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

25 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 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.

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.

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

Nakijkmodel/puntenverdeling:

let op: in dit document staat alles globaal, zodat het leesbaar blijft. De gerunde tests (en handmatige beoordeling indien nodig) zijn leidend.

waar je op kunt letten bij de inzage: **(1) doorrekenfouten.** (2) Dingen die fout zijn gerekend doordat er een rare **typo** in staat (we doen hier niet aan correct Nederlands!)

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
```

Puntenverdeling:

3 punten voor de toString(). Afwezig zijn van het woord "Route" leverde 1 aftrekpunt op. Er is niet op de precieze formulering gelet (zoals wel of niet hebben van "km", leestekens en het woord "start")

2 punten voor de werking v/d constructor met int als input (hierbij werd een veilige waarde, ruim boven de grens gebruikt). 1 punt aftrek als de default-waarde van start niet goed was.

2 punten voor de werking v/d constr. met int en String inputs.

3 punten als de ene constructor (of de andere) maar 1 regel Java heeft (dus 1x puntkomma). Dit kon je doen door this(...) te gebruiken, maar het had ook gekund door een methode te gebruiken om de waardes te zetten en die aan te roepen in de constructor (wel denken aan private, anders kan iedereen die aanroepen).

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.

Omdat de vragen onafhankelijk zijn, is ook zoveel mogelijk geprobeerd fouten in de ene vraag niet te laten meetellen bij de volgende vraag. Maar dat kan mis gegaan zijn. Let hierop bij controle/inzage van je werk.

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
```

Puntenverdeling:

3 punten voor het private maken van de te beschremen attributen

2 punten voor het goed beschermen van het getal (beide constructoren moeten niet te lage waardes toestaan)

Uiteraard geen punten als te korte routes gewoon geaccepteerd worden

1 punt aftrek als het op de grenswaarde verkeerd gaat

1 punt aftrek als de default waarde van de afstand niet goed was (moest -1 zijn) Let er ook op, dat als je een setAfstand(...) methode gebruikt (aangeroepen in je constructor-code) dat die dan private is, omdat anders de waarde alsnog veranderd kan worden.

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.

Puntenverdeling:

1 punt voor het niet-crashen. De verkeerde waarde returnen (-1 was de bedoeling) in geval van een fout, kostte 1 punt.

4 punten voor de manier waarop, namelijk met een juiste exception: In de vraag werd duidelijk gemaakt dat je niet Exception mocht gebruiken maar wel klassen die daarvan overerven. Concreet kon je RuntimeException en ArithmeticException gebruiken (de laatste is de meest specifieke).

Als je toch Exception had, dan kostte je dat 2 punten (toont in ieder geval aan dat try-catch foutafhandeling deels onder de knie is.

Als je if/else gebruikte ipv try-catch, kostte dat 4 punten

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
```

Puntenverdeling:

2 punten voor het hebben van een statisch printMinMax functie die iets deed, en 1 punt voor het hebben van statische attributen om de vraag te kunnen oplossen (n.b. niet gekeken naar public/private)

de overige 7 punten voor de correcte werking. De meest voor de hand liggende manier was, om 2 static int's te maken, en in de ene het minimum en in de ander het maximum op te slaan (met wat if-else logica in beide constructors).

3 punten waren voor het minimum (iets lastiger dan het maximum), 4 voor het maximum. Er is hier niet gekeken naar de precieze formulering (de hele zin) maar alleen naar de getallen.

Er is 1 punt aftrek, als de waardes niet juist zijn nadat maar 1 Route-object gemaakt is (min en max zouden dan hetzelfde moeten zijn)

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
```

Puntenverdeling:

4 punten voor de correcte signatuur. Er werd gevraagd naar de equals die van Object overgeerfd werd, dus: equals(Object)

6 punten voor de correcte werking.

Als twee rechthoeken met identieke waardes false opleverden: alle punten kwijt.

als twee rechthoeken met identieke waardes, maar de ene gedraaid (dus bv. 2x4 en 4x2), 'false' opleverden: 2 punten aftrek.

