Objektorienterad modellering och diskreta strukturer / design

Mål och innehåll

Lennart Andersson

Reviderad 2012–09–04

2012

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) {
```

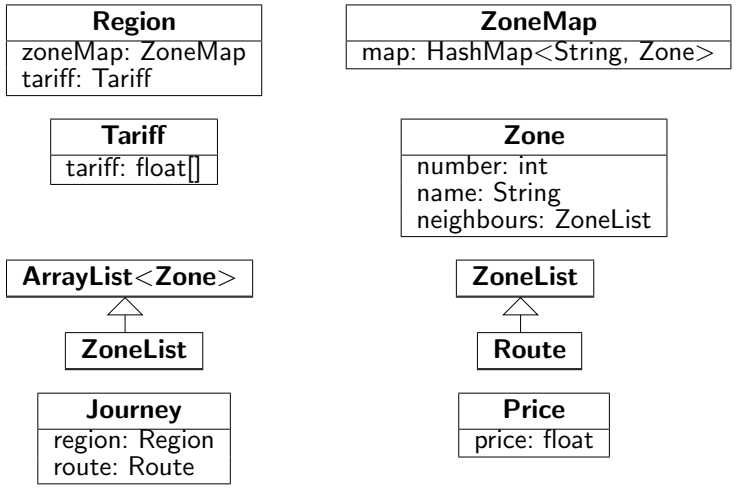
Tänkbara klasser

SkåneTrafiken	System	**Journey**
Region	**Zone**	**Price**
Route	**Tariff**	Number
Name	Traveller	Driver

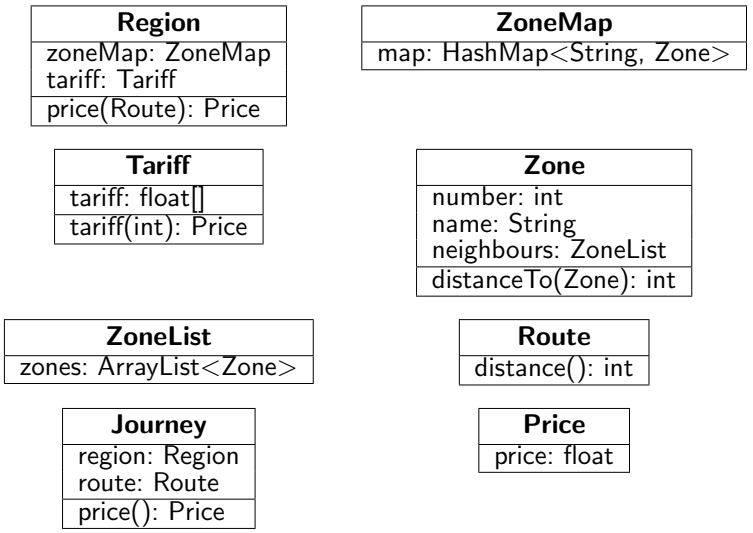
Designuppgift

Skånetrafiken har ett system för att ta betalt för resor i regionen, se `www.skanetrafiken.se`. Regionen är indelad i ett hundratal zoner och priset för en resa beror på hur många zoner resvägen passerar och framgår av en prislista.
En zon har både ett nummer och ett namn. Vanliga resenärer använder nästan alltid namnet medan en bussförare kanske föredrar numret när resvägen skall registreras. Ibland vill man inte åka närmaste vägen utan via en eller flera andra zoner.

Naturliga attribut



Nödvändiga metoder



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
}
```

... och sträcka

```
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?

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, 2012. Finns på kurshemsidan.
- Lennart Andersson: Diskreta strukturer, Datavetenskap LTH, 2010. (EDAF10). Finns på hemsidan.
- Övnings-, laborations- och projektuppgifter publiceras på hemsidan.

Schema Modellering och design

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

Övningar vecka 37–39

- På övningarna förväntas studenterna presentera lösningar till förelagda problem.
- Det är frivilligt att presentera lösningar, men den som är beredd att göra det får upp till 15% bonus vid första ordinarie tentamenstillfälle.
- Samarbete rekommenderas!

Anmälan till schemapass sker via hemsidan.

Laboration med Eclipse

Innehåll

- refaktorisering
- UML-verktyget

Utförande

- ej schemalagd
- utföres på egen hand
- redovisas ej

Projekt

1. Computer (vecka 37-38)
2. XL – kalkylprogram
   (EDA061: vecka 39-42, EDAF10: vecka 44-46)

- Grupper om 4 studenter
- 3 obligatoriska designmöten med handledare

Tentamen

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

Tidsbudget

EDA061		
antal	**moment**	**tid**
7	föreläsningar	28
3	övningar	24
2	projektuppgifter	46
1	laboration	2
	tentamen	20

EDAF10		
antal	**moment**	**tid**
15	föreläsningar	60
6	övningar	48
2	projektuppgifter	46
4	laborationer	14
	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 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, 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");
```

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!

Aktivitet

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

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.

