HO
GENT

H7 Pijlers van OO - Werkcollege

Table of Contents

1. Doelstellingen per oefening

1.1. Oefening 2.1: Overerving: ontwerp en implementatie

1.2. Oefening 2.2: Overerving: implementatie

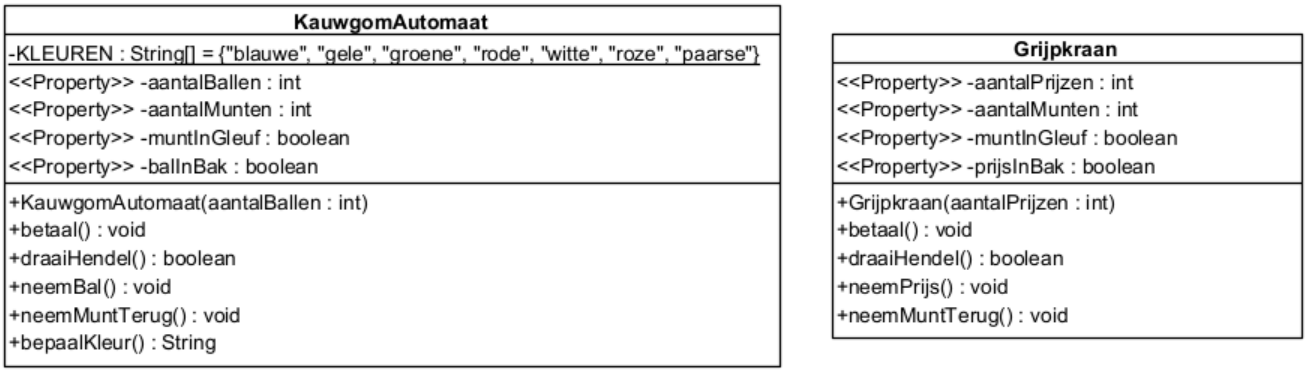
1.3. Oefening 2.3: Polymorfisme: welke toekenningen en methode-aanroepen zijn geldig?

1.4. Oefening 2.4: Polymorfisme: implementatie

2. Oefeningen

2.1. KauwgomAutomaat en GrijpKraan: ontwerp en implementatie

Gegeven volgende UML:



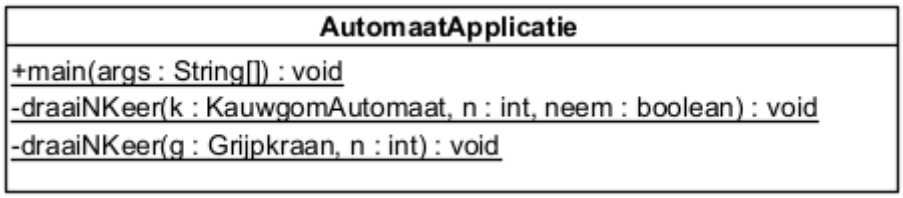
Bekijk waar je in deze UML overerving zou kunnen gebruiken. Maak, indien mogelijk, een superklasse met de gemeenschappelijke kenmerken van deze klasses.

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 array kleuren

Zorg ook voor een implementatie van alle klassen uit je ontwerp en voor een kleine applicatie die voldoet aan volgende UML en voorbeelduitvoer:



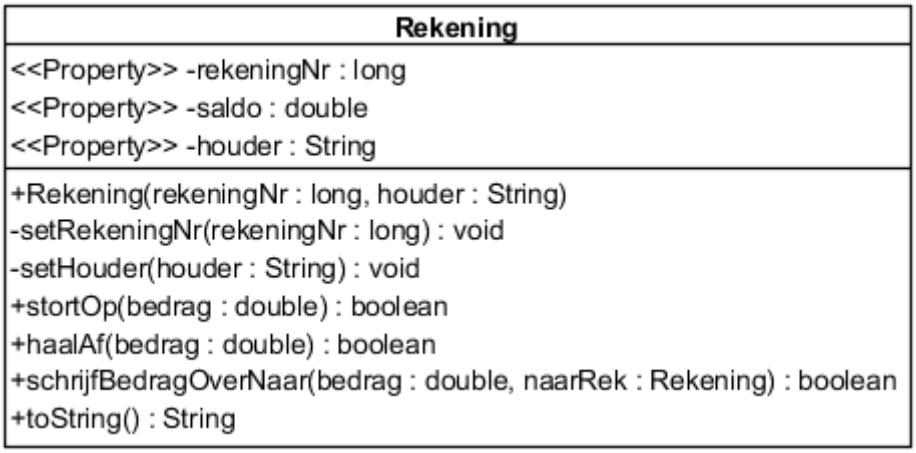
```
10 keer aan de hendel draaien van een kauwgomautomaat met 200 ballen
Poging 1: witte kauwgombal
Poging 2: paarse kauwgombal
Poging 3: witte kauwgombal
Poging 4: blauwe kauwgombal
Poging 5: gele kauwgombal
Poging 6: witte kauwgombal
Poging 7: roze kauwgombal
Poging 8: gele kauwgombal
Poging 9: roze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

2.2. Rekening met subklasses: implementatie

Gegeven: een klasse Rekening (zie startproject H7_WC_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:



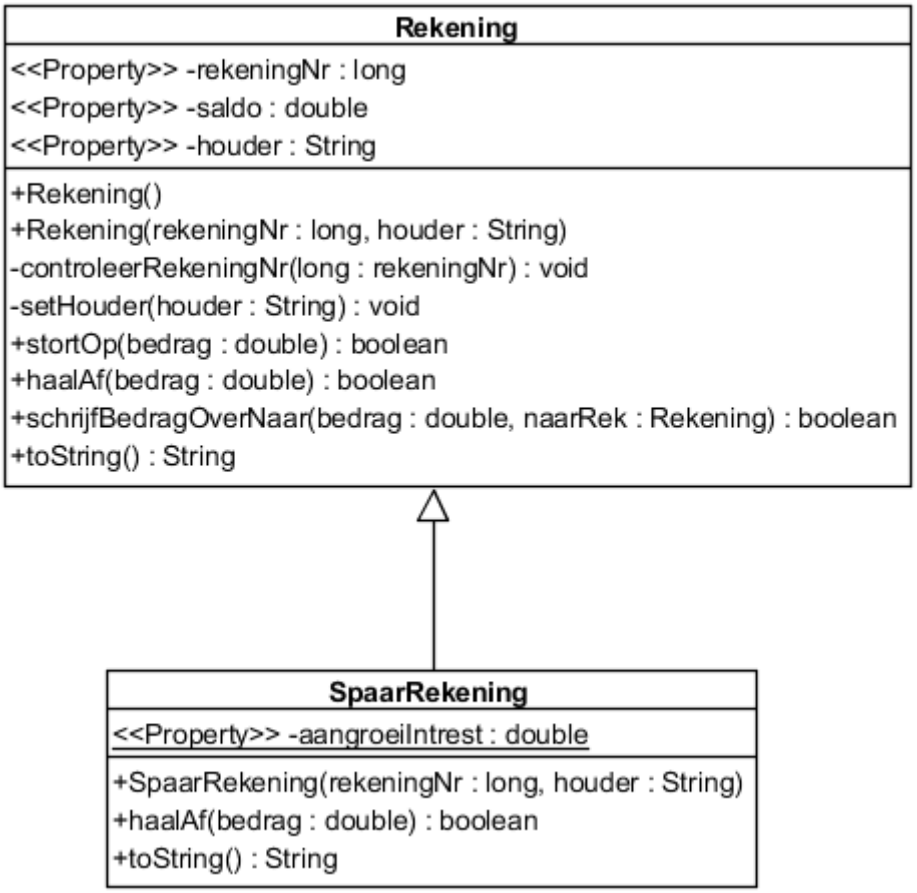
In het startproject zijn ook 3 applicaties gegeven, die bij elk van de onderstaande 3 stappen horen.

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:

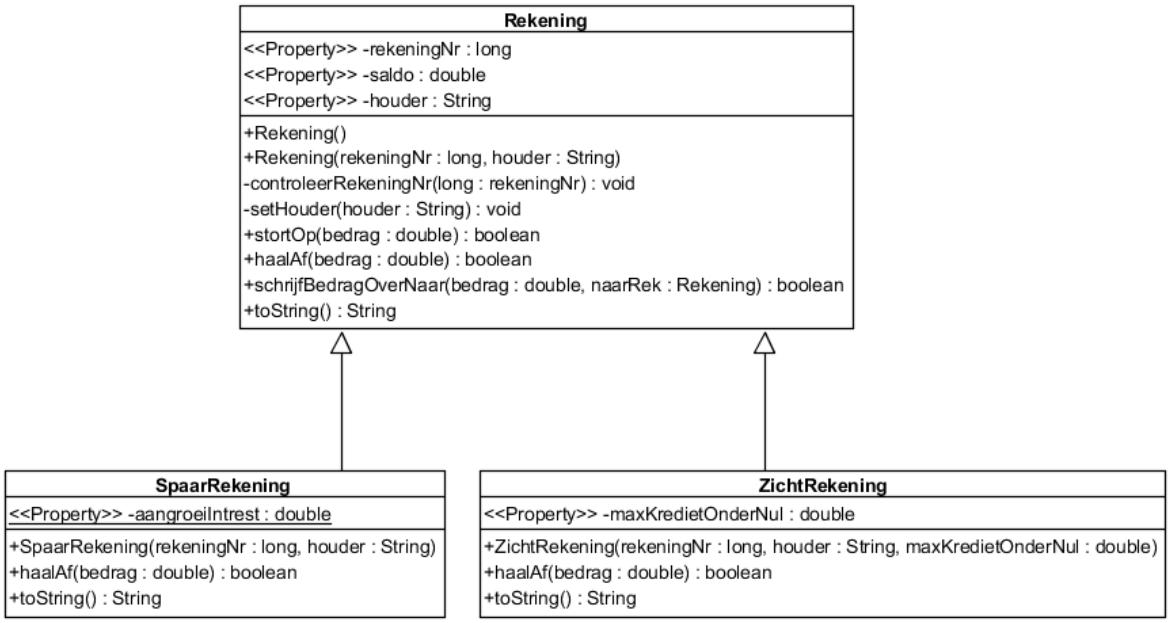
```
SpaarRekening met rekeningnummer 123-1234569-86
staat op naam van piet
en bevat 450,00 euro. Aangroeiintrest = 1,50%
```



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

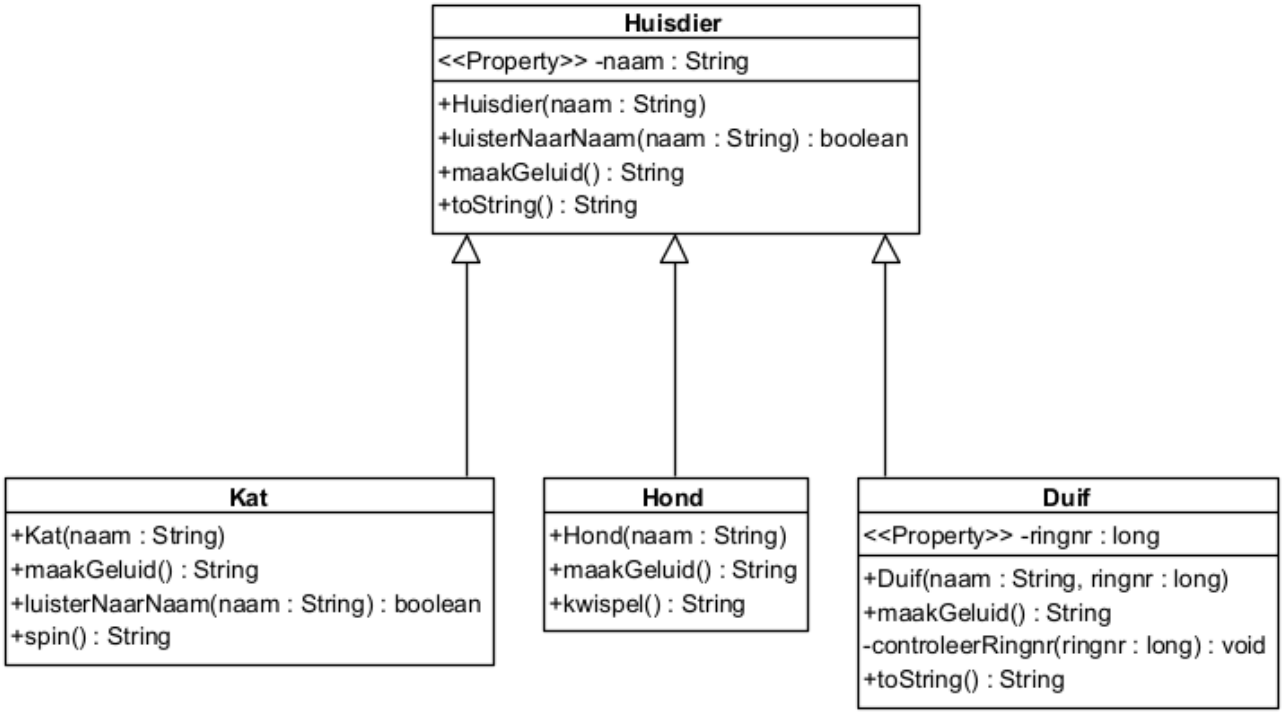
   uitvoer bij het aanroepen de toString-methode:

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



2.3. Polymorfisme: wat is toegelaten?

Gegeven volgende UML:



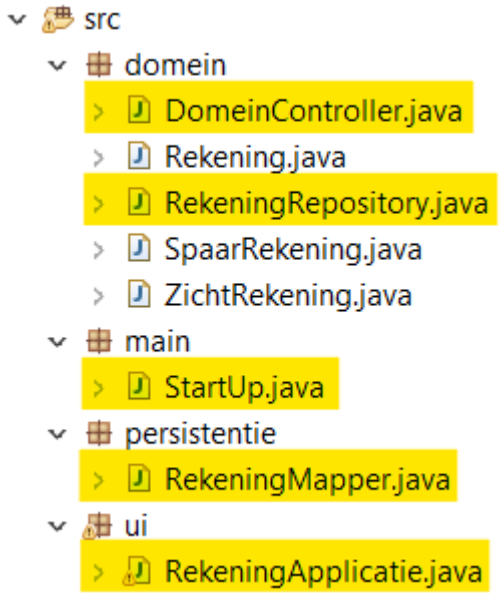
Zijn volgende declaraties en statements juist of fout? Waarom wel/niet?

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

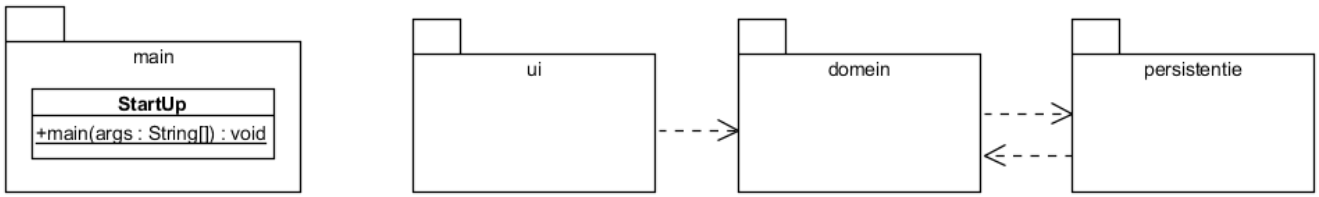
- Huisdier dier = new Kat("Goofie");
  return dier.spin();
