DD2385
Programutvecklingsteknik
Några bilder till föreläsning 2
31/3 – 2014

Innehåll

- Grafik-introduktion i Java
- Interface i Java
- Lyssnarinterface för grafisk interaktion i Java

Grafiska program i Java

- Använd grafik-bibliotek (awt, swing)
- Välj en fönsterklass som superklass
- Skriv subklass
- Lägg in andra grafiska objekt

Grafiska komponenter i Java: paketet awt

Klasserna i listan visar en klasshierarki under klassen Component

- Button
- Canvas
- Label
- TextComponent
  - TextField liten textrad
  - TextArea flera rader text
- Container kan innehålla andra komponenter
  - Panel
    - Applet visas på webbsida
  - ScrollPane
  - Window fristående fönster
    - Dialog används i
    - Frame fristående program
- Scrollbar

Första grafiska programmet blir en Applet

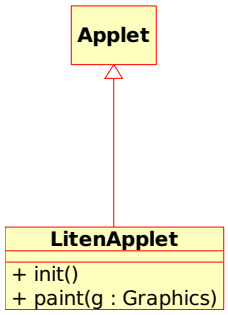
Skriv en subklass till bibliotheksklassen Applet

Definiera om metoderna `init()` och `paint()`

`init()` – initierar
`paint()` – ritar

UML för LitenApplet

Applet kommer från biblioteket awt
LitenApplet är vår egen klass
init() och paint() ärvs tomma och definieras om



Många metoder ärvs från Applet och kan användas i subklassen.

Javakod för LitenApplet

```
import java.applet.*;
import java.awt.*;
public class LitenApplet extends Applet {

  public void init () {
    add (new Label("En liten applet for kursen"+
                   " programutvecklingsteknik"));
  }

  public void paint (Graphics g) {
    g.setColor(Color.cyan);
    g.fillRect(100,100,50,50);
    g.setColor(Color.magenta);
    g.fillRect(150,100,50,50);
    g.setColor(Color.yellow);
    g.fillRect(200,100,50,50);
  }
}
```

Appletkoden är hela programmet!

- Javasystemet skapar ett objekt av klassen `LitenApplet`
- Javasystemet anropar metoden `init()` en gång
- Javasystemet anropar metoden `paint()` många gånger

Kör appleten!

Fil: LitenApplet.java

Kompilera:
```
>>>javac LitenApplet.java
      => LitenApplet.class
```

Skriv en html-fil:

```
<applet code = LitenApplet.class
width = 400 height = 200></applet>
```

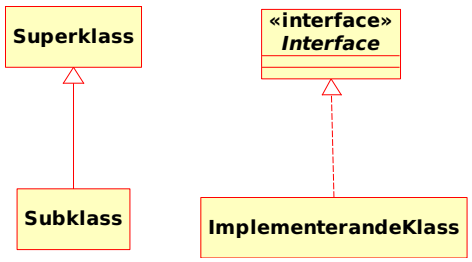
Kör appleten:
```
>>>appletviewer LitenApplet.html
```

eller lägg på webbsida

Interface

- Beskriver ett *abstrakt beteende*.
- Nära släkt med abstrakt klass.
- Interface/abstrakta klasser är mycket viktiga i
  - OO–prog i Java.
  - Java-biblioteken
  - Designmönster

UML för arv och interface



Interface

Typdefinition, beskriver ett *abstrakt beteende*.

```
public interface Monster {
    public void walk();
    public void scream();
    public void eat();
}
```

Alla Monster-objekt har metoderna

```
...
Monster aake = ...
...
aake.eat();
aake.scream();
aake.walk();
```

Interface

aake måste vara ett konkret Monster, t.ex. ett CookieMonster

```
class CookieMonster implements Monster {

    public void eat(){
        // asks for cookies and eats them
    }
    public void scream(){
        // muffled sound,
        // mouth is usually full of cookies
    }
    public void walk() {
        // slow and peaceful
    }
}
```

Metoderna måste fyllas med verkligt innehåll
Attribut och fler metoder får finnas i CookieMonster

