Föreläsning 22

- Klassen Object
- Överskuggning av toString, equals
- Abstrakt klass
- Ärva grafisk komponent
- Applet – program som exekverar i browsern

JF 8.3-8.5

Viktiga OO-begrepp

- Klass
- Objekt
- Överskugga
- Överlagra
- Ärva
- Polymorfism
- Dynamisk bindning
- Abstrakta klasser
- Gränssnitt

Klassen Object

Om man inte anger arv (extends) av en klass så ärvs klassen Object automatiskt. Det innebär att alla klasser i java ärver av klassen Objekt. Klassen Object har ett antal metoder, bl.a. (se dokumentationen över Javas API)

- public boolean equals(Object obj)
  Indicates whether some other object is "equal to" this.
- public Object getClass()
  Returns the runtime class of an object.
- public String toString()
  Returns a string representation of the object.

Klassen Object – direkt arv

Klassen **Subclass1** ärver direkt av klassen Object:

```
public class Subclass1 {
}
```

Subclass1.java

TestSubclass.java

Testkod

```
Subclass1 sc1 = new Subclass1();
// Anrop av toString-metoden ärvd av klassen Object
System.out.println(sc1.toString());
```

Körresultat

```
f22.Subclass1@82ba41
```

Klassen Subclass1 skulle även kunna deklareras så här vilket ger samma resultat:

```
public class Subclass1 extends Object {
}
```

Klassen Object – indirekt arv

Klassen **Subclass2** ärver av klassen Object via Subclass1:

```
public class Subclass2 extends Subclass1 {
}
```

Testkod

```
Subclass2 sc2 = new Subclass2();
// Anrop av toString-metoden ärvd av klassen Object
System.out.println(sc2.toString());
```

Körresultat

```
f22.Subclass2@130c19b
```

Subclass2java

TestSubclass.java

Exempel på överskuggning

Klassen **Point1** ärver av klassen **Object**.
Klassen **Point1** har en egen **toString**-metod som överskuggar den ärvda **toString**-metoden.
Klassen **Point1** använder den ärvda **equals**-metoden.

Point1
-x : double -y : double
+ getX() : double + getY() : double + toString() : String

```
Point1 p1 = new Point1(10.2, 2.4);
Point1 p2 = new Point1(10.2, 2.4);
Point1 p3 = p1;

System.out.println( p1 + ”\n” + p2 + ”\nSamma: ” + p1.equals(p2) );
System.out.println( p1 + ”\n” + p3 + ”\nSamma: ” + p1.equals(p3) );
```

Körresultat

```
(10.2, 2.4)
(10.2, 2.4)
Samma: false
(10.2, 2.4)
(10.2, 2.4)
Samma: true
```

Point1

TestPoint1

equals-metoden kontrollerar om objekten-referenserna (p1 och p2 resp. p1 och p3) refererar till samma objekt. Detta är fallet med p1 och p3 medan p1 och p2 refererar till olika objekt.

Exempel på överskuggning

Klassen **Point2** ärver av klassen **Object**.
Klassen **Point2** har egen **toString**-metod
och **equals**-metod som överskuggar de ärvda
metoderna **toString** och **equals**.

Point2
-x : double -y : double
+ getX() : double + getY() : double + toString() : String + equals(Object) : boolean

```
Point2 p1 = new Point2(10.2, 2.4);
Point2 p2 = new Point2(10.2, 2.4);
Point2 p3 = p1;

System.out.println( p1 + ”\n” + p2 + ”\nSamma: ” + p1.equals(p2) );
System.out.println( p1 + ”\n” + p3 + ”\nSamma: ” + p1.equals(p3) );
```

Körresultat

```
(10.2, 2.4)
(10.2, 2.4)
Samma: true
(10.2, 2.4)
(10.2, 2.4)
Samma: true
```

Point2

TestPoint2

Metoden equals i Point2

```
public boolean equals(Object obj){
   Point2 aPoint;
   if( obj instanceof Point2 ) {
      aPoint = (Point2) obj;
      if ( (x==aPoint.getX()) && (y==aPoint.getY()) )
         return true;
   }
   return false;
}
```

Metoden jämför två objekt av typen Point2. Metoden returnerar true, dvs. två punkter är samma, om de har

samma x-koordinat:	**(x == aPoint.getX())**
och	**&&**
samma y-koordinat:	**(y == aPoint.getY())**

