Objektorienterad modellering och diskreta strukturer / design

Inför nästa projekt

Lennart Andersson

Reviderad 2011–09–15

2011

Dagens program

- Felhantering
- Syntaxanalys
- Tillståndsdiagram
- Introduktion av projekt 2
- Swing-komponenter och GUI-design

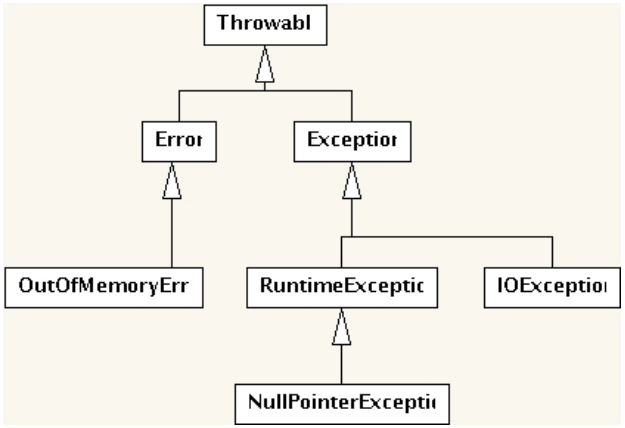
Vecka 4-6

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

Vecka 7

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

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

```
class Div extends BinExpr {
  Div(Expr expr1, Expr expr2) {
    super(expr1, expr2);
  }
  public 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;
}
```

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));
        }
    }
}
```


Instruktionsfabriken ...

```
public class InstructionFactory {
  private OperandFactory opFactory = 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 = opFactory.build(tokenizer.nextToken());
      op2 = opFactory.build(tokenizer.nextToken());
      op3 = opFactory.build(tokenizer.nextToken());
      return new Add(op1, op2, (Address) op3);
    } ...
```

... Instruktionsfabriken ...

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


... Instruktionsfabriken

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


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 ::= '*' | '/'
```

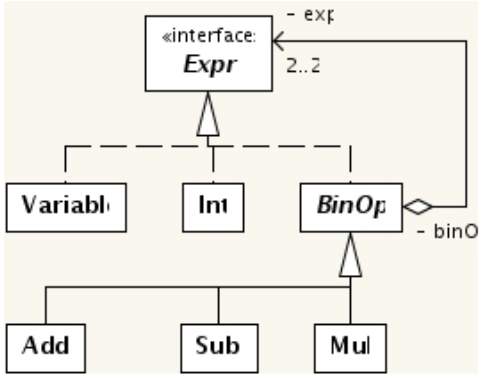
Operatorer

- `*` 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 = -3;

  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();
    }
```

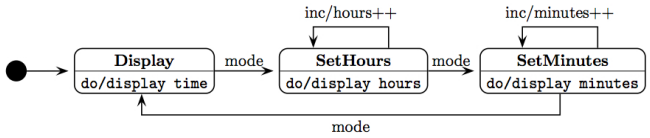
Analys av faktorer

```
private Expr factor() throws IOException {
  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);
  }
}
```

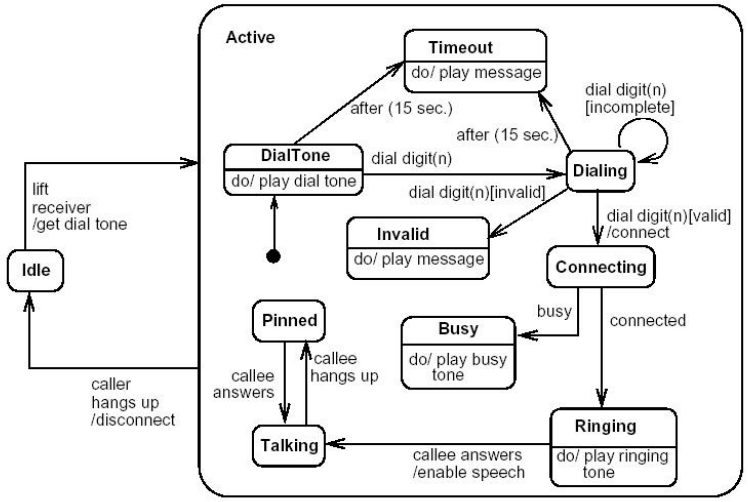

Analys av termer

```
private Expr term() throws IOException {
    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;
}
```

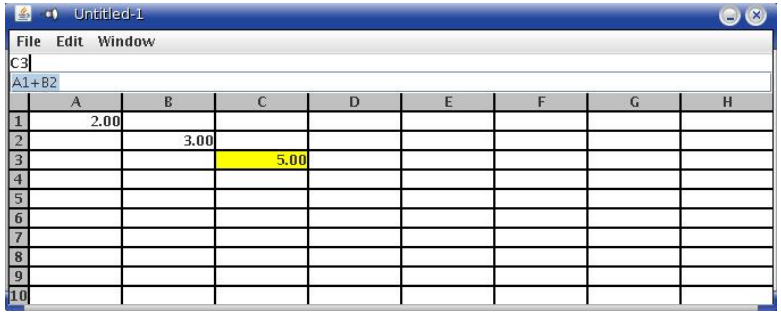

Tillståndsdiagram



En telefon



Projekt 2 – XL



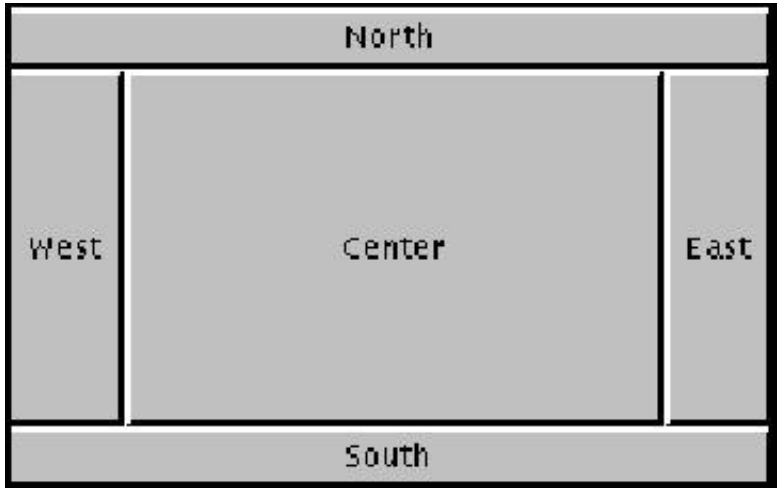
XL – Gui

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

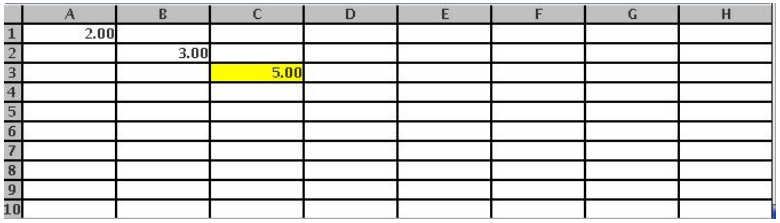


JPanel

statusPanel

editor

storagePanel



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);
```

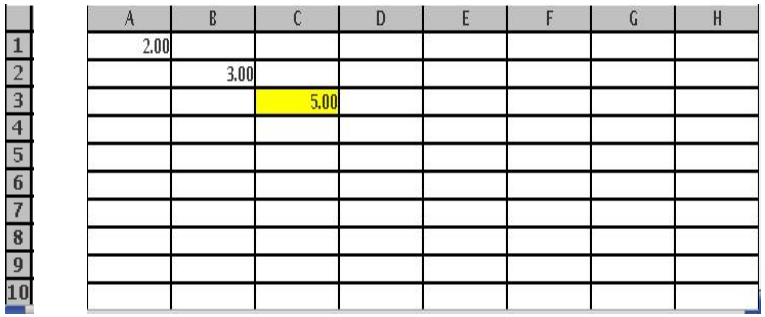

statusArea

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

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

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

storagePanel



GridLayout(11, 1, 2, 2)

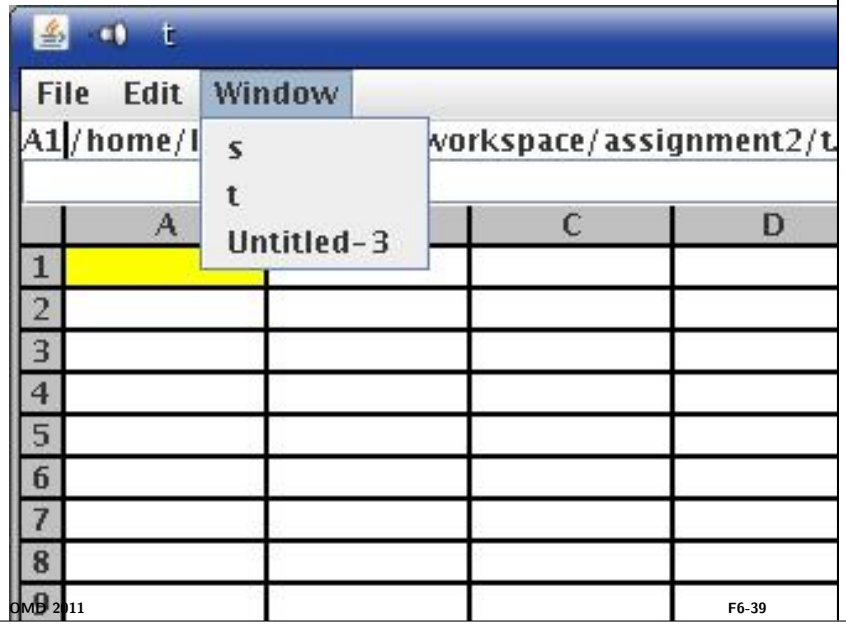
GridLayout(11, 8, 2, 2)

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());
    }
}
```