Aktivitet

Implementera 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(1), new Num(2)),
  new Num(3))
```

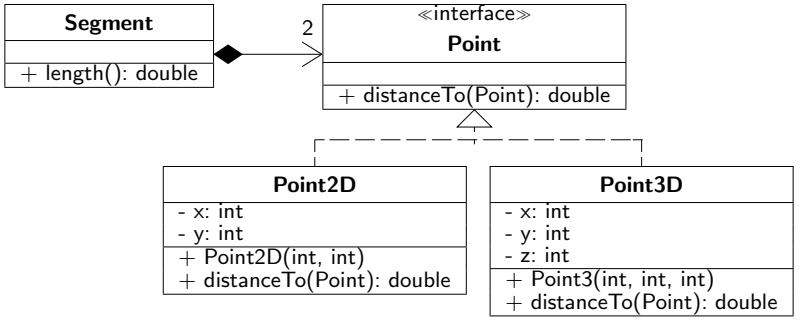

UML

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

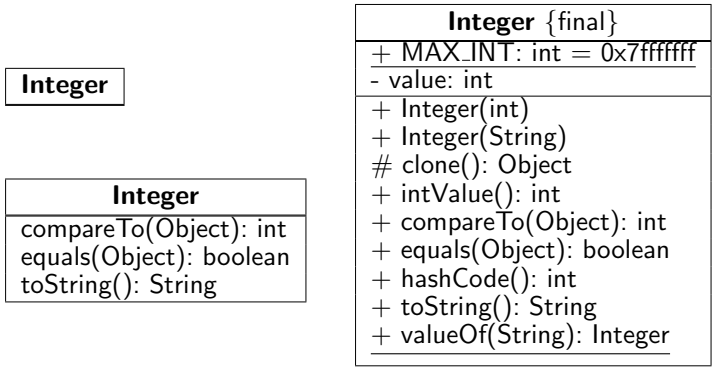
Diagram

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

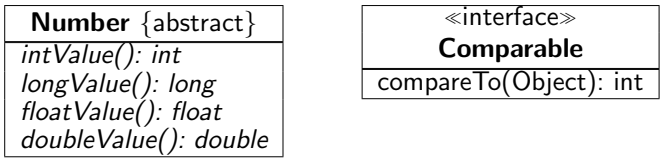
Ett klassdiagram



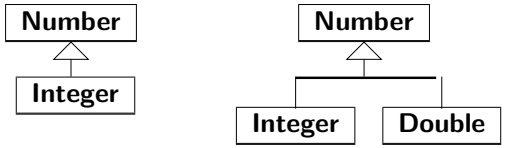
Klasser



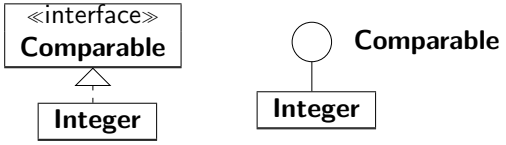
Abstrakta klasser och gränssnitt



Generalisering

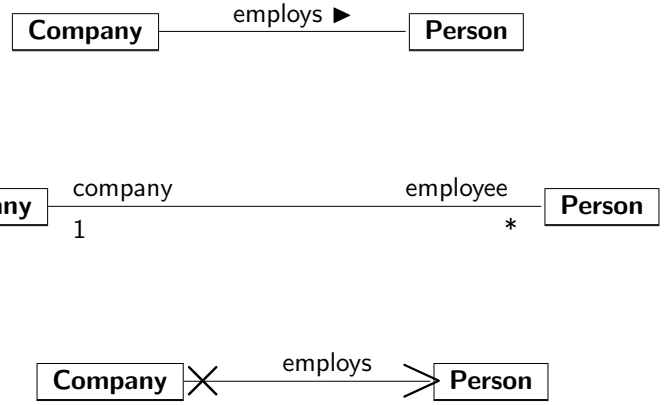


Realisering

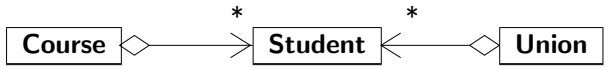


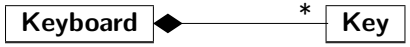
Parametriserade klasser

Associationer



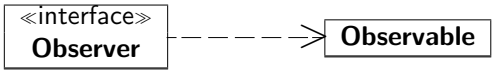
Aggregering och sammansättning





Beroende

```
interface Observer {
    Object update(Observable observable, Object object);
}
```



Aktivitet 11

```
public class Time implements Comparable<Time> {
    private int sec;
    public Time difference(Time other)
    public int compareTo(Time other)
    public String toString()
    public Time(String time)
}
```


Aktivitet 14

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.

Aktivitet 36

Vilka klasser skall finnas?

uttryck	interface Expr
tal	class Num implements Expr
addition	class Add implements Expr
multiplikation	class Mul implements Expr
subtraktion	class Sub implements Expr
division	class Div implements Expr

Aktivitet 39

```
public class Add implements Expr {
  private Expr expr1, expr2;
  public int value() {
    return expr1.value() + expr2.value();
  }
}
```