KU LEUVEN

Objectgerichte Softwareontwikkeling 2021-2022

Professoren: S. Desmet en K. Gerling
Labo-assistenten: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken en J. Wauters.

Labsessie week 11
Overerving

Doel

In deze oefensessie leren we overerving te gebruiken.

Formaat

Elke labosessie is onderverdeeld in drie verschillende delen: startoefeningen, labo-oefeningen en thuisoefeningen. **Startoefeningen** zijn oefeningen waarvan we verwachten dat je ze thuis oplost, vóór de labosessie. Op die manier geraak je vertrouwd met de materie en check je of je de theorie van de hoorcolleges goed begrepen en verwerkt hebt. **Labo-oefeningen** zijn oefeningen die tijdens de labosessie worden behandeld. **Thuisoefeningen** zijn oefeningen die dieper ingaan op de materie. Ze zijn optioneel voor wie sneller werkt, of als extra oefeningenmateriaal bij het studeren/oefenen thuis. Neem contact op met jouw labo-assistent voor feedback op deze oefeningen.

Leerdoelstellingen

Na deze sessie moet je in staat zijn om de volgende concepten te begrijpen en te implementeren:

- Inzien wanneer overerving een toepasbaar abstraheermiddel kan zijn.
- Overerving implementeren in de praktijk.

KU LEUVEN

Objectgerichte Softwareontwikkeling 2021-2022

Professoren: S. Desmet en K. Gerling
Labo-assistenten: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken en J. Wauters.

Startoefeningen

Open het project “StarterExercises” dat je vindt op Toledo.

1. Inspecteer de klasse StaffMember. De klasse houdt een naam bij en of de medewerker al dan niet een vast contract heeft. Maak een object “staffMem1” aan van deze klasse. Inspecteer de attributen van dit object en pas de methodes erop toe.
2. Inspecteer de klasse Teacher. Een object van de klasse Teacher is ook een StaffMember, maar heeft een extra attribuut, dat bijhoudt hoeveel uren hij of zij les geeft. Maak een object “teacher1” aan van deze klasse. Inspecteer opnieuw de attributen en methoden. Merk op dat dit object twee methoden laat zien als je er rechts op klikt, hoewel er wel degelijk méér op van toepassing zijn. De andere vind je onder “Inherited from StaffMember” en zijn inderdaad overgeërfd van StaffMember: elk van de methoden van StaffMember kan je ook toepassen op een Teacher-object.
3. Inspecteer de klasse Desk. Een object van deze klasse heeft een nummer en één of geen StaffMember-objecten die aan dit bureau zitten. Maak een instantie aan van deze klasse. Gebruik de setter voor “staffMemberSittingHere” om staffMem1 plaats te laten nemen aan dit bureau. Vervang hem of haar door teacher1. Waarom lukt dit laatste, hoewel de parameter van de setter van de klasse StaffMember moet zijn?
4. Inspecteer de klasse Course. Een object van deze klasse heeft een naam (String) en één Teacher-object, meegegeven bij constructie. Maak een object aan van deze klasse. Probeer staffMem1 door te geven als parameter. Probeer daarna met teacher1. Hoe verklaar je het verschil?

KU LEUVEN

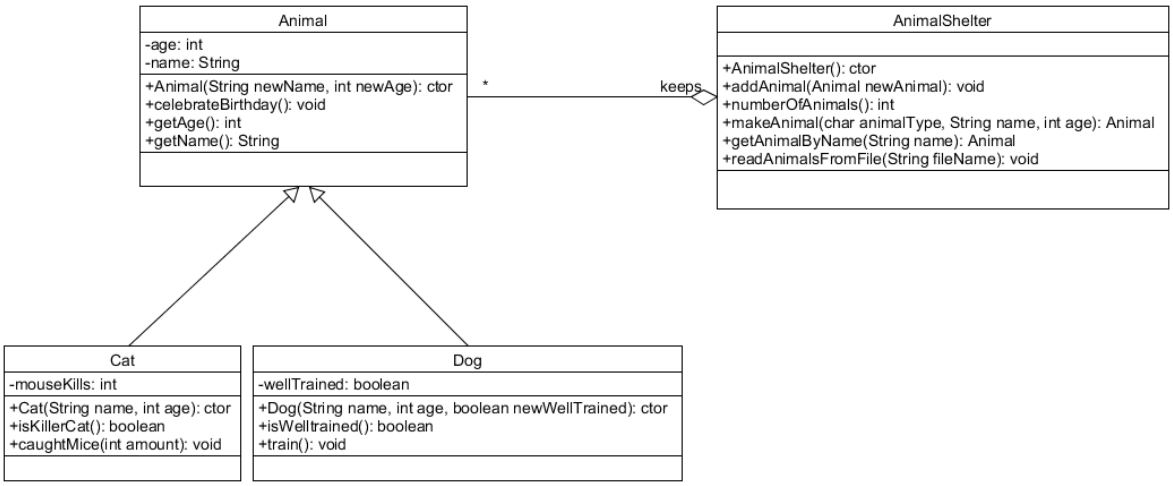
Objectgerichte Softwareontwikkeling 2021-2022

Professoren: S. Desmet en K. Gerling
Labo-assistenten: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken en J. Wauters.

