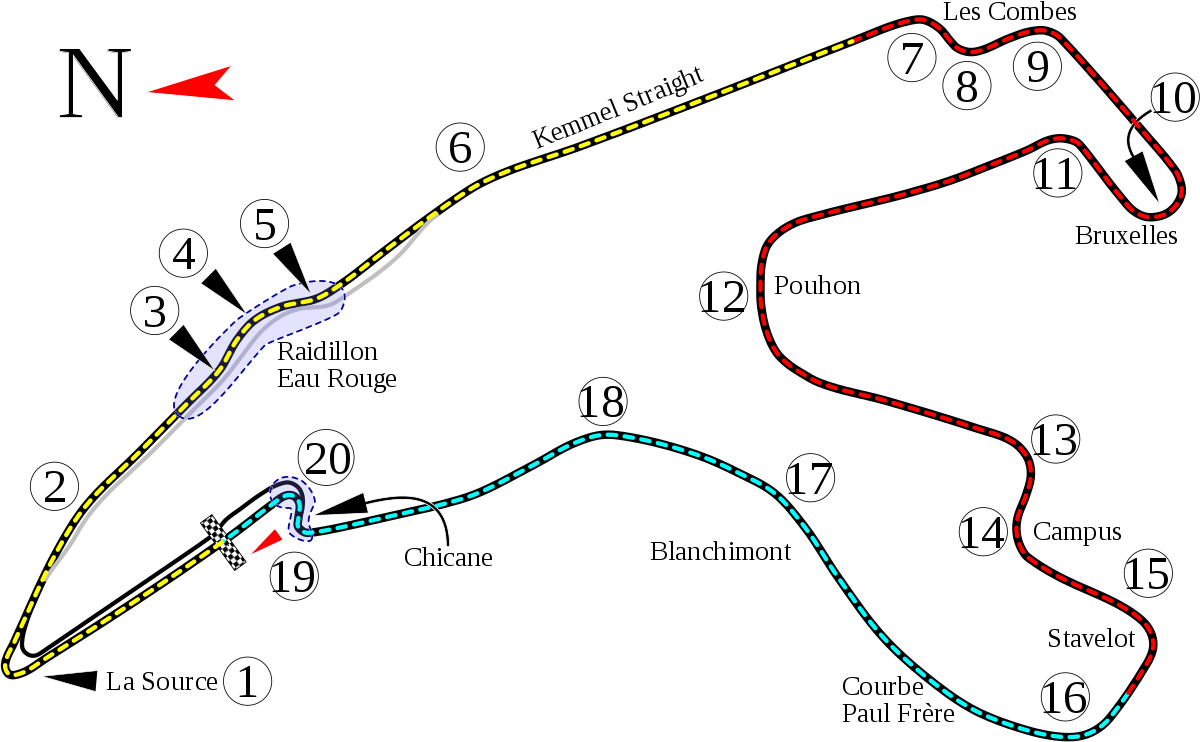
Alexander van Dam | | March 26, 2019

Race Statistics

Ontwerpdocument



Contents

[Architectuur 2](#_Toc4505742)

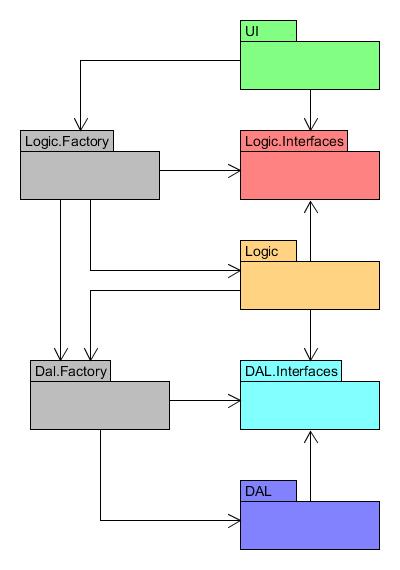
[Klassendiagram logica 3](#_Toc4505743)

[Klassendiagram Dal 4](#_Toc4505744)

[DBO 4](#_Toc4505745)

# Architectuur

De architectuur van de applicatie ziet er als volgt uit (merk op dat elke laag een eigen kleur heeft, in de verdere klassendiagrammen worden dezelfde kleuren gebruikt):

