

Les 4 - 3 lagen model

Oefeningen

Oefening 1 - Leeftijd

1.1 Functionaliteit

Het doel van de oefening is om de gebruiker een naam te laten ingeven en een leeftijd terug te krijgen.

1.2 Domain laag

Maak een Person klasse aan met twee properties: name & age.

Maak een interface aan die je IPersonRepository noemt, deze heeft een methode `GetPersonByName(string name)`.

Maak een DomainController aan die een IPersonRepository verwacht als parameter in zijn constructor. Geef de DomainController ook de methode `GetPersonAge(string input)` mee, deze geeft de leeftijd van een persoon terug door gebruik te maken van `GetPersonByName(string name)` uit de IPersonRepository.

1.3 Persistence laag

Zet een PersonMapper klasse op die in zijn constructor een paar Person objecten in een lijst steekt. Laat deze mapper de IPersonRepository interface implementeren. Voeg de methode uit de interface toe aan de klasse, de methode doorloopt de lijst tot er een persoon gevonden wordt die overeenkomt met de string parameter. Indien geen enkele persoon matched throw je een `ArgumentException` met een duidelijke foutboodschap.

1.4 Presentation laag

Maak in deze laag één klasse aan: `PersonApplication`, deze verwacht een referentie naar `DomainController` in zijn constructor. Zorg er hier ook voor dat de `ArgumentException` opgevangen wordt, de message uitgeprint wordt voor de gebruiker en daarna de gebruiker vraagt om opnieuw te proberen.

1.5 Startup

Maak hier een klasse `StartUp` aan, deze bevat de `Main` methode waar de applicatie mee zal opstarten. In de `main` methode maak je een `PersonMapper` object aan, die geef je mee aan een nieuw `DomainController` object wat je op zijn beurt mee geeft aan een nieuw `PersonApplication` object.

1.6 Verdere implementatie

Probeer nu zelf de individuele delen met elkaar te verbinden en de applicatie af te werken. Hou wel met het volgende rekening:

- In de presentatie laag (met uitzondering van de `StartUp` klasse) mag zo weinig mogelijk logica staan, hou deze zo 'dom' mogelijk. Steek alle complexiteit in je domainlaag.
- Ga nooit rechtstreeks van de presentatie laag naar de persistentie laag, dit mag enkel via de domein laag.
- Je vertrekt vanuit je applicatie klasse in de presentatie laag om zo te zien welke implementaties en verbindingen je nog mist

Oefening 2 - Dieren Quiz

2.1 Functionaliteit

Het doel van de oefening is om de gebruiker te ondervragen over zijn kennis van dieren. Als je gaat kijken in de CSV file zie je dat die een lijst met dieren bevat met elk een groep.

We tonen de gebruiker een dier en vragen tot welke groep dit dier behoort. We slaan elk antwoord op en houden bij of het antwoord juist was. Als tien dieren ondervraagd zijn geven we de gebruiker zijn score terug die we berekenen aan de hand van het aantal juist gesorteerde dieren.

2.2 Domain laag

Maak een enum aan genaamd `AnimalGroup` waar je de zes groepen in steekt (`Mammal`, `Invertebrate`, `Amphibian`, `Reptile`, `Fish`, `Bird`).

Maak dan een `Animal` klasse aan met twee properties: `name` & `group`.

Maak een interface aan die je `IAAnimalRepository` noemt, deze heeft twee methodes: `GetAllAnimals()` en `GetAnimalByName(string name)`.

Maak een `DomainController` aan die een `IAAnimalRepository` verwacht als parameter in zijn constructor.

Maak tenslotte een `AnimalQuiz` klasse aan, deze verwacht een lijst met `Animal` objecten in zijn constructor en heeft vier methoden:

- `StoreAnswer(Animal animal, bool correct)`: slaat een gegeven antwoord op in het `AnimalQuiz` object en of het juist of fout was.
- `GetNextAnimal()`: het `AnimalQuiz` object houdt bij welke dieren nog ondervraagd moeten worden en geeft één van deze terug. Zorg ervoor dat het dier dat je hier terug geeft uit de collectie met nog te ondervragen dieren verwijderd wordt.
- `HasMoreAnimals()`: geeft `true` of `false` terug afhankelijk of er nog dieren te ondervragen zijn.
- `GetScore()`: geeft terug hoeveel correcte antwoorden er gegeven zijn.

