|  |
| --- |
| Övningsuppgift |

Anpassad samling med bilar

Steg 3

|  |
| --- |
| *Författare:* Mats Loock  *Kurs:* Inledande programmering med C#  *Kurskod:*1DV402 |

**Upphovsrätt för detta verk**

Detta verk är framtaget i anslutning till kursen Inledande programmering med C# vid Linnéuniversitetet.

**Du får använda detta verk så här:**

Allt innehåll i verket Anpassad samling med bilar av Mats Loock, förutom Linnéuniversitetets logotyp, symbol och kopparstick, är licensierad under:

|  |  |
| --- | --- |
|  | Creative Commons Erkännande-IckeKommersiell-DelaLika 2.5 Sverige licens. <http://creativecommons.org/licenses/by-nc-sa/2.5/se/> |

**Det betyder att du i icke-kommersiella syften får:**

* kopiera hela eller delar av innehållet
* sprida hela eller delar av innehållet
* visa hela eller delar av innehållet offentligt och digitalt
* konvertera innehållet till annat format
* du får även göra om innehållet

Om du förändrar innehållet så ta inte med Linnéuniversitetets logotyp, symbol och/eller kopparstick i din nya version!

Vid all användning måste du ange källan: ”Linnéuniversitetet – Inledande programmering med C#” och en länk till <https://coursepress.lnu.se/kurs/inledande-programmering-med-csharp> och till Creative Common-licensen här ovan.

**Innehåll**

[Uppgift 3](#_Toc332880267)

[Problem 3](#_Toc332880268)

[Klassen Car 3](#_Toc332880269)

[Klassen Program 3](#_Toc332880270)

[Testa programmet 4](#_Toc332880271)

[Mål 4](#_Toc332880272)

[Tips 4](#_Toc332880273)

[Lösning 5](#_Toc332880274)

[Car 5](#_Toc332880275)

[Program 6](#_Toc332880276)

[Kommentar 6](#_Toc332880277)

# Uppgift

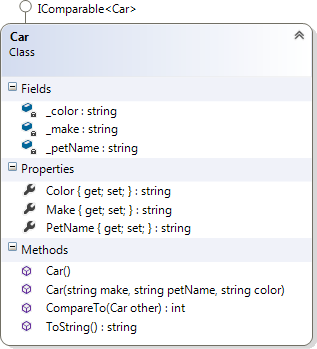
## Problem

Ett program ska hantera information om bilar. Den information som är intressant är bilens märke, smeknamn och färg. För att kunna hantera varje enskild bils information på ett rationellt och enkelt sätt skapar och använder du klassen Car. Då flera bilar ska kunna hanteras lagrar du referenser till Car-objekten i en generisk samling av typen List<T>.

Din uppgift är att implementera klassen Car enligt klassdiagrammet i figur 1 och skriva ett program som fyller en lista, samlingen List<T>, med referenser till ett antal Car-objekt. Därefter ska du sortera listan och presentera den igen.

### Klassen Car

Objekt instansierade från klassen Car ska på ett enkelt sätt kunna presenteras och sorteras.



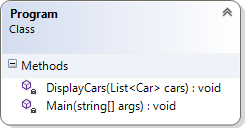
Figur 1. Klassdiagram över klassen Car som implementerar interfacet IComparable<Car>.

Genom att låta klassen överskugga metoden ToString kan du enkelt skapa en textbeskrivning av varje enskilt objekt som innehåller bilens märke, smeknamn och färg.

En förutsättning för att listan med referenser till Car-objekt ska kunna sorteras är att klassen Car implementerar interfacet IComparable<T>, där T ska ersättas med namnet på klassen, d.v.s. IComparable<Car>. IComparable<T> definierar att metoden CompareTo måste implementeras. Du kan läsa mer om IComparable<T> och CompareTo på onlinehjälpen. De olika Car-objekten ska sorteras på märke, smeknamn och färg. Eftersom det är samma ordning som textbeskrivningen bestämmer du enklast om ett objekt ska sorteras för ett annat eller inte genom att jämföra textbeskrivningarna med varandra.

