DD2385

Programutvecklingsteknik

Några bilder till föreläsning 3

15/4 2013

Innehåll

- Lite swing-intro
- Javas lyssnarinterface och lyssnarklasser
- Inre klasser
- Abstrakta klasser
- Klasshierarki och typhierarki
- Polymorfism och dynamisk bindning
- Polymorfi-exempel: Schack

Grafiska komponenter i Java: paketet awt

Klasserna i listan är exempel på subklasser till Component.

- Button
- Canvas
- Label
- TextComponent
  - TextField liten textrad
  - TextArea flera rader text
- Container kan innehålla andra komponenter
  - **swing** (många många subklasser)
  - Panel
    - Applet
  - ScrollPane
  - Window fristående fönster
    - Dialog
    - Frame
- Scrollbar

Swinggrafik

- Bygger på awt, alla ärver från Container
- Swing-komponenterna heter JFrame, JButton, JLabel ...
- Mer avancerat och flexibelt än awt
- Lite långsammare än awt
- Lättviktskomponenter – Java istället för lokalt OS
  Utom JFrame JApplet JDialog JWindow
- Samma utseende i Windows, Mac, Unix ...
  men utseendet kan ändras av programmeraren
- Finns i paketet `javax.swing`

JFrame

Innanför det yttersta fönstret finns en lättviktsbehållare.
Där läggs komponenterna.

```
class OnlyJFrame extends JFrame {

    OnlyJFrame() {
        setSize(400, 300);
        Container container = getContentPane();
        container.setBackground(Color.blue);

        // container.add ( ... )

        setDefaultCloseOperation( ... );
        setVisible(true);
    }
}
```

`add(component)` fungerar ju också! Lite "lurigt"!

Varför interface för lyssnare?

Liten liten del av klassen Button:

```
class Button extends Component {
   ...
   public void addActionListener(____ li) {...}
   ...
}
```

Vilken typ har parametern li ? Vad ska det stå vid ____ ?

Svar: ActionListener

Enda kravet på lyssnarobjektet är att det implementerar

```
interface ActionListener {
   public void actionPerformed(ActionEvent e);
}
```

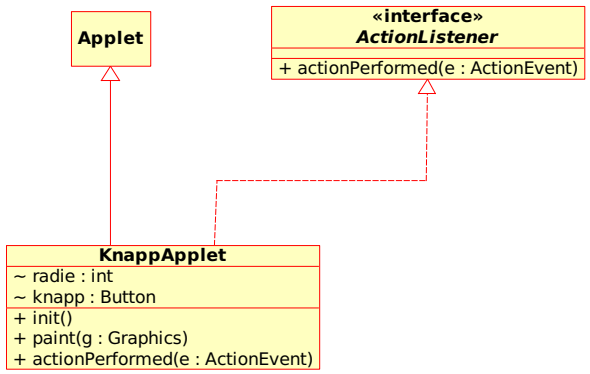
Andra egenskaper är irrelevanta för addActionListener!

Olika sätt att göra lyssnare

En lyssnare kan vara

1. Huvudklassen själv/klassen själv
2. Separat klass
3. Inre namngiven klass
4. Inre anonym klass

1) Huvudklassen är lyssnare, t.ex. i KnappApplet



2) Lyssnaren är en separat klass

Lyssnare.java

```
class Lyssnare implements ActionListener {
    ....
  public void actionPerformed (ActionEvent e){
      // do something
  }
}
```

Huvud.java

```
class Huvud .... {
    :
  knapp.addActionListener(new Lyssnare());
    :
}
```

Inre klass

- Definieras inuti en annan klass.

```
class Yttre {
    ...
    class Inre { ... }
    ...
}
```

- Inre är liten.
- Inre används bara inuti Yttre.
- Kompileras till Yttre.class och Yttre$Inre.class
- Passar för små lyssnare
- Passar för andra små hjälpklasser

3) Lyssnaren är en inre klass

Huvud.java

```
class Huvud ... {
   ...
   class Lyssnare implements ActionListener {
    ....
      public void actionPerformed (ActionEvent e){
         // do something
      }
   }
   :
      // in a method in Huvud:
      knapp.addActionListener(new Lyssnare());
   :
}
```

4) Lyssnaren är en anonym inre klass som bara används en gång

Huvud.java

```
class Huvud ... {
   ...
   // in a method in Huvud:
   knapp.addActionListener(new ActionListener(){

      public void actionPerformed (ActionEvent e){
          // do something
      }

   });
   :
}
```

Klassen som implementerar ActionListener definieras samtidigt som objekt av den skapas och kopplas som lyssnare

- Java har ca 10 lyssnarinterface, t.ex.
  - ActionListener
  - MouseListener
  - MouseMotionListener
  - AdjustmentListener
  - KeyListener
- De flesta har mer än en metod
- Alla metoder måste implementeras, även om man inte behöver dem.

interface MouseListener, ur biblioteket

```
public interface MouseListener {

  public void mouseClicked(MouseEvent e);
  public void mouseEntered(MouseEvent e);
  public void mouseExited(MouseEvent e);
  public void mousePressed(MouseEvent e);
  public void mouseReleased(MouseEvent e);

}
```

Implementation av MouseListener

Detektera musklick:

Alla metoder måste definieras även om de inte ska användas

```
class MyMouseL implements MouseListener {

  public void mouseClicked(MouseEvent e) {
    // define mouse-click-action
  }

  // empty methods
  public void mouseEntered(MouseEvent e){};
  public void mouseExited(MouseEvent e){};
  public void mousePressed(MouseEvent e){};
  public void mouseReleased(MouseEvent e){};
}
```

Adapterklasserna

Till varje XxxListener med mer än en metod finns en klass XxxAdapter som implementerar alla metoderna, tomma.

