EDA061
Objektorienterad modellering och design

Föreläsning 1: Introduktion

Christian.Soderberg@cs.lth.se

29 augusti 2016

Administration

- Kurs: EDA061 – Objektorienterad modellering och design
- Omfattning: 4.5 hp
- Kursansvariga: Ulf Asklund och Christian Söderberg
- Kurshemsida: `www.cs.lth.se/EDA061`
- Föreläsare: Christian Söderberg
- Kursombud: ?, ?

Exempel

```
public static void generateCode(Instruction c) {
    switch (c.opCode) {
    case 0: //MOV
        if (!(c.arg1 instanceof Current || c.arg1 instanceof Next) &&
            !(c.arg2 instanceof Current || c.arg2 instanceof Next)) {
            out.writeBytes("mov " + code.convert(c.arg1) + " , " +
                           code.convert(c.arg2) + "\n");
        } else if ((c.arg1 instanceof IntConst || c.arg1 instanceof BoolConst) &&
                    c.arg2 instanceof Current) {
            out.writeBytes("mov " + code.convert(c.arg1) +
                           ", "+ code.saveTo Reg(c.arg1) +"\n");
            out.writeBytes("set " + code.convert(c.arg2,-staticlevel1) +
                           " , " + code.saveToReg(c.arg2) +"\n");
            out.writeBytes("st " + code.saveToReg(c.arg1) + " , " +
                           "[" + code.saveToReg(c.arg2) + "]\n");
        } else if((c.arg1 instanceof Temp) && c.arg2 instanceof Current) {
            out.writeBytes("set " + code.convert(c.arg2,-staticlevel1) + " , " +
                           code.saveToReg(c.arg2) +"\n");
            out.writeBytes("st " + code.saveToReg(c.arg1) + " , " +
                           "[" + code.saveToReg(c.arg2) +"]\n");
        } else if(c.arg1 instanceof Current && c.arg2 instanceof Temp) {
            out.writeBytes("set " + code.convert(c.arg2) + " , " +
                           code.saveToReg(c.arg2) +"\n");
            out.writeBytes("ld " + "["+ code.saveToReg(c.arg2) +"]" + " , " +
                           code.saveToReg(c.arg1) +"\n");
        } else if(c.arg1 instanceof Current && c.arg2 instanceof Current) {
```

Några begrepp för att diskutera kod

- *Cohesion*: grad av samstämmighet/sammanhållning (koherens på svenska)
- *Coupling*: graden av koppling mellan olika delar
- Vi strävar efter att skapa system med *"high cohesion"* och *"low coupling"*

Några "Code Smells"

- *Duplicated code*: identisk, eller nära identisk kod på flera ställen
- *Long method*: metod/funktion som är onödigt lång
- *Large class*: klass som är onödigt stor

"Code Smells" i kursboken

- *Rigidity*: Svårt att ändra koden, eftersom en ändring påverkar många delar av programmet
- *Fragility*: Ändringar får till synes orelaterade delar av programmet att sluta fungera
- *Immobility*: Det är svårt att bryta ut och återanvända delar av koden
- *Viscosity*: Ändringar blir enklare att göra om man 'fuskar'
- *Needless complexity*: Använder komplicerad infrastruktur som inte gör någon nytta
- *Needless repetition*: Upprepad kod som kunde ha brutits ut till en klass eller en funktion
- *Opacity*: Svårläst kod, visar inte intentionen tydligt

Exempel

```
public static void generateCode(Instruction c) {
    switch (c.opCode) {
    case 0: //MOV
        if (!(c.arg1 instanceof Current || c.arg1 instanceof Next) &&
            !(c.arg2 instanceof Current || c.arg2 instanceof Next)) {
            out.writeBytes("mov " + code.convert(c.arg1) + " , " +
                           code.convert(c.arg2) + "\n");
        } else if ((c.arg1 instanceof IntConst || c.arg1 instanceof BoolConst) &&
                    c.arg2 instanceof Current) {
            out.writeBytes("mov " + code.convert(c.arg1) +
                           ", "+ code.saveTo Reg(c.arg1) +"\n");
            out.writeBytes("set " + code.convert(c.arg2,-staticlevel1) +
                           " , " + code.saveToReg(c.arg2) +"\n");
            out.writeBytes("st " + code.saveToReg(c.arg1) + " , " +
                           "[" + code.saveToReg(c.arg2) + "]\n");
        } else if((c.arg1 instanceof Temp) && c.arg2 instanceof Current) {
            out.writeBytes("set " + code.convert(c.arg2,-staticlevel1) + " , " +
                           code.saveToReg(c.arg2) +"\n");
            out.writeBytes("st " + code.saveToReg(c.arg1) + " , " +
                           "[" + code.saveToReg(c.arg2) +"]\n");
        } else if(c.arg1 instanceof Current && c.arg2 instanceof Temp) {
            out.writeBytes("set " + code.convert(c.arg2) + " , " +
                           code.saveToReg(c.arg2) +"\n");
            out.writeBytes("ld " + "["+ code.saveToReg(c.arg2) +"]" + " , " +
                           code.saveToReg(c.arg1) +"\n");
        } else if(c.arg1 instanceof Current && c.arg2 instanceof Current) {
```

