Smart coding

Taakgroep: Generics

Context

Als je de aanleiding van het project en de domeinbeschrijving nogmaals leest zal je tot de conclusie komen dat je nog niet klaar bent met de simulator. Het is leuk dat de deelnemers mooi gevisualiseerd worden maar het ontbreekt aan bruikbare gegevens. Zonder gegevens hebben klanten niets aan de simulatie.

Ondertussen heb je wellicht wel door dat de SOLID principes belangrijk zijn om jouw werk begrijpelijk en makkelijk uitbreidbaar te houden. Misschien heb je wel methodes geschreven die meer dan 1 verantwoordelijkheid hebben, die worden met elke taak lastiger uit te breiden.

Belangrijk aspect van de SOLID principes is dat je ook geen klassen en methodes hebt die hetzelfde doen. Tijdens deze level ga je gegevens opslaan over de races en de competitie waarbij je rekening houdt met de SOLID principes.

Taak: Smart coding

De opslag van de gegevens is prima te doen. Je schrijft een aantal klassen in je model project om die vervolgens in je Data klasse te gebruiken. Maar naast de opslag van de gegevens wil je deze gegevens ook gebruiken. Bijvoorbeeld je krijgt de vraagt om de deelnemer van met de meeste punten in de competitie op te vragen. Je schrijft een methode die de deelnemer met de meeste punten retourneert. Vervolgens krijg je de vraag om de deelnemer met de snelste rondetijd te bepalen. Je gaat weer aan de slag en schrijft braaf weer een nieuwe methode. Met elke extra aanvraag groeit het aantal methodes en moet je steeds creatiever worden met de naamgeving van je methoden.

Wanneer je meer abstract kijkt naar de verzoeken lijken ze erg veel op elkaar. Stel je voor je hebt 2 deelnemers. Wanneer je moet vaststellen welke deelnemer de meeste punten heeft vergelijk je de puntenaantallen met elkaar. Bij het bepalen van de snelste deelnemer doe je ook een vergelijking maar dan met een tijd. Zelfs wanneer je de namen van de deelnemers in alfabetische volgorde wilt plaatsen doe je een vergelijking, namelijk tussen 2 strings. Voor situaties zoals hierboven beschreven bestaat het `generic` principe.

Aanpak

Tijdens de voorgaande taken heb je al, misschien onbewust, gebruik gemaakt van `generic` klassen. Denk bijvoorbeeld aan `List<T>`, `Queue<T>` en `LinkedList<T>`. Deze klassen bieden bepaalde functionaliteiten waarbij het niet uitmaakt welke objecttypes jij hebt gebruikt.

Voor de opslag van gegevens ga je ook gebruik maken van een `generic` klasse. Dit ga je doen in de volgende stappen.

- Maak in het `controller` project een normale klasse aan. Bedenk een naam de past bij een klasse die gegevens van de race opslaat en kan ophalen.

  Je herkent een `generic` methode of klasse vaak aan `T` of `<T>`. Dit mag ook een andere letter zijn maar vaak is het `T` (de T van Type). `T` is vanuit zichzelf niets, ook geen `object`. Het krijgt pas betekenis wanneer jij aangeeft wat `T` is.

  Het onderstaande voorbeeld is een `generic` klasse.

```
public class MyClass<T>
{


}
```

  Het initialiseren van een generic klasse gaat als volgt:

```
var c1 = new MyClass<int>();
var c2 = new MyClass<string>();
```

  Bij de instantie `c1` is `T` synoniem voor `int`. En bij de instantie `c2` staat `T` voor `string`. Je zult misschien denken, wat heb ik hieraan? Denk even terug aan de lijsten waarmee je al hebt gewerkt. Tijdens de initialisatie van `LinkedList<T>` gaf je een type mee. Bij voorbeeld `Section`, zoals in het onderstaande voorbeeld.

```
var lijst = new LinkedList<Section>();
```

  Hiermee weet de klasse `LinkedList` dat je gaat werken met `Section` en daarmee kan je ook alleen nog maar objecten toevoegen aan de lijst van het type `Section`. Bij de volgende taken krijg je een idee waarom dit zo krachtig is.

