

# **Föreläsning 10**

**Mer om grafiska komponenter  
Händelsestyrda program**

# Layout Managers

Utplaceringen av komponenter i en behållare styrs med en *layout manager*.

Det finns olika layout managers:

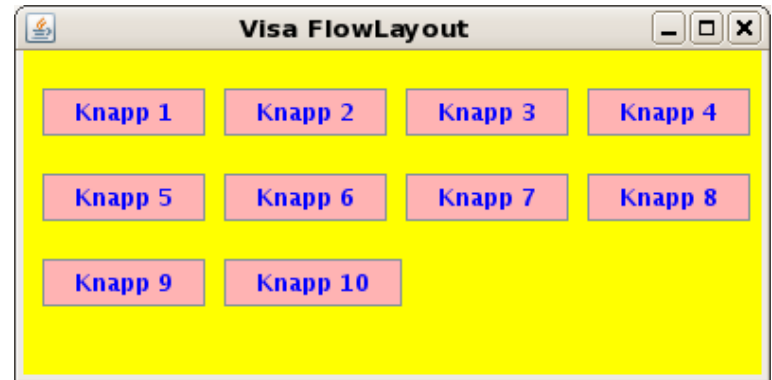
- FlowLayout
- GridLayout
- BorderLayout
- CardLayout
- GridBagLayout

# FlowLayout

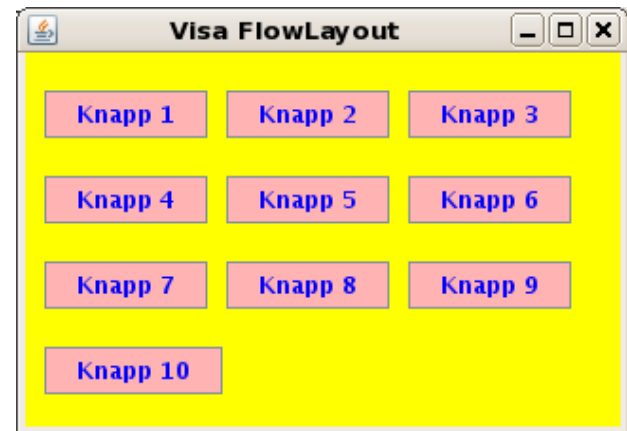
Den enklaste (och kanske minst användbara) layout managern är FlowLayout.

Komponenterna placeras ut radvis.  
Får inte en komponent plats på en rad  
påbörjas automatiskt nästa rad.

Man styra huruvida raderna skall  
fyllas på från vänster eller höger. Det  
går även att styra avståndet mellan  
komponenterna i x- respektive y-led.



Komponenterna förflyttas då  
storleken på fönstret förändras.



# Exempel på FlowLayout

```
import java.awt.*;
import javax.swing.*;
public class ShowFlowLayout extends JFrame {
    private JButton[] knappar = new JButton[10];
    public ShowFlowLayout() {
        getContentPane().setBackground(Color.YELLOW);
        setTitle("Visa FlowLayout");
        setLayout(new FlowLayout(FlowLayout.LEFT,10,20));
        for (int i = 0; i < knappar.length; i = i + 1) {
            knappar[i] = new JButton("Knapp " + (i+1));
            knappar[i].setForeground(Color.BLUE);
            knappar[i].setBackground(Color.PINK);
            add(knappar[i]);
        }
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    } //constructor
```

```
public static void main(String[] args) {
    JFrame f = new ShowFlowLayout();
    f.setSize(400,200);
    f.setVisible(true);
} //main
} //ShowFlowLayout
```

# Exempel på FlowLayout

Använda metoder:

|                            |                                             |
|----------------------------|---------------------------------------------|
| <code>add</code>           | placeras ut komponenter i behållaren        |
| <code>setBackground</code> | sätter bakgrundsfärg                        |
| <code>setForeground</code> | sätter förgrundsfärg                        |
| <code>setTitle</code>      | sätter titeln i fönsterramen                |
| <code>setSize</code>       | sätter storleken                            |
| <code>setVisible</code>    | anger om komponenten skall visas eller inte |

Ett objekt av klassen **JFrame** har flera olika delar, t.ex. en menyrad, fönstrets kant och huvudytan. Huvudytan, som används för att lägga olika komponenter i, kommer man åt med metoden `getContentPane()`.

Satsen

`setDefaultCloseOperation(EXIT_ON_CLOSE)`

gör det möjligt att avsluta programmet genom att stänga fönstret via stängningsrutan i fönstrets ram.

# GridLayout

När **GridLayout** används delas behållaren in i ett antal rader och kolumner, i vilka komponenterna placeras ut radvis.

Utplaceringen sker från vänster till höger. Det är möjligt att styra avståndet mellan komponenterna i x- respektive y-led.



Komponenterna förflyttas INTE då storleken på fönstret förändras, utan storleken på komponenterna anpassas efter fönstrets storlek.



# Exempel på GridLayout

```
import java.awt.*;
import javax.swing.*;
public class ShowGridLayout extends JFrame {
    private JButton[] knappar = new JButton[10];
    public ShowGridLayout() {
        getContentPane().setBackground(Color.YELLOW);
        setTitle("Visa GridLayout");
        setLayout(new GridLayout(4, 3, 10, 20));
        for (int i = 0; i < knappar.length; i = i + 1) {
            knappar[i] = new JButton("Knapp " + (i+1));
            knappar[i].setForeground(Color.BLUE);
            knappar[i].setBackground(Color.PINK);
            add(knappar[i]);
        }
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    } //constructor
```

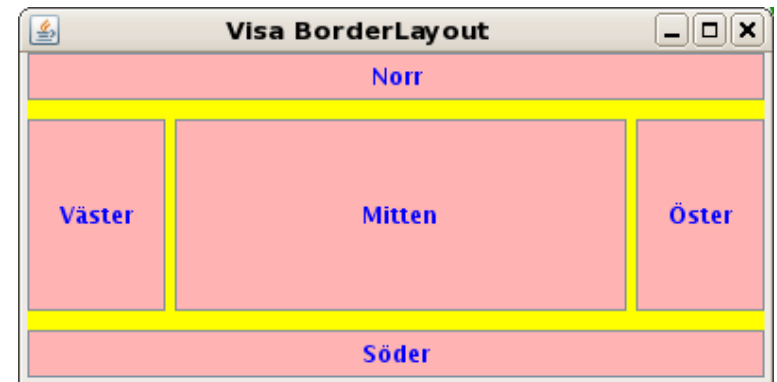
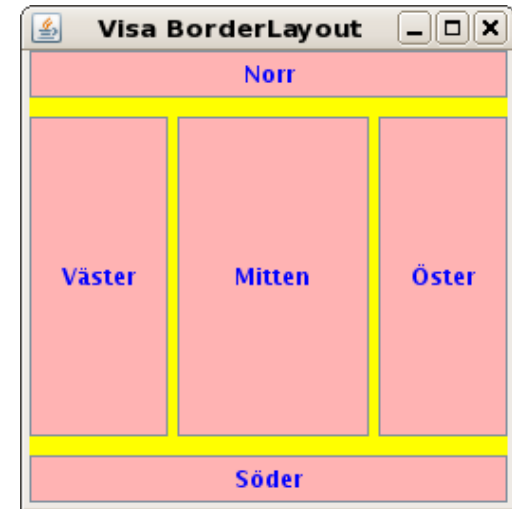
```
public static void main(String[] args) {
    JFrame f = new ShowGridLayout();
    f.setSize(400,200);
    f.setVisible(true);
} //main
} //ShowGridLayout
```

# BorderLayout

När **BorderLayout** används delas behållaren in i fem delar. Dessa delar kallas **North**, **South**, **West**, **East** och **Center**. Vid utplacering av en komponent anges var komponenten skall placeras.

Det är möjligt att styra avståndet mellan komponenterna i x- respektive y-led.

När storleken på fönstret förändras får platsen **Center** det utrymme som "blir över".



# Exempel på BorderLayout

```
import java.awt.*;
import javax.swing.*;
public class ShowBorderLayout extends JFrame {
    private JButton east = new JButton("Öster");
    private JButton south = new JButton("Söder");
    private JButton west = new JButton("Väster");
    private JButton north = new JButton("Norr");
    private JButton center = new JButton("Mitten");
    public ShowBorderLayout() {
        getContentPane().setBackground(Color.YELLOW);
        east.setForeground(Color.BLUE); east.setBackground(Color.PINK);
        south.setForeground(Color.BLUE); south.setBackground(Color.PINK);
        west.setForeground(Color.BLUE); west.setBackground(Color.PINK);
        north.setForeground(Color.BLUE); north.setBackground(Color.PINK);
        center.setForeground(Color.BLUE); center.setBackground(Color.PINK);
        setTitle("Visa BorderLayout");
        setLayout(new BorderLayout(5, 10));
        add("East", east); add("South", south); add("West", west);
        add("North", north); add("Center", center);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    } //constructor
```

```
public static void main(String [] args) {
    JFrame f = new ShowBorderLayout();
    f.setSize(400,200);
    f.setVisible(true);
} //main
} //ShowBorderLayout
```

