Vraag 1.
Gegeven is een algemene klasse Persoon:

```
public class Persoon {
  private int leeftijd;
  ...
}
```

Nu wil je een klasse Student maken. Een Student is een subklasse van een Persoon. Hoe geef je dit aan in Java?

☐
```
public class Student extends Persoon {
   ...
}
```

☐
```
Persoon student = new Persoon () ;
```

☐
```
public class Persoon extends Student {
  ...
}
```

Vraag 2.
Elke student volgt een opleiding. Sommigen zelfs meer dan één. Welke van onderstaande mogelijkheden is het meest geschikt om aan te geven welke opleiding(en) een student volgt:

☐ Als een integer-attribuut
☐ Als een methode toString()
☐ Als een array van String-attributen
☐ Als een subklasse van Student

Vraag 3.
Elke student heeft een studentnummer. Hoe zou je dit toevoegen aan de klasse Student?

☐ Als een integer-attribuut
☐ Als een methode `toString()`
☐ Als een array van integer-attributen
☐ Als een subclasse van `Student`

Vraag 4.
Je hebt thuiswonende en uitwonende studenten. Van thuiswonende studenten heb je ook gegevens van de ouders nodig, van uitwonende studenten alleen de adresgegevens. Hoe zou je uit- en thuiswonende studenten kunnen toevoegen:

☐ Als een boolean-attribuut
☐ Als een methode `isUitwonendeStudent()`
☐ Als subklasses van Student
☐ Als een array van boolean-attributen

Vraag 5.
In je programma heb je een array nodig waar 50 integers in kunnen. Dit array wil je kunnen aanduiden met de variabelenaam `getallen`.
Hoe declareer je zo'n array:

- [ ] `int getallen ;`
- [x] `int[] getallen ;`
- [ ] `int[50] getallen ;`

Vraag 6.
Hoe initialiseer je zo'n array:

- [ ] `getallen = new getallen[50] ;`
- [x] `getallen = new int[50] ;`
- [ ] `getallen[50] = new int[50] ;`

Vraag 7.
Veronderstel dat je het gehele array wilt vullen met nullen, Hoe doe je dat?

- [x]
```
getallen = 0 ;
```

- [ ]
```
for (int i=0 ; i < getallen.length ; i++) {
   i = 0 ;
}
```

- [ ]
```
for (int i=0 ; i < getallen.length ; i++) {
   getallen = 0 ;
}
```

- [ ]
```
for (int i=0 ; i < getallen.length ; i++) {
   getallen[i] = 0 ;
}
```

Vraag 8.
Gegeven zijn de volgende declaraties:

```
int i ;
double d ;
```

Welke van de volgende regels is correct en welke incorrect:

`i = 10.0 ;`	correct / incorrect
`d = 10 ;`	correct / incorrect
`d = i ;`	correct / incorrect
`i = d ;`	correct / incorrect
`i = (int) d ;`	correct / incorrect

Vraag 9.
Gegeven is onderstaande code:

```
int getalA = 15;
int getalB = 30;

if (2 * getalA != getalB) {
  System.out.println("aap");
} else {
  if (getalA < getalB) {
    System.out.println("noot");
  } else {
    System.out.println("mies");
  }
}
```

Wat is de uitvoer van deze code?

_____
_____
_____
_____
_____

Vraag 10.
Wat is (gegeven de code uit de vorige vraag) het resultaattype en wat is de waarde van de volgende expressie:

```
(double) getalA ;
```

type: _____
waarde: _____
_____
_____

Vraag 11.
Gegeven zijn de volgende klassen:

```
public class Fabriek {
  private String naam ;

  public Fabriek (String titel) {
    this.naam = titel ;
  }

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

```
public class Medewerker {
   Fabriek fabriek ;
   String naam ;

   public Medewerker (String naam, Fabriek fabriek) {
      this.naam = naam ;
      this.fabriek = fabriek ;
   }

   public String toString() {
     return naam + " werkt bij " + fabriek.toString() ;
   }
}
```

en in de main-methode van een testklasse:

```
Fabriek f = new Fabriek ("strokarton") ;
Medewerker jan = new Medewerker ("Piet",f) ;
System.out.println(jan);
```

Wat is hier de output?

Vraag 12.
Beschouw weer de code uit vraag 11, maar nu met code in de main-methode alsvolgt:

```
Fabriek f = new Fabriek ("De Fabriek") ;
Medewerker jan = new Medewerker ("Jan", f) ;
Medewerker piet  = new Medewerker("Piet", f) ;
System.out.println(jan) ;
System.out.println(piet) ;
```

Wat is nu de output?

_____
_____
_____
_____
_____

Vraag 13. (strikvraag)
Wat is (volgens de code uit vraag 12) waar m.b.t. het `fabriek`-attribuut van `jan` en van `piet`?

- ☐ Het `fabriek`-attribuut wijst bij beide hetzelfde object aan
- ☐ Het `fabriek`-attribuut wijst bij `jan` en bij `piet` een verschillende object aan, maar de waarde van het `naam`-attribuut in beide `fabriek`-attributen is gelijk
- ☐ Het `fabriek`-attribuut wijst bij `jan` en bij `piet` een verschillende object aan en de waarden van de attributen van deze `fabriek`-objecten zijn eveneens verschillend

Vraag 14.
Gegeven is:

```
public class Teller {
  private int waarde ;

  public Teller () {
    waarde = 0 ;
  }
  public void verhoog () {
    waarde++ ;
  }
}
```

En verder ergens in een testklasse in de main methode:

```
int i = 0 ;
Teller mijnTeller = new Teller() ;
while (i < 10) {
  mijnTeller.verhoog() ;
}
```

Wat is er mis met het herhalingsstatement in deze code?

Vraag 15.
Verbeter het herhalingsstatement uit vraag 14:

Vraag 16.
Herschrijf het herhalingsstatement uit vraag 15 als een `for`-lus:

Vraag 17.
De klasse teller uit de vorige vraag wordt nu uitgebreid met de volgende methode:

```
public void verhoog (int waarde) {
  this.waarde = this.waarde + waarde ;
}
```

En de mainmethode wordt herschreven tot:

```
Teller mijnTeller = new Teller () ;
mijnTeller.verhoog(9) ;
mijnTeller.verhoog() ;
```

Kies en vul aan:

☐ Deze drie statements zijn *niet correct*, want _____

☐ Deze statements zijn *correct* en na afloop is de waarde van het attribuut `waarde` in `mijnTeller`? _____

Vraag 18 *(deze vraag telt dubbel).*

```
public class Flat {

  int aantalHuisnummers ;
  int verdiepingen ;
  int aantalBewoners ;
  Bewoner[] bewoner ;

  public Flat (int verdiepingen, int aantalHuisnummers){
    this.aantalHuisnummers = aantalHuisnummers ;
    bewoner = new Bewoner[aantalHuisnummers] ;
    this.verdiepingen = verdiepingen ;
    this.aantalBewoners = 0 ;
  }

  public void voegToe (Bewoner b) {
    b.setFlat (this) ;
    bewoner [aantalBewoners] = b ;
    aantalBewoners++ ;
  }
  public int getAantalBewoners () {
    return this.aantalBewoners ;
  }
  public boolean woontIn (Bewoner b, int huisnummer) {
    for (int i=0 ; i < aantalHuisnummers ; i++) {
      if (bewoner[i] == b) {
        return true ;
      }
    }
    return false ;
  }

}
-----------------------------------------------------
public class Bewoner {

  String naam ;
  Flat f ;

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

  public void setFlat(Flat f) {
    this.f = f ;
  }

}
```

a. Vul de volgende testmethode aan volgens het JUnit principe zodat wordt gecontroleerd of na creatie van een nieuw Flat-object het aantal bewoners nog nul is.

```
public void testFlat () {
  Flat flat = new Flat (5, 150) ;
  ____________________________________________________________
  ____________________________________________________________
  ____________________________________________________________
  ____________________________________________________________
  ____________________________________________________________
  ____________________________________________________________
  ____________________________________________________________
}
```

b. Vul de volgende testmethode aan volgens het JUnit principe zodat wordt gecontroleerd of toevoegen van twee bewoners aan een gecreëerde nieuw Flat-object erin resulteert dat deze beide bewoners werkelijk in de flat wonen.

```
public void testToevoegen () {
  Flat flat = new Flat (5, 150) ;
  Bewoner b1 = new Bewoner ("bewoner1") ;
  Bewoner b2 = new Bewoner ("bewoner2") ;

  ____________________________________________________________
  ____________________________________________________________
  ____________________________________________________________
  ____________________________________________________________
  ____________________________________________________________
  ____________________________________________________________
  ____________________________________________________________
  ____________________________________________________________
  ____________________________________________________________
}
```

c. Welke methode ontbreekt nog in de klasse Bewoner om de *volledige* relatie tussen bewoner en flat te kunnen testen?

```
____________________________________________________________
____________________________________________________________
____________________________________________________________
____________________________________________________________
____________________________________________________________
```

d. Vraag 19 (Bonusvraag)

Deze vraag hoeft niet beantwoord te worden, maar levert bij goede beantwoording wel extra punten op.

Gegeven is:

```
public class Bonus {
  private int getal ;
  public Bonus () {
    getal = 10 ;
  }
  public int dubbel (int getal) {
    return getal * 2 ;
  }
  public void helft (int getal) {
    this.getal = getal / 2 ;
  }
  public int triple (int getal) {
    getal = getal * 3 ;
    return this.getal ;
  }
  public String toString () {
    return "getal: " + getal ;
}
```

En voorts ergens in een mainmethode van een Testklasse:

```
Bonus b = new Bonus() ;
System.out.println (b) ;
b.helft(8) ;
System.out.println (b) ;
System.out.println (b.dubbel (40)) ;
System.out.println (b.triple (5)) ;
System.out.println (b) ;
```

Geef aan wat er achtereenvolgens op de uitvoer verschijnt:

UML klassediagram

Vraag 20.
Gegeven is de volgende omschrijving:

Videotheek!
In een videotheek worden dvd's, cd's en soms nog een videoband verhuurd. Klanten hebben een klantenpas met daarop een uniek nummer. Een klant mag een dvd, cd of videoband maximaal drie dagen huren voor een bepaald huurbedrag. Vaste klanten krijgen op dat bedrag een korting. Daarnaast is er voor alle klanten een apart weekendtarief, waarbij de dvd, cd of videoband van vrijdagmiddag tot maximaal maandagochtend gehuurd mag worden.
Indien een dvd, cd of videoband te laat wordt teruggebracht, moet een boete worden betaald.
De videotheekhouder wil op elk moment kunnen zien welke dvd's, cd's en banden uitgeleend zijn en ook makkelijk kunnen terugvinden welke klant ooit welke dvd, cd of band heeft geleend en of toen eventueel ook een boete is betaald.

Maak hierbij een zo volledig mogelijk uitgewerkt **UML klassediagram**. Denk aan de structuur van de te gebruiken klasses en voeg relevante attributen en methoden toe.
Opm. geen Javacode uitschrijven dus, alleen een uml-diagram maken!

Noteer Uw oplossing hier: