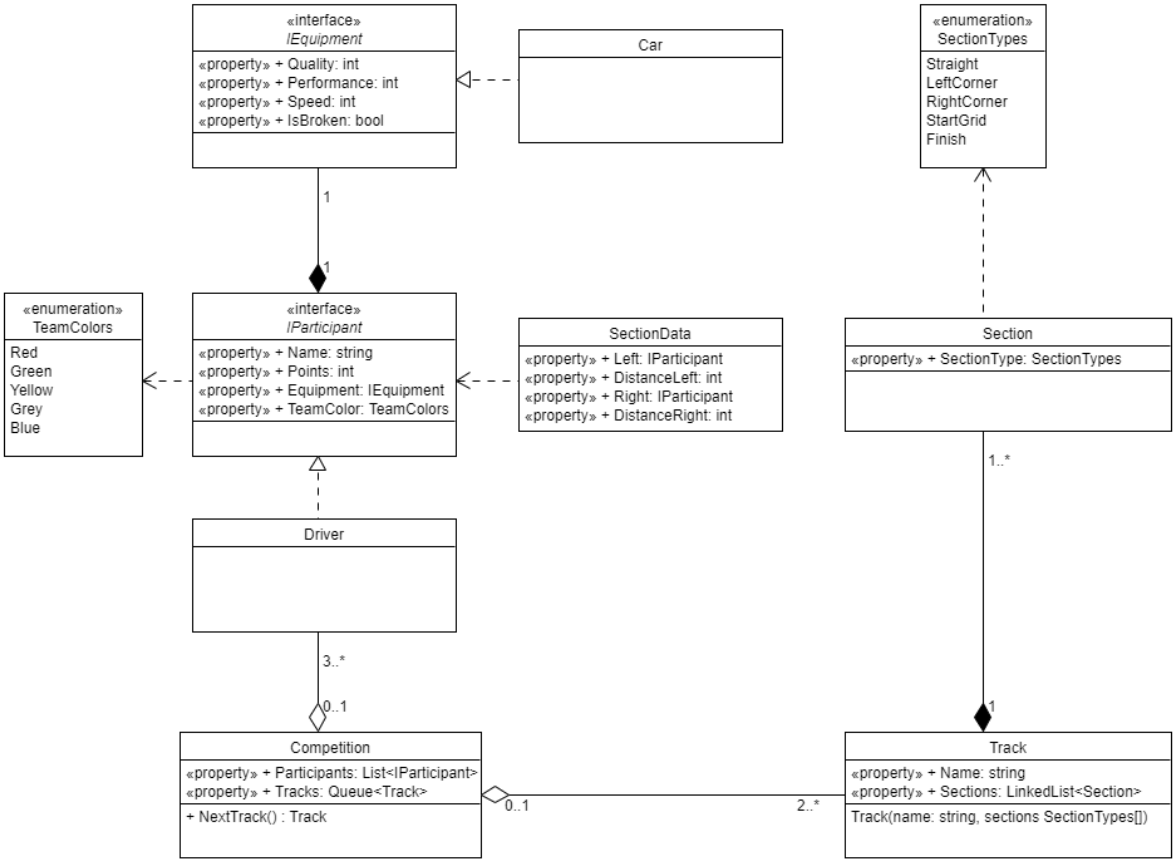
Taakgroep: Expanding projects

Context

In het vorige level heb je verschillende projecten opgezet. Elk project heeft een eigen 'verantwoordelijkheid' binnen de solution. In dit level gaan we de projecten inhoud geven. Het 'model' project krijgt het doel om de templates te bevatten om alle gegevens in op te slaan. Het 'controller' project krijgt het doel om alle logica te bevatten en bewerkingen op de gegevens te doen.

Het weergeven van het gegevens model in een applicatie wordt vaak gedaan met een klassendiagram. Het onderstaande klassendiagram ga je in stappen implementeren.

Je begint met het implementeren van de `interfaces` en `enums`. Wanneer dit gedaan is worden de klassen geïmplementeerd waarbij je gebruik gaat maken van de `interfaces` en `enums`.



Klassendiagram

Taak: Aanmaken interfaces

Met deze opdracht ga je eerst de interfaces aanmaken en implementeren in het 'model' project.
In een klassendiagram zijn de interfaces te herkennen aan de tekst `<<interface>>` en doordat de naam schuingedrukt is.

Aanpak

Voor elke interface in het klassendiagram doorloop je dezelfde stappen.

