DOMINIEK VANDEWALLE

LEREN PROGRAMMEREN

EEN OBJECT-GEORIËNTEERDE AANPAK,

JAVA IN BLUEJ

TWEEDE EDITIE

HANDLEIDING

acco

HANDLEIDING

LEREN PROGRAMMEREN

EEN OBJECTGEORIËNTEERDE AANPAK
JAVA IN BLUEJ

Tweede Editie

Dominiek Vandewalle

Acco Leuven / Den Haag

Handleiding bij ISBN 978-94-6344-694-5

Omslagontwerp: www.frisco-ontwerpbureau.be

Bewerking Omslagontwerp: Dominiek Vandewalle

Inhoudsopgave

Inleiding 0

In deze handleiding vind je een voorbeeldoplossing van alle vragen in het handboek **Leren Programmeren, een object-georiënteerde aanpak, Java in BlueJ** (ISBN 978-94-6344-694-5).

Wat moet je weten:

- De oplossingen werden zonder verdere uitleg bij de vragen geschreven.
- Commentaar bij codefragmenten werd verwijderd.
- De oplossingen volgen de stijl die in het handboek gehanteerd werd. Een professioneel programmeur zal de antwoorden in vele gevallen korter kunnen programmeren.

Alvorens naar de oplossingen van de oefeningen kijken, nog een aantal didactische wenken.

0.1 BlueJ

Sommige vragen handelen over het ontdekken van een project. Bij deze oefeningen is het belangrijk dat leerlingen zelf aan de slag gaan. Zeker in de eerste hoofdstukken is het belangrijk om de werking van de programmeeromgeving *BlueJ* onder de knie te krijgen. Laat de begrippen klasse, klassendefinitie, object in de objectenbank voldoende aan bod komen. Laat objecten uitvoerig inspecteren. Door de eenvoud van *BlueJ* zal je merken dat het leren werken met *BlueJ* geen grote uitdaging is.

0.2 Klassendefinitie opbouwen

Leerlingen die hoofdstuk 2 aanvatten zonder duidelijk het verschil te weten tussen een klasse en een object zal je onderweg gauw verliezen. Hoofdstukken 1 en 2 zijn zo eenvoudig mogelijk gemaakt voor de beginnende programmeur. Moeilijke begrippen zijn bereik en levensduur. Trek voor deze onderwerpen voldoende tijd uit. Ook de `this.` en de puntnotatie in het bijzonder is een belangrijk concept. Voor een beginnend programmeur is het bijvoorbeeld minder belangrijk om te weten hoe groot de grootste `int`-waarde is.

0.3 Analyse

Maak bij elke oefening steeds een analyse van het probleem. Dit kan mondeling, schriftelijk, in *BlueJ* enz. Een klassendiagram maak je best op papier of met behulp van UML-software. Hetzelfde geldt voor het programmeren van methoden. Aangezien commentaar schrijven wel wat tijd kost en het commentaar bij de code van een beginnend programmeur goed lijkt op de code zelf, wordt de analyse van methoden soms snel achterwege gelaten. Omdat dit toch wel belangrijk is, geef ik een mogelijk stappenplan.

Oefening 0.1 Deze oefening komt uit hoofdstuk 6. De `ArrayList lades` bevat objecten van het type `PapierLade`.

- `void vulPapierlade(String, int)`
  Deze methode vult het papier bij van de papierlade met het papiertype dat met de eerste parameter wordt meegegeven. Het aantal bladen waarmee je de papierlade aanvult, vind je in de tweede parameter.

Antwoord – In de eerste stap schrijven we de header van de methode. Vergeet niet om onmiddellijk commentaar te schrijven voor het genereren van een API-pagina.

```
/**
 * Aanvullen van een papierlade met aantal bladen.
 *@param papiertype  Het papiertype van de bladen
 *@param aantal      Het aantal bladen waarmee je wenst aan te vullen
 */
 public void vulPapierlade(String papiertype, int aantal)
 {

 }
```

Antwoord – Vervolgens schrijven we in commentaar en in mensentaal de functionaliteit van de methode neer.

```
/**
 * Aanvullen van een papierlade met aantal bladen.
 *@param papiertype  Het papiertype van de bladen
 *@param aantal      Het aantal bladen waarmee je wenst aan te vullen
 */
 public void vulPapierlade(String papiertype, int aantal)
 {
     //Zoek in de collectie van papierlades naar de index
     //van een papierlade met het juiste papiertype.

     //Index gevonden?
     //--> Ja, vul papier aan met het gegeven aantal bladen.
 }
```

0.4 Programmeren

Pas nadat we in woorden genoteerd hebben wat we willen programmeren, gaan we effectief aan de slag.

Antwoord – Nu is het tijd om een skelet van controlestructuren te bouwen.

- We hebben een while-lus nodig die door een collectie zoekt tot er iets gevonden wordt.
- We hebben een selectiestatement nodig om te controleren of we een geschikte papierlade gevonden hebben.

```
/**
 * Aanvullen van een papierlade met aantal bladen.
 *@param papiertype  Het papiertype van de bladen
 *@param aantal      Het aantal bladen waarmee je wenst aan te vullen
 */
 public void vulPapierlade(String papiertype, int aantal)
 {
     //Zoek in de collectie van papierlades naar de index
     //van een papierlade met het juiste papiertype.
     while(...)
     {
         ...
     }

     //Index gevonden?
     if(...)
     {
         //--> Ja, vul papier aan met het gegeven aantal bladen.
     }
 }
```

Antwoord – We bouwen nu verder het skelet van de `while`-lus op. De voorwaarde in het `if`-statement behoort tot onze parate kennis.

```
/**
 * Aanvullen van een papierlade met aantal bladen.
 *@param papiertype  Het papiertype van de bladen
 *@param aantal      Het aantal bladen waarmee je wenst aan te vullen
 */
public void vulPapierlade(String papiertype, int aantal)
{
    int index = 0;

    //Zoek in de collectie van papierlades naar de index
    //van een papierlade met het juiste papiertype.
    while(index < lades.size() && ...)
    {
        index++;
    }

    //Index gevonden?
    if(index < lades.size())
    {
        //--> Ja, vul papier aan met het gegeven aantal bladen.
    }
}
```

Antwoord – Nu pas komen de twee moeilijke expressies aan bod:

- Een voorwaarde om verder te zoeken in de collectie van papierlades. Dit is hetzelfde als controleren of de papierlade waar we via de index aanbeland zijn, geen goede papierlade is. Geen goede papierlade betekent verder zoeken in de collectie.
- Een statement om papier toe te voegen aan de gevonden papierlade.

```
/**
 * Aanvullen van een papierlade met aantal bladen.
 *@param papiertype  Het papiertype van de bladen
 *@param aantal      Het aantal bladen waarmee je wenst aan te vullen
 */
public void vulPapierlade(String papiertype, int aantal)
{
    int index = 0;

    //Zoek in de collectie van papierlades naar de index
    //van een papierlade met het juiste papiertype.

    while(index < lades.size() &&
            ! lades.get(index).checkPapiertype(papiertype))
    {
        index++;
    }

    //Index gevonden?
    if(index < lades.size())
    {
        //--> Ja, vul papier aan met het gegeven aantal bladen.
        lades.get(index).vulPapierladeBij(aantal);
    }
}
```

0.5 Testen

Vergeet niet om steeds het resultaat te testen. Logische fouten of denkfouten worden tijdens het compileren niet gedecteerd door *BlueJ*. Bovendien, als je stap voor stap alles correct programmeert, werkt testen motiverend.

Objecten en klassen 1

Oefening 1.1 Bij de afbeelding van het spelletje FIFA 17 hebben we heel wat objecten benoemd. Kan je nog drie andere objecten benoemen?

Antwoord – Een aantal correcte antwoorden zijn:
- scheidsrechter,
- doel,
- recclameborden.

Tip: schrijf ze met kleine letter omdat het objecten zijn. Dit komt verderop aan bod.

Oefening 1.2 Bij de afbeelding van het spelletje FIFA 17 hebben we heel wat objecten benoemd. Kan je ook verschillende klassen opsommen waartoe deze objecten behoren?

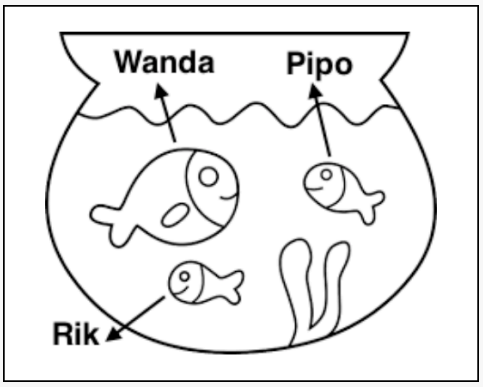
Antwoord – Een aantal correcte antwoorden zijn:
- Speler,
- Bal,
- Scheidsrechter.

Tip: Namen van klassen beginnen met een hoofdletter. Dit komt verderop nog aan bod.

Oefening 1.3 Kan je een situatie bedenken waar je meer klassen dan objecten kan opsommen? Waarom wel? Waarom niet?

Antwoord – Indien je een situatie kan bedenken waarin je meer klassen dan objecten kan opsommen, dan komt dit omdat er niet-gebruikte klassen geprogrammeerd werden. Stel je de klasse `Vis` voor bij het spel Mini Metro.

Oefening 1.4 – Aquarium Wanda, Rik en Pipo zijn drie vissen die gelukkig rondzwemmen in een aquariaum.



Figuur 1.1: DCliparting, Image #35455

Zijn onderstaande uitspraken waar of vals?

- Op de afbeelding staan drie objecten van de klasse `Vis`.
- Pipo is een instantie van de klasse `Aquarium`.
- Op de afbeelding staat juist één instantie van de klasse `Aquarium`.

(Naar een oefening van Goderik Lefebvre, Waregem)

Antwoord – De correcte antwoorden staan tussen de vragen.

- Op de afbeelding staan drie objecten van de klasse `Vis`.
  Deze uitspraak is waar: Wanda, Pipo en Rik zijn objecten van de klasse `Vis`.
- Pipo is een instantie van de klasse `Aquarium`.
  Deze uitspraak is vals. Pipo is een instantie van de klasse `Vis` *of* `ZoetWaterVis` *maar niet van* `Aquarium`.
- Op de afbeelding staat juist één instantie van de klasse `Aquarium`.
  Deze uitspraak is waar. We zien exact één afbeelding van een aquarium.

Oefening 1.5 Open je favoriete e-mailclient. Welke objecten zijn een instantie van welke klasse?

Antwoord – Mogelijke antwoorden zijn:
- Verschillende e-mailobjecten van de klasse `Mail`.
- Inbox, outbox, sent enz. zijn instanties van de klasse `Mailbox`.

Oefening 1.6 Maak een object aan met *BlueJ*-naam `mario`.

- Bestudeer de toestand van object `mario`. Welke waarden houdt het object bij?
- Laat het object tweemaal naar rechts bewegen, vervolgens éénmaal springen en ten slotte éénmaal naar links bewegen.
- Bestudeer de nieuwe toestand van het object. Doet het object wat je verwacht?

Antwoord – De correcte antwoorden staan tussen de vragen.

- Bestudeer de toestand van object `mario`. Welke waarden houdt het object bij?
  De x-coördinaat en y-coördinaat van het object op het scherm.
- Bestudeer de nieuwe toestand van het object. Doet het object wat je verwacht?
  Niet helemaal. Bij het springen blijft het object zweven.

Oefening 1.7 Stel dat men jou vraagt een voetbalspeler uit FIFA 17 te programmeren en je op het idee komt de klasse `VoetbalSpeler` te programmeren.

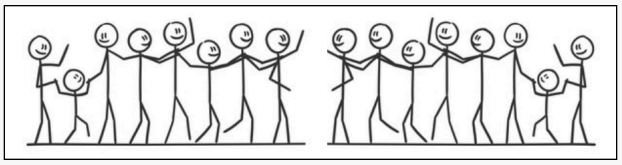
- Welke waarden moet de voetbalspeler kunnen bijhouden?
- Welke informatie moet je kunnen vragen aan een voetbalspeler?
- Wat moet de voetbalspeler allemaal kunnen doen?

Antwoord – Een aantal correcte antwoorden staan tussen de vragen.

- Welke waarden moet de voetbalspeler kunnen bijhouden?
  - *naam*,
  - *positie*,
  - *shirt-nummer*,
  - *plaats op het veld*.
- Welke informatie moet je kunnen vragen aan een voetbalspeler?
  - *naam*,
  - *de waarde van de speler uitgedrukt in euro*.
- Wat moet de voetbalspeler allemaal kunnen doen?
  - *Zicht verplaatsen op het veld*.
  - *De bal naar een medespeler passen*.

Oefening 1.8 – Polonaise In het grote feestwoordenboek staat bij polonaise de volgende definitie:

Een sliert feestvierende personen, elk met de handen op de schouder van de volgende persoon of met de handen zwaaiend, die luidruchtig door de kamer of zaal bewegen.



Figuur 1.2: Polonaise dans, de.fotolia.com

Schrap wat niet past in de uitspraken over bovenstaande afbeelding:

- We zien twee ***objecten / klassen*** van ***de klasse / het object*** `Polonaise`.
- Een instantie van ***de klasse / het object*** `Polonaise` bevat op haar beurt verschillende ***objecten / klassen*** van ***de klasse / het object*** `Feestneus`.
- In ***de klasse / het object*** `Polonaise` programmeert men het best de mogelijkheid om meerdere ***objecten / klassen*** van ***de klasse / het object*** `Feestneus` bij te houden.

(Naar een oefening van Goderik Lefebvre, Waregem)

Antwoord – De correcte uitspraken zijn:

- We zien twee ***objecten*** van ***de klasse*** `Polonaise`.
- Een instantie van ***de klasse*** `Polonaise` bevat op haar beurt verschillende ***objecten*** van ***de klasse*** `Feestneus`.
- In ***de klasse*** `Polonaise` programmeert men het best de mogelijkheid om meerdere ***objecten*** van ***de klasse*** `Feestneus` bij te houden.

Oefening 1.9 – Factuur De volgende afbeelding toont een factuur van een zaak gespecialiseerd in duurzame verzorgingsproducten.

aantal	omschrijving		per stuk	totaal
2x	Lanaform Etherische Olie	€	9,50 €	19,00
1x	Little Diva Handzeep	€	7,95 €	7,95
1x	Dermolin bodymilk	€	5,99 €	5,99
		subtotaal inclusief BTW	€	**32,94**
		totaal exclusief 19% BTW	€	27,68
		BTW 19%	€	5,26
		subtotaal	€	**32,94**
		totaal	€	**32,94**

Figuur 1.3: Twinkle Digital Commerce

Aangezien een factuur uit de printer van een computer rolt, kunnen we de factuur modelleren.

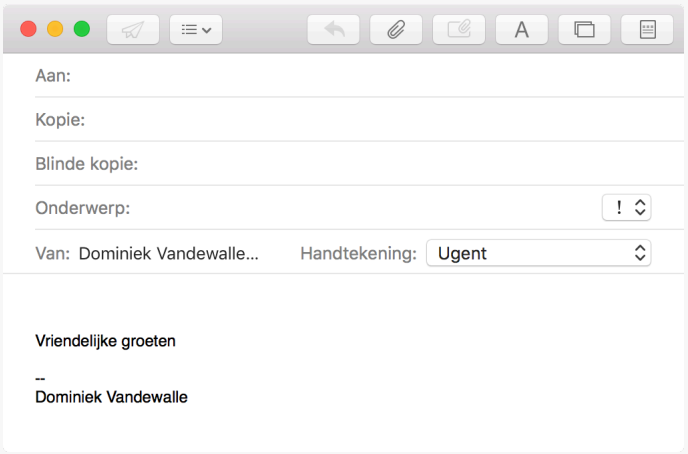
- Welke objecten vind je terug op de afbeelding?
- Van welke klassen zouden deze objecten instanties zijn?
- Wat houdt de toestand van elk van deze objecten bij?

Antwoord – De meest voorkomende fout is dat je objecten als aantal, omschrijving, enz. opsomt. Correct zijn:

- Drie objecten van de klasse `Product`. Een object van de klasse `Product` houdt behalve de omschrijving van het product ook de prijs per stuk bij.
- Drie objecten van de klasse `BesteldProduct`. Let op, dit is moeilijk want een object van `BesteldProduct` houdt behalve een instantie van de klasse Product, ook het aantal bestelde items bij. Hier kan je verwijzen dat zo een klasse in hoofdstuk 5 aan bod komt. Vermeld ook dat subtotaal een berekening is en dus niet hoeft bijgehouden te worden. Het is voldoende om aantal en prijs per stuk te weten. Prijs per stuk wordt bijgehouden door een instantie van de klasse `Product`.
- Een object van de klasse `Factuur` die drie objecten bijhoudt van de klasse `BesteldProduct`. Hier kan je verwijzen dat zo een klasse in hoofdstuk 6 aan bod komt.

Oefening 1.10 – E-mail client Stel dat je de klasse `Mail` moet programmeren.

- Welke waarden moet een e-mail kunnen bijhouden?
- Welke informatie moet je allemaal opgeven aan een e-mail?
- Welke informatie moet je kunnen opvragen van een e-mail?
- Wat moet de e-mail allemaal kunnen doen?



Figuur 1.4: Mail van Apple Inc.

Antwoord – Een aantal correcte antwoorden staan tussen de vragen.

- Welke waarden moet een e-mail kunnen bijhouden?
  onderwerp, geadresseerden, bericht.
- Welke informatie moet je allemaal opgeven aan een e-mail?
  onderwerp, geadresseerden, prioriteit, bijlage.
- Welke informatie moet je kunnen opvragen van een e-mail?
  onderwerp, bericht, bijlage.
- Wat moet de e-mail allemaal kunnen doen?
  Het onderwerp tonen, bijlagen aanbieden om op te slaan.

Oefening 1.11 – Supermario Open het project *SuperMario*. Dit project vind je in de map oefeningen bij hoofdstuk 1. Maak een object aan van deze klasse en bestudeer de werking en de toestand van het object.

- Wanneer wordt een object 'groot' en 'klein'?
- Kan SuperMario door de grond zakken? Of schuin omhoog springen?

Antwoord – Een aantal correcte antwoorden staan tussen de vragen.

- Wanneer wordt een object 'groot' en 'klein'?
  Een object wordt 'klein' wannneer je het commando `pijlOmLaagIndrukken()` *uitvoert. Een object wordt 'groot' wanneer je het commando* `pijlOmLaagLoslaten()` *uitvoert.*
- Kan SuperMario door de grond zakken? Of schuin omhoog springen?
  Door de grond zakken gaat niet. Schuin omhoog springen met een enkel commanda gaat evenmin.

Klassendefinitie 2

Oefening 2.1 Test of je in plaats van `public class Product` ook `class public Product` mag schrijven.

Antwoord – Neen je mag de sleutelwoorden niet wisselen van plaats.

Oefening 2.2 Open het *BlueJ*-project *OefeningenHoofdstuk2*. Je vindt dit project bij de oefenbestanden van hoofdstuk 2.

Schrijf de header van de volgende klassen:

- `VoetbalSpeler`,
- `MetroStel`.

Kies in *BlueJ* voor *Nieuwe Klasse...*, tik de naam van de klasse correct in. *BlueJ* gebruikt steeds een sjabloon waarvan je kan vertrekken. Dit is leuk, maar voor de oefeningen in dit hoofdstuk mag je alle *Java*-code eerst verwijderen.

Typ nu in elke klasse de header. Het inwendige van de klasse mag je leeg laten. Vergeet niet de accolades te typen! Om te testen klik je het best eerst op *Compileren*.

Antwoord – De headers van de klassen `VoetbalSpeler` en `MetroStel` programmeer je als volgt:

```
1  public class VoetbalSpeler
2  {
3  }
```

```
1  public class MetroStel
2  {
3  }
```

Oefening 2.3 Wat zijn de namen van de volgende velden?

- `private String ipAddress;`
- `private double lengte;`
- `private int aantal;`

Antwoord – De namen van de velden zijn:

- `ipAddress`,
- `lengte`,
- `aantal`.

Oefening 2.4 Wat is het type van de volgende velden?

- `private boolean isVrouw;`
- `private Student bieke;`
- `private char maat;`

Antwoord – Het type van de velden zijn:

- `boolean`,
- `Student`,
- `char`.

Oefening 2.5 In het project *OefeningenHoofdstuk2* schrijf je de header en de declaratie van de velden van de volgende klassen (die je het best eerst aanmaakt):

- `Leerling`,
- `Mail`,
- `SoftwarePakket`.

Antwoord – Header en velden van de klasse `Leerling`:

```
1    public class Leerling
2    {
3        private String naam;
4        private int leeftijd;
5        private char geslacht;
6    }
```

Antwoord – Header en velden van de klasse `Mail`:

```
1    public class Mail
2    {
3        private String aan;
4        private boolean isGelezen;
5        private String onderwerp;
6    }
```

Antwoord – Header en velden van de klasse `SoftwarePakket`:

```
public class SoftwarePakket
{
    private String naam;
    private double prijs;
    private double aantalMB;
    private int serienummer;
}
```

Oefening 2.6 `public Schaakstuk()` is de header van de constructor van een klasse.

Programmeer in het project *OefeningenHoofdstuk2* die je vindt bij de oefenbestanden van hoofdstuk 2, de header van de klasse en declareer minstens twee velden.

Antwoord – Een mogelijke oplossing is:

```
public class Schaakstuk
{
    private String schaakstuk;
    private boolean isZwart;
}
```

Oefening 2.7 In het project *OefeningenHoofdstuk2* schrijf je een constructor bij de klassendefinitie van de volgende klassen:

- `Leerling`,
- `Mail`,
- `SoftwarePakket`.

In de constructor geef je de velden van de klasse via een toekenningsstatement een gepaste waarde.

Antwoord – Constructor voor de klasse `Leerling`:

```
public Leerling()
{
    naam = "Ann-Sophie";
    leeftijd = 16;
    geslacht = 'V';
}
```

Antwoord – Constructor voor de klasse `Mail`:

```
public Mail()
{
    aan = "tim.cook@apple.com";
    isGelezen = false;
    onderwerp = "Update naar iOS 12";
}
```

Antwoord – Constructor voor de klasse `SoftwarePakket`:

```
    public SoftwarePakket()
    {
        naam = "BlueJ";
        prijs = 0.0;
        aantalMB = 191.0;
        serienummer = 437897;
    }
```

Oefening 2.8 Kan je aan de hand van de parameters van de constructor iets weten over het type van de velden van een klasse?

Antwoord – Indien de waarde van een parameter gebruikt wordt in een toekenningsstatement met een veld, dan is het type van de parameter een goede voorspeller van het type van een veld.

Oefening 2.9 Verander de code van de constructor van de klasse `Product`. Schrijf devolgende code:

```
naam = naam;
prijs = prijs;
aantal = aantal;
productKlasse = klasse;
bewaarInKoelcel = false;
```

Overal werd `this.` weggelaten. Maak een instantie van de klasse `Product` met behulp van de gewijzigde constructor. Inspecteer de velden van het object. Wat stel je vast?

Antwoord – De actuele waarden van de parameters worden doorgegeven aan de parameters zelf en niet aan de velden. De velden behouden dus hun defaultwaarde.

Oefening 2.10 Bij oefening 2.7 heb je in het project *OefeningenHoofdstuk2* een constructor geschreven die de velden van de volgende klassen een vaste waarde gaf.

- `Leerling`,
- `Mail`,
- `SoftwarePakket`.

Pas de constructor aan zodat de gebruiker van de klasse zelf kan kiezen welke waarde de velden krijgen. Misschien neem je best eerst een back-up van je project.

Antwoord – Constructor voor de klasse `Leerling`:

```
    public Leerling(String naam, int leeftijd, char geslacht)
    {
        this.naam = naam;
        this.leeftijd = leeftijd;
        this.geslacht = geslacht;
    }
```

Antwoord – Constructor voor de klasse `Mail`:

```
    public Mail(String aan, boolean isGelezen, String onderwerp)
    {
        this.aan = aan;
        this.isGelezen = isGelezen;
        this.onderwerp = onderwerp;
    }
```

Antwoord – Constructor voor de klasse `SoftwarePakket`:

```
    public SoftwarePakket(String naam, double prijs, double aantalMB, int
        serienummer)
    {
        this.naam = naam;
        this.prijs = prijs;
        this.aantalMB = aantalMB;
        this.serienummer = serienummer;
    }
```

Oefening 2.11 Verander in de klasse `Product` de naam van de accessormethode `getNaam()` in `getProductNaam()`. Compileer de klasse en test deze methode.

Is er een verband tussen de naam van een accessormethode en de veldnaam van de retourwaarde?

Antwoord – Neen, er is geen verband tussen de naam van een veld en de naam van een accessormethode. Al is het wel een goede attitude om de naam van het veld te vermelden in de accessormethode. Is de naam van het veld naam, gebruik dan `getNaam()`. Is de naam van het veld `productNaam` gebruik dan `getProductNaam()`

Oefening 2.12 Kan je een accessormethode schrijven zonder `return`-statement? Kan je een accessormethode schrijven zonder het retourtype te vermelden?

Antwoord – Neen. Wanneer je een retourtype verschillend van void vermeldt, dan moet de methode altijd een retourwaarde teruggeven.

Oefening 2.13 Schrijf in het project *OefeningenHoofdstuk2* voor elk veld van de volgende klassen een accessormethode.

- `Leerling`,
- `Mail`,
- `SoftwarePakket`.

Antwoord – Accessormethoden voor de klasse `Leerling`:

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

    public int getLeeftijd()
    {
        return leeftijd;
    }

    public char getGeslacht()
    {
        return geslacht;
    }
```

Antwoord – Accessormethoden voor de klasse `Mail`:

```
    public String getAan()
    {
        return aan;
    }

    public boolean getIsGelezen()
    {
        return gelezen;
    }

    public String getOnderwerp()
    {
        return onderwerp;
    }
```

Antwoord – Accessormethoden voor de klasse `SoftwarePakket`:

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

    public double getPrijs()
    {
        return prijs;
    }

    public double getAantalMB()
    {
        return aantalMB;
    }

    public int getSerienummer()
    {
        return serienummer;
    }
```

Oefening 2.14 Schrijf in het project *OefeningenHoofdstuk2* voor elk veld van de volgende klassen een mutatormethode.

- `Leerling`,
- `Mail`,
- `SoftwarePakket`.

