Objektorienterad modellering och diskreta strukturer / design

Mål och innehåll

Lennart Andersson

Reviderad 2011–08–29

2011

Kursstruktur

kurs	HT1 v1-4	HT1 v4-7	TP 1	HT2 v1-3
EDAF10	Principer och mönster, Projekt Computer	Diskreta strukturer	Tentamen	Projekt XL
EDA061	Principer och mönster, Projekt Computer	Projekt XL	Tentamen	

Vad är Objektorienterad modellering?

Modellering av verkligheten med intelligenta objekt och design av program som implementerar modellen.

- Vilka klasser skall finnas?
- Vilka metoder skall de tillhandahålla?
- Hur skall de paketeras?

Programmering utan OMD

```
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.saveToReg(c.arg1) +"\n");
        out.writeBytes("set "  + code.convert(c.arg2,staticlevel-1) +" , " +
            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,staticlevel-1) +
            " , " + 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) {
        if(!(c.arg1.toString().equals(c.arg2.toString()))) {
            out.writeBytes("set "  + code.convert(c.arg1,staticlevel-1) +
                " , "+ code.saveToReg(c.arg2) +"\n");
```

28 sidor senare

```
case 20:  //OR
    if (c.arg1 instanceof BoolConst) {
        reg1 = code.saveToReg(c.arg1);
        out.writeBytes("mov " + code.convert(c.arg1) + " , " +
            reg1 + "\n");
    } else if (c.arg1 instanceof Current) {
        reg1 = code.saveToReg(c.arg1);
        out.writeBytes("set " + code.convert(c.arg1, staticlevel-1) +
            " , " + reg1 +"\n");
        out.writeBytes("ld [" + reg1 +"] , " + reg1  + "\n");
    } if(c.arg2 instanceof Current) {
        reg2 = code.saveToReg(c.arg2);
        out.writeBytes("set " + code.convert(c.arg2, staticlevel-1) +
            " , " + reg2 +"\n");
        out.writeBytes("ld [" + reg2 + "] , " + reg2 + "\n");
    } if(!(c.arg1 instanceof Temp)) {
        out.writeBytes("or " + reg1 +" , ");
    } else {
        out.writeBytes("or " + code.convert(c.arg1) + " , ");
    } if(c.arg2 instanceof Current) {
```


Aktivitet

Vid en motorcykeltävling registreras start- och sluttider på filer på följande format.

```
StartNr; Starttid    StartNr; Sluttid
1; 12.00.00          3; 13.05.06
2; 12.01.00          2; 13.15.16
3; 12.02.00          1; 13.23.34
```

Ett sorteringsprogram läser filerna och producerar följande resultat.

```
StartNr; Totaltid; Starttid; Sluttid
1; 1.23.34; 12.00.00; 13.23.34
2; 1.14.16; 12.01.00; 13.15.16
3; 1.03.06; 12.02.00; 13.05.06
```

Konstruera en klass som kan användas för att representera tider i start- och resultatlistor.

Punkt med dålig design . . .

```
public class Point {
    private double x, y;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return x;
    }

    public void setX(double x) {
        this.x = x;
    }
//  omissions
}
```


. . . Dålig design

```
public class Segment {
    private Point point0, point1;

    // Standard constructor omitted

    public double length() {
        double dx = point1.getX()-point0.getX();
        double dy = point1.getY()-point0.getY();
        return Math.sqrt(dx*dx+dy*dy);
    }
}
```

Aktivitet

Vad är det som är dåligt i detta program?

1. `Point` saknar intelligens.
2. `Point` saknar integritet; lämnar ut attributen i onödan.
3. `Segment` är onödigt beroende av `Point`; man kan inte byta till 3 dimensioner utan att ändra båda klasserna.

Kursens syfte

Kursen ger förmåga till hållbar och resursmedveten utveckling av program som kan återanvändas och modifieras med hänsyn till förändrade krav i ett industriellt sammanhang.

Kursens mål

Efter genomgången kurs ska studenten kunna

- lokalisera och känna igen användning av gängse designprinciper och designmönster i givna program,
- utforma och implementera objektorienterade program med många klasser och några paket,
- välja och implementera lämpliga designmönster i typiska problem,
- använda centrala delar av en integrerad utvecklingsmiljö för design, implementering och omstrukturering av program,
- beskriva programdesign med UML (Unified Modeling Language),
- utvärdera en programdesign med avseende på designprinciper samt
- skriva program som är lätta att förstå för den som behöver göra modifieringar.

