Datalogi C
Henning Christiansen

Opgave knyttet til forelæsningen 21. september 2004

Der er ikke øvelser fredag 24. september pga. årsfest, så I må arbejde selvstændigt med opgaven. Vi samler op på den ved forelæsning 28. september eller øvelserne 1. oktober.

Opgaven drejer sig om at forstå nedarvning og hvordan det kan bruges til at skabe generiske klasser. I opgaven skal I konstruere nogle klasser og lave nogen testprogrammer for at overbevise jer om, at jeres kode faktisk virker.

Jeres løsninger bliver ikke til kørende programmer, men til klasser, man kunne tænke sig indgår i en general værktøjskasse, som andre programmører kan benytte sig af.

I den forrige opgave udviklede vi en repræsentation af ord, således at hvert ord efter indlæsningen var identificeret ved et tal, således at den efterfølgende behandling kan foregå mere effektivt idet sammenligning af strenge undgås.

Opgaven her handler om at lave en generisk klasse som rummer princippet med at repræsentere arbitrære objekter (ikke bare teksstrenge) ved tal.

Iøvrigt: I må meget gerne tage udgangspunkt i den udleverede løsning til opgaven fra sidste gang. Udover selve den tekniske løsning indeholder omtalte opgaveløsning en del diskussion af uhensigtsmæssigheder i Java — det anbefales at man her ser helt bort fra det og koncentrerer sig om at løse opgaven »efter kogebogen«. Løsningen kommer så sandsynligvis til at afspejle

Spørgsmål 1

Benyt de redskaber i Java som blev gennemgået ved forelæsningen til at skrive en generisk klasse SmartRepresentation som svarer klassen Word fra forrige opgave. Dog således at den ikke er begrænset til at indeholde opbevare tekstrenge, men arbitrære objekter.

Den skal kunne benyttes noget i denne stil:

```
//Her benytter vi den på samme måde som den tidligere klasse Word
SmartRepresentation w1 = new SmartRepresentation (“monkey”);
SmartRepresentation w2 = new SmartRepresentation (“donkey”);
if( w1.equals(w2) ) .... ;

//Her benytter vi den til andre objekter
SmartRepresentation  myCar = new SmartRepresentation ( new car(“Ferrari”,  “575 GTC”, RED));

System.out.println(myCar);
```

Skriv en sådan klasse SmartRepresentation med konstruktor og metoderne equals og toString; internt skal den fungere ligesom sin forgænger Word ved at hvert objekt (sendt med som argument til konstruktoren) identificeres med et passende tal. Skriv nogle små testprogrammer — lidt i stil med ovenstående testeksempler, men gerne lidt mere interessante.

Spørgsmål 2

Skriv en generisk version af klassen Sequence fra forrige opgave, men således at den kan indeholde sekvenser af længde 1, 2, 3 eller 4 af arbitrære objekter; implementer og aftest den præcis som i spørgsmål 1.

At se på sekvenser af ord er en oplagt anvendelse; det er måske ikke så tit man kigger på sådanne sekvenser af biler, men prøv at se om du kan finde på andre interessante anvendelser hvor sekvenser optræder (du behøver ikke implementere det).

Spørgsmål 3 (ekstraspørgsmål til de hurtige)

I stedet for at sætte en øvre grænse på 4 for længden af sekvenser, skal klassen ændres, så den kan håndtere arbitrært lange sekvenser af objekter, dvs. af længde 0, 1, 2, ..., 1000, ...

Et væsentligt spørgsmål er hvordan gode konstruktorer kan designes. Bemærk, at på grund af at hver sekvens har et nummer inden i sig, som afhænger entyding af, hvilke objekter der er i sekvensen og hvilken rækkefølge, de optræder. Derfor giver det ikke mening at ændre på et Sequence-objekt efter det er oprettet, så man kan ikke slippe uden om konstruktorproblemet med en metode addOneMore som benyttes mySequence.addOneMore(new SmartRepresentation(...)).