  Voor nu ga je de net aangemaakte klasse ook `generic` maken.

- Maak van de net aangemaakte klasse een `generic`.

  Omdat de klasse het doel krijgt om gegevens op te slaan moet je iets regelen voor de opslag. Hiervoor ga je `List<T>` gebruiken. Alleen wat moet je nu invullen voor `T`? Immers je weet nog niet welke type objecten in de lijst worden geplaatst. Eigenlijk is het eenvoudig, je kunt hier gewoon `T` laten staan. Zie ook het onderstaande voorbeeld.

```
public class MyClass<T>
{
    private List<T> _list = new List<T>();
}
```

  In het geval bij `c1` staat er eigenlijk `private List<int> _list = new List<int>();` en bij `c2` `private List<string> _list = new List<string>();`.

- Voeg aan de door jou gemaakte klasse ook een private variabele van het type `List<T>`

  De reden waarom de variabele private is omdat we bij de volgende taken zelf invloed willen hebben op de waarden die toegevoegd worden. Je moet alleen nu wel zelf een methode schrijven om waarden toe te kunnen voegen aan de lijst.

- Schrijf voor de klasse een methode waarmee je waarden op de lijst kan plaatsen. Deze methode heeft 1 parameter en retourneert geen waarde. Tip: voor de parameter, denk aan `T`.

  Je hebt nu een `generic` klasse geschreven waarmee je waarden op een lijst kan plaatsen.

Ondersteunende informatie

Voor informatie over `generics` kan je hier terecht.

Taak: Data templates

Voordat je gebruik kan maken van je generic klasse zal je iets moeten maken om je data in op te kunnen slaan. In deze taak ga je het model project verder uitbreiden met verschillende klassen die dienen als templates voor je data.

Aanpak

De eenvoudigste gegevens zijn de punten van deelnemers in de competitie. Ze bestaan maar uit 2 waarden, namelijk de naam van de deelnemer en de hoeveelheid behaalde punten.

- Breid je model project uit en voeg een klasse toe. Geef deze klasse een property voor de naam van een deelnemer en een property voor de behaalde punten.

  Een andere veelgebruikte gegeven is de tijd. Denk bijvoorbeeld aan rondetijden.

- Schrijf een klasse, ook met 2 properties, waarmee je de naam en een tijd kan opslaan. Voor de property voor de tijd gebruik het type `TimeSpan`.

Ondersteunende informatie

Voor meer informatie over `TimeSpan` lees dan deze pagina.

Taak: Recording information

Nu je de data templates gemaakt heb kan je aan de slag om gegevens bij te gaan houden.

Aanpak

Deze taak ga je in 2 stappen uitvoeren. De eerste stap is om een locatie voor je gegevens te kiezen en te implementeren. De andere stap is om daadwerkelijk de gegevens op te halen en op te slaan.

De eerste gegevens die je makkelijk kan vaststellen zijn de behaalde punten na een race. Je bent vrij om zelf te kiezen hoeveel punten een eerste, tweede, enz. plaats na een race oplevert.

Stap 1 is het kiezen van een geschikte locatie om de gegevens op te slaan. De klasse `Competition` is een goede kandidaat aangezien hier ook andere informatie wordt opgeslagen welke betrekking hebben op de competitie.

- Breid de klasse `Competition` uit met een extra property. Gebruik de aangemaakte generic klasse en een geschikte data template klasse als type voor de property.

  Nu je een locatie hebt voor de opslag van de gegevens is het vervolgens tijd voor stap 2: ophalen van de gegevens. Het bepalen van de behaalde punten doe je aan het einde van de race. Voordat je punten kan uitdelen moet je wel de eindstand vaststellen. Bij de volgende stap mag je zelf bedenken en bepalen hoe je het probleem wilt oplossen.

- Breid de klasse `Race` uit met een methode die de eindstand kan bepalen en retourneert.

  Wanneer je de eindstand weet kan je de behaalde punten ook vaststellen en opslaan in je aangemaakte generic klasse.