Labo-oefeningen

Oefening 1: Dierenasiel

In deze oefening implementeren we een dierenasiel. Start van het project dat je vindt op Toledo en maak gebruik van de testcode om het volgende UML-diagram te implementeren.



Zoals gewoonlijk: schrijf de code nodig om de test te doen slagen, haal die test uit commentaar, zorg dat die slaagt. Dan ga je verder met de volgende test, et cetera. Pas vanaf test 3 is er overerving in het spel.

Test 1 test de klasse Animal. Een object van deze klasse houdt een attribuut bij voor de naam en leeftijd van het dier, beide doorgegeven bij constructie. Er is een methode om het dier te laten verjaren en er zijn getters voor beide attributen.

Test 2 test de klasse AnimalShelter. Zorg dat je Animal-objecten kan bijhouden in een ArrayList. Er is ook een methode die opvraagt hoeveel dieren het asiel telt.

Test 3 test de klasse Cat. Aangezien veel van de dieren katten en honden zijn, waarvoor we extra functionaliteit willen inbouwen, schrijven we een aparte klasse Cat (en Dog in de volgende test). Aangezien een kat natuurlijk ook gewoon een dier is, kunnen we code-duplicatie vermijden door deze klasse te laten overerven van de klasse Animal. De katten, majestueuze jagers, worden ingezet om muizen te vangen rond het asiel. Een kat heeft een attribuut (zie UML) dat bijhoudt hoeveel muizen ze gevangen heeft (0 bij constructie). Er is een methode om gevangen muizen te registreren en een methode die teruggeeft of de kat al dan niet een *killer cat* is: true indien de kat 5 of meer muizen heeft gevangen, false anders.

Test 4 test de klasse Dog. Ook deze klasse erft over van de klasse Animal. Een object van deze klasse houdt bij of de hond al dan niet welgemanierd is. Bij constructie is een hond, malicieuse creaturen als ze nu eenmaal zijn, niet welgemanierd. Er is een methode om een hond te trainen en een getter voor het attribuut.

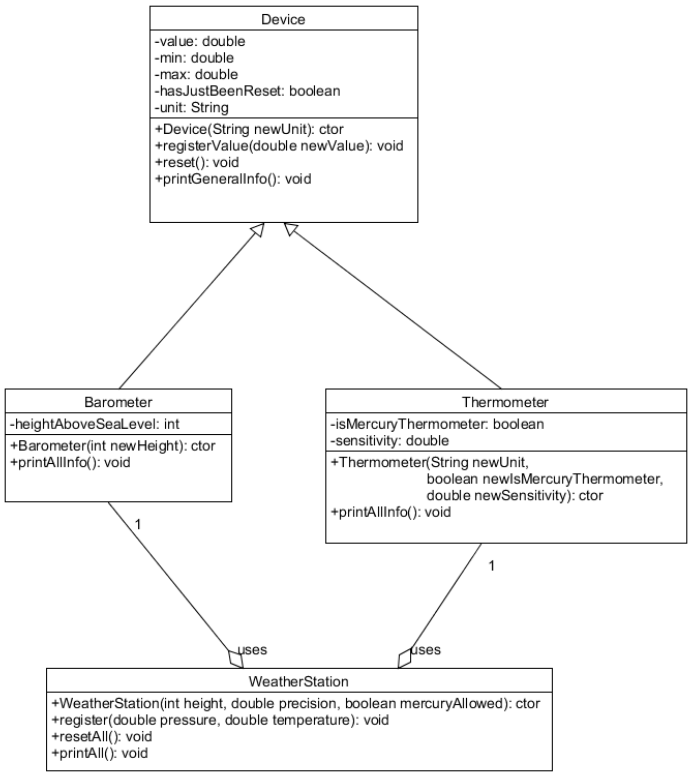
Test 5 test of het asiel een dier kan opzoeken op naam. De methode returnt het gevonden dier, *null* indien het dier niet voorkomt. Je mag ervan uitgaan dat de namen van de dieren uniek zijn.