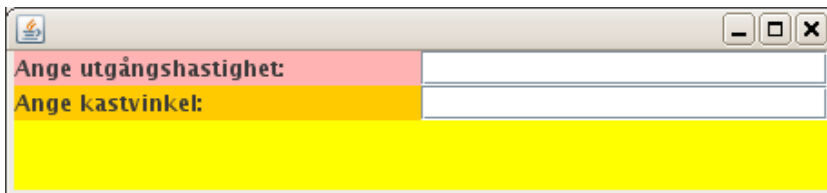
# Exempel från förra föreläsningen:

```
import javax.swing.*;
import java.awt.*;
public class Shoot1 extends JFrame {
    private JTextField hField = new JTextField(20);
    private JTextField vField = new JTextField(20);
    private JLabel resultLabel = new JLabel();
    public Shoot1() {
        JLabel hLabel = new JLabel("Ange utgångshastighet: ");
        hLabel.setBackground(Color.PINK);
        hLabel.setOpaque(true);
        JLabel vLabel = new JLabel("Ange kastvinkel: ");
        vLabel.setBackground(Color.ORANGE);
        vLabel.setOpaque(true);
        JPanel inputPanel = new JPanel();
        inputPanel.setLayout(new GridLayout(2,2));
        inputPanel.add(hLabel);
        inputPanel.add(hField);
        inputPanel.add(vLabel);
        inputPanel.add(vField);
```

```
import javax.swing.*;
public class MainShoot1 {
    public static void main(String[] args) {
        JFrame w = new Shoot1();
    } //main
} //MainShoot1
```

Observera att programmet  
inte är komplett, eftersom vi  
ännu inte har några lyssnare  
i programmet!

```
resultLabel.setBackground(Color.YELLOW);
resultLabel.setForeground(Color.MAGENTA);
resultLabel.setOpaque(true);
setLayout(new GridLayout(2,1));
add(inputPanel);
add(resultLabel);
pack();
setVisible(true);
setDefaultCloseOperation(EXIT_ON_CLOSE);
} //constructor
} //Shoot1
```



# Händelsehantering

Händelsestyrda program brukar bestå av två faser:

- Först genomlöps initieringsfasen. I den initierar man de olika grafiska komponenterna som skall användas i programmet.

Man definierar också de sk *callback-funktioner* som skall ta hand om de händelser man är intresserad av.

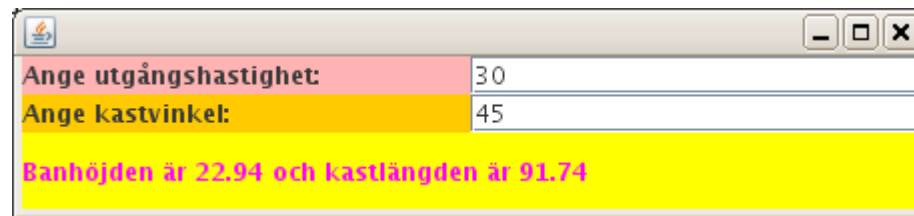
I Java definieras callback-funktionerna som metoder i speciella *lyssnare*.

- När initieringsfasen är klar går programmet in i väntefasen.

I väntefasen ligger programmet och väntar på att en yttre händelse skall inträffa. Händelsen tas om hand av lyssnaren och den callback-funktion som är kopplad till händelsen utförs.

# Händelsehantering

Det önskade beteendet i vårt program är att om vi i någon av inmatningsrutorna ändrar datavärde skall programmet beräkna och skriva ut nya utdatavärden från de aktuella indatavärdena genom att vi i någon av inmatningsrutorna trycker på sändningstangenten.