2.3 Persistence laag

Zet een `AnimalMapper` klasse op die uit de `animals.csv` alle dieren inleest en ze opslaat in een collection. Laat deze mapper de `IAAnimalRepository` interface implementeren. Voeg twee methodes toe aan deze klasse. De functionaliteit van de methodes kan je hopelijk afleiden uit de namen.

2.4 Presentation laag

Maak twee klassen aan: `AnimalQuizApplication`, deze verwacht een referentie naar `DomainController` in zijn constructor.

2.5 Startup

Maak hier een klasse `StartUp` aan, deze bevat de `Main` methode waar de applicatie mee zal opstarten. In de `main` methode maak je een `AnimalMapper` object aan, die geef je mee aan een nieuw `DomainController` object wat je op zijn beurt mee geeft aan een nieuw `AnimalQuizApplication` object.

2.6 Verdere implementatie

Probeer nu zelf de individuele delen met elkaar te verbinden en de applicatie af te werken. Hou wel met het volgende rekening:

- In de presentatie laag (met uitzondering van de StartUp klasse) mag je geen complexe types gebruiken, enkel strings, integers, booleans, etc.
- Ga nooit rechtstreeks van de presentatie laag naar de persistentie laag, dit mag enkel via de domein laag.
- Je vertrekt vanuit je AnimalQuizApplication klasse in de presentatie laag om zo te zien welke implementaties en verbindingen je nog mist
- Er zijn opzettelijk methodes weggelaten uit de opgave, dus waar nodig moet je aanvullen met je eigen implementaties. De opgave is slechts een startpunt.

Oefening 3 - Bankrekening

Herwerk de oefening bankrekening van les 2, gebruik hier eventueel het gegeven antwoord voor. Deze oefening heeft twee onderdelen:

1. Herschik de methodes, klassen en projecten volgens de principes van het drie lagen model.
2. Zorg er voor dat een gebruiker bij het opstarten van de applicatie een naam moet ingeven, bij het ingeven van deze naam worden alle IBeheerskosten objecten uitgeprint die dezelfde houder hebben. De rest van de methoden (storten, afhalen) gebruiken we niet in deze oefening.
3. Voorzie ook een foutboodschap indien er geen rekeningen gevonden werden voor de ingegeven naam.

Indien je wat extra testdata wil kan je deze code copy pasten in de juiste klasse:

```
new List<IBeheersKost>
{
    new SpaarRekening(873693157653,"Jean") { Saldo = 176000},
    new SpaarRekening(166471512255,"Luc") { Saldo = 11570},
    new SpaarRekening(399819657942,"Philippe") { Saldo = 2500},
    new SpaarRekening(646528875235,"Mohamed") { Saldo = 3300},
    new SpaarRekening(434902601370,"Thomas") { Saldo = 106000},
    new SpaarRekening(970062757885,"Martine") { Saldo = 250},
    new SpaarRekening(656724917471,"Rita") { Saldo = 99900},
}
```

```
new ZichtRekening(639540497707,"Luc",-2000) {Saldo = 7435},
new ZichtRekening(471849033525,"Jan",-2000) {Saldo = 1971},
new ZichtRekening(500222956997,"Thomas",-2000) {Saldo = 714},
new ZichtRekening(175190687257,"Patrick",-1000) {Saldo = 64},
new ZichtRekening(278580338046,"Patrick",-1000) {Saldo = 280},
new ZichtRekening(311115232429,"Anne",-500) {Saldo = 3000},
new ZichtRekening(192453282759,"Nicole",0) {Saldo = 1587},
new ZichtRekening(445892726430,"Christine",0) {Saldo = 70},

new Kluis("Michel",100),
new Kluis("Philippe",101),
new Kluis("David",102),
new Kluis("Rita",103),
new Kluis("Maria",104),
new Kluis("Isabelle",105),

};
```

Oefening 4 - Fitness drie lagen

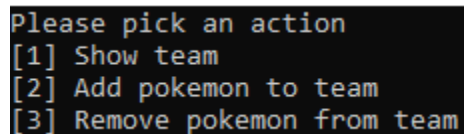
Pas nu de principes van het drie lagen model toe op de fitness oefening uit les 1. Je mag hier je eigen oplossing of de gegeven oplossing voor gebruiken.