Interface

Ytterligare en sorts Monster: ScaryMonster

```
class ScaryMonster implements Monster {

    public void eat(){
        // EATS YOU, UNLESS YOU HAVE A PIZZA,
        // THEN IT TAKES YOUR PIZZA
    }
    public void scream(){
        // OOOAAAAAAGHHHHHH - LOUD AND SCARY
    }
    public void walk() {
        // HUGE LEAPS, QUICK AND UNPREDICTABLE
    }
}
```

Metoderna måste fyllas med verkligt innehåll
Attribut och fler metoder får finnas i CookieMonster

Interface

Skapa och hantera Monster

```
Monster aake = new CookieMonster();
Monster loke = new ScaryMonster();
Monster aasa = new ScaryMonster();
```

```
void feedMonsters(Monster[] monsterfamily) {
      :
    for (Monster m : monsterfamily) {
        m.eat();
        if (Math.random() > 0.7)
            m.scream();
    }
      :
}
```

`m.eat()` och `m.scream()` gör olika saker för
`CookieMonster` respektive `ScaryMonster`

Interface

Interface kan kombineras med

- Arv från annan klass
- Nya metoder
- Nya attribut

En klass kan ärva från högst en annan klass men implementera flera interface

```
class MySubclass extends BigSuperclass
                         implements I1, I2, I3 {

   // concrete implementations of all methods
   // in I1, I2, I3 required

   // constructor, other methods and attributes
   // allowed
}
```

Objekt av `MySubclass` är typmässigt `I1` och `I2` och `I3` och ...

Javas lyssnarinterface

- Interaktion i grafiska fönster genererar *händelser (events)*
- Interface används för att definiera *lyssnarklasser*
- När händelse detekterats anropas metod i *lyssnarobjekt*.
- Lyssnarobjektet implementerar lämpligt *lyssnarinterface*.
- Lyssnarobjekt kopplas till komponenten där händelse väntas.
- En händelse generar anrop av metod i lyssnarobjektet.

Att göra

- Skapa grafiskt objekt att lyssna på, t.ex. en knapp *b*
- Implementera lyssnarinterface ⇒ lyssnarklass ⇒ *metod*
- Koppla objekt av lyssnarklassen till *b*

Tryck på knappen så anropas *metod*

Javas lyssnarinterface

Java har många inbyggda lyssnarinterface
ActionListener lyssnar bl.a. på knapptryckningar,
finns paketet `java.awt.event`

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

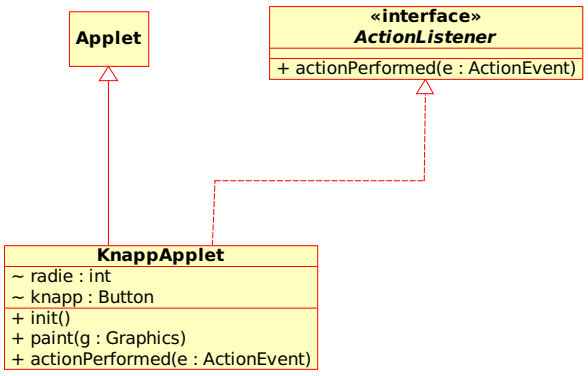
Implementeras:

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

Vilken klass ska implementera? Flera möjligheter finns.

UML för KnappApplet

Applet-klassen får vara lyssnarklass.



Javakod för KnappApplet

Applet-klassen får vara lyssnarklass.

```
import java.awt.event.*;
import java.applet.*;
import java.awt.*;
public class KnappApplet extends Applet
                    implements ActionListener {

    int radie = 5;
    Button knapp = new Button("Tryck␣har");

    public void init () {...}

    public void actionPerformed (ActionEvent e) {...}

    public void paint (Graphics g) {...}
}
```

Metoderna init, actionPerformed och paint

```
public void init () {
  setBackground(Color.white);
  add(knapp);
  knapp.addActionListener(this);
}

public void actionPerformed (ActionEvent e) {
  radie +=5;
  repaint();
}

public void paint (Graphics g) {
  int x = getWidth()/2;
  int y = getHeight()/2;
  g.setColor(Color.red);
  g.fillOval(x-radie,y-radie,2*radie,2*radie);
}
```

Vi bygger upp programmet stegvis under föreläsningen.

Kör appleten!