Java Essentials

Hoofdstuk 9

Associaties

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Inhoud

1. Inleiding
2. Associaties
3. Aggregaties
4. Composities
5. High cohesion
6. Samenvatting

1. Inleiding

Objecten worden beschreven in klassen
→ variabelen en methoden

Wat is het voordeel van OO-programmeren?
→ Hergebruik van bestaande code

Als een klasse goed gedefinieerd is kan ze in een andere context hergebruikt worden.

Dit doen we door relaties te leggen tussen klassen (en uiteindelijk tussen objecten)

2. Associaties

Associatie

= algemene term voor een relatie tussen klassen of objecten

Het komt erop neer dat het ene object gebruik maakt van de mogelijkheden van een ander object

Vb een Persoon heeft een Adres

een Persoon heeft een geboorteDatum

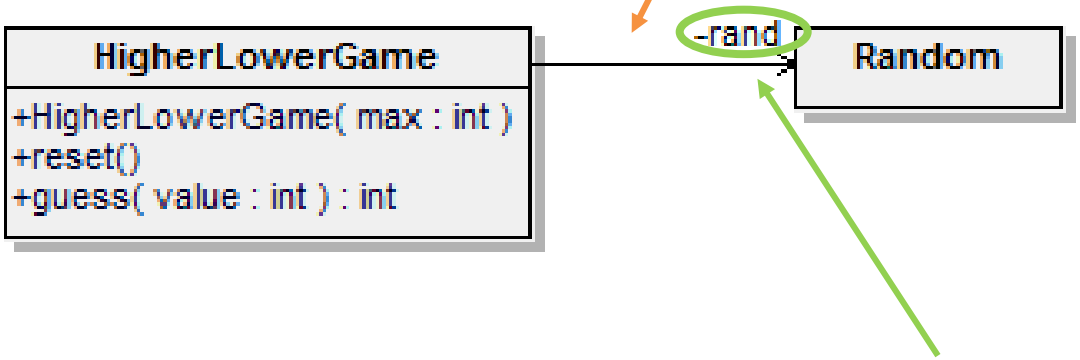
een Boek heeft een Auteur

associatie is een 'HAS A'-relatie

2. Associaties

Voorbeeld: klasse Random

→ Het spel ‘hoger-lager’ maakt gebruik van de klasse Random



→ Het spel ‘hoger-lager’ heeft een instance-variabele van de klasse Random

Voorbeeld: klasse HigherLower

```
package examples.associations;
import java.util.*;

public class HigherLowerGame {
   private int value;
   private int max;
   private Random rand;

   public HigherLowerGame(int max) {
      this.max = max;
      rand = new Random();
      reset();
   }

   public void reset() {
      value = rand.nextInt(max + 1);
   }

   public int guess(int guessValue) {
      if (guessValue < value) {
         return -1;
      } else if (guessValue > value) {
         return 1;
      } else {
         return 0;
      }
   }
}
```

- In de klasse HigherLower is een referentie naar de klasse Random (niet omgekeerd!!)
- Daardoor kunnen we de functionaliteit van die klasse oproepen

2. Associaties

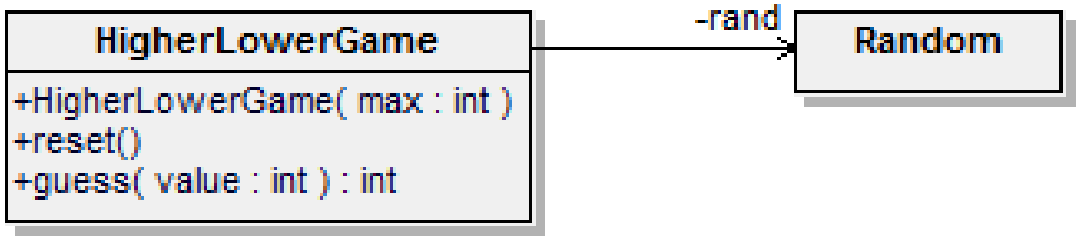
Navigeerbaarheid

= de mogelijkheid om vanuit de ene klasse de andere aan te spreken

Het lijkt alsof we van de ene klasse kunnen navigeren naar de andere.

Deze navigeerbaarheid is meestal in 1 richting (uni-directioneel).

