hogeschool Windesheim

Techniek - HBO - ICT

Naam: Studentnummer: Klas:

Practicum JAVA Theorie (108924)

Vakcode	: ICT.P.JAVA1.V20 (ICT.P.JAVA.V19, ICT.P.JAVA.V18, ICT.P.JAVA.V17, ICT.P.JAVA.V16) (t1)
Datum	: vrijdag 9 april 2021
Tijd	: 11.30 - 13.30 uur

Klas:	Lokaal:	Aantal:
ICTM2a t/m i , ICTM2n t/m r , ICTM2tt	volgt	390

Opgesteld door	: Wilco Moerman
Docenten	: WPH01, LNR08, DSW01, DFG01, NMJ01, MNC07 VEE02, RWM02, FAP02, CNW01, SSW02, KGW01, DAS01, CSI01
Gecontroleerd door	: Wouter Keuning, Wietske Doornbos, Gerben de Wolf, Ilja Clabbers
Rekenmachine	: alle rekenmachines toegestaan
Literatuur	: toegestaan
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:			

Deze pagina is expres leeg gelaten

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 die in deze toets getoond wordt, kun je vinden in het het bestand **`"startsituatie_theorie_9-april-2021.zip"`** in de folder `"inleverpunt theorie 9 april"`.

De getoonde voorbeeld- en testcode is niet volledig. Je zult zelf moeten testen/onderzoeken of je code doet wat deze moet doen volgens de vraag.

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.

Opgave 1: Beschermen en loggen [30 punten]

De klasse `Beschermd` heeft een attribuut `getal` dat 'beschermd' moet worden.

inhoud van klasse Beschermd

```
public class Beschermd
{
	// moet altijd groter of gelijk aan 10 zijn.
	public int getal = 12345;
}
```

a) [5 punten]
Zorg dat onderstaande code werkt en de gewenste output geeft (zoals aangegeven in de comments).

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

```
public static void main(String[] args)
{
	Beschermd a = new Beschermd( 88888 ); // ok
	System.out.println( a );              // Beschermd: getal = 88888
}
```

(de "ok" hoort bij opgave (b))

b) [10 punten]
Pas encapsulatie toe om `getal` te beschermen tegen ongeldige wijzigingen van buitenaf. Het mag nooit kleiner zijn dan **10.**

Voeg de setter toe en pas de constructor aan:

- Pas het getal alleen aan als de waarde groter of gelijk aan **10** is.
- Print **`"ok"`** in de console als het aanpassen mag en **`"mag niet"`** als de aanpassing illegaal is (en dus niet uitgevoerd wordt).

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

```
public static void main(String[] args)
{
      Beschermd a = new Beschermd( 777777 ); // ok
      a.setGetal( 2 );                       // mag niet
      System.out.println( a );               // Beschermd: getal = 777777
      a.setGetal( 333 );                     // ok
      System.out.println( a );               // Beschermd: getal = 333

      Beschermd b = new Beschermd( -1 );     // mag niet
      System.out.println( b );               // Beschermd: getal = 12345
}
```

c) [15 punten]

Het management wil weten hoevaak er geprobeerd wordt legale of illegale waardes aan een `Beschermd`-object te geven.

Zorg ervoor dat de `printLogging`-methode de actuele stand van de aantallen uitprint.

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

```
public static void main(String[] args)
{
      Beschermd.printLogging();              // # ok = 0, # mag niet = 0

      Beschermd a = new Beschermd( 777 );    // ok
      Beschermd b = new Beschermd( -1 );     // mag niet
      b.setGetal( 123 );                     // ok
      Beschermd c = new Beschermd( 98765 ); // ok

      Beschermd.printLogging();              // # ok = 3, # mag niet = 1

      Beschermd d = new Beschermd( 100 );    // ok
      d.setGetal( -1 );                      // mag niet
      Beschermd.printLogging();              // # ok = 4, # mag niet = 2
}
```

Opgave 2: Plaatjes delen [15 punten]

Iemand heeft voor ons een geweldig nieuw platform bedacht, genaamd `PicPoc`.

