ex2methods

Joachim von Hacht

Överlagrade Metoder

```
// Get max of two values
double max( double d1, double d2){...}
float max( float d1, float d2){...}
int max( int d1, int d2){...}
int[] max( int[] d1, int[] d2){...}
```

2

Metoder inom samma synlighetsområde får ha samma namn …

- … men då måste parameterlistorna skilja sig åt!
  - Returtypen spelar ingen roll, bara parametrarna räknas.
- Kallas att metoderna är **överlagrade** (overloaded)
  - Innebär att redan vid kompileringen väljs vilken metod som skall anropas vid körningen.
  - Vilken metoden som väljs beror på parameterlistan
    - Implicita typomvandlingar kan förekomma för att hitta matchande metod
  - Jämför +-operatorn vars beteende beror på operanderna!
- Tanken med överlagrade metoder är att man skall slippa hitta på nya namn på metoder som gör "samma sak" men med olika antal eller parametertyper.
  - Metoder som gör olika saker skall inte ges samma namn, mycket viktigt med bra namn (verb)!

Begränsningar Överlagring

```
// Limitations
void shuffle(int[] arr) {     // Overloaded methods

}

void shuffle(double[] arr) { … }

void shuffle(String[] arr) { … }

void shuffle(Player[] arr) { … }

// Etc ... phui ...
```


Ibland räcker inte överlagring till.

- T.ex väldigt generella operationer t.ex shuffle …
- Så fort vi skapar en egen typ måste vi lägga till en överlagrad metod.

Generiska Metoder

```
// Generic version of Fisher Yates. <T> tells
// it's a generic method T stands for any reference type
<T> void shuffle(T[] arr) {
   for (int i = arr.length; i > 1; i--) {
       int j = rand.nextInt(i);
       T tmp = arr[j];          // T is any type
       arr[j] = arr[i - 1];
       arr[i - 1] = tmp;
   }
}
```

4

Då överlagring inte räcker till är generiska metoder ett alternativ.

Generisk metoder kan ta vilken referenstyp som helst som parameter och returtyp

- Generiska metoder kan inte ta primitiva typer
  - För att lösa detta kan man använda omslagstyper
- Man anger att metoden är generisk m.h.a. av en typparameter inom vinkelparenteser, normalt kallad T, först av allt i metodhuvudet (före returtypen)

Generisk Sökning/Sortering

```
// Comparable will guarantee there is a compareTo method.
<T extends Comparable<T>> T max(T[] arr) {
   T max = arr[0];
   for (int i = 1; i < arr.length; i++) {
       if (arr[i].compareTo(max) > 0) {  // Here!
           max = arr[i];
       }
   }
   return max;
}
```

5

Att bara ange en typ T gör att vi i princip inte kan göra något med objekten

- Hur skall kompilatorn veta att det vi gör är ok då den inte vet något specifikt om typen?
- För att lösa detta kan man ange att typen har vissa operationer t.ex. att det går att jämföra element av typen.
- Kan anges som i bilden med extends (mer i senare kurser)

Metoder med Sidoeffekter

```
// Intuitively 0 ... (?)
out.println(c.getValue1() - c.getValue1());
```

6

Tidigare haft uttryck med sidoeffekter (x++ t.ex.)

- Samma fenomen kan uppträda för metoder.
- Innebär i vårt fall att metoden förändrar en instansvariabel
- Dvs: Om en metod returnerar ett resultat och på samma gång ändrar en instansvariabel så har vi en metod med sidoeffekt
    - ** Undvik **!

Referentiell transparens (? svenska begrepp saknas, referential transparency)

- Enkelt sagt: Givet samma indata, får man alltid samma utdata från metoden?
    - Om så är det lättare att resonera om program (korrekthet)
- Metoder med sidoeffekter innebär att programmet inte blir referetiellt transparent
    - Kan inte undvikas i imperativ programmering, …
    - … men man kan försöka minimera
    - Metoder med returvärden undviker att ändra instansvariabler
    - void-metoder ändrar ofta instansvariabler, men metoderna är inga uttryck, vi får inget värde..
- Försök alltid att skriva metoder som bara tar indata och returnerar utdata …
    - …om tvunget, använd instansvariabler.

- Du behöver instansvariabler om något måste kommas ihåg mellan metodanropen.

Rekursiva Metoder

Matematisk definition

$F_n = F_{n-1} + F_{n-2}$ (1, 1, 2, 3, 5, 8, 13, ...)

$F_1 = 1, F_2 = 1$

```
// Recursive method
int fibonacci(int n) {
   if (n <= 1) {
       return n;
   } else {
       return fibonacci(n - 1) + fibonacci(n - 2);
   }
}
```

7

En rekursiv metod är en metod som anropar sig själv

- Varje anrop skapar nya variabler på anropsstacken.
- För att inte fylla anropsstacken måste metoden avslutas inom rimligt antal anrop.
- I bilden: n minskas hela tiden och blir förr eller senare 1 (då sker inga fler anrop)

Rekursiva metoder kan ibland ge eleganta lösningar till knepiga problem

- Ofta då man har mer komplicerade datastrukturer (grenande)
- Metoderna kan ibland direktöversättas från matematiska definitioner
- I Bilden: Rekursiv variant av fibonacci talen (1,1,2,3,5,8, …)
    - … tyvärr väldigt ineffektiv. Många metodanrop (går att förbättra)!

StackOverflowError

```java
void program() {
   program();  // Oh, ooh
}
```

```
/home/hajo/devel/jdk1.8.0/bin/java ...
Exception in thread "main" java.lang.StackOverflowError
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
    at samples.basics.ForLoop.program(ForLoop.java:14)
```


Felaktiga rekursiva anrop leder till StackOverflowError.