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 5 en 6
Communicatie tussen objecten

Theorie
Theorie die je vóór deze labosessie moet verwerkt hebben, heb je gezien in het hoorcollege van week 2: interactie tussen objecten.

Doel
In het verleden waren wij degene die objecten maakten en methodes aanriepen. Zo werken andere programma's natuurlijk niet. Een simpele klik in een programma kan duizenden objecten creëren die methodes op elkaar aanroepen. In dit lab leren we hoe we objecten kunnen maken vanuit objecten. Daarnaast zullen we complexere methodes op deze objecten aanroepen om hun toestand te veranderen of we zullen de return-values gebruiken om de toestand van andere objecten te veranderen. We hebben een voorproefje gezien van de communicatie tussen objecten in de testklassen van week 4, waarbij een methode in de klasse Barrel een parameter van de klasse Barrel binnenkreeg, om vloeistof over te gieten.

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:

- Objecten aanmaken en hun methodes oproepen vanuit andere klassen
- Nullpointer exceptions begrijpen
- Method visibility begrijpen
- Simpele controlestructuren gebruiken om objecten te manipuleren

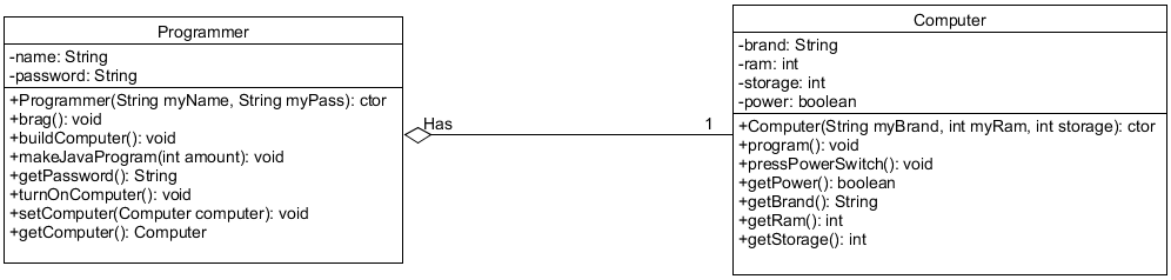
Objectgerichte Softwareontwikkeling 2021-2022

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

Oefening 1: The Fellowship of the Computer

Open het FellowShipOfTheComputer Project van Toledo. Bekijk in de code eerst de klasse Computer en daarna de klasse Programmer. Het UML-diagram vind je hieronder.



Let op het volgende:

1) Programmer heeft een instantievariabele genaamd Computer myComputer. Maar deze instantievariabele is niet expliciet aanwezig bij de instantievariabelen in de UML. Dit wordt in het UML-diagram weergegeven met de aggregatie 

. Dit geeft aan dat Programmer een referentie heeft aan één object van de klasse Computer. Dit betekent dat objecten van de Programmerklasse een verwijzing hebben naar een object van de computerklasse en de **public** methodes van dat object kan gebruiken.
2) Verwijzingen naar objecten is één van de belangrijkste bronnen van fouten tijdens het programmeren. Als we methodes aanroepen op objecten die null zijn (*nullpointer*, dus geen verwijzing), krijgen we een NullPointerException (soms ook aangeduid als “NPE”). Maak een object van Programmer. Zoals we kunnen zien in de constructor, is de myComputer instance variabele standaard null. Inspecteer de brag()-methode en roep deze aan. Wat gebeurt er?

3) Het terminalvenster vertelt ons dat er een NullPointerException was. Het is zelfs zo vriendelijk om ons te vertellen waar het probleem is: regel 31. De regel wordt ook in het blauw aangegeven in de BlueJ-editor. Kijk naar de methode buildComputer(). Voer deze methode uit en voer dan brag() uit. Zijn er nog steeds fouten?

4) Er zijn andere manieren om objecten te linken. Maak een Computer- en een Programmerobject. Verbind ze dan allebei met de setComputer()-methode van de Programmer-instantie. Voer brag() uit om te zien of je geen fouten krijgt.

Oefening 2: The Two Programmers

Tot nu toe waren al onze methoden public en waren alle instantievariabelen private. Public methodes en variabelen kunnen door elk object worden opgeroepen, ook buiten de klasse. Private methodes en variabelen kunnen alleen door de klasse zelf worden gebruikt. Kijk maar eens naar de Programmer Testclass. Het is logisch dat we niet willen dat ons wachtwoord voor iedereen toegankelijk is. Stel de toegangsmodifier van de getPassword()-methode in op private, compileer en probeer de test opnieuw uit te voeren. Probeer een object van de klasse Programmer te maken en probeer de methode zelf uit te voeren. Beide zouden niet mogelijk moeten zijn. Het maken van private methodes zal ons in staat stellen om gebruikers of andere objecten te "blokkeren" van het gebruik van methodes die alleen voor intern gebruik zijn.

