COLLECTIONS

WAS SIND COLLECTIONS?

In den letzten Videos haben wir uns die Collections angeschaut und festgestellt, dass diese eine tolle Alternative zur einfachen Datenstruktur Array sind. In diesem Dokument fassen wir das Thema nochmal kurz zusammen.

Collections (zu Deutsch „Sammlungen“) sind wie der Name es schon vermuten lässt Datensammlungen (sogenannte Datenstrukturen), welche einem das Verwalten von mehreren zusammengehörenden Werten vereinfachen sollen. Es gibt verschiedene Arten von Collections, also verschiedene Klassen bzw. Typen die einem dabei zu Verfügung stehen:

- Lists
- Stacks
- Queues
- Dictionaries

Es gibt noch mehr Collection-Klassen, die oben genannten sind aber die wichtigsten.

COLLECTIONS VS. ARRAYS. WO LIEGT DER UNTERSCHIED?

Bevor wir uns die verschiedenen Arten von Collections anschauen, stellen wir uns erst einmal folgende Frage: „Wo liegt eigentlich der Unterschied zwischen Collections und einfachen Arrays?“

Collections besitzen im Gegensatz zu Arrays viele hilfreiche Methoden für das hinzufügen und entfernen von Werten innerhalb der Datenstruktur. Sie besitzen also zum einen keine festgelegte und konstante Größe, sondern können dynamisch verändert werden. Dazu kommt, dass Collections, je nach Art der Collection, unterschiedlich agieren was das hinzufügen, lesen und entfernen von Werten betrifft. Somit eignen sich für verschiedene Anwendungsgebiete unterschiedliche Collections. Möchte man zum Beispiel eine normale Liste von Personen anlegen, greift man zum Beispiel zur Klasse „List“. Wenn man hingegen ein Kartendeck in einem Videospiel verwalten, dann ist die „Stack“ (Stapel) Klasse die passende Variante, da diese sich wie ein Stapel verhält.

Du merkst also, dass Collections einfacher und mächtiger in der Nutzung sind als einfache Arrays.

LISTS

„Lists“ sind wie der Name es schon vermuten lässt ganz normale Listen von Werten. Wir können Werte zu dieser dynamisch hinzufügen und entfernen. Eine „List“ initialisiert man folgendermaßen:

```
static void Main(string[] args)
{
    List<string> personenListe = new List<string>();
}
```

Wie du siehst muss man zuerst den Typ „List“ angeben gefolgt von einem spitzen Klammerpaar in welchen sich der Typ befindet, den die Werte innerhalb dieser Liste haben sollen. Die spitzen Klammern deuten darauf hin, dass es sich bei der Klasse „List“ um eine generische Klasse handelt. Mehr dazu lernst du im nächsten Modul. Dann geben wir der Variable einen beliebigen Bezeichner, in unserem Fall „**personenListe**“ und dieses List-Objekt initialisieren wir dann mit **„= new List<string>();“.**

Soweit so gut, unser List-Objekt wurde erstellt. Diesem können wir nun auch ganz einfach Werte zuweisen.

Um einen Wert zu einer List hinzuzufügen, verwenden wir die „Add()“ Methode der Klasse „List“.

```
static void Main(string[] args)
{
    //Liste initialisieren
    List<string> personenListe = new List<string>();

    //Namen hinzufügen
    personenListe.Add("Peter");
    personenListe.Add("Günther");
}
```

Um diese Werte zu lesen, müssen wir wie bei Arrays auch den Index ansprechen.

```
//Werte lesen
Console.WriteLine(personenListe[0]);
Console.ReadKey();
```

Außerdem ist es möglich diese Datenstruktur mithilfe einer Foreach-Schleife zu durchlaufen.

```
//Werte lesen
foreach(string name in personenListe)
{
    Console.WriteLine(name);
}

Console.ReadKey();
```

Um einen Wert aus der Datenstruktur zu entfernen, können wir die Remove()-, RemoveAt()-, Clear()- und weitere Methoden verwenden. Bei der Remove()-Methode müssen wir als Parameter den exakten Wert mitgeben, den wir löschen wollen.

Bei der RemoveAt()-Methode müssen wir den zu entfernenden Index in Form eines Integer-Wertes mitgeben.

Die Clear()-Methode säubert die ganze Liste.

STACKS

Die „Stack“ Collection bietet uns die Möglichkeit einen „Datenstapel“ zu erstellen. Wenn wir einen Wert zu dem Stack hinzufügen, dann wird dieser ganz oben angefügt (wie wenn man eine Karte auf einen Stapel von Karten legt). Die Stack-Klasse bietet uns nun Methoden zum lesen der Werte an, welche immer nur den obersten Wert lesen, der sich auf dem Stack befindet (so als würde man die oberste Karte von einem Kartendeck aufdecken).

Ein Stack erstellt man wie eine List auch.

```
//Stack initialisieren
Stack<string> spielkarten = new Stack<string>();
```

Um nun einen neuen Wert auf den Stapel zu legen, benutzen wir die „Push()-Methode“ aus der Stack-Klasse.

```
//Werte zum Stack hinzufügen
spielkarten.Push("Herz 2");
spielkarten.Push("Pik 3");
spielkarten.Push("Karo 4");
```

In diesem Beispiel ist der oberste Wert auf dem Stapel nicht „Herz 2“ sondern „Karo 4“, da dieser zuletzt hinzugefügt wurde. Die Werte werden also mit „Push()“ immer ganz oben auf den Stapel gelegt.

