MENU

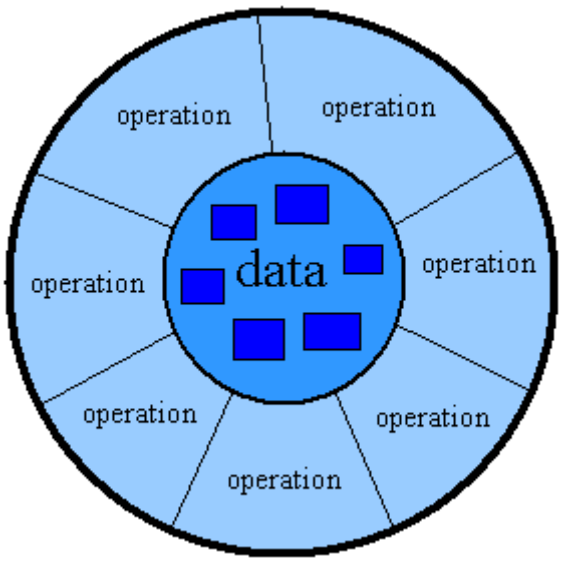
Introduktion

Hvad er objektorienteret programmering? Det er programmering med objekter!

I virkeligheden er det ikke mere kompliceret end det! Men historien er naturligvis meget længere. For hvad er objekter? Og hvad vil det sige at man programmerer med dem?

1. Objekter

En tegning siger mere end tusind ord, så lad os derfor se et objekt:



Figur 1:
Et objekt

Et objekt består af to ting:

1.1 Data

Datakernen

Datakernen ligger beskyttet inde midt i objektet. Udefra er der ikke mulighed for hverken at aflæse eller ændre data i objektet. Det er meget centrale i objekt-begrebet at der er denne beskyttelse. Den opleves dog ofte som en spændetrøje blandt begyndere i objektorienteret programmering. De mener at det ville være så meget nemmere hvis alle data var frit tilgængelige. Det har de delvis ret i. Det vil

MENU

være nemmere at lave "ad hoc"-programmering uden nævneværdige design-overvejelser, og man kan hurtigt få noget fra hånden. Problemet er at det går ud over næsten alt andet. Det er vanskeligere at finde fejl, for hvem var det nu lige der ændrede datakernen så den fik et forkert indhold? Hvad hvis man ønsker at lave ændringer der involverer bestemte data? Så skal man principielt til at lede hele kildeteksten igennem for at se hvordan disse data anvendes alle steder, så man ikke introducerer fejl, ved at der opstår modsætninger mellem de forskellige anvendelser. Ved at tillade fri adgang til datakernen mister man grebet - resultatet bliver uoverskueligt, og i længden håbløst at arbejde med. Det er netop dette der ligger til grund for begynderens oplevelse. Han skal ikke lave store programmer eller senere videreudvikle de programmer han laver.

Indkapsling

Dette, at datakernen ligger beskyttet inde i objektet, kaldes **indkapsling** og er et af de fundamentale begreber i objektorienteret programmering.

Slå skruer i med en skruetrækker

En anden grund til at man kan føle indkapsling som et irritationsmoment, er manglende indsigt i objektorientering. Hvis man tidligere har programmeret i procedurelle programmeringssprog (f.eks. Pascal eller C) vil man som udgangspunkt forsøge at "gøre det samme" som man altid har gjort, når man i stedet skal anvende et objektorienteret sprog (F.eks. Java, C++ eller C#). Det er simpelthen et spørgsmål om at man prøver at mase en form for programmering ned i en kasse, der er beregnet til en anden form for programmering. Hvis man har været vant til at bruge hammer og søm, men nu i stedet skal bruge en skruetrækker og skruer, er det klart at man efter at have slået på skruen med skaftet af skruetrækkeren, ikke synes at den nye teknologi er et fremskridt.

1.2 Operationer

Operationerne ligger beskyttende uden om datakernen. De beskytter og styrer den indirekte adgang til data. Som nævnt har ingen udenfor direkte adgang objektets data, men operationerne deles om datakernen og ved at anvende dem kan man udefra *indirekte* påvirke datakernen.

Skruer er dårlige søm

Når begyndere designer objekter, ser man ofte at de laver operationerne så de reelt giver direkte adgang til datakernen. De laver f.eks. operationer til at aflæse data og tilsvarende operationer til at ændre data. Dermed bruger de operationerne til at bryde indkapslingen og på den måde få skruerne banket ind i væggen, som de søm de burde være.

