HO
GENT

H7 Pijlers van OO - Oefeningen

Table of Contents

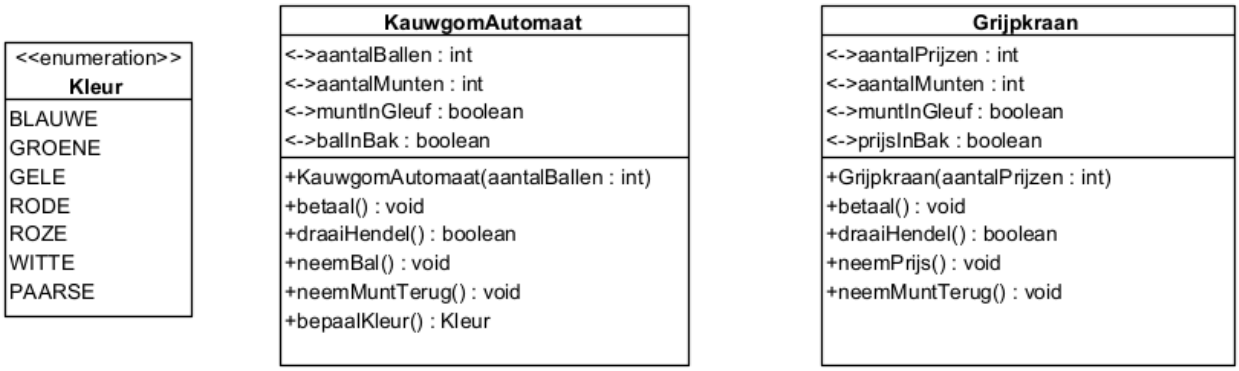
1. Doelstellingen per oefening

- Oefening 2.1: Overerving: ontwerp en implementatie
- Oefening 2.2: Overerving: implementatie
- Oefening 2.3: Polymorfisme: welke toekenningen en methode-aanroepen zijn geldig?
- Oefening 2.4: Polymorfisme: implementatie
- Oefening 2.5 en 2.6: implementatie overerving inoefenen
- Oefening 2.7: implementatie polymorfisme inoefenen

2. Oefeningen

2.1. KauwgomAutomaat en GrijpKraan: ontwerp en implementatie

Gegeven volgende UML:



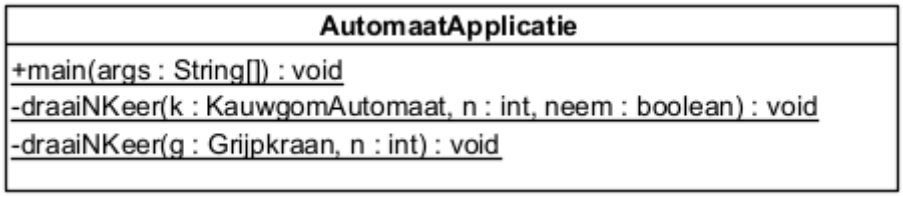
Bekijk waar je in deze UML overerving zou kunnen gebruiken. Maak, indien mogelijk, een superklasse met de gemeenschappelijke kenmerken van deze klasses.

Hou ook rekening met volgende implementatie voor de verschillende methodes:

- **constructor**: stelt het aantal ballen van de KauwgomAutomaat of het aantalPrijzen van de Grijpkraan in
- methode **betaal**: zorgt ervoor dat er een munt in de gleuf zit
- methode **draaiHendel**:
  - implementatie in klasse **KauwgomAutomaat**:
    - Controleer eerst of er betaald werd, of er niks meer in de bak ligt en of er nog wel ballen zijn
    - Indien aan deze voorwaarden voldaan werd, pas dan het aantal ballen en het aantal munten aan, verwijder de munt uit de gleuf en leg een bal in de bak. In dit geval is het draaien gelukt.

        - Indien niet aan een van bovenstaande voorwaarden is voldaan, dan moet de eventuele betaalde munt teruggegeven worden. In dit geval is het draaien mislukt.
    - implementatie in klasse **Grijpkraan**:
        - Eerst wordt random bepaald of je gewonnen hebt (50% kans).
        - Indien je gewonnen hebt, dan gebeurt hetzelfde als wat hierboven beschreven werd voor de implementatie van draaiHendel in KauwgomAutomaat.
        - Indien je niet gewonnen hebt, dan ben je je munt kwijt en is het draaien niet gelukt.
- methodes **neemBal** en **neemPrijs**: zorgt ervoor dat er geen bal/prijs meer in de bak zit
- methode **neemMuntTerug**: zorgt ervoor dat er geen munt meer in de gleuf zit
- methode **bepaalKleur**: returnt een random kleur uit de opgesomde kleuren in enum Kleur

Zorg ook voor een implementatie van alle klassen uit je ontwerp.
Op chamilo vind je een kleine applicatie AutomaatApplicatie_start.java, die voldoet aan volgende UML en voorbeelduitvoer.



```
10 keer aan de hendel draaien van een kauwgomautomaat met 200 ballen
Poging 1: witte kauwgombal
Poging 2: paarse kauwgombal
Poging 3: witte kauwgombal
Poging 4: blauwe kauwgombal
Poging 5: gele kauwgombal
Poging 6: witte kauwgombal
Poging 7: roze kauwgombal
Poging 8: gele kauwgombal
Poging 9: roze kauwgombal
Poging 10: rode kauwgombal

10 keer aan de hendel draaien van een grijpkraan met 20 ballen
Poging 1: prijs gewonnen
Poging 2: geen prijs gewonnen
Poging 3: prijs gewonnen
Poging 4: prijs gewonnen
Poging 5: prijs gewonnen
Poging 6: geen prijs gewonnen
Poging 7: prijs gewonnen
Poging 8: prijs gewonnen
Poging 9: geen prijs gewonnen
Poging 10: geen prijs gewonnen

Zonder te betalen... mislukt

Munt teruggenomen... mislukt

2 pogingen, maar vergeten item er uit te nemen na de eerste...
Poging 1: roze kauwgombal
Poging 2: Kauwgombal zit er nog in!

En als alle items op zijn...
Poging 1: blauwe kauwgombal
Poging 2: geen kauwgom meer!
```