Oefeningen 5 - Pokédex

Bij deze oefening maken we een applicatie waar je pokémon inleest uit een file en ze daarna kan toevoegen aan een team.

Presentation

Bij het starten van de applicatie krijgt de gebruiker een hoofdmenu te zien met drie mogelijkheden:



```
Please pick an action
[1] Show team
[2] Add pokemon to team
[3] Remove pokemon from team
```

1. "Show team" waar de gebruiker ofwel een lijst krijgt met alle pokémon in het team of een melding krijgt dat het team nog leeg is.

```

Please pick an action
[1] Show team
[2] Add pokemon to team
[3] Remove pokemon from team
1
#001 Bulbasaur (Grass/Poison)
#004 Charmander (Fire)

```

2. “Add to team” de applicatie print eerst een lijst af van alle types die een pokémon kan hebben en vraagt de gebruiker om een type te kiezen. Dan printen we alle pokémon af die dit type hebben. Daarna geeft de gebruiker hier aan de hand van het nummer van de pokémon mee welke moet toegevoegd worden aan het team.

Zorg er voor dat incorrecte input van de gebruiker een duidelijke foutboodschap toont en dat de applicatie terug keert naar het hoofdmenu.

```

Please pick a pokemon to add to your team
Please pick a type:
[0] Normal
[1] Fire
[2] Water
[3] Grass
[4] Electric
[5] Ice
[6] Fighting
[7] Poison
[8] Ground
[9] Flying
[10] Psychic

```

```

1
#004 Charmander (Fire)
#005 Charmeleon (Fire)
#006 Charizard (Fire/Flying)
#037 Vulpix (Fire)
#038 Ninetales (Fire)
#058 Growlithe (Fire)

```

3. “Remove from team” de applicatie print een lijst af met alle pokémon die op dit moment toegevoegd zijn aan het team. Daarna geeft de gebruiker het nummer in van de pokémon die verwijderd moet worden.

Opnieuw moet incorrecte input van de gebruiker een duidelijke foutboodschap tonen en dat de applicatie terug keren naar het hoofdmenu bij foutieve input.

```

Pick a pokemon to remove from the team
#001 Bulbasaur (Grass/Poison)
#004 Charmander (Fire)
#813 Scorbunny (Fire)

```

Domain

DomainController

De domaincontroller heeft een publieke constant integer **MaxTeamSize** die de waarde zes heeft. Het is de bedoeling dat we hier de grootte van ons team voor de volledige applicatie kunnen wijzigen. Zorg er dus voor dat je nergens anders het magic number 6 gebruik en dus de waarde van deze constant mee geeft aan alle klassen die deze waarde gebruiken.

Pokémon klasse

De pokémon klasse heeft drie properties: een id (moet altijd uit drie tekens bestaan), een naam en een lijst van types (waar altijd één of twee types in moeten zitten). Deze lijst met types kan je in een enum steken (interessant omdat deze lijst altijd hetzelfde blijft) maar je kan ook werken met een lijst van strings. Implementeer voor je eigen gemak ook de equals, gethashcode en toString methodes. De output van de toString methode moet er als volgt uit zien: **"#001 Bulbasaur (Grass/Poison)"**.

Zorg er voor dat het omzetten van een string lijn uit de CSV naar een pokémon object gebeurt in de constructor.

Repository interfaces

Voorzie hier twee repository interfaces voor twee mappers, zie het persistence deel hier onder voor verdere uitleg.

Persistence

PokémonMapper

Deze mapper leest bij het opstarten van de applicatie de CSV file in en houdt dan alle pokémon bij. Deze klasse bevat ook alle benodigde methodes om te filteren uit deze verzameling (ophalen aan de hand van id, type, ...)

TeamMapper

Deze mapper neemt het bijhouden van het team voor zijn rekening. Het toevoegen en verwijderen van pokémon alsook er voor zorgen dat het team nooit groter wordt dan

MaxTeamSize uit de domaincontroller. Indien het team te groot zou worden voorzie je een exception die in de presentation laag aan de gebruiker getoond wordt.