Introduktion

Den objektorienterade modellen
- Grundläggande begrepp
- Klass
- Klassen som abstraktion

Klasser -objekt-attribut - metoder
- Klasser
- Metoder
- Attribut

Objekt
- Vad är ett objekt?

Relationer mellan objekt/klasser
- Arv
- Arv (- är en)
- Multipelt arv
- Aggregat
- Association
- Användning

Polymorfism
- Förutsättningar för polymorfism..
- Persistence

Den objektorienterade modellen

Grundläggande begrepp

Nedanstående begrepp måste stödjas för att ett språk eller verktyg till fullo skall stödja den objektorienterade modellen;

```
Klass
Abstraktion
Inkapsling
Objekt
Metoder
Attribut
Arv
Aggregat
Association
Polymorfism
Beständighet (persistence)
```

Klass

En klass är en beskrivning av olika objekt som har något gemensamt (av samma typ). Klassen kan betraktas som en mall för vilka data, metoder och meddelanden som är giltiga för en viss typ av objekt. Ett objekt skapas när ett annat objekt tar emot ett meddelande om att detta ska ske. Objektet skapas då enligt den beskrivning som finns i klassdefinitionen (mallen). Objektet är en instans (förekomst) av klassen (typen).

Klassen som abstraktion

När vi definierar en klass gör vi en förenkling av verkligheten. Vi gör en abstraktion där vi tar med sådant som är intressant just i det här fallet. Vi inför en abstrakt datatyp (ADT). Man väljer sedan vilken förenklingsgrad (abstraktionsnivå) som är lämplig vid varje tillfälle. Det är viktigt att klassens gränssnitt är väldesignat och kommer att kunna vara bestående. Denna process kan liknas vid vetenskaps-mannens strävan att systematisera verkligheten med ett speciellt perspektiv eller barnets (dvs människans) strävan att systematisera omvärlden för att kunna förstå den.

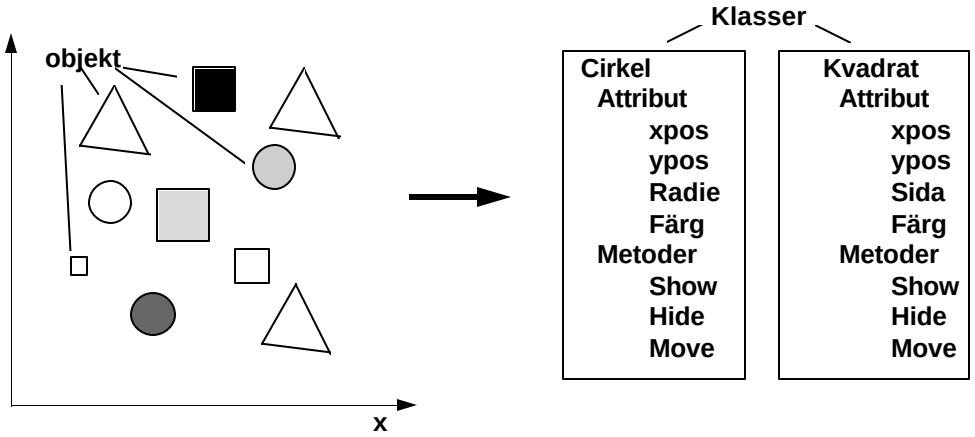
Klasser -objekt-attribut - metoder

Inom ett problemdomän kan vi identifiera objekt som liknar varandra i avseende på utseende, uppförande, funktionalitet osv. De tillstånd och egenskaper som är intressanta och gemensamma för objekt som liknar varandra (är av samma typ) blir attribut och metoder. Attribut är de egenskaper som behövs för att beskriva varje objekts tillstånd. Metoder är meddelanden som vi

vill kunna skicka till ett objekt och som förändrar eller använder objektets attribut (och därigenom dess tillstånd).