Met `PicPoc` kun je heel makkelijk een heel erg klein `Plaatje` (met twee pixels) delen met de hele wereld!

code van de klassen PicPoc en Plaatje:

```
public class PicPoc {

	private Plaatje img;

	public void uploadPlaatje( Plaatje i ) { img = i; }

	public Plaatje downloadPlaatje() {
		img.toon(); // dit plaatje wordt gedownload.
		return img;
	}

	public void toon() { img.toon(); }
}
```

```
class Plaatje {

	public int pixel1;
	public int pixel2;

	public Plaatje( int pixel1, int pixel2 ) {
		pixel1 = pixel1;
		pixel2 = pixel2;
	}

	public void toon() {
		System.out.println( "Plaatje: " + pixel1 + ", " + pixel2 );
	}
}
```

Helaas zijn er twee bugs ontdekt:

- **bug #1:** Na het *uploaden* ontdekken we dat alle pixels op 0 staan.

- **bug #2:** Er gaat iets mis bij het *downloaden* voor eigen gebruik: als iemand een gedownload plaatje aanpast, verandert het plaatje in ons account ook.

Repareer beide bugs.
Voeg ook comments toe waarin je kort uitlegt *waarom* beide bugs ontstonden.

Hieronder zie je hoe `PicPoc` *zou* moeten werken.
Als je deze code zelf runt, zullen de bugs optreden.

In comments staan de verwachte waardes van de plaatjes

```
public static void main(String[] args)
{
      PicPoc account1 = new PicPoc();
      Plaatje img = new Plaatje( 1, 2 );                  // Plaatje: 1, 2
      account1.uploadPlaatje( img );

      // klopt het plaatje nog wel?
      account1.toon();                                    // Plaatje: 1, 2

      // deel het met de wereld!
      Plaatje gedownload = account1.downloadPlaatje();

      // iemand verandert het gedownloade plaatje
      gedownload.pixel1 = 777;
      gedownload.toon();                                  // Plaatje: 777, 2

      // Bekijk het plaatje in de account nog eens.
      // Dat zou natuurlijk hetzelfde moeten zijn als aan het begin
      account1.toon();                                    // Plaatje: 1, 2
}
```

Opgave 3: Verkiezingen [25 punten]

De verkiezingen zijn net geweest, en dat gaan we modelleren mbv. een ARRAY:

Klasse Verkiezingen:

```
public class Verkiezingen {

      // vraag (a)
      public Verkiezingen( int aantalKandidaten ) { /*...todo...*/ }

      // vraag (b)
      public void stemOp( int kandidaatnr ) { /*...todo...*/ }

      // vraag (b)
      public void print() { /*...todo...*/ }
}
```

a) [5 punten]
Voeg als attribuut een `int`-array met de naam `stemmen` toe aan klasse `Verkiezingen`. (In deze `array` gaan de stemmen op de kandidaten bijgehouden worden.)

Maak ook de gegeven constructor af. Deze moet ervoor zorgen dat de `stemmen`-array aangemaakt wordt met de juiste grootte (namelijk: `aantalKandidaten`).

b) [10 punten]
Maak de methodes `stemOp` die het **nummer** van de kandidaat als input heeft.

- Kandidaten zijn genummerd van **1** t/m `aantalKandidaten`.
- De methode verhoogt de waarde van `stemmen` voor die kandidaat met 1.

Zorg ervoor dat deze methode **niet** kan crashen.

- Je mag hier **geen** `if ... else ...` logica voor gebruiken.
- Als de kandidaat niet bestaat, wordt een foutmelding geprint.

Maak ook de `print`-methode, die het aantal stemmen per kandidaat uitprint. Hieronder zie je de werking van beide methodes.

werking van stemOp en print (in de comments staat de verwachte output)

```
public static void main(String[] args)
{
      Verkiezingen v = new Verkiezingen( 3 );
      v.stemOp( 2 );
      v.stemOp( 1 );
      v.stemOp( 1 );
      v.stemOp( 1 );
      v.stemOp( 777 );                          // kandidaat 777 bestaat niet
      v.print();
                                                // kandidaat nr 1 heeft 3 stemmen
                                                // kandidaat nr 2 heeft 1 stem
                                                // kandidaat nr 3 heeft 0 stemmen
}
```

c) [10 punten]
Maak de methode `int getWinnaar()`, die het **nummer** van de kandidaat met de meeste stemmen returnt.

