Litt om Javas håndtering av tall
MAT-INF 1100 høsten 2005

9. september 2005

En viktig del av den første obligatoriske oppgaven er å få erfaring med hvordan Java håndterer tall. Til å begynne med kan dette virke nokså vilkårlig, men det ligger klare regler bak. Her skal vi oppsummere noen av disse. Mesteparten av dette stoffet vil du finne igjen ulike steder i læreboka i INF 1000 (eller en annen bok om Java).

1 Deklarasjon og tilordning av variable

For å forstå hvordan beregninger utføres i Java er det viktig å ha litt kunnskap om hva en deklarasjon som

```
double x;
```

innebærer. Deklarasjonen `double x` gir beskjed om at variabelen `x` skal være av typen `double`. Dette betyr at `x` assosieres med en lagerplass (boks) i maskinen som inneholder 64 binære sifre. Det betyr også at hver gang du forsøker å legge et tall inn i `x` vil det bli sjekket om tallet er av en type som kan passe inn i en `double`-variabel. Det å legge en verdi inn i `x` kan du for eksempel gjøre ved tilordninger som

```
x = 1.0 + 3.0;
x = a;
```

I det første tilfellet har vi et eksplisitt talluttrykk på høyre side som regnes ut til et tall. I det andre tilfellet vil verdien av variabelen `a` bli hentet fram, og det resulterende tallet, som vil bli forsøkt lagt inn i `x`, vil ha samme type som `a`.

Det ovenstående gjelder i og for seg for alle typer variable, men reglene for hva som kan legges inn i ulike variabeltyper varierer naturlig nok. De viktigste reglene er oppsummert under.

Lagring i `double`-variable. Det som kan legges inn i en `double`-variabel er

- Tall av typen `double`.
- Tall av typen `float`. Et slikt tall vil bli gjort om til `double` ved å legge til ekstra siffer. Disse vil vanligvis ikke være korrekte.
- Heltall av typen `long` og `int` (også tall av typen `short`, `byte` og `char`, men disse skal vi overse her). Et slikt tall vil bli gjort om til `double`. Dersom antall siffer i tallet er større enn det som kan lagres i en `double`-variabel vil de minst viktige sifrene kunne gå tapt.

Lagring i `float`-variable. Du kan legge tall av typen `float`, `long` og `int`, inn i en `float`-variabel. Analogt til det som skjer ved tilordning til `double`-variable vil sifre kunne gå tapt hvis du legger et stort heltall inn i en `float`. Legg merke til at du får feil ved kompilering om du forsøker å legge et `double`-tall inn i en `float`-variabel.

Lagring i `long`-variable. I `long`-variable kan du legge tall av typen `long` og `int`. Dette vil alltid gå bra i og med at det alltid er plass til en `int` i en `long`. Forsøker du å legge et tall av typen `double` eller `long` inn i en `long`-variabel får du feilmelding.

Lagring i `int`-variable. I `int`-variable kan du legge tall av typen `int`. Forsøk på å legge et tall av type `double`, `float` eller `long` inn i en `int`-variabel gir feil ved kompilering.

Det ovenstående kan oppsummeres ved å ordne variabeltypene i en rangert liste:

1. `double` (64 bits flyttall)
2. `float` (32 bits flyttall)
3. `long` (64 bits heltall)
4. `int` (32 bits heltall)

I denne rangeringen står flyttallsvariable altså over heltallsvariable og innen samme gruppe står 64 bits variable over 32 bits variable. Reglene over kan oppsummeres ved følgende: *En variabel av type nr. i kan motta et tall av type i eller større, men ikke et tall av en lavere type enn i.*

2 Utregning av talluttrykk

Når vi nå vet hva som kan legges inn i ulike variabeltyper må vi vite hvordan typiske talluttrykk som forekommer på høyre side av en tilordning får en type

slik at vi kan avgjøre om det vi til slutt forsøker å legge inn i variabelen på venstre side blir det vi ønsker. La oss se på eksempelet

```
float x;
x = 1 + 2.0;
```

Fra det ovenstående vet vi at vi vil få protester fra kompilatoren om ikke høyresiden er av type `float`, `long` eller `int`. Den grunnleggende regelen er som følger: *Alle tall i et uttrykk blir gjort om til den typen i uttrykket som står høyest i typerangeringen over.*

For at denne regelen skal gi mening må vi kunne si hvilken type et tall som inngår i et uttrykk har. Her gjelder følgende regler.

1. Et heltall, for eksempel -10, tolkes som en `int`.
2. Et desimaltall, for eksempel 21.3, tolkes som en `double`.
3. Et heltall etterfulgt av L, for eksempel 45L, tolkes som en `long`.
4. Et tall etterfulgt av F, for eksempel 101F, tolkes som en `float`.
5. Et tall etterfulgt av D, for eksempel 101D, tolkes som en `double`.