Du kan redan ...

- det mesta i programspråket Java
- vanliga datastrukturer
- skriva fungerande program

Kurslitteratur

- Robert C. Martin: Agile Software Development - Principles, Patterns, and Practices, Prentice Hall, 2003, ISBN 0-13-597444-5. Ny upplaga 2011.
- Lennart Andersson: UML – Syntax, Datavetenskap LTH, 2010. Finns på nätet.
- Lennart Andersson: Diskreta strukturer, Datavetenskap LTH, 2010. (EDAF10). Finns på nätet.
- Övnings-, laborations- och projektuppgifter publiceras på hemsidan.

Schema Modellering och design

v35	måndag 8.15 torsdag 10.15	Kårhör E:A	föreläsning föreläsning
v36	måndag 8.15 onsdag eller fredag torsdag 10.15	Kårhör E:?	föreläsning övning föreläsning laboration
v37	måndag 8.15 tis-tor onsdag eller fredag torsdag 10.15	Kårhör inst E:A	föreläsning projektmöte övning föreläsning
v38	onsdag eller fredag		övning
v41	torsdag 10.15	E:A	föreläsning

Övningar vecka 37–39

Det finns två sorters övningspass:

- pass där studenterna förväntas presentera lösningar till förelagda problem.
- pass där en lärare presenterar lösningar till de förelagda problemen.

Anmälan till schemapass sker via hemsidan.

Bonus

Deltagande på övningarna är frivilligt, men de som är beredda att presentera sina lösningar inför övningsgruppen har möjlighet att få upp till 15% bonus för detta vid första ordinarie tentamenstillfälle.

Samarbete rekommenderas!

Laboration med Eclipse

Innehåll

- refaktorisering
- UML-verktyget

Utförande

- ej schemalagd
- utföres på egen hand
- redovisas ej

Projekt

1. Computer (vecka 36-37)
2. XL – kalkylprogram
   (EDA061: vecka 38-41, EDAF10: vecka 43-45)

- Grupper om 4 studenter
- 3 designmöten med handledare

Tentamen

Läroboken (Martin), UML-häftet och Diskreta strukturer får medföras på tentamen.

Tidsbudget

EDA061			EDAF10		
antal	**moment**	**tid**	**antal**	**moment**	**tid**
7	föreläsningar	28	15	föreläsningar	60
3	övningar	24	6	övningar	48
2	projektuppgifter	46	2	projektuppgifter	46
1	laboration	2	4	laborationer	14
	tentamen	20		tentamen	32

Redesign

```
public class Point {
    private double x, y;

    public double distanceTo(Point other) {
        double dx = this.x - other.x;
        double dy = this.y - other.y;
        return Math.sqrt(dx * dx + dy * dy);
    }
}
```


Redesign

```
public class Segment {
    private Point point0, point1;

    public double length() {
        return point1.distanceTo(point0);
    }
}
```


3D-punkter . . .

```
public interface Point {
    public double distanceTo(Point other);
}
```


. . . 2D-punkter . . .

```
public class Point2d implements Point {
    private double x, y;

    public double distanceTo(Point point) {
        Point2d other = (Point2d) point;
        double dx = this.x - other.x;
        double dy = this.y - other.y;
        return Math.sqrt(dx * dx + dy * dy);
    }
}
```

. . . 3D-punkter

```
public class Point3d implements Point {
    private double x, y, z;

    public double distanceTo(Point point) {
        Point3d other = (Point3d) point;
        double dx = this.x - other.x;
        double dy = this.y - other.y;
        double dz = this.z - other.z;
        return Math.sqrt(dx * dx + dy * dy + dz * dz);
    }
}
```


Oförändrad

```
public class Segment {
  private Point point0, point1;

  public Segment(Point point0, Point point1) {
    this.point0 = point0;
    this.point1 = point1;
  }

  public double length() {
    return point1.distanceTo(point0);
  }
}
```


Designprinciper för programmering

- Abstraktion
- Modularisering
- Integritet – skydda representationen
- Lokalisering – implementera operationer där operanderna finns