- Schrijf een methode in de klasse `Competition` die een eindstand als parameter accepteert. Deze methode gebruikt de eindstand om de deelnemers punten te geven. Gebruik hiervoor de data template die je hebt gebruikt voor property met de generic klasse. Gebruik de methode van de generic klasse om de data toe te voegen.
- Wanneer de race afgelopen is gebruik dan de return value van aangemaakte methode in de klasse `Race` om de nieuwe methode in de klasse `Competition` aan te roepen. Tip: Gebruik hiervoor het `RaceFinished` event.

  Wanneer je de simulator laat lopen kan je met de debugger controleren of de behaalde punten ook daadwerkelijk aan de lijst wordt toegevoegd.

Taak: More data

Bij de vorige taak heb je de simulator uitgebreid zodat je weet wie uiteindelijk de competitie wint. Maar er is natuurlijk veel meer data aanwezig in jouw race simulator. Je moet het alleen nog even ophalen en opslaan. Denk bijvoorbeeld aan rondtijden, afstanden tussen deelnemers, etc..

In deze taak ga je, net zoals bij de vorige taak, nog een aantal data bronnen implementeren met behulp van de data templates en de generic klasse.

Aanpak

Informatie waar je veel mee kan is de tijd hoelang de deelnemer doet over een sectie. Bijvoorbeeld de som hiervan is namelijk de rondetijd. Of je kan bepalen wat de afstand, in tijd, is tussen twee deelnemers. Bij een vorige taak heb je een data template gemaakt waarbij je de deelnemer en de tijd kan opslaan. Om de tijd per sectie bij te houden zal er 3e property aangemaakt moeten worden, namelijk de sectie. Hierbij heb je 2 keuzes: Of je past de data template aan met de extra property of je maakt een nieuwe data template waarbij je overerft van de originele data template. Aan jou de keuze hoe je dit wilt aanpakken bij de volgende stap.

- Breid de applicatie uit waarbij je de tijd, per sectie en deelnemer, opslaat. Tip: Gebruik de `ElapsedEventArgs` parameter in de klasse `Race` om de tijd te bepalen.

  Bedenk nu zelf tenminste 2 andere data bronnen om op te halen en op te slaan. Maak desnoods een extra data template aan zoals je dat bij het begin van de level ook hebt gedaan.

- Implementeer 2 andere data bronnen en sla deze gegevens op. Gebruik hiervoor de generic klasse en een bestaande of nieuwe data template.

Taak: Abstraction

Je hebt nu een aantal data templates gemaakt. Wanneer je inhoudelijk kijkt naar deze templates zijn het willekeurige klassen die weinig met elkaar te maken hebben. Als ontwikkelaar moet je altijd met een "helikopterview" kijken naar de wensen van de klant en het werk wat je maakt. Als je de vraag zou krijgen om alle data templates in 1 zin te beschrijven zou je dat kunnen doen met "de klasse slaat een resultaat op van een deelnemer". Met de term "resultaat" kan je nog alle kanten op maar de term "deelnemer" is heel specifiek.

Waar de data templates nog specifiek zijn is de generic klasse juist totaal niet specifiek. Hierdoor kan je moeilijk en effectief invulling geven aan de generic klasse.

In deze taak ga je deze `abstractie` omzetten in code door middel van een `interface`. Deze `interface` ga je vervolgens gebruiken om je generic klasse te `scopen`.

Aanpak

In level 2 heb je al een aantal `interfaces` aangemaakt en gebruikt bij een aantal klassen.

- Maak in jouw model project een extra `interface`. Geef de `interface` een property voor een deelnemer vergelijkbaar met die van de data templates.
- Implementeer de aangemaakte interface bij de data templates.

Wat je nu effectief gedaan hebt is de overeenkomt tussen alle data templates samenbrengen in een interface. Door de data templates weer gebruik te laten maken van de nieuwe interface heb je tussen de data templates een functionele koppeling gelegd.