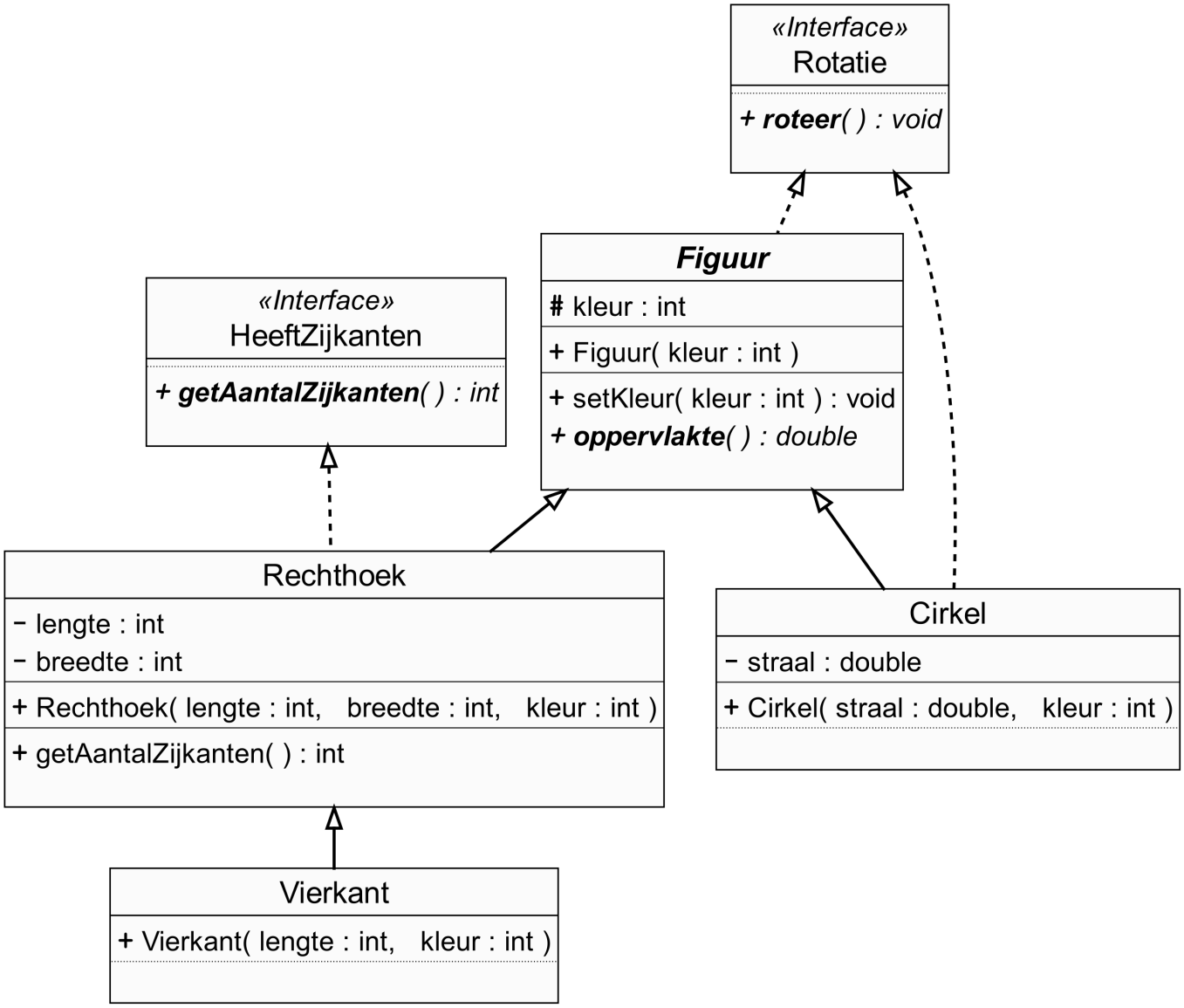
Als de rechthoeken alleen verschilden in 1 waarde (lengte of breedte) en er werd 'true' gereturned: 1 aftrek.

Als alleen de kleur ongelijk was, maar er toch 'true' als uitkomst was, 2 punten aftrek

Niet dezelfde vorm (lengte en breedte anders) en niet dezelfde kleur, maar toch true: 2 punten aftrek.