Klasser

Antag att vi i ett problemdomän identifierar objekt som har viss form, position, storlek och färg. Vi vill kunna skicka meddelande som resulterar i att objekten förflyttar sig, blir synliga och gömmer sig. Efter en analys finner vi att några av objekten kan hänföras till klasserna **Cirkel** och **Kvadrat**. Med hänsyn till det system som vi skall konstruera är det viktigt dessa objekt har för position, storlek samt färg samt att man genom meddelanden kan be dessa visa, gömma samt flytta sig.



Metoder

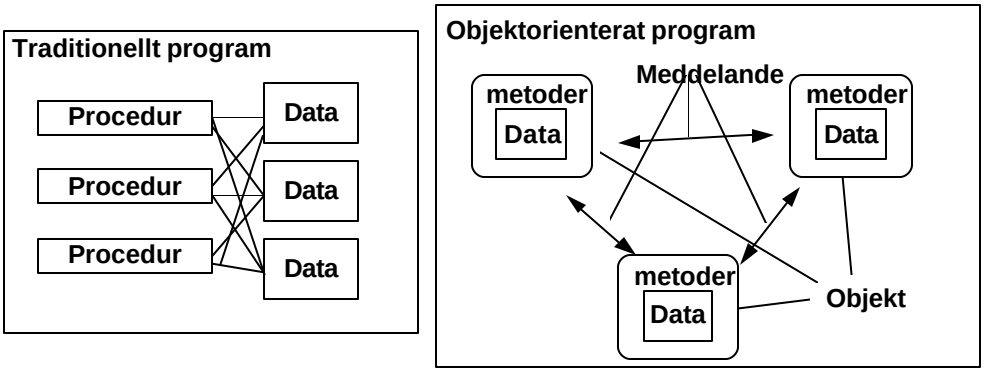
I ett traditionellt program är data passivt. För att åstadkomma något måste vi använda funktioner. Inom objektorientering kan objektet, som respons på ett meddelande agera med hjälp av sina egna metoder.Vi använder orden metod, operation, meddelande, tjänst och medlemsfunktion som synonymer och vi kan t.v anta att vi hela tiden menar ungefär samma sak.

Attribut

Med hjälp av attribut bestämmer vi bl.a. objektets tillstånd. Vi väljer ofta att endast tillåta förändring av objektets attribut, och därigenom tillstånd, genom anrop av objektets metoder. Vi låter ofta attributen vara inkapslade och därigenom bli del av objektets interna struktur.

Objekt

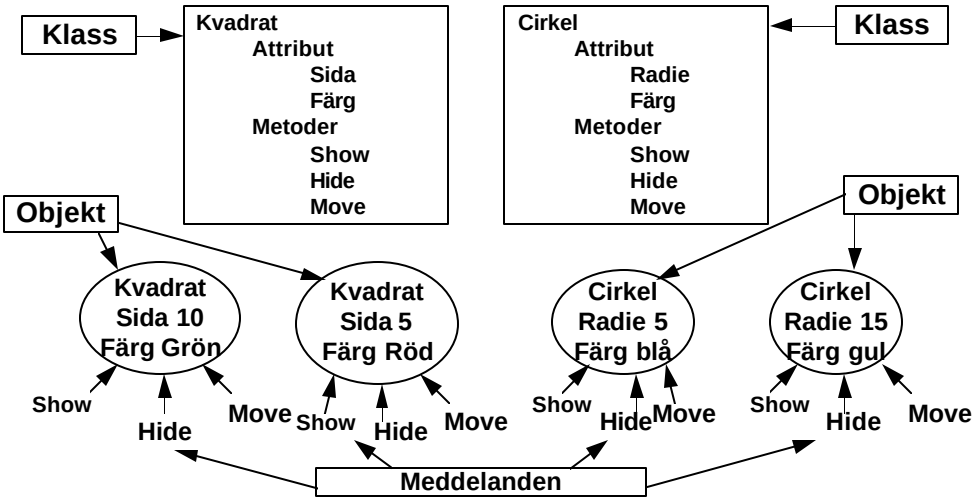
Ett traditionellt program består av funktioner-procedurer och data. Ett objektorienterat program består av objekt. Ett objekt kan sedan innehålla data samt de funktioner (metoder) som ska kunna appliceras på data.



