Objektorienterad modellering och diskreta strukturer / design

Inför nästa projekt

Lennart Andersson

Reviderad 2012–09–20

2012

Dagens program

- Strategimönstret och de fyras gäng
- Felhantering
- *Factory Method*-mönstret
- Syntaxanalys
- Tillståndsdiagram
- Introduktion av projekt 2
- Swing-komponenter och GUI-design

Alternativ kurslitteratur

Det finns en annan lärobok som skulle kunna vara kursbok.

Henrik Bærbak Christensen: *Flexible, Reliable Software — Using Patterns and Agile Development*, CRC Press, 2010. ISBN 978-4200-9326-9

3-1-2 – processen

Henrik Christensen sammanfattar användning av Strategi-mönstret i tre punkter:

3 Identify some behaviour that varies.
1 Describe this behaivior as an interface.
2 Implement each behaviour by a class.

Den egendomliga numreringen beror på att momenten anges i en annan ordning i Christensens inspirationskälla:

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software*

"Gang of Four"

Factory method–mönstret

Factory method används när man

- måste avgöra vilken sorts objekt som skall skapas vid exekveringen
- vill undvika beroende av konkreta klasser

Factory method – i Computer

- I Computer skapas operander och instruktioner direkt med konstruerare.
- Man skulle kunna skapa dessa från den externa representation.
- Vi gör en fabrik för operander
- och en för instruktioner.

Operandfabriken

```
public class OperandFactory {
    public Operand build(String string) {
        if(string.charAt(0)=='[') {
            return new Address(Integer.parseInt(
                string.substring(1,string.length()−1)));
        } else {
            return new Word(Integer.parseInt(string));
        }
    }
}
```

I programspråket Scala skulle man kallat metoden `Operand` eftersom det nästan är en konstruerare.

Instruktionsfabriken ...

```
public class InstructionFactory {
  private OperandFactory operandFactory = new OperandFactory();
  public Instruction build(String string) {
    StringTokenizer tokenizer = new StringTokenizer(string);
    Operand op1, op2, op3;
    String opCode = tokenizer.nextToken();
    if (opCode.equals("ADD")) {
      op1 = operandFactory.build(tokenizer.nextToken());
      op2 = operandFactory.build(tokenizer.nextToken());
      op3 = operandFactory.build(tokenizer.nextToken());
      return new Add(op1, op2, (Address) op3);
    } ...
```

... Instruktionsfabriken ...

```
} else if (opCode.equals("CPY")) {
  op1 = operandFactory.build(tokenizer.nextToken());
  op2 = operandFactory.build(tokenizer.nextToken());
  return new Copy(op1, (Address) op2);
} ...
```


... Instruktionsfabriken

```
    } else if (opCode.equals("HLT")) {
      return new Halt();
    } else {
      throw new RuntimeException("syntax error");
    }
  }
}
```


Vecka 4-6

vecka	ed061	edaf10
4		föreläsning DS
	avsluta Computer	
	XL design	föreläsning DS
	övning OMD	
5		föreläsning DS
		laboration DS
	designmöte	föreläsning DS
		övning DS
6		föreläsning DS
		laboration DS
	XL impl	föreläsning DS
		övning DS

Vecka 7

vecka	ed061	edaf10
7		föreläsning DS
	XL designmöte	laboration DS
	föreläsning OMD	
		övning DS

Att göra

EDA061 Anmälan till XL-projektet senast tisdagen den 25 september. Inlämning 3-4 oktober.

EDAF10 Anmälan till laborationer senast fredagen den 25 september. Börjar vecka 5.

EDAF10 Omanmälan för onsdags-övare senast fredagen den 25 september. Börjar vecka 5.

Deadline

equals i Object

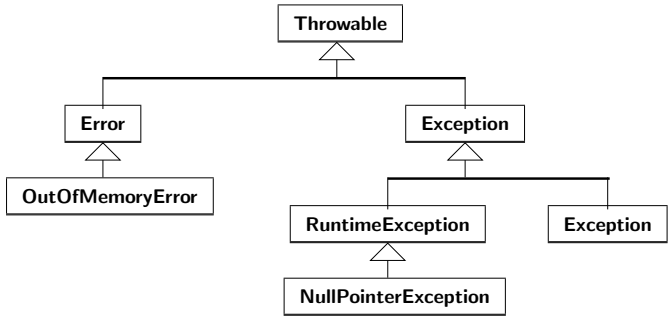
The equals method implements an equivalence relation on non-null object references: it must be reflexive, symmetric, transitive, and consistent.

Vad betyder detta?

Om man läst diskret matematik så vet man.

Alla kvalificerade datautbildningar i hela världen utom Lund har betydande inslag av diskret matematik och logik i början av utbildningen. I Lund finns det bara två poäng och dessa ligger i OMD-kursen.

Felhantering



Felhantering

```
public class XLException extends RuntimeException {
    private Object object;
    public XLException(String message, Object object) {
        super(message);
        this.object = object;
    }
    public Object getObject() {
        return object;
    }
}
```

Feldetektering

```
public class Div extends BinExpr {
  public Div(Expr expr1, Expr expr2) {
    super(expr1, expr2);
  }
  protected double op(double op1, double op2) {
    if (op2 != 0)
      return op1 / op2;
    else
      throw new XLException("division by zero", this);
  }
}
```


Felhantering

```
try {
    value = expr.value();
} catch (XLException e) {
    report(e.getMessage(), e.getObject());
} catch (RuntimeException e) {
    recompute();
    throw e;
}
```


Aritmetiska uttryck — Konkret syntax

- Ett *uttryck* består av en eller flera *termer* separerade av enkla plus- eller minus-tecken.
- En *term* består i sin tur av en eller flera *faktorer* separerade av enkla multiplikations- eller divisions- tecken.
- En *faktor* är ett *tal*, en *variabel* eller ett *uttryck* inom parenteser.
- Ett *tal* består av en eller flera siffror och får inledas med ett minustecken.
- En *variabel* består av en eller flera bokstäver bland a–z.

Konkret grammatik — BNF

expr ::= term (addop term)∗
term ::= factor (mulop factor)∗
factor ::= number | name | '(' expr ')'
addop ::= '+' | '−'
mulop ::= '∗' | '/'

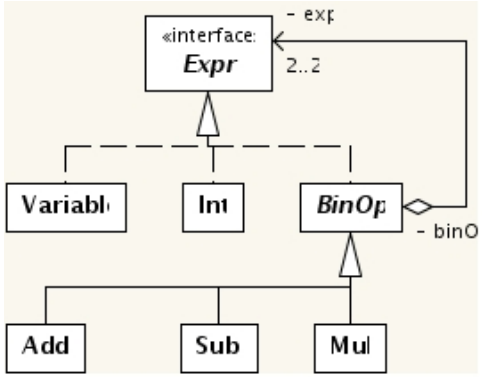
Operatorer

- ∗ upprepa 0 eller flera gånger
- | eller
- ' ' literalt

Tal — BNF

number ::= unsignedNumber | '−' unsignedNumber
unsignedNumber ::= digit (digit)∗
digit ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' |
'7' | '8' | '9'
name ::= letter (letter)∗
letter ::= 'a' | 'b' | ... | 'z'

Abstrakt representation — abstrakt grammatik



java.io.StreamTokenizer

```
public class public class StreamTokenizer {
  public double nval;
  public String sval;
  public int ttype = −4;
  public static final int
   TT_EOF = −1, TT_EOL = 10, TT_NUMBER = −2, TT_WORD = -

  public StreamTokenizer(Reader r)
  public int nextToken() throws IOException
  public void ordinaryChar(int ch)
  \\ omissions
}
```


java.io.StreamTokenizer

```
public class ExprParser extends StreamTokenizer {
    private int token;
    public ExprParser(String string) throws IOException {
        super(new StringReader(string));
        ordinaryChar('−');
        ordinaryChar('/');
        token = nextToken();
    }
```

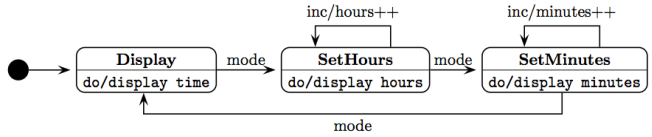
Analys av faktorer

```
private Expr factor() {
  Expr e;
  switch (token) {
  case '(':
    token = nextToken(); e = expr(); token = nextToken();
    return e;
  case TT_NUMBER:
    double x = nval; token = nextToken();
    return new Num(x);
  case TT_WORD:
    String s = sval; token = nextToken();
    return new Variable(s);
  }
}
```

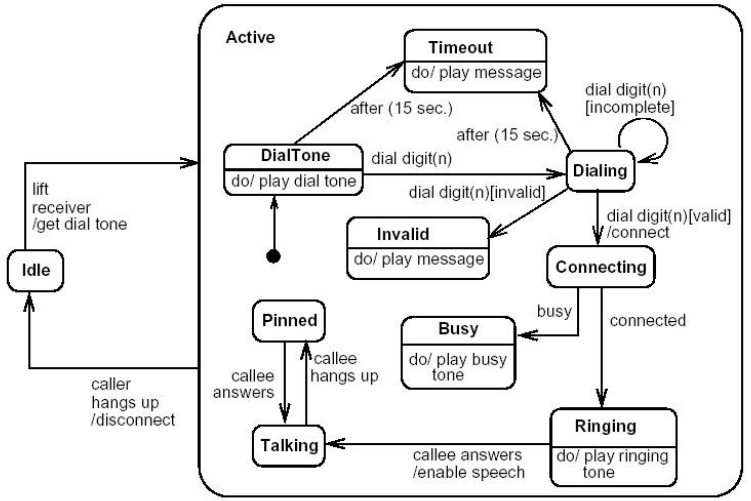

Analys av termer

```
private Expr term() {
    Expr result, factor;
    result = factor();
    while (token == '*' || token == '/') {
      int op = token;
      token = nextToken();
      factor = factor();
      switch (op) {
      case '*' :
        result = new Mul(result, factor); break;
      case '/' :
        result = new Div(result, factor); break;
      }
    }
    return result;
}
```


Tillståndsdiagram



En telefon



Namngivning av klasser

- Ett klassnamn är normalt ett substantiv hämtat från uppdragsgivarens beskrivning eller den verklighet som programmet modellerar.
- Namnet inleds med en versal och följs av gemener. På engelska använder man versal i början på varje ord när namnet är sammansatt.

Namngivning av klasser

- Konstruera inte klasser som bäst beskrivs med namn som slutar på `Handler`, `Controller`, `Manager` och liknande. De indikerar ofta dålig design. `Builder` och `Parser` kan vara helt OK för en "fabrik".
- Undvik namn som `Item`, `Element`, `Value` och `Data` om det inte handlar om generiska typer.
- Undvik att inkludera representationstypen i namnet om typen bara förekommer i ett attribut.

  class StringList **extends** ArrayList<String>

  är OK, medan

  ```
  class StringArray {
    private String string [];
    \\ omissions
  }
  ```
  inte är det.

Namngivning av gränssnitt

- Samma principer som för klassnamn.
- Namn på gränssnitt används oftare än namn på konkreta klasser; därför är det viktigare att dessa namn är bra.
- Undvik att indikera att det handlar om gränssnitt som i `ITree` `TreeInterface`. Hierarkien som i `List`, `AbstractList` och `ArrayList` är bra när alla nivåerna behövs.

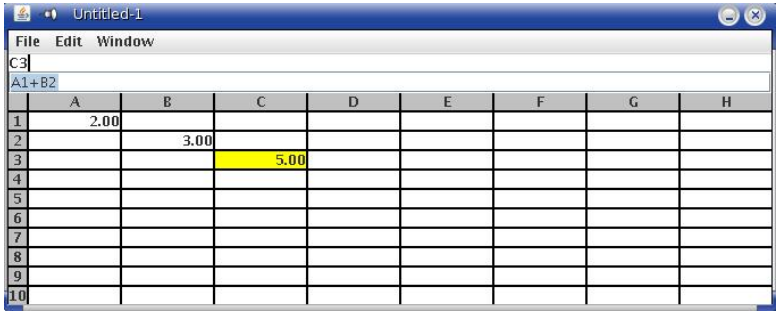
Namngivning av metoder

- Metodnamn inleds med en gemen och följande ord i sammansatta namn med versal.
- Metoder som förändrar objektets tillstånd bör ha ett verb som namn.
- Metoder som inte förändrar objektets tillstånd bör ha ett substantiv som namn eller inledas med `is` om de implementerar ett predikat.
- Namn som inleds med `set` och `get` bör reserveras för de tillfällen då det handlar om att sätta och hämta attribut.

Namngivning av attribut, lokala variabler och parametrar

- Ett attribut kan ofta ha samma namn som sin typ, fast med inledande gemen.
- Om det finns flera attribut av samma typ som används på samma sätt kan man numrera dem: `Expr expr1, expr2;`
- När en variabel används som index i en "array" använder man gärna konventionella namn som `i` och `j`.
- För strängar använder man gärna namn som indikerar användningen: `String title;`
- Använd inte `the` som ett prefix i attributnamn. När det behövs är `this.` det självklara alternativet.

