# Week 2-3: Het Klassediagram

De opgaven 1 t/m 5 zijn oefenopgaven. Met deze opgaven kun je oefenen. Je kunt de docent feedback vragen op je uitwerking van deze opgaven.

**Opgave 6 is een toetsopgave. Deze moet je inleveren.** De opgave wordt beoordeeld. Je moet de uitwerking mondeling verdedigen. Tijdens de verdediging krijg je feedback op de uitwerking. Er wordt verwacht dat je de feedback gebruikt voor het verbeteren van volgende opgaven.

## Opgave 1

In het college van week 2 wordt een ontwerp van een Employee klasse vertaald naar C# code.

Dit vertalen kan echter nog sneller door gebruik te maken van ‘automatic properties’ (zie HFC# H5 pag. 205.). Bij gebruik van automatic properties is het niet nodig datavelden in de code te declareren voor de opslag van de gegevens waartoe de property toegang biedt. Doordat het declareren van datavelden bij gebruik van automatic properties niet nodig is, is er minder kans op fouten in de code (hoe meer code, hoe groter de kans op fouten) en is je code overzichtelijker. Herschrijf onderstaande klasse mbv ‘automatic properties’. En vergelijk de lengte van de code.

class Employee

{

private string name;

private decimal salary;

public string Name

{

get { return name; }

set { name = value; }

}

public decimal Salary { get { return salary; } }

public Employee(string name, decimal salary)

{

this.name = name;

this.salary = salary;

}

public void RaiseSalary(int percentage)

{

salary = salary + (salary \* percentage / 100);

}

}

## Opgave 2

Vertaal de volgende UML klassen naar code voor C# klassen en test deze in een Forms applicatie. Het kan zijn dat klassen implementatiedetails hebben die niet voorkomen in het model van de klasse (encapsulatie!). Die moet je zelf dus aanvullen in je code.

Opmerking bij de ‘Dice’ (dobbelsteen) klasse: Deze klasse vertoont het gedrag van een dobbelsteen. Het aanroepen van de constructor zonder parameters creëert een (standaard) dobbelsteen met 6 zijden. Met de andere constructor kun je een dobbelsteen aanmaken met een ander aantal zijden. Er moeten altijd 3 of meer zijden zijn. De Throw() methode is een analogie voor het gooien van de dobbelsteen en levert het aantal ogen dat gegooid is (dit is een random waarde). De methode NrOfTimesThrown levert het aantal keer dat een zijde boven is komen te liggen. Bijvoorbeeld: NrOfTimesThrown(6) levert het aantal keer dat 6 is ‘gegooid’.

Na het aanroepen van ResetStatistics() wordt de historie, waarin het aantal keer dat een zijde is boven komen te liggen, gewist.

## Opgave 3

Vertaal het gegeven klassediagram naar code voor C# klassen. Maak daarna een C# Forms applicatie die de functionaliteit van de klassen in het klassediagram gebruikt en test.



## Opgave 4

Vertaal het gegeven klassediagram naar code voor C# klassen. Maak daarna een C# Forms applicatie die de functionaliteit van de klassen in het klassediagram gebruikt en test.



## Opgave 5

In deze opgave is de multipliciteit van de (navigatie) associatie aangepast.

Wat verandert er nu aan de benodigde operaties (constructoren / methoden)?

Bedenk zelf welke operaties je nodig hebt. Vul deze in, in het klassediagram. Vertaal vervolgens je klassediagram naar code voor C# klassen. Maak daarna een C# Forms applicatie die de functionaliteit van de klassen in het klassediagram gebruikt en test.



## Opgave 6. Deze opgave moet je inleveren en wordt beoordeeld.

Vertaal het gegeven klassediagram (zie volgende pagina) naar code voor C# klassen. Maak daarna een C# Forms applicatie waarmee een gebruiker de geboden functionaliteit van klassen kan gebruiken.

Met de User Interface van de forms applicatie moet de gebruiker

* artiesten, liedjes en playlists aan een MusicPlayer kunnen toevoegen.
* een bestaande playlist kunnen verwijderen.
* bestaande songs en playlists kunnen ‘afspelen’ (zie noot 1).
* het afspelen stoppen.

**Opmerking 1**: Het echt afspelen van liedjes (met geluid) is geen doel van deze oefening. De twee Play methoden hoeven dus geen code te bevatten die het geluid van het liedje hoorbaar maken. Maar als je dit werkend krijgt en de rest werkt ook goed, dan levert dit natuurlijk pluspunten op.

**Opmerking 2**: Het kan zijn dat er ‘details’ missen in het klassediagram, die je wel nodig hebt in de C# code van je klassen. Denk hierbij o.a. aan de *private* members (datavelden, properties en methoden). Indien nodig moet je deze toevoegen. Zorg verder voor correcte *encapsulatie* (zie H5 HFC#) van de attributen van de klassen.

**Opmerking 3**: Het implementeren van de ToString() methode kan voor een foutmelding zorgen. Zet ‘override’ in de declaratie van de methode om dit op te lossen. Zie het onderstaande voorbeeld. In de module van week 5-6, inheritance, leer je waarom dit nodig is.

public override string ToString()

{

return "Please make this method return a useful string!!";

}



De criteria voor de beoordeling zijn:

* De klassen, attributen, operaties en associaties zijn op correcte wijze vertaald naar C# code.
* De gevraagde functionele eisen zijn geïmplementeerd.
* Je kan het ‘hoe en waarom’ van de gemaakte code verdedigen.
* Het programma vertoont geen bugs (blijft hangen, crashed, …)
* De klassen zelf hebben geen bugs (blijft hangen, crashed, …) en zijn hiervoor niet afhankelijk van correct gebruik door andere klassen (bijv. de UI).
* De gemaakte code is netjes: goede encapsulatie (private/public), duidelijke variabele namen, netjes inspringen, geen dubbele code, zelf toegevoegde methoden hebben een duidelijk doel en een duidelijke naam.
* Juist gebruik van loops (for/while/foreach)
* Juist gebruik van keuze structuren (if/else/else if/switch).
* Bonus: De applicatie speelt ook werkelijk liedjes af die op de harde schijf staan.