1.3 "Æble med kerne"-modellen

[Model af objekt - konkret billede - kan beskrives på andre måder - Jacobson's Object Advantage]

2. Objekt-systemer

Én svale gør ingen

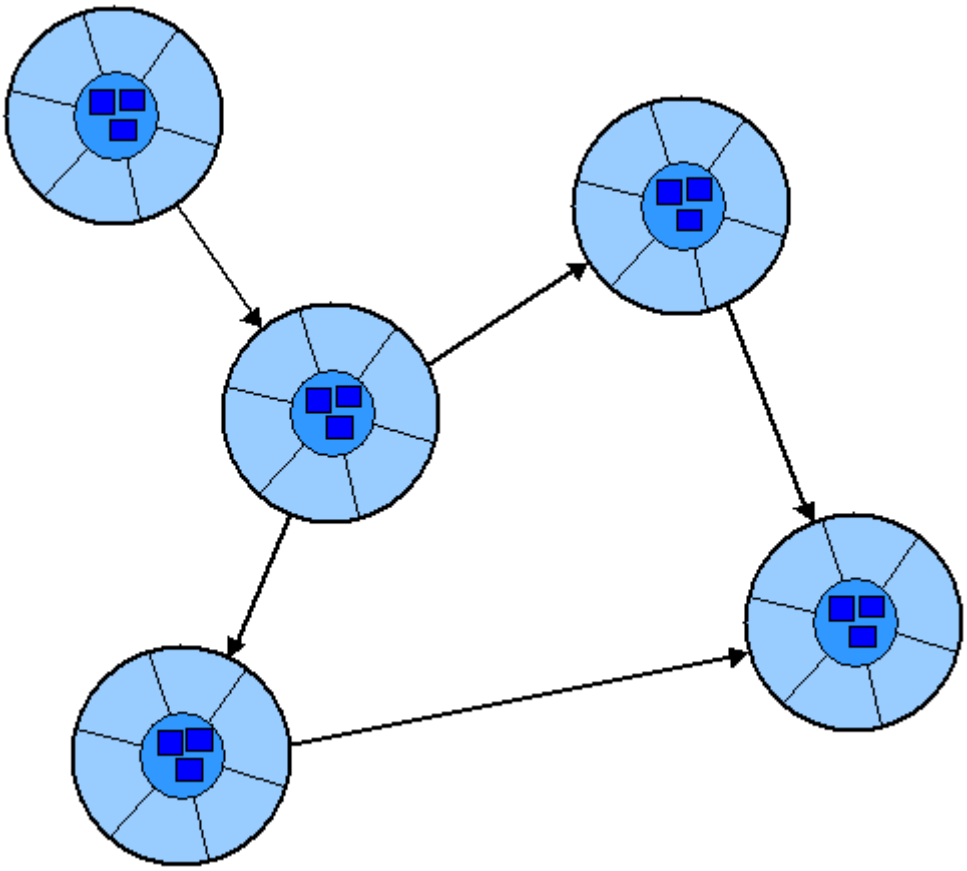
Et program med ét objekt er ikke meget bevendt. Én svale gør ingen sommer! Idéen i objektorienteret programmering er at have flere objekter der fungerer sammen i et **objektsystem**. Dette samarbejde er den mest centrale tanke bag

sommer

moderne objektorienteret programmering. I de senere år har "Design Pattern bølgen" skyllet gennem den objektorienterede verden, og den har bidraget til at formidle erfaring med at opbygge objektsystemer og realiserer deres samarbejdsmønstre. Bølgen har også bredt sig til andre emner, og i dag findes der patterns inden for mangt et område, også dem der intet har med programmering og systemudvikling at gøre.

