HO
GENT

OOSDII

Polymorfisme & Interfaces Werkcollege

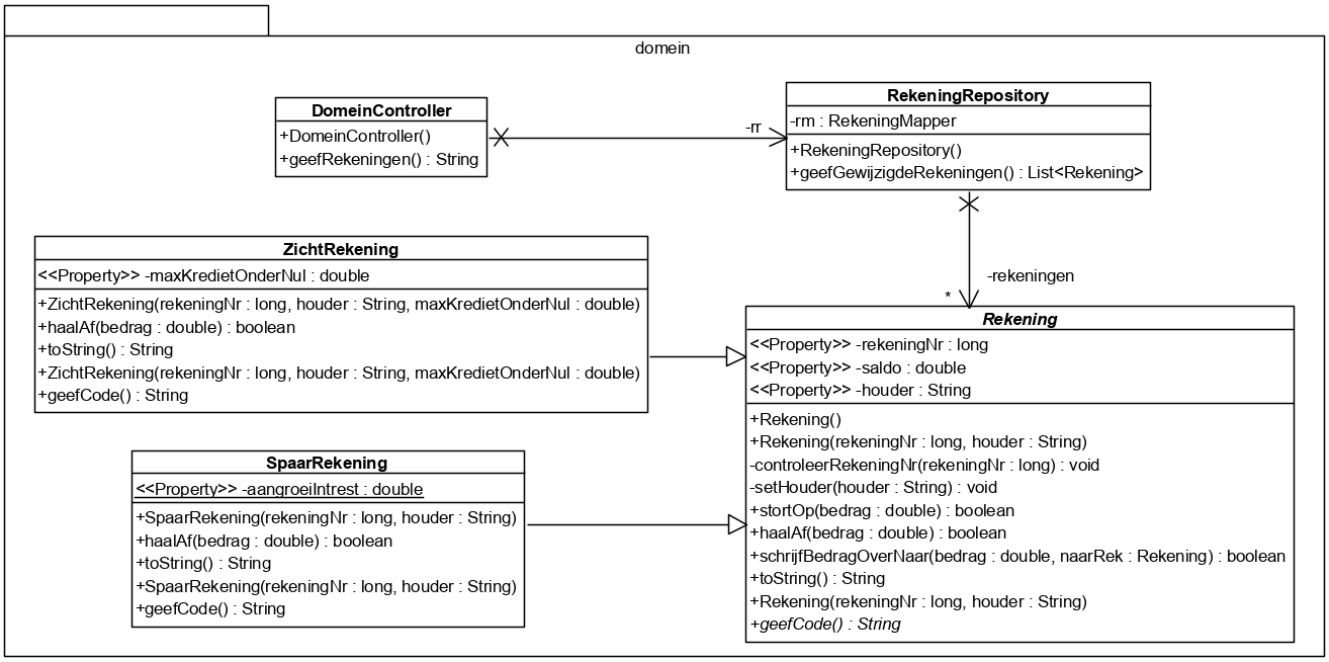
Table of Contents

1. Oefening 1

1.1. Gegeven: Rekening: Rode draad oefening

Gegeven een klasse Rekening (Versie werkcollege OOSDII - Overerving).

De eigenschappen, het gedrag en de relaties van en tussen klassen kan je aflezen uit onderstaand klassediagram:

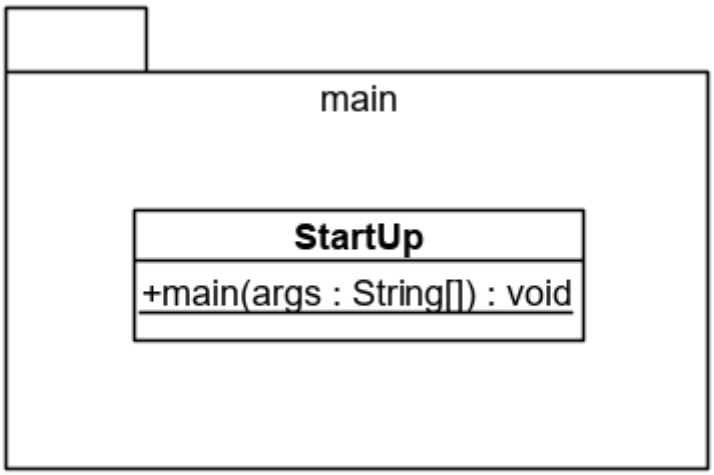


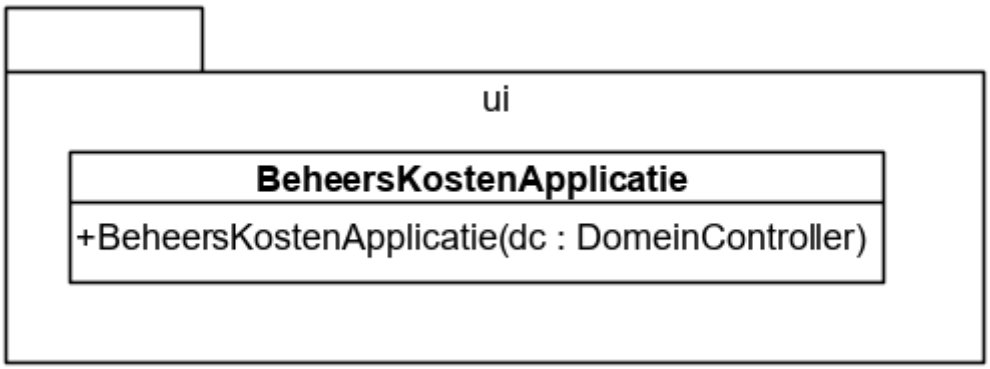
1.2. Gevraagd

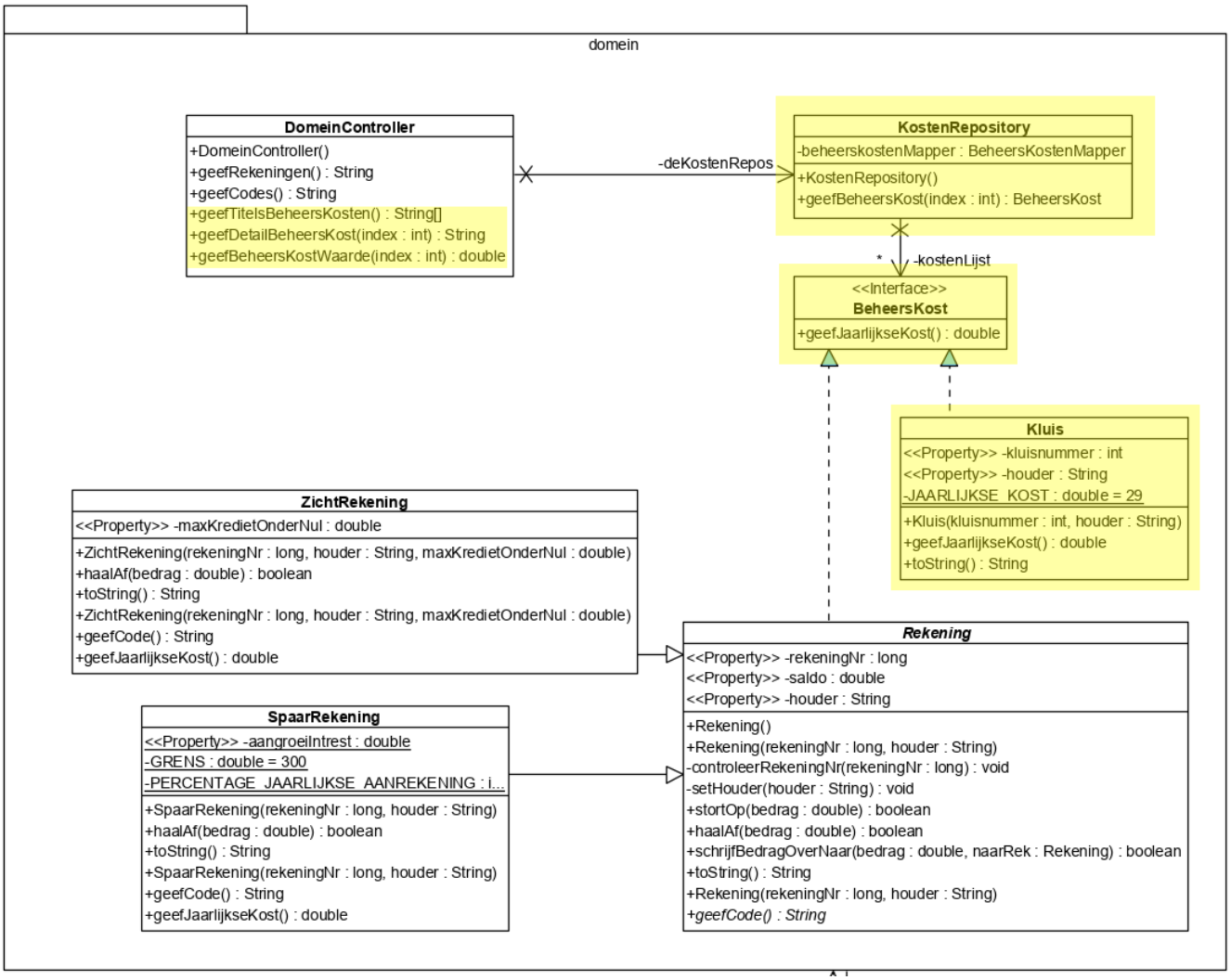
Uitbreidingen:

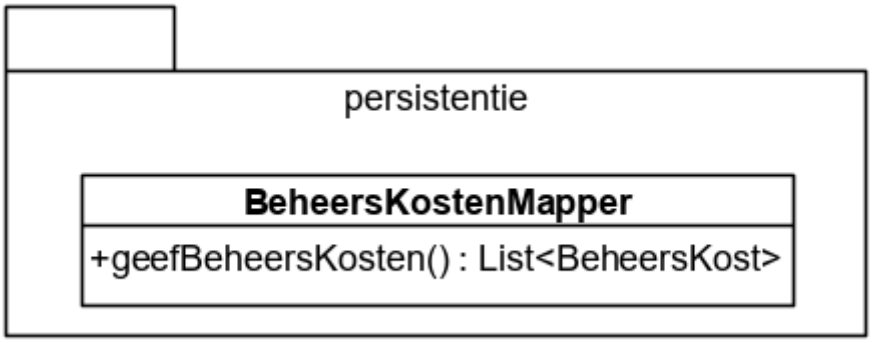
- Er zijn twee verschillende producten die een kost met zich meedragen: Rekening (zowel zicht als spaar) en het huren van een Kluis. Op het eerste zicht hebben Rekening en Kluis niets met elkaar te maken, toch willen we deze objecten op een zelfde manier behandelen…
- We voegen de BeheersKostenMapper toe.