Projekt 2 – XL



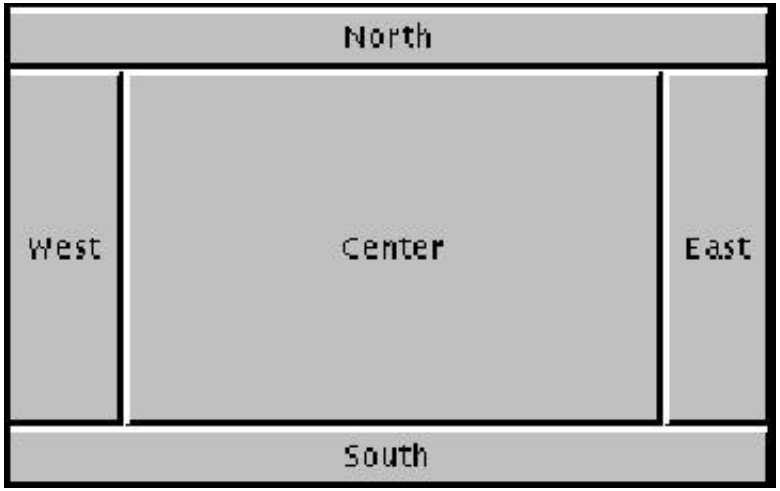
XL – Gui

JFrame ger ett eget fönster på skärmen.

```
import javax.swing.JFrame;
import java.awt.BorderLayout;

public class Gui extends JFrame {
    public Gui(int count) {
        super("Untitled-" + count);
        // omissions
        pack();
        setVisible(true);
    }
}
```


BorderLayout



JPanel

statusPanel

C3 /home/lennarta/omd/workspace/assignment2/s.xl processed

editor

A1+B2

storagePanel

	A	B	C	D	E	F	G	H
1	2.00							
2		3.00						
3			5.00					
4								
5								
6								
7								
8								
9								
10								

BorderLayout

```
import java.awt.BorderLayout;
import javax.swing.JPanel;

public class Gui extends JFrame {

    public Gui(int count) {
        ...
        setLayout(new BorderLayout());
        JPanel statusPanel = new BorderPanel();
        JPanel storagePanel = new BorderPanel();
        ...
        Editor editor = new Editor(this);
        add(BorderLayout.NORTH, statusPanel);
        add(BorderLayout.CENTER, editor);
        add(BorderLayout.SOUTH, storagePanel);
        ...
    }
}
```


BorderPanel

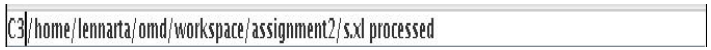
```
import java.awt.BorderLayout;
import java.awt.Color;
import javax.swing.JPanel;

public class BorderPanel extends JPanel {
    public BorderPanel() {
        super(new BorderLayout(2, 2));
        setBackground(Color.BLACK);
    }
}
```


statusPanel

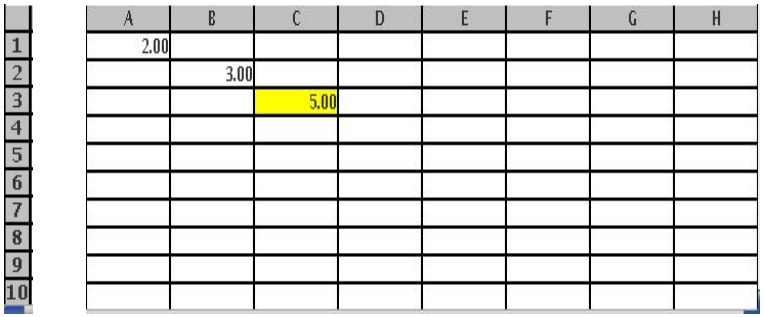
```
JPanel statusPanel = new BorderPanel();
CurrentView currentView = new CurrentView();
StatusArea statusArea = new StatusArea();
statusPanel.add(BorderLayout.WEST, currentView);
statusPanel.add(BorderLayout.CENTER, statusArea);
```

statusArea




```
public class CurrentView extends JLabel {
    public CurrentView() {
        super("A1");
        setBackground(Color.WHITE);
        setOpaque(true);
    }
}
```


storagePanel



GridLayout(11, 1, 2, 2)