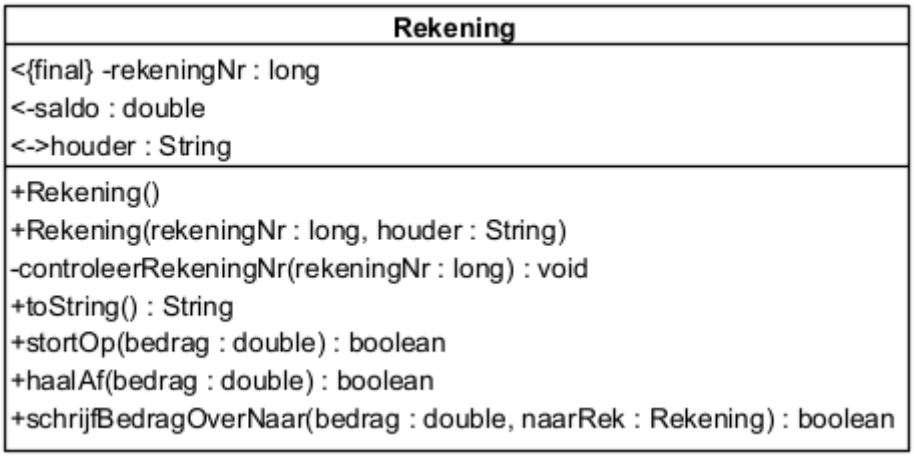
Je kan deze applicatie uitvoeren ter controle van je eigen geïmplementeerde klassen. Hierbij moet worden opgemerkt dat deze testapplicatie niet volgens de regels van de 3 lagenarchitectuur is opgesteld, en dus enkel hier als hulp dient ter controle van je eigen code.

2.2. Rekening met subklasses: implementatie

Gegeven: een klasse Rekening (zie startproject H7_Oefening2_start):

- Attributen: rekeningNr (long), saldo (double) en houder (String)
- Methoden: constructor, get- en setmethoden, controleermethode voor het rekeningNr, stortOp, haalAf, schrijfBedragOverNaar en toString

Zie ook volgende UML:



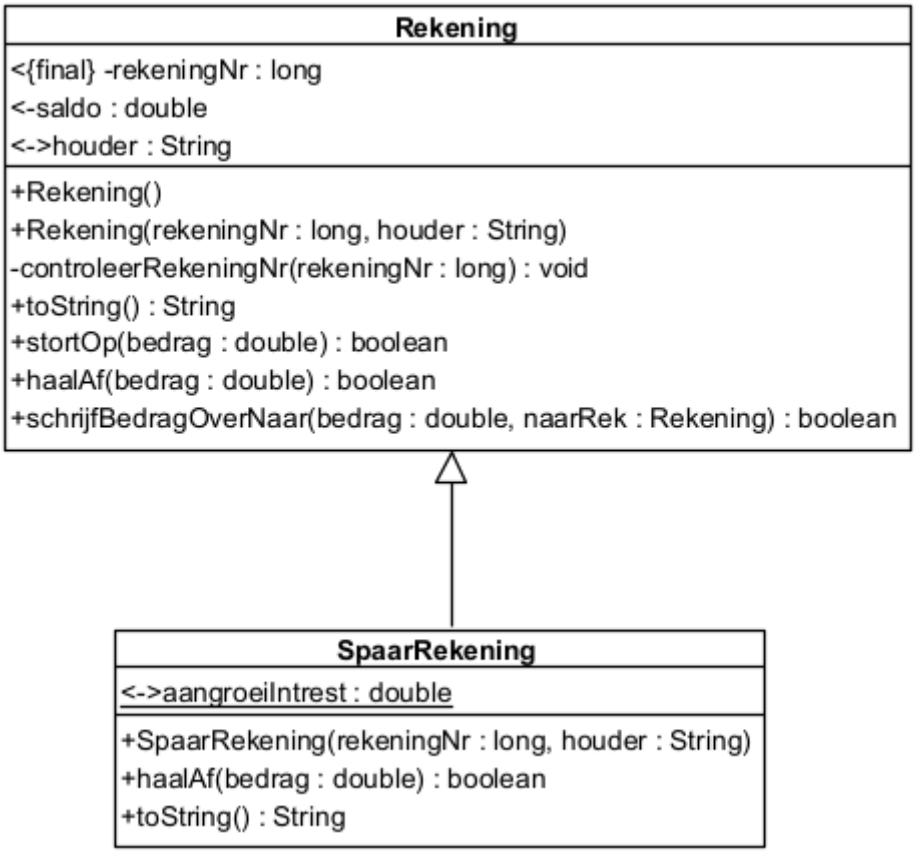
In het startproject zijn ook 3 testklassen gegeven, die bij elk van de onderstaande 3 stappen horen.

Gevraagd:

1. Pas de klasse Rekening aan:
   a. Het attribuut rekeningNr mag na creatie niet meer gewijzigd worden. Doe de nodige aanpassingen.
   Moet er iets wijzigen aan:
      i. Het attribuut?
      ii. De constructoren?
      iii. De setter?
   b. Maak in de methode toString gebruik van this.getClass().getSimpleName() om de naam van de klasse op te vragen
   c. Maak een default constructor: rekeningNr 0L, houder "onbekend"
2. Definieer nu de subklasse SpaarRekening:
   a. Extra attribuut: aangroeiIntrest (double): voor alle spaarRekeningen dezelfde waarde, kan wel gewijzigd worden
   b. Extra methodes: constructor, getter en setter met controle: aangroeiIntrest niet onder 0!
   c. Override methodes: haalAf (mag niet onder 0!), toString

      uitvoer bij het aanroepen de toString-methode:

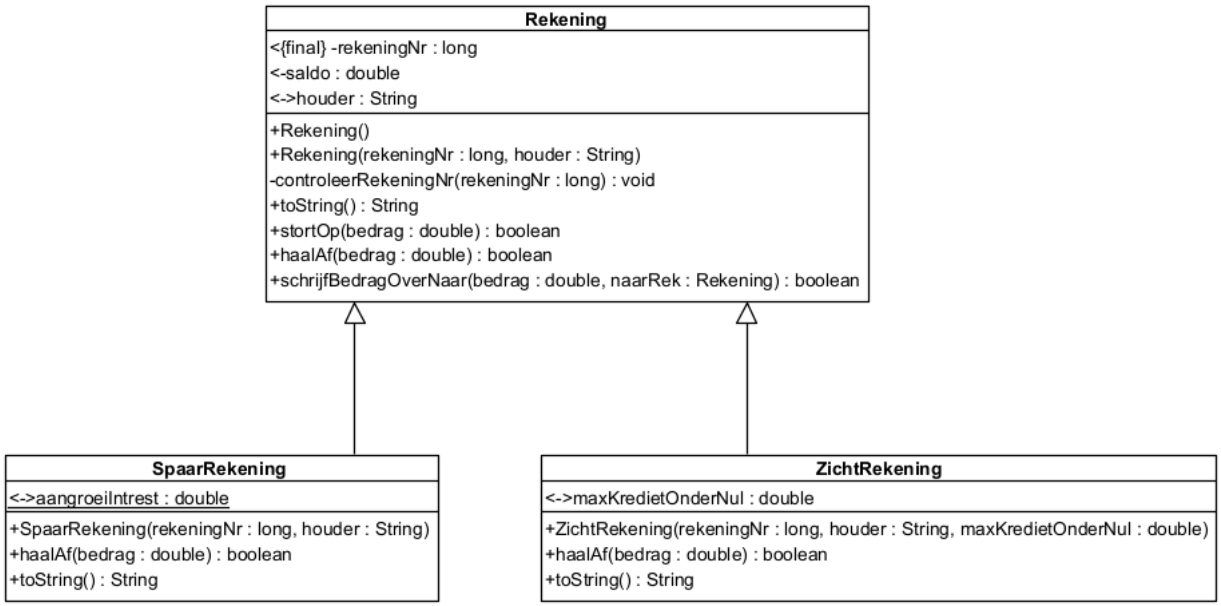
```
SpaarRekening met rekeningnummer 123-1234569-86
staat op naam van piet
en bevat 450,00 euro. Aangroeiintrest = 1,50%
```