- Klik op de `model` project in de `Solution explorer`
- Ga via het contextmenu (klik met rechter muisknop) naar `Add > Class...`
- Selecteer `Interface` en geef bij `Name` de naam van de interface.
- Implementeer de properties zoals aangegeven in het klassendiagram
- Bij de property `TeamColor` bij de `interface IParticipant` moet een extra stap uitgevoerd worden. Voordat je deze property kan implementeren zal je eerst een `enum` moeten aanmaken met de naam `TeamColors`. Deze `enum` kan je bijvoegen in hetzelfde bestand als die van `IParticipant`.

Ondersteunende informatie

Microsoft heeft een handig overzicht met de richtlijnen voor het geven van namen aan identifiers.

Properties wijken af van globale variabelen doordat er gebruikt gemaakt wordt van getters en setters. Meer informatie hierover kan je hier lezen.

`enums` zijn een manier om waarden `strong typed` te maken. Voor uitleg hoe `enums` werken en kunt implementeren kan je hier meer informatie vinden.

Taak: implementeren model

In de vorige taak heb je alle interfaces aangemaakt in het 'model' project.
Tijdens deze taak ga alle klassen aanmaken en van de interfaces gebruik maken.

Aanpak

Voor elke klasse in het klassendiagram doorloop je dezelfde stappen. Let op: de klassen `Driver` en `Car` mag je zelf vervangen met andere namen passend bij jouw type race.

- Klik op de `model` project in de `Solution explorer`
- Ga via het contextmenu (klik met rechter muisknop) naar `Add > Class...`
- Selecteer `Class` en geef bij `Name` de naam van de klasse
- Implementeer de properties zoals aangegen in het klassendiagram. Kijk bij de ondersteunende informatie wanneer je bij een bepaalde type property niet uitkomt.
- Wanneer van toepassing, implementeer de aangegeven interface(s)
- Gebruik een `constructor` om de properties te initialiseren.

Ondersteunende informatie

Visual Studio biedt veel ondersteuning tijdens het programmeren.
Op deze pagina wordt uitgelegd hoe Visual Studio kan helpen bij het implementeren van interfaces.

Elke klasse heeft een `constructor` zelfs als je er zelf geen maakt. De `constructor` heeft als doel om de klasse een start te geven. Meer informatie over `constructors` kan je hier lezen.

De `List<Participants>` is een generic list. Meer informatie hierover kan je hier lezen

De `Queue<Track>` is een generic queue. Meer informatie hierover kan je hier lezen

De `LinkedList<Section>` Lijkt veel op een generic list alleen staat de volgorde van de items vast. Voor meer informatie kan je hier gaan.

De `[]` staat voor een `array` bij de constructor in klasse `Track`. Voor meer informatie over `array` is te vinden op deze pagina.

Taak: model gebruiken

Zoals aangegeven maak je gebruik van het MVC patroon. De model heb je in de vorige 2 taken geïmplementeerd.
Aangezien de verantwoordelijkheid van de model alleen is om gegevens te bevatten ga je nu verder met de controller.

De gegevens in de model gaan voornamelijk over een competitie met alle bijbehorende attributen.
In deze taak gaan je een klasse aanmaken welke een competitie declareert en initialiseert.

Doordat je gebruik maakt van onderdelen uit een ander project moet elke klasse een verwijzing krijgen naar het `model` project.

Punt van aandacht is dat de door jou gemaakte code testbaar moet zijn. Dit kan je doen door bij alle functies het `single responsibily` principe toe te passen.

Aanpak

- In het `controller` project maak je de `static` klasse `Data` aan
- Bovenaan de klasse `Data` voeg een je een extra `using` toe met de namespace `Model`
  Wanneer de naam `Model` rood onderstreept is dan kan het betekenen dat de referenties naar de andere projecten nog niet goed staan. Zie hiervoor level 1.
  Of jij hebt gekozen voor een andere project naam voor het `model` project. In dit geval moet je de naam bij de using de juiste naam geven die je zelf hebt gekozen.
- Geef de klasse `Data` een property van het type `Competition`. Deze property moet uiteraard ook `static` zijn.
- Doordat de klasse `Data static` is mist de mogelijkheid van een eigen constructor. Om de property te kunnen initialiseren maak je een methode `Initialize`. In deze methode initialiseer je de property `Competition`.

  Je hebt nu een static klasse waarin je alle gegevens van een competitie kan opslaan. Om gegevens toe te voegen ga je hiervoor methodes schrijven waarbij je rekening houdt met het `single responsibily` principe. Om te beginnen ga je methodes schrijven om banen en deelnemers toe te voegen aan de competitie.