Vad är ett objekt?

Ett objekt är något som existerar och är skapat enligt en mall (klass). Det kan finnas många objekt av samma typ (klass). Man använder också begreppet "instans" när man menar ett objekt . Ett objekt är en instansiering av en klass. Ett objekt kan skapas och sedan raderas. Klassen finns kvar så att vi kan skapa nya objekt senare. En del objekt måste vara bestående (persistant) så att de finns kvar.

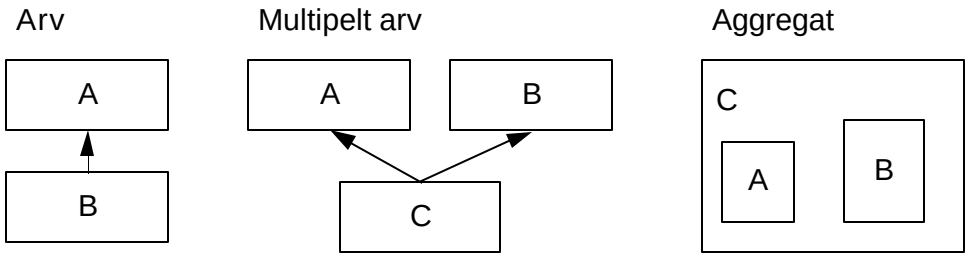
När vi har definierat mallen kan vi börja använda den och skapa objekt. Lägg märke till att vi kan skapa flera objekt av samma typ. Objekten skiljer sig i avseende på tillstånd men är ändå av samma typ.

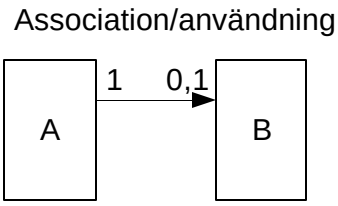


Relationer mellan objekt/klasser

Antag att vi har tre objekt av olika typ; A,B och C. Dessa **kan** vara relaterade till varandra på följande sätt:

```
Arv (ett B är ett A)
Multipelt arv (ett C är ett A och ett B)
Aggregat (ett C består av ett A och B)
Association (ett A kan ha en relation till ett B)
Användning (ett A använder ett B)
```