Antwoord – Mutatormethoden voor de klasse `Leerling`:

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

    public void setLeeftijd(int leeftijd)
    {
        this.leeftijd = leeftijd;
    }

    public void setGeslacht(char geslacht)
    {
        this.geslacht = geslacht;
    }
```

Antwoord – Mutatormethoden voor de klasse `SoftwarePakket`:

```
    public void setAan(String aan)
    {
        this.aan = aan;
    }

    public void setIsGelezen(boolean isGelezen)
    {
        this.isGelezen = isGelezen;
    }

    public void setOnderwerp()
    {
        return onderwerp;
    }
```

Antwoord – Mutatormethoden voor de klasse `SoftwarePakket`:

```
    public void setNaam(String naam)
    {
        naam;
    }

    public void setPrijs(double prijs)
    {
        return prijs;
    }

    public void setAantalMB(double aantalMB)
    {
        return aantalMB;
    }

    public void setSerienummer(int serienummer)
    {
        return serienummer;
    }
```

Oefening 2.15 Maak een instantie van de klasse `Product` aan. Zorg dat je in het *Terminalvenster* het stukje code te zien krijgt dat hiervoor verantwoordelijk is. In de *Objectenbank* vraag je vervolgens de naam op van het product en verander je de prijs.

Antwoord – Deze oefening werd opgegeven als inleiding op een stukje leerstof die je terugvindt in het handboaek.

Oefening 2.16 Verwijder het object dat je aangemaakt hebt in oefening 2.15. Typ nu in het *evaluatievak* de volgende regel code en klik op [ENTER]:

```
Product raider = new Product("Twix", 2.49, 'A', 3);
```

Antwoord – Deze oefening werd opgegeven als inleiding op een stukje leerstof die je terugvindt in het handboaek.

Oefening 2.17 – Concepten In de volgende tabel vind je de omschrijving van concepten. Schrijf telkens het juiste concept naast de omschrijving.

Beschrijving	Concept
Wordt gebruikt om een object op de juiste manier in te stellen wanneer het wordt gemaakt.	
Definieert het deel van de broncode van waaruit de variabele kan worden benaderd.	
Leveren aanvullende informatie aan een methode.	
Retourneren informatie over de toestand van een object.	

Antwoord – De antwoorden zijn: constructor, bereik, parameters, accessormethoden.

Oefening 2.18 – Declaratie velden In de volgende tabel vind je de beschrijvingen van velden van klassen. Schrijf telkens de declaratie van het veld.

Beschrijving	Declaratie
Het veld `isbnNummer` houdt het ISBN-nummer bij van een object van de klasse `Boek`. bv: 978-90-816283-1-0	
Declareer een veld dat bijhoudt of de lichten aan een spoorwegoverweg rood of wit zijn.	
Het veld `wisselkoers` drukt de wisselkoers uit tussen de euro en de dollar.	
In het spelletje FIFA 17 wordt de positie van een speler voorgesteld door de letter. G = Goalkeeper, M = Midfielder enz.	
Tijdens de Tour de France zie je op het tv-scherm steeds het gehele aantal seconden die verstreken zijn nadat de eerste wielrenner de finish bereikt heeft.	

Antwoord – De antwoorden zijn:

- `private String isbnNummer;`
- `private boolean isRood;`
- `private double wisselkoers;`
- `private char positie;`
- `private int seconcen;`

Oefening 2.19 – Product Vul de klasse `Product` aan met de ontbrekende accessoren en mutatoren.

Antwoord – De ontbrekende accessormethoden zijn:

```
1     public double getPrijs()
2     {
3         return prijs;
4     }
5
6     public char getProductKlasse()
7     {
8         return productKlasse;
9     }
10
11    public boolean getBewaarInKoelcel()
12    {
13        return bewaarInKoelcel;
14    }
```

Antwoord – De ontbrekende mutatormethoden zijn:

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

    public void setAantal(int aantal)
    {
        this.aantal = aantal;
    }

    public void setProductKlasse(char productKlasse)
    {
        this.productKlasse = productKlasse;
    }
```

Oefening 2.20 – Bereik en levensduur Duid in de volgende klassendefintie het bereik aan van `bedragInAutomaat`, `muntstuk` en `hulp`.

```
public class GeldAutomaat
{

    private double bedragInAutomaat;

    public GeldAutomaat()
    {
        bedragInAutomaat = 0;
    }

    public double getBedragInAutomaat()
    {
        return bedragInAutomaat;
    }

    public void werpMuntstukInAutomaat(double muntstuk)
    {
        bedragInAutomaat = bedragInAutomaat + muntstuk;
    }

    public double resetAutomaat()
    {
        double hulp;
        hulp = bedragInAutomaat;
        bedragInAutomaat = 0;
        return hulp;
    }
}
```

Bespreek de levensduur van `bedragInAutomaat` en van `muntstuk`.

Antwoord – Goede antwoorden van de vragen over bereik zijn:

- Het bereik van `bedragInAutomaat` is de volledige klassendefinitie. Een veld kan je gebruiken in elke methode van de klassendefinitie.
- Het bereik van de parameter `muntstuk` is beperkt tot de methode waarvan `muntstuk` parameter is. In dit geval de methode `void werpMuntstukInAutomaat(double)`.
- Het bereik van de methodevariabele `hulp` is beperkt tot de methode `double resetAutomaat()`. Enkel in deze methode kan je de waarde van `hulp` opvragen of wijzigen.

Antwoord – Goede antwoorden van de vragen over levensduur zijn:

- De levensduur van `bedragInAutomaat` is gelijk aan de levensduur van een instantie van de klasse `Geldautomaat`. Zolang het object bestaat, zal `bedragInAutomaat` een waarde hebben.
- De levensduur van de parameter munstuk is beperkt tot de uitvoeringstijd van de methode `void werpMuntstukInAutomaat(double)`. Op een moderne computer is dat een fractie van een seconde.

Oefening 2.21 – VoorbeeldMethod Bij het aanmaken van een nieuwe klasse gebruikt *BlueJ* steeds hetzelfde sjabloon. In dat sjabloon wordt elke keer één methode gebruikt.

```
public int voorbeeldMethod(int y)
{
    // schrijf hier jouw code
    return x + y;
}
```

- Geef de naam van deze methode.
- Geef de naam en het gegevenstype van de parameter van deze methode.
- Wat is het retourtype van deze methode?
- Deze methode gebruikt een veld van de klasse. Geef de naam van dit veld.

Antwoord – De correcte antwoorden zijn:

- `voorbeeldMethod`,
- `int y`,
- `int`,
- `x`.

Oefening 2.22 – iTunes Mediaplayers zoals iTunes gebruiken een mappenstructuur om mediabestanden op te slaan. Een muziekbestand kan verschillende bestandsformaten zoals *mp4* of *aac* hebben. Een liedje heeft niet alleen een titel, een uitvoerder en een album, maar ook een duur.

- Schrijf de header van de klasse `MuziekNummer`.

  Antwoord – Een correct antwoord is:

  ```
  1 public class MuziekNummer
  2 {
  3 }
  ```

- Haal de velden uit deze omschrijving van de klasse en declareer ze in de klassendefinitie van de klasse `MuziekNummer`.

  Antwoord – Een correct antwoord is:

  ```
  1     private String titel;
  2     private String uitvoerder;
  3     private String album;
  4     private int duur;
  5     private String bestandsformaat;
  ```

- Programmeer een constructor voor de klasse `MuziekNummer` die aan alle velden een waarde geeft.

Antwoord – Een correct antwoord is:

```
    public MuziekNummer(String titel, String uitvoerder, String album,
        int duur, String bestandsformaat)
    {
        this.titel = titel;
        this.uitvoerder = uitvoerder;
        this.album = album;
        this.duur = duur;
        this.bestandsformaat = bestandsformaat;
    }
```

- Programmeer voor alle velden accessor- en mutatormethoden.

Antwoord – Een correct antwoord is:

```
    public String getTitel()
    {
        return titel;
    }

    public String getUitvoerder()
    {
        return uitvoerder;
    }

    public String getAlbum()
    {
        return album;
    }

    public int getDuur()
    {
        return duur;
    }

    public void setTitel(String titel)
    {
        this.titel = titel;
    }

    public void setUitvoerder(String uitvoerder)
    {
        this.uitvoerder = uitvoerder;
    }

    public void setAlbum(String album)
    {
        this.album = album;
    }

    public void setDuur(int duur)
    {
        this.duur = duur;
    }

    public void setBestandsformaat(String bestandsformaat)
    {
        this.bestandsformaat = bestandsformaat;
    }
```

Oefening 2.23 – Gamma Te veel reclame in de brievenbus? Willen we met z'n allen de papierberg verkleinen dan moeten we misschien denken aan een digitale kortingsbonnen.

Maak de klasse `KortingsBon` aan en schrijf code bij de volgende vragen:

- Voorzie in velden voor de korting, het aantal dagen dat de bon geldig is en of de korting al dan niet geldig is voor GAMMAplus-kaarthouders. Bovendien hou je het nummer bij de barcode bij in veld van het type `String`.
- Programmeer een constructor die elk veld een waarde geeft via een parameter.
- Programmeer voor alle velden een accessormethode.
- Wanneer de kortingsactie verlengd wordt, moet je dit kunnen veranderen. Schrijf een passende mutatormethode.

(Naar een oefening van Annick Renders, Veurne)

Antwoord – Een correcte klassendefinitie is:

```
public class Kortingsbon
{
    private double korting;
    private int aantalDagenGeldig;
    private boolean isGeldigVoorGammaPlus;
    private String barcode;

    public Kortingsbon(double korting, int aantalDagenGeldig, boolean
        isGeldigVoorGammaPlus, String barcode)
    {
        this.korting = korting;
        this.aantalDagenGeldig = aantalDagenGeldig;
        this.isGeldigVoorGammaPlus = isGeldigVoorGammaPlus;
        this.barcode = barcode;
    }

    public double getKorting()
    {
        return korting;
    }

    public int getAantalDagenGeldig()
    {
        return aantalDagenGeldig;
    }

    public boolean getIsGeldigVoorGammaPlus()
    {
        return isGeldigVoorGammaPlus;
    }

    public String getBarcode()
    {
        return barcode;
    }

    public void setAantalDagenGeldig(int aantalDagenGeldig)
    {
        this.aantalDagenGeldig = aantalDagenGeldig;
    }
}
```

Oefening 2.24 – Vliegtuigtickets Wanneer je tijdens het online boeken van een vliegtuigticket alle vluchtgegevens invult, wordt op de achtergrond een object aangemaakt die deze gegevens verzamelt. Achteraf kan je dan jouw ticket printen en meebrengen naar de luchthaven. Elk vliegtuigticket heeft een uniek nummer, gecodeerd in de streepjescode. Je kiest achtereenvolgens het vluchtnummer, bijvoorbeeld *WAL 833 451*, je plaats in het vliegtuig, *23B*, en of je al dan niet first class wil vliegen. Uiteraard is je naam ook verplicht. Het nummer van de gate wordt achteraf toegevoegd.

- Schrijf de header van de klasse `VliegtuigTicket`.

  Antwoord – Een correct antwoord is:

```
public class VliegtuigTicket
{
}
```

- Haal uit de omschrijving van het vliegtuigticket de nodige velden.

  Antwoord – Een correct antwoord is:

```
    private int volgnummer;
    private String vluchtnummer;
    private String stoelnummer;
    private boolean inEersteKlasse;
    private String naam;
    private int gate;
```

- De constructor van een instantie van de klasse `VliegtuigTicket` vraagt alle waarden die je aan de velden geeft op, behalve het nummer van de gate en de klasse waarin je vliegt. Deze velden geef je expleciet een defaultwaarde.

  Antwoord – Een correct antwoord is:

```
    public VliegtuigTicket(int volgnummer, String vluchtnummer, String
        stoelnummer, boolean inEersteKlasse, String naam)
    {
        this.volgnummer = volgnummer;
        this.vluchtnummer = vluchtnummer;
        this.stoelnummer = stoelnummer;
        this.inEersteKlasse = inEersteKlasse;
        this.naam = naam;
        gate = 0;
    }
```

- Programmeer in de klasse `VliegtuigTicket` accessoren voor alle velden.

Antwoord – Een correct antwoord is:

```
    public int getVolgnummer()
    {
        return volgnummer;
    }
    public String getVluchtnummer()
    {
        return vluchtnummer;
    }
    public String getStoelnummer()
    {
        return stoelnummer;
    }
    public boolean getInEersteKlase()
    {
        return inEersteKlasse;
    }
    public String getNaam()
    {
        return naam;
    }
    public int getGate()
    {
        return gate;
    }
```

- Programmeer voor alle velden de passende mutatoren.

Antwoord – Een correct antwoord is:

```
    public void setVolgnummer(int volgnummer)
    {
        this.volgnummer = volgnummer;
    }
    public void setVluchtnummer(String vluchtnummer)
    {
        this.vluchtnummer = vluchtnummer;
    }
    public void setStoelnummer(String stoelnummer)
    {
        this.stoelnummer = stoelnummer;
    }
    public void setInEersteKlase(boolean inEersteKlasse)
    {
        this.inEersteKlasse = inEersteKlasse;
    }
    public void setNaam(String naam)
    {
        this.naam = naam;
    }
    public void setGate(int gate)
    {
        this.gate = gate;
    }
```

Oefening 2.25 – Treinstel Schrijf de klasse `Treinstel`. Deze klasse beschrijft een treinstel voor passagiers.

Bedenk zelf wat de velden kunnen zijn van een instantie van de klasse `Treinstel`. Het zijn zaken die eigen zijn aan één specifiek treinstel. Met de constructor geef je een waarde aan alle velden. Voorzie alleen voor het veld `huidigeBezetting` een mutatormethode. Programmeer voor alle velden accessormethoden.

Voorzie de klasse van de nodige commentaar zodat je een API-pagina kan genereren voor de gebruiker van je klasse.

Antwoord – Een correcte klassendefinitie van de klasse `Treinstel` is:

```
/**
* class Treinstel - Deze klasse beschrijft een Treinstel
*
* @author Dominiek Vandewalle
* @version 2017-12-20
*/
public class Treinstel
{
    private int aantalPlaatsen;//aantal zitplaatsen in het treinstel
    private boolean isEersteKlasse;//true = 1e klasse, false = 2e klasse

    /**
    * Constructor voor objecten van de klasse Treinstel
    * @param   aantalPlaatsen  Het aantal zitplaatsen in het treinstel.
    * @param   isEersteKlasse  True indien het treinstel een 1e klasse stel is
        .
    */
    public Treinstel(int aantalPlaatsen, boolean isEersteKlasse)
    {
        this.aantalPlaatsen = aantalPlaatsen;
        this.isEersteKlasse = isEersteKlasse;
    }

    /**
    * Geeft het aantal plaatsen in het treinstel terug.
    * @return  Het aantal plaatsen in het treinstel.
    */
    public int getAantalPlaatsen()
    {
        return aantalPlaatsen;
    }

    /**
    * Aangeven of het treinstel van eerste klasse of tweede klasse is.
    * @param   isEersteKlasse  True indien het treinstel van eerste klasse is.
    */
    public void setIsEersteKlasse(boolean isEersteKlasse)
    {
        this.isEersteKlasse = isEersteKlasse;
    }
}
```

Expressies en statements 3

Oefening 3.1 – Rekenmachine Het project *Rekenmachine* vind je bij de oefenbestanden van hoofdstuk 3. Het project bevat alleen de klasse Rekenmachine. Programmeer de volgende methoden:

- `int modulo(int, int)` geeft de rest na deling van de eerste parameter door de tweede parameter terug.

  Antwoord – Een correct antwoord is:

```
1     public int modulo(int getal1, int getal2)
2     {
3         return getal1 % getal2;
4     }
```

- `double gemiddelde(int, int)` geeft het gemiddelde van twee gehele getallen terug.

  Antwoord – Een correct antwoord is:

```
1     public double gemiddelde(int getal1, int getal2)
2     {
3         return (getal1 + getal2) / 2.0;
4     }
```

- `double korting(double, double)` berekent de totale prijs inclusief een korting. De eerste parameter is de prijs zonder korting, de tweede parameter het kortingspercentage. Het kortingspercentage is een getal tussen 0 en 1.

  Antwoord – Een correct antwoord is:

```
1     public double korting(double prijs, double kortingspercentage)
2     {
3         return prijs * (1 - kortingspercentage);
4     }
```

- `double discriminant(double a, double b, double c)` berekent de discriminant van een tweedegraadsvergelijking $ax^2 + bx + c = 0$. De discriminant van een tweedegraadsvergelijking bereken je als volgt: $D = b^2 - 4ac$.

Antwoord – Een correct antwoord is:

```
    public double discriminant(double a, double b, double c)
    {
        return (b * b) - (4 * a * c);
    }
```

Oefening 3.2 – Schoolsecretariaat Het project *SchoolSecretariaat* vind je bij de oefenbestanden van hoofdstuk 3. Het project bevat alleen de klasse `SchoolSecretariaat`. Programmeer zelf de volgende methoden:

- `String geboortedatum(int, int, int)` plaatst tussen de waarden voor dag, maand en jaar, meegegeven met de parameters, een slash of '/'.

Antwoord – Een correct antwoord is:

```
    public String modulo(int dag, int maand, int jaar)
    {
        return dag + "/" + maand + "/" + jaar;
    }
```

- `String klas (int jaar, String richting)` geeft de klas van een leerling terug.

Antwoord – Een correct antwoord is:

```
    public String klas(int jaar, String richting)
    {
        return jaar + richting;
    }
```

- `String mededeling(String, String, String)` genereert een mededeling die op de schoolrekening staat. Een voorbeeld van een mededeling is:
+++500/1502/50144+++. De actuele parameters van deze methode krijgen als waarde de stukjes getallen in de mededeling maar dan in `String`-formaat.

Antwoord – Een correct antwoord is:

```
    public String klas(String deel1, String deel2, String deel3)
    {
        return "+++" + deel1 + "/" + deel2 + "/" + deel3 + "+++";
    }
```

Oefening 3.3 – DoeDeTest Het project *DoeDeTest* vind je bij de oefenbestanden van hoofdstuk 3. Programmeer de volgende methoden in de klasse `DoeDeTest`:

- `boolean isMeerWaard(double, double)` controleert of de transferwaarde van de eerste speler groter is dan de transferwaarde van de tweede speler.

  Antwoord – Een correct antwoord is:

```
1    public boolean isMeerWaard(double waardeS1,double waardeS2)
2    {
3        return waardeS1 > waardeS2;
4    }
```

- `boolean isZelfdeLeerling(int,int)` controleert of de leerling-ID's van twee leerlingen in de administratieve databank van de school gelijk zijn. Wanneer ze gelijk zijn, laat je `true` teruggeven. In het andere geval laat je `false` teruggeven.

  Antwoord – Een correct antwoord is:

```
1    public boolean isZelfdeLeerling(int leerling1, int leerling2)
2    {
3        return leerling1 == leerling2;
4    }
```

- `boolean isAndereCode(int, int)` controleert of twee codes verschillend zijn. Je laat `true` teruggeven indien dit zo is, anders `false`.

  Antwoord – Een correct antwoord is:

```
1    public boolean isAndereCode(int code1, int code2)
2    {
3        return code1 != code2;
4    }
```

Oefening 3.4 – DoeDeTest Het project *DoeDeTest* vind je bij de oefenbestanden van hoofdstuk 3. Programmeer zelf de volgende methoden in de klasse `DoeDeTest`:

- `boolean isEenWinnaar(int)` krijgt als parameter het aantal ogen van een dobbelsteen mee. Alleen wanneer er geen 3 of 6 gegooid wordt, win je een prijs. In dat geval laat je de methode `true` teruggeven, anders `false`.

  Antwoord – Een correct antwoord is:

```
1    public boolean isEenWinnaar(int ogen)
2    {
3        return ! (ogen == 3 || ogen == 6);
4    }
```

- boolean isVIP(char,int) controleert of het lidnummer begint met een 'P' of een 'Y' (eerste parameter) en het lid ingeschreven werd voor het jaar 2000 (tweede parameter). Indien aan deze twee voorwaarden voldaan is, laat je true teruggeven, anders false.

  Antwoord – Een correct antwoord is:

```
1    public boolean isVIP(char prefix, int nummer)
2    {
3        return (prefix == 'P' || prefix == 'Y') && nummer < 2000;
4    }
```

- boolean isSomGroterDanNul(int, int) controleert of de som van twee getallen groter is dan nul. Je laat true teruggeven indien dit zo is, anders false.

  Antwoord – Een correct antwoord is:

```
1    public boolean isSomGroterDanNul(double getal1, double getal2)
2    {
3        return (getal1 + getal2) > 0;
4    }
```

Oefening 3.5 De klasse GeldAutomaatAdvanced vind je in het project *GeldAutomaatAdvanced* bij de bronprojecten van hoofdstuk 3. De klasse bezit behalve twee velden bedragInAutomaat en aantalMuntstukkenInAutomaat ook de volgende methode:

```
public double resetAutomaat()
{
   double hulp;
   hulp = bedragInAutomaat;
   bedragInAutomaat = 0;
   aantalMuntstukkenInAutomaat = 0;
   return hulp;
}
```

Verklaar wat deze methode precies doet. Je gebruikt hierbij het best een schematische voorstelling van het geheugen gebruikt door een instantie van de klasse GeldautomaatAdvanced.

Antwoord – In deze methode wordt de methodevariabele hulp gedeclareerd. De variabele hulp krijgt op regel 2 de waarde van het bedrag dat reeds in de automaat geworpen werd. Vervolgens worden de velden bedragInAutomaat en aantalMuntstukken op 0 gezet. De laatste regel geeft het oorsponkelijk bedrag dat in de automaat zat en tijdelijk in hulp werd opgeslagen, terug.

Oefening 3.6 – Borddienst De methode int bordDienst(int) berekent voor elke week van het schooljaar het volgnummer van de leerling met borddienst. De leerling met borddienst moet de hele week na elke les het bord afvegen.

```
public int bordDienst(int weeknummer)
{
    return (weeknummer % 11) + 11;
}
```

Laten we veronderstellen dat er 24 leerlingen in de klas zitten.

- Wat betekent de %-operator?
- Wie heeft in week 42 borddienst?
- Welke leerlingen hebben nooit borddienst?

(Naar een oefening van Goderik Lefebvre, Waregem)

Antwoord – De antwoorden vind je tussen de vragen:

- Wat betekent de %-operator?
  Dit is de modulo operator. De modulo-operator gebruik je om de rest na een deling te berekenen.
- Wie heeft in week 42 borddienst?
  De leerling met klasnummer 20*, want* $42\%11 = 9$ *en* $9 + 11 = 20$
- Welke leerlingen hebben nooit borddienst?
  De leerlingen met klasnummer 22*,* 23 *en* 24*. weeknummer % 11 kan hoogstens* 10 *zijn. En* $10 + 11 = 21$*.*
  De leerling met klasnummer 21 *is de leerling met het hoogste klasnummer die borddienst kan hebben.*

Oefening 3.7 Omcirkel het juiste antwoord.

- Het resultaat van `!('q' == 'a')` is
  a) `true` b) `false`
- Het resultaat van `!(4 == 5) || (4 > -4)` is
  a) `true` b) `false`
- Het resultaat van `10 - 6 % 2` is
  a) `1` b) `6` c) `8` d) `10`
- Het resultaat van: `"20"+ '3'` is
  a) `"23"` b) `'23'` c) `"203"` d) `203`
- ```
  int a = 5;
  int b = 7;
  ```
  Het resultaat van `(a != b) && !(2 * a < b)` is
  a) `true` b) `false`

(Naar een oefening van Marianne Claeys, Tielt)

Antwoord – De correcte antwoorden zijn: a, a, d, c, a.

Oefening 3.8 – Visitekaartje Maak in het project *HerhalingsOefeningenHoofdstuk3* de klasse `Visitekaartje` aan met:
  ```

- Velden: voornaam, familienaam, werkgever en functie.

Antwoord – Een correct antwoord is:

```
public class Visitekaartje
{
    private String voornaam;
    private String familienaam;
    private String werkgever;
    private String functie;
}
```

- Constructor: vraagt de voornaam en familienaam.

Antwoord – Een correct antwoord is:

```
    public Visitekaartje(String voornaam, String familienaam)
    {
        this.voornaam = voornaam;
        this.familienaam = familienaam;
        werkgever =  "";
        functie = "";
    }
```

- Mutatoren: voor de velden werkgever en functie.

Antwoord – Een correct antwoord is:

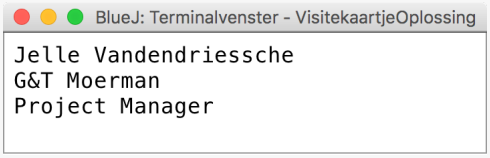
```
    public void setWerkgever(String werkgever)
    {
        this.werkgever = werkgever;
    }
    public void setFunctie(String functie)
    {
        this.functie = functie;
    }
```

- Accessoren: voor alle velden.

Antwoord – Een correct antwoord is:

```
    public String getVoornaam()
    {
        return voornaam;
    }
    public String getFamilienaam()
    {
        return familienaam;
    }
    public String getWerkgever()
    {
        return werkgever;
    }
    public String getFunctie()
    {
        return functie;
    }
```

- Methode: `void printVisitekaartje()` geeft het volgende *Terminalvenster*:



Antwoord – Een correct antwoord is:

```
    public void printVisitekaartje()
    {
        String info = "";

        info = voornaam + " " + familienaam + "\n";
        info = info + functie + "\n";
        info += werkgever;

        System.out.println(info);
    }
```

Oefening 3.9 – Cowboy Maak in het project *HerhalingsOefeningenHoofdstuk3* de klasse `Cowboy` aan. Deze klasse is een eerste ontwerp voor een spelletje.

Een instantie van de klasse `Cowboy` krijgt bij initialisatie drie levens mee. Telkens wanneer de cowboy geraakt wordt door een kogel, vermindert zijn aantal levens met één. Wanneer de cowboy een hartje opraapt van het spelveld, krijgt het object een extra leven.

- Programmeer de velden `naam` en `aantalLevens`.

Antwoord – Een correct antwoord is:

```
public class Cowboy
{
    private String naam;
    private int aantalLevens;
}
```

- Via een parameter geef je aan de constructor de naam van de cowboy mee. Het andere veld wordt in de constructor geïnitialiseerd.

Antwoord – Een correct antwoord is:

```
    public Cowboy(String naam)
    {
        this.naam = naam;
        aantalLevens = 3;
    }
```

- Programmeer accessormethoden voor alle velden.

Antwoord – Een correct antwoord is:

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

    public int getAantalLevens()
    {
        return aantalLevens;
    }
```

- `void schietOpMij()` kan je gebruiken om op de cowboy te schieten.

Antwoord – Een correct antwoord is:

```
    public void schietOpMij()
    {
        aantalLevens--;
    }
```

- `void hartjeOprapen()` gebruik je om een leven toe te voegen.

Antwoord – Een correct antwoord is:

```
    public void HartjeOprapen()
    {
        aantalLevens++;
    }
```

- `boolean isZombie()` test of het aantal levens kleiner of gelijk aan nul is. Indien dit het geval is, laat je `true` teruggeven, anders `false`.

Antwoord – Een correct antwoord is:

```
    public boolean isZombie()
    {
        return aantalLevens <= 0;
    }
```

Oefening 3.10 – Auto Programmeer in het project *HerhalingsOefeningenHoofdstuk3* de klasse `Auto`. Elke auto heeft een chassisnummer, bijvoorbeeld *WVWZZZ9NZ3W0000001*. Na assemblage in de fabriek is de snelheid 0,0 km/h. Wanneer de auto rijdt, kunnen we de snelheid van de auto aanpassen met de methoden `void versnel()` en `void vertraag()`. Beide methoden veranderen de snelheid met 1 km/h. Afhankelijk van het land waar de auto verkocht wordt, kunnen we de snelheid van de auto opvragen met de methode `double snelheidKilometerH()` of `double snelheidMilesH()`. De auto verbruikt het minste brandstof wanneer de snelheid tussen 70 km/h en 90 km/h ligt. De methode `boolean isEconomisch()` test of de auto met een economisch voordelige snelheid rijdt. De methode `boolean isInBeweging()` test of de auto in beweging is (bedenk twee mogelijkheden om dit te programmeren).

Programmeer aan de hand van deze analyse de klasse `Auto`. Houd rekening met het feit dat de klasse maar twee velden heeft: `chassisnummer` en `snelheid`.

Antwoord – Een correct antwoord is:

```java
public class Auto
{
    private String chassisnummer;
    private int snelheid;

    public Auto(String chassisnummer)
    {
        this.chassisnummer = chassisnummer;
        snelheid = 0;
    }

    public void versnel()
    {
        snelheid++;
    }

    public void vertraag()
    {
        snelheid--;
    }

    public int snelheidKilometerH()
    {
        return snelheid;
    }

    public double snelheidMilesH()
    {
        return snelheid * 0.6214;
    }

    public boolean isEconomisch()
    {
        return snelheid >= 70 && snelheid <= 90;
    }

    public boolean isInBeweging()
    {
        return snelheid != 0;//return snelheidKilometerH() != 0;
    }

    public void print()
    {
        String info = "";

        info = "chassisnummer: " + chassisnummer + "\n";
        info += "snelheid: " + snelheidKilometerH();

        System.out.println(info);
    }
}
```

Oefening 3.11 – DNA Maak in het project *HerhalingsOefeningenHoofdstuk3* de klasse `DNAStreng`.