- Maak een static methode, zonder parameters en returnvalue, in de klasse `data` aan die als doel heeft om `Participants` (deelnemers) toe te voegen aan de competitie.
- In de aangemaakte methode voeg je een aantal deelnemers toe aan de competitie.
- Roep de aangemaakte methode aan in de methode `Initialize`.
- Herhaal de bovenstaande 3 stappen waarbij je nu een aantal `Tracks` toevoegt aan de competitie.

Ondersteunende informatie

De `using` statement kan verschillend worden gebruikt. De meest voorkomende variant is boven elke klasse te vinden namelijk referenties naar andere klasses en namespaces. Meer informatie kan je hier lezen.

Verdere uitleg over `Namespaces` kan je hier lezen.

Het `single responsibility` principe is onderdeel van het SOLID principe.

Verdere uitleg over `static` kan je hier lezen.

Uitleg en voorbeeld over hoe items toegevoegd kunnen worden aan de `generic list` kan je hier vinden

Uitleg en voorbeeld over hoe items toegevoegd kunnen worden aan de `generic queue` kan je hier vinden

Taak: Creating a race

De deelnemers staan te popelen om te gaan racen. Omdat de `responsibility` van de property `Competition` in de static class `Data` is om alle gegevens van de competitie te bevatten moet er voor een race een nieuwe class aangemaakt worden. De `responsibility` van deze class is om een race te simuleren. Tijdens de simulatie moeten er een aantal gegevens opgeslagen worden. Denk aan de `Track` waar de race zich plaats vindt of aan de positie van de deelnemers op de `Track`.

Aanpak

- Maak in het controller project een klasse `Race` aan.
- Geef deze klasse de volgende properties: `Track` van het type `Track`, `Participants` van het type `List<IParticipant>` en `StartTime` van het type `DateTime`

  Om de race onvoorspelbaar te maken ga je gebruik maken van een `Randomizer`.

- Geef de klasse `Race` een private attribuut `_random` van het type `Random`.

  De laatste informatie die nodig is zijn de posities van de deelnemers op de track. Dit is een belangrijk onderdeel waarbij je rekening moet houden met een aantal regels. De belangrijkste regel is dat er maximaal twee deelnemers op 1 `Section` mogen zijn; links en rechts. Daarnaast moet bijgehouden wat de positie van de deelnemer op de `Section` is.

  We gaan hiervoor gebruik maken van een `Dictionary`. Zoals je kan lezen bij de ondersteunende informatie bevat een `Dictionary` een `key` en een `value`. De `key` wordt een `Section` waardoor elke `Section` maar 1x mag voorkomen in de lijst. De `value` moet verschillende gegevens bevatten dat aansluit bij de uitleg hierboven. Hiervoor gaan we de klasse `SectionData` gebruiken die je tijdens een vorige taak hebt gemaakt.

- Gebruik de onderstaande code om de klasse `Race` de private variable `_positions` te geven.

```
private Dictionary<Section, SectionData> _positions
```

  Doordat `_positions` private is heb je meer controle over de lijst. Je gaat nu een Getter maken voor `_positions`. Normale manier van `{get;set;}` gaat hier niet werken omdat je dan het `object _positions` beïnvloed en niet de inhoud van de lijst.

- Maak de methode `GetSectionData` waarbij een `Section` als parameter meegegeven wordt
- Probeer de `value` uit de lijst `_positions` op te halen gegeven de `Section` parameter.
- In het geval dat er geen `value` is gegeven de `Section` parameter voeg dan een nieuwe `SectionData` object toe aan de lijst met de `Section` parameter als `key`.
- Retourneer de gevonden of nieuwe de `SectionData`

  Een Setter is niet nodig doordat de Getter al een nieuwe `SectionData` aanmaakt wanneer nodig.

  Nu alle properties gedeclareerd zijn moeten ze nog geïnitialiseerd worden.

- Geef de klasse `Race` een constructor waarbij alle properties geïnitialiseerd worden. Deze constructor heeft als `parameters`: `Track` en `List<IParticipant>`. Gebruik de `parameters` om de waarden van de properties `Track` en `Participants` te zetten.

  De private attribuut `_random` kan je initialiseren door `_random = new Random()` alleen is die dan niet erg random. Dit kan je terug lezen bij de ondersteunende informatie.
  Dit kan ondervangen worden door de private attribuut `_random` te initialiseren door `_random = new Random(DateTime.Now.Millisecond)`.

  Om de race spannend te maken gaan je de waarden van de apparatuur bij de deelnemers wat aanpassen. Dit maakt elke race wat onvoorspelbaarder.