3. Definieer een tweede subklasse ZichtRekening:
   a. Extra attribuut: maxKredietOnderNul (double)
   b. Extra methodes: constructor, getter en setter met controle op maxKrediet: is negatief en kleiner of gelijk aan huidig saldo
   c. Override methodes: haalAf (mag niet onder maxKredietOnderNul), toString

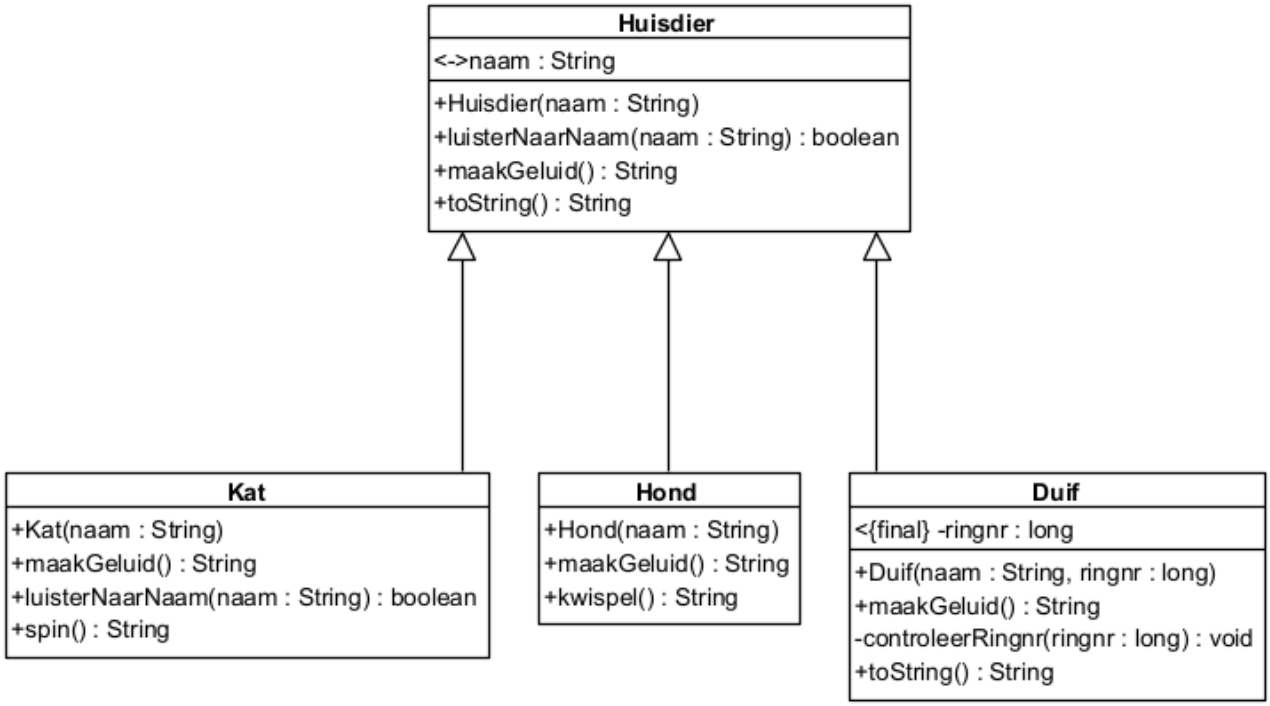
   uitvoer bij het aanroepen de toString-methode:

```
ZichtRekening met rekeningnummer 123-1234569-86
staat op naam van piet
en bevat 450,00 euro. Max krediet onder nul = -1500,00
```


Rekening
<{final} -rekeningNr : long
<-saldo : double
<->houder : String
+Rekening()
+Rekening(rekeningNr : long, houder : String)
-controleerRekeningNr(rekeningNr : long) : void
+toString() : String
+stortOp(bedrag : double) : boolean
+haalAf(bedrag : double) : boolean
+schrijfBedragOverNaar(bedrag : double, naarRek : Rekening) : boolean
SpaarRekening
<->aangroeiIntrest : double
+SpaarRekening(rekeningNr : long, houder : String)
+haalAf(bedrag : double) : boolean
+toString() : String
ZichtRekening
<->maxKredietOnderNul : double
+ZichtRekening(rekeningNr : long, houder : String, maxKredietOnderNul : double)
+haalAf(bedrag : double) : boolean
+toString() : String

2.3. Polymorfisme: wat is toegelaten?

Gegeven volgende UML:

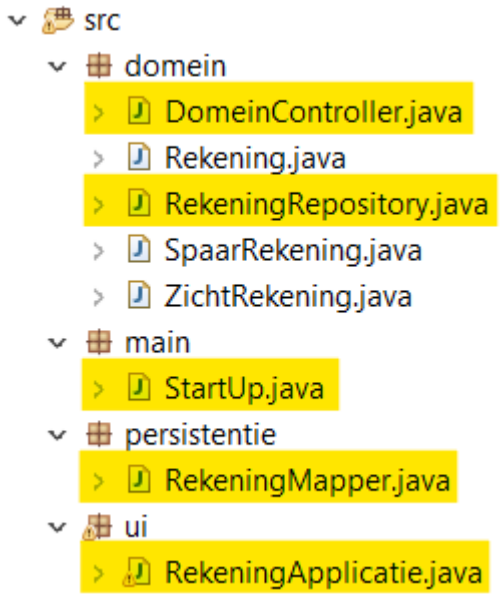


Zijn volgende declaraties en statements juist of fout? Waarom wel/niet?

