Collections i C#

Tidigare i kursen har collections i form av listor, stackar, köer och arrayer undersökts hur de fungerar och hur de kan användas på ett effektivt vis. Nu när lite mer kunskap om collections inskaffats skall vi skapa vår egen collection. En Kundvagn.

1. Typ specifik kundvagn

Likt de collections som nämnts skall vår collection vara typspecifik. Med detta menas att den enbart tar in en typ av element. Exempelvis kan vi ha en List<string> (Utläses: List of strings) eller en List<itn> (list o fintegers), på samma vis skall kundvagnen fungera. Dock behöver kundvagnen enbart behandla string objekt tills vidare.

För att skapa detta läggs <datatyp> till i klassdeklarationen. Skapa klassen ShoppingCart<string>.

2. Datalagring

För att kunna fortsätta behöver vår kundvagn en intern datalagring. Precis som de tidigare nämnda collections skall detta vara en array. Som bekant från FuB finns dock ingen bra funktionalitet färdigbyggd för arrayer, det kommer behöva skrivas.

Skapa en string array i ShoppingCart klassen som ni döper till InternalStorage

Skapa variabeln ”int FirstOpenSlot” i ShoppingCart klassen och sätt den till noll

Skapa Add(string ItemToAdd) {…} metoden. I denna skapar ni funktionalitet för att:

1. Se första lediga platsen
2. Om den är tom lägga ItemToAdd på platsen

Om så:

3. Kolla om det finns en ny ledig plats
4. Om så: Sätt FirstOpenSlot till den nya lediga platsen.
5. Om inte: Sätt FirstOpenSlot till den sista platsen i arrayen.

Om inte:

3. Skapa en ny array som är 1 element större än föregående
4. Kopiera samtliga värden till den nya arrayen
5. Lägg till det nya värdet sist i arrayen
6. Sätt InternalStorage till den nya arrayen

Skapa Remove(string ItemToREmove) och RemoveElementAt(int index) metoderna. Där den första letar igenom arrayen efter en matchande string och tar bort den. RemoveElementAt tar bort elementet på den platsen som angivits. De båda metoderna sätter ”FirstOpenSlot” till den platsen där de tog bort ett element OM det inte fanns en lucka tidigare i arrayen.

Skapa void SetOpenSlot() metoden. Denna skall från index 0 till arrayens längd leta efter den första lediga platsen och sätta FirstOpenSlot till det värdet. Använd nu denna metod på alla ställen där ni manuellt letar efter det första öppna värdet.

3. Enumerering och Foreachloopar

För att en collection skall kunna foreachloopas måste den använda sig av interfacet IEnumerable. Ett interface används på samma vis som arv, alltså genom att skriva public klass : interface, i detta fall ”public class ShoppingCart<string> : IEnumerable<string>”. Därefter finns det funktionalitet i VisualStudio som låter er få de funktionsbenämningar som behövs genom att högerklicka på interface namnet och clicka Implement Interface.

När ni gör detta får ni två metoder. En public IEnumerator<string> GetEnumerator och en System.Colelctions.Enumerator System.Collection.Enumerable.GetEnumerator(). Dessa två skall vi nu ge funktionalitet.

För att göra detta på enklast vis skall vi använda oss av *yield* vilket låter oss returna fler värden, som en samling, från en loop.

I IEnumerator<string> GetEnumerator() skall ni loopa från index 0 till InternalStorage’s max index. I for-loopen skall ni enbart göra följande: ”yield return InternalStorage[index]” där index är det nuvarande loop-talet. Detta för att från början till slut skicka tillbaka värden i rätt ordning.

Därefter skall ni i den andra metoden skriva: ”return this.GetEnumerator()”. Vad gör den här metoden? Fundera och diskutera!

När de båda metoderna finns kan ShoppingCart enkelt foreach-loopas.

4. Testa

Det är nu dags att testa ShoppingCart. Skriv enkel kod i Program.cs för att testa om du kan lägga till, ta bort och foreach loopa på ShoppingCart. Notera att det är på kundvagnen du skall försöka **inte** på ShoppingCart.InternalStorage.

5. Item klassen

För att göra kundvagnen mer verklighetstrogen skall Item klassen skapas. En definition av Item som innehåller variabler och properties för följande:
Varans namn
Varans pris
Varans beskrivning

6. ShoppingCart of Item

Ändra nu ShoppingCart klassen från att behandla typen string (ShoppingCart<string>) till att behandla typen Item. Genom att byta ut alla förekomster av string till item.

7. Komplex funktionalitet

Nu när kundvagnen inte längre enbart hanterar strängar är det hög tid för den att få lite funktionalitet som särskiljer den från de collections som redan finns. Implementera följande metoder för kundvagnen:

CalculatePrice() //Går igenom InternalStorage och beräknar det totala priset
GetMostExpensiveItem //Letar efter det dyraste i kundvagnen
Clear() // Tömmer kundvagnen
Samt två egna metoder som kan komma till nytta för en kundvagn.

8. Framtidsförberedelser

För att förbereda för framtiden och vidareutveckling, skall kundvagnen fungera även om nästkommande utvecklare bestämmer sig för att kategorisera Item-klassen mer noggrant med subklasser. Detta kan enkelt åstadkommas genom Generics och Restrictions. Vi skall alltså än en gång omvandla kundvagnens typ.

För att göra detta skall kundvagnen görst bli generisk. Vilket görs genom att byta ut alla nuvarande förekomster av ”Item” mot T. Detta medför dock att samtlig funktionalitet som kräver item typen går förlorad. Men det kommer lösas väldigt snart.

Lägg till ”where T : Item” under klass deklarationen så det blir något likt detta:

```
public class ShoppingCart<T> : IEnumerable<T>
                    where T : Item
```

Detta kommer göra att T enbart kan vara Item eller subklasser till Item.

Varför är detta mer föredraget än att skapa ShoppingCart<Item>? Sök information, fundera och diskutera!