3 Klasser (provavsnitt)

Att hantera många objekt

Fundera ett ögonblick över hur många objekt vi beskriver i en objektmodell:

- Små objektmodeller (t ex ett sällskapsspel) – kanske några hundra objekt
- Stora objektmodeller (t ex bank) – allt från tusentals till miljoner objekt

Hur klarar vi av att hantera så många objekt?

Jämför med hur vi människor hanterar komplexitet i vardagen:

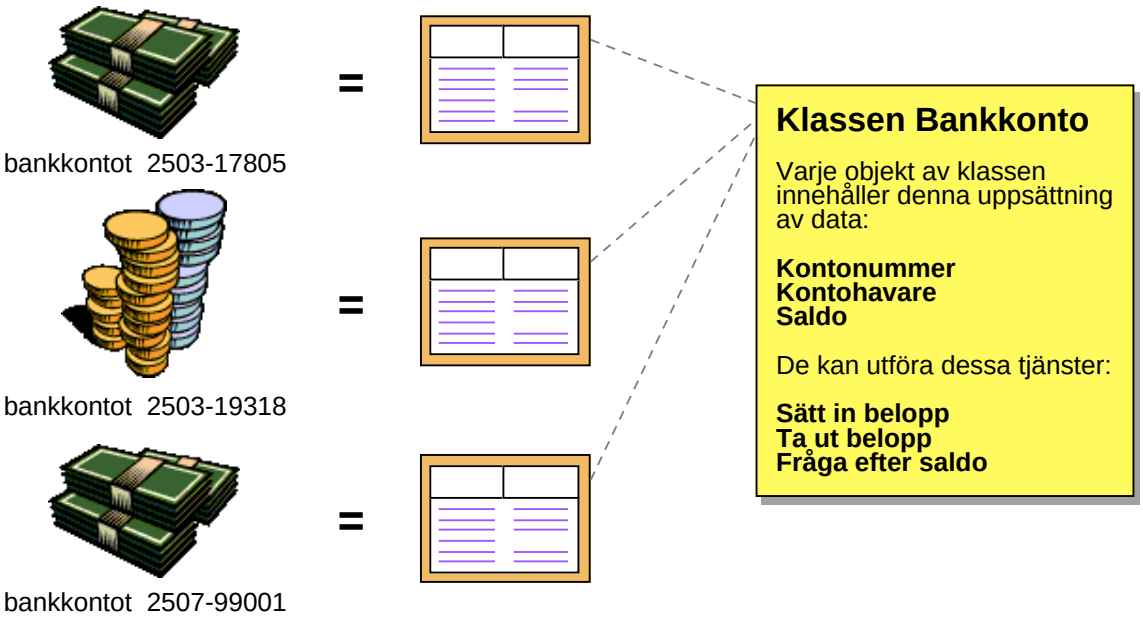
- Vi skiljer på helheter och dess beståndsdelar (jämför med objektindelning)
- Vi filtrerar bort oväsentlig information (eliminera onödiga objekt ur modellen)
- Vi delar in saker och ting i *kategorier* och lär oss fakta om dem i stället

En liten objektmodell för ett väldigt begränsat verksamhetsområde som t ex ett sällskapsspel kanske endast innehåller några hundra objekt. Stora objektmodeller kan beskriva system med allt från tusentals till miljontals objekt. Hur kan vi någonsin beskriva och administrera ett system med så många objekt?

Svaret ligger närmare oss än vi tror. De som forskar i hur den mänskliga hjärnan fungerar kan förklara att vi alla använder oss av ett antal kraftfulla tekniker för att göra omvärlden hanterbar. Utan dem skulle vi vara nästan hjälplösa i vårt vanliga liv. Tänk ett tag: De flesta klarar utan problem av att ta sig från sängen i sitt hem till en arbetsplats varje dag. Ändå kräver det att vi lär oss att hantera tusentals saker – objekt om vi så vill – korrekt varje dag. Några objekt använder vi för att göra oss rena och presentabla på morgonen, andra sätter vi på oss, vissa objekt äter vi till frukost, osv. Under resvägen till jobbet använder vi vissa objekt – cyklar, bilar, bussar, pengar, biljetter, automater osv – medan vi undviker andra, som t ex lyktstolpar. Som människor måste vi hålla reda på information om hur tusentals av dessa objekt ska hanteras – ändå har de flesta inga problem att göra detta rätt gång efter gång. Hur går det här till?