A screenshot of a Java Swing window with a title bar containing a small icon and standard window controls (minimize, maximize, close). The window contains two input fields with labels on the left and text boxes on the right. The first row has a pink background for the label 'Ange utgångshastighet:' and the value '30'. The second row has an orange background for the label 'Ange kastvinkel:' and the value '45'. Below these is a large yellow text area containing the text 'Banhöjden är 22.94 och kastlängden är 91.74' in pink font.

|                                             |    |
|---------------------------------------------|----|
| Ange utgångshastighet:                      | 30 |
| Ange kastvinkel:                            | 45 |
| Banhöjden är 22.94 och kastlängden är 91.74 |    |

Vi måste således förse inmatningsrutorna med en lyssnare och skriva en *callback-funktion* som utför de beräkningar vi vill göra.

# ActionEvent och ActionListener

När vi trycker på sändningstangenten i en inmatningsruta (JTextField) eller med musknappen på en knapp (JButton) inträffar en händelse av klassen **ActionEvent**.

De lyssnare som lyssnar på händelser av klassen **ActionEvent** är av klassen **ActionListener**.

Det finns många olika händelsetyper i Java, och varje händelsetyp har sin egen typ av lyssnare. Några av de vanligaste händelsetyperna är:

|                |                                                                                                                             |
|----------------|-----------------------------------------------------------------------------------------------------------------------------|
| ActionEvent    | Vanliga händelser, som t ex att någon har tryckt på en knapp (JButton)                                                      |
| MouseEvent     | Händelser med musen, som t ex att en musknapp har tryckts ned eller släppts                                                 |
| MouseEvent     | När muspekaren har rört på sig                                                                                              |
| KeyEvent       | Tangentbordshändelser. Man kan lyssna på när tangenter trycks ned och släpps upp samt när de faktiskt producerar ett tecken |
| ComponentEvent | Förändringar av en komponent, t ex att dess storlek har ändrats (ofta som följd av att hela fönstrets storlek har ändrats)  |
| WindowEvent    | Händelser för ett fönster, som t ex att det är på väg att stängas eller har ikonifierats                                    |

# Hur använder man lyssnare?

Följande tre steg måste göras för att kunna lyssna på, fånga upp och agera på en händelse av typen `ActionEvent`:

1. Skriv en `actionPerformed`-metod i huvudklassen. Detta görs genom att lägga till metoden

```
public void actionPerformed(ActionEvent e)
```

i klassen: Det är denna metod som exekveras när händelsen inträffar.

2. Ange att huvudklassen har en lyssnare, dvs att klassen innehåller metoden `actionPerformed`. Detta görs genom att ange att klassen implementerar interfacet `ActionListener`

```
public class Klass extends JFrame implements ActionListener
```

3. Registrera att huvudklassen skall lyssna efter `ActionEvent`-händelser på den aktuella komponenten:

```
komponent.addActionListener(this)
```

Till `actionPerformed` skickas ett objekt av klassen `ActionEvent`. Detta objekt kan man använda till att ta reda på för vilken komponent händelsen inträffat. Genom att anropa metoden

```
e.getSource( )
```

fås en referens till komponenten där händelsen `e` inträffat.

# Komplettering av vårt tidigare exempel

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;

public class Shoot extends JFrame implements ActionListener {
    private JTextField hField = new JTextField(20);
    private JTextField vField = new JTextField(20);
    private JLabel resultLabel = new JLabel();
    public Shoot() {
        JLabel hLabel = new JLabel("Ange utgångshastighet: ");
        hLabel.setBackground(Color.PINK);
        hLabel.setOpaque(true);
        JLabel vLabel = new JLabel("Ange kastvinkel: ");
        vLabel.setBackground(Color.ORANGE);
        vLabel.setOpaque(true);
        hField.addActionListener(this);
        vField.addActionListener(this);
        JPanel input = new JPanel();
        input.setLayout(new GridLayout(2,2));
        input.add(hLabel);
        input.add(hField);
        input.add(vLabel);
        input.add(vField);
```

```
        resultLabel.setBackground(Color.YELLOW);
        resultLabel.setForeground(Color.MAGENTA);
        resultLabel.setOpaque(true);
        setLayout(new GridLayout(2,1));
        add(input);
        add(resultLabel);
        pack();
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    } //constructor
```



Registrera lyssnare  
på komponenterna

# Fortsättning: lyssnare och callback-funktion