Een DNA-streng is opgebouwd uit een lange reeks van vier verschillende basen: guanine, cytosine, adenine en thymine, afgekort met respectievelijk G, C, A en T. Een voorbeeld van een DNA-streng is *GGCTAGATCTG*. Bij het automatiseren van de detectie van DNA-strengen komt een stukje software kijken die het mogelijk maakt een DNA-streng bij te houden, aan te vullen en weer te geven.

Antwoord – Header, velden, constructor, accessor- en mutatormethode van de klasse `DNAStreng`:

```
public class DNAStreng
{
    private String streng;

    public DNAStreng()
    {
        streng = "";
    }

    public String getStreng()
    {
        return streng;
    }

    public void setStreng(String streng)
    {
        this.streng = streng;
    }
}
```

De klasse `DNAStreng` bevat slechts één veld `streng` en voorziet naast accessor en mutator voor het veld `streng` in de volgende methoden:

- `void voegBaseAchteraanToe(char)`
  Voegt een base (G, C, A of T) achter aan de DNA-streng toe.

  Antwoord – Een correct antwoord is:

  ```
      public void voegBaseAchteraanToe(char base)
      {
          streng = streng + base;
      }
  ```

- `void voegBaseVooraanToe(char)`
  Voegt een base (G, C, A of T) aan het begin van de DNA-streng toe.

  Antwoord – Een correct antwoord is:

  ```
      public void voegBaseVooraanToe(char base)
      {
          streng = base + streng;
      }
  ```

- `void printDNA()`
  Toont de DNA-streng in het *Terminalvenster*.

  Antwoord – Een correct antwoord is:

```
    public void printDNAStreng()
    {
        System.out.println(streng);
    }
```

Oefening 3.12 – Marktkramer Maak in het project *HerhalingsOefeningenHoofdstuk3* de klasse `Marktkramer`.

Antwoord – Een correct antwoord is:

```
public class Marktkramer
{
    //constructor hoeft niet
    public Marktkramer()
    {
    }
}
```

Programmeer in `Marktkramer` de volgende methoden:

- `double totalePrijs(int, double)`
  Berekent de totale prijs gegeven het aantal items dat je van een product wenst te kopen en de stukprijs van het product.

  Antwoord – Een correct antwoord is:

```
    public double totalePrijs(int aantalItems, double prijsPerStuk)
    {
        return aantalItems * prijsPerStuk;
    }
```

- `int totaalAantalAppels(int)`
  Berekent eerst hoeveel appels je gratis krijgt bij aankoop van een grote hoeveelheid. Per tien appels krijg je één appel gratis. Om de smaak te pakken te krijgen, krijg je onmiddellijk bij aankoop van de eerste appel ook al één gratis exemplaar. Het aantal appels dat je wenst te kopen geef je mee als parameter. De methode geeft de som van de gratis appels en de appels die je wenst te kopen, terug. Je mag ervan uitgaan dat je deze methode alleen gebruikt wanneer minstens één appel gekocht wordt.

  Antwoord – Een correct antwoord is:

```
    public int totaalAantalAppels(int aantalAppels)
    {
        return 1 + aantalAppels + (aantalAppels / 10);
    }
```

- `double zonderBTW(double)`
  Vermindert de prijs die je als parameter meegeeft met 6 procent. Het resultaat geef je terug.

Antwoord – Een correct antwoord is:

```
    public double zwart(double prijs)
    {
        return prijs * 0.94;
    }
```

Selectie 4

Oefening 4.1 – Ridder In de methoden `void raapSchildOp()` en `void doorZwaardGeraakt()` wordt steeds gecontroleerd of de ridder nog in leven is. Verander de voorwaarde in het `if`-statement door gebruik te maken van de private methode `boolean isDood()` in plaats van te testen op `aantalLevens`.

Antwoord – Een correct antwoord is:

```
1   public void doorZwaardGeraakt()
2   {
3       if(! heeftSchild && ! isDood())
4       {
5           aantalLevens--;
6       }
7
8       if(heeftSchild)
9       {
10          heeftSchild = false;
11      }
12  }
13
14  public void raapSchildOp()
15  {
16      if(! isDood())
17      {
18          heeftSchild = true;
19      }
20  }
```

Oefening 4.2 – Postweegschaal Open bij de oefeningenbestanden van hoofdstuk 4 het project *Post*. Je werkt in de klasse `PostWeegschaal`. In deze oefening automatiseer je het prijzen van een postpakketje. Bij het versturen van een postpakketje rekent de post de volgende bedragen aan:

- Voor een pakje van twee kilogram of minder betaalt men € 18.
- Voor elke kilogram meer wordt er een extra bedrag van € 5 aangerekend. Ook voor de laatste onvolledige kilogram moet je de volledige € 5 aanrekenen. Vandaar dat het type van de parameter `int` is.

In de klasse `Postweegschaal` programmeer je de methode `int geefPrijs(int)`. Gebruik de voorgaande analyse bij het programmeren van de body van deze methode.

Antwoord – Een correct antwoord is:

```
    public int geefPrijs(int gewicht)
    {
        int prijs;

        if (gewicht <= 2)
        {
            prijs = 18;
        }
        else
        {
            prijs = 18 + (gewicht - 2) * 5;
        }

        return prijs;
    }

    //of korter
    public int geefPrijs(int gewicht)
    {
        int prijs = 18;

        if (gewicht > 2)
        {
            prijs += (gewicht - 2) * 5;
        }

        return prijs;
    }
```

Oefening 4.3 – Lijnwinkel Open bij de oefeningenbestanden van hoofdstuk 4 het project *Autobus*. Je werkt in de klasse `Lijnwinkel`.

Een busmaatschappij hanteert de volgende prijzen bij een toeristische rondrit:

- Als je minder dan 5 kaartjes koopt, kosten ze € 5 per stuk.
- Koop je minstens 5 kaartjes maar minder dan 10 kaartjes, dan kosten ze € 4 per stuk.
- Anders kosten ze € 3 per stuk.

In de klasse `Lijnwinkel` programmeer je de methode `int geefKostprijs(int)`.

Antwoord – Een correct antwoord is:

```
    public int geefKostprijs(int aantal)
    {
        int prijs = 0;

        if (aantal < 5)
        {
            prijs = aantal * 5;
        }
        else if (aantal < 10)
        {
            prijs = aantal * 4;
        }
        else
        {
            prijs = aantal * 3;
        }

        return prijs;
    }
```

Oefening 4.4 – Pensioenregeling Open bij de oefeningenbestanden van hoofdstuk 4 het project *Pensioen*. Je werkt in de klasse `PensioenRegeling`. `public double zwart(double prijs) return prijs * 0.94;` Een land treft de volgende pensioenregeling:

- Een man krijgt € 250 per week als hij ouder is dan 65 jaar, en nog eens € 20 extra als hij ouder is dan 70 jaar.
- Een vrouw krijgt € 220 per week als ze ouder is dan 60 jaar, en nog eens € 50 extra als ze ouder is dan 65 jaar.

Programmeer `int geefPensioen(int, boolean)` in de klasse `Pensioenregeling`.

Antwoord – Een correct antwoord is:

```
    public int geefPensioen(int leeftijd, boolean man)
    {
        int pensioen = 0;

        if(man)
        {
            if(leeftijd > 65)
            {
                pensioen = 50;

                if (leeftijd > 70)
                {
                    pensioen += 20;
                }
            }
        }
        else
        {
            if(leeftijd > 60)
            {
                pensioen = 45;

                if(leeftijd > 65)
                {
                    pensioen += 25;
                }
            }
        }

        return pensioen;
    }
```

Oefening 4.5 – Ridder In de klassendefinitie van de klasse `Ridder` voer je de volgende wijzigingen uit:

- Verwijder in de methode `void herleef()` de statements `break;`. Wat zijn de gevolgen voor instanties van de klasse `Ridder`?

  Antwoord – Zonder `break;` wordt elk statement uitgevoerd. Het laatste statement geeft de waarde 1 aan het veld `aantalLevens`.

- Wanneer de ridder jouw naam draagt, krijg je 5 leventjes telkens wanneer je de ridder laat herleven.
- Wanneer de ridder de naam draagt van een van twee willekeurige leerkrachten van jou, dan krijgt de ridder 10 leventjes extra.
- Wanneer de ridder niet dood was, verliest de ridder al zijn leventjes. Eigen schuld!

Antwoord – Een correct antwoord is:

```
    public void herleef()
    {
        if(isDood())
        {
            switch (naam)
            {
                case "Richard the Lionheart":
                case "Joan of Arc":
                    aantalLevens = 3;
                    break;
                case "Blackadder":
                    aantalLevens = 2;
                    break;
                case "Dominiek Vandewalle"
                    aantalLevens = 5;
                    break;
                case "Marianne Claeys":
                case "Marijke Lisabeth":
                    aantalLevens = 10;
                    break;
                default:
                aantalLevens = 1;
                break;
            }
        }
        else
        {
            aantalLevens = 0;
        }
    }
```

Oefening 4.6 – Scouts Open bij de oefeningenbestanden van hoofdstuk 4 het project *Jeugdbeweging*. Je werkt in de klasse `Scouts`.

In de Vlaamse scoutsbeweging worden de leden ingedeeld in leefdtijdscategorieën. De zeescouts hebben voor de jongste leden een afwijkende naam. Hieronder de tabel met leeftijdscategorieën:

Categorie	Leeftijd
bevers	5 tot 7 jaar
zeehonden	5 tot 7 jaar
welpen	8 tot 10 jaar
aspiranten	11 tot 13 jaar
juniors	14 tot 15 jaar
seniors	16 tot 17 jaar
stam	vanaf 18 jaar

In de klasse `Scouts` schrijven we de methode `void leeftijdsCategorie(String)`. De methode print in het terminalvenster de leeftijdsgrenzen van een categorie. De categorie (bv. welpen) is de parameter van deze methode. Je gebruikt het `switch`-statement.

Antwoord – Een correct antwoord is:

```
    public void leeftijdsCategorie(String categorie)
    {
        String leeftijd;

        switch(categorie)
        {
            case "bevers":
            case "zeehonden":
                leeftijd = "5 tot 7 jaar";
                break;
            case "welpen":
                leeftijd = "8 tot 10 jaar";
                break;
            case "aspiranten":
                leeftijd = "11 tot 13 jaar";
                break;
            case "juniors":
                leeftijd = "14 tot 15 jaar";
                break;
            case "seniors":
                leeftijd = "16 tot 17 jaar";
                break;
            case "stam":
                leeftijd = "18+ jaar";
                break;
            default:
                leeftijd = "onbekende categorie";
                break;
        }

        System.out.println(leeftijd);
    }
```

Oefening 4.7 – Blad-steen-schaar Open bij de oefeningenbestanden van hoofdstuk 4 het project *Spelletje*. In de klasse `BladSteenSchaar` simuleren we het spelletje blad-steen-schaar. Bij dit spelletje beelden twee spelers, verborgen achter hun rug, een blad (vlakke hand), steen (gesloten vuist) of schaar (V-teken) uit. Gelijktijdig tonen ze het uitgebeelde voorwerp aan elkaar. De winnaar wordt als volgt bepaald:

- Blad wint van steen.
- Steen wint van schaar.
- Schaar wint van blad.
- Bij gelijke voorwerpen wordt opnieuw gespeeld.

In de methode `String wieWint(String,String)` programmeren we dit spelletje. De eerste parameter is het voorwerp van de eerste speler, de tweede parameter het voorwerp van de tweede speler. De mogelijke retourwaarden zijn:

- `speler 1`
- `speler 2`
- `speel opnieuw`

Antwoord – Een correct antwoord is:

```
    public String wieWint(String voorwerp1, String voorwerp2)
    {
        String winnaar;

        if(voorwerp1.equals(voorwerp2))
        {
            winnaar = "Speel opnieuw";
        }
        else if(voorwerp1.equals("blad") && voorwerp2.equals("steen"))
        {
            winnaar = "speler1";
        }
        else if (voorwerp1.equals("steen") && voorwerp2.equals("schaar"))
        {
            winnaar = "speler1";
        }
        else if (voorwerp1.equals("schaar") && voorwerp2.equals("steen"))
        {
            winnaar = "speler1";
        }
        else
        {
            winnaar = "speler2";
        }

        return winnaar;
    }
```

Oefening 4.8 Zoek in de *Java-API* naar de methode `substring(int)`. Wat zijn de verschillen met `substring(int, int)`?

Antwoord – De methode `substring(int)` geeft het deel van een `String` terug te beginnen vanaf de index meegegeven door de parameter tot het einde van de `String`.

Oefening 4.9 – Streepjes Open bij de oefeningenbestanden van hoofdstuk 4 het project *Turven*. In de klasse `StreepjesTellen` simuleren we het turven van fouten die de leerkracht op het bord schrijft. Met turven bedoelen we streepjes zetten. Als de leraar bijvoorbeeld drie fouten maakt, dan houden we dat als volgt bij: "|||".

Je krijgt twee velden, `naam` en `streepjes`, beiden van het type `String`, cadeau. De constructor vraagt alleen de naam van de leraar. Programmeer de volgende methoden:

- `void print()`
  Toont de naam van de leerkracht in hoofdletters, gevolgd door het aantal streepjes of fouten die de leraar gemaakt heeft.

Antwoord – Een correct antwoord is:

```
    public void print()
    {
        System.out.println(naam.toUpperCase() + ": " + streepjes);
    }
```

- `void zetStreepje()`
  De leraar maakt een fout, dus zet maar een streepje.

  Antwoord – Een correct antwoord is:

```
1    public void zetStreepje()
2    {
3        streepjes += "|";
4    }
```

- `int aantalStreepjes()`
  Geeft het aantal streepjes of fouten terug.

  Antwoord – Een correct antwoord is:

```
1    public int aantalStreepjes()
2    {
3        return streepjes.length();
4    }
```

- `boolean moetTrakteren()`
  Vanaf 10 streepjes moet de leraar trakteren. Indien de leraar minstens 10 fouten gemaakt heeft, dan is de retourwaarde `true`. Anders wordt `false` teruggegeven.

  Antwoord – Een correct antwoord is:

```
1    public boolean moetTrakteren()
2    {
3        return aantalStreepjes() > 10;
4    }
```

- `void getrakteerd()` Nadat de leraar getrakteerd heeft, mogen er 10 streepjes gewist worden. Let wel, de leraar mag pas trakteren indien er 10 fouten geturfd werden.

  Antwoord – Een correct antwoord is:

```
1    public void getrakteerd()
2    {
3        if(moetTrakteren())
4        {
5            streepjes = streepjes.substring(1, aantalStreepjes() - 10);
6        }
7    }
```

(Naar een oefening van Goderik Lefebvre, Waregem)

Oefening 4.10 – Kruiwagen Open bij de oefeningenbestanden van hoofdstuk 4 het project *Kruiwagen*.

De klasse `Kruiwagen` heeft drie velden:

- `laadVermogen`: de maximale massa die je in een kruiwagen kan laden.
- `massaHuidigeLading`: de massa van de huidige lading.
- `lading`: een `String` die de lading van de kruiwagen voorstelt. Bijvoorbeeld: "stenen aarde".

Via de constructor kan je aan het veld `laadVermogen` een waarde geven. Het veld `massaHuidigeLading` wordt vanzelfsprekend op nul geïnitialiseerd. Het veld `lading` krijgt als waarde de lege `String`.

Programmeer de volgende methoden:

- `void voegMassaToe(String)`
  Via de parameter van het type `String` geef je telkens een nieuwe lading mee. Voorbeelden van ladingen zijn `"05kg - stenen"` of `"12kg - aarde"`. Je mag er zeker van zijn dat de massa van de lading die je toevoegt steeds een getal is van twee cijfers. Bovendien staat het steeds op de eerste plaats vermeld in de parameter. Wanneer je een lading toevoegt, mag je nooit het laadvermogen overschrijden. Wanneer een nieuwe lading dat toch dreigt te doen, voeg je de lading niet toe aan de kruiwagen.

  Antwoord – Een correct antwoord is:

```
    public void voegMassaToe(String extraLading)
    {
        String massaString = extraLading.substring(0,2);

        int massaInt = Integer.parseInt(massaString);

        if(massaInt + massaHuidigeLading <= laadVermogen)
        {
            massaHuidigeLading += massaInt;
        }
    }
```

- `void voegLadingToe(String)`
  Deze methode doet hetzelfde als de vorige methode. Bovendien haal je uit de parameter het type van de lading en voeg je dit toe aan het veld `lading`. Voeg je `"05kg - stenen"` en `"12kg - aarde"` toe, dan moet de methode `void printLading()` de volgende boodschap weergeven in het *terminalvenster*: `"17 kg - stenen aarde"`.

  Antwoord – Een correct antwoord is:

```
    public void voegLadingToe(String extraLading)
    {
        String massaString = extraLading.substring(0,2);
        int massaInt = Integer.parseInt(massaString);

        String ladingString = extraLading.substring(7,extraLading.length
            ());

        if(massaInt + massaHuidigeLading <= laadVermogen)
        {
            massaHuidigeLading += massaInt;
            lading += " " + ladingString;
        }
    }
```

Oefening 4.11 – Verdeel Bij de jaarlijkse quiz van de dansclub wordt een beroep gedaan op de volgende methode om ploegjes te vormen.

```
public int groep(double lengte, int leeftijd)
{
    int groep = 0;

    if(leeftijd > 18)
    {
        if(lengte < 1.6)
        {
            groep = 1;
        }
        else
        {
            groep = 2;
        }
    }
    else
    {
        if(leeftijd < 12 && lengte < 1.3)
        {
            groep = 3;
        }
        else if(leeftijd >= 12 || lengte >= 1.3)
        {
            groep = 4;
        }
        else if(leeftijd >= 16)
        {
            groep = 5;
        }
    }

    return groep;
}
```

Wanneer je `int groep(int,int)` aanroept met de parameters uit de volgende tabel, wie zit dan in welke groep?

Naam	Lengte	Leeftijd	Groep
Emiel	1.83	17	
Marie	1.23	7	
Heleen	1.59	22	
Aaron	1.27	15	

Antwoord – Emiel zit in groep 4, Marie zit in groep 3, Heleen zit in groep 1 en Aaron zit in groep 4.

Is er een groep die, ongeacht de waarde van de parameters, leeg zal blijven

Antwoord – Groep 5 blijft steeds leeg. Personen met ee leeftijd vanaf 12 tot en met 17 jaar komen steeds in groep 4 terecht. Groep 5, voor personen van 16 of 17 jaar oud, blijft daarom leeg.

Open bij de oefeningenbestanden van hoofdstuk 4 het project *Verdeel*. Het project bevat alleen de klasse `VerdeelInGroepen`. Programmeer de methode `int ploeg(int)` die op basis van de geboortemaand van de deelnemer een quizploeg toewijst. Gebruik het `switch`-statement.

- groep 1: maanden met een volgnummer vanaf 10.
- groep 2: maanden waarin het minstens één dag lente is.
- groep 3: maanden met een even volgnummer.
- groep 4: alle andere maanden.

Antwoord – Een correct antwoord is:

```
    public int ploeg(int maand)
    {
        int ploeg = 0;

        switch(maand)
        {
            case 10:
            case 11:
            case 12:
                ploeg = 1;
                break;
            case 3:
            case 4:
            case 5:
                ploeg = 2;
                break;
            case 2:
            case 6:
            case 8:
                ploeg = 3;
                break;
            default:
                ploeg = 4;
                break;
        }

        return ploeg;
    }
```

Zijn er groepen die kans maken op minder deelnemers dan gemiddeld?

Antwoord – Alle groepen vertegenwoordigen drie maanden maar groep 3 vertegenwoordigd het minst aantal dagen.

Oefening 4.12 – Seizoenen Open bij de oefeningenbestanden van hoofdstuk 4 het project *Seizoen*. Het project bevat alleen de klasse `Seizoen`.

In de klasse `Seizoen` programmeer je de methode `String bepaalSeizoen(int, int)`. De eerste parameter is de dag van de maand en de tweede parameter het volgnummer van de maand. De methode geeft het seizoen terug. Enkele voorbeelden:

- `bepaalSeizoen(24, 10)` → `"herfst"`
- `bepaalSeizoen(21, 3)` → `"lente"`
- `bepaalSeizoen(1, 1)` → `"winter"`

(Naar een oefening van Marijke Lisabeth, Tielt)

Antwoord – Een correct antwoord is:

```
1   public String bepaalSeizoen(int dag, int maand)
2   {
3       String seizoen = "";
4
5       if(maand == 1 || maand == 2 || (maand == 12 && dag >= 21) || (maand ==
            3 && dag < 21))
6       {
7           seizoen = "winter";
8       }
9       else if(maand == 4 || maand == 5 || (maand == 3 && dag >= 21) || (
            maand == 6 && dag < 21))
10      {
11          seizoen = "lente";
12      }
13      else if(maand == 7 || maand == 8 || (maand == 6 && dag >= 21) || (
            maand == 9 && dag < 21))
14      {
15          seizoen = "zomer";
16      }
17      else if(maand == 10 || maand == 11 || (maand == 9 && dag >= 21) || (
            maand == 12 && dag < 21))
18      {
19          seizoen = "herfst";
20      }
21
22      return seizoen;
23  }
```

Oefening 4.13 – Orkaan Open bij de oefeningenbestanden van hoofdstuk 4 het project *Storm*. De schaal van Saffir-Simpson wordt gebruikt om orkanen in te delen in verschillende categorieën volgens hun kracht:

Categorie	Windsnelheden (km/u)
1	$119 - 153$
2	$154 - 177$
3	$178 - 209$
4	$210 - 249$
5	> 249

In de klasse `Orkaan` schrijven we de volgende methoden:

- In een eerste methode `boolean isOrkaanLang(int)` bepalen we of de opgegeven windsnelheid al dan niet overeenkomt met een orkaan. Windsnelheden groter dan of gelijk aan 119 km/u worden als orkaan beschouwd.

Antwoord – Een correct antwoord is:

```
    public boolean isOrkaanLang(int windsnelheid)
    {
        boolean result = true;

        if (windsnelheid < 119)
        {
            result = false;
        }

        return result;
    }
```

- In een tweede methode `int geefCategorie(int)` wordt aan de hand van de tabel weergegeven in welke categorie een opgegeven windsnelheid thuishoort. Bij windsnelheden kleiner dan 119 km/u (geen orkaan), wordt in ons geval categorie = 0 teruggegeven. Vul ook hier de code aan, zodat als resultaat de categorie wordt teruggegeven.

Antwoord – Een correct antwoord is:

```

    public int geefCategorie(int windsnelheid)
    {
        int categorie = 5;

        if (windsnelheid < 119)
        {
            categorie = 0;
        }
        else if (windsnelheid <= 153)
        {
            categorie = 1;
        }
        else if (windsnelheid <= 177)
        {
            categorie = 2;
        }
        else if (windsnelheid <= 209)
        {
            categorie = 3;
        }
        else if (windsnelheid <= 249)
        {
            categorie = 4;
        }

        return categorie;
    }
```

- Probeer de methode `boolean isOrkaanLang(int)` in één regel code te herschrijven. Dit doe je in `boolean isOrkaanKort(int)`.

Antwoord – Een correct antwoord is:

```
    public boolean isOrkaanKort(int windsnelheid)
    {
        return windsnelheid >= 119;
    }
```

Oefening 4.14 – Verkeerslichten Open bij de oefeningenbestanden van hoofdstuk 4 het project *Kruispunt*. Het project bevat alleen de klasse `Verkeerslichten`.

Je krijgt twee velden, `kleurHoofdweg` en `kleurZijweg`, beiden van het type `String`, cadeau. De constructor zet de verkeerslichten van de hoofdweg op `"groen"` en van de zijweg op `"rood"`.

Programmeer de methode `void volgendeFase()`. Telkens wanneer je deze methode oproept veranderen de lichten volgens het volgende patroon:

	Fase 1	Fase 2	Fase 3	Fase 4	Fase 1
Hoofdweg	"groen"	"oranje"	"rood"	"rood"	"groen"
Zijweg	"rood"	"rood"	"groen"	"oranje"	"rood"

Let dus op, na fase 4 komen we steeds terug in fase 1 of de oorspronkelijke situatie terecht.

Antwoord – Een correct antwoord is:

```
1   public void volgendeFase()
2   {
3       if(kleurHoofdweg.equals("groen"))
4       {
5           kleurHoofdweg = "oranje";
6       }
7       else if(kleurHoofdweg.equals("oranje"))
8       {
9           kleurHoofdweg = "rood";
10          kleurZijweg = "groen";
11      }
12      else if(kleurZijweg.equals("groen"))
13      {
14          kleurZijweg = "oranje";
15      }
16      else if(kleurZijweg.equals("oranje"))
17      {
18          kleurZijweg = "rood";
19          kleurHoofdweg = "groen";
20      }
21  }
```

Oefening 4.15 – Drankautomaat Bij de bronbestanden van hoofdstuk 4 vind je het project *Drankautomaat*. In dit project vul je enkele methoden van de klasse `Drankautomaat` verder aan:

- `void werpMuntstukIn(double)`

Antwoord – Een correct antwoord is:

```
1   public void werpMuntstukIn(double muntstuk)
2   {
3       if ((muntstuk == 0.05) || (muntstuk == 0.10) ||
4           (muntstuk == 0.20) || (muntstuk == 0.50) ||
5           (muntstuk == 1) || (muntstuk == 2))
6       {
7           ingeworpenBedrag += muntstuk;
8       }
9   }
```

- `void kiesDrankje(String)`

Antwoord – Een correct antwoord is:

```
    public void kiesDrankje(String drankje)
    {
        double prijs = prijsDrankje(drankje);

        if (ingeworpenBedrag >= prijs &&
            aantalDrankjesInAutomaat(drankje) > 0)
        {
            verminderAantalDrankjes(drankje);
            wisselgeld += ingeworpenBedrag - prijs;
            bedragInAutomaat += prijs;
        }
    }
```

- `double prijsDrankje(String)`

Antwoord – Een correct antwoord is:

```
    public double prijsDrankje(String drankje)
    {
        double prijs;

        if (drankje == "water")
        {
            prijs = 0.60;
        }
        else if ((drankje == "cola") || (drankje == "fanta") || (drankje
            == "sprite"))
        {
            prijs = 0.90;
        }
        else if (drankje == "fruitsap")
        {
            prijs = 1.20;
        }
        else
        {
            prijs = 0;
        }

        return prijs;
    }
```

- `int aantalDrankjesInAutomaat(String)`

Antwoord – Een correct antwoord is:

```
    private int aantalDrankjesInAutomaat(String drankje)
    {
        int aantal;

        switch(drankje)
        {
            case "water":
                aantal = aantalWater;
                break;
            case "cola":
                aantal = aantalCola;
                break;
            case "fanta":
                aantal = aantalFanta;
                break;
            case "sprite":
                aantal = aantalSprite;
                break;
            case "fruitsap":
                aantal = aantalFruitsap;
                break;
            default:
                aantal = 0;
                break;
        }

        return aantal;
    }
```

- `void verminderAantalDrankjes(String)`

Antwoord – Een correct antwoord is:

```
    private void verminderAantalDrankjes(String drankje)
    {
        if(drankje.equals("water"))
        {
            aantalWater--;
        }
        else if(drankje.equals("cola"))
        {
            aantalCola--;
        }
        else if(drankje.equals("fanta"))
        {
            aantalFanta--;
        }
        else if(drankje.equals("sprite"))
        {
            aantalSprite--;
        }
        else if(drankje.equals("fruitsap"))
        {
            aantalFruitsap--;
        }
    }
```

Sommige methoden zijn `private` methoden. Voor het testen van je code maak je ze misschien het best tijdelijk `public`.

De werking van de methoden die je moet aanvullen, staat telkens in de commentaarregels boven de methoden. Alvorens je de oefening aanvat, maak je het best een aantal objecten van deze klasse om de reeds aanwezige methoden te testen. Zo krijg je inzicht in de probleemstelling.

Oefening 4.16 – Woordslang Bij de bronbestanden van hoofdstuk 4 vind je het project *Woordslang*.

Een woordslang begint steeds met het woord "tomaat". Wil je de woordslang goed aanvullen dan moet het volgende woord met de laatste letter van het vorige woord beginnen, in dit geval een 't'. Zo is "trein" een goed woord om de woordslang aan te vullen. Een voorbeeld van een goed rijtje van vijf woorden:

"tomaat" → "trein" → "narcis" → "sleutel" → "list"

In de volgende klassendefinitie doet de methode void voegWoordToe(String) niet helemaal wat je zou kunnen verwachten. Wat doet deze methode dan precies?

```
public class Woordslang
{
    private String laatsteWoord;
    private int aantalWoorden;

    public Woordslang()
    {
        this.laatsteWoord = "tomaat";
        aantalWoorden = 0;
    }

    public void voegWoordToe(String woord)
    {
        if(woord.substring(woord.length() - 1, woord.length()).equals(laatsteWoord.substring(0,1)))
        {
            laatsteWoord = woord;
            aantalWoorden++;
        }
    }
}
```

Antwoord – Deze methode speelt woordslang door steeds vooraan een woord toe te voegen. Er wordt gekeken naar de eerste letter van het laatste woord en de laatste letter van het volgende woord. Enkel indien deze letters gelijk zijn, wordt het woord toegevoegd aan de woordslang.

Programmeer de correcte methode void woordslang(String).

Antwoord – Een correct antwoord is:

```
1    public void voegWoordToe(String woord)
2    {
3        if(woord.substring(0,1).equals(
4            laatsteWoord.substring(woord.length() - 1, woord.length())))
5        {
6            laatsteWoord = woord;
7            aantalWoorden++;
8        }
9    }
```

Oefening 4.17 – Telegram Open bij de oefeningenbestanden van hoofdstuk 4 het project *Telegram*. Een telegram bestaat uit een `bericht`, van het type `String`. In de klasse `Telegram` programmeer je een aantal methoden:

- `void voegZinToe(String)`
  Voegt aan het telegrambericht een stukje tekst toe. Hoofdletters worden vervangen door kleine letters en na het toegevoegde stukje tekst wordt steeds `" -(STOP)- "` (let op de spaties) toegevoegd.

  Antwoord – Een correct antwoord is:

```
    public void voegZinToe(String zin)
    {
        bericht += zin.toLowerCase() + " STOP ";
    }
```

- `double kostPrijs()`
  Berekent de prijs van het telegram. Wanneer het telegram uit tien letters of minder bestaat, is de kostprijs € 15. Per extra letter komt er € 0,50 bovenop de prijs. Een telegram van 20 letters kost dus € 20.

  Antwoord – Een correct antwoord is:

```
    public double kostPrijs()
    {
        double kostprijs = 15;

        if (bericht.length() > 10)
        {
            kostprijs += (bericht.length() - 10) * 0.5;
        }
        return kostprijs;
    }
```

- `void verkleinBericht(int)`
  Kort het bericht in door achteraan in de tekst een aantal letters weg te laten. Het aantal letters dat je weglaat, wordt via een parameter doorgegeven.

  Antwoord – Een correct antwoord is:

```
    public void verkleinBericht(int aantal)
    {
        if (aantal < bericht.length())
        {
            bericht = bericht.substring(0,bericht.length()-aantal);
        }
    }
```

Interactie tussen objecten 5

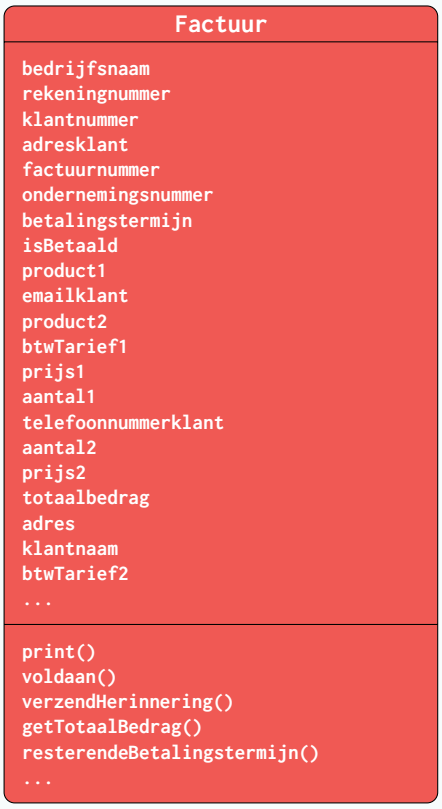
Oefening 5.1 We wensen een facturatieprogramma te maken met de volgende eigenschappen:

- Het programma moet een factuur kunnen aanmaken.
- Facturen die betaald werden, moet je op voldaan kunnen zetten.
- Een factuur moet je kunnen printen.

Bedenk één klasse die dit alles kan. Om het niet te moeilijk te maken mag je aannemen dat op een factuur slechts één verkocht product kan staan waarvan een aantal items verkocht werden.

Tip: zoek op het internet wat allemaal op een factuur moet staan.

Antwoord – Een mogelijk antwoord is:

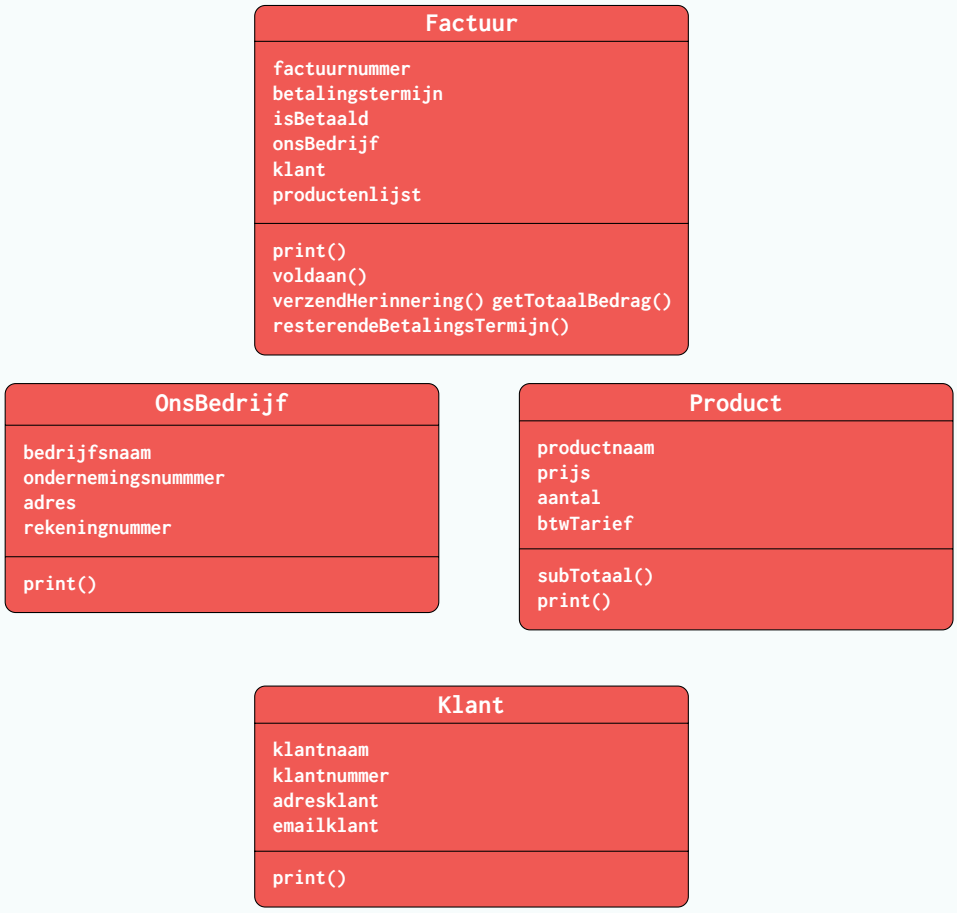


Oefening 5.2 Met abstractie en modularisatie in gedachten, bekijken we opnieuw de vraag naar een facturatieprogramma met de volgende eigenschappen:

- Het programma moet een factuur kunnen aanmaken.
- Facturen die betaald werden, moet je op voldaan kunnen zetten.
- Een factuur moet je kunnen printen.

Bedenk nu dus uit welke verschillende onderdelen een factuur bestaat en koppel aan die onderdelen een klasse. Probeer voor elke klasse kort velden en methoden te schetsen.

Antwoord – Een mogelijk antwoord vind je in het volgende diagram. Vanzelfsprekend kan je nog veel verder gaan in het modularisatieproces.



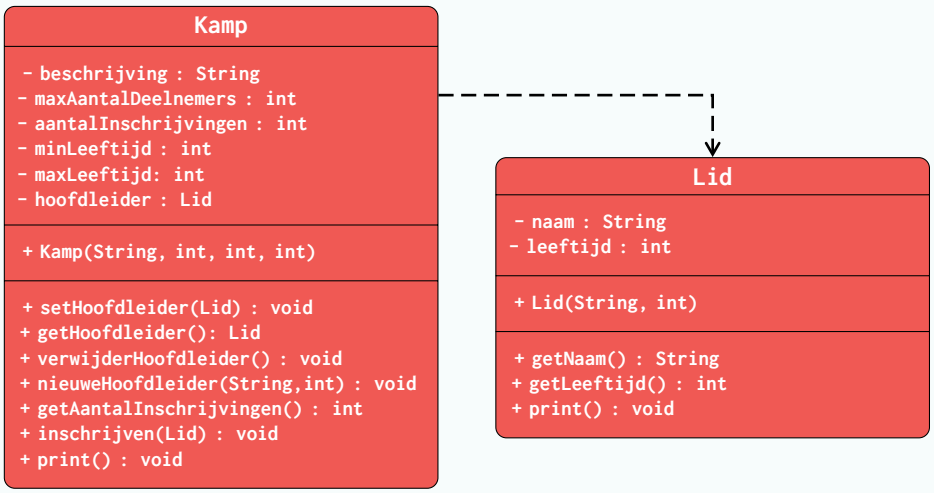
Oefening 5.3 – Scoutskamp In deze oefening gaan we een klassendiagram maken om het inschrijven van scoutsleden voor een scoutskamp te automatiseren. Haal uit de beschrijving van het probleem de klassen, velden, constructoren en methoden. Maak vervolgens een passend klassendiagram.

De leden van de scouts hebben allen een naam en een zekere leeftijd. Wanneer ze zich inschrijven voor een scoutskamp dan wordt gekeken of de leeftijd van het scoutslid aan de leeftijdscategorie waarvoor het kamp georganiseerd wordt voldoet. Wanneer dit het geval is, en het maximale aantal deelnemers nog niet overschreden werd, wordt het aantal deelnemers van het kamp verhoogd met één.

Behalve de beschrijving van het kamp houden we ook een verwijzing naar de hoofdleider van het kamp bij. De hoofdleider is een gewoon scoutslid dat toevallig voor dit kamp meer bevoegdheden gekregen heeft. Een hoofdleider moet je kunnen verwijderen. Bovendien vragen we je via naam en leeftijd een hoofdleider toe te wijzen aan een kamp.

Van elk kamp moet je de naam van de hoofdleider en het aantal inschrijvingen kunnen printen naar het *Terminalvenster*.

Antwoord – Een correct antwoord is:



Oefening 5.4 – Scoutskamp In oefening 5.3 hebben we gevraagd om een klassendiagram te maken bij een beschrijving van een scoutskamp. We bouwen op deze oefening verder.

Voorbereidend werk:

- Maak een nieuw *BlueJ*-project met naam *Scoutskamp* aan.
- Programmeer eerst de klasse `Lid`. Compileer en test deze klasse grondig!

Antwoord – Een correct antwoord is:

```
1   public class Scoutslid
2   {
3       private String naam;
4       private int leeftijd;
5
6       public Scoutslid(String naam, int leeftijd)
7       {
8           this.naam = naam;
9           this.leeftijd = leeftijd;
10      }
11
12      public int getLeeftijd()
13      {
14          return leeftijd;
15      }
16
17      public void print()
18      {
19          System.out.println(naam + " - " + leeftijd + " jaar");
20      }
21  }
```

- Programmeer de velden en de constructor van de klasse Kamp. In de constructor krijgen alle velden expliciet een waarde, ook al is dit soms de defaultwaarde.

Antwoord – Een correct antwoord is:

```
public class Kamp
{
    private String beschrijving;
    private Scoutslid hoofdleider;
    private int minLeeftijd;
    private int maxLeeftijd;
    private int maxDeelnemers;
    private int inschrijvingen;

    public Kamp(String beschrijving, int minLeeftijd, int maxLeeftijd,
    int maxDeelnemers)
    {
        this.beschrijving = beschrijving;
        this.minLeeftijd = minLeeftijd;
        this.maxLeeftijd = maxLeeftijd;
        this.maxDeelnemers = maxDeelnemers;
        hoofdleider = null;
        inschrijvingen = 0;
    }

}
```

Een mogelijke oplossing bij oefening 5.3 vind je in het klassendiagram. Programmeer nu de methode void setHoofdleider(Lid). Het lid dat je als actuele parameter meegeeft aan deze methode wordt de hoofdleider van het scoutskamp.

Antwoord – Een correct antwoord is:

```
    public void setHoofdleider(Scoutslid hoofdleider)
    {
        this.hoofdleider = hoofdleider;
    }
```

Oefening 5.5 – Scoutskamp Je werkt verder in het project dat je aangemaakt hebt bij oefening 5.4. Programmeer de volgende methode:

- void nieuweLeider(String, int) geeft aan het kamp een nieuwe hoofdleider. De naam en de leeftijd van de hoofdleider vind je terug in de parameters van deze methode.

Antwoord – Een correct antwoord is:

```
    public void nieuweHoofdleider(String naam, int leeftijd)
    {
        hoofdleider = new Scoutslid(naam, leeftijd);
    }
```

Oefening 5.6 – Scoutskamp Je werkt verder in het project dat je aangemaakt hebt bij oefening 5.4. Programmeer de volgende methode:

- `Lid getHoofdleider()` geeft een verwijzing naar de hoofdleider terug.

Antwoord – Een correct antwoord is:

```
    public Scoutslid getHoofdleider()
    {
        return hoofdleider;
    }
```

Oefening 5.7 – Scoutskamp Je werkt verder in het project dat je aangemaakt hebt bij oefening 5.4. Programmeer de volgende methode:

- `void verwijderLeider()`: deze methode verwijdert de hoofdleider van het vakantiekamp. Het kamp heeft nu geen hoofdleider meer.

Antwoord – Een correct antwoord is:

```
    public void verwijderLeider()
    {
        hoofdleider = null;
    }
```

Oefening 5.8 – Scoutskamp Je werkt verder in het project dat je aangemaakt hebt bij oefening 5.4. Programmeer de volgende methode:

- `void print()`: deze methode print alle informatie van het vakantiekamp af in het *Terminalvenster*. Allen wanneer er een hoofdleider is, toon je de naam en de leeftijd.

Antwoord – Een correct antwoord is:

```
    public void print()
    {
        String info = "Beschrijving: " + beschrijving + "\n";
        info += "Leeftijdscategorie: " + minLeeftijd + " - " + maxLeeftijd +
                " jaar\n";
        info += "Bezettingsgraad: " + inschrijvingen + "/" + maxDeelnemers;

        System.out.println(info);

        if(hoofdleider != null)
        {
            System.out.print("Hoofdleider: ");
            hoofdleider.print();
        }
        else
        {
            System.out.println("Geen hoofdleider toegewezen");
        }
    }
```

Gebruik in deze methode slechts één methode van de klasse `Lid`. Verkies je deze manier van werken boven een `String toString()`-methode zoals in de klasse `Harnas`? Waarom?

Antwoord – Eigenlijk maakt het niet zoveel verschil uit. Maar voor de leesbaarheid is `hoofdleider.print()` beter dan `System.out.println(hoofdleider.toString())`.

Oefening 5.9 – Scoutskamp Je werkt verder in het project dat je aangemaakt hebt bij oefening 5.4. Programmeer de volgende methoden:

- `void inschrijven(Lid)`: het lid dat je via de parameter meegeeft aan deze methode schrijf je in voor het kamp. Controleer eerst de leeftijd van het lid en het aantal beschikbare plaatsen alvorens het lid in te schrijven.

Antwoord – Een correct antwoord is:

```
1     public void inschrijven(Scoutslid lid)
2     {
3         if( (lid.getLeeftijd() >= minLeeftijd &&
4                 lid.getLeeftijd() <= maxLeeftijd) &&
5                 inschrijvingen < maxDeelnemers)
6         {
7             inschrijvingen++;
8         }
9     }
```

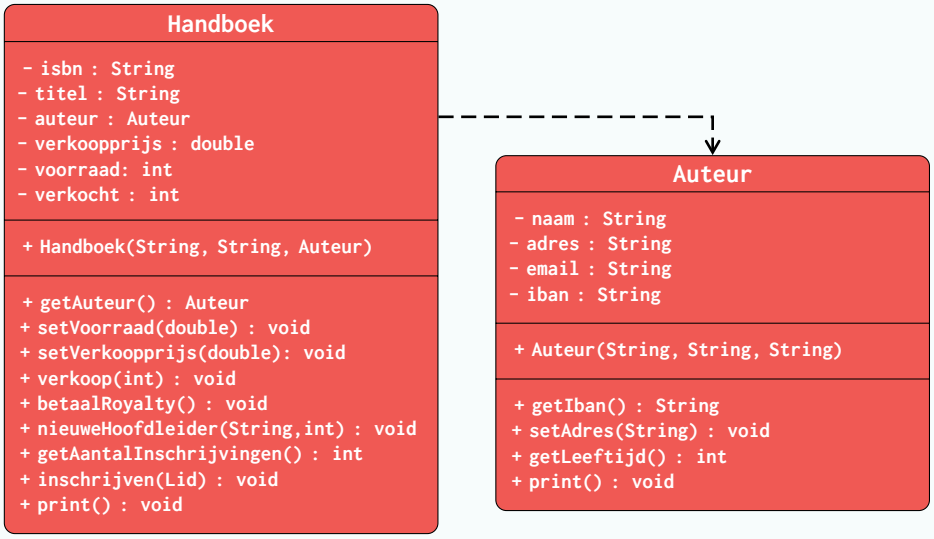
Oefening 5.10 – Acco Wanneer je een handboek wenst uit te geven, zal een uitgeverij steeds je naam, adres, e-mailadres en rekeningnummer vragen. Een kwestie van te weten aan wie royalty's te beurt vallen.

De uitgeverij van dit handboek, Acco, heeft een voorraad handboeken liggen in de drukkerij. Voor Acco zijn ISBN-nummer, titel, auteur, verkoopprijs en het aantal stuks in voorraad belangrijke parameters. Wanneer ze in hun beheersprogramma informatie over dit handboek opvragen, dan verschijnt de volgende informatie op het scherm:

Schrijf bij dit scenario een klassendiagram.

(Naar een oefening van Annick Renders, Veurne)

Antwoord – Een correct antwoord is:



Oefening 5.11 – Ctrl + c en Ctrl + v In deze oefening gaan we een bestand proberen te kopiëren van een computer naar een USB-stick. We hebben dus een computer, USB-stick en een bestand nodig.

Een computer houdt twee zaken bij:

- de computernaam,
- een verwijzing naar een USB-stick die eventueel ingeplugd werd.

Een USB-stick houdt volgende zaken bij:

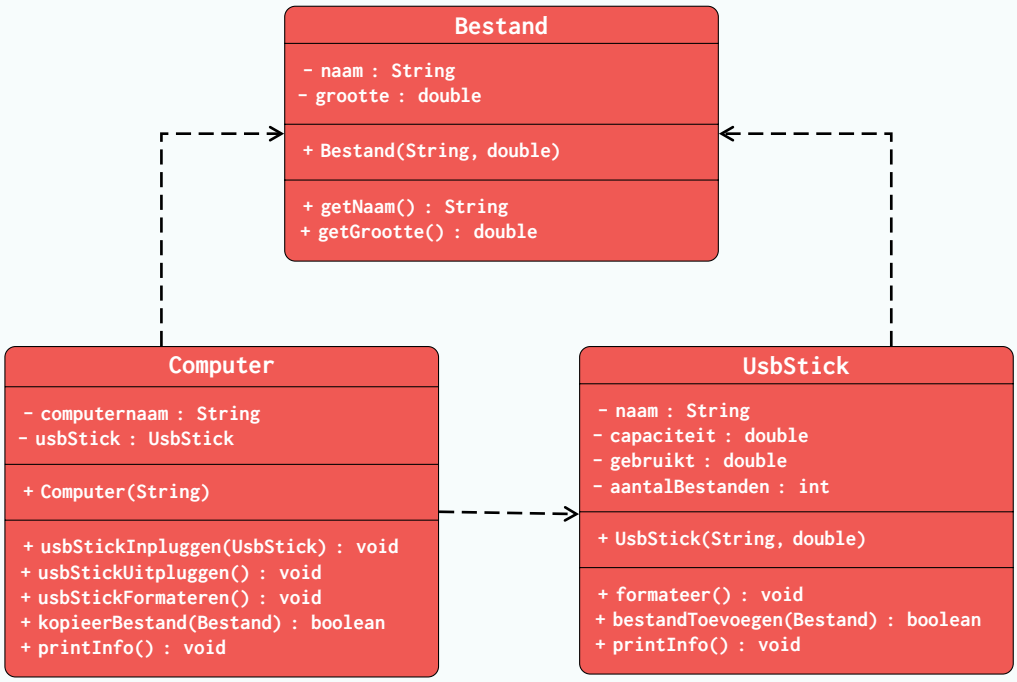
- de naam van de USB-stick,
- de capaciteit uitgedrukt in GB,
- de gebruikte capaciteit uitgedrukt in GB,
- het aantal bestanden die er op de USB-stick staan.

Een bestand houdt slechts één zaak bij: de grootte uitgedrukt in MB.

Schrijf een klassendiagram dat het mogelijk maakt om een bestand vanaf je computer te kopiëren naar een USB-stick. Het spreekt voor zich dat je de USB-stick moet kunnen in- en uitpluggen. Zowel van de computer als van de USB-stick moet je informatie kunnen tonen in het *Terminalvenster*. Als de USB-stick ingeplugd is in de computer, kan je de USB-stick formatteren.

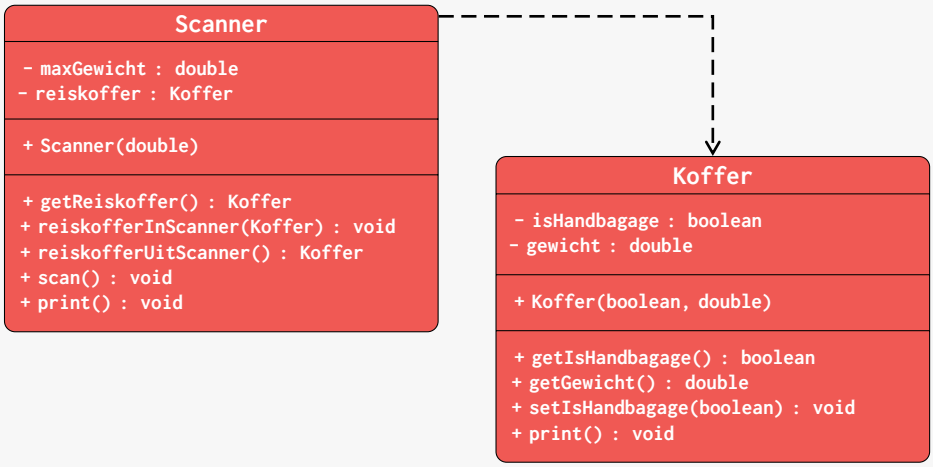
(Naar een oefening van Annick Renders, Veurne)

Antwoord – Een correct aantwoord is:



Bij het kopiëren van een bestand, laten we de methoden `kopieerBestand(Bestand)` van de klasse `Computer` en `bestandToevoegen(Bestand)` van de klasse `UsbStick` een booleaanse waarde terug geven. Zo weten we of het kopiëren gelukt is. Het kopiëren kan mislukken omdat er geen USB-stick aanwezig is of omdat de resterende geheugenplaats van de USB-stick ontoereikend is.

Oefening 5.12 – Bagagescanner In deze oefening simuleer je de werking van de bagagescanner in een luchthaven. Het klassendiagram werd voor jou reeds gemaakt.



Een woordje uitleg bij de twee klassen:

- `Koffer`
  Een koffer heeft een gewicht en een type. Ofwel is de koffer handbagage, ofwel is de koffer een reiskoffer die in het ruim van het vliegtuig geladen moet worden. Dit laatste type koffer is het duurste type. Veel passagiers proberen dan ook een te zware koffer als handbagage mee te nemen. Vandaar dat een bagagescanner het type van de koffer moet kunnen wijzigen.

Antwoord – Een correcte implementatie van de klasse `Koffer` is:

```
public class Koffer
{
    private boolean isHandbagage;
    private double gewicht;

    public Koffer(boolean isHandbagage, double gewicht)
    {
        this.isHandbagage = isHandbagage;
        this.gewicht = gewicht;
    }

    public boolean getIsHandbagage()
    {
        return isHandbagage;
    }

    public double getGewicht()
    {
        return gewicht;
    }

    public void setIsHandbagage(boolean isHandbagage)
    {
        this.isHandbagage = isHandbagage;
    }

    public void print()
    {
        String info = "gewicht: " + gewicht + " kg\n";

        if(! isHandbagage)
        {
            info += "geen";
        }

        info += "handbagage";

        System.out.println(info);
    }
}
```

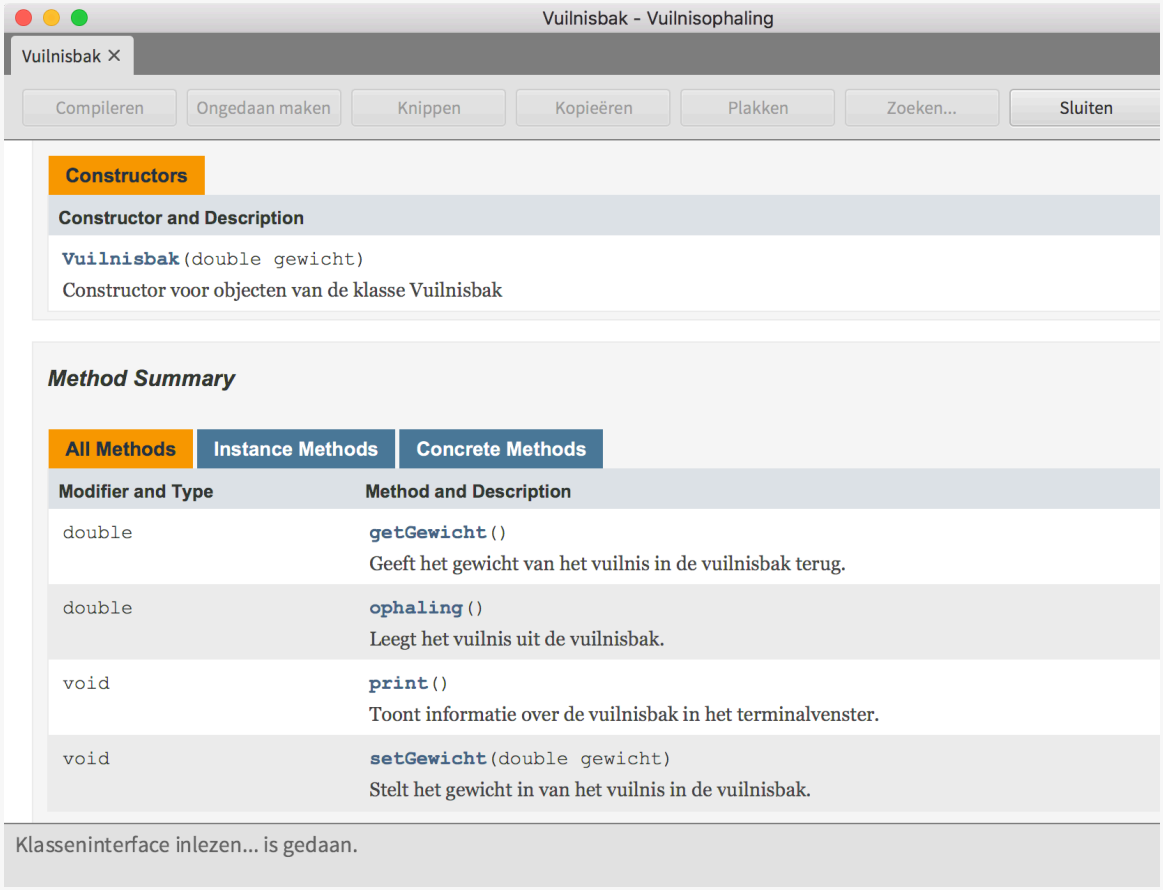
- Scanner
  De bagagescanner weegt een koffer. De scanner controleert of elke koffer het juiste type gekregen heeft. Koffers die meer wegen dan een zeker maximumgewicht mogen niet gelabeld zijn als handbagage. Lichte koffers mogen als handbagage mee in het vliegtuig. De bagagescanner kan alleen koffers scannen die in het toestel aangebracht werden. Je kan nooit een koffer in de bagagescanner plaatsen wanneer nog een koffer aanwezig is.

