C# - List & Dictionary

Fler samlingar

Tidigare har vi gått igenom arrayer/vektorer och nu introducerar vi objekten List och Dictionary. Dessa, likt arrayer/vektorer, används för att lagra variabler och objekt men det som gör List och Dictionary speciella är att de är mer flexibla än en array. Med att de är mer flexibla menar jag då att vi exempelvis inte behöver definiera hur många positioner de ska ha utan att det sker dynamiskt. Då dessa samlingar är objekt har de också inbyggda metoder som adderar till flexibiliteten och vi ska kolla på några utav dessa metoder.

För att besvara frågan ”Varför använda en array när en List<T> är mer flexibel?” så tänker jag bara nämna att en array är generellt snabbare och drar mindre processorkraft till skillnad från List<T>. Så för att summera: Arrayer/vektorer är snabbare, List<T> och Dictionary erbjuder mycket större flexibilitet.

List<T>

Ett List<T> objekt skapar man precis likadant som med andra objekt. Exempel:
List<T> myList = new List<T>();
T:et står för Type, dvs vilken typ vi vill lagra(allt från primitiva typer som int, float etc till objekt och så vidare). Nedanför är visar jag exempel på hur en lista skapas och hur vissa metoder används.

```
// Skapa en lista som ska lagra datatypen integer
List<int> lista = new List<int>();

// Använd .Add-metoden och lägg till två int
lista.Add(10);
lista.Add(5);

// Array med 3 värden
int[] array = new int[] { 1, 2, 3 };

// Lägg till flera värden samtidigt exempelvis
// från en array med metoden .AddRange
lista.AddRange(array);

// Ta bort den första instansen av "2" ur listan
lista.Remove(2);

// Skriv ut antal element i listan med hjälp av
// .Count egenskapen
Console.WriteLine(lista.Count);

// Rensar hela listan
lista.Clear();
```

Dessa är bara några få metoder som finns tillgängliga för oss att använda. De andra som finns kan ni läsa om i docs eller från listan som kommer upp efter ”.” noten.

Precis som med en vanlig array kan vi nå specifika element genom att skriva lista[n].

```
// Precis som med en array kan vi använda
// oss av en indexer för att nå element
for (int i = 0; i < lista.Count; i++)
{
    Console.WriteLine(lista[i]);
}
```

Här används en for-loop som exempel. När vi vill kolla storleken på en array skriver vi ”.Length” men med en lista måste vi använda oss av egenskapen ”Count”.

En List är väldigt nyttig att använda när vi känner att vi behöver flexibilitet och inte vet exakt hur många positioner vi behöver fylla. Ett exempel på detta kan vara att vi skapar ett program som behöver skapa nya objekt beroende på användarens önskemål(exempelvis ett register av någon typ).

Dictionary<Key, Value>

Dictionary används även för lagring men till skillnad från övriga samlingar använder sig en Dictionary av Key/Value par. Detta betyder att för varje värde vi lägger till behöver vi definiera en nyckel.

```
// Skapa ett Dictionary-objekt med int för Key och string för värde
Dictionary<int, string> dict = new Dictionary<int, string>();

// Lägg till element genom att först definiera
// nyckeln och sedan värdet
dict.Add(123, "Hej");
dict.Add(456, "Då");

// Om "dict" innehåller nyckeln "123"
// skriv ut värdet till konsolen
if (dict.ContainsKey(123))
    Console.WriteLine(dict[123]);

// Tar bort värdet "Då"
dict.Remove(456);
```

Här visar jag några funktioner som en dictionary har. Som ni ser när jag skapar en mitt Dictionary objekt så tar den in två argument inom vinkelparanteserna. Dessa är för Key(nyckel) och Value(värde).

Som ni ser i if-satsen använder jag ContainsKey() metoden och letar efter nyckeln ”123”. Om den existerar skriver vi ut värdet genom att anrop dict med nyckeln som argument. Här finns det en innbyggd ”try” metod för att hämta ett värde, exempel:

string value;
bool getValue = dict.TryGetValue(123, out value);
Som ni ser fungerar den precis likadant som en TryParse förutom att den söker med nyckeln och om den hittas returnerar den värdet.

Här kommer ett verkligt exempel på hur man skulle kunna använda en Dictionary:

```
// Skapa en dictionary med sträng som Key och List<string> som Value
Dictionary<string, List<string>> animalDict = new Dictionary<string, List<string>>();

// Lista för hundar
List<string> dogList = new List<string>();
dogList.Add("Jack Russel");
dogList.Add("That dog from Lassie");
// Lista för katter
List<string> catList = new List<string>();
catList.Add("Garfield");

// Lägg till namnen som nycklar och listorna som värden
animalDict.Add("Dogs", dogList);
animalDict.Add("Cats", catList);
```

Här har vi skapat en Dictionary som ska hålla data om djur och dess raser. Här blir nyckeln djurnamnet och rasen hamnar i ett List-objekt som blir värdet. Det vill säga när vi hämtar ett element med exempelvis nyckeln ”Dogs” så returneras värdet i form av en lista med olika hundraser.