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
```

Puntenverdeling:

Aangezien deze vraag vooral gaat om overerving/interfaces, gaan vrijwel alle punten naar het correct implementeren van de relaties in het klassendiagram, namelijk 13. De overige 2 zijn voor correcte compilatie (n.b. er is dus niet gekeken de code voor de oppervlakte van een cirkel correct overgetypt is uit de toets, of dat voor rechthoeken de opp. met lengte * breedte berekend wordt.)

De voorbeeldcode van de werking was vooral gegeven met als doel de vraag concreter te maken en om te helpen met zien wat er allemaal geimplementeerd moest worden, meer niet (m.u.v. de werking van toString voor Rechthoek die rekening moest houden met Vierkant omdat die zelf geen toString mocht hebben).

1 punt voor elke fout. Dat kunnen zijn: missende of overbodige attributen, missende functies of overbodige functies (dus ook: juiste functienaam, maar verkeerde input parameters/return) en verkeerde extends of implements. Als je een extends mist en ook een functie vergeten bent die bij de te extenden klasse hoort, dan is dat 2 fouten en wordt niet als doorrekenfout beschouwd.

Er zijn meerdere geldige opties (eentje staat in voorbeelduitwerking). Het kan zijn dat we een geldige optie over het hoofd hebben gezien. Denk aan varianten op: Cirkel hoeft Rotatie niet te implementeren omdat Figuur die interface al implementeert met een (niet abstracte) methode en Cirkel die overerft. Let hierop bij de inzage!

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
{
	/*PARAMSUBST_START
	PARAMSUBST_END */
	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`.

Puntenverdeling:

10 punten voor de juiste uitkomst (dat de naam van de F16 wel aangepast is door de Garage).

2 punten aftrek als de garage nog steeds een nieuw object aanmaakt maar daar niks mee doet. Een garage update een vliegtuig, en het is niet de bedoeling dat een vliegtuig ineens op myserieuze wijze gekopieerd wordt. Dat was juist de bug.

5 ipv. 2 punten aftrek als de garage het nog steeds een nieuw aangemaakt vliegtuig ook nog opslaat in de hangar op de plek waar het originele vliegtuig stond (Dan kan de naam van het originele object natuurlijk nog steeds correct veranderd zijn)

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.

Puntenverdeling:

2 punten voor juist aanmaken van atrribuut (Vliegtuig[] hangar;). Niet gelegd op public/private.

3 punten voor het initialiseren van de array in de constructor (met = new Vliegtuig[grootte];)

geen punten voor de hele opgave, als je een ArrayList gebruikt ipv een array

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
```

Puntenverdeling

toevoegen: 3 punten voor het correct toevoegen van een vliegtuig aan de hangar.

2 punten aftrek als het toe te voegen vliegtuig op een plek gezet kon worden die al bezet was

2 aftrek als er geen bericht werd geprint als een vliegtuig niet op een plek gezet mocht worden omdat ie al bezet was.

3 punten voor het zorgen dat de zetBinnen-methode niet kan crashen. 2 punten aftrek als dat met try-catch gebeurde (want dat mocht niet).

1 punt aftrek voor een verkeerde/missende foutmelding.

1 punt aftrek als het fout gaat als je een vliegtuig op bv. plek -1 of een ander negatief getal wilt zetten.

1 punt aftrek als het mis gaat als je een vliegtuig op een plek wil zetten die te hoog is (>= lengte van array).

printen: 4 punten voor de correcte werking van print().

1 punt aftrek als een lege plek niet als --- maar gewoon met null werd weergegeven.

1 punt aftrek als de plekken (getallen) niet werden geprint

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
```

Puntenverdeling:

3 punten voor het eruit halen van het vliegtuig dat op de aangegeven plek staat.

2 punten voor het leegmaken van de plek in de hangar waar het vliegtuig stond

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
```

Puntenverdeling:

Punten gesplitst in 7 punten voor het vinden van de vrije plek als er een vrije plek is, en 3 voor de werking als er geen plek vrij was

nog wel een plek vrij (7 punten): Als het vliegtuig iig. op de eerste vrije plek terecht komt. Maar 3 punten eraf als de plekken erna ook gevuld worden (doordat de for-loop niet afgebroken wordt).

Ook 2 punten aftrek als de return waarde niet klopt. (n.b. als je dus meer dan 1 vrije plek vult met het vliegveld en een verkeerde waarde returnt: 5 punten eraf)

3 punten aftrek als het goed gaat aan de randen, dus als plek 0 vrij is en ook als het juist alleen nog maar de laatste plek in de array

niks is meer vrij (3 punten): Geen punten als de hangar-array gewijzigd is ondanks dat die al vol was, en 1 punt aftrek als de index niet -1 is

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!