Associering

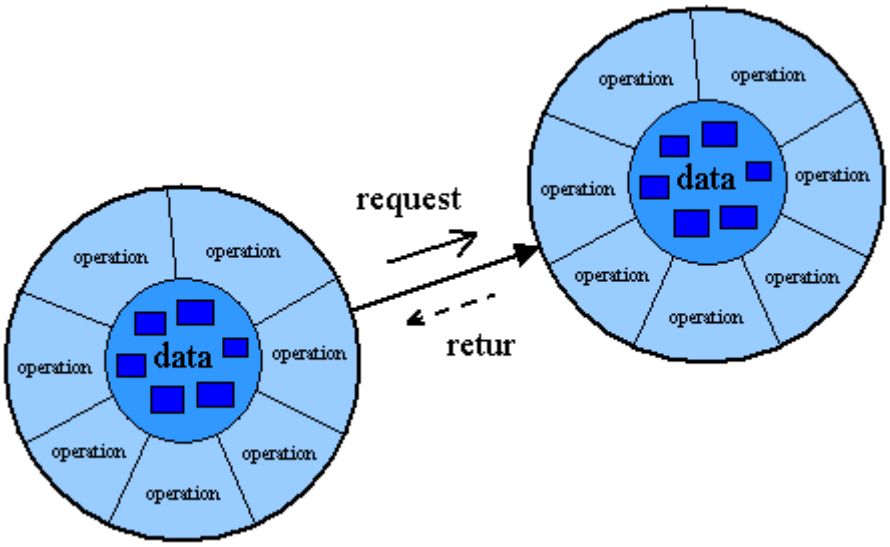
Måden objekter samarbejder på, kræver at de kender hinanden. Dette kendskab kaldes associering. Man er associeret et objekt, hvis man er kendt af det pågældende objekt. Objekter realiserer associering ved at have referencer til hinanden; hvor igennem de kan kommunikere. Vi vil illustrere dette med pile, der viser hvem der kender hvem.



Figur 2:
Objektsystem

Objekter kommunikerer ved, at et objekt sender en besked til et andet. Det modtagende objekt returnerer efterfølgende et svar. En besked afsendt fra et objekt kaldes en **request**. Når et objekt modtager en request reagerer den ved at gøre "et eller andet" og dernæst returnere "et eller andet":

Figur 3:
En request

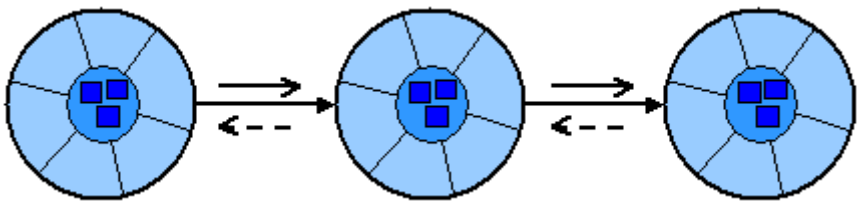


Indirekte læse eller ændre

Det "et eller andet" objektet gør, kan f.eks. bevirke at der sker ændringer i dets datakerne. På den måde har det objekt som sendte requesten indirekte fået adgang til at *ændre* datakernen. Det der returneres kan være et resultat, der baserer sig på hvilke data der var i kernen, og på den måde får det objekt der sendte requesten, indirekte adgang til at *læse* i datakernen. Det er væsentligt at bemærke, at det er objektet selv der har kontrol med: hvad der skal ændres, og hvad det vil returnere af oplysninger.

Én request går ud på at udføre én af objektets operationer. Udførelsen af en operation begrænser sig ikke kun til at arbejde med objektets datakerne. Det kan indbefatte at operationen selv sender requests til andre objekter, og på den måde samarbejder med andre om at besvare den request der blev modtaget. Det at sende en opgave helt eller delvist videre kaldes delegering. Delegering er det væsentligste samarbejdsmønster mellem objekter:

Figur 4:
Delegering



3. Skabeloner

Hvad gør man når man vil lave et objekt?

Instantiering

Man skal naturligvis *beskrive* objektet: Hvilke data og hvilke operatorer det skal have. En sådan beskrivelse vil vi i første omgang kalde en skabelon. Når man først har lavet en skabelon, kan man lave objekter ved at henvise til skabelon, idet man siger: "Jeg vil have sådan én".

