hieronder zie je een eerste opzet van `Vliegveld`:

Startsituatie voor klasse Vliegveld:

```
public class Vliegveld
{

      public Vliegveld( int grootte ) { /*...todo...*/ }

      public void zetBinnen( Vliegtuig v, int plek ) { /*...todo...*/ }

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

      public Vliegtuig haalEruit( int plek ) { /*...todo...*/ }

      public int zetOpEersteVrijePlek( Vliegtuig v ) { /*...todo...*/ }
}
```

a) [5 punten]
Voeg een attribuut met de naam `hangar` van het type `Vliegtuig`-array toe aan klasse `Vliegveld`. In deze array gaan in de volgende vragen `Vliegtuig`-Objecten geplaatst worden.

Maak de `Vliegveld`-constructor af. Deze heeft als input de gewenste grootte van het `hangar`-attribuut en zorgt ervoor dat de array die grootte krijgt.

b) [10 punten]
Maak de `print`-methode, die de inhoud van de `hangar` laat zien.

Maak ook de `zetBinnen`-methode:

- Deze zet het meegegeven `Vliegtuig` op de aangegeven `plek` in de `hangar`, als die plek vrij is.
- Als de aangegeven `hangar`-plek al bezet is door een `Vliegtuig`-Object, kan het `Vliegtuig` niet geplaatst worden. In dat geval moet er een foutmelding worden geprint.

De `zetBinnen`-methode crasht, als er geprobeerd wordt een `Vliegtuig` te plaatsen op een `plek` die buiten de array valt:

- Zorg ervoor dat deze methode niet crasht
- Print een foutmelding.
- Je mag **geen** gebruik maken van `try...catch`.

In onderstaande voorbeeldcode zie je hoe `print()` werkt en wat de foutmeldingen van `zetBinnen(...)` zijn.

Werking van zetBinnen en print (in de comments staat de verwachte output)

```
Vliegveld veld = new Vliegveld( 3 );

Vliegtuig f16 = new Vliegtuig( "F-16" );
veld.zetBinnen( f16, 2 );

Vliegtuig boeing = new Vliegtuig( "Boeing 747" );
veld.zetBinnen( boeing, 2 );                    // hangar 2 is al bezet
veld.zetBinnen( boeing, 3210 );                 // hangar 3210 bestaat niet!
veld.zetBinnen( boeing, 1 );

veld.print();
                                                // hangar:
                                                // * plek 0: ---
                                                // * plek 1: Boeing 747
                                                // * plek 2: F-16
```

Let op: 4(c) en **4(d)** kunnen los van elkaar gemaakt worden en het is bij **4(c)** en **4(d)** niet erg als je `zetBinnen`-methode nog kan crashen.

c) [5 punten]
Maak de methode `haalEruit(...)` die het `Vliegtuig`-Object returnt dat op de meegegeven `plek` in de `hangar` staat:

- Als er een `Vliegtuig` staat, moet dat gereturned worden en de plek in de `hangar` moet weer leeg (`null`) worden.
- Je hoeft je bij deze methode geen zorgen te maken over ongeldige inputs

Werking van haalEruit (in de comments staat de verwachte output)

```
Vliegveld veld = new Vliegveld( 3 );

Vliegtuig boeing = new Vliegtuig( "Boeing 747" );
veld.zetBinnen( boeing, 2 );

Vliegtuig f16 = new Vliegtuig( "F-16" );
veld.zetBinnen( f16, 1 );

veld.print();
                                                    // hangar:
                                                    // * plek 0: ---
                                                    // * plek 1: F-16
                                                    // * plek 2: Boeing 747

Vliegtuig x = veld.haalEruit( 2 );
System.out.println( x );                            // Boeing 747

veld.print();
                                                    // hangar:
                                                    // * plek 0: ---
                                                    // * plek 1: F-16
                                                    // * plek 2: ---

Vliegtuig y = veld.haalEruit( 0 );
System.out.println( y );                            // null
```

d) [10 punten]
Maak de methode `zetOpEersteVrijePlek(...)` die de eerste plek in de `hangar`-array zoekt waar nog geen `Vliegtuig` staat:

- Als de vrije plek gevonden wordt, wordt het meegegeven `Vliegtuig`-Object daar geplaatst en index van die plek gereturned.
- Als er geen vrije plek meer is, moet het getal **`-1`** worden gereturned.

werking van zetOpEersteVrijePlek (comments bevatten de verwachte output)

```
Vliegveld veld = new Vliegveld( 5 );

Vliegtuig dc10 = new Vliegtuig( "DC-10" );
veld.zetBinnen( dc10, 4 );

Vliegtuig f16 = new Vliegtuig( "F-16" );
veld.zetBinnen( f16, 0 );

Vliegtuig boeing = new Vliegtuig( "Boeing 747" );
veld.zetBinnen( boeing, 1 );

veld.print();
                                                    // hangar:
                                                    // * plek 0: F-16
                                                    // * plek 1: Boeing 747
                                                    // * plek 2: ---
                                                    // * plek 3: ---
                                                    // * plek 4: DC-10

Vliegtuig concorde = new Vliegtuig( "Concorde" );
int vrij = veld.zetOpEersteVrijePlek( concorde );
System.out.println( vrij );                         // 2

veld.print();
                                                    // hangar:
                                                    // * plek 0: F-16
                                                    // * plek 1: Boeing 747
                                                    // * plek 2: Concorde
                                                    // * plek 3: ---
                                                    // * plek 4: DC-10
```

Einde Tentamen

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

Upload je `zip/rar`-bestand op ELO in het inleverpunt in de folder genaamd: **`"inleverpunt theorie 8 juni"`**

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