Övningar till kapitel 3

3.1 Modifiera programmet **`ArrayOfRef`** (sid 70). Deklarera klassen **`Fish`**:s datamedlemmar som **`private`** och metoderna som **`public`**. Förse klassen med en konstruktor och en strängrepresentationsmetod **`AsString()`**. I övrigt ska det modifierade programmet göra samma sak som det ursprungliga.

3.2 Skriv ett program som läser in 10 heltal från konsolen, lagrar dem i en array och skriver ut dem i omvänd ordning.

3.3 Skriv ett program som slumpar fram 1000 heltal mellan 60 och 140 (tänkbara hastigheter på en motorväg), lagrar dem i en array kallad **`hastighet`**, beräknar och skriver ut deras medelvärde med förklarande text. Använd klassen **`RandArray`** (sid 73) som extern modul.

3.4 Skriv ett program som läser in en sträng, lagrar den i en array av **`char`** och skriver ut den baklänges. Använd tekniken i programmet **`EncryptCharTest`** (sid 88) för att omvandla den inlästa strängen i en array av **`char`**.

3.5 Skriv ett program som läser in text i gemener, lagrar den i en array av **`char`** och skriver ut den framhävd i versaler och med mellanslag mellan varje tecken.

3.6 Skriv ett program som frågar efter användarens för- *och* efternamn, hälsar sedan användaren i en utskrift med fullständiga namnet, förnamnets längd samt efternamnets första och sista bokstav. Lös uppgiften generellt utan att använda information om något speciellt för- och efternamn.

3.7 Skriv ett program där **`Main()`** läser in en persons fullständiga namn och hälsar tillbaka med namnets initialer. Dessa ska bestämmas och skrivas ut i en annan metod – med huvudet **`static void Initials(char[] name)`** – som anropas i **`Main()`**.

3.8 Modifiera programmet **`Lista`** (sid 91) så att sorteringen av slumptalen görs med vår egen bubbelsorteringsmetod **`sort()`** (sid 78) istället för med den fördefinierade **`List`**-metoden **`Sort()`**. Testa först med array-notationen som **`sort()`** är skriven i. Försök sedan att skriva om **`sort()`** till en **`List`**-version.

3.9 Följande algoritm i pseudokod beräknar summan av de första *n* positiva heltalen:

```
Start Sum(n)
    OM n = 1
        returnera 1
    ANNARS
        returnera n + Sum(n-1)
Slut Sum(n)
```

Vi vill låta datorn beräkna: Sum(10) = 1 + 2 + 3 + . . . + 9 + 10
Sum(100) = 1 + 2 + 3 + . . . + 99 + 100 osv.

a) Implementera algoritmen som en metod i C#. Bygg metoden in i ett program och anropa den för n = 10, 100, 1000 och 10 000. Skriv ut resultaten på ett användarvänligt sätt.

b) Är algoritmen (metoden) rekursiv? Om ja, var finns rekursiviteten? Förklara!

c) I vilken ordning adderar algoritmen de pos. heltalen, fram- eller baklänges? Förklara varför den gör så.

3.10 Skriv en rekursiv metod **`Faculty()`** som implementerar den matematiska funktionen *n Fakultet*: $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$

Testa metoden i en klass genom att anropa den för n = 1, 2, 3, … , 12.
Skriv ut resultaten på ett användarvänligt sätt.
Kontrollera dina resultat med en miniräknare.

3.11 Skriv om den rekursiva metoden **`Factorize()`** (sid 102) som implementerar faktoriseringen av primtal, till en *iterativ* metod. Dvs ersätt rekursionen med en vanlig loop, t.ex. **`while`**, och testa den nya metoden genom att anropa den i programmet **`PrimeFactors`** (sid 102).