C# - Ett objektorienterat programspråk

# Bakgrund:

C# är framtaget av Microsoft och var från början closed source. Företaget hade ett behov av ett objektorienterat programspråk för sina interna miljöer. Målet var att det skulle ha de kvalitéer som C[[1]](#footnote-0) hade. Det skulle vara ett GPL[[2]](#footnote-1)-baserat programspråk, eller som de själva uttrycker det C-lite. GPL är kort för general purpose programming language och idag skulle vi nog anse det uttryckssättet lite konstigt. De stora språken, som Python, C#, JavaScript och Perl är idag GPL. Så har det däremot inte alltid sett ut utan programspråk har varit nischade för funktion. Ett exempel på ett sånt programspråk som fortfarande används idag är SQL.

## Objektorienterad programmering:

Föregångaren till objektorienterad programmering heter procedurellprogrammering. Den innebär i korthet att man fokuserar på datastrukturer, variabler och subrutiner.[[3]](#footnote-2) Det fungerar utmärkt, men har sina nackdelar. Koden blir långa sjok och subrutiner blir svåra att hitta. Jag har stött på Assembly och Basic, som procedurella språk.

Objektorienterad programmering innebär å andra sidan att man bryter ner delar av programmet till objekt. I C# gör vi detta genom att skapa klasser. Dessa klasser kan innehålla variabler och olika data, som hanteras genom metoder. Fördelen med detta är att om något behöver ändras i ett program, så behöver inte hela koden förändras, utan det räcker att man förändrar koden som hanterar objektet. Med detta uppnår man ett av huvudsyften inom objektorienterad programmering. Den blir skalbar. Den går alltså att skala upp och ner beroende på behov.

# C# - Grundbultarna:

## Variabler:

Lekmannaförklaringen av en variabel är att datorn behöver någonstans att minnas vad användaren eller programmet gör med vissa värden. Den förklaring som jag ger är att man kan se en variabel som en låda som man skriver en etikett på. Ex Jag skriver nummer på en låda. Jag ber sedan datorn minnas mitt nummer. Datorn stoppar då ner numret i lådan. Nästa gång jag frågar vad numret är, öppnar datorn lådan och kollar vad numret är och berättar det för mig. Datorn i sig minns aldrig vad numret är, men den minns var lådan finns och att den kan kolla i lådan.

Det vi gör i C# och många andra programspråk är att vi deklarerar variabler. Vi säger till programmet att vi vill ha en variabel och att vi vill ha den av en viss typ. Dator allokerar då en del av sitt minne till just den variabel. I många programspråk behöver vi säga åt programmet vilken typ av variabel det är. Anledningen är att datorn behöver allokera olika mycket minne åt olika typer av variabler.

Minnet delar upp i stack och heap. Stacken är begränsad och har bara en viss mängd minne. Det är iofs heapen också, men den har möjligheten att använda virtuellt minne, vilket inte stacken har. Stacken är snabbare en heapen iom att där är varje bit direktallokerad. Det är den inte i heapen utan allokeras efter behov. Stacken jobbar med Last In First Out, den allokerar alltså från toppen hela tiden och tar bort ifrån botten. Den är som sagt snabbt, men kan bli överfull om den anropas för mycket. Då får man en så kallad Stack Overflow.[[4]](#footnote-3) Det är ett felmeddelande som betyder att stacken blivit full och programmet kan inte fortsätta.[[5]](#footnote-4)

## Typer:

C# har en antal olika typer inbyggda. De faller under två kategorier.

### Värdetyper:

Hanterar olika numeriska värden. Det kan vara heltal, decimaltal, osv. De vi hanterat mest under kursen har varit int, char, float och bool.

Int, float, decimal och de andra som hanterar siffror handlar egentligen bara om hur mycket minne de allokerar. int allokerar minst mängd minne och float allokerar mest. Float hanterar det som på svenska heter flyttal[[6]](#footnote-5). Det är en uppskattning av det vi kallar reella tal. Anledningen till att det är en uppskattning, eller avrundning är att datorn i maskinkoden[[7]](#footnote-6) kommer arbeta med det binära talsystemet och översättningar mellan binära tal och decimaltal leder i många fall till avrundningar.

Char jobbar med en annan typ av minnesallokering. En char är åtta bits, eller en byte lång. Det ger oss 2^8 = 256 möjliga kombinationer för tecken. I C# har man däremot valt att utvidga char till 16 bitar, vilket ger 2^16 möjliga kombinationer för tecken.[[8]](#footnote-7)

Bool jobbar binärt. Det är en variabel som kan vara sann eller falsk. Den jobbar alltså binärt och är 1 eller 0. Den är som standard satt till 0, dvs false. Det innebär att om vi vill att den ska vara sann, måste vi deklarera det. Värdet av uppfinningen av variabeln bool är svårt för oss att förstå idag. Vi använder den utan att blinka, men att gå från att bara kunna kolla om ett variabel är sann eller inte, från att behöva kolla om en metod returnerar ett visst värde är svår att greppa.

### Objekttyper:

Objekttyper är så kallade referenstyper. De refererar tillbaka till sitt objekt. Värdetyper innehåller enbart sina värden eller data.[[9]](#footnote-8) Fördelen med objekttyper är att flera variabler kan referera tillbaka till ett och samma objekt. Det går inte med värdetyper. Antingen får man ett fel i kompileringen, eller så kommer de innehålla ett nytt värde om man försöker använda dem flera gånger.

Det finns tre inbyggda i C#: dynamisk, objekt och sträng.