- Breid de klasse `Race` uit met een methode `RandomizeEquipment`. `Itereer` over alle deelnemers in de competitie. Geef de properties `Quality` en `Performance` van de property `Equipment` een willekeurige waarde.

Ondersteunende informatie

Voor meer informatie over de `Randomizer` kan je hier terecht.

Meer informatie over `Dictionaries` kan je hier vinden.

Uitleg over `parameters` is hier te lezen.

`Itereren` over lijsten kan op verschillende manieren. Lees dit voor meer informatie.

Taak: Starting a race

In de voorgaande taak heb je de klasse `Race` aangemaakt in het `controller` project. In deze taak ga je een instantie maken van de klasse `Race` waarbij je eerstvolgende `Track` uit de queue haalt en gebruikt. Om de race spannend te maken ga je vervolgens de gebruikte apparatuur van de deelnemers willekeurige waarden gegeven.

Aanpak

Om een race mogelijk te maken moet er een `Track` beschikbaar zijn in de `Competition`.

- In de klasse `Competition` heb je al een methode `NextTrack` aangemaakt. Implementeer deze methode waarbij de methode de eerst volgende `Track` van de queue haalt en deze retourneert. Wanneer de queue leeg is retourneer dan de waarde `null`.

  Nu je de mogelijkheid hebt om een `Track` op te halen kunnen we een instantie maken van de klasse `Race`. Voordat we dit doen heb je een plek nodig om de `Race` te declareren.

- Geef de klasse `Data` een extra property `CurrentRace` van het type `Race`.
- Breidt de klasse `Data` uit met een nieuwe methode `NextRace`. Deze methode gebruikt de methode `NextTrack` bij de property `Competition`. Wanneer de geretourneerde waarde niet `null` is initialiseer dan `CurrentRace` waarbij de `Track` meegegeven wordt als parameter aan de constructor.

Ondersteunende informatie

Meer weten over de `return` statement? Lees dan dit

Informatie over de `null` keyword kan je hier vinden

Taak: Visualisation

Wanneer je de voorgaande taken goed hebt uitgevoerd heb je de basis gelegd voor een race simulator. De competitie heeft een aantal deelnemers met voertuigen en een aantal racebanen om te racen. In deze taak ga je een begin maken om de eerste race te starten en dit te visualiseren.

De visualisatie ga je maken in het console project in je solution. Deze visualisatie zal dus plaatsvinden op de console met behulp van (gekleurde) tekst.

Aanpak

- Klik op het console project in de `Solution explorer`. Wanneer je tijdens het aanmaken van de `solution` gekozen hebt voor een `console application` is de naam van het project hetzelfde als die van de `solution`.
- Open vanuit de `Solution Explorer` het bestand `Program.cs`. In de code zie je de static methode `Main(string[] args)`. Net zoals bij JAVA is dit het startpunt van de applicatie binnen het project.

  Als je alle voorgaande taken goed hebt uitgevoerd staat er nu een `Race` klaar. Voor nu ga je alleen de naam van de `Track` tonen op de `Console`. Breid de static methode `Main(string[] args)` met de volgende stappen:

- Roep de `Initialize` methode aan van de static class `Data`. Let op dat de referenties tussen de projecten goed moet staan en een using toegevoegd moet worden aan `Program.cs`.
- Roep de methode `NextRace` aan van de static class `Data`.
- Print op de `console` de naam van de `Track` van de `CurrentRace`.

  Het is nu tijd om eindelijk wat op beeld te krijgen.

- Klik nu met je rechtermuisknop op de naam van het console project. Kies uit het context menu `Set as Startup Project`. Start het project nu met `F5` en kijk wat er gebeurt.

  Als je goed oplet zie je heel snel een console verschijnen en meteen weer afsluiten. Uiteraard is dit niet de bedoeling, zeker niet wanneer je een race wilt visualiseren.

- Breid de static methode `Main(string[] args)` verder uit met de onderstaande code:

```
for (; ; )
{
  Thread.Sleep(100);
}
```

Met de bovenstaande code wordt een `game loop` gemaakt. De console blijft nu net zolang staat totdat er op het kruisje wordt gedrukt.

- Druk nogmaals op `F5`. Als het goed is staat nu de naam van de eerste `Track` op de queue in de competitie.
- Wanneer dit gelukt is laat de uitwerking aan je docent zien om de opdracht af te laten tekenen.

Ondersteunende informatie

Meer informatie over de methode `main` kan je hier lezen.

Tips en uitleg over het gebruik van de `Console` kan je hier vinden.