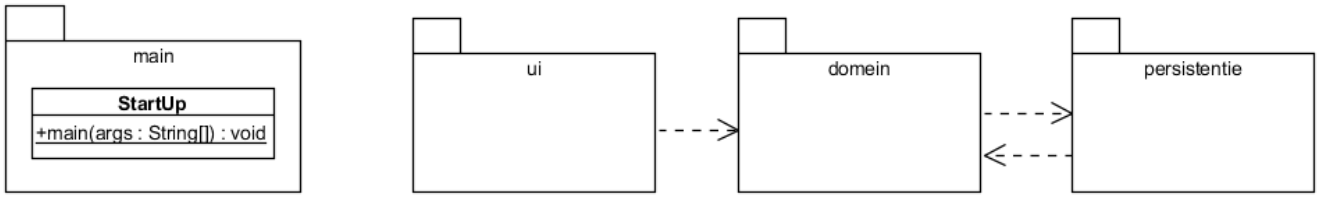
- Huisdier dier = new Duif("Grijsje");
- Huisdier dier = new Huisdier("Pluk");
- Kat kat = new Huisdier("Felix");
- Duif duif = new Duif("Knabbel",20181111111L);
  return duif.getNaam();
- Huisdier dier = new Kat("Goofie");
  return dier.spin();
- Kat dier = new Kat("Goofie");
  return dier.spin();
- Duif duif = new Hond("Tarzan");
  return duif.kwispel();
- Huisdier dier = new Hond("Tarzan");
  return ((Hond)dier).kwispel();

2.4. Polymorfisme: aanpassing oefening Rekening

We gaan verder met de vorige versie van de oefening van de Rekeningen (oefening 2). We behouden de 3 domeinklasses Rekening, SpaarRekening en ZichtRekening en breiden de oefening verder uit met de klasses die met fluo zijn aangeduid in de volgende figuur.

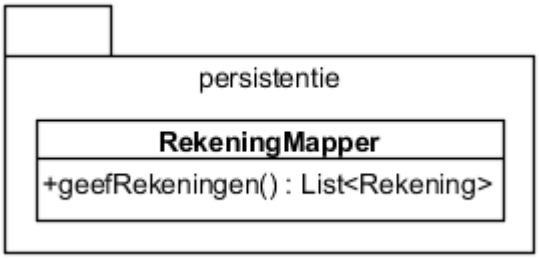


We maken met de bestaande domeinklasses een volledige applicatie met drie lagen.



1. De ui die we hier gebruiken is nog een cui (console user interface). In de toekomst (OOSD II) kan dit ook een gui (grafische user interface) worden.
2. In de presentatielaag (cui) maken we gebruik van DTO's in plaats van "echte" objecten!

1. We beginnen met de klasse **RekeningMapper** uit de package **persistentie**.



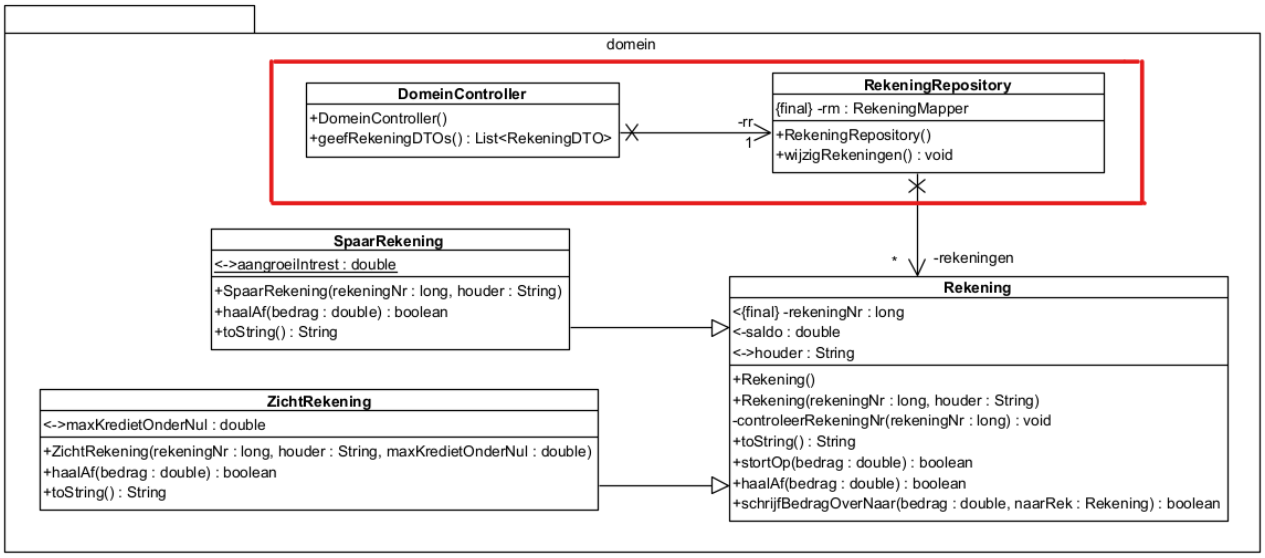
Deze bevat slechts één methode: geefRekeningen, die een lijst van Rekening-objecten teruggeeft. We vullen de lijst op met een Spaar- en ZichtRekening naar keuze, bijvoorbeeld:

```
Rekening rekening1 = new ZichtRekening(123456700082L, "Jan",-2000);
```

```
rekening1.stortOp(1200);
Rekening rekening2 = new SpaarRekening(123456780009L, "Sandra");
rekening2.stortOp(5000);
```

Vergeet ook niet de aangroeiIntrest in te stellen, bijvoorbeeld op 2%.

2. Daarna gaan we verder in de **domeinlaag**. Hier moeten we nog 2 klassen (de rood omlijnde) aanvullen: de **DomeinController** en de **RekeningRepository**. We beginnen met die laatste.