Test 6 test of het asiel dieren van het juiste type kan aanmaken. De methode *makeAnimal* heeft drie parameters: een *char animalType* die ofwel 'c', 'd' of 'o' is, de naam van het aan te maken dier en de leeftijd. De methode maakt afhankelijk van *animalType* een object aan van respectievelijk de klasse Cat, Dog of Animal (voor andere dieren die geen kat of hond zijn) en returnt dit. Merk op dat het returntype van de methode Animal is. Aangezien Cat en Dog subklassen zijn van Animal, werkt het om een Cat of Dog te returnen *in de rol van* een Animal! In de testcode doen we een *cast* van a1 naar een Cat-object, om te kijken of de "katspecifieke" methode *isKillerCat* correct toegepast kan worden, analoog voor de hond.

Test 7 test of je een asiel kan vullen met dieren uit een bestand, meegegeven als parameter aan de methode. Het bestand bevat, lijn per lijn, drie gegevens over een dier: type dier, naam, leeftijd. Voor katten is het type "cat", voor honden "dog", andere woorden leiden tot een generiek Animal. Hergebruik de methodes die je al geschreven hebt! Als het bestand niet is gevonden, worden er ook geen dieren toegevoegd.

Oefening 2: Weerstation

In sessie 5-6 heb je een weerstation ontworpen, waarbij een weerstation een barometer en thermometer bijhoudt. Je hebt gemerkt dat veel functionaliteit voor beide toestellen gemeenschappelijk is, zoals het bijhouden van een minimum en maximum, printen van de informatie etc. We gaan dit idee verder verfijnen en uitbreiden. Start van de code die je vindt op Toledo.



Inspecteer de klasse WeatherStation. Deze klasse bevat de functionaliteit die de gebruiker toelaten om te interageren met het weerstation (zie UML-diagram hierboven). De gebruiker maakt zelf geen toestellen aan, maar past enkel methoden toe op een instantie van de klasse WeatherStation. Haal stelselmatig de code uit commentaar zodra je in de relevante klassen Barometer en Thermometer de

corresponderende methode hebt voorzien (zie verder). De constructor van WeatherStation heeft als parameters de hoogte van het station boven zeeniveau, de precisie van de thermometer en of dit al dan niet een kwikthermometer mag zijn (zie verder).

Zoals je kan zien in het UML-diagram, is er een klasse Device, die functionaliteit verzorgt die *gemeenschappelijk* is voor barometers en thermometers (bijvoorbeeld bijhouden van de eigenlijke meetwaarde, minimum en maximum). Je krijgt deze klasse. Inspecteer haar methoden.

Schrijf zelf de klassen Thermometer en Barometer. Ze zijn subklassen van de klasse Device. Zij bevatten methoden en functionaliteit die *specifiek* zijn voor een thermometer of barometer. Merk op dat deze structuur makkelijk uit te breiden valt: als we morgen een pluviometer willen toevoegen aan het weerstation, kan deze ook een subklasse zijn van de klasse Device!

De klasse Barometer:

Voor een object van de klasse Barometer instantieer je de variabele *unit* als “hPa”. Aangezien de luchtdruk afneemt met de hoogte boven zeeniveau, houdt een barometer ook een attribuut bij dat aangeeft hoe hoog de barometer hangt boven zeeniveau, dit wordt meegegeven bij constructie. Schrijf de methode die alle relevante informatie print. Maak het een uitdaging om de print *exact* juist te krijgen (ook spaties!). Een voorbeeld. Voor een net-geresette (of net aangemaakte) barometer die op een hoogte van 100m hangt, krijg je de print:

Barometer at altitude 100m: device does not contain meaningful measurements.

Voor een barometer die op 100m hoogte hangt, en waarop achtereenvolgens metingen 1010, 1015 en 1013 zijn doorgegeven, krijg je de print:

Barometer at altitude 100m: 1013.0hPa, minimum was: 1010.0hPa, maximum was: 1015.0hPa.

Tip: schrijf een methode die de relevante informatie print en vervolgens *printGeneralInfo* oproept.

De klasse Thermometer:

Voor een object van de klasse Thermometer wordt de eenheid meegegeven bij constructie: hetzij “°C”, hetzij “°F”. Er wordt bij constructie meegegeven (en opgeslagen in een attribuut) of de thermometer een kwikthermometer is of niet. Verder wordt bij constructie een *sensitivity* doorgegeven (en opgeslagen in een attribuut). Schrijf de methode die de relevante informatie print. Maak het een uitdaging om de print *exact* juist te krijgen (ook spaties!). Een voorbeeld. Voor een net-geresette (of net aangemaakte) thermometer, die geen kwikthermometer is en met gevoeligheid 1, krijg je de print:

Thermometer with sensitivity 1.0: device does not contain meaningful measurements.