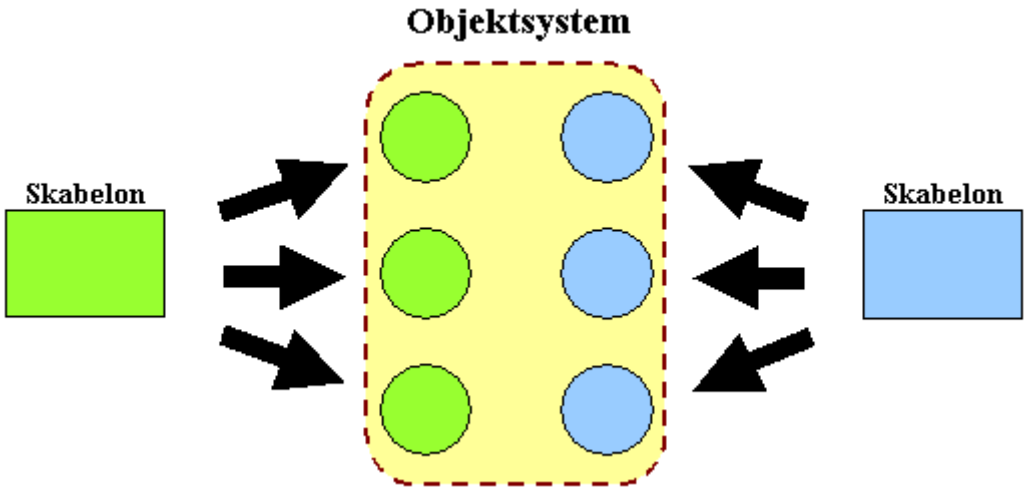
En af fordelene ved en skabelon er at man kan lave vilkårlig mange objekter ved at henvise til den. Når man laver objektsystemer vil man ofte bruge flere objekter af samme slags. Hvad skal man med ens objekter?

Uafhængige

Objekterne vil ikke være helt ens. Nok starter de med at være ens, men det bliver

af hinanden

sjældent ved ret længe. Ved forskellige requests vil datakernerne i de forskellige objekter løbende ændre sig og de vil leve hver deres liv. Det der vedbliver med at være ens er operationerne, der kan udføres ved requests, samt hvilke *slags* data der er i deres respektive datakerner.



Figur 5:
Instantiering

Her har vi to skabeloner: Grøn og Blå, og vi laver tre objekter ud fra hver af de to skabeloner. De seks objekter bruger vi til at lave et objektsystem.

Klasser og instanser

Skabeloner kalder man normalt **klasser**. Det at lave et objekt ud fra en klasse kalder man at **instantiere**. Af samme grund kalder man ofte objekter, der er lavet ved instantiering for **instanser**. Man siger ligeledes at et objekt er en **instans af** en klasse, hvis det er lavet ved at instantiere ud fra klassen.

I vores eksempel laves der altså tre instanser af hver af de to klasser: Grøn og Blå. De seks instanser bruges til at opbygge et objektsystem.

4. Nedarvning

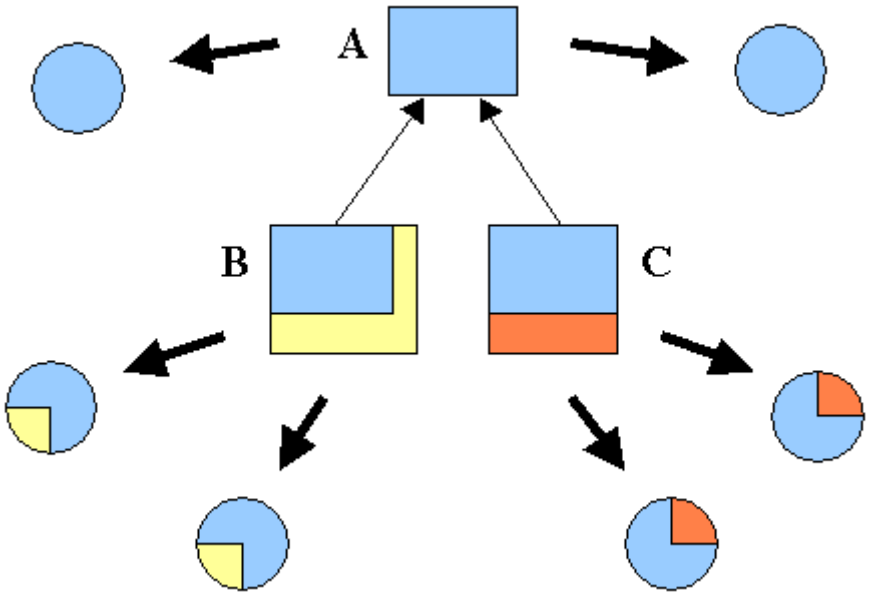
Vi kan nu lave lige så mange objekter af en bestemt slags som vi måtte ønske. Det eneste der kræves er at vi har en klasse der beskriver, hvordan de skal være opbygget. Hvis vi vil have en anden slags objekter skal vi blot lave en ny klasse.