Två tekniker som den mänskliga hjärnan använder för att förstå världen och reducera komplexitet är att skilja på helheter och dess beståndsdelar, samt att ta bort oväsentlig information. Detta motsvarar direkt hur vi ser på skillnaden mellan objekt och dess beståndsdelar, samt hur vi eliminerar onödiga objekt från objektmodellen. En tredje, mycket kraftfull teknik är att dela in saker och ting i *kategorier* och lära sig information om dessa kategorier i stället. Dvs vi lär oss att lyktstolpar är ting som alltid står stilla (utom i skaderapporter till försäkringsbolag) och som vi ska undvika att köra in i, i stället för att behöva lära oss detta om varje enskild lyktstolpe för sig. Detta är en enorm förenkling. Om exemplet låter löjligt, är det för att det är så naturligt för oss vuxna människor att tänka på det här sättet, att vi måste anstränga oss för att inte göra det. Detta är dock inte medfött – det är något vi lär oss i unga år. Barn lär sig t ex identifiera djur som exemplar ur kategorier som exempelvis katter och hundar, och kan plötsligt placera in nya husdjur de aldrig tidigare träffat i de kategorierna och börja dra slutsatser om dem.

Klasser



- Alla objekt av en klass delar på samma beskrivning
- Enorm förenkling i arbetsmängd och systemstorlek

Det är förmågan till kategoribildning som ger oss lösningen på hur vi ska hantera mångfalden av objekt i våra system.

Vi bildar *klasser* av de objekt som kan beskrivas på samma sätt. Dessa beskrivningar behöver då bara skapas och underhållas på ett ställe per klass, i stället för att ha en komplett beskrivning av varje objekt. Förenklingen blir enorm: färre beskrivningar är inte bara lättare att ta fram och underhålla; de motsvarar också betydligt mindre källkod och en mycket lägre minnesförbrukning i ett objektorienterat system – alla fördelar på en gång, med andra ord. Och det vi gör är något som alla människor (utom möjligen programmerare som levt för länge med procedurorientering) gör naturligt: bildar kategorier som motsvarar klasser av objekt.

Krav för att bilda en klass

För att en grupp av objekt ska kunna tillhöra samma klass måste de ha:

- samma utseende på det privata minnet
  - ➜ samma uppsättning av privata variabler
  - ➜ samma variabelnamn och typ för varje privat variabel
- samma uppsättning av tjänster
  - ➜ samma tjänstenamn, antal och typ av argument för varje tjänst
- samma implementation av metoder för dessa tjänster

Det enda som skiljer två objekt av en klass är innehållet i deras privata data.

Vi bildar alltså klasser av de objekt som kan beskrivas på samma sätt. Nu går det inte vara så släpphänt att det räcker med en så pass otydlig formulering. Objektorientering definierar tre strikta krav för att en grupp av objekt ska kunna höra till samma klass. De måste ha:

- samma utseende på det privata minnet – samma antal, typ och variabelnamn för privata data
- samma uppsättning av tjänster – samma tjänstenamn, samma antal och typ av argument
- samma implementation av metoder för dessa tjänster

Med andra ord, det enda som skiljer mellan två objekt av samma klass är innehållet (värdena) i deras privata data, så alla skillnader i hur de beter sig måste härröra från deras olika datainnehåll. Allt annat är identiskt lika för samtliga objekt inom klassen, och det är det som gör att vi bara behöver lämna en beskrivning per klass.

Betrakta ett antal bankkonton i vårt exempel. Vad vi säger betyder alltså att om det privata minnet för ett bankkonto består av kontonummer, kontohavare och saldo (tre variabler, deklarerade med vissa namn), och uppsättningen av tjänster omfattar att kunna

- sätta in ett belopp
- ta ut ett belopp
- fråga efter saldo

och vi dessutom har en viss implementation av metoder för dessa tjänster, så måste alla objekt som står för bankkonton beskrivas på exakt detta sätt, om de ska få tillhöra samma klass. Däremot kan varje objekt av klassen ha sina egna värden för variablerna i det privata minnet.