→ In UML-diagram aangeduid met een pijl



3. Aggregaties

Associatie

= ene klasse gebruikt functionaliteit van de andere

Vb. klasse HigherLower gebruikt Random

Aggregatie

→ Relatie tussen 2 klassen is nauwer

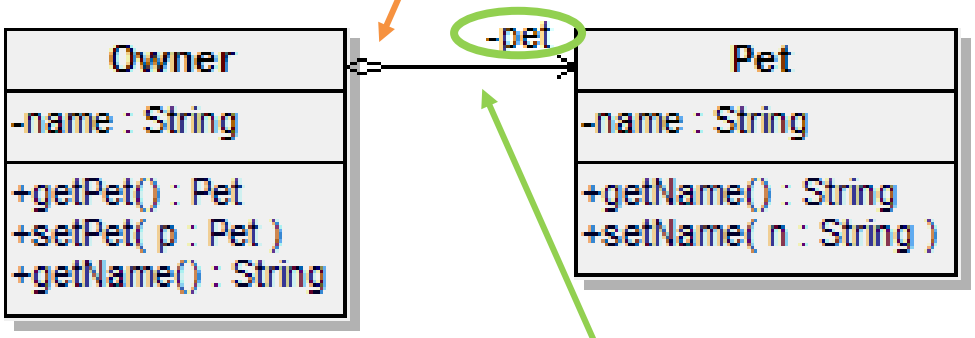
→ Ene object is eigenaar van het andere

→ Is een bijzondere vorm van een associatie

3. Aggregaties

Voorbeeld: klasse Eigenaar en klasse Huisdier

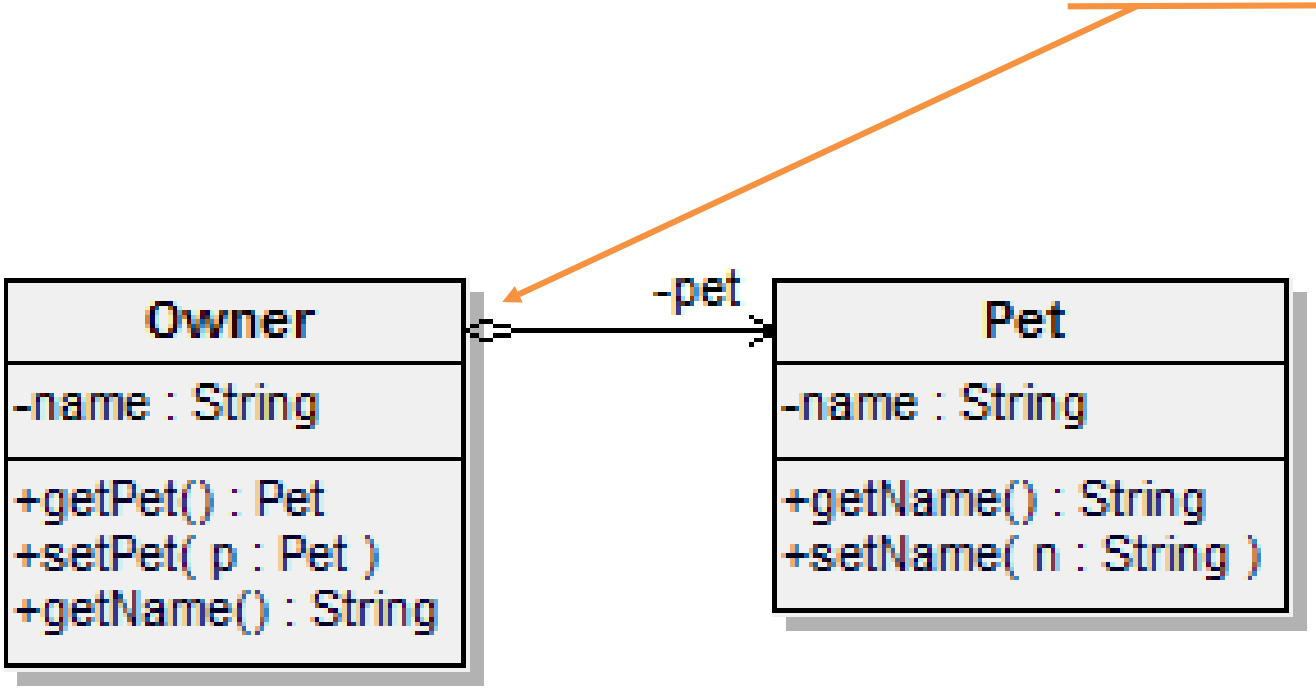
→ Het huisdier is eigendom van de eigenaar



→ De klasse Owner heeft een instance-variabele van de klasse Pet

3. Aggregaties

→ UML-diagram: aggregatie wordt voorgesteld met een open ruit



Voorbeeld: klasse Pet

```
package examples.associations;

public class Pet {
   private String name ="";

   public Pet() {
   }

   public Pet(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

}
```

- **Een Pet heeft als eigenschap een name.**
- **GEEN referentie naar de klasse Owner**

Voorbeeld: klasse Owner

```
package examples.associations;

public class Owner {
   private String name;
   private Pet pet;

   public Owner(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }

   public Pet getPet() {
      return pet;
   }

   public void setPet(Pet pet) {
      this.pet = pet;
   }
}
```

- **Een Owner heeft als eigenschap een name en een Pet.**
- **Ook methode setPet(), want een eigenaar kan van huisdier veranderen!**

3. Aggregaties: belangrijk!

Beide klassen zijn niet onlosmakelijk aan elkaar gekoppeld!