***Lidt* anderledes**

Det med at lave en ny klasse kan være upraktisk, hvis det kun er en *lidt* anderledes objekter vi ønsker. I en sådan situation kunne man ønske at tage udgangspunkt i den klasse man allerede har, og blot angive de ting der skal være anderledes.

Det kan man ved hjælp af nedarvning, hvor en klasse laves med udgangspunkt i en anden. Man siger, at man nedarver fra en klasse til en anden. Den klasse man arver fra, er den man allerede har, mens den man arver til, er den lidt anderledes klasse man gerne vil lave.

Figur 6:
Nedarvning



Super- og sub-klasser

I figuren er nedarvning vist med en pil fra den klasse der ønsker at arve til den klasse der arves fra. Man kalder den klasse der arves fra: en **super-klasse**. I vores eksempel er klassen **A** en super-klasse. En klasse der arver fra en anden klasse kaldes en **sub-klasse**. I vores eksempel har vi to sub-klasser: **B** og **C**, der begge arver fra super-klassen **A**.

Som man ser i figuren, tilføjer vi nye egenskaber i de to sub-klasser. I sub-klassen **B** er disse illustreret med gult, og i subklasse **C** med rødt. Disse egenskaber kan være såvel operationer som data. I instanserne ser man, at det blå stadig er der, men at der nu også er de nye egenskaber der supplerer de nedarvede.

Variation

Resultatet bliver, at vi har tre slags objekter, instanser af tre forskellige klasser. Klasser, der har en indbyrdes sammenhæng, i kraft af nedarvningen. Dette er et eksempel på det man kalder variation. Vi har lavet variation vha. nedarvning, idet vi nu kan lave objekter der er lidt anderledes end dem vi ellers kunne lave.

Men hvad med Java - lad os få noget objektorienteret kode på banen!

Det får vi i næste kapitel.

Repetitionsspørgsmål

1 Hvad er datakernen i et objekt.

2 Hvad er indkapsling?

3 Hvad er operationernes rolle i et objekt?

4 Hvordan kan man bryde indkapsling?

5 Hvad er et objektsystem?

Hvad er associering?

6

7 Hvordan forløber håndteringen af en request?

8 Hvad er delegering?

9 Hvad er en klasse?

10 Hvad vil det sige, at instantiere?

11 Hvad er en instans?

12 Hvad er nedarvning?

13 Hvad er henholdsvis en super- og en sub-klasse?

14 Hvad er variation?

Svar på repetitionsspørgsmål

1 Datakernen indeholder objektets data.

2 Indkapsling er at datakernen ikke kan læses/ændres udefra.

3 Operationerne gør det muligt indirekte at arbejde med objektets data.

4 Man kan bryde indkapslingen ved at lave metoder til direkte at læse/ændre datakernens indhold.

5 Et objektsystem, er en gruppe af objekter der samarbejder.

6 Associering er når et objekt kender et andet objekt.

7 Når et objekt modtager en request "gør den et eller andet". Efter at have gjorte dette eller hint, returnerer den "et eller andet" til det objekt som sendte requesten.

8 Delegering er når et objekt får hjælp af et andet objekt til at besvare en request. På den måde kommer et andet objekt til helt eller delvist at løse den opgave der skal løses før der kan returneres.

9 En klasse er en skabelon, der beskriver hvordan et objekt skal opbygges. Man kan derfor lave objekter ud fra en klasse.

10 At instantiere, er at lave objekter ud fra en klasse.

11 En instans er et objekt der er lavet ud fra en klasse.

12 Nedarvning er når man med udgangspunkt i en klasse laver en anden klasse. Instanser af den klasse der nedarves til, vil have de samme egenskaber som

instanser af den klasse der arves fra, *men* man vil have suppleret med *yderligere* egenskaber.

13 En super-klasse er en klasse der nedarves *fra*, mens en sub-klasse er en klasse der nedarves *til*.

14 Variation er når man laver noget der er lidt anderledes end det man allerede har. F.eks. kan man lave variation vha. nedarvning.