```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource() == hField || e.getSource() == vField)  
        calculate();  
}  
//actionPerformed  
  
public void calculate( ) {  
    final double G = 9.81;  
    String indata = hField.getText();  
    double v = Double.parseDouble(indata);  
    indata = vField.getText();  
    double alfa = Double.parseDouble(indata);  
    alfa = Math.toRadians(alfa);  
    double h = (Math.pow(v, 2) * Math.pow(Math.sin(alfa), 2))/(2*G);  
    double d = (Math.pow(v, 2) * Math.sin(2*alfa))/G;  
    NumberFormat r = NumberFormat.getInstance();  
    r.setMaximumFractionDigits(2);  
    r.setMinimumFractionDigits(2);  
    resultLabel.setText("Banhöjden är " + r.format(h) + "\n"  
                        + " och kastlängden är " + r.format(d));  
}  
//calculate  
} //Shoot
```

```
import javax.swing.*;  
public class MainShoot {  
    public static void main(String[] args) {  
        JFrame w = new Shoot();  
    }  
    //main  
} //MainShoot
```

# Model-View-Controller (MVC)

MVC-arkitekturen är en allmänt accepterad princip för program som använder grafiska gränssnitt. MVC-arkitekturen separerar de tre rollerna:

**Model** (modell) den bakomliggande datastruktur som programmet bearbetar.

**View** (vy) den visuella presentationen av modellen som användaren ser.

**Controller** (styrfunktion) den klass som tar emot indata från användaren och utifrån detta först ber modellen och sedan vyn att uppdatera sig.

# Model

```
public class Model {  
    private double angle;  
    private double speed;  
    private static final double G = 9.81;  
    public void setAngle(double angle) {  
        this.angle = angle;  
    }  
    public void setSpeed(double speed) {  
        this.speed = speed;  
    }  
    public double getHeight() {  
        return (Math.pow(speed, 2) * Math.pow(Math.sin(Math.toRadians(angle)), 2)) / (2*G);  
    }  
    public double getDistance() {  
        return (Math.pow(speed, 2) * Math.sin(2*Math.toRadians(angle))) / G;  
    }  
} //Model
```

# View

```
import javax.swing.*;
import java.awt.*;
import java.text.*;
public class GraphicalView extends JPanel {
    private JLabel resultatLabel = new JLabel();
    private Model model;

    public GraphicalView(Model model) {
        this.model = model;
        resultatLabel.setBackground(Color.YELLOW);
        resultatLabel.setForeground(Color.MAGENTA);
        resultatLabel.setOpaque(true);
        add(resultatLabel);
        setBackground(Color.YELLOW);
    } // constructor

    public void showResult() {
        NumberFormat r = NumberFormat.getInstance();
        r.setMaximumFractionDigits(2);
        r.setMinimumFractionDigits(2);
        resultatLabel.setText("Banhöjden är " + r.format(model.getHeight()) + "\n"
                               + " och kastlängden är " + r.format(model.getDistance()));
    } // showResult
} // GraphicalView
```

# Controller

```
import javax.swing.*; import java.awt.*; import java.awt.event.*;
public class GraphicalController extends JPanel implements ActionListener {
    private JTextField hField = new JTextField(20);
    private JTextField vField = new JTextField(20);
    private Model model;
    private GraphicalView view;
    public GraphicalController(Model model, GraphicalView view) {
        this.model = model;
        this.view = view;
        JLabel hLabel = new JLabel("Ange utgångshastighet: ");
        hLabel.setBackground(Color.PINK);
        hLabel.setOpaque(true);
        JLabel vLabel = new JLabel("Ange kastvinkel: ");
        vLabel.setBackground(Color.ORANGE);
        vLabel.setOpaque(true);
        hField.addActionListener(this);
        vField.addActionListener(this);
        setLayout(new GridLayout(2,2));
        add(hLabel);
        add(hField);
        add(vLabel);
        add(vField);
    } //constructor

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == hField)
            model.setSpeed(Double.parseDouble(hField.getText()));
        else if (e.getSource() == vField)
            model.setAngle(Double.parseDouble(vField.getText()));
        view.showResult();
    } //actionPerformed
} //GraphicalController
```

# Sammankoppling av komponenterna

```
import javax.swing.*;
import java.awt.*;
public class GraphicalWindow extends JFrame {
    public GraphicalWindow() {
        Model model = new Model();
        GraphicalView view = new GraphicalView(model);
        GraphicalController controller = new GraphicalController(model, view);
        setLayout(new GridLayout(2,1));
        add(controller);
        add(view);
        pack();
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        }//konstruktor
    } //GraphicGraphical
```

```
import javax.swing.*;
public class MainMVC {
    public static void main(String[] args) {
        JFrame w = new GraphicalWindow();
    }//main
} //MainMVC
```

# Ett exempel till

Antag att vi har en klass `GraphicDie` som definierar en grafisk tärning. I klassen finns bl.a följande metoder:

`GraphicDie()`

konstruktör som skapar en "tom" tärning

`GraphicDie(int dots)`

konstruktör som sätter tärningens värde till dots

`roll()`

kastar tärningen

`int getValue()`

avläser värdet på tärningen

`setEmpty()`

sätter tärningen till "tom"

Vi vill skriva ett program där vi upprepade gånger kan kasta tärningen och kunna se tärningens värde dels grafiskt, dels som text samt hur många kast som gjorts och den sammanlagda summan av dessa kast (enligt figuren nedan).



## Implementation:

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class TestDie extends JFrame implements ActionListener {
    private JButton tossButton = new JButton("KASTA");
    private GraphicDie theDie = new GraphicDie();
    private JTextArea textLabel = new JTextArea();
    private int nrOfThrows = 0;
    private int sum = 0;

    public TestDie() {
        setLayout(new GridLayout(3,1));
        add(theDie);
        add(textLabel);
        textLabel.setBackground(Color.WHITE);
        textLabel.setEnabled(false);
        tossButton.addActionListener(this);
        add(tossButton);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    } //konstruktor
```

```
        public void actionPerformed(ActionEvent e) {
            if (e.getSource() == tossButton) {
                theDie.roll();
                theDie.repaint();
                nrOfThrows = nrOfThrows + 1;
                int dots = theDie.getValue();
                sum = sum + dots;
                textLabel.setText("Poäng: " + dots
                                + "\nAntal kast: " + nrOfThrows
                                + "\nTotal summa: " + sum);
            }
        } //actionPerformed

    public static void main(String [] args) {
        TestDie td = new TestDie();
        td.setSize(300, 300);
        td.setVisible(true);
    } //main
} //TestDie
```

## MVC-implementation: Model

```
public class DieModel {  
    private int value;  
    private int sum;  
    private int nrOfThrows;  
    public DieModel() {  
        value = 0;  
        sum = 0;  
        nrOfThrows = 0;  
    }  
    public int getValue() {  
        return value;  
    }  
    public int getSum() {  
        return sum;  
    }  
    public void update(int value) {  
        this.value = value;  
        sum = sum + value;  
        nrOfThrows = nrOfThrows + 1;  
    }  
    public int getNrOfThrows() {  
        return nrOfThrows;  
    }  
} //DieModel
```

## MVC-implementation: View

```
import javax.swing.*;
import java.awt.*;
public class DieView extends JTextArea {
    private DieModel model;
    public DieView(DieModel model) {
        this.model = model;
        setEnabled(false);
        setFont(new Font("SansSerif", Font.BOLD,18));
    }

    public void update() {
        setText("Tärningspoängen: " + model.getValue()
            + "\n Antal kast: " + model.getNrOfThrows()
            + "\nTotal summa: " + model.getSum());
    }
} //DieView
```

## MVC-implementation: Controller

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class DieController extends JButton implements ActionListener {
    private GraphicDie theDie;
    private DieModel model;
    private DieView view;

    public DieController(DieModel model, DieView view, GraphicDie theDie) {
        this.model = model;
        this.view = view;
        this.theDie = theDie;
        addActionListener(this);
        setText("KASTA");
    } //constructor

    public void actionPerformed(ActionEvent e) {
        theDie.roll();
        theDie.repaint();
        model.update(theDie.getValue());
        view.update();
    } //actionPerformed
} //DieController
```

## MVC-implementation: Sammankoppling av komponenterna

```
import javax.swing.*;
import java.awt.*;
public class TestDieWindow extends JFrame {
    public TestDieWindow() {
        DieModel model = new DieModel();
        DieView view = new DieView(model);
        GraphicDie theDie = new GraphicDie();
        DieController controller = new DieController(model, view, theDie);
        setLayout(new GridLayout(3,1));
        add(theDie);
        add(view);
        add(controller);
        setSize(300, 300);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    } //constructor

    public static void main(String[] args) {
        JFrame w = new TestDieWindow();
    } //main
} //TestDieWindow
```

# Allmänna tips

Swing är stort, och det är svårt (och onödigt) att hålla alla detaljer i minnet angående vilka klasser och metoder som finns tillgängliga.

Det rekommenderas därför starkt att man har tillgång till en bra bok och framför allt online-dokumentation när man programmerar.