GRADUAAT PROGRAMMEREN

Object Oriented Design

Oefeningenboek

1.0

Versie

Wim Vanden Broeck

Auteur(s)

Academiejaar 2022-2023

HISTORIEK

Datum	Versie	Omschrijving	Auteur
09/04/2022	1.0		Wim Vanden Broeck

VOORAF

Dit document maakt gebruik van de volgende stijlen en conventies.

Bijkomende informatie.

Idee, voorbeeld of tip.

Waarschuwing, belangrijke info.

Verdieping.

Vraag aan de lezer.

Doelstelling.

Oefening, taak of opdracht voor de lezer.

Bijkomend materiaal of referentie.

INHOUD

1 Inleiding

Dit oefeningenboek bevat oefeningen voor de les Object Oriented Design. Je kan je kennis testen op basis van praktische opdrachten. De oplossingen van al de opdrachten vind je onder de titel *Oplossingen*.

Soms hebben oefeningen uitgewerkte solutions. De naam van de solution en het project staan steeds bij de oefeningen/oplossingen vermeld. Je vind deze terug onder **Cursus Downloads > Solutions**.

Soms werken we vanaf een bestaand project. In dit geval ga je dan kleine code aanpassingen moeten doen. Dit wordt als volgt in het oefeningenboek weergegeven:

De code voor deze oefening kan je terugvinden onder de solution **Ucll.OOD.Hoofdstuk.Oefeningen**, en daarin project **Oefenproject**

De oplossingen zijn ook steeds uitgewerkt in code. De verwijzing naar het project met de oplossing in, vindt u terug onder de Oplossingen.

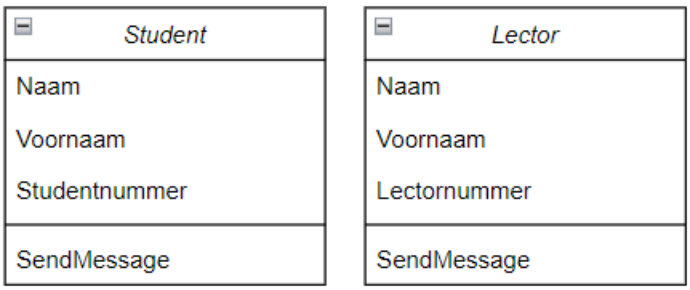
2 Object Oriented Design

2.1 Herhaling OO-programmeren

Stel, we willen een systeem maken om studenten en lectoren bij te houden. We willen zo veel mogelijk gebruik maken van classes of interfaces. We willen ook de volgende gegevens bijhouden:

- Student
  - Naam
  - Voornaam
  - Student nummer
  - Kan een boodschap sturen die “Hallo” zegt (lees: functie dat string returned)
- Lector
  - Naam
  - Voornaam
  - Lectornummer
  - Kan een boodschap sturen die “Welkom” zegt (lees: functie die string returned)

Hoe zouden we dit best modellen in verschillende classes? De meest standaard manier zou de volgende zijn.



Hoe kunnen we echter overervingen en/of interfacing introduceren in dit model?

Oplossing: Herhaling OO-programming

2.2 Single Responsibility Principle (SRP)

De code voor deze oefening kan je terugvinden onder de solution **Ucll.OOD.Basis.Oefeningen**, en daarin project **SRP**

Dit project zorgt ervoor dat je mensen kan toevoegen met een rijksregisternummer. Je kan enkel geldige rijksregisternummers registreren. We hebben dus een controle nodig op het rijksregisternummer, alvorens we dit kunnen toevoegen.
Tip: de controle zelf wordt momenteel gedaan in Program.cs

De bestaande code werkt, maar hanteert niet het SRP-principe. Vul de ValidateSSN methode aan van de Validators class, en implementeer deze in de bestaande code.