→ Eigenaar kan ander huisdier nemen

→ Huisdier kan van eigenaar veranderen

Daarom is in de klasse Owner een methode setPet()

Zowel de eigenaar als het huisdier kunnen los van elkaar bestaan!

Opdracht 1

1. Maak een project aan met naam “H9”.
2. Maak een package met naam “be.pxl.h9.opdracht1_aggregatie.
3. Importeer in deze package de klassen “Owner” en “Pet”.
4. Maak een object van de klasse Owner en van de klasse Pet (geef het dier onmiddellijk een naam).
5. Koppel het dier aan zijn baas.
6. Zorg voor een afdruk als volgt: Guido heeft een dier met naam Blacky. Je haalt de gegevens van het dier op via het Owner-object.
7. Maak een tweede Owner-object aan.
8. Het huisdier wisselt van baas. Ken het dier toe aan het tweede Owner-object.
9. De nieuwe baas geeft het dier een nieuwe naam.
10. Maak eenzelfde soort afdruk als in item 6.

4. Composities

Compositie

→ 2 objecten zijn onlosmakelijk met elkaar verbonden

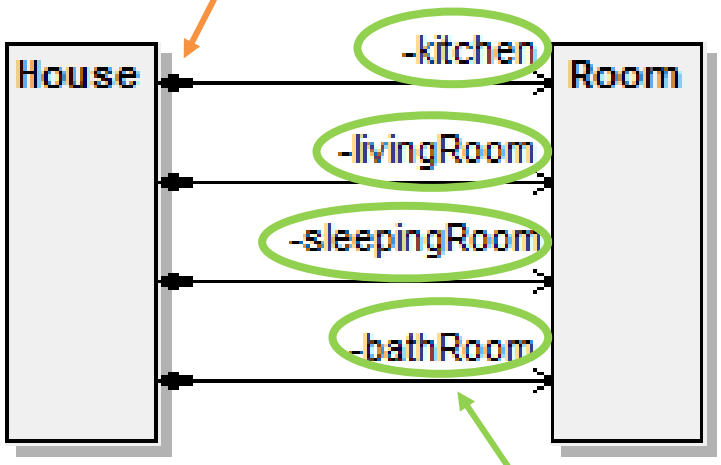
→ Het ene object kan niet bestaan zonder het andere

→ Als het ene ophoudt te bestaan zal het andere ook ophouden te bestaan

4. Composities

Voorbeeld: relatie tussen een huis en een kamer

→ Als het huis wordt afgebroken zal de kamer verdwijnen
(de kamer kan niet bestaan zonder het huis)