(Je hoeft geen rekening te houden met kandidaten met een gelijk aantal stemmen.)

bepalen van de winnaar (comments bevatten de verwachte output)

```
public static void main(String[] args)
{
      Verkiezingen v = new Verkiezingen( 4 );
      v.stemOp( 1 );
      v.stemOp( 2 );
      v.stemOp( 1 );
      v.stemOp( 1 );
      v.print();
                                                // kandidaat nr 1 heeft 3 stemmen
                                                // kandidaat nr 2 heeft 1 stem
                                                // kandidaat nr 3 heeft 0 stemmen
                                                // kandidaat nr 4 heeft 0 stemmen
      //
      System.out.println( v.getWinnaar() );     // 1
}
```

Opgave 4: bankbiljetten [15 punten]

In deze opgave moet je de klassen `Bankbiljet` en `Munteenheid` repareren.

code van Bankbiljet en Munteenheid:

```
public class Bankbiljet {

	private int getal;
	private Munteenheid munteenheid;

	public Bankbiljet() { }

	public Bankbiljet( int getal, Munteenheid munteenheid ) {
		this.getal = getal;
		this.munteenheid = munteenheid;
	}

	// gelijk als getal en munteenheid gelijk zijn.
	public boolean equals( Bankbiljet obj ) {
		return
				this.getal == obj.getal
				&& this.munteenheid == obj.munteenheid;
	}
}
```

```
class Munteenheid {

	private String valuta;

	public Munteenheid( String valuta ) {
		this.valuta = valuta;
	}
}
```

De `equals`-methode die `Bankbiljet` van `Object` overerft en die overschreven (*override*) moet worden, werkt nog niet goed.

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

- return't *alleen* `true` als de `getal`-waardes gelijk zijn en de `munteenheid`-objecten dezelfde `valuta` hebben.
- als de input geen `Bankbiljet` is, dan is de uitkomst uiteraard `false`.

Repareer de code van beide klassen, zodat ze voldoen aan bovenstaande eisen.

Let op: De *access modifiers* van methodes en attributen mogen *niet* veranderd worden en je mag geen *getters* toevoegen.