Antwoord – Een correcte implementatie van de klasse Scanner is:

```
public class Scanner
{
    private double maxGewicht;
    private Koffer reiskoffer;

    public Scanner(double maxGewicht)
    {
        this.maxGewicht = maxGewicht;
        reiskoffer = null;
    }

    public Koffer getReisKoffer()
    {
        return reiskoffer;
    }

    public void kofferInScanner(Koffer reiskoffer)
    {
        if(this.reiskoffer == null)
        {
            this.reiskoffer = reiskoffer;
        }
    }

    public Koffer kofferUitScanner()
    {
        Koffer hulp = reiskoffer;
        reiskoffer = null;
        return hulp;
    }

    public void scan()
    {

        if(reiskoffer != null)
        {
            if(reiskoffer.getGewicht() > maxGewicht)
            {
                reiskoffer.setIsHandbagage(false);
            }
            else
            {
                reiskoffer.setIsHandbagage(true);
            }
        }
    }

    public void print()
    {
        System.out.println("Het maximaal gewicht voor handbagage is "
        + maxGewicht + " kg");

        if(reiskoffer !=null)
        {
            reiskoffer.print();
        }
        else
        {
            System.out.println("Er zit geen reiskoffer in de scanner");
        }
    }
}
```

Oefening 5.13 – Vuilnisophaling In deze oefening simuleren we het ophalen van vuilnisbakken door de vuilniswagen.

De klasse `Vuilnisbak` werd reeds geprogrammeerd. De broncode kan je niet bekijken noch veranderen. Je vindt wel een API-pagina wanneer je in *BlueJ* op de klasse klikt. Bestudeer de werking van de klasse grondig!



Het ophalen van het huisvuil gebeurt in drie fasen:

- De vuilnisbak wordt geklikt aan de vuilniswagen.
- Vervolgens wordt de vuilnisbak omgekeerd zodat het vuilnis in de vuilniswagen gestort wordt.
- Ten slotte wordt de vuilnisbak weer losgemaakt van de vuilniswagen.

De werking van de vuilniswagen ga jij in de klasse `Vuilniswagen` programmeren. In de klassendefinitie vind je drie velden terug:

- `lading`: de totale hoeveelheid vuilnis (uitgedrukt in kg) die de vuilniswagen reeds opgehaald heeft,
- `maxLading`: de maximale hoeveelheid vuilnis (uitgedrukt in kg) die de vuilniswagen kan ophalen,
- `vuilnisbak`: de vuilnisbak die wordt vastgeklikt aan de vuilniswagen (of niets).

Naast de velden krijg je ook de constructor cadeau. Programmeer volgende methoden:

- `void vuilnisbakAanhaken(Vuilnisbak)`
  De vuilnisbak in de parameter wordt aan de vuilniswagen vastgeklikt.

Antwoord – Een correct antwoord is:

```
1    public void vuilnisbakAanhaken(Vuilnisbak vuilnisbak)
2    {
3        this.vuilnisbak = vuilnisbak;
4    }
```

- `void vuilnisbakLoshaken()`
  De vuilnisbak wordt losgehaakt van de de vuilniswagen.

  Antwoord – Een correct antwoord is:

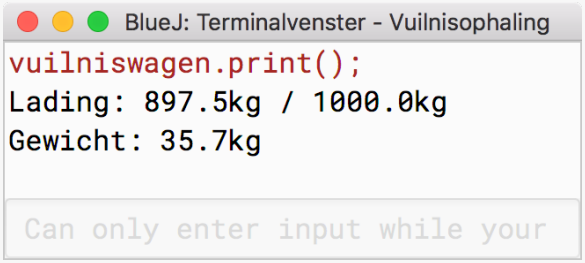
```
    public void vuilnisbakLoshaken()
    {
        this.vuilnisbak = null;
    }
```

- `void vuilnisbakLegen()`
  Wanneer een vuilnisbak aangehaakt werd en het maximumgewicht van het vuilnis in de vuilniswagen niet overschreden zal worden, dan wordt het vuilnis uit de vuilnisbak gestort in de vrachtwagen. Na het legen van de vuilnisbak kan de toestand van de vuilnisbak en de vuilniswagen gewijzigd zijn. Om dit na te gaan, inspecteer je na het gebruik van deze methode de testobjecten in de *Objectenbank*.

  Antwoord – Een correct antwoord is:

```
    public void vuilnisbakLegen()
    {
       if(vuilnisbak != null && (vuilnisbak.getGewicht() + lading <
           maxLading))
       {
            lading += vuilnisbak.ophaling();
       }
    }
```

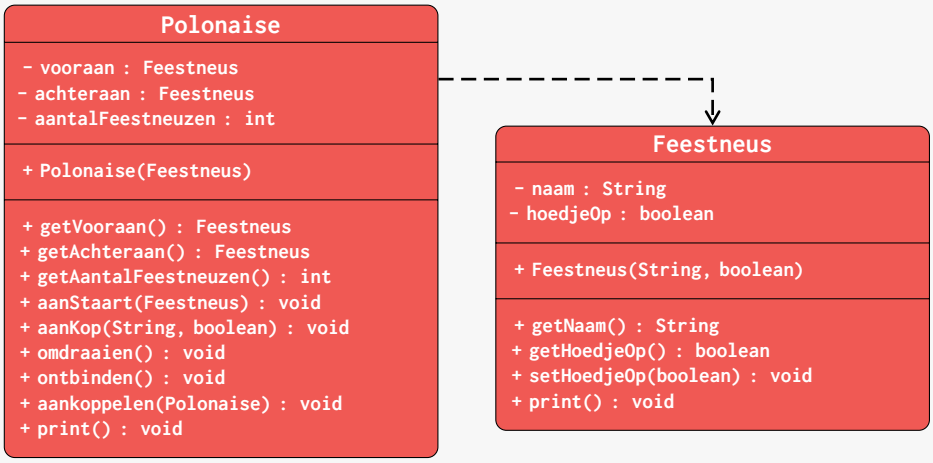
- `void print()`
  Deze methode toont informatie over de vuilniswagen in het *Terminalvenster*. In het volgende voorbeeld bevat de vuilniswagen 879,5 kg vuilnis. De maximale lading is 1000 kg. Er werd net een vuilnisbak met 35,7 kg vuilnis aangehaakt aan de vuilniswagen.



  Antwoord – Een correct antwoord is:

```
    public void print()
    {
        System.out.println("Lading: " + lading + "kg / " + maxLading + "
            kg");

        if(vuilnisbak != null)
        {
            vuilnisbak.print();
        }
        else
        {
            System.out.println("Geen vuilnisbak vastgeklikt");
        }
    }
```

Oefening 5.14 – Polonaise Ooit al eens de polonaise gedanst? Niet moeilijk, gewoon allemaal achter elkaar op een rijtje lopen en gek doen. Bij deze oefening vind je het klassendiagram die hoort bij dit feestje met daarna een woordje uitleg.



Een klein woordje uitleg bij de twee klassen:

- `Feestneus`
  Objecten van deze klasse hebben een naam. De echte feestneuzen onderscheiden zich door het dragen van een hoedje. Wie het feesten moe is, kan gerust zijn of haar hoedje afzetten om het later weer op te zetten.

Antwoord – Een correcte implementatie van de klasse Feestneus is:

```
public class Feestneus
{
    private String naam;
    private boolean hoedjeOp;

    public Feestneus(String naam, boolean hoedjeOp)
    {
        this.naam = naam;
        this.hoedjeOp = hoedjeOp;
    }

    public String getNaam()
    {
        return naam;
    }

    public boolean getHoedjeOp()
    {
        return hoedjeOp;
    }

    public void setHoedjeOp(boolean hoedjeOp)
    {
        this.hoedjeOp = hoedjeOp;
    }

    public void print()
    {
        String info = naam + " heeft ";

        if(! hoedjeOp)
        {
            info += "g";
        }

        info += "een hoedje op";

        System.out.println(info);
    }
}
```

- Polonaise
  Een polonaise is een rijtje mensen die achter elkaar lopen. Onze polonaise houdt niet alleen het aantal mensen maar ook de eerste en de laatste feestneus bij. Wanneer de dj 'omdraaien' roept, wordt de eerste feestneus plots de laatste en omgekeerd. De polonaise registreert steeds wanneer nieuwe feestneuzen voor- of achteraan aansluiten. Wanneer de ambiance er echt in zit, kan de ene polonaise aansluiten bij de andere. Bij een slechte dansplaat gaat iedereen liever zitten om een biertje te drinken.

Antwoord – Een correcte implementatie van de klasse Polonaise is:

```
public class Polonaise
{
    private Feestneus vooraan; private Feestneus achteraan;
    private int aantalFeestneuzen;

    public Polonaise(Feestneus feestneus)
    {
        this.vooraan = feestneus; this.achteraan = feestneus;
        aantalFeestneuzen = 1;
    }
    public Feestneus getVooraan()
    {
        return vooraan;
    }
    public Feestneus getAchteraan()
    {
        return achteraan;
    }
    public int getAantalFeestneuzen()
    {
        return aantalFeestneuzen;
    }
    public void aanKop(String naam, boolean hoedjeOp)
    {
        vooraan = new Feestneus(naam, hoedjeOp);
        aantalFeestneuzen++;
    }
    public void aanStaart(Feestneus achteraan)
    {
        this.achteraan = achteraan;
        aantalFeestneuzen++;
    }
    public void ontbinden()
    {
        aantalFeestneuzen = 0;
        vooraan = null; achteraan = null;
    }
    public void omdraaien()
    {
        Feestneus hulp = achteraan;
        achteraan = vooraan; vooraan = hulp;
    }
    public void koppelen(Polonaise anderePolonaise)
    {
        achteraan = anderePolonaise.getAchteraan();
        aantalFeestneuzen += anderePolonaise.getAantalFeestneuzen();
        anderePolonaise.ontbinden();
    }
    public void print()
    {
        System.out.println("Aantal dansers: " + aantalFeestneuzen);

        if(aantalFeestneuzen != 0)
        {
            System.out.print("Vooraan: "); vooraan.print();
            System.out.print("Achteraan: "); achteraan.print();
        }
    }
}
```

(Naar een oefening van Goderik Lefebvre, Waregem)

Oefening 5.15 – iMessage Open bij de oefeningenbestanden van hoofdstuk 5 het project *iMessage*.

De klasse `Message` stelt een berichtje voor dat we tussen twee smartphones kunnen versturen. Deze klasse is heel eenvoudig: het veld `tekst` kan je via de constructor initialiseren en je hebt alleen de accessormethode `String getTekst()` ter beschikking.

De klasse `Smartphone` stelt een smartphone voor waarop één berichtje kan staan. Dit bericht kan het bericht zijn dat je wenst te versturen of een bericht dat je wenst te ontvangen. Je kan er ook voor zorgen dat er geen bericht staat op de smartphone. Het bericht wordt bijgehouden in het veld `bericht`, een instantie van de klasse `Message`.

Al het werk gebeurt zoals in de echte wereld met de smartphone. Je zal dus in de *objectenbank* geen instantie van de klasse `Message` moeten aanmaken. Je programmeert de volgende methoden in de klasse `Smartphone`:

- `void schrijfBericht(String)`:
  Je geeft deze methode een `String` mee met de boodschap die je wenst te versturen. De smartphone maakt een instantie van de klasse `Message` aan. Dit object is het enige bericht dat de smartphone kan bijhouden.

  Antwoord – Een correct antwoord is:

```
    public void schrijfBericht(String tekst)
    {
        bericht = new Message(tekst);
    }
```

- `void printBericht()`:
  Print het bericht op de smartphone af in het *Terminalvenster*. Indien er geen bericht staat op de smartphone, print je de boodschap "`Geen berichten`".

  Antwoord – Een correct antwoord is:

```
    public void printBericht()
    {
        if(bericht != null)
        {
            System.out.println(bericht.getTekst());
        }
        else
        {
            System.out.println("Geen berichten");
        }
    }
```

- `void verwijderBericht()`:
  Verwijdert het bericht van de smartphone.

  Antwoord – Een correct antwoord is:

```
    public void verwijderBericht()
    {
        bericht = null;
    }
```

- `void ontvangBericht(Message)`:
  Ontvangt het bericht in de parameter, ook als er op de smartphone reeds een bericht staat (het bericht wordt dan overschreven).

  Antwoord – Een correct antwoord is:

```
1    public void ontvangBericht(Message bericht)
2    {
3        this.bericht = bericht;
4    }
```

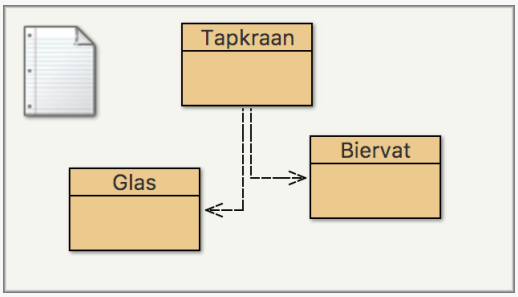
- `void verzendBericht(Smartphone)`:
  Verstuurt het bericht op de smartphone naar de smartphone waar de parameter naar verwijst. Uiteraard moet op de smartphone van de verzender een bericht staan om te kunnen verzenden. Zo niet gebeurt er niets.

  Antwoord – Een correct antwoord is:

```
1    public void verzendBericht(Smartphone ontvanger)
2    {
3        ontvanger.ontvangBericht(this.bericht);
4    }
```

Oefening 5.16 – Bier tappen Open bij de oefeningenbestanden van hoofdstuk 5 het project *Bier*. De oefening speelt zich af in het café en simuleert het tappen van een biertje.

Hieronder vind je het klassendiagram bij dit project:



Het is duidelijk dat de klasse Tapkraan het werk coördineert:

- Aan de tapkraan kan je een vat bier aan- of afkoppelen.
- Je kan een glas bier onder de tapkraan houden om een biertje in te schenken.

In deze oefening vullen we ontbrekende methoden in de drie klassen aan.

- `Biervat`
  - Objecten van deze klasse stellen biervaten met een inhoud uitgedrukt in liter voor. De inhoud kan je als parameter meegeven met de constructor. Telkens wanneer een biertje getapt wordt, moeten we de inhoud verminderen met een zekere hoeveelheid. We kunnen biertjes tappen tot het vat leeg is.
  - De methode die je nog moet aanvullen is de methode `double tapBier (double)`. Aan deze methode kunnen we via de parameter de hoeveelheid bier meegeven die men wenst te tappen, uitgedrukt in liter. De inhoud van het vat bier wordt verminderd en de getapte hoeveelheid bier, opnieuw uitgedrukt in liter, wordt als retourwaarde teruggegeven (bier vloeit vanuit het vat naar de tapkraan). Denk goed na wat er gebeurt indien het vat minder bier bevat dan gevraagd!

Antwoord – Een correct antwoord is:

```
public double tapBier(double gevraagd)
{
    double gepompt = 0;

    if(gevraagd <= inhoud)
    {
        gepompt = gevraagd;
        inhoud -= gevraagd;
    }
    else
    {
        gepompt = inhoud;
        inhoud = 0;
    }

    return gepompt;
}
```

- Glas
  - Een instantie van de klasse Glas houdt het volume van een glas bij (veld volume) en de inhoud bier in het glas (veld inhoud). Naast de reeds geprogrammeerde accessormethode en de methode void adFundum() vinden we twee methoden zonder body.
  - De methode void vulBij(double) vult het glas bij met een zekere hoeveelheid bier, uitgedrukt in liter. Indien je meer wil bijvullen dan mogelijk, loopt het glas over en gaat er bier verloren.
  - Met de methode void drink(double) kan je een hoeveelheid bier, uitgedrukt in liter drinken. Je kan maar zoveel bier drinken als in het glas zit.

Antwoord – Een correct antwoord is:

```
    public void vulBij(double hoeveelheid)
    {
        inhoud += hoeveelheid;

        if(inhoud > volume)
        {
            inhoud = volume;
        }
    }

    public void drink(double hoeveelheid)
    {
        inhoud -= hoeveelheid;

        if(inhoud < 0)
        {
            inhoud = 0;
        }
    }
```

Wanneer de klassen Biervat en Glas goed functioneren, programmeer je de klasse Tapkraan. De klasse Tapkraan bestaat uit een defaultconstructor en het veld vat van het type Biervat. Wanneer je een tapkraan aanmaakt, is het logisch dat er nog geen vat is aangesloten.

- Tapkraan
  - void sluitVatAan(Biervat): het vat dat je via de parameter aanbiedt, wordt aangesloten aan de tapkraan.

- `void verwijderVat()`: het vat bier wordt van de tapkraan losgekoppeld.
- `void tapBiertje(Glas)`: vult het glas bier tot aan de rand vol.

Antwoord – Een correct antwoord is:

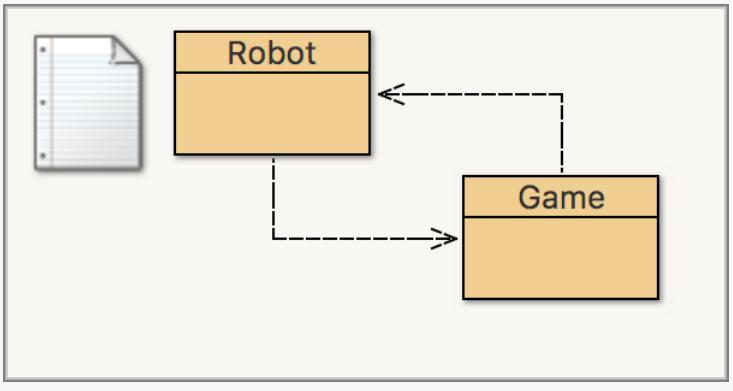
```
    public void sluitVatAan(Biervat vat)
    {
        this.vat = vat;
    }

    public void verwijderVat()
    {
        this.vat = null;
    }

    public void tapBiertje(Glas glas)
    {
        double gevraagd = glas.getVolume() - glas.getInhoud();

        glas.vulBij(vat.tapbier(gevraagd));
    }
```

Als uitbreiding zou je zelf de methode `void tapBiertje(Glas, double)` kunnen programmeren. Je geeft dan de hoeveelheid op waarmee je het bierglas wenst aan te vullen.

Oefening 5.17 – Robottikkertje Open bij de oefeningenbestanden van hoofdstuk 5 het project *Robottikker*. In deze oefening zullen we het spelletje tikkertjes tussen robots simuleren.

Om een tikspelletje te starten zullen we eerst een spel moeten aanmaken. Dit object is een instantie van de klasse `Game` en is in staat om robots aan te maken. In *BlueJ* zullen we via de klasse `Robot` geen robots aanmaken. Wanneer we minstens twee robots hebben, kunnen we de robots laten samenspelen. Het spel zal dan alleen nog bijhouden wie de tikker is.

Hieronder vind je het klassendiagram:



Jouw programmeeropdracht:

- `Game`
  - De klasse `Game` heeft slechts één veld: `tikker`. Dit veld verwijst naar de robot die de tikker is.
  - Voor het veld `tikker` programmeer je de mutator- en accessormethode.

- Robot
  - De klasse Robot heeft twee velden: de naam van de robot en een verwijzing naar het spel waar de robot deel van uitmaakt.
  - Voor het veld naam programmeer je de bijbehorende accessormethode.
  - De constructor geeft waarde aan de twee velden. Deze waarden worden via twee parameters geleverd. De header van de constructor is dus `Robot(String, Game)`.
  - Ten slotte programmeer je ook de methode `void tikRobot(Robot)`. Alleen een robot die tikker is, kan een andere robot tikken. De getikte robot laat dan het spel weten dat er een nieuwe tikker is.

Antwoord – Een correcte implementatie van de klasse Robot is:

```
1  public class Robot
2  {
3      private String naam;
4      private Game spelletje;
5
6      public Robot(String naam, Game spelletje)
7      {
8          this.naam = naam;
9          this.spelletje = spelletje;
10     }
11
12     public String getNaam()
13     {
14         return naam;
15     }
16
17     public void tikRobot(Robot robot)
18     {
19         if(spelletje.getTikker() == this)
20         {
21             spelletje.setTikker(robot);
22         }
23     }
24 }
```

- Game
  - Programmeer de methode `Robot nieuweRobot(String)`. Deze methode vraagt de naam van de nieuwe robot. Wanneer het spel de eerste deelnemende robot van het tikspelletje aanmaakt, is dit eerste object de tikker.
  - Ten slotte geeft de methode `String wieIsTikker()` de naam van de tikker terug.

Antwoord – Een correcte implementatie van de klasse `Game` is:

```
1  public class Game
2  {
3      private Robot tikker;
4
5      public Game()
6      {
7      }
8
9      public void setTikker(Robot robot)
10     {
11         tikker = robot;
12     }
13
14     public Robot getTikker()
15     {
16         return tikker;
17     }
18
19     public Robot nieuweRobot(String naam)
20     {
21         Robot robot = new Robot(naam,this);
22
23         if(tikker == null)
24         {
25             tikker = robot;
26         }
27
28         return robot;
29     }
30
31     public String wieIsTikker()
32     {
33         return tikker.getNaam();
34     }
35 }
```

Objecten groeperen in collecties 6

Oefening 6.1 – Lift Wanneer je het project *Lift* grondig getest hebt, kan je de volgende vragen zeker beantwoorden:

- Hoeveel personen kunnen in de lift?
- Wat gebeurt er wanneer meer dan 10 personen de lift wensen te gebruiken?
- Kan een persoon tweemaal in de lift stappen zonder tussendoor uit te stappen?
- Krijgt een persoon een volgnummer in de lift?
- Wat gebeurt er wanneer je een persoon verwijdert uit de lift en weer toevoegt. Waar in de lijst komt die persoon terecht?

Antwoord – De antwoorden vind je tussen de vragen:

- Hoeveel personen kunnen in de lift?
  Zolang het maximaal gewicht niet overschreden wordt, kunnen personen blijven instappen in de lift.
- Wat gebeurt er wanneer meer dan 10 personen de lift wensen te gebruiken?
  Zolang het maximaal gewicht niet overschreden wordt, is dit geen probleem. In BlueJ lijkt het alsof het aantal plaatsen beperkt is tot 10. Dit is slechts schijn.
- Kan een persoon tweemaal in de lift stappen zonder tussendoor uit te stappen?
  Ja, je kan een persoon zoveel keer laten instappen als je zelf wenst.
- Krijgt een persoon een volgnummer in de lift?
  Ja, elke persoon krijgt een volgnummer. Volgnummers starten bij 0.
- Wat gebeurt er wanneer je een persoon verwijdert uit de lift en weer toevoegt. Waar in de lijst komt die persoon terecht?
  Wanneer je een persoon midden in de lijst verwijderd, dan schuiven alle personen met een hoger volgnummer een plaats op naar voor. Voeg je de persoon terug toe, dan komt deze persoon zoals alle nieuwe personen achteraan in de lijst te staan.

Oefening 6.2 Welk type van objecten houden de volgende lijsten bij:

- In de klasse `CD`: `private ArrayList<MuziekNummer> tracks;`
- In de klasse `Factuur`: `private ArrayList<Product> verkocht;`
- In de klasse `Mailbox`: `private ArrayList<Mail> inbox;`

Antwoord – Dit is achtereenvolgens `MuziekNummer`, `Product` en `Mail`.

Oefening 6.3 Declareer en initialiseer telkens de `ArrayList`.

- De `Arraylist bibliotheek` met elementen van het objecttype `Boek`.
- De `Arraylist school` met elementen van het objecttype `Klas`.
- De `Arraylist activiteit` met elementen van het objecttype `Scoutslid`.

Antwoord – Het correct antwoord is:

```
    ArrayList<Boek> bibliotheek = new ArrayList<Boek>();
    ArrayList<Klas> school = new ArrayList<Klas>();
    ArrayList<Scoutslid> activiteit = new ArrayList<Scoutslid>();
```

Oefening 6.4 – Boodschappen Open bij de oefeningenbestanden van hoofdstuk 6 het project *Boodschappen*. In dit project vind je twee klassen terug:

- De klasse `Product` beschrijft de producten die je wenst te kopen in de winkel. Een instantie van de klasse `Product` heeft velden `naam` en `hoeveelheid` die je via de constructor een waarde geeft. Voor de velden beschikt de klasse alleen over accessormethoden.

- De klasse `Boodschappenlijstje` houdt het lijstje producten bij die op een boodschappenlijstje staan.

Programmeer in de klasse `Boodschappenlijstje` het veld `lijstje`. Dit veld is een instantie van de generieke klasse `ArrayList`. In de `ArrayList` verzamelen we objecten van het type `Product`.

Zorg ook voor een gepaste constructor. De constructor heeft geen parameters en initialiseert het veld `lijstje`.

Antwoord – Een correct antwoord is:

```
public class Boodschappenlijstje
{
    private ArrayList<Product> lijstje;

    public Boodschappenlijstje()
    {
        lijstje = new ArrayList<Product> ();
    }
}
```

Oefening 6.5 – Boodschappen In de klasse `Boodschappenlijstje` programmeer je volgende methoden:

- `void voegProductToe(Product)`
  Voegt een product toe aan het boodschappenlijstje `lijstje`.

  Antwoord – Een correct antwoord is:

  ```
      public void voegProductToe(Product product)
      {
          lijstje.add(product);
      }
  ```

- `void verwijderProduct(int)`
  Verwijdert een product op een gegeven index in boodschappenlijstje `lijstje`. Let op, controleer de gel-

digheid van de parameter! Wanneer de index ongeldig is, verwijder je geen enkel productobject uit `lijstje`.

Antwoord – Een correct antwoord is:

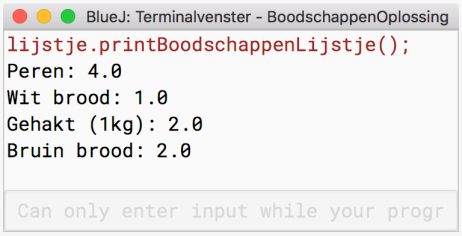
```
    public void verwijderProduct(int index)
    {
        if(index >= 0 && index < lijstje.size())
        {
            lijstje.remove(index);
        }
    }
```

- `Product geefProduct(int)`
  Geeft een product terug op een gegeven index. Let op, controleer de geldigheid van de parameter! Wanneer de index ongeldig is, geef je een `null`-verwijzing terug.