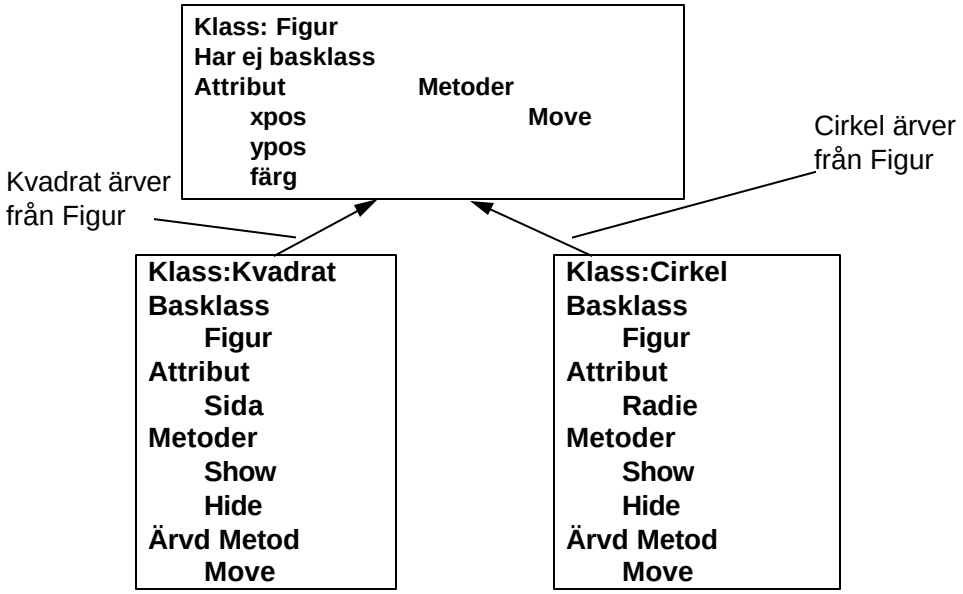




Arv

Genom arv kan den objektorienterade programmeraren konstruera nya klasser (subklasser) genom att låta dessa ärva egenskaper från någon annan klass (basklass). Programmeraren behöver sedan bara tillföra det som skiljer den nya subklassen från basklassen. I objekt (instanser) av den nya klassen finns sedan även de data, metoder och meddelanden som ingår i basklassen.

Ibland finner vi att en klass är ett specialfall av en annan klass. Det är då vi kan använda arv och därigenom ärva egenskaper. Man brukar samla generella egenskaper i en basklass och sedan låta en subklass ärva och eventuellt tillföra ytterligare egenskaper.Genom arv kan vi bygga upp mer eller mindre stora klasshierarkier. Det förekommer också att vi använder multipelt arv vilket betyder att en klass ärver av mer än en klass.

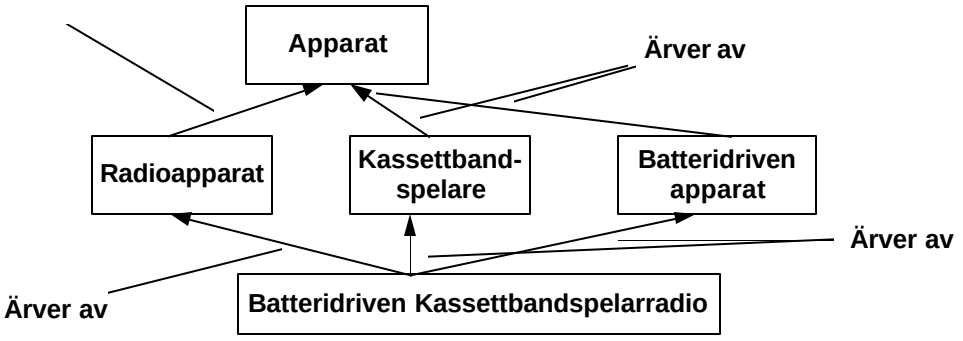


Arv (- är en)

Arv kan gälla om det stämmer att ett objekt av subklasstyp ÄR ETT objekt av basklasstyp. Arv är alltid ett exempel på relationen generell-speciell. I föregående exempel gäller att subklasserna "Kvadrat" och "Cirkel" är specialfall av typen "Figur" som är basklass. En "Cirkel är en Figur" och en "Kvadrat är en Figur". I basklassen samlar vi gemensamma egenskaper. I subklasserna kan vi omdefiniera eller ärva attribut och metoder.

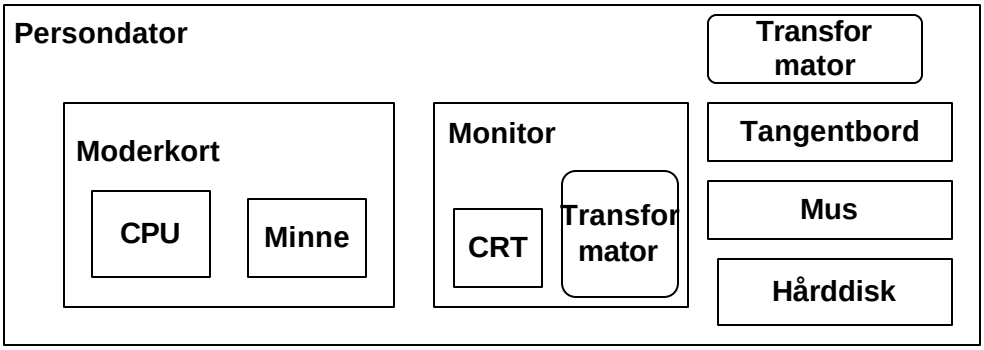
Multipelt arv

En klass kan ärva av flera andra klasser. I så fall gäller att ett objekt av subklasstypen är flera basklassobjekt samtidigt.