Voor een kwikthermometer in °C, met gevoeligheid 0.1, en waarop achtereenvolgens metingen -5, -1 en -3 zijn doorgegeven, krijg je de print:

Thermometer with sensitivity 0.1: -3.0°C, minimum was: -5.0°C, maximum was -1.0°C.

Caution: mercury thermometer – handle with care!

Dat laatste zinnetje wordt dus niet geprint als het geen kwikthermometer betreft.

KU LEUVEN

Objectgerichte Softwareontwikkeling 2021-2022
Professoren: S. Desmet en K. Gerling
Labo-assistenten: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken en J. Wauters.

Oefening 3: UML-diagram ontwerpen

Modelleer het volgende scenario in een klassediagram. Denk na over welke methodes je (minstens) nodig hebt om de gevraagde functionaliteit te implementeren. Modelleer, schrijf dus (nog) geen code, maar gebruik je object-georiënteerde codeerervaring om de voorzien wat je nodig hebt qua design/attributen/methoden!

Een plantenzaak wil een ICT-systeem uitdokteren zodat men kan bijhouden welke planten ze in hun voorraad hebben, en in welke hoeveelheid. Een plant heeft een naam, een bepaalde prijs en een gewicht. Hou ook bij of de plant licht- of schaduwminnend is. Daarnaast heeft iedere plant ook een lijst van chemische stoffen waar ze niet tegen kunnen. Nu heeft de plantenzaak ook een aparte stock met een aantal giftige planten, die chemische stoffen vrijgeven waar sommige andere planten niet tegen kunnen. Elke giftige plant produceert maar één chemische stof. Zorg ervoor dat de zaak een lijst kan printen voor de medewerkers, die voor elke plant aangeeft naast welke giftige planten ze niet mogen staan, en waarom. De giftige planten zijn zelf resistent voor alle chemische stoffen. Bijvoorbeeld:

> Plant a mag niet naast plant x staan, want allergisch aan stof S.
>
> Plant a mag niet naast plant y staan, want allergisch aan stof T.
>
> Plant b mag niet naast plant x staan, want allergisch aan stof S.
>
> …

De zaak kan ook een lijst printen van ofwel alle licht- of schaduwplanten, zowel voor de giftige als niet-giftige planten. Ook moeten planten toegevoegd kunnen worden aan de gewone stock (die van de giftige planten houden we constant).

Vermijd codeduplicatie (eens je dit zou implementeren)!

KU LEUVEN

Objectgerichte Softwareontwikkeling 2021-2022

Professoren: S. Desmet en K. Gerling
Labo-assistenten: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken en J. Wauters.

Thuisoefeningen

Oefening 1: Meerdere niveaus

1. De KU Leuven maakt in haar administratie gebruik van een klasse Course, die cursussen voorstelt. Een cursus heeft een naam en een geheel aantal studiepunten. Schrijf verder een klasse Student die een ArrayList bijhoudt van Course-objecten. Zorg dat je een cursus kan toevoegen, namen van cursussen van een student kan printen etc.
2. Groep T wil voor cursussen graag een boolean toevoegen om aan te geven of deze in het eerste of tweede semester plaatsvindt[1], maar krijgt geen toestemming om dit attribuut toe te voegen. Los dit op door een subklasse GroupTCourse van Course te schrijven.
3. Groep T wil nog verder onderscheid kunnen maken met eerstejaarsvakken: een GroupTCourse houdt een lijst bij van FirstYearGroupTCourse-objecten, namelijk de eerstejaarsvakken waarvoor een student moet slagen om tot dit vak toegelaten te worden. Een FirstYearGroupTCourse zelf heeft natuurlijk een lege lijst als volgtijdelijkheden. Vermijd codeduplicatie: pas nogmaals overerving toe.

Oefening 2: Plantenzaak

Implementeer het UML-diagram dat je hebt ontworpen in oefening 3.

Oefening 3: Overerving binnen Javaklassen

Bekijk opnieuw de code die je geschreven hebt in oefening 1 van de labo-oefeningen. Allicht had je iets als dit:

```
try
        {
          Scanner sc = new Scanner(new File(fileName));
          while (sc.hasNext())
          {
            //…
          }
          sc.close();
        }
        catch (FileNotFoundException fnfe)
        {
          System.out.println(fnfe);
        }
```

[1] We negeren even jaarvakken…

Pas het catch-blok aan naar dit:

```
catch (Exception e)
{
   System.out.println(e);
}
```

Waarom kan je dit doen? Tip: bekijk de documentatie van de FileNotFoundException-klasse.