Oefening 3: Return of the Value

Nu we weten hoe we referenties kunnen gebruiken om andere objecten te gebruiken, kunnen we meer gebruik maken van return-waarden. Kijk eens naar de brag()-methode. We kunnen in regel 31 zien dat we een if-statement gebruiken dat als eerste getRam() van het myComputer-object uitvoert, de waarde ophaalt, en controleert of die groter is dan 4. De turnOnComputer()-methode gebruikt ook “getter”-methodes om te weten te komen of de computer al aan staat of niet. Dit is een eenvoudig voorbeeld van hoe we de gereturnde waarden kunnen gebruiken om logica in objecten te implementeren, iets wat je in de komende labsessies zal moeten gebruiken.

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: The FellowShip Of The Computer verbeteren

Open het FellowShipOfTheComputer project opnieuw en implementeer de volgende functionaliteit:

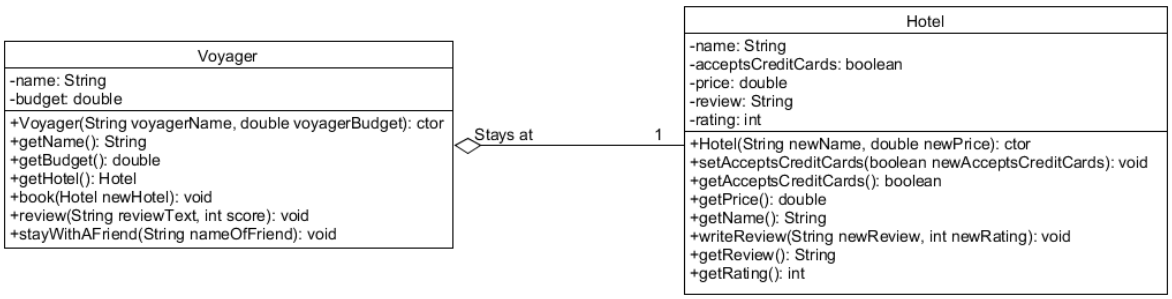
1) Maak de brag() methode nullpointerexception-veilig. Print "Ik heb nog geen computer". Als er geen computer is ingesteld.
2) Maak een methode turnOffComputer() die de computer uitzet als deze ingeschakeld is. Verifieer je functionaliteit door te kijken of de Boolean in het computerobject verandert.

Oefening 2: Reiziger

Dit is een oefening in test-driven design. Je vertrekt van een klassendiagram, enkele extra vereisten en een testklasse die je terugvindt in VoyagerStart op Toledo. Het is aan jou om de klassen Voyager en Hotel te implementeren. Je mag niets veranderen in het schema, zelfs de namen niet. Je kan echter zoveel variabelen en hulpmethoden toevoegen als je wilt. Werk op een gestructureerde manier: lees test 1, begrijp wat je moet doen, implementeer de code, haal test 1 uit commentaar en voer alle testen uit. Na test 1 ga je verder met test 2 en zo voort.

Probleemomschrijving:

Je werkt in dit project met twee klassen: Voyager (reiziger) en Hotel. Het idee is dat een reiziger een reis maakt en daarbij een hotel kan boeken. Na creatie van een Voyager-object is er nog geen Hotel aan hem of haar gekoppeld. We gaan ervan uit dat de reiziger een weekendtripper is en dus één nacht blijft. De reiziger heeft wel een budget waar hij of zij niet over wil gaan. Je zal dus een hotel pas aan een reiziger kunnen koppelen als het binnen het budget past. Er is echter een uitzondering: als het hotel kredietkaarten toelaat, kan iedere reiziger, los van zijn of haar budget, dit hotel boeken. Standaard aanvaardt een hotel geen kredietkaarten. De reiziger kan ook een review geven aan het hotel. In deze primitieve versie houden we geen “geschiedenis” bij van reviews: de laatste reiziger bepaalt de score van het hotel. De review zal bestaan uit een geschreven tekst (String) en een score (int tussen 0 en 5). Een review met een score die hier niet binnen ligt wordt genegeerd. Als laatste functionaliteit die je toevoegt, sta je een reiziger toe om “vals te spelen”: de reiziger kan bij een kennis verblijven in plaats van in een hotel. Dan maakt de reiziger zelf een “hotel” aan met de naam van zijn of haar kennis. De prijs van dit verblijf is altijd €5 (voor een klein presentje).