Det vi jobbat med mest är objekt och strängar.

Objekt är fria att innehålla vad vi vill att objektet ska ha. Det kan alltså innehålla en mängd olika typer av variabler.

Strängar används för att hantera text. Det finns även inbyggda metoder i C# för att hantera strängar. De kan alltså manipuleras på olika sätt. Vi har gått igenom ett flertal sätt att hantera strängar, men ett av de första vi gick igenom var konkatenering av strängar. Dvs möjligheten att lägga ihop en eller flera strängar till en ny.[[10]](#footnote-9)

## Asynkrona metoder:

I de program vi skrivit hittills har vi egentligen inte behövt asynkrona metoder. Våra program har hanterats i minnet och åtkomsten till data har varit snabb. Våra listor har inte heller varit särskilt långa, så metoderna har hanterat dem så snabbt att vi inte ens märkt att de körts.

Det här är inte fallet när vi t ex läser stora filer, eller jobbar med stora databaser. Då kan vi stöta på metoder, som tar tid och därmed fryser vårt program. Det är inte speciellt bra för användaren. För att komma runt det här kan man använda asynkrona metoder. Asynkron betyder att den går ut ur synk. Det innebär att även om en metod tar tid, så kan den göra det i bakgrunden och vi kan be andra metoder vänta med sina beräkningar tills den andra metoden är klar.

Som jag nämnt innan blir de asynkrona metoderna viktiga ju tyngre och större våra program blir. Jobbar vi mot databaser blir de dessutom i många fall nödvändiga.[[11]](#footnote-10)

## Exceptions:

Det finns inbyggda funktioner för exceptions i C#, men det finns också möjlighet att definiera egna exceptions. När man har med användare att göra, så kan man inte alltid förutse vad som händer. Vår kod kan förutspå allt, utan vi har därför möjligheten att anropa exceptions. Vi anger för användaren att något oväntat har hänt och kan dessutom iom detta ge information till användaren om vad vi tror gått fel.

Exceptions kan också hjälpa oss när vi programmerar och inte är säkra på hur vår kod fungerar. Vi ber då programmet om en exception och kan se vad som gick fel. Visual Studio har möjligheten att sätta brytpunkter, vilket ger oss samma möjlighet. Vi kan då köra programmet och komma till brytpunkten. Där och då kan vi se vad våra variabler och objekt innehåller för data, vilket är oerhört hjälpsamt.

## .NET

Det finns en inbyggd klass för filhantering. Klassen kan skapa och ta bort filer och mappar. Den kan även läsa och skriva till dem.

Det finns också en klass som heter stream. Den gör möjligheten att göra allt som file-klassen gör, fast med stream kan man välja om det ska ske synkront eller asynkront.

.NET har också inbyggda funktioner för att hantera tid. Den jag bekantat mig mest med är DateTime, men det finns derivat av den som t ex DateOnly och TimeOnly. Gemensamt för dem är att de hanterar tid. Det finns även inbyggda funktioner i .NET för att göra parses på DateTime. [[12]](#footnote-11) Det ger oss möjligheten att enkelt göra DateTime till strängar och tvärtom. DateTime till sträng är intressant om vi t ex vill visa något för en användare gällande tid i en textbox.

## LINQ och reguljära uttryck

LINQ ger oss möjligheten i C# att skapa listor, och ställa frågor till databaser. Vi behöver inte skriva egna metoder för att hantera listor, utan det sköter LINQ automatiskt. LINQ är ett bibliotek som spänner över en massa olika funktioner och programspråk, men det intressanta är Q:et i LINQ[[13]](#footnote-12). Det står för query, alltså möjligheten att ställa frågor. Det här görs på ett strukturerat och enkelt sätt och påminner om sättet att göra det i SQL. Vi använder t ex where för undersöka ett påstående både i LINQ och SQL.

IEnumerable<string> query = fruits.Where(fruit => fruit.Length < 6);

Tar ut alla frukter som har färre än 6 bokstäver i listan. Uttrycker med fat arrow => är ett lambdauttryck och de kan användas för att utföra en uppgift som inte behöver en egen metod, utan behöver göras precis där och då.[[14]](#footnote-13)

1. https://en.wikipedia.org/wiki/C\_(programming\_language) [↑](#footnote-ref-0)
2. https://en.wikipedia.org/wiki/General-purpose\_programming\_language [↑](#footnote-ref-1)
3. https://en.wikipedia.org/wiki/Procedural\_programming [↑](#footnote-ref-2)
4. https://www.techtarget.com/whatis/definition/stack-overflow [↑](#footnote-ref-3)
5. https://medium.com/c-programming/c-memory-management-part-1-c03741c24e4b [↑](#footnote-ref-4)
6. https://sv.wikipedia.org/wiki/Flyttal [↑](#footnote-ref-5)
7. https://sv.wikipedia.org/wiki/Maskinkod [↑](#footnote-ref-6)
8. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/char [↑](#footnote-ref-7)
9. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/reference-types [↑](#footnote-ref-8)
10. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/reference-types [↑](#footnote-ref-9)
11. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/async [↑](#footnote-ref-10)
12. https://learn.microsoft.com/en-us/dotnet/api/system.datetime.-ctor?view=net-7.0 [↑](#footnote-ref-11)
13. https://www.tutorialsteacher.com/linq/what-is-linq [↑](#footnote-ref-12)
14. https://www.geeksforgeeks.org/lambda-expressions-in-c-sharp/ [↑](#footnote-ref-13)