Oplossing: Single Responsibility Principle (SRP)

2.3 Open/Closed Principle

De code voor deze oefening kan je terugvinden onder de solution **Ucll.OOD.Basis.Oefeningen**, en daarin project **OCP**

Dit project neem verschillende soorten inkomsten aan, zoals:

- Netto-inkomen
- Avond toeslag
- …

En telt deze op en geeft het resultaat terug. We krijgen echter te horen dat de business verwacht dat er nog een 10-tal toeslagen gaan toegevoegd worden de komende 2 jaar.
Zorg ervoor dat de methode TotalIncomeCalculator.CalculateTotal hier vlot mee om kan, zonder dat we deze telkens moeten gaan aanpassen als er een toeslag bijkomt. Maak met andere waarde de CalculateTotal methode **open** voor uitbreidingen, maar **gesloten** voor aanpassingen.
Oplossing: Open/Closed Principle

2.4 Interface Segregation Principle (ISP)

De code voor deze oefening kan je terugvinden onder de solution **Ucll.OOD.Basis.Oefeningen**, en daarin project **ISP**

In dit project hebben een IGeneralStore interface, die geïmplementeerd wordt door twee grote internetwinkels die zowat alles verkopen. We gaan ons programma echter uitbreiden dat we ook lokale winkels gaan toevoegen. We gaan een winkel FashionShop toevoegen, die enkel make-up en kleren verkoopt, en een ComputerShop, die enkel hard- en software verkoopt.
Voeg twee classes toe:

- FashionShop
- ComputerShop

En refactor de bestaande code zodat we het Interface Segregation Principle niet verbreken. Zorg er wel voor dat alle classes onder het contract van een interface vallen.

Oplossing: Interface Segregation Principle (ISP)

2.5 Dependency Inversion Principle (DIP)

De code voor deze oefening kan je terugvinden onder de solution **Ucll.OOD.Basis.Oefeningen**, en daarin project **DIP**

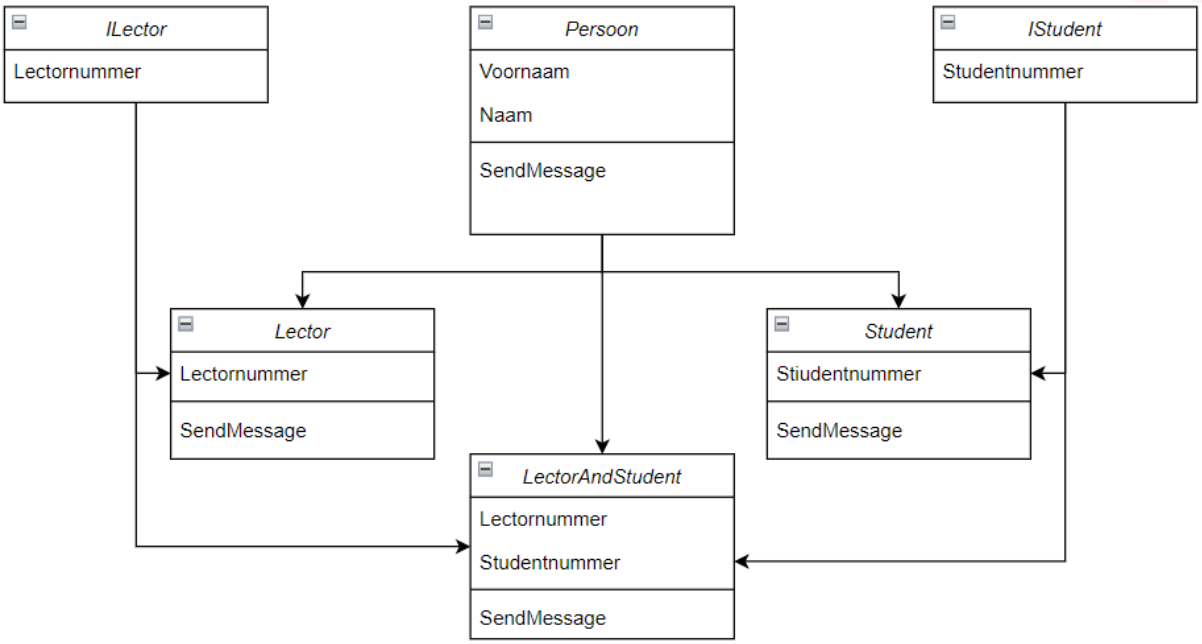
In dit project hebben een manager, die werknemers coacht. Deze werknemers zijn onderverdeeld in types en kunnen aan een manager toegewezen worden. De manager class is echter afhankelijk van deze types. Stel, we willen ook analisten toewijzen aan een manager, dan moeten we redelijk wat aanpassingen doen. Verwijder de afhankelijkheid op het specifiek type, en zorgt ervoor dat we via DIP makkelijk verschillende soorten werknemers kunnen toewijzen aan een manager.