1.2.1. Volledig UML


main
StartUp
+main(args : String[]) : void

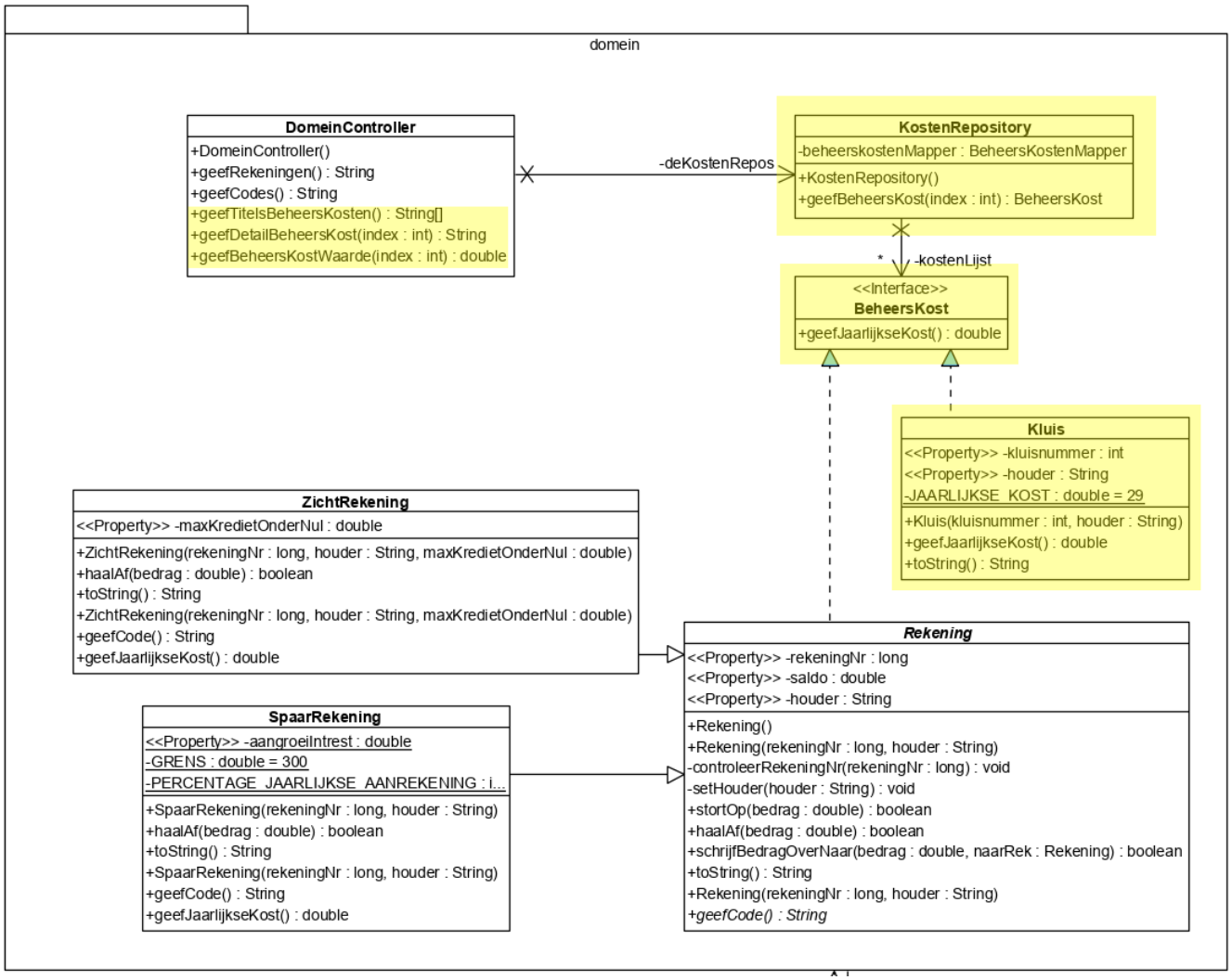

ui
BeheersKostenApplicatie
+BeheersKostenApplicatie(dc : DomeinController)


domein
DomeinController
+DomeinController()
+geefRekeningen() : String
+geefCodes() : String
+geefTitelsBeheersKosten() : String[]
+geefDetailBeheersKost(index : int) : String
+geefBeheersKostWaarde(index : int) : double
-deKostenRepos
KostenRepository
-beheerskostenMapper : BeheersKostenMapper
+KostenRepository()
+geefBeheersKost(index : int) : BeheersKost
* -kostenLijst
<<Interface>>
BeheersKost
+geefJaarlijkseKost() : double
Kluis
<<Property>> -kluisnummer : int
<<Property>> -houder : String
-JAARLIJKSE_KOST : double = 29
+Kluis(kluisnummer : int, houder : String)
+geefJaarlijkseKost() : double
+toString() : String
ZichtRekening
<<Property>> -maxKredietOnderNul : double
+ZichtRekening(rekeningNr : long, houder : String, maxKredietOnderNul : double)
+haalAf(bedrag : double) : boolean
+toString() : String
+ZichtRekening(rekeningNr : long, houder : String, maxKredietOnderNul : double)
+geefCode() : String
+geefJaarlijkseKost() : double
Rekening
<<Property>> -rekeningNr : long
<<Property>> -saldo : double
<<Property>> -houder : String
+Rekening()
+Rekening(rekeningNr : long, houder : String)
-controleerRekeningNr(rekeningNr : long) : void
-setHouder(houder : String) : void
+stortOp(bedrag : double) : boolean
+haalAf(bedrag : double) : boolean
+schrijfBedragOverNaar(bedrag : double, naarRek : Rekening) : boolean
+toString() : String
+Rekening(rekeningNr : long, houder : String)
+geefCode() : String
SpaarRekening
<<Property>> -aangroeiIntrest : double
-GRENS : double = 300
-PERCENTAGE_JAARLIJKSE_AANREKENING : i...
+SpaarRekening(rekeningNr : long, houder : String)
+haalAf(bedrag : double) : boolean
+toString() : String
+SpaarRekening(rekeningNr : long, houder : String)
+geefCode() : String
+geefJaarlijkseKost() : double



