1.6 Referensvariabler

En *referensvariabel* – kort *referens* – är en ny typ av variabel:

> Referens är en variabel vars datatyp är en klass.

Ett exempel är variabeln **a** i programmet **EmpTest** (sid 24) där **a** deklareras till **Emp** som är en klass: `Emp a;` Med satsen `Emp a;` skapas inget objekt utan endast en referensvariabel. Självaste objektet skapas med koden `new Emp()`.

Den allmänna formen *hur* referensvariabler kopplas till objekt, ser ut så här:

```
Klass referensvariabel = new Klass();
```

Exempel med klassen **Emp**: `Emp a = new Emp();`

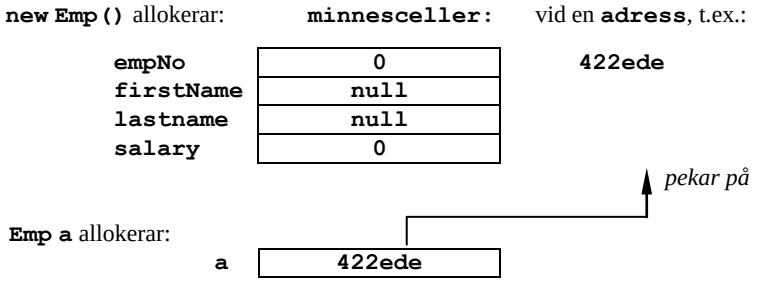
new kallas för *minnesallokeringsoperatorn* och allokerar (reserverar) minnesutrymme för **Emp**-objektet. Storleken bestäms av klassen **Emp**. Minnesadressen tilldelas referensvariabeln **a** som deklareras i samma sats. Operatorn **new** följs *inte* av parenteser. Klasserna på tilldelningens båda sidor måste vara identiska: **new Emp()** allokerar minne för lagring av ett **Emp**-objekt och returnerar minnets adress till referensen **a** vars datatyp är klassen **Emp**. Tilldelning till en referens av en annan klass skulle ge kompileringsfel.

Referensvariabler är ett verktyg för att kunna komma åt objekt. De ersätter objektens namn. Man kan jämföra detta med tyglar till en häst, där tyglar är referensen och hästen objektet. Eller fjärrkontrollen (referens) till en TV (objekt). Båda är lätthanterade verktyg för styrning av tunga objekt. Andra jämförelser är länkar till webbsidor eller namnskyltar: Den lilla skylten *Gamla Stan* (referens) pekar på den stora ön *Gamla Stan* (objekt). En referens lagrar den ”lilla” minnesadressen till ett stort objekt och tar jämfört med det tunga objektet så litet minne som en vanlig **int**: 4 bytes. Detta innebär minnesekonomiskt en stor effektivitet vid exekvering av minneskrävande program.

Vid vanliga variabler av enkel datatyp hänvisar vi till minnescellerna med *variabelnamn*. Vid objekt gör vi det med deras *adresser* i form av objektens referenser. När man vant sig vid att använda referenser kan man t.o.m. tycka att hanteringen av data via adresser är det naturliga sättet, vilket inte är någon dum idé med tanke på att variabelnamn ändå är en slags mjukvarulänk till hårdvarans minnesadress. Det är exakt samma sak som C++ gör med *pekare*.

Referensen ”pekar” på objektet

Vad händer exakt när satsen **`Emp a = new Emp();`** exekveras i programmet **`EmpTest`**? För det första deklareras referensen **`a`**, för det andra skapar **`new`** ett **`Emp`**-objekt dvs allokerar minnesutrymme för objektet. För det tredje tilldelas minnesadressen till referensen **`a`**. Adressen tilldelas referensen **`a`** vilket gör att **`a`** nu *pekar på* den av **`new`** allokerade minnescellen, så att minnesbilden i datorns RAM efter satsen ovan ser ut så här:



Tilldelningsoperatorn mellan **`Emp a`** och **`new Emp()`** gör att objektets adress hamnar i referensvariabeln **`a`**:s minnescell vilket resulterar i att **`a`** *pekar på* objektet och vi därför kan och måste referera till objektet genom att använda refrensen **`a`**. Alla objekt i C# kan hanteras med referenser. Det är avgörande att inte förväxla *objekt* med *referens*: Det är två helt olika typer av saker och ting med två olika minnesplatser och olika egenskaper som relateras till varandra på det ovan beskrivna sättet. Men varför hamnar **`0`** och **`null`** i objektets minnesceller? Och vad är överhuvudtaget **`null`**? Att de hamnar där beror på anropet av default konstruktorn **`Emp()`** som automatiskt initierar datamedlemmarna **`empNo`**, **`firstName`**, **`lastname`** och **`salary`** till s.k. defaultvärden.

Datamedlemmarnas defaultvärden

Samtidigt som **`new`** allokerar minne för objektet anropar koddelen **`Emp()`** en metod som initierar objektets datamedlemmar med vissa defaultvärden som beror på deras datatyper. Denna metod som heter klassens *konstruktor* har samma namn som klassen samt den viktiga uppgiften att initiera objektets datamedlemmar. En *default konstruktor* skapas alltid automatiskt med när man deklarerar en klass utan att själv skriva en konstruktor. Default konstruktorn initierar datamedlemmarna, när ett objekt skapas, automatiskt till följande defaultvärden:

`0`	om de är tal,
`null`	om de är referenser,
`'\0'`	om de är av typ **`char`**,
`false`	om de är av typ **`bool`**.

Exempel på **`0`** och **`null`** har vi i programmet **`EmpTest`**: datamedlemmen **`a.firstName`** initieras till **`null`** eftersom den är av typ referens, **`a.empNo`** initieras

till `0` eftersom den är av typ `int` och `a.salary` som är en `float` får `0` som initialvärde. Källan till datatypinformationen är förstås klassen `Emp` där dessa datamedlemmar är deklarerade till sina resp. datatyper (sid 23). Alla objekt är ju kopior av klassen. Vi hittar i klassen `Emp` deklarationen `String firstName;` men hävdar ändå att datamedlemmen `firstName` inte är en sträng utan en *referens* till strängar. Anledningen är att `String` är en *klass* och inte en enkel datatyp. Variabler vars datatyper är klasser är referensvariabler. Därför är `firstName` en referens och inte en sträng. Man kan bevisa det genom att t.ex. lägga in följande rader i programmet `EmpTest` (sid 24) mellan objektets definition `a = new Emp();` och tilldelningen av nya värden till objektets datamedlemmar (som börjar med `a.empNo = 123;`) dvs när datamedlemmarna fortfarande har defaultvärden:

```
if (a.firstName == null) Console.WriteLine("null");
if (a.firstName == "")  Console.WriteLine("tom sträng");
```

Man kommer att få `null` och inte `tom sträng`, därför att det är `null` som är defaultvärdet för referenser, vilket visar att `a.firstName` är en referens och inte en sträng.

Referensen `null`

`null` i C# betyder *inget objekt*, dvs en referens som inte pekar på något objekt. Själva ordet `null` hittar man bland språkets reserverade ord. `null` är ett värde som kan tilldelas referensvariabler, nämligen när referensen inte lagrar någon adress till ett objekt. Just därför kallas `null` för referensernas *defaultvärde*. Även om `null` representeras i datorn med `0`, får det inte förväxlas varken med *talet* `0` eller med *tecknet* `'0'`. `null`:s datatyp är varken `int` eller `char`, utan `null` är av referenstyp, dvs ett värde som endast kan tilldelas referenser. Och vi vet ju att referensvariablernas datatyper alltid är klasser. En variabel av referenstyp kan endast lagra minnesadresser.

> `null` är i C# referensvariablernas defaultvärde med ASCII-koden 0. `null` är av referenstyp.
> En referens med värdet `null` pekar på inget objekt.

`null`-referenser kan jämföras med *parkerade* bilar, om bilar *i fart* jämförs med referenser som pekar på objekt. Man kan ”sätta igång” `null`-referenser när som helst genom att tilldela objektadresser till dem. Omvänt kan man ”parkera” dem igen genom att tilldela `null` till dem. Observera att `null`-referenser inte alls är samma sak som oinitierade referenser. Till skillnad från `null`-referenser leder oinitierade referenser precis som alla andra oinitierade variabler till kompileringsfel när de används. Till skillnad från oinitierade referenser *har* `null`-referenser ett värde, bara att deras värde `null` inte är en adress till ett befintligt objekt utan bara en symbol som betyder ”referens i väntan på att få en objektadress” precis som en parkerad bil i väntan på att sättas i fart. Man använder `null`-referenser i C#-pro-

gram för att initiera referensvariabler direkt efter deklarationen när det vid deklarationens tidpunkt inte kan avgöras vilket objekt de ska bindas till. På så sätt vill man förhindra oinitierade referenser som alltid bär risken med sig att de används av misstag innan de binds till ett objekt. Det är rekommenderad att alltid initiera sina lokala referensvariabler till **`null`** om de inte kan tilldelas ett objekt när de skapas. Förväxla inte **`null`** med nolltecknet:

Nolltecknet `'\0'`

I listan över defaultvärden till de olika datatyperna på förra sidan dyker upp *nolltecknet* som defaultvärdet för teckenvariabler dvs till datatypen **`char`**. Man stöter på det när man försöker skriva ut datatypen **`char`**:s undre gräns och använder sig av explicit typkonvertering för att omvandla den till ASCII-koden 0. Sedan kan man framställa det oskrivbara och osynliga nolltecknet med hjälp av escapesekvensen **`'\0'`**. Då känns det naturligt att det allra första tecknet i ASCII-tabellen med koden 0 används som defaultvärde för teckenvariabler. Fysiskt består det alltså av av 2 bytes dvs 16 bitar fyllda med endast nollor. På den fysiska bitnivån representeras även **`null`** av nollor. Däremot skiljer de sig på den logiska programnivån via sina datatyper: Medan nolltecknet är av typ **`char`** är **`null`** av typ referens.

> Nolltecknet **`'\0'`** är i C# **`char`**-variablernas defaultvärde med ASCII-koden 0.
> Nolltecknets datatyp är **`char`**.

Anonyma objekt

Man kan även skapa s.k. *anonyma objekt*, dvs objekt utan referens. Anonyma objekt skapas direkt när man behöver dem. T.ex. om metoden **`AsString()`** är definierad i klassen **`Emp`** kan man anropa den så här: **`new Emp().AsString()`**. Då är **`new Emp()`** ett *anonymt* objekt av klassen **`Emp`**, därför att det skrivs utan referens (namn). Nackdelen med anonyma objekt är att man inte längre kan komma åt dem efteråt. Testa gärna själv: Skriv i programmet **`EmpTest`** (sid 24) **`new Emp().AsString()`** istället för **`a.AsString()`** där **`a`** är en referens till ett **`Emp`**-objekt.

De tomma parenteserna efter klassnamnet i exemplet **`Emp()`** och även i den allmänna formuleringen Klass() får absolut inte utelämnas även om de är tomma. De anropar klassens konstruktor (sid 28), närmare bestämt default konstruktorn (sid 33). Allt detta under förutsättningen att man inte har definierat en egen konstruktor som i så fall skulle slå ut default konstruktorn. Har man däremot definierat en egen konstruktor måste parerentesen matcha den egna konstruktorns parameterlista.