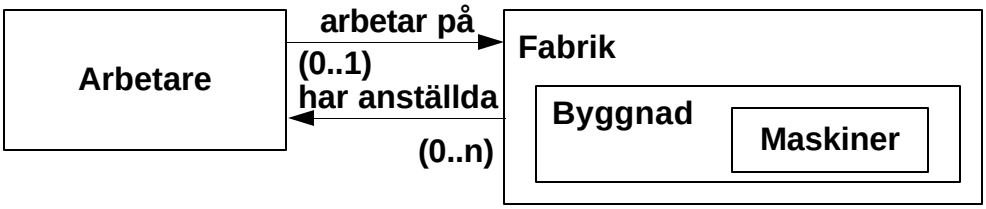
Aggregat

Aggregat kan gälla om relationen "ÄR EN DEL AV" är sann i avseende på ett objekt av en viss typ består av ett eller flera objekt av någon annan typ.



Association

Association erbjuder en något lösare koppling mellan objekt än arv och aggregat som alltid är mer tvingande. Exempel: Om vi skall relatera begreppen fabrik, arbetare, fabriksbyggnad och maskiner kan resultatet bli att fabriken och arbetarna har en ömsesidig association medan maskinerna är en del av fabriken (aggregat).



Användning

Det kan också vara så att objekt av en viss typ temporärt kräver att få använda andra objekt. Man brukar i sådana fall skilja på användning i gränssnitt (synligt) eller i implementation (ej synligt).

Polymorfism

Polymorfism är ett grekiskt ord som betyder "har många skepnader". Inom biologin används begreppet för att beskriva att individer inom samma art (samma typ) uppvisar olika utseende. Inom kemin används begreppet "polymorfa ämnen" om ämnen som uppvisar olika skepnader. Kol är t.ex. polymorft genom att det förekommer både som grafit och diamant. Inom objektorientering använder vi polymorfism när vi kombinerar arv och dynamisk bindning. Objekt som härstammar från samma hierarki kan själva definiera och ansvara för responsen för ett meddelande.

Polymorfism är ett centralt begrepp inom objektorientering och kanske tom. det mest betydelsefulla vid sidan av abstraktion.. Ett objektorienterat språk ska därför ha stöd för polymorfism. Om vi inte använder oss av polymorfism kommer vi att binda funktionsanrop vid kompilering/ länkning vilket betyder att vi får tidig bindning. Det betyder att vi vet exakt vilken

funktion som kommer att anropas.

Tidig bindning utan polymorfism är det vi använt och använder med icke objektorienterade språk.