Antwoord – Een correct antwoord is:

```
    public Product geefProduct(int index)
    {
        Product product = null;

        if(index >= 0 && index < lijstje.size())
        {
            product = lijstje.get(index);
        }

        return product;
    }
```

Oefening 6.6 – Boodschappen In de klasse Boodschappen`lijstje` programmeer je de methode `void printBoodschappenLijstje()`. De methode print het boodschappenlijstje in het *terminalvenster* af, steeds de hoeveelheid voorafgegaan door het product.



Antwoord – Een correct antwoord is:

```
    public void printBoodschappenLijstje()
    {
        for(Product product: lijstje)
        {
            product.print();
        }
    }
```

Oefening 6.7 – Boodschappen In de klasse `Boodschappenlijstje` programmeer je de methode `void printBoodschappenMet(String)`. De methode print slechts een deel van het boodschappenlijstje af. Via de parameter geef je een of meer woorden of een deel van een woord mee dat in het boodschappenlijstje gezocht moet worden.

De `String`-methode `boolean contains(CharSequence)` vraagt een parameter van het objecttype `String` (`String` is een soort `CharSequence`) en gaat na of de opgegeven parameter een deel (of geheel) is van de `String` waarop je de methode toepast. Een voorbeeld uit het *evaluatievak*:

```
String cursus = "Objecten in Java";
cursus.contains("Java")
    true  (boolean)
cursus.contains("C#")
    false  (boolean)
```

De methode `void printBoodschappenMet(String)` met parameter "Brood" uitvoeren op het boodschappenlijstje uit oefening 6.6 geeft het volgende resultaat:

```
BlueJ: Terminalvenster - Boodschappen
lijstje.printBoodschappenMet("brood");
Wit brood: 1.0
Bruin brood: 2.0

Can only enter input while your progr
```

Antwoord – Een correct antwoord is:

```
1   public void printBoodschappenMet(String deel)
2   {
3       for(Product product: lijstje)
4       {
5           if(product.getNaam().contains(deel))
6           {
7               product.print();
8           }
9       }
10  }
```

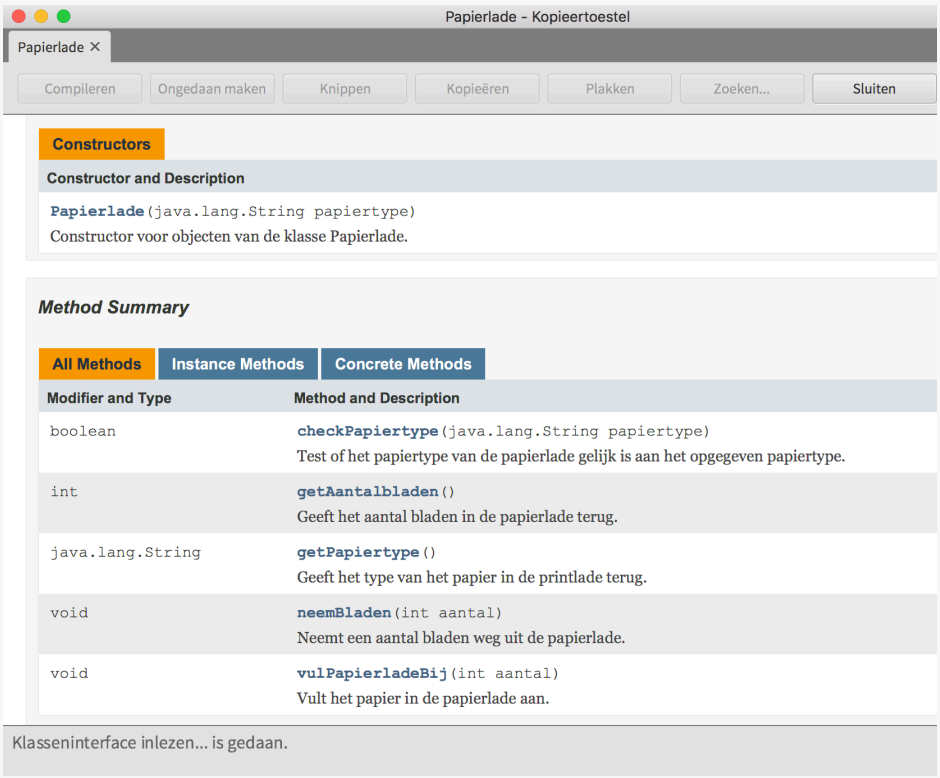
Oefening 6.8 – Boodschappen In de klasse `Boodschappenlijstje` programmeer je de methode `boolean staatInBoodschappenlijstje(String)`. De methode gaat na of de boodschap die je als parameter in meegeeft, reeds in het boodschappenlijstje staat. Indien dit het geval is, laat je `true` teruggeven. In het andere geval wordt `false` teruggegeven.

Antwoord – Een correct antwoord is:

```
1   public boolean staatInBoodschappenlijstje(String productnaam)
2   {
3       int index = 0;
4
5       while(index < lijstje.size() &&
6               ! lijstje.get(index).getNaam().equals(productnaam))
7       {
8           index++;
9       }
10
11      return index < lijstje.size();
12  }
```

Oefening 6.9 – Kopieertoestel Open het project *Kopieertoestel*. Je vindt het bij de oefeningenbestanden van hoofdstuk 6.

In deze oefening simuleren we de werking van een kopieertoestel waarmee je op verschillende papiertypes kan afdrukken. We geven een instantie van de klasse `Kopieertoestel` de mogelijkheid om verscheidene papierlades te beheren. In de echte wereld hebben kopieertoestellen de mogelijkheid om te kopiëren naar A3-formaat, A4-formaat enz. In deze oefening gaan we ervan uit dat elke papierlade een ander papiertype bevat. Er komen geen twee papierlades voor met hetzelfde papiertype. Je hoeft dit niet te programmeren, want je mag erop vertrouwen dat de gebruiker van je kopieertoestel dit weet (= slimme gebruiker). Wanneer je denkt lussen nodig te hebben, zorg je steeds voor de best passende lus in de gegeven context.



De klasse `Papierlade` krijg je cadeau. Hierboven vind je de API-pagina van `Papierlade`. Een instantie van deze klasse heeft twee velden:

- `papiertype`: het papiertype van de bladen in de papierlade,
- `aantalbladen`: het aantal bladen in de papierlade.

Zo kan de papierlade 64 A4-bladen bevatten of 27 A3-bladen enz. Let op, instanties van deze klasse kunnen een

onbeperkt aantal bladen bevatten. Bovendien kan het aantal bladen ook negatief worden.

Een instantie van de klasse `Kopieertoestel` heeft twee velden:

- `locatie`: de plaats van het kopieertoestel, bv. secretariaat of receptie,
- `lades`: verwijzingen naar instanties van de klasse `Papierlade`. Dit veld symboliseert de verschillende papierladen van een kopieertoestel.

Jouw programmeeropdracht:

- `void plugInNieuwePapierlade(String)`
  Deze methode maakt een nieuwe papierlade aan en voegt de papierlade toe aan het kopieertoestel. De parameter van deze methode maakt het mogelijk om het type papier te kiezen waarmee de papierlade aangevuld moet worden.

  Antwoord – Een correct antwoord is:

```
1     public void pluginNieuwePapierlade(String papiertype)
2     {
3         lades.add(new Papierlade(papiertype));
4     }
```

- `void plugInBestaandePapierlade (Papierlade)`
  Deze methode voegt de papierlade die je met de parameter meegeeft, toe aan het kopieertoestel.

  Antwoord – Een correct antwoord is:

```
1     public void pluginBestaandePapierlade(Papierlade lade)
2     {
3         lades.add(lade);
4     }
```

- `void verwijderPapierlade(int)`
  Deze methode verwijdert de papierlade in de `ArrayList` `lades` op de index die je via de parameter meegeeft. Let op, controleer of de papierlade bestaat!

  Antwoord – Een correct antwoord is:

```
1     public void verwijderPapierlade(int index)
2     {
3         if(index >= 0 && index < lades.size())
4         {
5             lades.remove(index);
6         }
7     }
```

- `void vulPapierlade(int, int)`
  Deze methode vult het papier bij van de papierlade op de index die met de eerste parameter wordt meegegeven. Het aantal bladen waarmee je de papierlade aanvult, vind je in de tweede parameter. Let op, controleer of de papierlade op de gevraagde index wel degelijk bestaat!

Antwoord – Een correct antwoord is:

```
    public void vulPapierlade(int index, int aantal)
    {
        if(index >= 0 && index < lades.size())
        {
            lades.get(index).vulPapierladeBij(aantal);
        }
    }
```

- `void vulPapierlade(String, int)`
  Deze methode vult het papier bij van de papierlade met het papiertype dat met de eerste parameter wordt meegegeven. Het aantal bladen waarmee je de papierlade aanvult, vind je in de tweede parameter.

Antwoord – Een correct antwoord is:

```
    public void vulPapierlade(String papiertype, int aantal)
    {
        int index = 0;

        while(index < lades.size() &&
                ! lades.get(index).checkPapiertype(papiertype))
        {
            index++;
        }

        if(index < lades.size())
        {
            lades.get(index).vulPapierladeBij(aantal);
        }
    }
```

- `void toonBeschikbarePapiertypes()`
  Deze methode toont in het terminalvenster het papiertype van de papierlades waar minstens één blad papier in zit. De papiertypes van de lege papierlades worden niet getoond!

Antwoord – Een correct antwoord is:

```
    public void toonBeschikbarePapiertypes()
    {
        for(Papierlade lade : lades)
        {
            System.out.println(lade.getPapiertype());
        }
    }
```

- `void kopieer(int, String)`
  De eerste parameter van de methode duidt aan hoeveel kopieën er genomen moeten worden. De tweede parameter geeft het papiertype mee. Je neemt alleen kopieën wanneer je een papierlade vindt met het vermelde papiertype. Bovendien moeten er genoeg bladen in de papierlade zitten. Wanneer je kan kopiëren, vergeet je het best niet het aantal bladen in de papierlade te verminderen met het aantal kopieën dat je wenst te nemen.

Antwoord – Een correct antwoord is:

```
1   public void kopieer(int aantal, String papiertype)
2   {
3       int index = 0;
4
5       while(index < lades.size() &&
6             ! lades.get(index).checkPapiertype(papiertype))
7       {
8             index++;
9       }
10
11      if(index < lades.size() && lades.get(index).getAantalBladen() <=
            aantal)
12      {
13            lades.get(index).neemBladen(aantal);
14      }
15  }
```

Oefening 6.10 – DHL Open het project *DHL*. Je vindt het bij de oefeningenbestanden van hoofdstuk 6.

In het project vind je twee klassen:

- `Pakje`: beschrijft een pakje met een zeker gewicht en een zekere bestemming.
- `Sorteermachine`: hier kan je pakjes aan toevoegen. De sorteermachine voert acties uit op de pakjes in de sorteermachine.

Hieronder vind je de API van de klasse `Pakje`:

De velden en constructor van de klasse `Sorteermachine` krijg je cadeau:

- `ArrayList<Pakje> pakjes`: een `Arraylist` die de pakjes bijhoudt die in de sorteermachine zitten.
- `double maxGewicht`: het maximale gewicht aan pakjes die je in de sorteermachine kan steken.
- `SorteerMachine(double)`: initialiseert de `ArrayList pakjes` en geeft de waarde van de parameter aan het veld `maxGewicht`.

Jouw programmeeropdracht:

- `void pakjeToevoegen(Pakje)`
  Voegt een pakje toe aan de `Arraylist pakjes`. Let op, je mag het maximale gewicht van de sorteermachine niet overschrijden!

  Antwoord – Een correct antwoord is:

```
    public void pakjeToevoegen(Pakje pakje)
    {
        pakjes.add(pakje);
    }
```

- `double berekenTotaalGewicht()`
  Berekent het totale gewicht van alle pakjes in de sorteermachine en geeft dit terug.

  Antwoord – Een correct antwoord is:

```
    public double berekenTotaalGewicht()
    {
        double totaalGewicht = 0.0;

        for(Pakje pakje : pakjes)
        {
            totaalGewicht += pakje.getGewicht();
        }

        return totaalGewicht;
    }
```

- `int aantalMetBestemming(int)`
  Berekent hoeveel pakjes in de sorteermachine bestemd zijn voor de postcode die je via de parameter meegeeft aan deze methode.

  Antwoord – Een correct antwoord is:

```
    public int aantalMetBestemming(int postcode)
    {
        int aantal = 0;

        for(Pakje pakje : pakjes)
        {
            if(pakje.getPostcode() == postcode)
            {
                aantal++;
            }
        }

        return aantal;
    }
```

- `boolean pakjeVoorGemeente(int)`
  Controleert of er een pakje bestemd is voor de gemeente waarvan je de postcode als parameter meegeeft aan de methode. Indien er minstens één pakje gevonden wordt voor de betreffende gemeente, dan wordt `true` teruggegeven, anders `false`.

Antwoord – Een correct antwoord is:

```
    public boolean pakjeVoorGemeente(int postcode)
    {
        int index = 0;

        while(index <= pakjes.size() && pakjes.get(index).getPostcode()
            != postcode)
        {
            index++;
        }

        return index <= pakjes.size();
    }
```

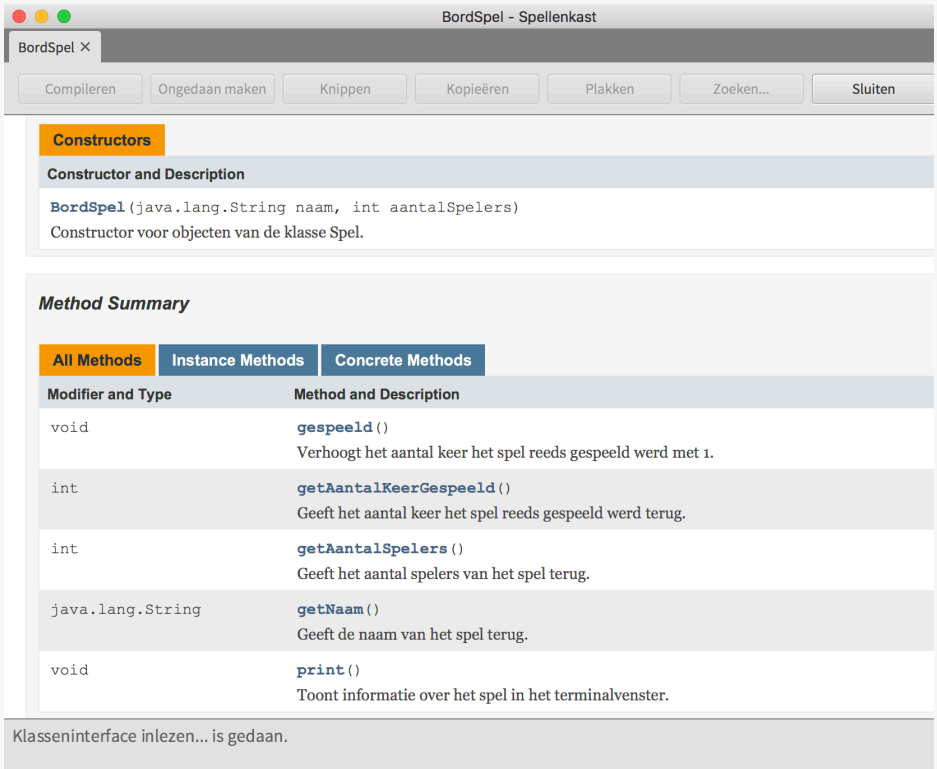
- `void verwijderPakje(int)`
  Verwijdert het pakje met gegeven index uit de `Arraylist` `pakjes`.

Antwoord – Een correct antwoord is:

```
    public void verwijderPakje(int index)
    {
        if(index >= 0 && index < pakjes.size())
        {
            pakjes.remove(index);
        }
    }
```

Oefening 6.11 – Spellenkast Open het project *Spellenkast*. Je vindt het bij de oefeningenbestanden van hoofdstuk 6.

Wie veel bordspelen bezit, moet op zoek naar een kast om ze in te verzamelen. In deze oefening zullen we objecten van de klasse `Bordspel` verzamelen in een instantie van de klasse `Spellenkast`.



Je krijgt de API van de klasse `Bordspel` cadeau. Bestudeer grondig de werking van een instantie van de klasse `Bordspel` alvorens je de klasse `Spellenkast` programmeert.

Van elk bordspel houden we drie eigenschappen bij:

- de naam van het spel,
- het aantal spelers dat kan deelnemen.
- het aantal keer dat het spel al gespeeld werd (belangrijk voor de echte bordspel-nerd).

In de klasse `Spellenkast` krijg je het enige veld én de constructor cadeau. Programmeer de volgende methoden in de klasse `Spellenkast`:

- `void voegSpelToe(Bordspel)`
  Voegt een bestaand bordspel toe aan de spellenkast.

  Antwoord – Een correct antwoord is:

```
1     public void voegSpelToe(BordSpel spel)
2     {
3         kast.add(spel);
4     }
```

- `void spelGespeeld(int)`
  Wanneer een spel uit de spellenkast gespeeld werd, moet je het aantal keer dat het spel gespeeld werd, verhogen met 1. De index van het spel in de spellenkast geef je mee als parameter.

  Antwoord – Een correct antwoord is:

```
1     public void spelGespeeld(int index)
2     {
3         if(index >= 0 && index < kast.size())
4         {
5             kast.get(index).gespeeld();
6         }
7     }
```

- `int spellenVoorAantalSpelers(int)`
  Geeft voor een gegeven aantal spelers het aantal spellen in de spellenkast terug.

  Antwoord – Een correct antwoord is:

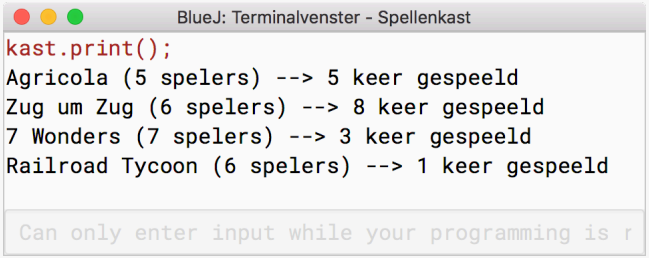
```
1     public int spellenVoorAantalSpelers(int aantalSpelers)
2     {
3         int aantal = 0;
4
5         for(BordSpel spel : kast)
6         {
7             if(spel.getAantalSpelers() == aantalSpelers)
8             {
9                 aantal++;
10            }
11        }
12
13        return aantal;
14    }
```

- `Bordspel geefSuggestie(int)`
  We zijn met zes spelers. Welk spel zouden we spelen? Deze methode suggereert het eerste spel in de spellenkast voor het gevraagde aantal spelers in de parameter. In dit geval *Zug um Zug*. Let op, geef niet de naam maar het volledige bordspel terug! Nemen we de spellenkast uit de afbeelding onderaan als voorbeeld:
  - `kast.geefSuggestie(5)` geeft een verwijzing naar het bordspel *Agricola* terug.
  - `kast.geefSugggestie(3)` geeft `null` terug.

Antwoord – Een correct antwoord is:

```
    public BordSpel geefSuggestie(int aantalSpelers)
    {
        int index = 0;
        BordSpel suggestie = null;

        while(index < kast.size() && kast.get(index).getAantalSpelers()
            != aantalSpelers)
        {
            index++;
        }

        if(index < kast.size())
        {
            suggestie = kast.get(index);
        }

        return suggestie;
    }
```

- `void print()`
  Toont informatie in het terminalvenster over alle spellen in de spellenkast. Toon de spellen zoals in de afbeelding hiernaast.



Antwoord – Een correct antwoord is:

```
    public void print()
    {
        for(BordSpel spel : kast)
        {
            spel.print();
        }
    }
```

Oefening 6.12 – Verloren voorwerpen Open het project *VerlorenVoorwerpen*. Je vindt het bij de oefeningenbestanden van hoofdstuk 6.

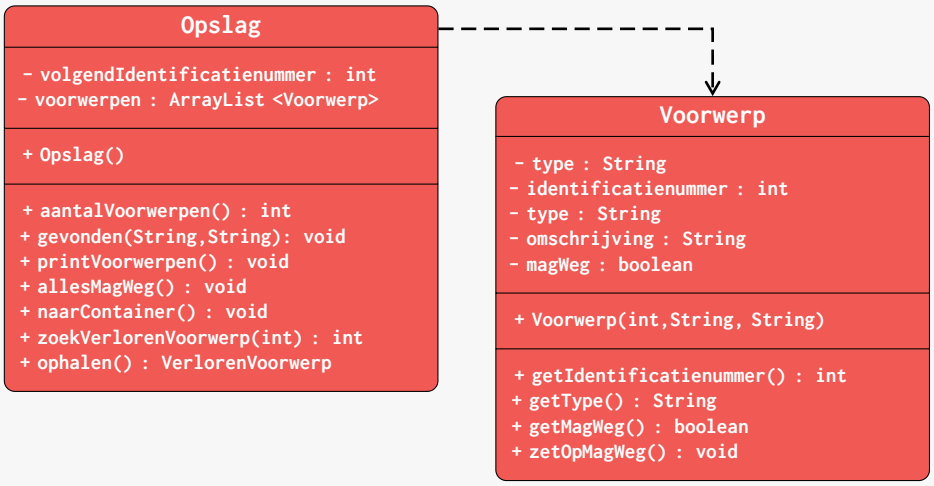
Dagelijks komen in een school heel wat verloren voorwerpen binnen: kledij, USB-sticks, schooletuis, ... je kan het zo gek niet bedenken.

Om een beetje organisatie te krijgen in het beheer van de verloren voorwerpen gaat men elk binnengebracht stuk labelen. Een memoblaadje met een uniek nummer is hier de truc. In de opslagruimte met de verloren voorwerpen vindt men bijvoorbeeld volgende vier memo-blaadjes vastgeprikt op verloren stukken terug:

- 501: Kledij - Superman T-shirt.
- 502: Etui - Kipling groen.
- 505: Bril - Blauwe montuur.
- 506: USB-stick - Zwarte Cruzer 8GB.

Het ideale is natuurlijk dat een leerling zijn/haar eigendom weer komt ophalen. Het betreffende memoblaadje verdwijnt dan richting papiermand. Omdat de opslagruimte beperkt is, komen voorwerpen soms op een 'mag weg' stapel terecht. Wekelijks komt de schoonmaakploeg eens voorbij om deze voorwerpen naar de container te brengen. Zolang de schoonmaakploeg niet langsgekomen is, kan de leerling nog 'op het nippertje' zijn verloren voorwerp komen redden.

Ook hier weer het klassieke klassendiagram:



De klasse `Voorwerp` (die we voor de eenvoud niet `VerlorenVoorwerp` noemen) werd al helemaal geprogrammeerd.

- Velden van de klasse `Voorwerp`:
  - `int identificatienummer`: het unieke nummertje dat men ook op het memoblaadje schrijft.
  - `String type`: stelt het type verloren voorwerp voor. Voorbeelden zijn *kledij*, *USB-stick*, *etui* enz.
  - `String omschrijving`: voorbeelden zijn *Superman T-shirt*, *Zwarte Cruzer 8 GB* enz.
  - `boolean magWeg`: Staat altijd op `false` tenzij beslist werd dat het verloren voorwerp niet meer bijgehouden zal worden. Pas als de schoonmaakploeg het kwam ophalen, is het voorwerp definitief verloren.

- De constructor `Voorwerp(int,String,String)` vraagt een identificatienummer, type en omschrijving.

- De methode `void zetOpMagWeg()` zet het veld `magWag` op `true`, klaar om door de schoonmaakploeg naar de container gebracht te worden.

De klasse `Opslag` dient om de verloren voorwerpen te beheren en heeft twee velden:

- `ArrayList voorwerpen`: houdt alle verloren voorwerpen die binnengebracht werden bij.
- `int volgendIdentificatienummer`: elk verloren voorwerp moet een unieke identificatienummer krijgen. Het veld `volgendIdenticatienummer` bevat het nummertje dat we aan het volgende gevonden voorwerp zullen toekennen. Telkens wanneer een nieuw voorwerp binnengebracht wordt, moet `volgendIdentificatienummer` met 1 verhogen.

In de constructor `Opslag()` laten we de `ArrayList voorwerpen` initialiseren. Ook geven we er het veld `volgendIdentificatienummer` de startwaarde `0`.

Programmeer in de klasse `Opslag` de volgende methoden:

- `boolean zijnErVerlorenVoorwerpen()`
  Deze methode gaat na of er verloren voorwerpen zijn. Deze methode geeft `false` terug wanneer er geen verloren voorwerpen zijn. In het andere geval wordt `true` teruggegeven.

  Antwoord – Een correct antwoord is:

```
1     public boolean zijnErVerlorenVoorwerpen()
2     {
3         return verlorenVoorwerpen.size() > 0;
4     }
```

- `void voorwerpToevoegen(String,String)`
  Deze methode dient om een verloren voorwerp toe te voegen aan onze `ArrayList` voorwerpen. Je krijgt het type en de omschrijving van het verloren voorwerp mee via de parameters. Je moet een instantie van de klasse `Voorwerp` aanmaken waarbij je rekening houdt met het veld `volgendIdenficatienummer` om het voorwerp een uniek identificatienummer te geven. Vergeet niet het veld `volgendIdentificatienummer` met 1 te verhogen, zodat een volgende verloren voorwerp onmiddellijk een correct identificatienummer krijgt.

  Antwoord – Een correct antwoord is:

```
1     public void voorwerpToevoegen(String type, String omschrijving)
2     {
3         verlorenVoorwerpen.add(new VerlorenVoorwerp(
4                     tellerIdentificatienummer,type, omschrijving));
5         tellerIdentificatienummer ++;
6     }
```

- `void printVoorwerpen(String type)`
  Bij deze methode geef je als parameter het type mee waarvan je de verloren voorwerpen wil zien. Je toont in het *terminalvenster* de identificatiecode en de omschrijving van alle gevonden producten van het gegeven type.

  Antwoord – Een correct antwoord is:

```
1     public void printVoorwerpen(String type)
2     {
3         System.out.println(type);
4
5         for(VerlorenVoorwerp v: verlorenVoorwerpen)
6         {
7             if (v.getType().equals(type))
8             {
9                 System.out.println(v.getIdentificatienummer() +
10                                    ": " + v.getOmschrijving());
11            }
12        }
13    }
```

- `void allesMagWeg()`
  Tijdens de zomervakantie wordt er grote schoonmaak gehouden. Alles wat niet opgehaald werd door een leerling mag dan weg. Laat daarom met deze methode alle voorwerpen in voorwerpen op *mag weg* zetten. Je hoeft niets te verwijderen uit de `ArrayList` voorwerpen!

  Antwoord – Een correct antwoord is:

```
    public void allesMagWeg()
    {
        for(VerlorenVoorwerp v: verlorenVoorwerpen)
        {
            v.zetOpMagWeg();
        }
    }
```

- `void naarContainer()`
  Wanneer je deze methode uitvoert, verwijdert het onderhoudsteam alle voorwerpen die op gelabeld zijn met *mag weg*. Het lokaal van de verloren voorwerpen is weer leeg. Voor de eenvoud mag je ervan uitgaan dat bij het uitvoeren van deze methode, alle verloren voorwerpen gelabeld zijn met *mag weg*.

  Antwoord – Een correct antwoord is:

```
public void naarContainer()
{
    for (int index = verlorenVoorwerpen.size() - 1; index >= 0; index--)
    {
        if (verlorenVoorwerpen.get(index).getMagWeg())
        {
            verlorenVoorwerpen.remove(index);
        }
    }
}
```

- `boolean zoekVerlorenVoorwerp(int)`
  Deze methode zoekt naar een voorwerp met het identificatienummer dat je via de parameter meegeeft. Indien een voorwerp met het betreffende identificatienummer gevonden werd, laat je `true` teruggeven. In het andere geval is de retourwaarde `false`.