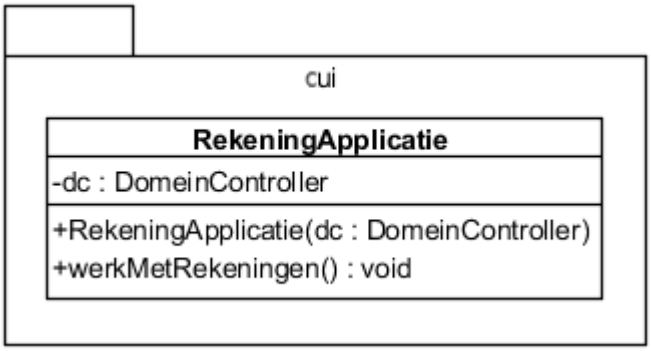


   a. De klasse **RekeningRepository** uit de package **domein**.
      i. attributen: rm (type: RekeningMapper), rekeningen (dynamische lijst van Rekening-objecten)
      ii. defaultconstructor: rekeningMapper wordt gecreëerd en het attribuut rekeningen wordt opgevuld met een lijst, afkomstig van de persistentielaag.
      iii. getter voor de rekeningen, zodat deze lijst straks door de DomeinController kan opgevraagd worden
      iv. methode wijzigRekeningen:
      Voer de volgende bewerkingen uit afhankelijk van het soort object:
         A. Voor alle SpaarRekeningen: saldo wordt verhoogd met aangroeiIntrest (let op: dit is een percentage!)
         B. Voor alle ZichtRekeningen: maxKredietOnderNul wordt met 10 verminderd.
   b. De klasse **DomeinController** uit de package **domein**.
      i. attributen: rr (type RekeningRepository)
      ii. defaultconstructor: maak een object van de RekeningRepository en ken het toe aan het attribuut rr
      iii. methode geefRekeningDTOs: wijzigt de lijst van Rekening-objecten in de repository en zet de resulterende lijst om naar een lijst van DTO's
      iv. voor dit laatste heb je dus ook een record nodig, genaamd **RekeningDTO**, in package **dto**. Aangezien de info van een Zichtrekening verschilt van deze van een Spaarrekening

schrijven we een record dat alle info bevat. Via een veld soort (bv. char soort) kan dan meegegeven worden over welk soort Rekening-object het precies gaat.

```
package dto;

public record RekeningDTO(long rekeningnummer, double saldo, String houder,
        double maxKredietOnderNul, double aangroeiIntrest,char soort)
{}
```

3. Vervolgens gaan we naar de **presentatielaag** om de klasse **RekeningApplicatie** te schrijven:



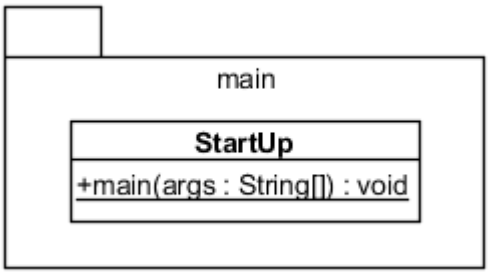
De klasse **RekeningApplicatie** zit in de package **cui** en zal dus alles bevatten qua code die nodig is voor de in- en uitvoer. De klasse moet kunnen communiceren met de domeinlaag en zal daarvoor één object van de DomeinController bijhouden (attribuut dc).

Maak ook nog een constructor waarmee het attribuut dc kan ingesteld worden.

De methode werkMetRekeningen zorgt ervoor dat de juiste methode(s) uit de DomeinController aangeroepen worden om straks volgende (voorbeeld)uitvoer te krijgen:

```
ZICHTREKENING 404 op naam van Jens heeft saldo van 4000,00 met maxKrediet van -2510,00
SPAARREKENING 202 op naam van Michiel heeft saldo van 2040,00 met aangroeiIntrest van 2,00
SPAARREKENING 101 op naam van Senne heeft saldo van 1020,00 met aangroeiIntrest van 2,00
ZICHTREKENING 303 op naam van Kamiel heeft saldo van 3000,00 met maxKrediet van -2010,00
```

4. Tenslotte schrijven we nog de code voor de klasse **StartUp** in de package **main**:

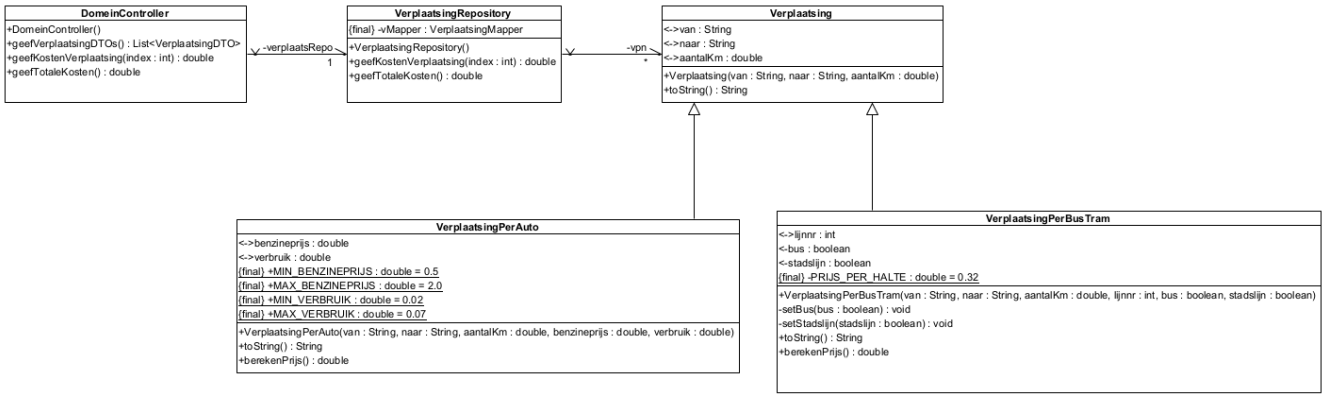


Deze klasse bevat de (enige) main-methode (van heel het project).
In deze main-methode komt code om het volgende te doen:
Maak een object van de DomeinController.
Maak een object van RekeningApplicatie en geef het DomeinController-object door.
Roep de methode werkMetRekeningen op via het gemaakte object.

2.5. Stageverplaatsingen

Een hogeschool vraagt aan de lectoren om de stageverplaatsingen bij te houden. Er bestaan 2 soorten verplaatsingen: per auto of per bus/tram. In beide gevallen moeten enkele gegevens worden bijgehouden. De bedoeling is dat de kostprijs kan worden berekend van elke verplaatsing en zo ook het totaal van alle verplaatsingskosten van een lector.

package domein:



superklasse Verplaatsing (zie ook UML en testklasse):

- 3 attributen: van (String), naar (String) en aantalKm (double).
- constructor met 3 parameters
- getter voor elk attribuut
- setter voor elk attribuut:
  - van en naar moeten ingevuld zijn, anders exception gooien met duidelijke boodschap
  - aantalKm moet een strikt positief getal zijn, anders exception met passende boodschap gooien
  - toString: plaats in 1 string: “verplaatsing van X naar Y” (met X en Y ingevuld met de gegeven waarden)

subklasse VerplaatsingPerAuto (zie ook UML en testklasse):