- Kat dier = new Kat("Goofie");
  return dier.spin();
- Duif duif = new Hond("Tarzan");
  return duif.kwispel();
- Huisdier dier = new Hond("Tarzan");
  return ((Hond)dier).kwispel();

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.

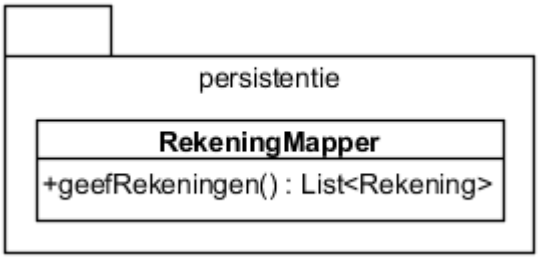


We maken met de bestaande domeinklasses een volledige applicatie met drie lagen.



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 (ui) mag geen gebruik gemaakt worden van objecten!

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

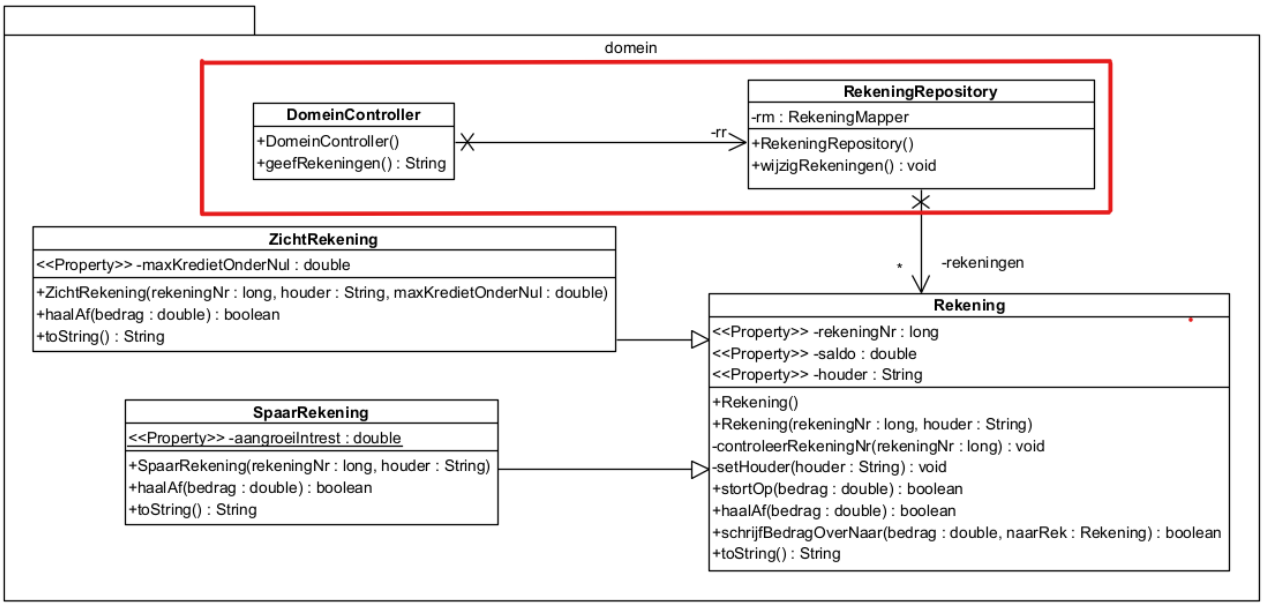


Deze bevat slechts één methode: geefRekeningen, die een lijst van Rekening-objecten teruggeeft. We vullen de lijst op met een Spaar- en ZichtRekening naar keuze, bijvoorbeeld:

```
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 5%.

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. voorzie ook een getter voor de lijst van rekeningen

      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.

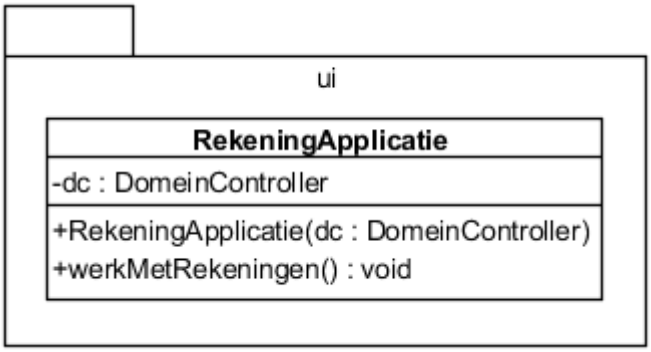
b. De klasse **DomeinController** uit de package **domein**.

i. attributen: rr (type RekeningRepository)

ii. defaultconstructor: maak een object van de RekeningRepository en ken het toe aan het attribuut rr

iii. methode geefRekeningen: zet de lijst van Rekening-objecten na wijzigingen uit de repository om naar een String waarin alle informatie over alle Rekeningen achter elkaar geplakt wordt

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



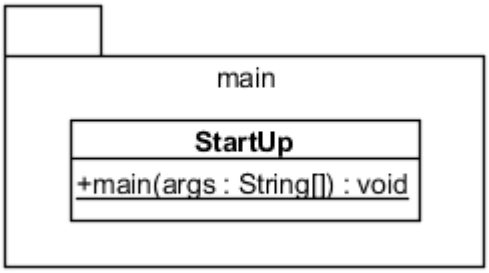
De klasse **RekeningApplicatie** zit in de package **ui** en zal dus alles bevatten qua code die nodig is voor de in- en uitvoer. De klasse moet kunnen communiceren met de domeinlaag en zal daarvoor één object van de DomeinController bijhouden (attribuut dc).

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:

```
SpaarRekening met rekeningnummer 000-0000002-02
staat op naam van Ruben
en bevat 204,00 euro. Aangroeiintrest = 2,00%
ZichtRekening met rekeningnummer 000-0000003-03
staat op naam van Nick
en bevat 300,00 euro. Max krediet onder nul = -60,00
ZichtRekening met rekeningnummer 000-0000004-04
staat op naam van Alexander
en bevat 400,00 euro. Max krediet onder nul = -3296,00
SpaarRekening met rekeningnummer 000-0000001-01
staat op naam van Maxime
en bevat 102,00 euro. Aangroeiintrest = 2,00%
```

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.