Föreläsning 5:
Om strängar, loopar och annat.

Christian.Soderberg@cs.lth.se

5 februari 2013

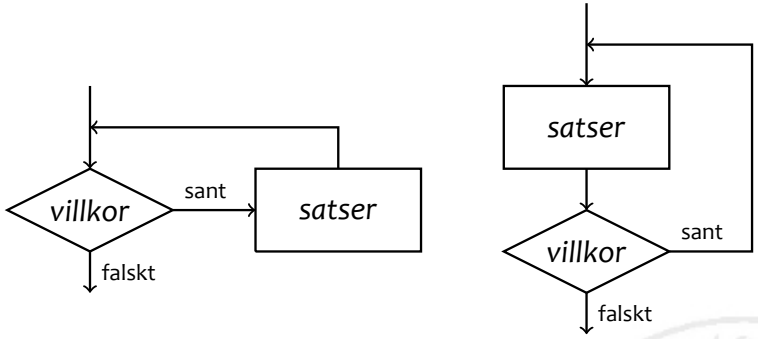
Vanliga operationer på och med strängar

- `String`-metoder:
  - `int length()`: ger strängens längd
  - `char charAt(int n)`: ger tecknet på plats *n* (som numreras från 0 och uppåt)
  - `int indexOf(char c)`: ger första platsen (0 . . .) för tecknet `c`, om det förekommer i strängen, annars −1.
- I klassen `Keyboard` finns dessutom metoden
  - `String nextLine()`: läser in en hel rad

Klassen `StringBuilder`

- En `String` är oföränderlig, vi kan inte ändra dess innehåll.
- Om vi vill bygga upp en sträng kan vi använda ett `StringBuilder`-objekt.
- Samma metoder som på `String`, och dessutom, bland andra:
  - `void append(char ch)`: lägger till ett nytt tecken
  - `String toString()`: ger tillbaka motsvarande sträng
  - `void setCharAt(int index, char ch)`: sätter in ett tecken på ett givet ställe
- Metoderna `append` och `setCharAt` returnerar egentligen en `StringBuilder` (objektet själv), men vi kommer att anropa dem som om de vore `void`-metoder.

`do-while`-satsen



- Den vanliga `while`-satsen testar sitt villkor *före* varje varv.
- `do-while`-satsen testar sitt villkor *efter* varje varv.
- Ibland har vi inget att testa förrän vi har kört ett varv – då är `do-while`-satsen enklare att använda.

Programkontroll

- Vi kan styra en loop med tre olika kommandon:
  - `return`: avbryter hela det underprogram vi befinner oss i (och därmed också eventuella loopar).
  - `break`: avbryter loopen, och hoppar direkt till första satsen efter loopen.
  - `continue`: påbörjar nästa varv i loopen direkt.
- Dessa kommandon gör det möjligt att testa loop-villkor inuti själva loopen, istället för bara före (eller efter) varje varv.

Eviga loopar

- Ofta är problemet med att skriva loopar att formulera dem så att vi enkelt kan testa villkoret före varje varv.
- Ett användbart alternativ är att skriva en 'evig' loop, och bryta den inuti med hjälp av `break` eller `return`.
- Två vanliga sätt att skriva eviga loopar:

```
while (true) ...

for (;;) ...
```

Ett nytt sätt att skriva ut

- Tre sätt att skriva ut:
  - `System.out.print` – skriver ut utan radbrytning
  - `System.out.println` – skriver ut med radbrytning efteråt
  - `System.out.printf("...", ..., ...)` – låter oss finjustera utskriften
- Olika typer i format-strängen:
  - "%..d" – heltal
  - "%..f" – reella tal
  - "%..s" – strängar
- `printf` är väldigt bekvämt, men inte statiskt typat.

Kopplade `if`-satser

- Ofta undersöker vi ett heltalsvärde och testar mot olika alternativ:

```
int choice = Keyboard.nextInt("Välj (1-4): ");
if (choice == 1) {
    ...
} else if (choice == 2) {
    ...
} ... {
    ...
} else {
    ...
}
```

- Vi kan istället använda `switch`-satsen.

`switch`-satsen

- Den vanligaste formen av `switch`-satsen är:

```
int choice = Keyboard.nextInt(”Välj (1-4): ”);
switch (choice) {
case 1:
    ...
    break;
case 2:
    ...
    break;
...
case default:
    ...
}
```

- Här är `break` viktigt, annars får vi *fall-through* (vilket kan vara fiffigt, men ofta är fel).

“Call by value”

- När vi anropar en funktion så får parametrarna *värden*, dvs ett uttryck beräknas och lagras i parametern.
- Parametern finns bara i underprogrammet, den påverkar inte någon variabel i det anropande programmet.
- Om vi skickar en referens som parameter så kan vi ändra i det objekt som parametern refererar till, men vi kan inte ändra på själva referensen.

Olika typer av fel

- Kompileringsfel
- Exekveringsfel
- Logiska fel

Att fånga exekveringsfel

- Normalt avbryts programmet när vi får ett exekveringsfel.
- Vi kan ’fånga’ exekveringsfel med hjälp av en `try-catch`-sats:

```
try {
    ... ’farliga’ rader ...
} catch (Exception e) {
    ... ta hand om felet ...
}
```

- Vi kan även fånga olika slags fel med specificerade `catch`-satser.

Att kasta exekveringsfel

- Ofta är det svårt att avgöra vad vi skall returnera om vi får orimliga indata till en funktion.
- Ett alternativ är att kasta ett exekveringsfel:

```
double triangleArea(double a, double b, double c) {
    if (!triangular(a, b, c)) {
        throw new RuntimeException("...");
    }
    ...
    return ...
}
```