1.2.2. Domeinlaag

We starten met de domeinlaag:

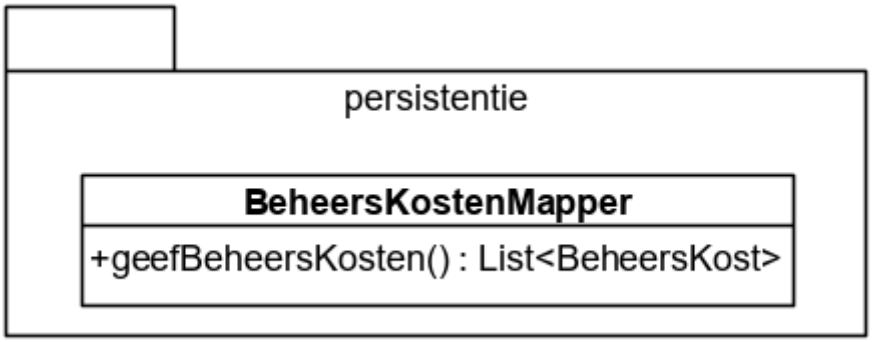


- Interface **BeheersKost**
  - geefJaarlijkseKost
- Klasse **Kluis** (implementeert interface BeheersKost):
  - final attributen: houder(String) en kluisnummer(int)
  - constructor: een kluis-object wordt aangemaakt en al zijn attributen worden opgevuld.
  - getters

    - toString: geeft het kluisnummer en houder als String terug
    - geefJaarlijkseKost: een kluis kost jaarlijks €29 (vaste prijs)
- De bestaande klasse **Rekening** implementeert ook de interface **BeheersKost**. Wat is het gevolg hiervan?
- De jaarlijkse kost van:
    - **Zichtrekening**: 1% van |maxKredietOnderNul|
    - **Spaarrekening**: indien saldo < 300, dan 5% van (300 - saldo) anders 0
- De **KostenRepository** houdt een lijst van BeheersKost-objecten bij. De objecten worden gemaakt in de persistentielaag, meer bepaald in de BeheersKostenMapper.

Werk nu dus eerst verder aan de persistentielaag. Daarna kan dit stuk verder afgewerkt worden.

1.2.3. Persistentielaag



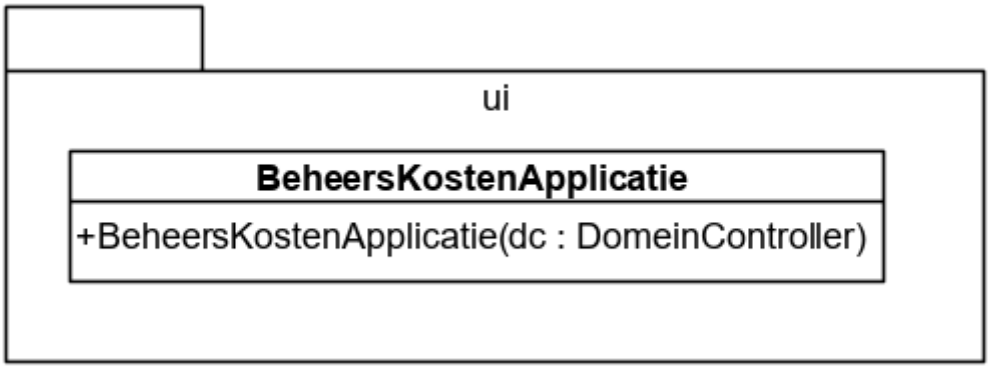
Klasse **BeheersKostenMapper** * De methode geefBeheersKosten(): geeft een lijst terug gevuld met 2 kluizen, 1 zichtrekening en 1 spaarrekening aan de KostenRepository. * De domeinlaag wil een lijst van **BeheersKosten** terugkrijgen en vraagt die aan de BeheersKostenMapper.