Dataabstraktion

- `00001001 01010010 00000000`
- `int [] word = new int[size]`
- `Memory memory`
-
```
public class Memory {
  private int [] word = new int [1024];
  public int getValue(int address) {
    return word[address];
  }
  public void setValue(int address, int value) {
    word[address] = value;
  }
}
```

Procedurell abstraktion

Före

```
if (c.arg1 instanceof Current) {
    reg1 = code.saveToReg(c.arg1);
    out.writeBytes("set " + code.convert(c.arg1, st
        " , " + reg1 +"\n");
    out.writeBytes("ld [" + reg1 +"] , " + reg1  +
} if(c.arg2 instanceof Current) {
    reg2 = code.saveToReg(c.arg2);
    out.writeBytes("set " + code.convert(c.arg2, st
        " , " + reg2 +"\n");
```

efter

```
c.arg1.generate(code, reg1);
c.arg2.generate(code, reg2);
```


Anderssons lokalitetsprincip, ALP

Operationer skall implementeras där operanderna är lättast tillgängliga.

Recept för ALP

Undvik getters!

Nim

Nim är ett tvåmansspel som spelas med ett antal högar av tändstickor. Spelarna turas om att göra drag. I varje drag skall spelaren minska antalet stickor i exakt en hög. Den spelare som eliminerar den sista högen vinner.
Designa ett objektorienterat program som spelar Nim med en människa vid tangentbordet.

Nim – substantiv

Nim är ett tvåmansspel som spelas med ett antal högar av tändstickor. Spelarna turas om att göra drag. I varje drag skall spelaren minska antalet stickor i exakt en hög. Den spelare som eliminerar den sista högen vinner.
Designa ett objektorienterat program som spelar Nim med en människa vid tangentbordet.

Nim – klasser

Nim	class Nim
spel	synonym till Nim
antal	ges av size() i PileList
tändsticka	saknar tillstånd och operationer
spelare	abstract class Player
drag	operation
högar	class PileList extends ArrayList<Pile>
antalet	attribut i Pile
hög	class Pile
program	class Computer extends Player
människa	class Human extends Player
tangentbordet	tillhör användargränssnittet

Metoder

Nim	void play()
	boolean isEnded()
	void reset()
PileList	boolean isEmpty()
	void remove(int index, int count)
	String toString()
Pile	void remove(int count)
	String toString()
Player	void move()

Användningsfall (Use cases)

- skapa högarna
- människan gör ett drag
- datorn gör ett drag
- avgör om någon vunnit

Aritmetiska uttryck

Ett *aritmetiskt uttryck* är antingen ett tal eller en addition, multiplikation, en subtraktion eller en division av två uttryck.

Modellera aritmetiska uttryck med substantivmetoden och förse klasserna med en metod som beräknar uttryckets värde.

Aktivitet

Ett *aritmetiskt uttryck* är antingen ett tal eller en addition, en multiplikation, en subtraktion eller en division av två uttryck.

Modellera aritmetiska uttryck med substantivmetoden och förse klasserna med en metod som beräknar uttryckets värde.

Expr – klasser

Vilka klasser skall finnas?

Expr och Num

```
public interface Expr {
    public int value();
}

public class Num implements Expr {
    private int value;
    public int value() {
        return value;
    }
}
```

Num behöver en standardkonstruerare

```
public class Num implements Expr {
    private int value;

    public Num(int value) {
       this.value = value;
    }

    public int value() {
        return value;
    }
}
```

Standardkonstruerare visas ej i fortsättningen.

Add

Expr – Exempel

- 1 `new Num(1)`
- $1 + 2$ `new Add(new Num(1), new Num(2))`
- $1 + 2 * 3$
```
new Add(
  new Num(1),
  new Mul(new Num(2), new Num(3)))
```
- $1 * 2 + 3$
```
new Add(
  new Mul(new Num(x1), new Num(2)),
  new Num(3))
```


UML

- Unified Modeling Language
- Booch, Rumbaugh, Jacobson, slutet av 90–talet
- många läroböcker och verktyg
- industristandard

Diagram

- Funktionell modell — Användningsfall (Use cases)
- Statisk modell — Klassdiagram
- Dynamisk modell — Sekvensdiagram, tillståndsdiagram

Ett klassdiagram