De generic klasse is nu nog te algemeen. Toch heb je deze generic klasse aangemaakt voor een bepaalde functionaliteit, namelijk het opslaan en verwerken van resultaten. Gelukkig heb je nu, door middel van de interface, een functionele koppeling. Deze functionele koppeling kan je doortrekken naar de generic klasse. Dit doe je door gebruik te maken van de interface en de `generic type constraint`. Het engelse woord "constraint" verraad al wat je eigenlijk doet, een beperking opleggen. Waar leg je de beperking op? Op de "type" van de generic klasse.

Wanneer je het voorbeeld hebt gevolgd heb jij voor de "type" de letter "T" gekozen, vergelijkbaar met het onderstaande voorbeeld:

```
public class MyClass<T>
{
    private List<T> _list = new List<T>();
}
```

Een `generic type constraint` doe je met `where`. Zie ook het onderstaande voorbeeld.

```
public class MyClass<T> where T : IMyInterface
{
    private List<T> _list = new List<T>();
}
```

In het bovenstaande voorbeeld mag je generic klasse `MyClass<T>` gebruiken MITS de type minimaal gebruik maakt van de interface IMyInterface. Hiermee voorkom je dus dat de generic type voor alles gebruikt kan worden.

- Breid jouw generic klasse uit en geef deze een `generic type constraint` met de interface die je in deze taak hebt gemaakt.

Nu ligt er een functionele koppeling tussen de data templates en de generic klasse.

Ondersteunende informatie

Meer informatie over `generic type constraints` lees dan dit

Hoe je de `where` kan gebruiken kan je hier lezen

Definitie van `abstractie`.

Taak: Same data different result

Wanneer je voorgaande taken uitgevoerd hebt, heb je nu tenminste 4 databronnen. Je gebruikt de generic klasse, die je zelf hebt ontwikkeld, om deze data op te slaan. De generic klasse zelf doet nu nog weinig spannends en heeft die nog maar 1 methode.

Op dit moment worden de uitslagen na een race toegevoegd aan de lijst. Na een aantal races staat elke deelnemer meerdere malen in de lijst met de behaalde punten. Bij een competitie ben je eigenlijk alleen geïnteresseerd in het totaal per deelnemer. Dit zou je kunnen bereiken door de toevoegen methode, van de generic klasse, aan te passen waarbij je de punten optelt bij een al bestaande record.

Tijdens de race zelf worden de tijden per deelnemer en sectie bijgehouden. Wanneer de race meerdere rondjes duurt heb je een probleem. Ineens heb je 2 tijden voor dezelfde sectie in de lijst staan. Welke is nu de meest recente? Eigenlijk wil je dat, in de generic klasse, de oude tijd overschreven wordt met de nieuwe tijd.

Je hebt nu te maken met 2 verschillende wensen die samenkomen in de generic klasse. In deze taak ga je stap voor stap de generic klasse echt generic maken waarbij je de SOLID principes blijft handhaven.

Aanpak

In de vorige taak heb je de generic klasse al functioneel gekoppeld aan de data templates door de interface te gebruiken voor de generic type constraint. Nu je dit gedaan hebt kan je de generic klasse meer specifieke functionaliteiten geven die aansluiten bij de functionaliteit die horen bij resultaten van deelnemers. Je gaat het beschreven probleem, met het toevoegen van nieuwe items, aanpakken.

Je zou ervoor kunnen kiezen om de methode, in de generic klasse, uit te breiden waarbij je met een `if statement` controleert wat het `type` van `T` is en daar de code te implementeren. Met 2 data templates nog wel te overzien. Maar hoe groot wordt je methode en generic klasse wanneer het gaat om 10 data templates, of 100....? Daarnaast doemt zich een ander probleem op. Stel je voor dat je een data template wilt wijzigen? Dan moet je niet alleen de data template wijzigen maar ook de generic klasse. Wanneer een andere ontwikkelaar verder gaat met jouw werk is het dan ook nog duidelijk? En wat als de ontwikkelaar gebruik maakt van de interface maar vergeet de generic klasse ook uit te breiden?

Je zult wel begrijpen dat je functionaliteiten zoveel mogelijk bij elkaar moet houden. Dus in het geval wil je het toevoegen van een item op de lijst eigenlijk implementeren in de data template zelf. De interface die je hebt gemaakt kan dit mogelijke maken.