Agile development

- 'Code repositories' (GitHub, GitLab, Bitbucket, …)
- Korta 'release'-cykler och utveckling i 'feature branches' – gör att vi alltid enkelt kan hämta en körbar version av programmet
- 'Pull requests' och 'code reviews', mycket social interaktion (om än ofta virtuell)
- System för att bygga och testa koden, 'regression testing'
- Koden designad för att vara lätt att läsa och lätt att anpassa till ändrade krav

Färdigheter som ni skall lära er

- Skriva kod som är:
  - lätt att läsa och förstå
  - lätt att ändra
- Granska kod och föreslå förbättringar
- Arbeta i grupp med ett 'code repository'

Vad vi kommer att behandla

- Terminologi för att diskutera kodkvalitet
- Viktiga principer för programkod
- Beprövade tekniker/mönster
- De idéer som vi kommer att behandla i kursen har växt fram ur erfarenheter som programmerare har samlat på sig under ca 60 år – *please, give them 5 minutes!*
  (`https://signalvnoise.com/posts/3124-give-it-five-minutes`)

Kursmoment/tidsbudget

Antal	Moment	Tid
7	Föreläsningar	28
3	Seminarier	24
2	Projektuppgifter	46
1	Laboration	2
	Tentamen	20
		120

Förväntade förkunskaper

- Att ni förstår begrepp som (på engelska): *static typing*, *object*, *class*, *abstract class*, *interface*, *overloading*, *overriding*, *subtype polymorphism*, *dynamic binding*
- Att ni kan läsa en enkel uppgiftstext och på datorn sätta ihop ett fungerande program
- Kontakta mig om det är något av detta du känner dig osäker på

Ändringar sedan förra året

- Kursboken är fortfarande läsvärd, men vi kommer även att lägga ut pdf-filer för att försöka täcka upp de viktigaste delarna av kursinnehållet
- Ni slipper att själva redovisa lösningar under övningarna, istället går jag igenom uppgifterna under gemensamma seminarieövningar
- Ni rekommenderas att arbeta med seminarieuppgifterna själva, och granska varandras lösningar i förväg (skicka gärna in lösningar som ni vill få diskuterade under seminarierna)
- Projekten körs med hjälp av ett riktigt projektverktyg (`bitbucket.com`)
- Vi sätter samman projektgrupperna

Exempel

Skriv ett program som skriver ut summan av de av kommandoradsparametrarna som är primtal.

Top-down design

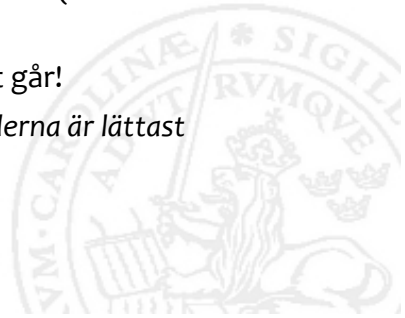
- Det är ofta en god idé att önsketänka, och anropa funktioner som hade varit användbara
- När vi sedan skriver dessa funktioner gör vi likadant, dvs anropar andra användbara funktioner
- Metoden kallas top-down design, och vi kan mycket väl även skapa objekt av önskade klasser och anropa önskade metoder på dessa
- Vi får på detta vis korta funktioner/metoder, och de har hög ’cohesion’

Exempel

Granska och refaktorisera givna klasser för punkter och segment.

Intern representation

- Det finns flera skäl att gömma objektens interna representation, bland andra:
  - Det gör klassen enklare att använda (färre detaljer att hålla reda på)
  - Det gör det enklare att ersätta objekten med objekt av andra, besläktade klasser
  - Det gör det möjligt att refaktorisera klassen utan risk att anropande program behöver skrivas om (vi minskar risken för 'coupling')
- Undvik `get`- och `set`-metoder, om det går!
- *Implementera operationer där operanderna är lättast tillgängliga!*



Exempel

Antag att vi även vill kunna använda 3D-punkter – hur gör vi, och vilka förändringar behövs i `Segment`*?*