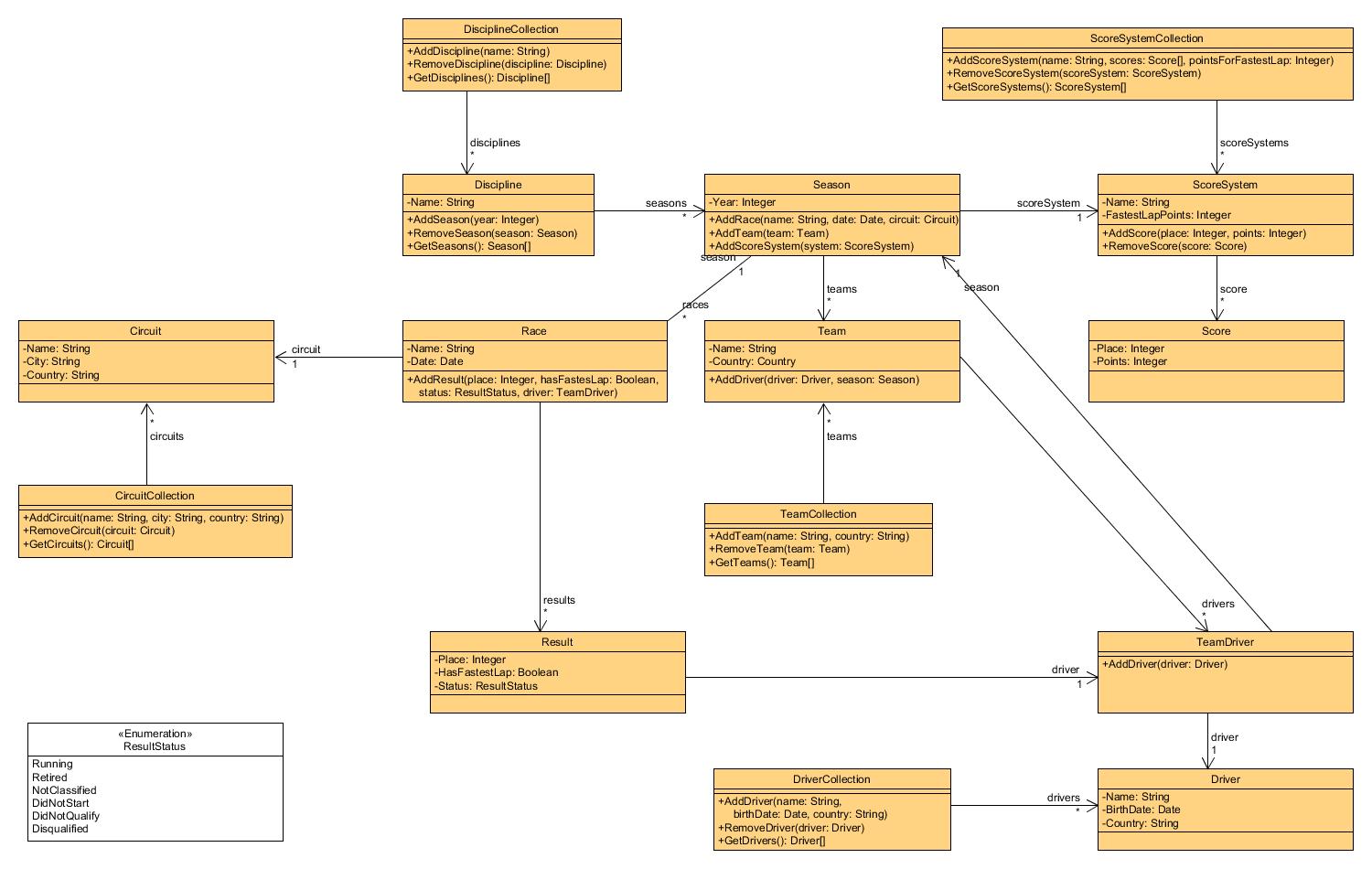


Er is gekozen voor een N-Tier architectuur (SRP, DIP), met expliciete interfaces tussen de lagen. De factories worden gebruikt om klassen uit de onderliggende lagen te instantiëren, zonder dat de laag erboven de implementatie kent (DIP). Toelichting per laag:

* UI: Presentatie laag. Bij het WPF project gaat het om de forms en de code behind achter de forms. Binnen het ASP project gaat het om de MVC implementatie van de website
* Logic.Interfaces: De interfaces van de logica modellen en excepties die de logica laag kan opwerpen.
* Logic.Factory: Factory patroon om objecten uit de logica te instantiëren op basis van de interfaces in Logic.Interfaces.
* Logic: Het gedrag van het systeem, implementeert de interfaces die in Logic.Interfaces aangeboden worden. Verder vertaalt de logica de data die uit de DAL komt naar objecten die in de UI en logica gebruikt kunnen worden. Zie voor meer informatie “Klassendiagram Logica”.
* Dal.Interfaces: In dit project zitten de interfaces van het repository pattern, samen met DTO’s (structs) waarin de data die van en naar de database gaan in bijgehouden worden. Verder zitten hier de excepties in die de DAL kan opwerpen.
* Dal: De code waarmee met de database gecommuniceerd wordt, opgezet via het repository patroon. Zie voor meer informatie “Klassendiagram Dal”
* Dal.Factory: Factory patroon om objecten uit de dal te instantiëren op basis van de interfaces in Dal.Interfaces.

# Klassendiagram logica

In onderstaand klassendiagram is te zien hoe de requirements en use cases in de logica terugkomen. Het gaat hierbij om de Must requirements. Elke klasse die vanuit de user interface aangeroepen kan worden heeft een interface waarin de functies en attributen die vanuit de UI aangeroepen kunnen worden zijn opgenomen.

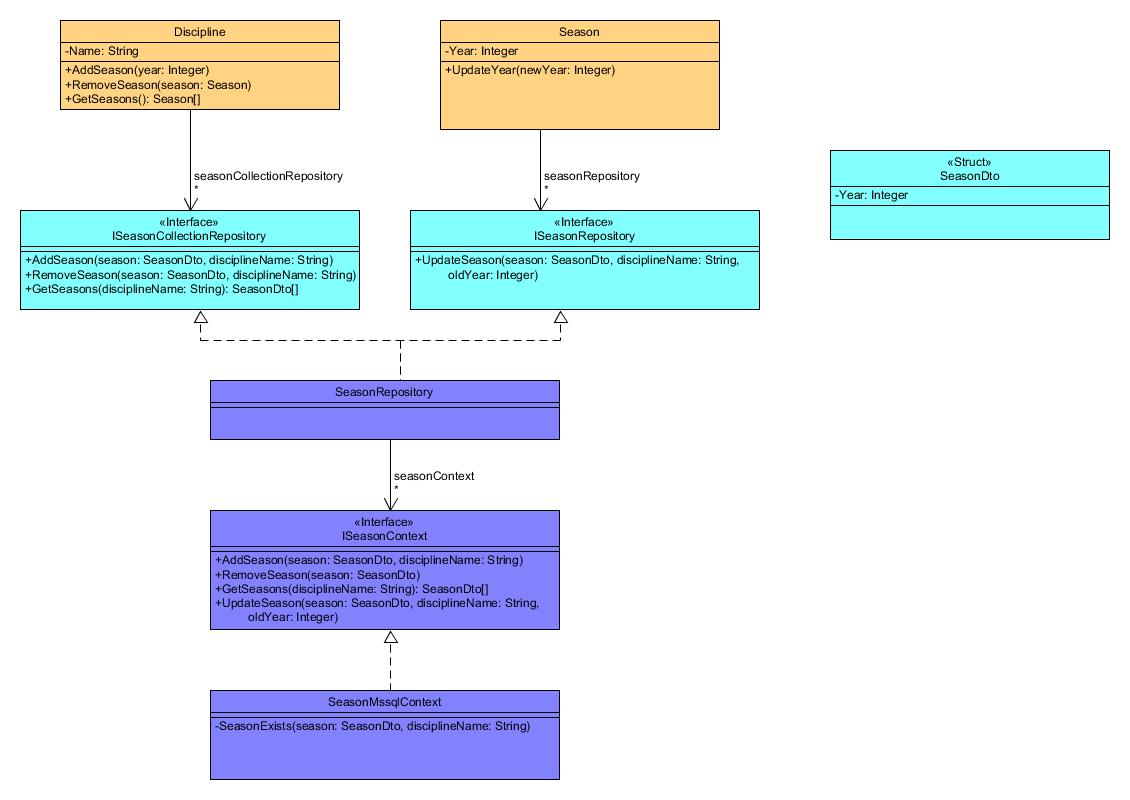


Aannames en gemaakte keuzes:

* De Teamdriver klasse is toegevoegd omdat een coureur in meerdere seizoenen bij meerdere teams kan zitten. Ook kan een coureur in één seizoen voor meerdere teams uitkomen. Om te voorkomen dat deze unieke rijder dan meerdere malen in het systeem voorkomt, wordt deze koppeling in TeamDriver bijgehouden.
* De attributen in het klassendiagram zijn aangemerkt als private. Dit betekent dat deze attributen wel gelezen, maar niet aangepast mogen worden door andere klassen.
* Alle relaties zijn private.
* Wanneer een lijst van objecten teruggegeven wordt uit een functie dan is dit een ‘unmodifyable collection’.

# Klassendiagram Dal

Voor de database wordt het repository pattern gebruikt. Hierdoor is het mogelijk eenvoudig andere databases te ondersteunen of tests te schrijven door andere contexten te implementeren. Het klassendiagram voor de entiteit Season staat hieronder uitgewerkt. Voor alle entiteiten werkt het hetzelfde.

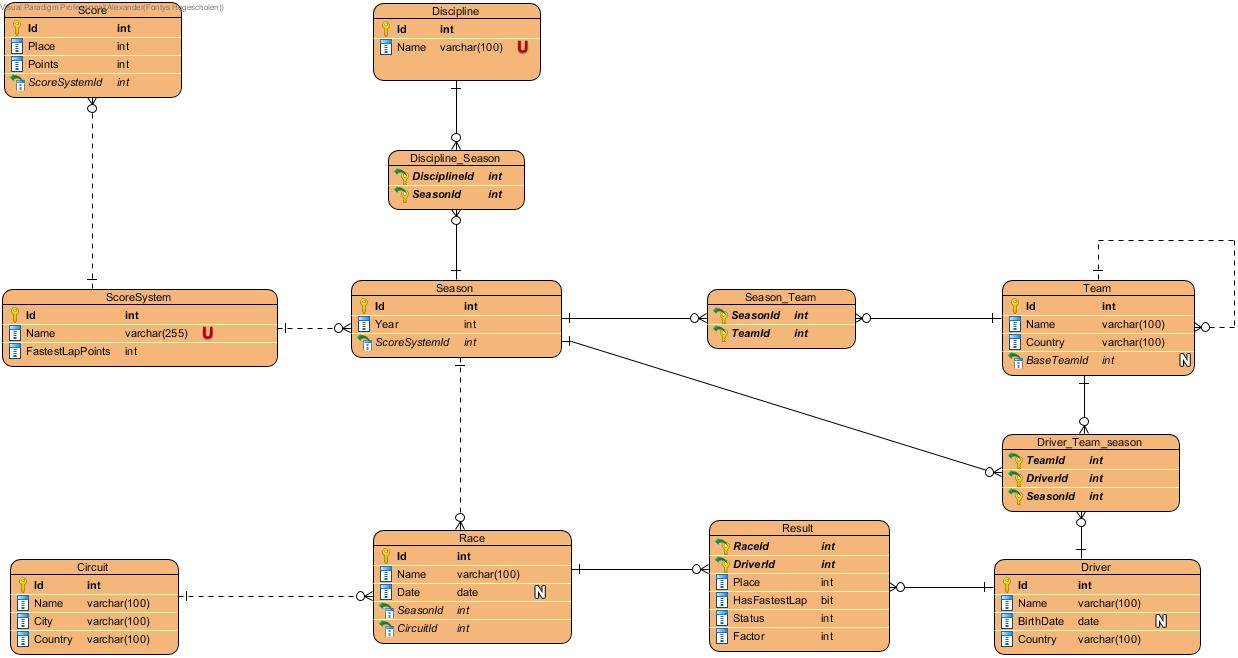


Keuzes:

* Er is gekozen voor een dubbele interface, om het ISP issue dat een individueel seizoen een ander seizoen zou kunnen toevoegen te voorkomen.
* De DTO’s zijn bedoeld als data-objecten om data van en naar de database te krijgen.

# DBO

Het database ontwerp zoals hieronder getoond volgt uit het conceptuele model.



Keuzes:

* De factor bij een result geldt voor een hele race. Om de relevante gegevens bij elkaar te houden is echter gekozen deze factor toch in result op te slaan. In dit geval weegt de duplicatie van data niet op tegen de overzichtelijkheid van het ontwerp.