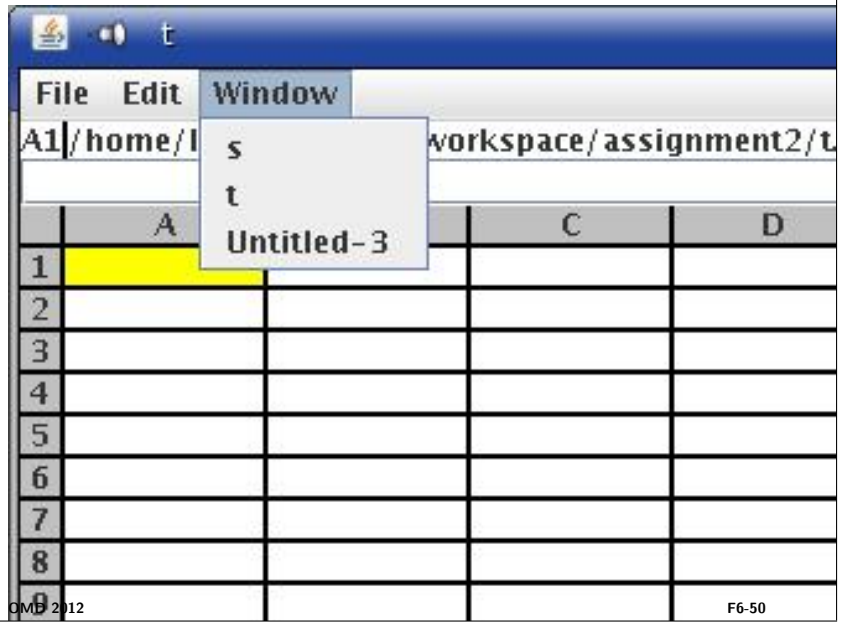
GridLayout(11, 8, 2, 2)

JTextField

JTextField används för textinmatning.

```
public class Editor extends JTextField
                    implements ActionListener {
    public Editor() {
        setBackground(Color.WHITE);
        addActionListener(this);
    }
    public void actionPerformed(ActionEvent event) {
        // called by Return key
        // contents returned by getText()
    }
}
```


WindowMenu



JMenuItem

```
class WindowMenuItem extends JMenuItem
                     implements ActionListener {
    private Gui gui;

    public WindowMenuItem(Gui gui) {
        super(gui.getTitle());
        this.gui = gui;
        addActionListener(this);
    }

    public void actionPerformed(ActionEvent event) {
        gui.toFront();
    }
}
```


JMenu

```
public class WindowMenu extends JMenu implements Observer {
    private GuiList guiList;

    public WindowMenu(GuiList guiList) {
        super("Window");
        this.guiList = guiList;
        guiList.addObserver(this);
        update(null, null);
    }

    public void update(Observable observable, Object object) {
        removeAll();
        for (Gui gui : guiList) {
            add(new WindowMenuItem(gui));
        }
    }
}
```


GuiList

```
public class GuiList extends Observable
                     implements Iterable<Gui> {
    private List<Gui> list = new ArrayList<Gui>();

    public void add(Gui gui) {
        list.add(gui);
        setChanged();
        notifyObservers();
    }

    public Iterator<Gui> iterator() {
        return list.iterator();
    }
    //omissions
}
```


MouseListener

Några swing-komponenter kan ha en `ActionListener`, t ex: JButton, JMenuItem, JTextField.

Alla komponenter kan ha en `MouseListener`. Den läggs till med

```
public void addMouseListener(MouseListener listener);
```

```
public interface MouseListener {
    void mouseClicked(MouseEvent event);
    void mouseEntered(MouseEvent event);
    void mouseExited(MouseEvent event);
    void mousePressed(MouseEvent event);
    void mouseReleased(MouseEvent event);
}
```

MouseAdapter

I regel vill man bara reagera på en av händelserna. Då är det bekvämt med

```
public abstract class MouseAdapter {
    public void mouseClicked(MouseEvent event){}
    public void mouseEntered(MouseEvent event){}
    public void mouseExited(MouseEvent event){}
    public void mousePressed(MouseEvent event){}
    public void mouseReleased(MouseEvent event){}
}
```


MouseListenerLabel

```
public class MouseListenerLabel extends JLabel {
    private class ClickListener extends MouseAdapter {
        public void mouseClicked(MouseEvent event) {
            setBackground(Color.YELLOW);
        }
    }
    public MouseListenerLabel() {
        setBackground(Color.WHITE);
        addMouseListener(new ClickListener());
    }
}
```