Får två objekt vara helt identiska?

Kan två objekt av samma klass ha exakt samma innehåll i det privata minnet?

- Objektorientering har inget generellt krav på unikt datainnehåll
- Det finns inga generella krav på unika "nycklar" för varje objekt
- Verksamhetsområdet kan dock ha andra krav

I ett objektorienterat system går det att skilja på objekt oavsett deras innehåll.

Även om det brukar vara så att två objekt av samma klass har olika värden i sina privata data, är det inte förbjudet att ha två objekt med helt identiskt innehåll. Detta är helt i sin ordning, men det är något som brukar bekymra de som kommer från datamodellering eller informationsmodellering till objektorienterad systemutveckling. Det finns alltså inget krav på någon unikt identifierande nyckel i varje objekt. Runtimesystem för objektorienterade programspråk har ändå alltid möjlighet att se skillnad på två objekt, och det brukar gå att i språket ta reda på om två objektreferenser går till samma fysiska objekt eller inte.

Däremot kan det i verksamheten finnas krav på att vissa data unikt ska identifiera ett objekt. I ett bank-system skulle det exempelvis krävas att två bankkonton aldrig får ha samma kontonummer, och därmed skulle aldrig innehållet i två bankkontoobjekt kunna vara identiskt.

Måste vi använda klasser i objektorientering?

Nej, det går att arbeta objektorienterat utan att utnyttja klasser:

- Det har förekommit konkreta objektorienterade språk utan klassbegrepp
- De blir extremt dynamiska till sin natur

Men det är som regel oerhört praktiskt att utnyttja klasser:

- Det blir krävande att hantera många objekt utan ett klassbegrepp
- Underhållet kan bli mycket tidskrävande
- Avsaknad av klasser kan innebära avsaknad av en struktur för systemet

I de flesta objektorienterade programspråk är klassbegreppet obligatoriskt:

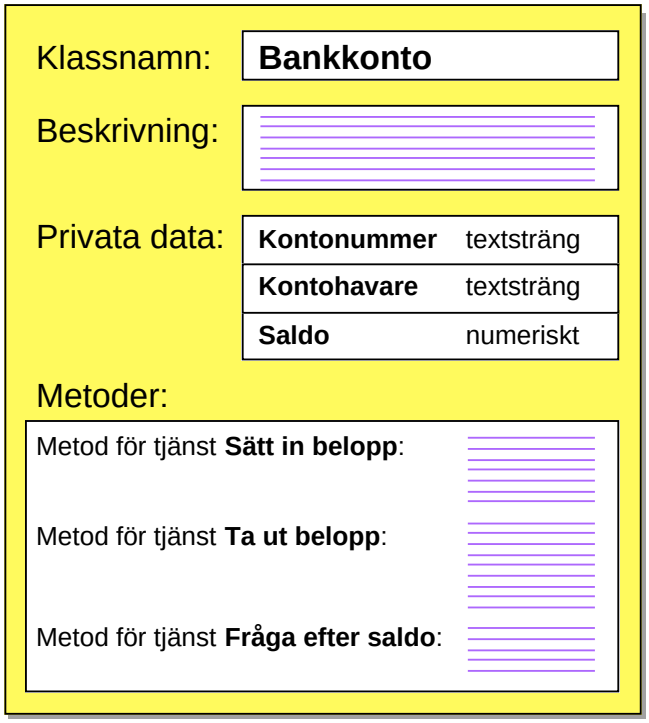
- Varje objekt måste då höra till en klass
- Vi måste beskriva en klass, även om det gäller ett enda objekt

Är det nödvändigt att använda klasser i ett objektorienterat system? Nej, men det är som regel oerhört praktiskt. I små system med ett måttligt antal objekt är det överkomligt att hantera unika objekt med varsin egen beskrivning. Apple Computers programspråk **HyperTalk**, som lanserades tillsammans med det innovativa programmet **HyperCard**, är kanske det mest kända exemplet på att det är möjligt.