### Klassen Program

Klassen innehåller två statiska metoder, DisplayCars och Main. Listan med bilar ska ju presenteras två gånger, innan sorteringen och efter sorteringen, varför det är lämpligt att bryta ut den koden från metoden Main och placera den i en egen metod. På så sätt undviker du att behöva skriva koden som loopar igenom och presenterar listans innehåll två gånger.



Figur 2. Klassdiagram över klassen Program.

Metoden Main ska skapa en lista och fylla den med referenser till Car-objekt (som du hårdkodar, d.v.s. du läser inte in något från tangentbordet). Lägg även till en null-referens i listan! Listan ska därefter presenteras genom att du anropar DisplayCars. Sortera därefter listan och presentera den på nytt, återigen genom att anropa DisplayCars.

I metoden DisplayCars använder du gärna en ”foreach”-sats för att iterera igenom listan med referenser till Car-objekt. Vill du att en text ska skrivas ut för null-referensen måste du själv ta hand om det.

## Testa programmet

Har du implementerat allt korrekt ska du få följande resultat av en programkörning:



Figur 3.

I figur 3 presenteras listan med bilar innan och efter sorteringen. Lägg märke till null-referensen.

## Mål

Efter att ha gjort uppgiften ska du känna till hur du:

* Använder en samling (*collection*) av typen List<T>.
* Låter en klass implementerar interfacet IComparable<T> så att objekt instansierade från klassen kan sorteras.

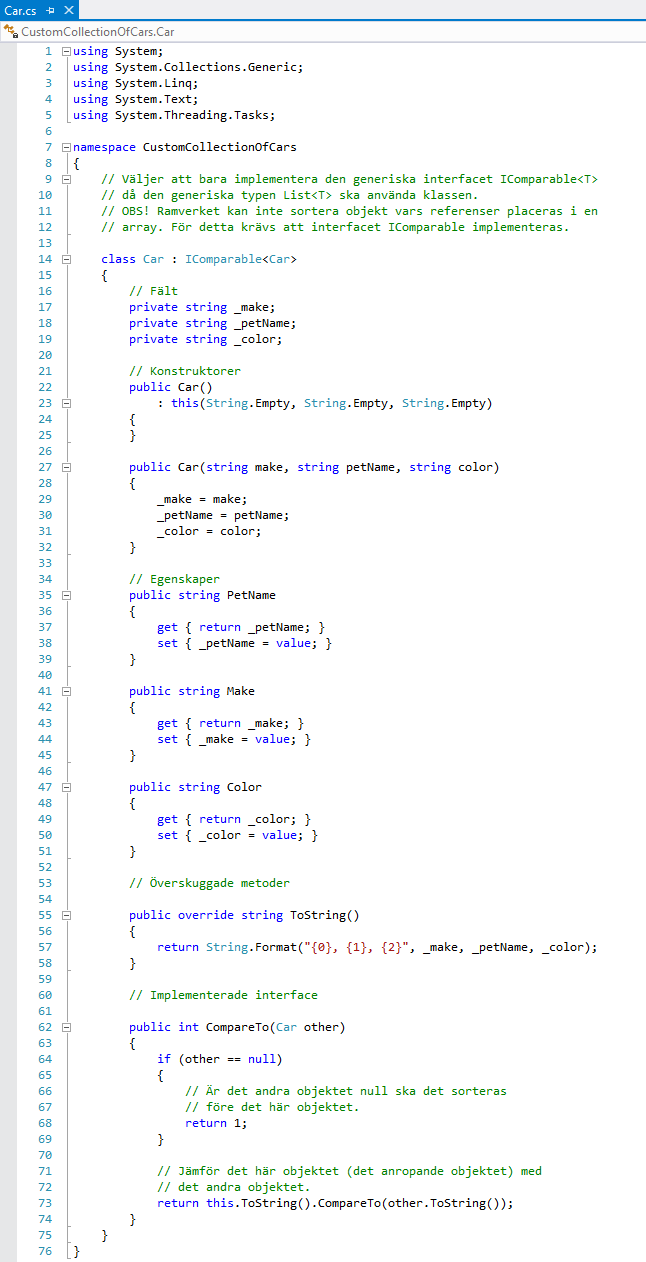
## Tips

Läs om:

* List<T> i kurslitteraturen, kapitel 16, under underrubriken *”List Collections: List<T>”*.

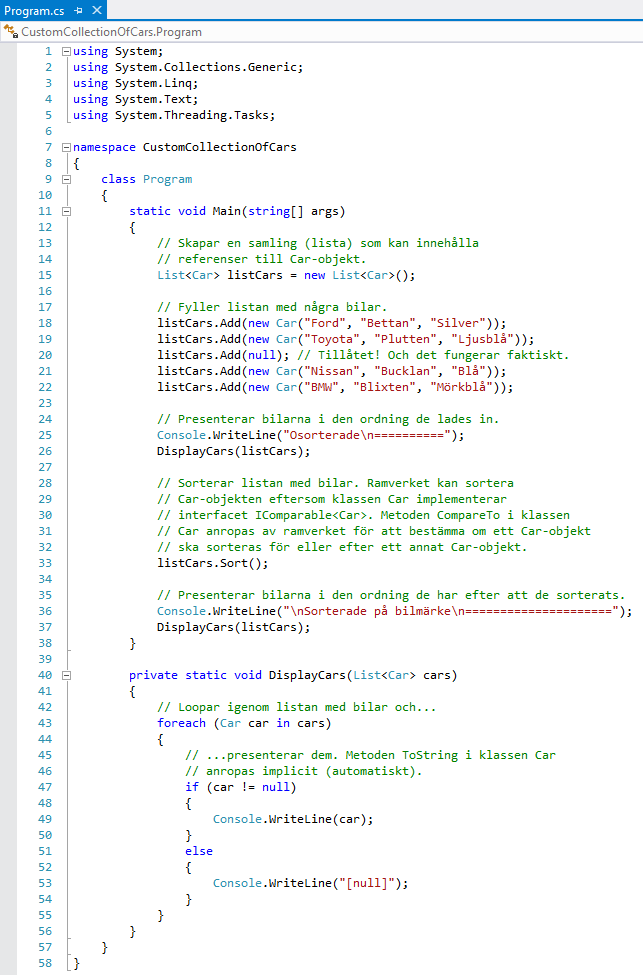
# Lösning

## Car



Figur 4. Implementationen av klassen Car.

## Program



Figur 5. Implementations av klassen Program.

## Kommentar

Hanteringen av listan i metoden Main är intressant. Till skillnad mot traditionella arrayer sätts inte dess storlek då den skapas. Listan skapas med new List<Car>() och för att lägga till nya element i listan anropas metoden Add. Klassen List<T> har ett flertal metoder varav Sort är en. För att Sort ska fungera måste objekten som referenserna i listan refererar till att kunna sorteras.

Klassen Car implementerar interfacet Comparable<T> genom metoden CompareTo. I och med detta kan objekt instansierade från klassen jämföras med varandra.

**OBS!** Comparable<T> är ett nyare interface (kom med .NET Framwork 2.0) och klasserna Array och ArrayList (fanns redan i.NET Framework 1.0) kan inte använda det. Används dessa klasser istället för generiska samlingar som List<T> måste interfacet IComparable också implementeras.

Metoden CompareTo börjar med att kontrollera om parametern other refererar till null eller inte. Refererar other till null ska det anropande objektet sorteras efter, enligt dokumentationen för CompareTo, varför ett tal större än 0 returneras. Refererar other till ett Car-objekt jämförs strängar som ToString returnerar och bilarna sorteras från A till Ö.