  Antwoord – Een correct antwoord is:

```
    public int zoekVerlorenVoorwerp(int identificatienummer)
    {
        int index = 0;

        while (index < verlorenVoorwerpen.size() && verlorenVoorwerpen.
            get(index).getIdentificatienummer() != identificatienummer)
        {
            index++;
        }

        if(index == verlorenVoorwerpen.size())
        {
            index = -1;
        }

        return index;
    }
```

- `Voorwerp ophalen(int)`
  Bij deze methode geef je het identificatienummer van het op te halen voorwerp mee als parameter. Vind je het verloren voorwerp dan verwijder je het uit de `ArrayList` voorwerpen. De methode geeft het gevonden verloren voorwerp terug ofwel `null` indien er geen voorwerp met het gegeven identificatienummer gevonden werd.

Antwoord – Een correct antwoord is:

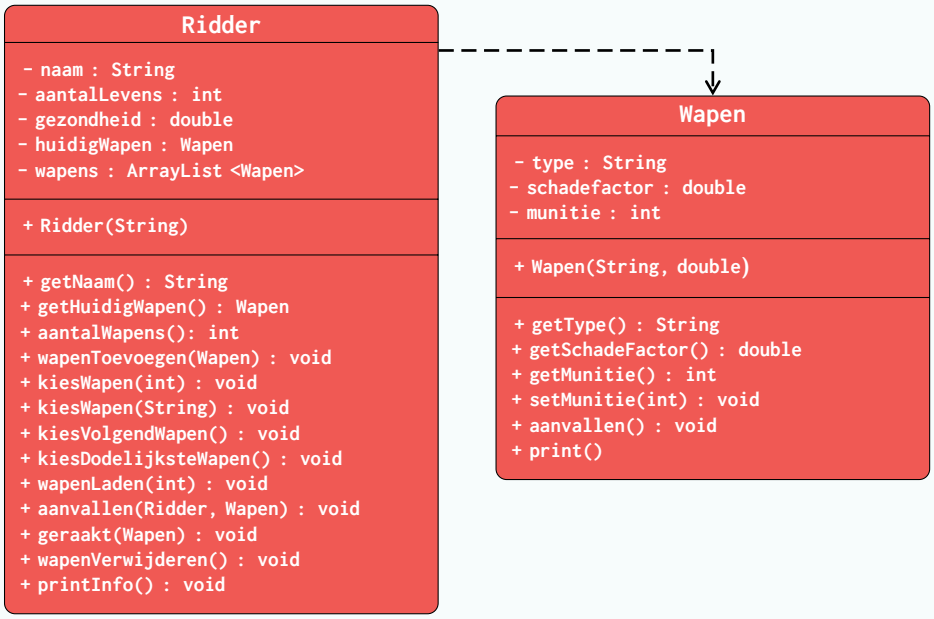
```
1     public VerlorenVoorwerp ophalen(int identificatienummer)
2     {
3         int index = zoekVerlorenVoorwerp(identificatienummer);
4         VerlorenVoorwerp verlorenVoorwerp = null;
5
6         if (index < verlorenVoorwerpen.size())
7         {
8             verlorenVoorwerp = verlorenVoorwerpen.get(index);
9             verlorenVoorwerpen.remove(index);
10        }
11
12        return verlorenVoorwerp;
13    }
```

(Naar een oefening van Goderik Lefebvre, Waregem)

Oefening 6.13 – Ridder Herinner je je het project *Ridder* nog uit hoofdstuk 5? Vergeet de ridders en harnassen en doe de volgende denkoefening.

Bedenk een klassendiagram dat het ridders mogelijk maakt om wapens te verzamelen. Ridders moeten vlot kunnen wisselen van wapen, bv. van zwaard naar pijl en boog. Misschien is het leuk te weten hoe dodelijk een wapen is. Zoekfuncties in je wapenarsenaal heb je dus zeker nodig!
Denk je aan ridders dan denk je aan een duel. Zorg ervoor dat ridders kunnen vechten met elkaar. De lichamelijke schade die een ridder oploopt is afhankelijk van het wapen waarmee de ridder geraakt werd.
Ridders zijn maar mensen en kunnen dus ook sterven. Een dode ridder kan nooit meer vechten, wapens verzamelen of wisselen van wapen.

Antwoord – Een correct antwoord is:



Wie wil kan het klassendiagram implementeren in *BlueJ*.

Overerving en abstractie 7

Oefening 7.1 – EenvoudigGevecht Wanneer je het project *EenvoudigGevecht* grondig getest hebt, kan je de volgende vragen zeker beantwoorden:

- Hoe evolueert de gezondheidstoestand van de indiaan wanneer de indiaan geraakt werd door een schot van de cowboy? En omgekeerd?
- Welke methoden van welke klassen worden uitgevoerd tijdens het schieten?
- Som de verschillen op in de klassendefinitie van beide klassen. Vind je dit veel of weinig?
- Wat zijn de vrijetijdsbestedingen van objecten van beide klassen? Welke invloed hebben ze op hun gezondheidstoestand?

Antwoord – De antwoorden vind je tussen de vragen:

- Hoe evolueert de gezondheidstoestand van de indiaan wanneer de indiaan geraakt werd door een schot van de cowboy? En omgekeerd?
  Wanneer de indiaan geraakt wordt, vermindert met zijn gezondheid met 60%. Bij de cowboy is dit slechts 40%.

- Welke methoden van welke klassen worden uitgevoerd tijdens het schieten?
  De cowboy gebruikt `schiet(Indiaan)`. *De indiaan gebruikt* `schiet(Cowboy)`

- Som de verschillen op in de klassendefinitie van beide klassen. Vind je dit veel of weinig?
  De indiaan heeft twee levens meer. De berekening van de evolutie van de gezondheidstoestand is bijna gelijk. De indiaan en de cowboy hebben een verschillende hobby. Al bij al zijn de verschillen niet zo groot.

- Wat zijn de vrijetijdsbestedingen van objecten van beide klassen? Welke invloed hebben ze op hun gezondheidstoestand?
  De indiaan rookt graag een vredespijp met een cowboy waardoor de indiaan een leven bij krijgt. De cowboy drinkt graag whisky, maar dat komt zijn gezondheidstoestand niet ten goede.

Oefening 7.2 – Strijder Wanneer je de klasse `Strijder` grondig getest hebt, kan je de volgende vragen zeker beantwoorden:

- Kan je een aantal strijders laten vechten met elkaar?
- Hoe evolueert de gezondheidstoestand van elke strijder? Vergelijk dit met de evolutie van de gezondheidstoestand van objecten van de klassen `Cowboy` en `Indiaan`.
- Hoe wordt informatie getoond in het *Terminalvenster*? Is er een verschil?

Antwoord – De antwoorden vind je tussen de vragen:

- Kan je een aantal strijders laten vechten met elkaar?
  Ja, je gebruikt hiervoor de methode `schiet(Strijder)`.
- Hoe evolueert de gezondheidstoestand van elke strijder? Vergelijk dit met de evolutie van de gezondheidstoestand van objecten van de klassen `Cowboy` en `Indiaan`.
  Er werd gekozen om de gezondheid van een strijder te laten evolueren zoals bij de cowboy
- Hoe wordt informatie getoond in het *Terminalvenster*? Is er een verschil?
  Alle strijders worden op een gelijkaardige manier getoond in het terminalvenster.

Oefening 7.3 – AllemaalStrijders Wanneer je het project *AllemaalStrijders* grondig getest hebt, kan je de volgende vragen zeker beantwoorden:

- Kan je met objecten van de klasse `Strijder` ook schieten naar een object van de klasse `Cowboy`?
- Wanneer je instanties van de klasse `Cowboy` of `Indiaan` aanmaakt, waar in *BlueJ* vind je de methode `schiet (Strijder)`?
- Kan je via het `Evaluatievak` een object van de klasse `Cowboy` laten vechten met een object van de klasse `Indiaan`?

Antwoord – De antwoorden vind je tussen de vragen:

- Kan je met objecten van de klasse `Strijder` ook schieten naar een object van de klasse `Cowboy`?
  Met een strijder kan je schieten naar een strijder, cowboy of indiaan.
- Wanneer je instanties van de klasse `Cowboy` of `Indiaan` aanmaakt, waar in *BlueJ* vind je de methode `schiet (Strijder)`?
  Niet in de klassendefinitie van `Cowboy` *of* `Indiaan`. *De enige methode* `schiet(Strijder)` *vind je in de klasse* `Strijder`.
- Kan je via het `Evaluatievak` een object van de klasse `Cowboy` laten vechten met een object van de klasse `Indiaan`?
  Ja, dat kan. Hier vind je vier statements die dit aantonen.

```
Cowboy henk = new Cowboy("Henk");
Indiaan winnetoe = new Indiaan("Winnetoe");
henk.schiet(winnetoe);
winnetoe.schiet(henk);
```

Oefening 7.4 – Goudzoeker In deze oefening werk je met het project *Goudzoeker*. Je vindt dit project bij de oefeningenbestanden van hoofdstuk 7.

- Maak de klasse `Goudzoeker` aan. `Goudzoeker` is een uitbreiding van `Strijder`.

Antwoord – Een correct antwoord is:

```
1 public class Goudzoeker extends Strijder
2 {
3 }
```

- Welke foutmelding krijg je wanneer je de klasse zonder inhoud compileert?

  Antwoord – Je ziet volgende foutmelding verschijnen:

  ```
  constructor Strijder in class Strijder cannot be applied to given
  types;
    required: java.lang.String
    found: no arguments
    reason: actual and formal argument lists differ in length
  ```

- Voeg het veld `aantalGoudstukken` toe. Dit veld houdt een geheel aantal goudstukken bij die door de goudzoeker gevonden werden in de rivier.

  Antwoord – Een correct antwoord is:

  ```
  1     private int aantalGoudstukken;
  ```

- Programmeer een constructor die het veld `aantalGoudstukken` initialiseert met behulp van een parameter.

  Antwoord – Een correct antwoord is:

  ```
  1     public Goudzoeker(String naam, int aantalGoudstukken)
  2     {
  3         super(naam);
  4         this.aantalGoudstukken = aantalGoudstukken;
  5     }
  ```

- Probeer ook de overgeërfde velden te initialiseren met behulp van de constructor. Lukt dit?

  Antwoord – Een correct antwoord is:

  ```
  1     public Goudzoeker(String naam, int aantalGoudstukken)
  2     {
  3         super(naam);
  4         this.aantalGoudstukken = aantalGoudstukken;
  5         aantalLevens = 5;
  6         gezondheid = 100;
  7     }
  ```

  Merk op dat je eerst via de super-aanroep de overgeërfde velden initialiseert en dat je dan via de constructor van `Goudzoeker` de velden `aantalLevens` en `gezondheid` een nieuwe waarde.

Oefening 7.5 – HR versie 1 Open het project *HR-v1*. Je vindt het bij de oefeningenbestanden van hoofdstuk 7.

Bestudeer de klasse `Werknemer`. Aan deze klasse voeg je vier velden toe:

- `ancienniteit`: houdt het aantal jaar dat een werknemer in dienst is bij,
- `maandloon`: houdt het maandloon van een werknemer bij,
- `naam`: houdt de naam van de werknemer bij,
- `aantalVerlofDagen`: houdt het aantal verlofdagen van de werknemer bij.

Alleen het veld `ancienniteit` moet bereikbaar zijn in een subklasse.

Antwoord – Een correct antwoord is:

```
public class Werknemer
{
    private String naam;
    private int aantalVerlofDagen;
    protected int ancienniteit;
    private double maandloon;
}
```

Oefening 7.6 – HR versie 2 Open het project *HR-v2*. Je vindt het bij de oefeningenbestanden van hoofdstuk 7.

Maak de klasse `ProjectLeider` aan. Deze klasse is een uitbreiding van de klasse `Werknemer`. In de klasse `ProjectLeider` programmeer je de volgende zaken:

- Het veld `aantalTeamLeden`: houdt het aantal leden bij van het softwareteam waarover een teamleider de verantwoordelijkheid draagt.

  Antwoord – Een correct antwoord is:

  ```
  public class ProjectLeider extends Werknemer
  {
      private int aantalTeamLeden;
  }
  ```

- Een mutatormethode voor het veld `aantalTeamLeden`.

  Antwoord – Een correct antwoord is:

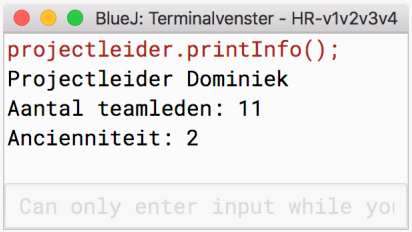
  ```
      public void setAantalTeamLeden(int aantalTeamLeden)
      {
          this.aantalTeamLeden = aantalTeamLeden;
      }
  ```

- De constructor `ProjectLeider`. De constructor vraagt de naam van de projectleider en het aantal leden van het softwareteam. Maak zeker gebruik van de constructor van de superklasse om de overgeërfde velden te initialiseren.

  Antwoord – Een correct antwoord is:

  ```
      public ProjectLeider(String naam, int aantalTeamLeden)
      {
          super(naam);
          this.aantalTeamLeden = aantalTeamLeden;
      }
  ```

Oefening 7.7 – HR versie 3 Open het project *HR-v3*. Je vindt het bij de oefeningenbestanden van hoofdstuk 7.



Programmeer de methode `void printInfo()` in de klasse `ProjectLeider`. Deze methode toont in het *Terminalvenster* informatie over een projectleider.

Waar het kan, roep je een methode van de superklasse op.

Antwoord – Een correct antwoord is:

```
1     public void printInfo()
2     {
3         System.out.println("Projectleider " + getNaam());
4         System.out.println("Aantal teamleden: " + aantalTeamLeden);
5         System.out.println("Ancienniteit: " + ancienniteit);
6     }
```

Oefening 7.8 – HR versie 4 Open het project *HR-v4*. Je vindt het bij de oefeningenbestanden van hoofdstuk 7.

In de klasse `ProjectLeider` overschrijf je volgende methoden:

- `int berekenAantalVerlofDagen()`
  Een ProjectLeider krijgt bijkomend drie verlofdagen.

  Antwoord – Een correct antwoord is:

  ```
  1     public int berekenAantalVerlofDagen()
  2     {
  3         return super.berekenAantalVerlofDagen() + 3;
  4     }
  ```

- `double berekenMaandLoon`
  De projectleider krijgt per teamlid €100 bovenop het maandloon van een werknemer.

  Antwoord – Een correct antwoord is:

  ```
  1     public double berekenMaandloon()
  2     {
  3         return super.berekenMaandloon() + (100 * aantalTeamLeden);
  4     }
  ```

Let op, het is niet omdat je een methode van de superklasse overschrijft, dat je de oorspronkelijke methode niet meer kan gebruiken!

Oefening 7.9 – HR versie 5 Open het project *HR-v5*. Je vindt het bij de oefeningenbestanden van hoofdstuk 7.

Naast de klasse `ProjectLeider` is ook de klasse `SoftwareOntwikkelaar` een uitbreiding van de klasse `Werkne-`

mer. Bestudeer de werking van beide klassen. Let op, de methode `void printInfo()` werd ook aan de superklasse toegevoegd.

De klasse `PersoneelsBeheer` beheert de personeelsdossiers van een fictief bedrijf. Hiervoor werd een `ArrayList` gebruikt van het type `Werknemer`. De constructor van de klasse krijg je cadeau.

Programmeer in de klasse `PersoneelsBeheer` de methode met naam `addWerknemer`. Deze methode voegt een werknemer toe aan de `ArrayList` `werknemers`. Let op, je moet behalve werknemers ook projectleiders en softwareontwikkelaars kunnen toevoegen!

Antwoord – Een correct antwoord is:

```
1    public void addWerknemer(Werknemer werknemer)
2    {
3        werknemers.add(werknemer);
4    }
```

Oefening 7.10 – HR versie 6 Open het project *HR-v6*. Je vindt het bij de oefeningenbestanden van hoofdstuk 7. De klasse `PersoneelsBeheer` beheert de personeelsdossiers van een fictief bedrijf. Hiervoor werd een `ArrayList` gebruikt van het type `Werknemer`.

Programmeer in de klasse `PersoneelsBeheer` de methode `void printInfo(int)` die informatie toont van een werknemer op een gegeven index. Controleer of de gegeven index geldig is binnen de `ArrayList`.

Antwoord – Een correct antwoord is:

```
1    public void printInfo(int index)
2    {
3        if(index >= 0 && index < werknemers.size())
4        {
5            werknemers.get(index).printInfo();
6        }
7    }
```

Kan je een goed antwoord verzinnen op de vraag waarom we in de superklasse `Werknemer` de methode `void printInfo()` toegevoegd hebben?

Antwoord – Wanneer een subklasse de methode `printInfo()` niet overschrijft, zijn we toch zeker dat er informatie getoond wordt in het *Terminalvenster*. In het geval er geen methode `printInfo()` ter beschikking zou zijn in de superklasse en de subklasse, dan krijgen we een foutmelding.

Oefening 7.11 – HR versie 7 Open het project *HR-v7*. Je vindt het bij de oefeningenbestanden van hoofdstuk 7.

De klasse `PersoneelsBeheer` beheert de personeelsdossiers van een fictief bedrijf. Hiervoor werd een `ArrayList` gebruikt van het type `Werknemer`.

Programmeer in de klasse `PersoneelsBeheer` de methode `double berekenLoonMassa()` die de totale loonmassa van alle werknemers teruggeeft.

Antwoord – Een correct antwoord is:

```
    public double berekenLoonMassa()
    {
        double loonmassa = 0;

        for(Werknemer w: werknemers)
        {
            loonmassa += w.berekenMaandloon();
        }

        return loonmassa;
    }
```

Oefening 7.12 – Post versie 1 Open het project *Post-v1*. Je vindt het bij de oefeningenbestanden van hoofdstuk 7.

De klasse `Brief` beschrijft op een heel eenvoudige manier een brief die je per post wenst te versturen. De brief is zo eenvoudig dat we alleen het postnummer van de geadresseerde bijhouden. Het postnummer kan je opvragen met de accessormethode `String getPostnummer()`.

De klasse `Postvak` stelt een postvakje voor in het sorteercentrum waar elke ochtend de brieven gesorteerd worden per postnummer. De klasse heeft twee velden. Het veld `postnummer` duidt aan voor welke gemeente het postvak brieven bevat. Het veld `brief` wijst naar de enige brief die het postvak kan bevatten.

In de klasse `Postvak` vind je ook twee methoden:

- `boolean correctPostnummer(Brief)`: controleert of een gegeven brief voor het postvak bestemd is,
- `void voegBriefToe(Brief)`: voegt een brief toe aan het postvak.

Gevraagd:

- Maak van de klasse `Postvak` een abstracte klasse en maak de twee methoden die hiervoor werden beschreven abstract.

Antwoord – Een correct antwoord is:

```
public abstract class Postvak
{
    protected String postnummer;
    protected Brief brief;

    public Postvak(String postnummer)
    {
        this.postnummer = postnummer;
    }

    public abstract boolean correctPostnummer(Brief brief);

    public abstract void voegBriefToe(Brief brief);
}
```

- Programmeer de subklasse EenvoudigPostvak. Deze klasse neemt het werk van de oorspronkelijke klasse Postvak over.

Antwoord – Een correct antwoord is:

```
public class EenvoudigPostvak extends Postvak
{
    public EenvoudigPostvak(String postnummer)
    {
        super(postnummer);
    }

    public void voegBriefToe(Brief brief)
    {
        if(correctPostnummer(brief))
        {
            this.brief = brief;
        }
    }

    public boolean correctPostnummer(Brief brief)
    {
        return postnummer == brief.getPostnummer();
    }
}
```

- Programmeer de subklasse GrootPostvak. Een instantie van deze klasse moet een aantal brieven bestemd voor hetzelfde postnummer kunnen bevatten.

Antwoord – Een correct antwoord is:

```
import java.util.ArrayList;

public class GrootPostvak extends Postvak
{
    private ArrayList<Brief> brieven;

    public GrootPostvak(String postnummer)
    {
        super(postnummer);
        brieven = new ArrayList<Brief>();
    }

    public void voegBriefToe(Brief brief)
    {
        if(correctPostnummer(brief))
        {
            brieven.add(brief);
        }
    }

    public boolean correctPostnummer(Brief brief)
    {
        return postnummer == brief.getPostnummer();
    }
}
```

Oefening 7.13 – Post versie 2 Open het project *Post-v2*. Je vindt het bij de oefeningenbestanden van hoofdstuk 7. Dit project bevat exact dezelfde klassen als het project *Post-v1*. Voor uitleg bij de klassen in dit project, verwijzen we naar oefening 7.12.
Gevraagd:

- Postvak
  Maak van de klasse Postvak de interface Postvak met de twee methoden beschreven in oefening 7.12 op pagina 105.

  Antwoord – Een correct antwoord is:

```
public interface Postvak
{
    public boolean correctPostnummer(Brief brief);
    public void voegBriefToe(Brief brief);
}
```

- EenvoudigPostvak
  Programmeer de klasse EenvoudigPostvak. Deze klasse implementeert de interface Postvak en neemt het werk van de oorspronkelijke klasse Postvak over.

  Antwoord – Een correct antwoord is:

```
public class EenvoudigPostvak implements Postvak
{
    private String postnummer;
    private Brief brief;

    public EenvoudigPostvak(String postnummer)
    {
        this.postnummer = postnummer;
        brief = null;
    }

    public void voegBriefToe(Brief brief)
    {
        if(correctPostnummer(brief))
        {
            this.brief = brief;
        }
    }

    public boolean correctPostnummer(Brief brief)
    {
        return postnummer == brief.getPostnummer();
    }
}
```

- GrootPostvak
  Programmeer de klasse GrootPostvak. Ook deze klasse implementeert de interface Postvak. Een instantie van deze klasse moet een aantal brieven bestemd voor hetzelfde postnummer kunnen bevatten.

Antwoord – Een correct antwoord is:

```
import java.util.ArrayList;

public class GrootPostvak implements Postvak
{
    private String postnummer;
    private ArrayList<Brief> brieven;

    public GrootPostvak(String postnummer)
    {
        this.postnummer = postnummer;
        brieven = new ArrayList<Brief>();
    }

    public void voegBriefToe(Brief brief)
    {
        if(correctPostnummer(brief))
        {
            brieven.add(brief);
        }
    }

    public boolean correctPostnummer(Brief brief)
    {
        return postnummer == brief.getPostnummer();
    }
}
```

- SorteerCentrum
  Programmeer de klasse SorteerCentrum. Deze klasse bevat een ArrayList met alle soorten postvakken. Zorg dat je een brief kan geven aan het sorteercentrum en dat deze brief in het juiste postvak terechtkomt.

Antwoord – Een correct antwoord is:

```
import java.util.ArrayList;

public class SorteerCentrum
{
    private ArrayList<Postvak> postvakken;

    public SorteerCentrum()
    {
        postvakken = new ArrayList<Postvak>();
    }

    public void postvakToevoegen(Postvak postvak)
    {
        postvakken.add(postvak);
    }

    public void sorteer(Brief brief)
        {
        int index = 0;

        while(index < postvakken.size() &&
                ! postvakken.get(index).correctPostnummer(brief))
        {
            index++;
        }

        if(index <= postvakken.size())
        {
            postvakken.get(index).voegBriefToe(brief);
        }
    }
}
```

Er werd geen rekening gehouden met postvakken met hetzelfde postnummer.

Oefening 7.14 – Overerving Gegeven zijn de volgende klassendefinities:

```
public class A
{
   public void print()
   {
      System.out.println("A");
   }
}

public class B extends A
{

}

public class C extends B
{
   public void print()
```

```
    {
        super.print();
        System.out.println("C");
    }
}
```

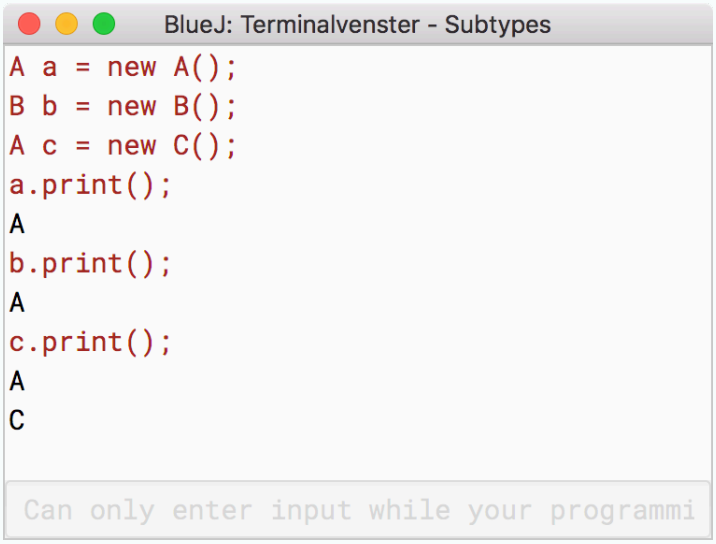
Stel dat we drie objecten aanmaken:

```
A a = new A();
B b = new B();
A c = new C();
```

Wat is het resultaat van de volgende drie statements?

- `a.print();`
- `b.print();`
- `c.print();`

Antwoord – Het correct antwoord is:



Oefening 7.15 – K3 Open het project *K3*. Je vindt het bij de oefeningenbestanden van hoofdstuk 7.

Je kan er niet aan ontkomen. Na enige tijd wordt het trio van de meidengroep K3 vervangen door een nieuw trio. En dat betekent een grote verkiezingsshow waarbij je kan stemmen op je favoriete kandidate. In elk van de drie verkiezingsrondes wordt de meest geliefde kandidate uitgekozen om het gezicht te worden van de vernieuwde K3.

Elke kandidate speelt natuurlijk haar eigen troeven uit maar toch bezitten alle kandidaten enkele gemeenschappelijke eigenschappen. We hebben die voor jou verzameld in de abstracte klasse `Zangeres`. Alle kandidaten zullen we groeperen in de klasse `K3`. Voor minstens drie verkiezingsrondes dromen ze er allen van om lid te worden van K3.

In de abstracte klasse `Zangeres` werden reeds twee methoden geprogrammeeerd:

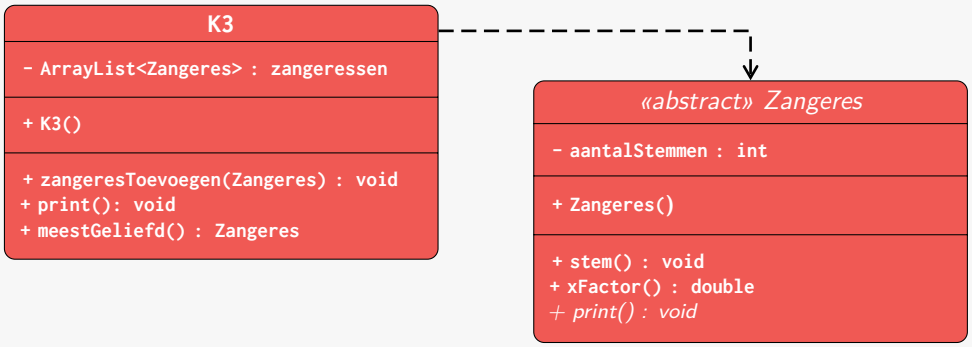
- `void stem()`
  Deze methode verhoogt het enige veld `aantalStemmen` met 1.

- `double xfactor()`
  De retourwaarde van deze methode is het aantal stemmen. Waarom het retourtype `double` is, kom je later te weten.

De klasse `K3` heeft slechts één veld. Het veld `zangeressen` houdt objecten van het type `Zangeres` bij in een `ArrayList`. In de constructor van `K3` werd `zangeressen` geïnitialiseerd.

In deze oefeningen zullen we 3 subklassen van de abstracte klasse Zangeres programmeren. Als kers op de figuurlijke taart programmeer je nog enkele methoden in de klasse K3.