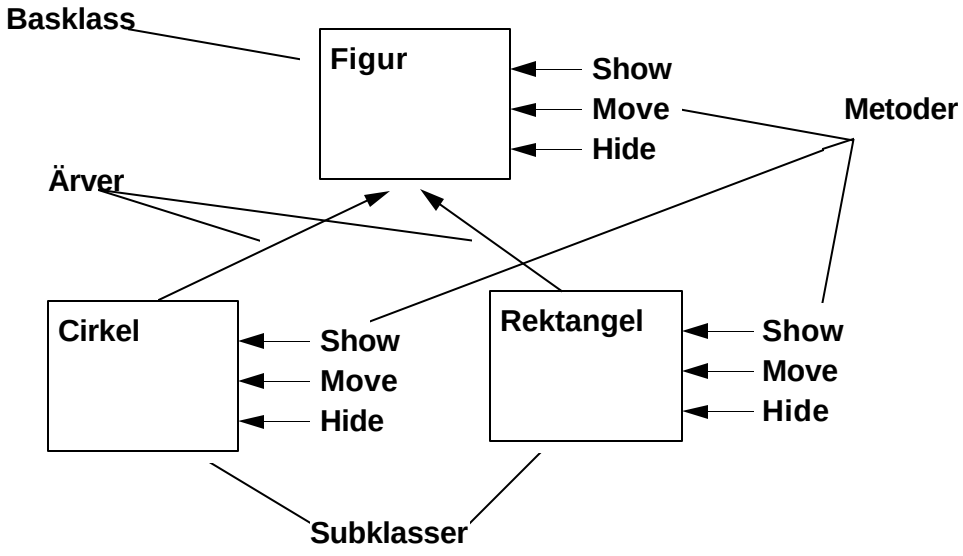
Om vi använder polymorfism och skickar ett meddelande till ett objekt i vårt program kommer ett funktionsanrop att innebära ett anrop av objektets egen funktion. Vilken funktion som anropas beror på vilket objekt som tar emot vårt meddelande och dess typ.

Eftersom objektets exakta typ ibland inte kan avgöras vid kompileringen kommer vi att få sen bindning. Vilken funktion som skall anropas avgörs först när vi exekverar programmet . Polymorfism bygger på att ett meddelande/en metod är giltig för flera typer av objekt som genom arv härstammar från samma hierarki och att objekten själva ansvarar för att definiera metoden.

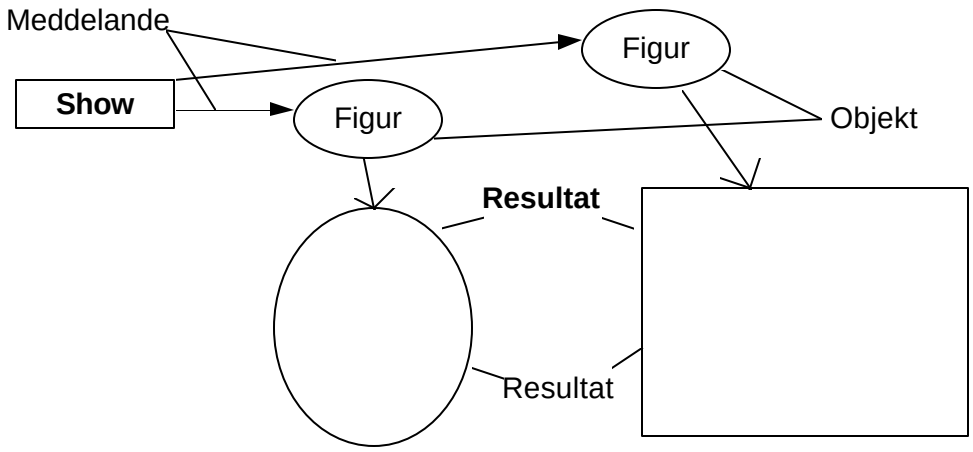
När vi skickar ett meddelande vill vi att det skall tas om hand av mottagaren och att rätt händelse skall inträffa. Utan polymorfism så är det sändaren (programmeraren/ användaren) som ansvarar för resultatet (att rätt funktion anropas). Om vi använder polymorfism så kommer vi att kunna lita på att det är mottagaren och mottagarens typ (objektets typ) som avgör vad det blir för resultat. Ett och samma meddelande får olika resultat beroende på vem (vilket objekt) vi skickar det till.

Förutsättningar för polymorfism..

En förutsättning för polymorfism är att det finns en hierarki med minst två klasser och att basklassen har en funktion som subklasserna kan omdefiniera.



Skickar vi meddelandet “Show” till ett Cirkelobjekt så blir det en cirkel och inte någon rektangel och omvänt.



Persistence

Vissa objekt är temporära och existerar under en kortare tid. Andra objekt är bestående (persistant=envis) och finns alltid eller under en längre tid. Om detta gäller i verkligheten måste vi också kunna använda denna egenskap i vårt system.