- 2 extra attributen: benzineprijs (double, in euro/liter) en verbruik (double, in liter/km)
- constructor met 5 parameters.
- getter voor elk extra attribuut
- setter voor elk extra attribuut :
  - benzineprijs moet tussen 0.50 en 2.00 euro liggen (grenzen inbegrepen), anders exception gooien
  - verbruik moet tussen 0.02 en 0.07 liter liggen (grenzen inbegrepen), anders exception gooien
- extra methode berekenPrijs: de prijs van een verplaatsing per auto (heen en terug) wordt berekend adhv de formule: verbruik * benzineprijs * aantalKm * 2

- toString: zie output

subklasse VerplaatsingPerBusTram (zie ook UML en testklasse):

- 4 extra attributen: lijnnr (int), bus (boolean), stadslijn (boolean), PRIJS_PER_HALTE (double). Het laatste attribuut is een vaste waarde voor alle verplaatsingen per bus of tram en zal in de loop van het programma niet gewijzigd kunnen worden.
- constructor met 6 parameters (alles behalve PRIJS_PER_HALTE).
- getter voor elk extra attribuut behalve PRIJS_PER_HALTE
- setter voor elk extra attribuut behalve PRIJS_PER_HALTE:
  - lijnnr moet een strikt positief geheel getal zijn, anders exception gooien
- extra methode berekenPrijs: de prijs van de verplaatsing (heen en terug) bereken je als volgt:
  - er is 1 halte per km, dus als je de afstand (het aantalKm) van de verplaatsing afrondt naar boven (en naar een geheel getal) dan ken je het aantal haltes
  - de prijs van de verplaatsing is dan 2 * aantalHaltes * PRIJS_PER_HALTE
  - voor een verplaatsing met een stadslijn krijg je 20% korting op deze prijs.
- toString: zie output

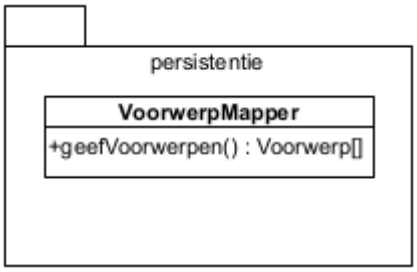
De klasse **VerplaatsingRepository**:

- haalt bij creatie de Verplaatsing-objecten uit de persistentielaag en houdt die bij in een array vpn.
- via de methode geefKostenVerplaatsing kan voor één verplaatsing op de meegegeven index in de array, de kostprijs berekend worden.
- via de methode geefTotaleKosten, kan de kost van alle verplaatsingen berekend worden

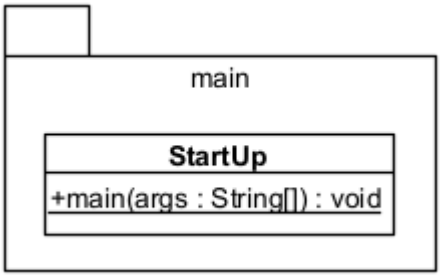
De klasse **DomeinController** :

- is voorzien van een methode geefVerplaatsingDTOs, die alle Verplaatsing-objecten als dto's teruggeeft. Voorzie in het VerplaatsingDTO enkel die velden die effectief gebruikt worden in de cui-klasse
- voor de overige methodes zal de DomeinController de andere domeinklasse(n) aanspreken, op de correcte manier volgens de 3-lagen architectuur

package persistentie:

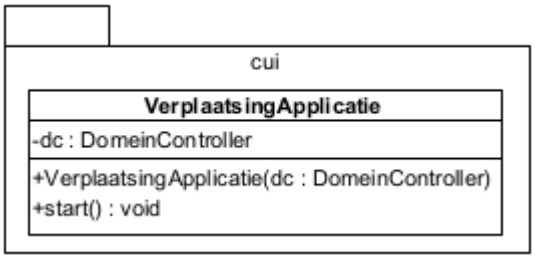


package main:



In de klasse **StartUp** wordt de main-methode uitgewerkt. In deze main-methode starten we de applicatie door de methode start aan te roepen uit de cui-klasse.

package cui:



Schrijf een applicatie waarin de array met alle Verplaatsing-objecten uit de persistentielaag wordt getoond. Werk volgens de correcte 3-lagen werkwijze!

Probeer de uitvoer op dezelfde manier als in onderstaand voorbeeld te tonen.

Geef eerst alle informatie over de verplaatsing per auto of de verplaatsing per bus of tram, met kost, weer. Ten slotte worden de totale kosten voor alle verplaatsingen samen getoond.

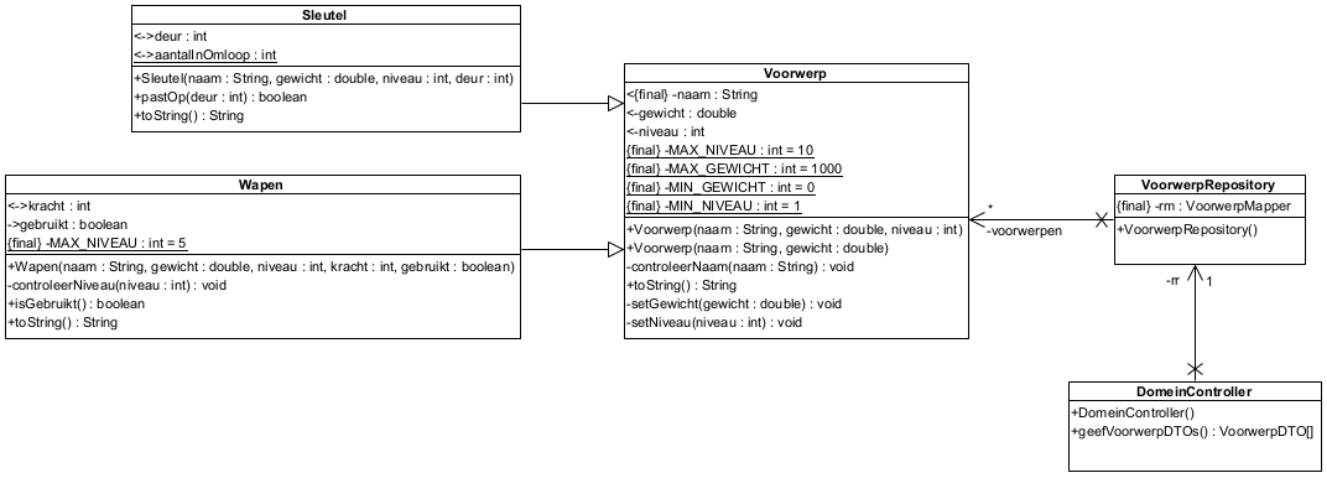
Mogelijke uitvoer:

```
verplaatsing van HoGent campus Schoonmeersen Gent naar EFFIX Waregem per auto
Kosten voor deze verplaatsing: € 4,58

verplaatsing van HoGent campus Aalst naar Brenso NV Affligem per auto
Kosten voor deze verplaatsing: € 1,89

verplaatsing van HoGent campus Schoonmeersen Gent naar Technologiepark Zwijnaarde met stadsbus 70
Kosten voor deze verplaatsing: € 2,56

verplaatsing van Sint-Pietersstation Gent naar Vijfwindgatenstraat Gent met stadstram 22
Kosten voor deze verplaatsing: € 1,54

Totale kosten voor alle verplaatsingen samen: € 10,57
```

