3.4 Hantering av array med referens

Man kan effektivisera hanteringen av arrays inte bara med **foreach**-satser utan även genom att använda sig av en s.k. *initieringslista* som slår ihop definitionen med initieringen – en kortform som ersätter koden **new**, men bibehåller dess egenskaper:

```
// ArrayRef.cs
// Initieringslista: Kortform för definition och initiering
// av en array i en och samma sats, inkluderar new implicit
// Utskrift av arrayens element med foreach-satsen
using System;

class ArrayRef
{
   static void Main()
   {
      int[] no = { 64, 86, 34, -6 };     // Initieringslista:
                                         // Definition OCH ini-
                                         // tiering av en array
//    int[] no = new int[4] { 64, 86, 34, -6 }; Gör samma sak

      Console.Write("\n\tVärdena från arrayen skrivs ut " +
                    " med referensen:\n\n\t");
      foreach (int element in no)
         Console.Write(element + "\t");
      int[] copy = no;                  // Ny referens copy tillde-
                                        // las referensen no
      Console.Write("\n\n\tArrayens värden skrivs ut" +
                    " med den nya referensen copy:\n\n\t");
      foreach (int element in copy)
         Console.Write(element + "\t");
      Console.WriteLine("\n\n\tEndast referensen kopieras," +
                                         " inte arrayen.\n");
   }
}
```

En körning visar att värdena i initieringslistan som först tillelas arrayen **no** verkligen kopierats över till arrayen **copy**, för det är de som skrivs ut:

```
      Arrayens värden skrivs ut med referensen no:

      64      86      34      -6

      Arrayens värden skrivs ut med den nya referensen copy:

      64      86      34      -6

      Endast referensen kopieras, inte arrayen.
```

Både definitionssatsen och initieringssatserna i programet **ArrayObj** (sid 99) – det är de 5 första satserna i **Main()** – kan slås ihop till en enda sats:

```
int[] no = { 64, 86, 34, -6 };
```

Satsen ovan är bara en förkortning på:

```
int[] no = new int[4] { 64, 86, 34, -6 };
```

Dvs initieringslistan kan skrivas efter **new int[4]** som egentligen skapar eller definierar arrayen. Men **new int[4]** får utelämnas. Detta visar att den förkortade versionen gör två saker: Först, fram till tilldelningstecknet definieras referensen **no** (*utan* någon uppgift om arrayens storlek). Sedan, från och med tilldelningstecknet tilldelas arrayen **no**:s element fyra värden som står i en kommaseparerad lista grupperad inom klamrarna **{ }** som kallas arrayens *initieringslista*. Kortformen gör precis samma sak som satsen med **new**. Kompilatorn får informationen om arrayens storlek genom att i initieringslistan räkna antalet element inom klamrarna **{ }**. Det är inte ens tillåtet att explicit ange det korrekta antalet element inom hakparenteserna **[ ]**. Det blir kompileringsfel om man gör det, därför att **no** endast är en referens till en array, inte arrayen själv. Observera även att man inte får använda initieringslistan separat utan endast i samma sats som definitionen.

Valet av variabelnamnet **copy** kan vara missledande i följande sats av programmet **ArrayRef** om man inte beaktar skillnaden mellan referens och array:

```
int[] copy = no;
```

copy blir nämligen en kopia av referensen **no** i satsen ovan, inte av arrayen – en ny referens som kommer att peka på samma array som den gamla referensen **no** pekar på. Det skapas ingen ny array eftersom det varken finns någon **new** eller någon initieringslista som skulle ersätta **new**. Anledningen till detta är – som vi konstaterat tidigare – följande viktigt faktum:

> En array i C# är alltid ett objekt som behöver en referens.

För att skapa ett objekt måste en **new**-sats skrivas. En referens definieras utan **new**.

Minnesmässigt lagras arrayen på *en och samma* adress som från programmet kan nås med referenserna **no** eller **copy**: