Läsanvisningar

Börja med att läsa kapitlen i kursboken, därefter resurserna på Internet och till sist själva lektionsmaterialet.

Kursboken (5:e upplagan): Kapitel 13 - 14
Kursboken (6-8:e upplagan): Kapitel 12, 14

Internet: The Java Tutorial, Trail: Creating a GUI With JFC/Swing: - Writing Event Listeners (http://docs.oracle.com/javase/tutorial/uiswing/events/index.html)

Internet: The Java Tutorial, Trail: Creating a GUI With JFC/Swing: - How to Use Menus (http://docs.oracle.com/javase/tutorial/uiswing/components/menu.html)

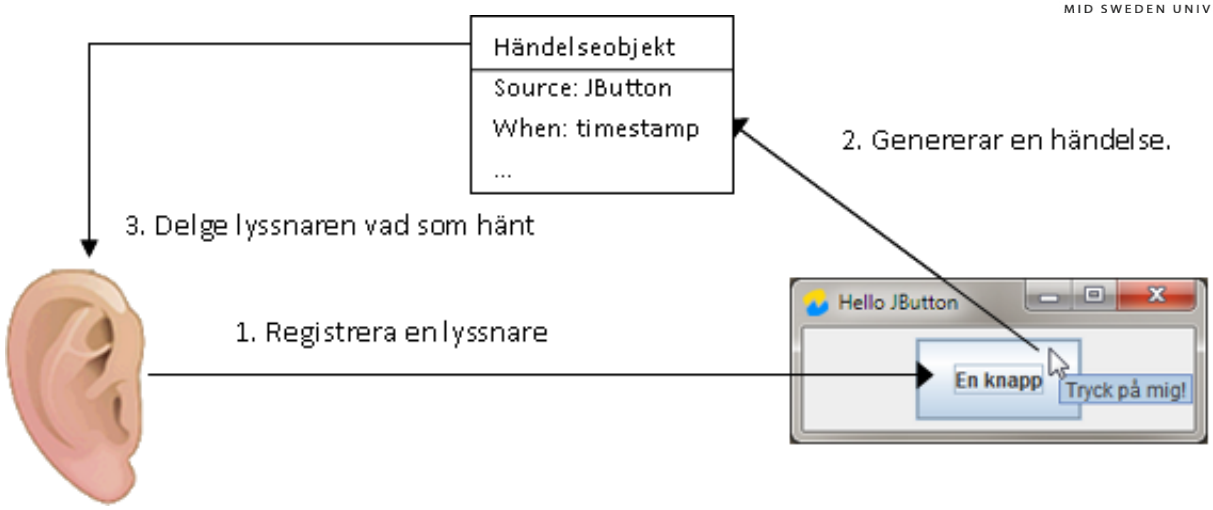
Användargränssnitt del 2

I dina tidigare program har vad som ska ske och i vilken ordning detta ska ske helt och hållet styrts efter den ordning koden skrivits. Programmen har exekverat rad för rad, med början i main, till dess att sista kodraden exekverats för att då avslutas. Ibland har programmen väntat på inmatning från användaren (via tangentbordet), men oavsett hur lång tid inmatningen tog, fortsatte exekveringen på nästa rad.

Ett program med ett grafiskt användargränssnitt beter sig annorlunda. Här är begreppet händelser centralt och mycket av programmets funktionalitet kommer av att reagera på och hantera händelser av olika slag. En händelse genereras till exempel när en av musens knappar trycks ned, en annan händelse genereras när muspekaren rör sig över fönstret, ytterligare en händelse genereras när användaren trycker på en knapp i användargränssnittet och när en tangent på tangentbordet trycks ned genereras en annan typ av händelse. Ett program med ett grafiskt användargränssnitt sägs vara händelsestyrt och måste kunna hantera de händelser som uppstår. Vi ska i denna lektion titta närmare på hur vi hanterar dessa händelser och skriver kod som ska utföras när en viss typ av händelse inträffat. Lektionen kommer även att ta upp hantering av menyer.

Javas modell för händelsehantering

Till vår hjälp för att styra vad programmet ska göra när en händelse inträffar finns det så kallade lyssnare och händelseklasser.



1. I vår kod måste vi registrera en lyssnare på komponenter som lyssnar efter en specifik typ av händelse. Lyssnarna är objekt av klasser som implementerar ett eller flera lyssnargränssnitt (interface).
2. När en komponent genererar en händelse till följd av en yttre aktivitet (till exempel att användaren trycker på en knapp) skapar den ett objekt av en händelseklass, ett så kallat händelseobjekt. Händelseobjektet innehåller information om bland annat vilken komponent som genererat händelsen samt diverse information om vad som hänt (till exempel tidpunkt när händelsen inträffade, x- och y-koordinater vid musklick med mera).
3. Komponenten delger (meddelar) sen den registrerade lyssnaren om vad som hänt genom att skicka händelseobjektet till lyssnaren.
4. Lyssnaren undersöker händelseobjektet och vidtar lämpliga åtgärder.

Lyssnargränssnitt

Dessa är några av de vanligaste lyssnargränssnitten och de som ni kan tänkas ha användning av under denna kurs:

WindowListener

Denna används när man vill styra hur och vad programmet ska göra när man utför händelser på ett fönster (JFrame eller JDialog). Här måste man implementera en rad olika metoder i sin lyssnarklass, nämligen:

windowActivated(WindowEvent e)	anropas när fönstret blir aktivt.
windowDeactivated(WindowEvent e)	anropas när fönstret inte längre är aktivt.
windowOpened(WindowEvent e)	anropas när fönstret öppnas/visas.
windowClosed(WindowEvent e)	anropas när fönstret stängs ned.
windowClosing(WindowEvent e)	anropas när man försöker stänga fönstret.
windowIconified(WindowEvent e)	anropas när fönstret minimeras.
windowDeiconified(WindowEvent e)	anropas när fönstret återställs från minimering.

Dessa anropas vid olika tillfällen. Om användaren till exempel minimerar fönstret är det metoden windowIconified som anropas.Till metoderna skickas ett objekt (ett händelseobjekt) av händelseklassen WindowEvent. Detta objekt innehåller en del information om händelsen som inträffat, bland annat det fönster som genererade händelsen. För att registrera en WindowListener används metoden:

```
addWindowListener(WindowListener listener);
```

till vilken vi som argument skickar det objekt som ska lyssna efter händelser. Om vi valt att aktuell klass själv ska lyssna efter händelserna kan vi skriva följande:

```
addWindowListener(this);
```

Annars måste man referera till klassen som skall hantera det.

ItemListener

Denna används när man vill reagera på att en komponent (subklasser till AbstractButton) markeras eller avmarkeras. Här måste man implementera följande metod:

itemStateChanged(ItemEvent e)	anropas när en komponent har blivit markerad eller avmarkerad.

ItemEvent är händelseobjektet som skickas till metoden. ItemEvent innehåller bland annat information om vilken komponent som genererade händelsen och om komponenten i fråga markerades eller avmarkerades.

ChangeListener

Denna används när man vill lyssna efter förändringar hos komponenter. Exempelvis en radioknapp eller checkbox ändrar de val som är markerade eller icke markerade. Här måste man implementera följande metod:

stateChanged(ChangeEvent e)	anropas när en komponent ändrar status (t.ex. ett val på en radioknapp, en checkbox osv).

ChangeEvent är objektet som skickas till metoden och innehåller endast information om vilken komponent som genererade händelsen. Det är inte möjligt att undersöka vad som har ändrats hos komponenten utan en ChangeListener används endast om vi behöver veta när en komponent förändrats på något vis, men strutar i vad som ändrats.

MouseListener

Denna används när man vill lyssna efter olika mushändelser på en komponent (finns för alla komponenter i Swing som ärver JComponent). Här måste man implementera följande fem metoder:

mouseClicked(MouseEvent e)	anropas när musens knapp blivit nedtryckt och uppsläppt igen.
mouseEntered(MouseEvent e)	anropas när musen befinner sig över/ovanpå en komponent.
mouseExited(MouseEvent e)	anropas när musen lämnar en komponent.
mousePressed(MouseEvent e)	anropas när musens knapp blivit nedtryckt.
mouseReleased(MouseEvent e)	anropas när musens knapp släppts upp.

MouseEvent är objektet som skickas till metoden. MouseEvent innehåller bland annat information om vilken komponent som genererat händelsen, en tidsstämpel för när händelsen inträffade, vilken

musknapp som användes samt x- och y-koordinater för muspekarens position.

MouseMotionListener

Denna används för att lyssna efter händelser som sker i en komponent när musen flyttas eller dras (flyttas med någon musknapp nedtryckt). Kan användas för alla komponenter i Swing som ärver JComponent. Här måste man implementera följande två metoder:

mouseDragged(MouseEvent e)	anropas när en musknapp tryckts ned på en komponent och man drar musen på komponenten.
mouseMoved(MouseEvent e)	anropas när muspekaren befinner sig över en komponent men musknappen inte tryckts ned.

Samma MouseEvent som i MouseListener.

Förutom att implementera interface går det också att använda sig av redan implementerade klasser som hanterar dessa olika händelser. Exempelvis finns klassen MouseAdapter, som redan har implementerat gränssnitten MouseListener och MouseMotion med bara tomma metoder som inte gör någonting. Den har även implementerat MouseWheelListener. Detta gör det möjligt att hantera alla olika mushändelser via en klass.

```
class MyMouseHandler extends MouseAdapter {
  public void mouseClicked(MouseEvent e) {
    // om man trycker ned vänster musknapp.
    if (e.getButton() == MouseEvent.BUTTON1) {
      // do something
    }
  }
}
```

När man utökar MouseAdapter på detta sätt behöver man inte implementera alla metoder som tidigare var ett måste med interfacen. Utan man implementerar bara de metoder som är av intresse för en. För att sedan koppla denna nya klass till en komponent görs följande:

```
JPanel jp = new JPanel();
MyMouseHandler myHandler = new MyMouseHandler();
jp.addMouseListener(myHandler);
```

KeyListener

Genereras av att användaren trycker på en tangent på tangentbordet. Kan användas i alla komponenter i Swing som ärver JComponent. Här måste man implementera följande tre metoder:

keyPressed(KeyEvent e)	anropas när en tangent har tryckts.
keyReleased(KeyEvent e)	anropas när en tangent har släppts upp.
keyTyped(KeyEvent e)	anropas när en tangent har blivit nedtryckt och släppts upp.

KeyEvent är objektet som skickas till metoden och innehåller bland annat information om den komponent varifrån händelsen genererades och vilken tangent som användes.

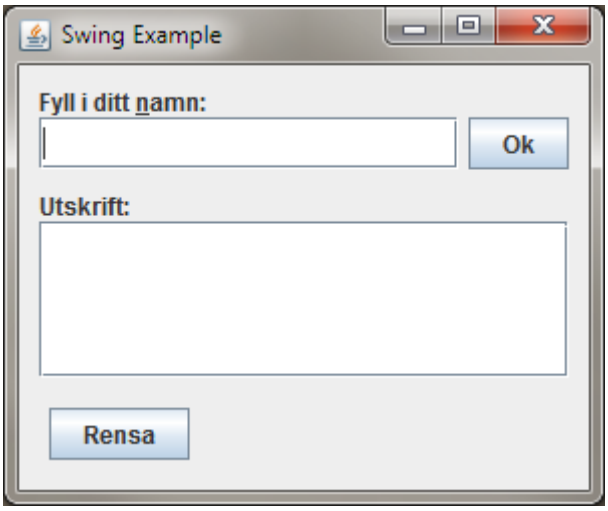
ActionListener

Denna är kanske den mest vanliga och används när man vill lyssna efter händelser som inträffat på en komponent (vanligtvis JButton och JTextField). Här måste man implementera följande metod:

actionPerformed(ActionEvent e)	anropas när en händelse inträffat.

Från det ActionEvent som genereras kan vi bland annat få reda på vilken komponent som genererade händelsen, tidpunkt för händelsen och om shift, alt eller ctrl hölls inne när händelsen genererades.

I lektion 4 tittade vi på hur JPanel kunde användas för att gruppera olika komponenter och som exempel användes en applikation med följande grafiska användargränssnitt.



Vi ska nu utöka exemplet och göra så att det namn användaren skriver in i textfältet hamnar i textrutan när användaren trycker på knappen Ok eller om användaren trycker på enter i textfältet. När användaren trycker på knappen Rensa ska all text i textrutan tas bort.

Både JButton och JTextField kan båda generera ActionEvent och därför börjar vi med att låta klassen implementera lyssnargränssnittet ActionListener (vi låter klassen själv vara lyssnare):

```
public class SwingExample extends JFrame implements ActionListener
```

I och med detta måste vi i klassen implementera lyssnarmetoden actionPerformed(ActionEvent) som lyssnargränssnittet ActionListener deklarerar.

```
// Lyssnarmetod för lyssnargränssnittet ActionListener
public void actionPerformed(ActionEvent e) {
  // kod för händelsehantering hamnar här
}
```

Nästa steg är att registrera den egna klassen som lyssnarobjekt hos de komponenter som ska generera händelser av händelseklassen ActionEvent.

```
okButton.addActionListener(this);
nameTextField.addActionListener(this);
clearButton.addActionListener(this);
```

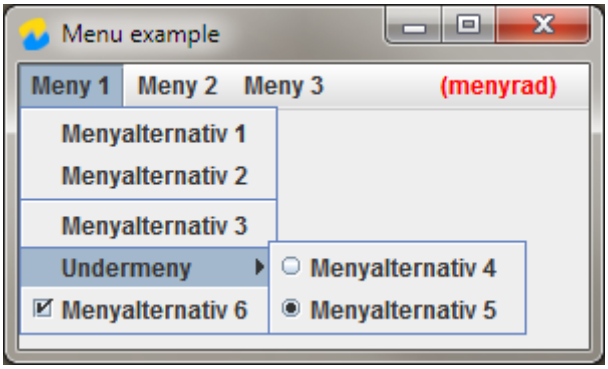
Till sist skriver vi kod i lyssnarmetoden actionPerformed som hanterar de händelser som genereras

av komponenterna. I en if-sats undersöker vi vilken komponent som genererat händelsen genom att anropa metoden getSource på händelseobjektet. Om det är Ok-knappen som genererat händelsen gör vi en extra koll om användaren höll inne shift-tangenten när knappen trycktes på. Om så är fallet rensar vi inte textfältet på den inmatade texten utan låter det stå kvar.

```
// Lyssnarmetod för lyssnargränssnittet ActionListener
public void actionPerformed(ActionEvent e) {
  // Har vi tryckt på Ok-knappen?
  if (e.getSource() == okButton) {
    // höll vi inne shift?
    boolean shift = (e.getModifiers() & ActionEvent.SHIFT_MASK) != 0;
    addNameToTextArea(shift);
  }
  else if (e.getSource() == clearButton) {
    outputTextArea.setText("");
  }
  else if (e.getSource() == nameTextField) {
    addNameToTextArea(false);
  }
}

private void addNameToTextArea(boolean keepText) {
  // Ge focus till textfältet så vi är redo för ny inmatning
  nameTextField.requestFocus();

  // Gör inget om inget namn är inmatat
  if (nameTextField.getText().length() == 0) {
    return;
  }

  // Hämta namnet
  String name = nameTextField.getText();

  // Ska vi rensa textfältet?
  if (!keepText) {
    nameTextField.setText("");
  }

  // Lägg till namnet i textrutan
  outputTextArea.append(name + "\n");
}
```

Menyer

För att lägga till en meny i ett grafiskt användargränssnitt är det framför allt tre klasser vi behöver känna till, nämligen JMenuBar, JMenu och JMenuItem. Den första klassen representerar själva menyraden i en applikation. Med klassen JMenu kan vi sen lägga till en eller flera menyer i menyraden. I varje meny placerar vi sen en eller flera JMenuItem som blir de olika menyalternativen användaren kan klicka på. En JMenu kan även innehålla andra JMenu om vi har behov av att skapa undermenyer.



Till JMenuItem finns subklasserna JCheckBoxMenuItem och JRadioButtonMenuItem som vi kan använda om vi har behov av att använda kryssrutor respektive radioknappar i en meny. I en meny kan vi även lägga till en separeringslinje för att separera två menyalternativ från varandra. Det kan vara användbart för att gruppera olika menyalternativ som hör ihop med varandra. En separeringslinje lägger vi till genom att anropa metoden addSeparator i klassen JMenu.

Vi kan faktiskt lägga till vilken komponent (JComponent) som helst i en meny. I bilden ovan har till exempel en JLabel lagts till i menyraden (för att illustrera vad som är menyraden).

För att skapa och lägga till en menyrad i ett applikationsfönster (JFrame) skriver vi:

```
JFrame frame = new JFrame("Menu example");
JMenuBar menuBar = new JMenuBar();
frame.setJMenuBar(menuBar);
```

För att skapa en meny och lägga till den i menyraden skriver vi sen:

```
JMenu menu1 = new JMenu("Meny 1");
JMenu menu2 = new JMenu("Meny 2");
JMenu menu3 = new JMenu("Meny 3");
JMenu subMenu = new JMenu("Undermeny");
menuBar.add(menu1);
menuBar.add(menu2);
menuBar.add(menu3);
```

I konstruktorn när vi skapar ett objekt av JMenu skriver vi vad som ska stå på menyn. Menyerna läggs till i menyraden i den ordning vi anropar add.

För att tillsist skapa menyalternativen och lägga till dessa i en meny skriver vi:

```
JMenuItem menuItem1 = new JMenuItem("Menyalternativ 1");
JMenuItem menuItem2 = new JMenuItem("Menyalternativ 2");
JMenuItem menuItem3 = new JMenuItem("Menyalternativ 3");
JRadioButtonMenuItem menuItem4 = new JRadioButtonMenuItem("Menyalternativ 4");
JRadioButtonMenuItem menuItem5 = new JRadioButtonMenuItem("Menyalternativ 5");
JCheckBoxMenuItem menuItem6 = new JCheckBoxMenuItem("Menyalternativ 6");

menu1.add(menuItem1);
menu1.add(menuItem2);
menu1.addSeparator();
menu1.add(menuItem3);
subMenu.add(menuItem4);
subMenu.add(menuItem5);
menu1.add(subMenu);
menu1.add(menuItem6);
```

När vi skapar de olika menyalternativen anger vi i konstruktorn vad som ska stå på menyalternativet. När vi lägger till menyalternativen i en meny hamnar de i samma ordning som add anropas (uppifrån och ner).

När det gäller radioknapparna kan vi lägga in dessa i en grupp om vi vill att endast ett alternativ ska kunna väljas. Vi använder då klassen ButtonGroup enligt följande:

```
ButtonGroup group = new ButtonGroup();
group.add(menuItem4);
group.add(menuItem5);
```

Nu kan endast ett av alternativen Menyalternativ 4 och Menyalternativ 5 vara markerat åt gången. Inledningsvis är inget alternativ markerat. Vill du att någon radioknapp ska vara markerad direkt måste du anropa setSelected(true) på den radioknapp som ska vara markerad.

En JMenuItem och dess subklasser ärver AbstractButton (samma klass som JButton ärver) och kan därför ses som ett slags knapp. På samma sätt som med JButton kommer därför en händelse av typen ActionEvent att genereras när användaren trycker på ett menyalternativ. Vill vi fånga och hantera denna typ av händelse måste vi implementera lyssnargränssnittet ActionListener och dess lyssnarmetod actionPerformed(ActionEvent). Givetvis måste vi även koppla en lyssnare till varje menyalternativ och det görs genom att anropa addActionListener på menyalternativet.

I lyssnarmetoden kan du göra som tidigare med JButton och anropa getSource() på händelseklassen ActionEvent för att jämföra vilket menyalternativ som genererade händelsen. Du kan även anropa metoden getActionCommand som returnerar en sträng vars innehåll är samma som texten som står i menyalternativet. Exempel:

```
@Override
public void actionPerformed(ActionEvent ae) {
  switch (ae.getActionCommand()) {
    case "Menyalternativ 1":
      output.setText("Menyalternativ: 1");
      break;
    case "Menyalternativ 2":
      output.setText("Menyalternativ: 2");
      break;
    case "Menyalternativ 3":
      output.setText("Menyalternativ: 3");
      break;
  }
}
```

Kortkommandon i menyer

För menyalternativ som används ofta är det vanligt att man lägger till kortkommandon. I många program används till exempel kortkommandot Ctrl-C för att kopiera markerad text eller Ctrl-S för att spara en öppen fil. I Java kan vi lägga till ett kortkommando för ett menyalternativ på två sätt. Antingen använder vi en mnemonic (som med JLabel och JButton) eller så använder vi en accelerator.

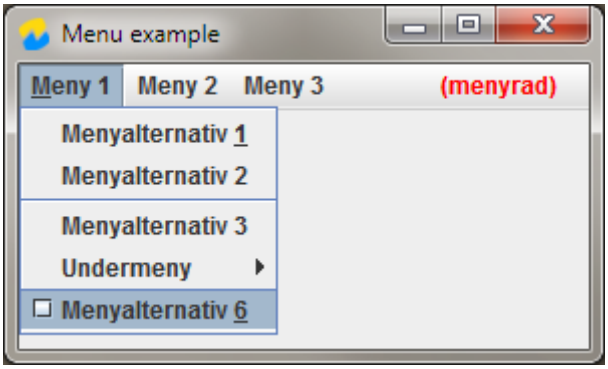
Skillnaden mellan dessa båda är att en mnemonic alltid kräver att alt-tangenten används samt att menyn måste vara öppnad. Med en accelerator kan vilken kontrolltangent (ctrl, alt, shift m.fl) som helst användas och menyn behöver inte vara öppnad. För en meny (JMenu) kan endast mnemonic

användas. Även om det finns en metod i JMenu för att ange en accelerator så fungerar det inte. Läser vi i API finner vi följande: ” setAccelerator is not defined for JMenu. Use setMnemonic instead”.

För att skapa ett kortkommando för en meny eller ett menyalternativ med hjälp av en mnemonic anropar vi metoden setMnemonic. För menyalternativ kan vi även ange en mnemonic i konstruktorn när menyalternativet skapas. Exempel:

```
menu1.setMnemonic(KeyEvent.VK_M);
menuItem1.setMnemonic(KeyEvent.VK_1);
JMenuItem menuItem6 = new JMenuItem("Menyalternativ 6", KeyEvent.VK_6);
```

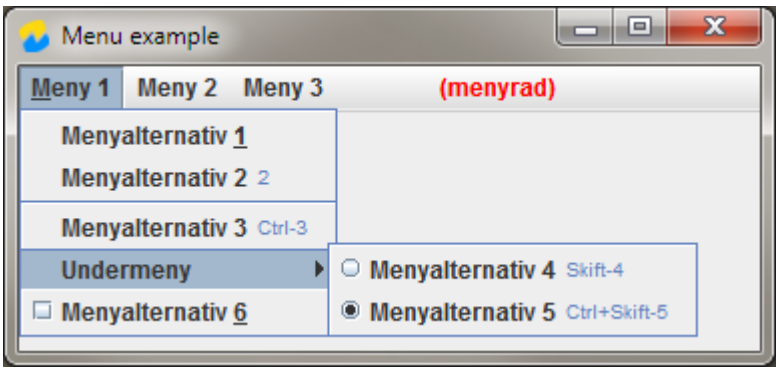
Användaren kan nu till exempel trycka på tangentbordskombinationerna alt-m följt av alt-1 för att välja menyalternativ 1.



För att i stället skapa ett kortkommando med hjälp av en accelerator måste vi anropa metoden setAccelerator. Det finns ingen möjlighet att ange en accelerator i konstruktorn. Exempel:

```
KeyStroke ks2 = KeyStroke.getKeyStroke(KeyEvent.VK_2, 0);
KeyStroke ks3 = KeyStroke.getKeyStroke(KeyEvent.VK_3, InputEvent.CTRL_MASK);
KeyStroke ks4 = KeyStroke.getKeyStroke(KeyEvent.VK_4, InputEvent.SHIFT_MASK);
KeyStroke ks5 = KeyStroke.getKeyStroke(KeyEvent.VK_5, InputEvent.SHIFT_MASK |
InputEvent.CTRL_MASK);
menuItem2.setAccelerator(ks2);
menuItem3.setAccelerator(ks3);
menuItem4.setAccelerator(ks4);
menuItem5.setAccelerator(ks5);
```

En accelerator representeras av klassen KeyStroke och skapas genom att anropa den statiska metoden getKeyStroke. Som första argument till konstruktorn anger vi vilken tangent som ska användas och som andra argument anges vilken eller vilka kontrolltangenter som ska tryckas ned.



För att välja menyalternativ 5 kan nu användaren trycka in tangentbordskombinationen Ctrl+Shift+5. Observera att menyn inte behöver vara öppnad för att välja menyalternativet. Om du inte vill använda en kontrolltangent anger du 0 som andra argument när du skapar din KeyStroke (se menyalternativ 2 ovan).