2.6. Spel met voorwerpen

In een spel worden wapens en sleutels gebruikt.

package domein:



Maak de domeinklasse **Voorwerp**:

- 3 attributen: naam (verplicht, mag niet gewijzigd worden), gewicht (in kg) en niveau (int)
- constructor met 3 parameters (controles zie beschrijving setter)
- constructor met 2 parameters. Niveau wordt hier standaard op 1 ingesteld.
- getter voor elk attribuut
- setter voor het attribuut:
  - gewicht: moet een positief getal zijn, kleiner dan 1000 kg
  - niveau: er zijn 10 niveaus in het spel, genummerd van 1 t.e.m. 10
  - geen setter voor naam want naam mag niet gewijzigd worden. Naam mag niet null of leeg zijn.
- toString `Voorwerp ⋯ met gewicht ⋯ (3 cijfers na komma) kg uit niveau ⋯`

Maak 2 subklassen:

De klasse **Wapen**:

- 2 extra attributen: kracht en gebruikt
- constructor
- getters voor de extra attributen
- setter voor elk extra attribuut:
  - kracht: is een positief getal
  - gebruikt: geen controle nodig
- wapens zijn enkel beschikbaar in niveaus 1 t.e.m. 5: voorzie hiervoor een controleerNiveau-methode

- toString (zie verder voor de gevraagde output)

De klasse **Sleutel**:

- 2 extra attributen: deur (int), nummer van de deur waarop de sleutel past en aantalInOmloop: stelt het aantal sleutels voor die aanwezig zijn in het spel
- constructor
- getters voor de extra attributen
- setter voor deur: moet een positief getal zijn
- toString (zie verder voor de gevraagde output)
- methode pastOp: geeft true/false terug naargelang de huidige sleutel op de als parameter meegegeven deur past of niet

De klasse **VoorwerpRepository**:

- haalt bij creatie de Voorwerp-objecten uit de persistentielaag en houdt die bij in een array voorwerpen.

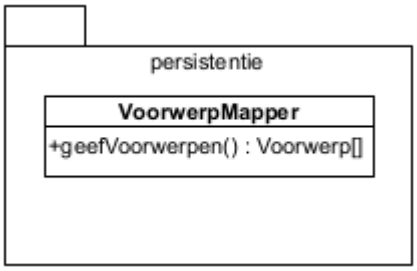
De klasse **DomeinController** :

- is voorzien van een methode geefVoorwerpDTOs, die alle Voorwerp-objecten als dto's teruggeeft. Ook hier heb je een record nodig, genaamd **VoorwerpDTO**, in package **dto**.

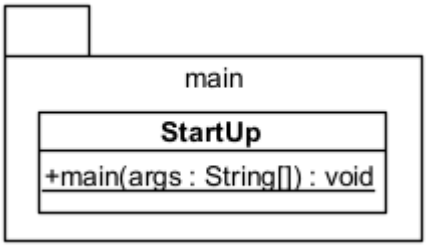
  Een Wapen en een Sleutel bevatten extra (verschillende) info, we schrijven een record dat alle info bevat. Via een veld soort (bv. char soort) kan dan meegegeven worden over welk soort Voorwerp-object het precies gaat. In dit geval kan je ook extra constructors voorzien die je dan gebruikt bij een Wapen-object (resp. een Sleutel-object).

```
package dto;

public record VoorwerpDTO(String naam,double gewicht,int niveau,int kracht,boolean
gebruikt,int aantalInOmloop,int deur,char soort)
{
    public VoorwerpDTO(String naam,double gewicht,int niveau,int kracht,boolean
gebruikt)
    {
        this(naam,gewicht,niveau,kracht,gebruikt,0,0,'W');
    }

    public VoorwerpDTO(String naam,double gewicht,int niveau,int aantalInOmloop,int
deur)
    {
        this(naam,gewicht,niveau,0,false,aantalInOmloop,deur,'S');
    }

}
```

package persistentie:

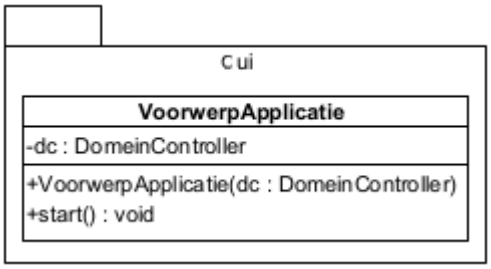


package main:



In de klasse **StartUp** wordt de main-methode uitgewerkt. In deze main-methode starten we de applicatie door de methode start aan te roepen uit de cui-klasse.

package cui:



Schrijf een applicatie waarin de array met Wapens én Sleutels uit de persistentielaag wordt getoond. Werk volgens de correcte 3-lagen werkwijze!