På bakgrunn av dette er det lett å se hva som skjer med kodebiten vi hadde over

```
float x;
x = 1 + 2.0;
```

Tallet 1 vil bli tolket som en `int` mens 2.0 tolkes som en `double`. Den høyest rangerte typen er `double` så 1 blir gjort om fra `int` til `double` før addisjonen utføres. Resultatet blir en `double` med verdi 3.0. Men Java nekter å legge et tall på `double`-format inn i en `float`-variabel som `x`, så kodebiten vil gi en kompileringsfeil.

Dersom vi holder oss til regelen om at vi bruker `int`-variable til heltall og `double`-variable til desimaltall vil det meste gå greit. Men det er en liten detalj som ofte kan gi frustrasjoner. Resultatet av kodebiten

```
double x;
x = 1/2;
```

er at `x` får verdien 0, noe som neppe vil være tilsiktet. Årsaken er at tallene på høyre side begge er heltall og dermed av type `int`. Divisjonen vil derfor foregå med to tall av type `int`. For slike tall er divisjonsoperatoren definert

som heltallsdivisjon, altså divisjon der resten hives. Svaret blir derfor 0 som så legges inn i `x`. Ut fra hva vi har sett over kan vi unngå dette ved å passe på at et av de to tallene 1 og 2 tolkes som en `double`. Det kan vi for eksempel få til ved å skrive `x = 1.0/2` eller `x = 1D/2`.

Problemet med divisjon er at Java bruker samme notasjon for to ulike matematiske operasjoner, nemlig vanlig divisjon og divisjon uten rest (heltallsdivisjon). Operatorene `+`, `-` og `*` gir ikke de samme problemene siden det ikke finnes flere versjoner av de tilsvarende matematiske operasjonene addisjon, subtraksjon og multiplikasjon.

3 Tvungen typekonvertering

Over har vi sett at Java er i stand til å konvertere tall fra en type til en annen ettersom behovet melder seg. Så langt har dette stort sett skjedd på Javas initiativ, men vi kan framtvinge en slik konvertering ved såkalt 'casting' (typekonvertering). Vi kan for eksempel si

```
double d = 1.0/3;
float f = (float) d;
```

Uten den tvungne konverteringen i linje to ville vi fått en feilmelding ved konvertering. Med samme teknikk kan vi konvertere mellom andre typer etter behov.

4 Bruk av matematiske funksjoner.

Når vi programmerer matematiske beregninger har vi ofte bruk for de vanlige, matematiske funksjonene. I Java kan vi for eksempel regne ut $\sqrt{2}$ ved å si `Math.sqrt(2.0)`. Resultatet vil være en tilnærming til $\sqrt{2}$ i form av et flyttall av typen `double`. Mange andre funksjoner er tilgjengelige på samme måte; for en fullstendig liste kan du slå opp i en Java-bok eller ta en titt på nettsiden

`http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Math.html`

(det ligger en peker til denne fra Java-siden på hjemmesiden til MAT-INF 1100). Legg merke til at på nettsiden er mange av funksjonene listet opp flere ganger, for eksempel absoluttverdifunksjonen `abs`. Hvis du ser litt nøyere etter så er forskjellen på de ulike variantene at de tar argumenter av ulike typer, i dette tilfellet alle fire typene `int`, `long`, `float` og `long`. For en bruker av funksjonen virker det da som om `math.abs`-funksjonen kan akseptere argument av alle typene, men i virkeligheten er dette fire forskjellige funksjoner som skiller seg ved argumentets type. Legg merke til at typen til

resultatet er listet opp i venstre kolonne i tabellen på den nevnte websida (overse ordet `static` i denne sammenhengen).

Noe som mangler i Java er en eksponensialfunksjon med heltallige argumenter. Funksjonen `math.pow(double a, double b)` beregner en `double`-tilnærming til a^b for `double`-variable `a` og `b`, men det fins altså ingen tilsvarende funksjon når `a` og `b` er `int`- eller `long`-variable. Skal du regne ut 3^8 kan du godt si

```
long i = (long) Math.pow(3,8);
```

som vil gi deg det ønskede resultatet `i = 6561`. Men prøver du deg med

```
long i = (long) Math.pow(3,38);
```

gir Java deg svaret `i = 1350851717672992000` mens det riktige svaret er $3^{38} = 1350851717672992089$. Vi ser at de siste sifrene er feil og dette kommer av at `double`-typen ikke opererer med tilstrekkelig mange siffer til å representere 3^{38} korrekt. Dette problemet oppstår kun når vi nærmer oss den øvre grensen for hva som kan representeres i en `long`, men er viktig å være klar over. `Math.pow` vil alltid gi riktig svar for `int`-variable siden disse ikke kan bli større enn omtrent 10^9 og `Math.pow(a, b)` vil regne ut a^b med omtrent 16 riktige sifre.