HyperCard är ett extremt dynamiskt system där nya objekt skapas genom att klona befintliga objekt och därefter börja modifiera dem: lägga till eller ta bort datainnehåll, och lägga till, ta bort eller byta ut metoder – inte bara när objektet är nyskapat, utan när som helst i objektets liv. Det kan låta som en objektorienterares drömvärld, men drömmen blir ofta till en mardröm när den ska underhållas. System som HyperCard/HyperTalk erbjuder helt enkelt för lite stadga och struktur, och klasser tillför det som saknas för att göra systemen lättare att hantera.

I de objektorienterade programspråk som valt att stödja ett klassbegrepp (och det omfattar de allra flesta, inklusive Smalltalk, C++ och Java) är klassbegreppet inte enbart där som ett frivilligt val – det är obligatoriskt. Objekt måste beskrivas av en klass, t o m om vi bara behöver ett enda. Det kan tyckas vara en onödig omväg att gå i just det fallet, men vi får igen det i enkelheten att veta att alla objekt hör till en klass. Vi har inga "ensamvargar" som beskrivs på ett helt annat sätt.

En klassbeskrivning



För varje klass utformar vi en *klassbeskrivning*, som bl a består av klassnamn, deklaration av vilka privata data som ingår i objekt av klassen, samt metoder för samtliga tjänster hos klassens objekt.

En klass identifieras av ett *klassnamn*, som inte har något med de enskilda objekten att göra. Klassnamnet existerar i första hand i källkoden för det objektorienterade programspråket, men det och andra data om klassen kan även finnas tillgängliga i systemet under exekvering. En vanlig konvention för klassnamn är att döpa klassen efter vad ett objekt av klassen är, i singularform. En klass som beskriver olika objekt som står för bankkonton skulle därför kunna kallas `Bankkonto`. Vi kommer att använda den vanligt förekommande konventionen att skriva klassnamn med stor begynnelsebokstav, för att göra det tydligt när vi talar om en klass.

För en klass är det också viktigt att vi inte bara ger klassen ett klassnamn, utan även en ordentlig beskrivning av vad objekt av klassen som helhet står för, och vad de olika tjänsterna uträttar för arbete. Många misstag som görs i objektorienterad programmering bottnar i tron att detta inte alltid är nödvändigt.

Hur ser objekten ur i ett exekverande system?

bankkontot 2503-17805

=

Bankkonto
2503-17805
Anna Ceric
11745,80

bankkontot 2503-19318

=

Bankkonto
2503-19318
Bertil Ek
7309,15

bankkontot 2507-99001

=

Bankkonto
2507-99001
Eva Barany
45088,35

Klassen Bankkonto

Varje objekt av klassen innehåller denna uppsättning av data:

Kontonummer
Kontohavare
Saldo

De kan utföra dessa tjänster:

Sätt in belopp
Ta ut belopp
Fråga efter saldo

- I verkligheten lagras endast värdena för de privata variablerna i objekten
- Dessutom måste varje objekt känna till sin klasstillhörighet

Hur ser objekten i ett exekverande system egentligen ut i ett läge då alla objekt hör till någon klass?

Eftersom hela beskrivningen av de objekt som hör till samma klass nu samlats till ett ställe – klassen – blir enbart datainnehållet kvar i de enskilda objekten. För objekt av klassen `Bankkonto` innebär det att varje objekt innehåller en uppsättning konkreta värden för kontonummer, kontohavare och saldo som motsvarar ett specifikt bankkonto i verkligheten. Såväl deklarationen av de privata variablerna som all exekverbar kod finns uteslutande i klassen.

Om detta faktiskt ska fungera krävs dock en sak till. Varje objekt måste känna till sin klasstillhörighet. Detta är nödvändigt eftersom objektets innehåll inte går att tolka om vi inte vet hur det privata minnet är deklarerat. Den informationen finns nu enbart i klassen. Där finns också de metoder som ska arbeta med objektets data. Alltså måste det i varje objekt finnas något som gör att vi kan avgöra objektets klasstillhörighet.

Hur ett objekt håller reda på sin klasstillhörighet är en implementationsangelägenhet i runtimesystemet för ett objektorienterat programspråk. Vi skulle t ex kunna lagra klassens namn i varje objekt, men det skulle kräva en hel del extra utrymme, och det skulle vara kännbart för små objekt. Vad som ofta förekommer i stället är något i stil med ett heltal som fungerar som index i en tabell över samtliga klasser i systemet.