3 Oplossingen

3.1 Herhaling OO-programming

Opdracht: Herhaling OO-programming

Oplossing:



We hebben een class Persoon, die onze gemeenschappelijke properties bevat. Van deze class kunnen dan twee childclasses laten **overerven**: Lector en Student. Deze twee classes **implementeren** ook twee interfaces, respectievelijk ILector en IStudent. Door onze opzet zo te maken, kunnen we ook een LectorAndStudent class creëren, waarbij iemand zowel les *geeft,* als een (andere) les *volgt*.

De code van dit voorbeeld kan je terugvinden onder de solution **Ucll.OOD.Basis.Oplossingen**, en daarin project **OO**

3.2 Single Responsibility Principle (SRP)

Opdracht: Single Responsibility Principle (SRP)

Oplossing:

We gaan de code vanuit Program.cs verplaatsen onder de static class `Validators`, onder methode `ValidateSSN`.

De twee keer dat we de logica gebruiken in Program.cs vervangen we dan door een call van deze methode. Zo bestaat onze logica op slechts één plek.

De code van de oplossing kan je terugvinden onder de solution **Ucll.OOD.Basis.Oplossingen**, en daarin project **SRP**

3.3 Open/Closed Principle

Opdracht: Open/Closed Principle

We kunnen dit vrij eenvoudig oplossen. Alle toeslagen worden opgeteld. Business heeft nood aan deze apart bij te houden voor business redenen, maar onze methode gaat eigenlijk een collectie van toeslagen moeten optellen. We kunnen hiervoor dus een lijst gebruiken.

De oplossing is dus CalculateTotal te *refactoren* zodat deze een List<int> aanneemt als parameter. In Program.cs kunnen we deze list dan opvullen.

De code van de oplossing kan je terugvinden onder de solution **Ucll.OOD.Basis.Oplossingen**, en daarin project **OCP**

3.4 Interface Segregation Principle (ISP)

We gaan onze mega interface IGeneralStore opsplitsen in verschillende interfaces:

- IComputerStore
- IFashionStore
- IToyStore

Onze kleinere winkels kunnen dan enkel de interface implementeren die ze nodig hebben. Onze grote winkels zullen dan meerdere interfaces implementeren.

De code van de oplossing kan je terugvinden onder de solution **Ucll.OOD.Basis.Oplossingen**, en daarin project **ISP**

3.5 Dependency Inversion Principle (DIP)

We kunnen een type abstract maken op twee manieren:

- Via interfacing
- Via inheritance

In dit geval hebben we gekozen voor inheritance, omdat de verschillende werkvormen allemaal Employees zijn. We maken dus een Employee class, en laten deze overerven naar Developper, Tester en Analyst. In onze Manager class gaan we dan enkel het type Employee kennen. We houden dus slechts één lijst bij.

De code van de oplossing kan je terugvinden onder de solution **Ucll.OOD.Basis.Oplossingen**, en daarin project **DIP**