1.2.4. Afwerken domeinlaag

Implementeer nu de ontbrekende methodes in de DomeinController:

- Methode geefTitelsBeheersKosten
    - Geeft per BeheersKost een String terug:
    - Kluis: nummer en houder (hebben we daar iets voor?)
    - Rekening: type (spaar of zicht) en houder
- Methode geefDetailBeheersKost(index:int)
    - Geeft een String terug voor deze specifieke BeheersKost:
    - Kluis: nummer en houder
    - Rekening: alle details van de rekening opsommen!
- Methode geefBeheersKostWaarde(index:int)
    - Geeft een bedrag (double) terug. Dit is de jaarlijkse kost voor deze specifieke **BeheersKost**.

1.2.5. Presentatielaag



Maak een applicatie die per **BeheersKost** een titel, de detailinfo en de kost uitschrijft.

De applicatie zelf wordt opgestart via de StartUp-klasse.

```
kluisnr = 100, houder = Tania
Detail: kluisnr = 100, houder = Tania
Kost: 29.0

zichtrekening, houder = Jan
Detail: ZichtRekening met rekeningnummer 123-4567000-82
staat op naam van Jan
en bevat 1500,00 euro. Max krediet onder nul = -2000,00
Kost: 20.0

kluisnr = 250, houder = Steve
Detail: kluisnr = 250, houder = Steve
Kost: 29.0

spaarrekening, houder = Sandra
Detail: SpaarRekening met rekeningnummer 123-4567800-09
staat op naam van Sandra
en bevat 1250,00 euro. Aangroeiintrest = 5,00%
Kost: 0.0
```

2. Oefening 2

2.1. Gegeven Movie - Comparable interface

In het startproject 'OOSDII_Polymorfisme_En_Interfaces_WC_ComparableExample_Start' vind je de klassen Movie en EqualsApp terug.

Laat de klasse Movie de interface `Comparable` implemeteren zodat je twee Movie objecten met elkaar (op natuurlijke wijze) kan sorteren.

Sorteer op basis van hun attributen `name` en `year`: sorteer eerst op basis van het attribuut `name`. Indien dit attribuut identiek is sorteer je op het attribuut `year`.

Let op: hoewel de compiler dit niet kan garanderen gaat men er wel van uit dat, indien twee objecten volgens hun `equal` methode gelijk zijn ze ook volgens de methode `compareTo` gelijk zijn.

Indien een lijst objecten bevat die de interface `Comparable` implementeren, dan kan deze lijst gesorteerd worden door de klasse methode `sort` in de klasse `Collections`. Deze methode is reeds aanwezig in de klasse ComparableApp maar staat in commentaar. Haal ze uit commentaar en beoordeel het resultaat.

```
Movies after sorting :
Empire Strikes Back 8,80 1980
Empire Strikes Back 8,80 2008
Force Awakens 8,30 2015
Return of the Jedi 8,40 1983
Star Wars 8,70 1977
```

3. Oefening 3

3.1. Gegeven Movie - Comparator interface

In het startproject 'OOSDII_Polymorfisme_En_Interfaces_WC_ComparatorExample_Start' vind je de klassen Movie en EqualsApp terug.

Implementeer een nieuwe klasse `RatingCompare` die de interface `Comparator` implementeert. Zorg dat de `compare` methode twee Movie objecten kan sorteren op basis van hun attribuut `rating`. Een hogere `rating` komt voor een lager.

Indien een lijst objecten bevat niet `Comparable` zijn, of je wil objecten op een andere manier sorteren dan op de natuurlijke wijze, dan kan deze lijst gesorteerd worden door de overloaded klasse methode `sort` in de klasse `Collections`. Deze methode verwacht als eerste parameter een lijst object, en als tweede parameter een `Comparator` object. Op basis van deze laatste zal de lijst gesorteerd worden.

Vul de klasse `ComparatorApp` aan zodat de gegeven lijst met Movie objecten gesorteerd wordt op basis van hun rating.

```
Sorted by rating
8.8 Empire Strikes Back 1980
8.7 Star Wars 1977
8.4 Return of the Jedi 1983
8.3 Force Awakens 2015
```

aangezien de klasse `RatingCompare` hoort bij de klasse `Movie` kan je ze ook definiëren als en static geneste klasse binnen de klasse `Movie`. Op die manier is het duidelijker dat beide klassen bij elkaar horen.