MouseAdapter.java, bibliotekskklass

```
public class MouseAdapter implements MouseListener{
   public void mouseClicked(MouseEvent e){};
   public void mouseEntered(MouseEvent e){};
   public void mouseExited(MouseEvent e){};
   public void mousePressed(MouseEvent e){};
   public void mouseReleased(MouseEvent e){};
}
```

Adapterklasserna

En lyssnarklass som ärver från en adapterklass kan definiera om bara de metoder som önskas.

```
public class MyMouseL extends MouseAdapter {

  public void mouseClicked(MouseEvent e){
     // define mouse-click-action
  }
  // no more methods needed
}
```

Problem: en Javaklass kan bara ärva från en annan klass

`class X extends XxxAdapter`

⇒ X stängd för ytterligare arv

Adapterklasser som felfälla

Följande går bra att kompilera men reagerar inte på musklick!

```
public class MyMouseL extends MouseAdapter{

  public void mouseclicked(MouseEvent e){
     // define mouse-click-action
  }
}
```

Abstrakt klass

- Inleds med `abstract class`
- Kombination av vanlig klass och interface
- Kan innehålla vanliga metoder och variabler
- Innehåller oftast minst en abstrakt metod som definieras i subklass
- Objekt av klassen kan *inte* skapas

Exempel: Implementation av Schackpjäser

- De olika pjäserna har mycket gemensamt
- Men förflyttning sker på olika sätt
- Definiera en abstrakt klass med allt gemensamt
- Definiera en konkret subklass för varje pjästyp:
  - Bonde
  - Springare
  - Löpare
  - Torn
  - Dam
  - Kung

Exempel: Skiss av klasser

```
abstract class Schackpjas {
  Color farg; Image bild;
  Brade brade; Ruta ruta;

  Schackpjas (...) {...}   // konstruktor

  abstract boolean dragOK(Ruta r1, Ruta r2);
}
```

```
class Bonde extends Schackpjas {
  boolean dragOK(Ruta r1, Ruta r2) {
    // implementera bondens drag
  }
}
```

Konstruktor m.m. visas ej.

Exempel: Skiss av klasser

```
class Dam extends Schackpjas {
  boolean dragOK(Ruta r1, Ruta r2) {
    // implementera damens drag
  }
}
```

```
class Torn extends Schackpjas {
  boolean dragOK(Ruta r1, Ruta r2) {
    // implementera tornets drag
  }
}
```

```
class Springare extends Schackpjas {
  boolean dragOK(Ruta r1, Ruta r2) {
    // implementera springarens drag
  }
}
```

Klasshierarki = Typhierarki

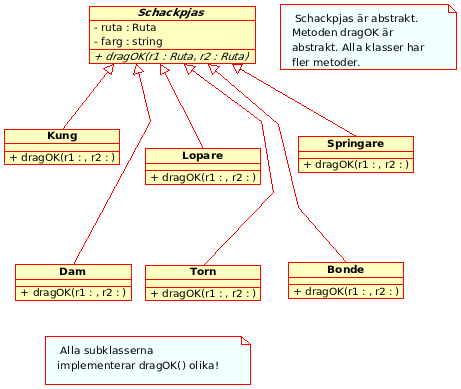
- Klasshierarkin är en trädstruktur
- Referenser av viss typ får referera till objekt av typer som är nedanför i trädet.
- Schackpjas får referera till Bonde, Dam, Kung, Torn ...
- En Schackpjas har många "former" (Bonde, Dam ...): POLYMORFISM

Dynamisk bindning

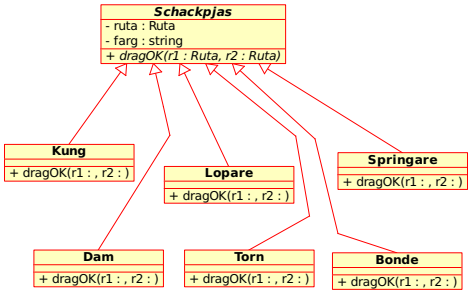
- En Schackpjas-referens "ser" endast Schackpjas-definitioner, t.ex. `dragOK`
- Om en metod, t.ex. `dragOK`, definierats om i en subklass
  så används den omdefinierade!
- Objektetets typ och inte referensens typ "väljer" metod.
  Detta är dynamisk bindning
- ```
  Schackpjas pjas = new ...//Dam, Kung, Torn...
  ...
  pjas.dragOK(..)  //Dynamiskt metodval
                   //beroende av typen
                   //hos pjas-objektet
  ```
  ```

Klass/typ - hierarki för Schackpjäserna

med UML-notislappar



Klass/typ - hierarki för Schackpjäserna



`dragOK()` är en *abstrakt* metod.
`dragOK()` implementeras olika i `Kung, Dam, Lopare ....`
Alla klasserna har fler metoder