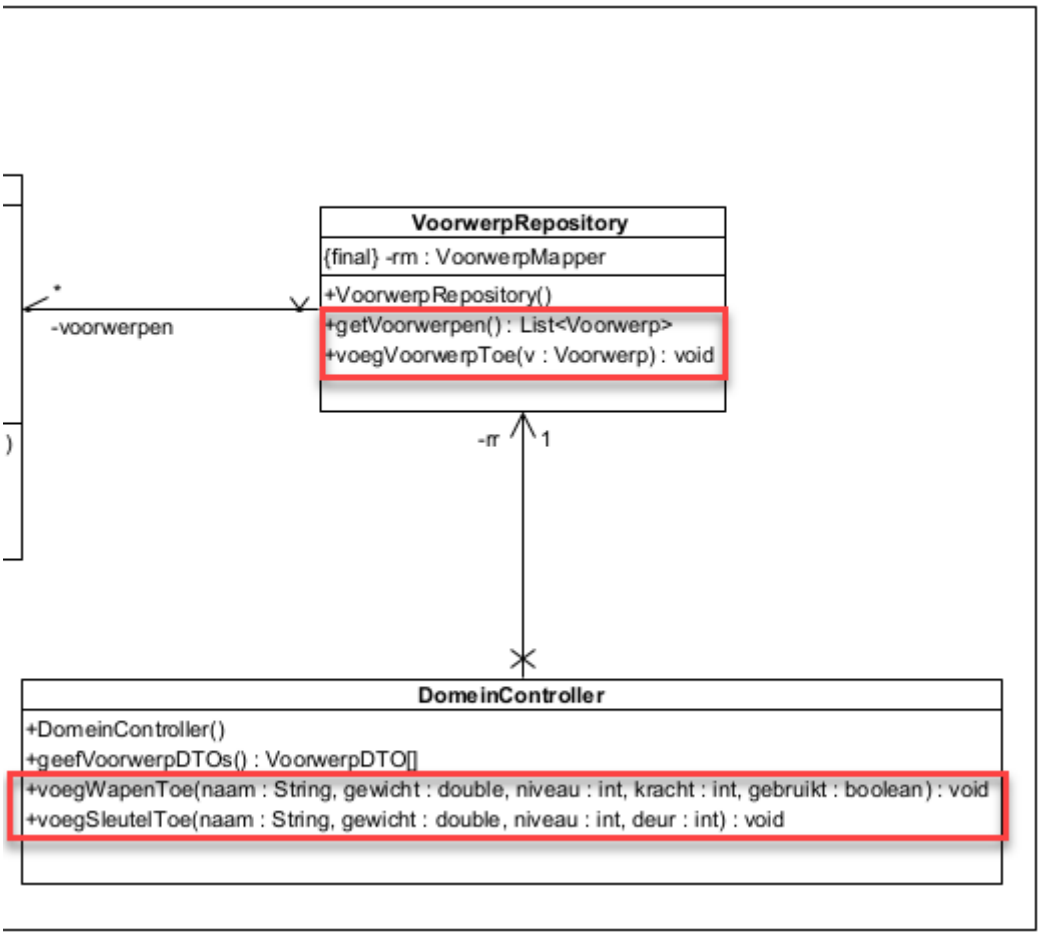
Probeer de uitvoer op dezelfde manier als in onderstaand voorbeeld te tonen.

Mogelijke uitvoer:

```
Wapen colt met gewicht 1,500 kg uit niveau 3 en met kracht 6 nog niet gebruikt.
Wapen brown met gewicht 0,500 kg uit niveau 1 en met kracht 23 al gebruikt.
Sleutel voordeur met gewicht 0,500 kg uit niveau 3 past op deur 1.
Er zijn 2 sleutel(s) in omloop.
Sleutel achterdeur met gewicht 0,500 kg uit niveau 1 past op deur 2.
Er zijn 2 sleutel(s) in omloop.
```

2.7. Spel met voorwerpen - vervolg

We voegen nu aan de DomeinController en de VoorwerpRepository extra methodes toe, die ons zullen toelaten om Voorwerpen toe te voegen.



De klasse **VoorwerpRepository** :

- haalt bij creatie de Voorwerp-objecten nog altijd uit de persistentielaag, maar houdt die nu bij in een **lijst** van voorwerpen.

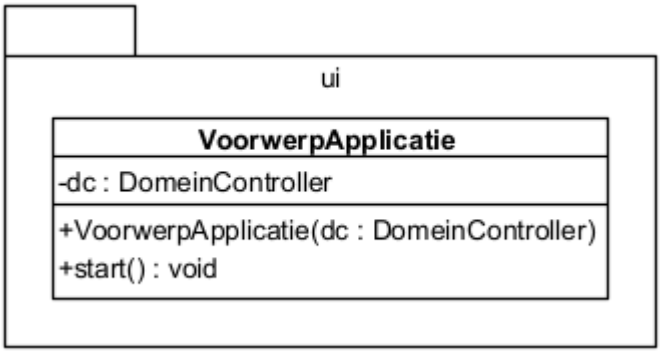
1. Een array kan via **Arrays.asList(naam van de array)** omgezet worden naar een lijst. Het enige nadeel is dat deze nieuwe lijst *statisch* is.
2. Om dit probleem te omzeilen, gebruiken we deze statische lijst als parameter in een nieuwe aangemaakte *dynamische* ArrayList via **new ArrayList<>(Arrays.asList(naam van de array))**

- voegVoorwerpToe met 1 parameter: een voorwerp

De klasse **DomeinController**:

- voegWapenToe met 5 parameters: roept de voegVoorwerpToe-methode aan in VoorwerpRepository
- voegSleutelToe met 4 parameters: roept de voegVoorwerpToe-methode aan in VoorwerpRepository

package cui:



Pas de applicatie aan met code die 2 extra sleutels en wapens in de repository plaatst. Toon de collectie van voorwerpen die in de repository aanwezig is VOOR en NA het toevoegen van deze objecten.

Mogelijke uitvoer:

```
Beginsituatie:
Wapen colt met gewicht 1,500 kg uit niveau 3 en met kracht 6 nog niet gebruikt.
Wapen brown met gewicht 0,500 kg uit niveau 1 en met kracht 23 al gebruikt.
Sleutel voordeur met gewicht 0,500 kg uit niveau 3 past op deur 1.
Er zijn 2 sleutel(s) in omloop.
Sleutel achterdeur met gewicht 0,500 kg uit niveau 1 past op deur 2.
Er zijn 2 sleutel(s) in omloop.


Na het toevoegen van enkele wapens en sleutels:
Wapen colt met gewicht 1,500 kg uit niveau 3 en met kracht 6 nog niet gebruikt.
Wapen brown met gewicht 0,500 kg uit niveau 1 en met kracht 23 al gebruikt.
Sleutel voordeur met gewicht 0,500 kg uit niveau 3 past op deur 1.
Er zijn 4 sleutel(s) in omloop.
Sleutel achterdeur met gewicht 0,500 kg uit niveau 1 past op deur 2.
Er zijn 4 sleutel(s) in omloop.
Wapen Long Rifle met gewicht 1,250 kg uit niveau 4 en met kracht 99 nog niet gebruikt.
Sleutel Garagedeur met gewicht 0,500 kg uit niveau 4 past op deur 3.
Er zijn 4 sleutel(s) in omloop.
Wapen Hagelgeweer met gewicht 0,750 kg uit niveau 1 en met kracht 35 al gebruikt.
Sleutel Tuinhuisdeur met gewicht 0,500 kg uit niveau 10 past op deur 4.
Er zijn 4 sleutel(s) in omloop.
```