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: Weerstation

Er is geen startproject voor deze oefening. Maak zelf een nieuw project aan en test de functionaliteit zelf uit. Vraag je labobegeleider om uitleg indien nodig.

Probleemomschrijving:

Een weerstation bevat twee meettoestellen: een thermometer en een barometer. De temperatuur wordt geregistreerd in °C en de luchtdruk in hectoPascal. De thermometer en de barometer registreren niet alleen de huidige waarde, maar onthouden ook het maximum en minimum van de geregistreerde waarden. Definieer de klasse(n) en het objectdiagram van je oplossing.

Wanneer je een weerrecorder (thermometer/barometer) maakt, worden alle waarden geïnitialiseerd tot zinvolle waarden. Schrijf in het weerstation een methode om nieuwe metingen te registreren (een methode die tegelijkertijd temperatuur en druk registreert). In werkelijkheid wordt deze waarde gegeven door de sensor, we simuleren dit door de waarden te in te geven. Het is ook mogelijk om het weerstation te “resetten”. Op dat moment worden alle extremen vergeten, dus na het verstrekken van een eerste meting moeten beide waarden, min en max. gelijk zijn.

Geef ook een methode om een overzicht te krijgen van de extremen.

Dit voorbeeld over objectcommunicatie is een situatie waarbij het 'geheel' (dat wil zeggen het weerstation) verantwoordelijk is voor het maken van alle onderdelen. De constructeur van het 'geheel' creëert dus instanties van de onderdelen.

[Uitdaging] Druk de waarden af met hun eenheden (bijv. °C en hPa). Maak het mogelijk om dit voor elke eenheid te doen, vermijd code-duplicatie.

Oefening 4: RFID Tags

Start van het RFIDTag-project van Toledo. Er is een testklasse beschikbaar om je functionaliteit te testen Er zijn twee lege tests, je zal deze zelf moeten implementeren aan de hand van de beschrijving in het commentaar.

Probleemomschrijving:

We willen de betaling van een winkelwagen simuleren met behulp van RFID-tags. Elk product in de winkelwagen is voorzien van een RFID-tag. Elke tag heeft een ID, een prijs en een betaalstatus (betaald of niet). Wanneer een RFIDTag door de scanner gaat, controleert de scanner eerst of hij deze tag kan verwerken. Om dit te weten heeft de scanner een reeks geldige ID's ([minimum, maximum]). Zorg ervoor dat wanneer je een scanner aanmaakt, het minimum kleiner is dan het maximum. Wanneer dit niet het geval is, wissel dan beide waarden. Wanneer de gescande tag in het bereik ligt, verhoogt de scanner zijn aantal gescande producten en werkt de totale te betalen prijs bij. De RFIDScanner moet

ook een overzicht kunnen genereren van het aantal gescande tags en hun totale prijs. Beide klassen hebben natuurlijk de nodige constructors, setters en getters.

In dit voorbeeld heb je de situatie dat één object (de scanner) achtereenvolgens moet kunnen communiceren met verschillende objecten van het type Tag. Je moet dus in staat zijn om de link tussen RFIDScanner en RFIDTag in de loop van de tijd te wijzigen.

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: Auto

In deze oefening ga je oplossingen uit vorige sessies integreren. Probeer je bestaande code zoveel mogelijk te hergebruiken. Er is geen reeds bestaand project, maak dat zelf aan.

Probleemomschrijving:

We willen een ritje met een auto programmeren. Onze auto is uitgerust met een brandstoftank met een bepaalde inhoud en heeft een bepaald brandstofverbruik uitgedrukt in zoveel liter per 100 km. Je dient zelf de benodigde klasse-eisen van Auto te definiëren. De brandstoftank is de Barrel van sessie 3-4. De auto zelf kan je hergebruiken uit sessie 2.

Als je een rit simuleert, wordt de inhoud van de brandstoftank geüpdatet en wordt de kilometerteller verhoogd. Bovendien verandert de x-coördinaat van je auto. Als er niet genoeg brandstof is, kan je auto natuurlijk eerder stoppen (of zelfs helemaal niet rijden). Een negatieve afstand verschuift je weergave naar links op de x-as, maar ook dan verbruik je wel degelijk brandstof.