- Breid de interface uit met de methode `Add`. Deze methode heeft geen return value en 1 parameter; een `List<T>` waarbij `T` de interface zelf is.

  Nu je de interface hebt uitgebreid moet je deze methode aanmaken bij alle klassen die gebruik maken van de interface.

- Geef alle klassen, die gebruik maken van de interface, de benodigde methode.

Je kan nu, per data template, differentiëren hoe een item wordt toegevoegd aan de lijst.

- Implementeer voor de data template, waar je de behaalde punten mee opslaat, de aangemaakte `Add` methode. Gebruik de meegegeven `List` om de aantal behaalde punten te verhogen voor de deelnemer wanneer er al een item is voor de deelnemer. In alle andere gevallen voeg de item (zichzelf dus met `this`) aan de meegegeven `List`.
- Implementeer voor de andere data template, waar je de tijd per sectie en deelnemer mee opslaat, de aangemaakte `Add` methode. Gebruik de meegegeven `List` en overschrijf de tijd bij de item die dezelfde deelnemer en sectie heeft. In alle andere gevallen voeg de item (zichzelf dus met `this`) aan de meegegeven `List`.
- Implementeer voor alle andere data templates ook de `Add` methode.

  Nu je voor elke data template een eigen, unieke, `Add` methode hebt geschreven kan je methode in de generic klasse aanpassen. Dit is de werkelijke magie van interfaces en generics. Met de `generic type constraints` zorg je ervoor dat elke waarde voor `T` minimaal de interface heeft geïmplementeerd. De interface verplicht elke klasse, die hiervan gebruik maakt, dat er een `Add` methode aanwezig is met een `List` als parameter. Dus zonder dat je weet wat `T` is, je kan altijd de `Add` methode ervan aanroepen.

- In de generic klasse, vervang de implementatie van de methode. Roep de `Add` methode aan van de meegegeven parameter. Geef aan de `Add` methode de private `List`, van de generic klasse, mee.

  Wanneer je moeite hebt om te begrijpen wat er precies gebeurt. Gebruik de debugger om de code stap voor stap te doorlopen.

Taak: Best participant

Nu je de generic klasse en de data templates hebt geïmplementeerd kan je aan de slag om zinvolle informatie op te halen. In deze taak ga je de beste deelnemer ophalen uit de lijst.

Aanpak

Om de beste deelnemer op te halen per data template doorloop je dezelfde stappen als bij de vorige taak.

- Geef de interface, die je gebruikt voor de data templates, een extra methode. Het doel van deze methode is om de beste deelnemer op te halen uit de lijst. Deze methode heeft ook een `List` als parameter en een `string` als return value.
- Implementeer voor elke data template een eigen implementatie waarbij je vaststelt welke deelnemer het best/snelst/etc is. Retourneer de naam van de gevonden deelnemer.
- Breid de generic klasse uit met een methode zonder parameters en een string als returnvalue. Wanneer de private list leeg is retourneer dan een lege string. In alle andere gevallen retourneer de gevonden deelnemer. Tip: Gebruik een item van de list om de methode aan te kunnen roepen.

  Je zal wel merken hoe je met de generic klasse en de interface je oplossing makkelijk uitbreidbaar en aanpasbaar hebt gemaakt.

Taak: Visualisation

Voor deze laatste taak ga je tenminste voor 1 data bron de beste deelnemer tonen.

Aanpak

Kies zelf een data bron waarvoor jij de beste deelnemer, tijdens de race, wilt tonen op de console.

- Breid de console visualisatie uit waarbij je voor tenminste 1 databron de beste deelnemer toont.
- Voeg aan de test project de benodigde tests toe om je code coverage weer op niveau te krijgen.

Level 6 is alweer klaar. Je hebt enorm veel gedaan. Wees trots op jezelf! Vanaf nu ga je werken aan een nieuwe visualisatie. Mocht je nog ontevreden zijn op de visualisatie op de console kan je dit nu nog aanpassen.