Hou het volgende klassendiagram in je achterhoofd. *Alvorens je een klasse programmeert vul je het best het klassendiagram aan met je nieuwe klasse.*



Nog een belangrijke tip meegeven. Om de naam van een klasse binnen de klassendefinitie te gebruiken, kan je rekenen op de volgende expressie:

```
this.getClass().getName()
```

Gebruik je deze expressie in de klasse K3 dan geeft deze expressie de String "K3" terug.

Jouw opdracht:

- subklasse Marthe (uitbreiding van Zangeres)
  - Programmeer het veld leeftijd.
  - De constructor vraagt alleen de leeftijd.
  - Programmeer de methode void print().

Antwoord – Een correct antwoord is:

```
public class Marthe extends Zangeres
{
    private int leeftijd;

    public Marthe(int leeftijd)
    {
        super();
        this.leeftijd = leeftijd;
    }

    public void print()
    {
        String info = "Naam: " + this.getClass().getName() + "\n";
        info += "Leeftijd: " + leeftijd + " jaar\n";
        info += "x-factor: " + xfactor();
        System.out.println(info);
    }
}
```

- subklasse Klaasje (uitbreiding van Zangeres)
  - Programmeer de constructor.
  - Programmeer de methode void print().

Antwoord – Een correct antwoord is:

```
public class Klaasje extends Zangeres
{
    public Klaasje()
    {
        super();
    }

    public void print()
    {
        System.out.println("Ik ben " + this.getClass().getName() + " en
            mijn x-factor is " + this.xfactor());
    }
}
```

- subklasse Hanne (uitbreiding van Zangeres)
  - Programmeer de constructor.
  - Overschijf de methode double `xfactor()`. De nieuwe methode roept `xfactor()` van de superklasse op, maar de stemmen worden vermenigvuldigd met 1.5 omdat Hanne vals wil spelen.
  - Programmeer de methode void `print()`.

Antwoord – Een correct antwoord is:

```
public class Hanne extends Zangeres
{
    public Hanne()
    {
    super();
    }

    public double xfactor()
    {
        return super.xfactor() * 1.5;
    }

    public void print()
    {
        System.out.println("Ik ben " + this.getClass().getName() + "
            en ik ben een mysterie.");
    }
}
```

- K3
  - `void zangeresToevoegen(Zangeres)` voegt een kandidaat toe aan het veld zangeressen.
  - `void print()` toont informatie over alle kandidaten in het Terminalvenster. Zorg voor een witregel tussen elke kandidaat.
  - `Zangeres meestGeliefd()` geeft een verwijzing naar de zangeres met de hoogste x-factor terug.

Antwoord – Een correct antwoord is:

```
import java.util.ArrayList;

    public class K3
    {
    private ArrayList<Zangeres> zangeressen;

    public K3()
    {
        zangeressen = new ArrayList<Zangeres>();
    }

    public void zangeresToevoegen(Zangeres zangeres)
    {
        zangeressen.add(zangeres);
    }

    public void print()
    {
        for(Zangeres zangeres : zangeressen)
        {
            zangeres.print();
            System.out.println("----------------------");
        }
    }

    public Zangeres meestGeliefd()
    {
        double hoogsteXFactor = -Double.MIN_VALUE;
        Zangeres meestGeliefdeZangeres = null;

        for(Zangeres zangeres : zangeressen)
        {
            if(zangeres.xfactor() > hoogsteXFactor)
            {
                hoogsteXFactor = zangeres.xfactor();
                meestGeliefdeZangeres = zangeres;
            }
        }

        return meestGeliefdeZangeres;
    }
}
```

(Naar een oefening van Goderik Lefebvre, Waregem)

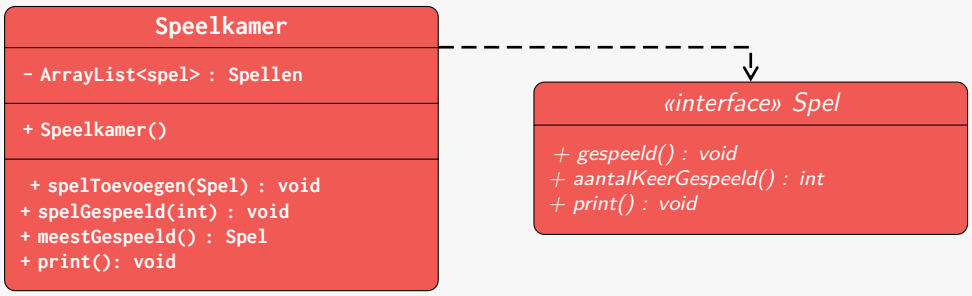
Oefening 7.16 – Speelkamer Open het project *Speelkamer*. Je vindt het bij de oefeningenbestanden van hoofdstuk 7.

In oefening 6 op pagina 94 hebben we bordspellen verzameld in een spellenkast. Maar natuurlijk spelen we ook graag eens een videospel. Een speelkamer zou dus van pas komen. Daar kunnen we zowel bordspellen als videospellen spelen.

In het klassendiagram hebben we voor een willekeurig spel in een interface voorzien. Elke implementatie van de interface `Spel` moet drie methoden implementeren.

De klasse `Speelkamer` verzameld instanties van klassen die de interface `Spel` implementeren in een `ArrayList`.

Hou het volgende klassendiagram in je achterhoofd. *Alvorens je een klasse programmeert vul je het best het klassendiagram aan met je nieuwe klasse.*



- `Bordspel` (implementatie van `Spel`)

  In de klasse `Bordspel` hebben we voor jou een aantal velden geprogrammeerd. Voorts vind je ook een voorlopige versie van de methoden die interface `Spel` eist te implementeren.

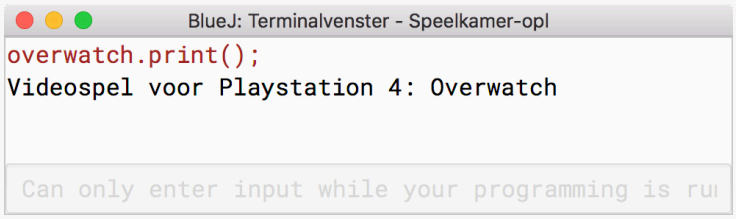
  - Programmeer een passende constructor die aan alle velden een waarde geeft.
  - Programmeer de drie methoden die interface `Spel` eist zodat ze doen wat ze voorspellen te doen. We geven wel de uitwerking van de methode `void print()` mee.

Antwoord – Een correct antwoord is:

```
1  public class Bordspel implements Spel
2  {
3      private int aantalKeerGespeeld;
4      private int aantalSpelers;
5      private String naam;
6
7      public Bordspel(String naam, int aantalSpelers)
8      {
9          this.naam = naam;
10         this.aantalSpelers = aantalSpelers;
11         aantalKeerGespeeld = 0;
12     }
13
14     public int aantalKeerGespeeld()
15     {
16         return aantalKeerGespeeld;
17     }
18
19     public void gespeeld()
20     {
21         aantalKeerGespeeld++;
22     }
23
24     public void print()
25     {
26         System.out.println("Bordspel: " + naam +
27                            " (" + aantalSpelers + " spelers) --> " +
28                            aantalKeerGespeeld + " keer gespeeld");
29     }
30 }
```

- `Videospel` (implementatie van `Spel`)

  Je programmeert zelf de klasse `Videospel`, een implementatie van de interface `Spel`. Vergeet dus niet om de drie methoden van interface `Spel` te implementeren. Maak zeker een veld dat bijhoudt voor welke spelconsole je videospel gemaakt is. We geven opnieuw alleen de werking van de methode `void print()` mee.



Antwoord – Een correct antwoord is:

```
public class Videospel implements Spel
{
    private String console;
    private String naam;
    private int aantalKeerGespeeld;

    public Videospel(String naam, String console)
    {
        this.console = console;
        this.naam = naam;
        aantalKeerGespeeld = 0;
    }

    public int aantalKeerGespeeld()
    {
        return aantalKeerGespeeld;
    }

    public void gespeeld()
    {
        aantalKeerGespeeld++;
    }

    public void print()
    {
        System.out.println("Videospel voor " + console + ": " + naam);
    }
}
```

- `Speelkamer`

  - Het enige veld `spellen` verzamelt instanties van klassen die de interface `Spel` implementeren in een `ArrayList`.
  - Programmeer de constructor die het veld `spellen` initialiseert.
  - `void SpelToevoegen(Spel)` voegt een spel toe aan de speelkamer.
  - `void spelGespeeld(int)` verhoogt het aantal keer dat het spel op de gegeven index (parameter) gespeeld werd met 1.
  - `Spel meestGeliefd()` zoekt tussen alle spellen in de speelkamer het meest gespeelde en dus het meest geliefde spel. De methode geeft een verwijzing naar het meest geliefde spel terug.
  - Ten slotte geven we nog de werking van de methode `void print()` mee.

BlueJ: Terminalvenster - Speelkamer-opl

```
speelkamer.print();
Videospel voor Playstation 4: Overwatch
Bordspel: Agricola (5 spelers) --> 4 keer gespeeld
Bordspel: Zug um Zug (6 spelers) --> 8 keer gespeeld
Bordspel: 7 Wonders (7 spelers) --> 3 keer gespeeld
Videospel voor Xbox One S: Fifa 17
Bordspel: Railroad Tycoon (6 spelers) --> 1 keer gespeeld

Can only enter input while your programming is running
```

Antwoord – Een correct antwoord is:

```
import java.util.ArrayList;

public class Speelkamer
{
    private ArrayList<Spel> spellen;

    public Speelkamer()
    {
        spellen = new ArrayList<Spel>();
    }

    public void spelToevoegen(Spel spel)
    {
        spellen.add(spel);
    }

    public void spelGespeeld(int index)
    {
        if(index >=0 && index < spellen.size())
        {
            spellen.get(index).gespeeld();
        }
    }

    public Spel meestGespeeld()
    {
        int max = Integer.MIN_VALUE;
        Spel maxSpel = null;

        for(Spel spel : spellen)
        {
            if(spel.aantalKeerGespeeld() > max)
            {
            max = spel.aantalKeerGespeeld();
            maxSpel = spel;
            }
        }

        return maxSpel;
    }

    public void print()
    {
        for(Spel spel : spellen)
        {
            spel.print();
        }
    }
}
```

Oefening 7.17 – Berichten Maak een nieuw project *Berichten* aan. Programmeer de volgende klassen:

- abstracte superklasse `Bericht`

  Een bericht bevat steeds een onderwerp en een boodschap. Je bent steeds verplicht om de naam van de afzender op te geven. Implementeer de methode `void toonBericht()`, die afzender, onderwerp en boodschap in het Terminalvenster toont. De methoden `void schrijfBoodschap()` en `void schrijfOnderwerp()` zijn abstracte methoden.

Antwoord – Een correct antwoord is:

```
public abstract class Bericht
{
    protected String onderwerp;
    protected String boodschap;
    protected String afzender;

    public Bericht(String afzender)
    {
        this.afzender = afzender;
        onderwerp = "";
        boodschap = "";
    }

    public abstract void schrijfOnderwerp(String onderwerp);
    public abstract void schrijfBoodschap(String boodschap);

    public void toonBericht()
    {
        String bericht = "Van: " + afzender + "\n";
        bericht += "Onderwerp: " + onderwerp + "\n";
        bericht += "---------------------------------\n";
        bericht += boodschap;

        System.out.println(bericht);
    }
}
```

- subklasse SMS (uitbreiding van Bericht)

  Een instantie van de klasse SMS kan boodschappen van maximaal 10 tekens (ideaal om straks je code te testen) bevatten. Geef je te veel tekens op, dan zullen alleen de eerste tien tekens de sms vormen. Een sms heeft geen onderwerp. Zorg ervoor dat elke sms een bestemming (gsm-nummer) krijgt. Alle gegevens van de sms moet je kunnen tonen in het *Terminalvenster*.

  Antwoord – Een correct antwoord is:

```
public class SMS extends Bericht
{
    private String bestemming;

    public SMS(String afzender, String bestemming)
    {
        super(afzender);
        this.bestemming = bestemming;
    }

    public void schrijfOnderwerp(String onderwerp)
    {
    }

    public void schrijfBoodschap(String boodschap)
    {
        if(boodschap.length() > 10)
        {
        boodschap = boodschap.substring(0,10);
    }

    this.boodschap = boodschap;
    }

    public void toonBericht()
    {
        String bericht = "***SMS*****************************\n";
        bericht += "Aan: " + bestemming + "\n";
        bericht += "Van: " + afzender + "\n";
        bericht += "Bericht: " + boodschap + "\n";
        bericht += "*********************************\n";

        System.out.println(bericht);
    }
}
```

- subklasse `Email` (uitbreiding van `Bericht`)

  Een instantie van de klasse `Email` heeft een e-mailadres als bestemming. De boodschappen kunnen eindeloos lang zijn. Je kan steeds een onderwerp geven aan een e-mail. Zorg ervoor dat je een e-mail kan tonen in het *Terminalvenster*.

  Antwoord – Een correct antwoord is:

```
public class Email extends Bericht
{
    private String bestemming;

    public Email(String afzender,String bestemming)
    {
        super(afzender);
        this.bestemming = bestemming;
    }

    public void schrijfOnderwerp(String onderwerp)
    {
        this.onderwerp = onderwerp;
    }

    public void schrijfBoodschap(String boodschap)
    {
        this.boodschap = boodschap;
    }

    public void toonBericht()
    {
        System.out.println("***E-mail**********************");
        System.out.println("Aan: " + bestemming);
        super.toonBericht();
        System.out.println("**********************************\n");
    }
}
```

- klasse `PostvakUit`

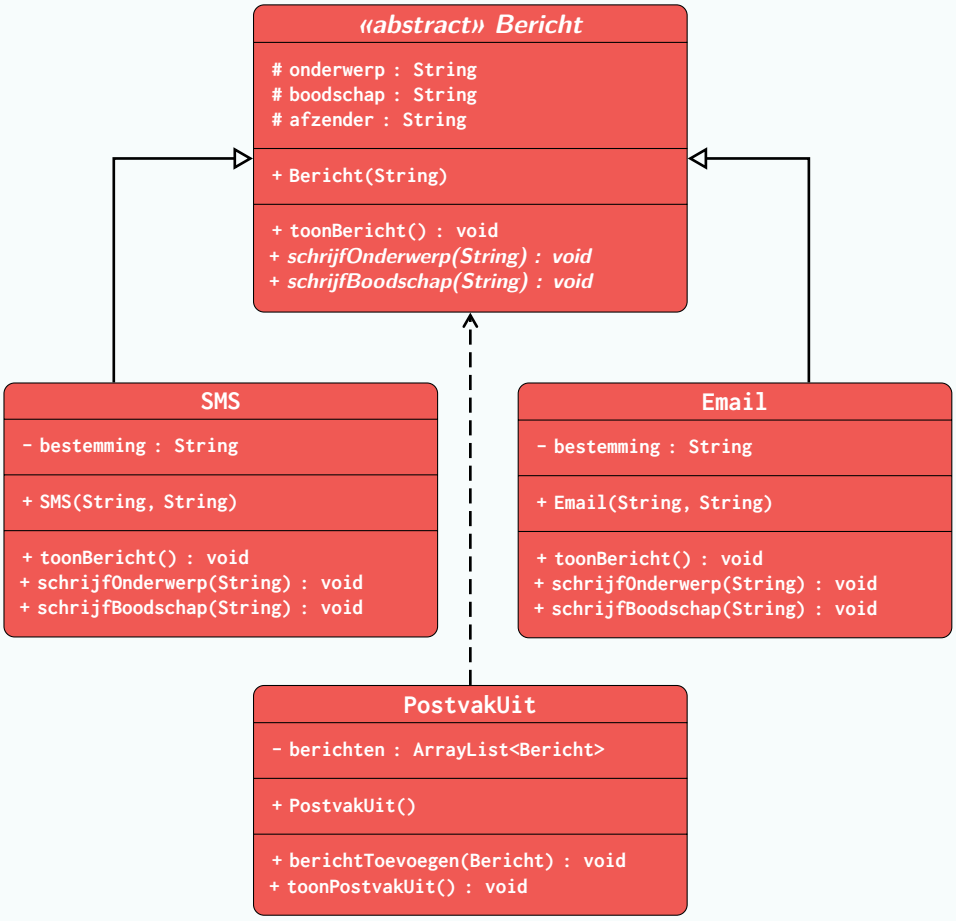
  Een instantie van de klasse `PostvakUit` beheert alle berichten van de afzender. Een object van de klasse moet minstens een collectie van berichten (sms'en en e-mails) bijhouden en kunnen weergeven in het `Terminalvenster`

  Antwoord – Een correct antwoord is:

```java
import java.util.ArrayList;

public class PostvakUit
{
    private ArrayList<Bericht> berichten;

    public PostvakUit()
    {
        berichten = new ArrayList<Bericht>();
    }

    public void berichtToevoegen(Bericht bericht)
    {
        berichten.add(bericht);
    }

    public void toonPostvakUit()
    {
        for(Bericht bericht : berichten)
        {
            bericht.toonBericht();
        }
    }
}
```

Maak eerst een klassendiagram alvorens je begint te programmeren!

Antwoord – Een correct klassendiagram is:



Arrays en for-lus A

Oefening A.1 – Radio Open het project *Radio* bij de oefeningenbestanden van appendix A. In dit project vind je twee klassen.

De klasse `Kanaal` bestudeer je grondig. Deze klasse beschrijft een radiokanaal dat we kunnen voorprogrammeren op een radiotoestel. Een instantie van de klasse `Kanaal` houdt de frequentie en naam van het radiostation bij. Behalve accessormethoden vind je ook een `print()`-methode.

De klasse `Toestel` beschrijft een radiotoestel. Het radiotoestel heeft een vast aantal radiostations die je kan voorprogrammeren. Vergeet niet om in de klassendefinitie het nodige commentaar te zetten.

- Programmeer veld en constructor:
  - Programmeer het enige veld `radioStations`. Het veld `radioStations` is een array die verwijzingen naar objecten van het type `Kanaal` bijhoudt.
  - Programmeer de constructor van de klasse `Toestel`. De constructor heeft één parameter die vermeldt hoeveel radiostations je kan voorprogrammeren op het radiotoestel.

Antwoord – Een correct antwoord is:

```
public class Toestel
{

    private Kanaal radioStations[];

    public Toestel(int aantalStations)
    {
        radioStations = new Kanaal [aantalStations];
    }

    ...
```

- `int aantalRadioStations()`
  Deze methode geeft je het aantal radiostations dat we op het radiotoestel kunnen voorprogrammeren. Een correct antwoord is:

Antwoord – Een correct antwoord is:

```
    public int aantalRadioStations()
    {
        return radioStations.length;
    }
```

- `void voegToe( Kanaal, int)`
  Deze methode voegt een instantie van de klasse `Kanaal` toe aan de array van radiozenders. De plaats waar je het kanaal moet toevoegen krijg je mee via de tweede parameter. Let op, je kan een kanaal alleen toevoegen op een plaats die nog niet voorgeprogrammeerd is.

  Antwoord – Een correct antwoord is:

```
    public void voegToe(Kanaal kanaal, int nummer)
    {
        if(nummer >= 0 && nummer < radioStations.length && radioStations[
            nummer] == null)
        {
            radioStations[nummer] = kanaal;
        }
    }
```

- `double zoekFrequentie(String)`
  Deze methode zoekt in de lijst van radiostations naar het radiostation waarvan je de naam kan meegeven met de parameter. De retourwaarde kan heel verschillend zijn:
  - `-1` indien de radiozender niet gevonden werd.
  - Indien de naam van de radiozender gevonden werd, wordt de frequentie van de radiozender teruggegeven.

  Antwoord – Een correct antwoord is:

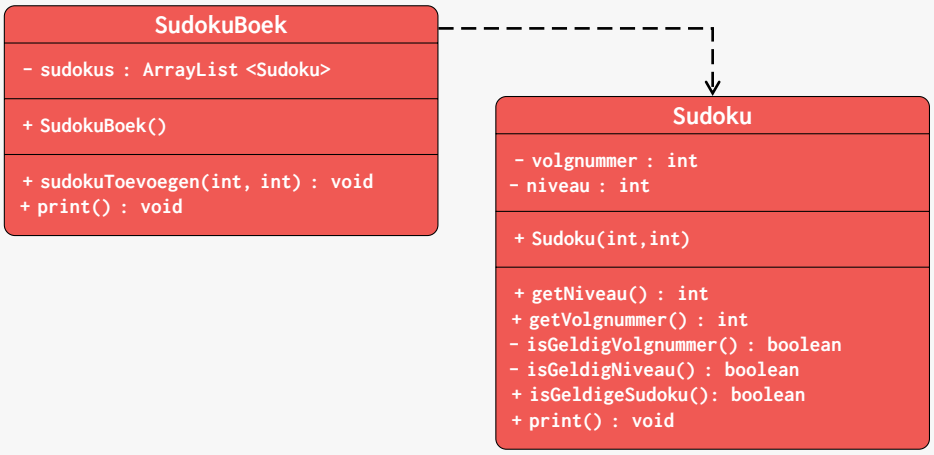
```
    public double zoekFrequentie(String naam)
    {
        int index = -1;

        while(index < radioStations.length && radioStations[index].
            getNaam() != naam)
        {
            index++;
        }

        if(index < radioStations.length)
        {
            return radioStations[index].getFrequentie();
        }
        else
        {
            return 0;
        }
    }
```

- `void printRadioStations()`
  De methode toont samen met het nummer de lijst van radiostations in het *Terminalvenster*. Maak gebruik van de methode `void printKanaal()` van de klasse `Kanaal`.

  Antwoord – Een correct antwoord is:

```
1
2     public void printRadioStations()
3     {
4         for(int i = 0; i < radioStations.length; i++)
5         {
6             if(radioStations[i] != null)
7             {
8                 System.out.print(i + ": ");
9                 radioStations[i].printKanaal();
10            }
11            else
12            {
13                System.out.println(i + ": -");
14            }
15    }
```

Oefening B.1 Open het project *Sudoku* bij de oefeningenbestanden van appendix B.



Programmeer de methode `void sudokuToevoegen(int, int)` die alleen geldige sudoku's toevoegt met een geldig en uniek volgnummer. Zorg voor een testklasse en de nodige testen. We beloven niet dat de gegeven methodes in de klasse Sudokuboek uiteindelijk zullen overleven. We beloven ook niet dat je geen extra methoden zal moeten toevoegen aan de klassendefinitie.

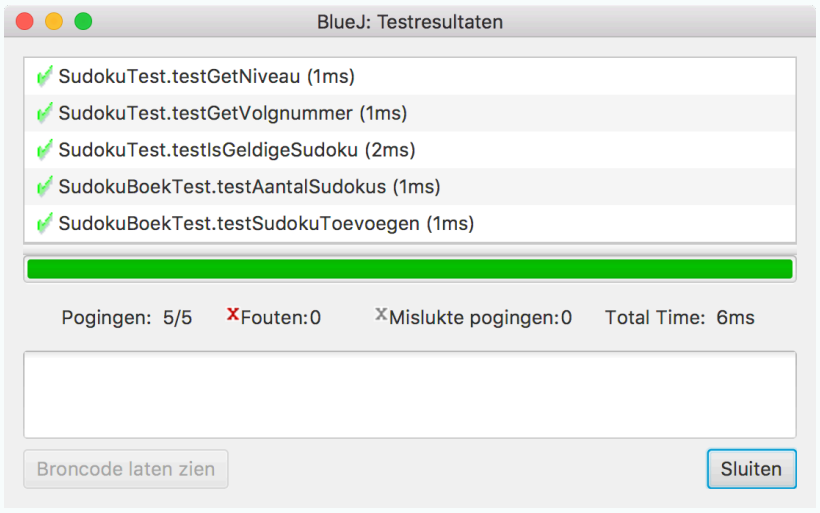
Antwoord – De klasse Sudoku en SudokuTest hoef je niet aan te passen. Aan SudokuBoek werden volgende methoden toegevoegd:

```
1     public boolean sudokuToevoegen(int niveau)
2     {
3         Sudoku sudoku = new Sudoku(volgendnummer, niveau);
4         boolean isGeldig = sudoku.isGeldigeSudoku();
5
6         if(isGeldig)
7         {
8             sudokus.add(sudoku);
9             volgendnummer++;
10        }
11
12        return isGeldig;
13    }
14
15    public int aantalSudokus()
16    {
17        return sudokus.size();
18    }
```

Het veld volgendnummer zorgt steeds voor een geldig volgnummer. Een sudoku wordt niet toegevoegd wanneer het moeilijkheidsniveau ongeldig is. De methode `int aantalSudokus()` komt van pas bij het testen.

Antwoord – Een klassendefinitie van de testklasse SudokuBoekTest is:

```
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class SudokuBoekTest
{
    private SudokuBoek sudokuBo1;

    public SudokuBoekTest()
    {
    }

    @Before
    public void setUp()
    {
        sudokuBo1 = new SudokuBoek();
    }

    @After
    public void tearDown()
    {
    }

    @Test
    public void testSudokuToevoegen()
    {
        assertEquals(false, sudokuBo1.sudokuToevoegen(0));
        assertEquals(true, sudokuBo1.sudokuToevoegen(1));
        assertEquals(true, sudokuBo1.sudokuToevoegen(2));
        assertEquals(true, sudokuBo1.sudokuToevoegen(3));
        assertEquals(false, sudokuBo1.sudokuToevoegen(4));
    }

    @Test
    public void testAantalSudokus()
    {
        assertEquals(false, sudokuBo1.sudokuToevoegen(0));
        assertEquals(true, sudokuBo1.sudokuToevoegen(1));
        assertEquals(true, sudokuBo1.sudokuToevoegen(2));
        assertEquals(true, sudokuBo1.sudokuToevoegen(3));
        assertEquals(false, sudokuBo1.sudokuToevoegen(4));
        assertEquals(3, sudokuBo1.aantalSudokus());
    }
}
```



Debuggen C

Oefening C.1 – Datum Open het project *Datum* bij de oefeningenbestanden van appendix C. In dit project vind je de klasse `Datum`.

De klasse bevat geen syntaxfouten en lijkt perfect te doen wat verwacht wordt, nl. het bijhouden van een datum. Van jou wordt verwacht om met een kritisch oog naar de code van de klasse te kijken. De klasse moet altijd een geldige datum bevatten:

- Een datum ten minste gelijk aan 1 januari 1900.
- Wanneer de gebruiker een ongeldig datum-object dreigt aan te maken, stel je de datum gelijk aan 1 januari 1900.
- Je mag voor de eenvoud aannemen dat er geen schrikkeljaren bestaan.

Je moet niet alleen een geldige datum bijhouden, je moet ook correct de datum met één dag kunnen verhogen.

Antwoord – Een correct antwoord is:

```
public class Datum
{
    private int dag;
    private int maand;
    private int jaar;

    public Datum(int dag, int maand, int jaar)
    {
        this.dag = dag;
        this.maand = maand;
        this.jaar = jaar;

        if(! geldigeDatum())
        {
            this.dag = 1;
            this.maand = 1;
            this.jaar = 1900;
        }
    }
    public int aantalDagenInMaand()
    {
        int dagen = 31;

        switch(maand)
        {
            case 4:
            case 6:
            case 9:
            case 11:
                dagen = 30;
                break;
            case 2:
                dagen = 28;
                break;
        }
        return dagen;
    }
    public boolean geldigeDatum()
    {
        boolean isGeldig = false;

        if(dag > 0 && dag <= aantalDagenInMaand() && jaar >= 1900)
        {
            isGeldig = true;
        }
        return isGeldig;
    }
    public void volgendeDag()
    {
        dag++;

        if(dag > aantalDagenInMaand())
        {
            maand++;
            dag = 1;
        }
        if(maand == 13)
        {
            jaar++;
            maand = 1;
        }
    }
}
```