Um einen Wert zu lesen, gibt es zwei verschiedene Möglichkeiten. Zum einen kann man den obersten Wert des Stapels lesen und gleichzeitig entfernen, zum anderen kann man den Wert einfach nur lesen und liegen lassen.

Um einen Wert zu lesen und gleichzeitig zu entfernen, verwenden wir die „Pop()-Methode“.

```
//Wert Lesen und entfernen
Console.WriteLine(spielkarten.Pop());
```

Um den Wert einfach nur zu lesen ohne ihn dabei zu entfernen, verwenden wir die „Peek()-Methode“.

```
//Wert nur lesen ohne ihn zu entfernen
Console.WriteLine(spielkarten.Peek());
```

Um ausnahmslos jeden Wert aus dem Stack zu löschen verwenden wir die „Clear()-Methode“.

QUEUES

Kommen wir nun zu den „Queues“ (zu Deutsch Warteschlangen). Diese stellen im Grunde genau das Gegenteil zu den gerade besprochenen Stacks dar. Eine Queue ist wie eine Warteschlange im Supermarkt zu verstehen. Der erste Mensch der sich in die Schlange einreiht, ist auch derjenige, der als erstes drankommt. Es wird beim lesen also nicht auf den letzten Wert in der Datenstruktur zugegriffen (wie bei den Stacks) sondern auf den letzten.

Eine „Queue“ erstellt man wie die vorherigen Collections auch.

```
//Queue Initialisieren
Queue<string> warteschlange = new Queue<string>();
```

Um neue Werte zur „Queue“ hinzuzufügen, also um neue Werte in die Warteschlange einzureihen, benutzen wir die „Enqueue()-Methode“.

```
//Werte zum Queue hinzufügen
warteschlange.Enqueue("Franz");
warteschlange.Enqueue("Josef");
warteschlange.Enqueue("Heinz");
```

Auch bei den Queues gibt es wie bei den Stacks zwei Varianten zum lesen der Werte. Einmal kann man den Wert lesen und gleichzeitig entfernen (mit der „Dequeue()-Methode“) und man kann die Werte lesen ohne sie zu entfernen (wieder mit der „Peek()-Methode).

Beim lesen eines Wertes aus der „Queue“ wird der aller erste Wert in der Datenstruktur geholt (also der erste Wert, der sich in die Warteschlange eingereiht hat). Um den ersten Wert zu lesen und gleichzeitig zu entfernen, verwenden wir die „Dequeue()-Methode“.

```
//Werte lesen und aus dem Queue entfernen
Console.WriteLine(warteschlange.Dequeue());
```

Um den Wert nur zu lesen ohne ihn zu entfernen, verwenden wir wie bei den Stacks auch schon die „Peek()-Methode“.

```
//Werte lesen ohne sie zu entfernen
Console.WriteLine(warteschlange.Peek());
```

Wollen wir den gesamten Inhalt der Queue löschen, verwenden wir wieder die „Clear()-Methode“.

DICTIONARIES

Dictionaries (zu Deutsch Wörterbücher) sind eine sehr spezielle Art von Datenstruktur. Sie bestehen aus Paaren von Keys (Schlüsseln) und Values (Werten). Um einen Wert lesen zu können, brauchen wir den dazugehörenden Schlüssel. Man hat also nicht einfach nur eine Liste von Werten in einem Dictionary, sondern Paare aus Schlüsseln und dazugehörenden Werten. Die Schlüssel und Werte können dabei von unterschiedlichen Datentypen sein.

Um ein Dictionary zu erstellen, müssen wir ein Objekt vom Typ Dictionary erstellen. Dieses mal müssen wir allerdings zwei Datentypen wählen. Einmal den Datentyp für die Keys und dann den Datentyp für die Values. Dies geschieht auch wieder zwischen den spitzen Klammern.

```
//Dictionary erstellen
Dictionary<string, int> wochentage = new Dictionary<string, int>();
```

Zu diesem Dictionary können wir nun Einträge hinzufügen, indem wir die „Add()-Methode“ verwenden. Dieser müssen wir als Parameter einen Key mitgeben und einen dazugehörenden Wert, für den der Key steht.

```
//Füge Werte hinzu
wochentage.Add("Montag", 1);
wochentage.Add("Dienstag", 2);
wochentage.Add("Mittwoch", 3);
wochentage.Add("Donnerstag", 4);
wochentage.Add("Freitag", 5);
```

Um einen Wert von einem Key zu lesen, müssen wir folgendes schreiben

```
//Lese Wert von einem Key
Console.WriteLine(wochentage["Donnerstag"]);
```

Das Ganze sieht fast so aus, als würden wir einen Index ansprechen wollen. Statt einen gewöhnlichen Index zwischen die eckigen Klammern zu schreiben, müssen wir den Key hineinschreiben.