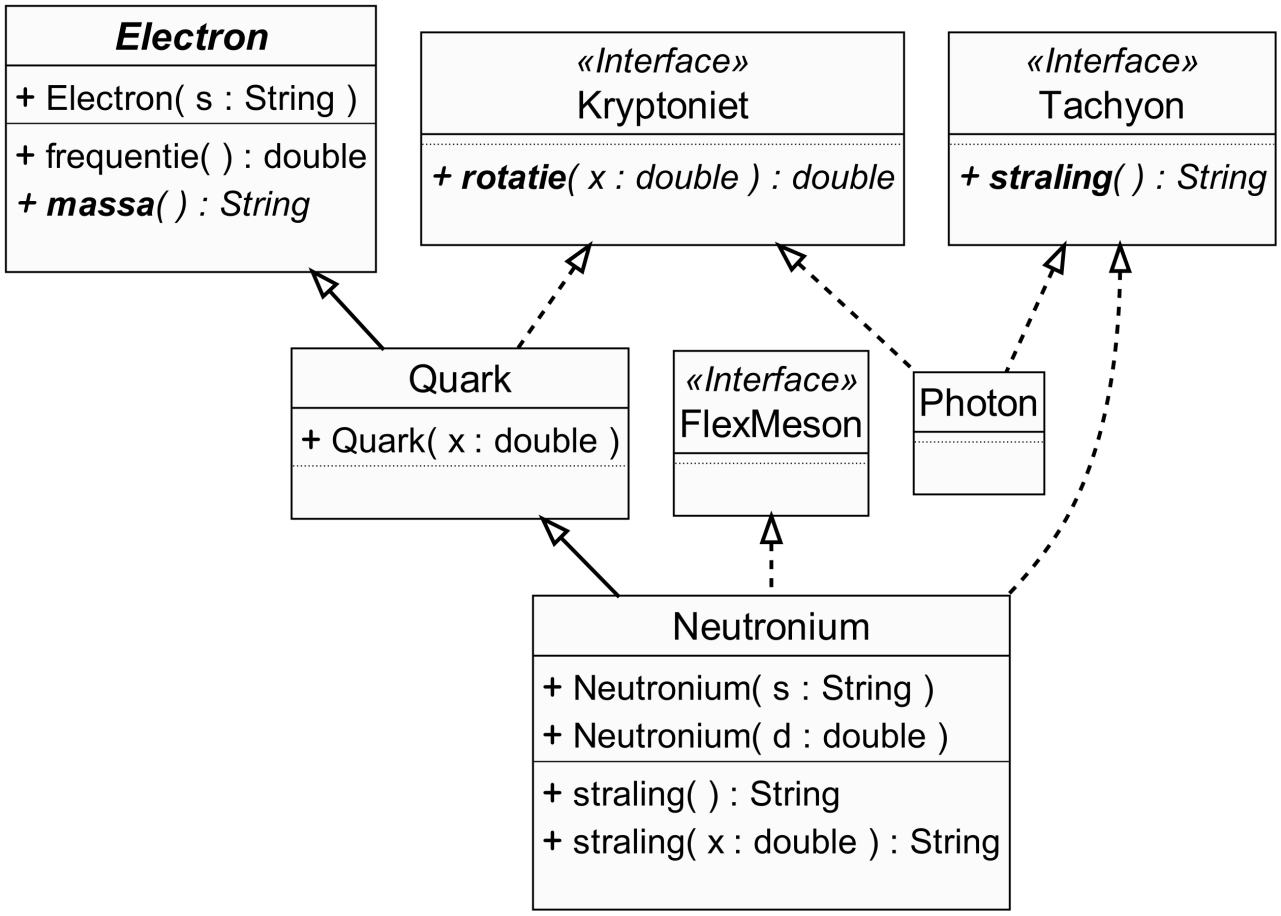
werking van equals(Object) van Bankbiljet (verwachte output in comments):

```
public static void main(String[] args)
{
	// twee keer dezelfde Munteenheid
	Munteenheid euro = new Munteenheid( "euro" );
	Munteenheid euro2 = new Munteenheid( "euro" );

	// twee keer hetzelfde Bankbiljet
	Bankbiljet tientje = new Bankbiljet( 10, euro );
	Bankbiljet tientje2 = new Bankbiljet( 10, euro2 );

	System.out.println( tientje.equals( tientje2 ) );	// verwacht: true

	// verschillende Munteenheid, zelfde getal
	Munteenheid dollar = new Munteenheid( "dollar" );
	Bankbiljet tienDollar = new Bankbiljet( 10, dollar );

	System.out.println( tientje.equals( tienDollar ) );	// verwacht: false

	// vergelijken met iets dat geen Bankbiljet is
	System.out.println( tientje.equals( "10 euro" ) );	// verwacht: false
}
```

Opgave 5: Klassendiagram [15 punten]

Van de elementaire deeltjes die door de deeltjesversneller van CERN ontdekt zijn, werd een klassendiagram gemaakt.

We weten niet wat de woorden in het klassendiagram allemaal betekenen, dus een zinnige *inhoud* kunnen we de constructors en methodes niet geven. Maar je kunt wel een correcte implementatie maken van de getoonde *relaties*.



(Abstracte onderdelen zijn ***vetgedrukt en cursief*** weergegeven) .

a) [10 punten]
Maak de code voor alle klassen/interfaces en hun relaties.

- de constructors mag je in vraag (a) negeren.
- methodes hoeven niks te doen behalve iets returnen.
- Als methodes een **`String`** returnen, gebruik dan de **naam van de klasse en methode** bv. `return "klasse = Electron en methode = straling";`.
- Als methodes een getal returnen, gebruik dan **`return 42;`**.

Let op: methodes die een klasse moet implementeren vanwege overerving of interfaces, staan in het klassendiagram *niet* vermeld in de klasse zelf. Je moet dus zelf bepalen welke methodes in welke klassen wel opgenomen en geimplementeerd (geprogrammeerd) moeten worden.

b) [5 punten]
In het klassendiagram staan ook enkele constructors. Voeg die nu ook toe aan je code:

- Maar **één** constructor *per klasse* mag het keyword **`super(...)`** gebruiken.
- Als je ergens een getal nodig hebt, gebruik dan **42**.
- Als je ergens een `String` nodig hebt, gebruik dan de **naam** van de klasse.
- De constructors hoeven niks op te slaan.

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 9 april"`**

Als ELO je vertelt dat je het bestand niet kunt uploaden omdat er al een bestand met dezelfde naam bestaat, pas je bestandsnaam dan iets aan, bv. door er de datum aan toe te voegen).

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