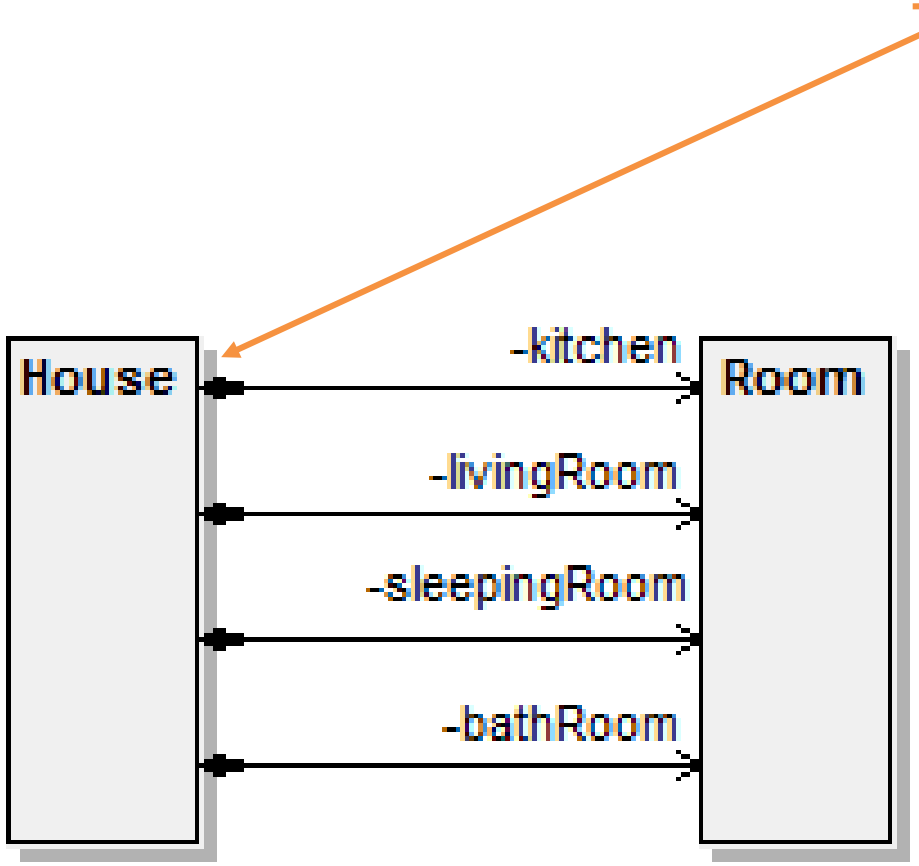


Een huis heeft 4 verschillende kamers:

→ De klasse House heeft 4 instance-variabelen van de klasse Room

4. Composities

→ UML-diagram: compositie wordt voorgesteld met een volle ruit



Voorbeeld: klasse Room

```
package examples.associations;

public class Room {
   private String name;

   public Room(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }
}
```

- **GEEN referentie naar de klasse House**

Voorbeeld: klasse House

```
package examples.associations;

public class House {
   private Room kitchen = new Room("kitchen");
   private Room bathRoom = new Room("bathroom");
   private Room livingRoom = new Room("livingroom");
   private Room sleepingRoom = new Room("sleepingroom");

   public Room getKitchen() {
      return kitchen;
   }

   public Room getBathroom() {
      return bathRoom;
   }

   public Room getLivingRoom() {
      return livingRoom;
   }

   public Room getSleepingRoom() {
      return sleepingRoom;
   }
}
```

- De instanties van de kamers worden gecreëerd op het moment dat het huis gecreëerd wordt
- Er zijn geen methoden om de kamers te wijzigen!

4. Composities: belangrijk!

Beide klassen zijn onlosmakelijk aan elkaar gekoppeld!

→ Kamers worden gecreëerd als het huis gecreëerd wordt

→ Als het huis niet meer bestaat, bestaan ook de kamers niet meer

→ Kamers kunnen niet gewijzigd worden

Daarom zijn in de klasse House geen methoden setRoom()

House en Room kunnen niet los van elkaar bestaan!

Opdracht 2

1. Maak een package met naam “be.pxl.h9.opdracht2_compositie.
2. Importeer in deze package de klassen “House” en “Room”.
3. Maak een object van de klasse House. Merk op dat tegelijkertijd alle kamers gecreëerd worden en dat je die niet kan wijzigen.
4. Maak een afdruk van de “kitchen” en van de “livingroom”.

5. High cohesion

High cohesion

→ Een klasse moet gericht zijn op 1 kerntaak

→ Daardoor kan die klasse later makkelijk hergebruikt worden

Vb. Klasse Random: enkel verantwoordelijk om een willekeurig getal te genereren

Opdracht 3

1. Maak een package met naam “be.pxl.h9.opdracht3.
2. Maak een klasse Auteur met eigenschappen naam en voornaam. Voorzie getters en setters en een toString()-methode die een tekst weergeeft zoals volgend voorbeeld: Auteur: Herman Brusselmans.
3. Maak een klasse Boek met eigenschappen isbn (String), titel, bladzijden en auteur. Voorzie een default constructor waar ook een default auteur gecreëerd wordt. Voorzie eveneens een constructor die alle eigenschappen een waarde geeft. Vermijd dubbele code.
   Voorzie getters en setters voor alle eigenschappen.
   Voorzie een methode toonBoekGegevens() die op een overzichtelijke manier alle informatie van een boek afdrukt.
4. Maak een klasse BoekApp waarbij je een boek creëert via de default-constructor. Ken vervolgens waarden toe aan alle eigenschappen.
   Vraag van je boek de auteursnaam op en druk af.
   Druk vervolgens alle gegevens van het boek af.

6. Samenvatting

Code kan hergebruikt worden door relaties te leggen tussen objecten.

3 soorten relaties:

1. Associaties: ene object maakt gebruik van het andere
2. Aggregaties: ene object is eigenaar van het andere, maar de objecten kunnen afzonderlijk bestaan
3. Compositities: objecten zijn onlosmakelijk met elkaar verbonden

High cohesion:

→ Een klasse moet gericht zijn op 1 kerntaak → voor hergebruik