Föreläsning 17:
Mer polymorfism, och `static`

Christian.Soderberg@cs.lth.se

22 februari 2012

Förra gången

- Polymorfism
- Olika slags kvalifikation
- Statisk typ-kontroll
- Bindning av metoder
- Klassen `Object`

Exempel

- Vad betyder egentligen uttrycket:

```
y = Math.sqrt(x);
```

- Vad är `Math.PI`?
- Vad betyder `System.out.print`?

Klassoperationer och klassattribut

- Vanliga attribut och operationer hör till det objekt som klassen definierar (`this`).
- Attribut och operationer som markeras med det reserverade ordet `static` hör istället till *klassen själv*.
- `static`-deklarerade attribut och operationer kallas *klassattribut* och *klassoperationer*, och de kan användas utan att vi skapar ett objekt av klassen.

Klassoperationer

Vi kan använda klassattribut och klassoperationer för att:

- lagra värden som alla objekt av en given klass kan dela
- göra saker som inte har med ett enskilt objekt att göra
- samla ihop relaterade operationer och värden som vi vill kunna anropa utan att behöva skapa nya objekt (exempelvis `Math`-klassen)
- skapa 'pseudo-konstruktorer', dvs metoder som returnerar nya objekt av klassen

Klassattribut och klassoperationer

- I en klassoperation har vi inget `this`, och kan därmed inte använda de vanliga attributen i klassen. De enda attribut och operationer vi kan använda är klassattributen och klassoperationerna.
- Vanliga operationer kan använda klassoperationer och klassattribut -- om de ändrar på ett klassattribut så kommer det nya värdet på klassattributet att gälla hela klassen.
- Huvudprogrammet är en klassoperation på huvudprogramsklassen.

Sortering

- Vi kan sortera en vektor på *många* olika sätt.
- De två enklaste är:
  - *Urvalssortering*
  - *Insättningssortering*

Urvalssortering

Sortera vektorn a med n element:

U1. Låt p vara 0.

U2. Låt i vara index för det minsta värdet i intervallet $a_p \ldots a_{n-1}$.

U3. Byt plats på a_p och a_i.

U4. Öka p med 1, hoppa till U2 om vi inte är klara.

Insättningssortering

Sortera vektorn a med n element:

I1. Låt p vara 1.

I2. Sortera in a_p bland $a_0 \ldots a_{p-1}$

I3. Öka p med 1, hoppa till I2 om vi inte är klara.

Tidsåtgång för sortering

- Urvalssortering:
  - Alltid samma arbete.
  - $T(n) = n + (n-1) + (n-2) + \ldots 1 = \frac{n(n+1)}{2} = O(n^2)$
- Insättningssortering:
  - I värsta fall: $T(n) = 1 + 2 + 3 + \ldots (n-1) + n = \frac{n(n+1)}{2} = O(n^2)$
  - I genomsnitt ungefär hälften så lång tid.
- Med dessa båda operationer kan vi sortera 1 000-tals tal, kanske 10 000-tals värden -- men inte gärna mer.
- Det finns *betydligt* snabbare metoder ($O(n \log n)$).

Sortering av referenser till objekt

- Vi kan sortera referenser till objekt på precis samma sätt som vi sorterar primitiva värden, men jämförelserna måste göras på något annat sätt.
- Ofta kan vi använda värdet på en operation som kriterium.
- Ibland kan vi anropa `compareTo`, som vi gör på strängar.
- Det finns i Javas standardklasser operationer som väldigt snabbt sorterar både vektorer och listor -- nästa vecka skall vi se hur man använder dem.