Metoden kan skrivas betydligt kortare:

```
public boolean equals(Object obj){
   return  (obj instanceof Point2)   &&
         ( (x == ((Point2)obj).getX()) && (y == ((Point2)obj).getY()) );
}
```

Metoden equals och ==

= = kontrollerar om två variabler refererar till samma objekt.

equals-metoden i Point2-klassen kontrollerar om två variabler refererar till Point2-objekt med samma x-koordingat och y-koordinat.

equals-metoden i Object-klassen utför samma test som = =. Detta kunde du konstatera när du testkörde TestPoint1.java.

Speciell kommentar:

Använd alltid equals-metoden (överskuggad i String-klassen) när du jämför strängar. Om du använder = = så

- Vill du kontrollera om två String-referenser är till samma objekt
- Vet du när java-kompilatorn använder redan skapade String-objekt. Information om detta finner du i API-dokumentationen om String.

Abstrakt klass

Man kan tänka sig att cirklar och rektanglar är figurer. De har ju visa egenskaper gemensamt. T.ex. kan figurer ha en area.
Klassen Figur måste dock bli lite mer specialiserad för att kunna skapas som en instans.

- En abstrakt klass deklareras abstract.
- En abstrakt klass kan inte skapas med new. En abstrakt klass är superklass till andra klasser.
- I abstrakta klasser finns ofta abstrakta metoder som subklasserna måste implementera.
  En abstrakt metod deklareras abstract och har ingen kropp.
- Abstrakta klasser kan ha vanliga metoder och instansvariabler.

Några extra punkter:

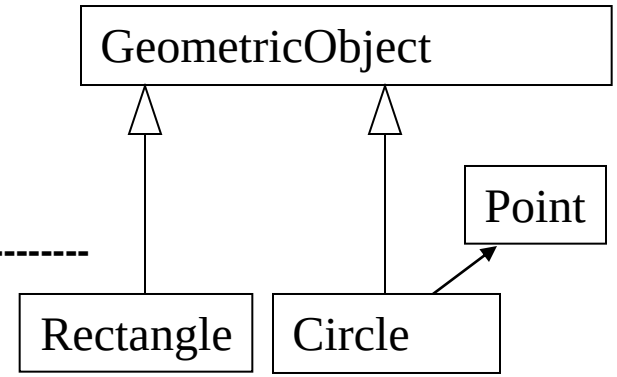
- En klass kan deklareras som abstact även om den inte innehåller några abstrakta metoder. Klassen kan då endast ärvas.
- En subklass som inte implementerar abstrakta metoder i superklassen måste deklareras som abstract.

GeometricObject

```
public abstract class GeometricObject {                          // deklaration
   protected GeometricObject() {                                 // konstruktor

   }

   public abstract double area();                                // abstrakt metod

   public double difference( GeometricObject geoObject ) {       // vanlig metod
      return area() - geoObject.area();
   }
}
```

```
public class Rectangle extends GeometricObject {
   :
   public double area() {
      return width * height;
   }
}
```

```
public class Circle extends GeometricObject {
   Point point;
   :
   public double area() {
      return Math.PI * radius * radius;
   }
}
```



- GeometricObject
- Circle + Point
- Rectangle

TestGeometricObject

```
public class TestGeometricObject {
   public static void main(String[] args) {
      GeometricObject geo1 = new Circle( 1, 1, 1 );
      GeometricObject geo2 = new Rectangle( 1, 1 );
      System.out.println( geo1.toString() + " och har arean " + geo1.area() + " a.e." );
      System.out.println( geo2.toString() + " och har arean " + geo2.area() + " a.e" );
      System.out.println( "Skillnaden i area är " + geo1.difference(geo2) +  " a.e.");
   }
}
```

Körresultat

Cirkeln har radien 1.0 l.e. och mittpunkten: (1.0, 1.0) och har arean 3.141592653589793 a.e.
Rektangelns bredd är 1.0 l.e. och höjd är 1.0 l.e. och har arean 1.0 a.e
Skillnaden i area är 2.141592653589793 a.e.

TestGeometricObject

Modifierare

- Synlighet

public	+	alla kommer åt
private	-	klassen kommer åt
protected	#	klassen och subklasserna kommer åt

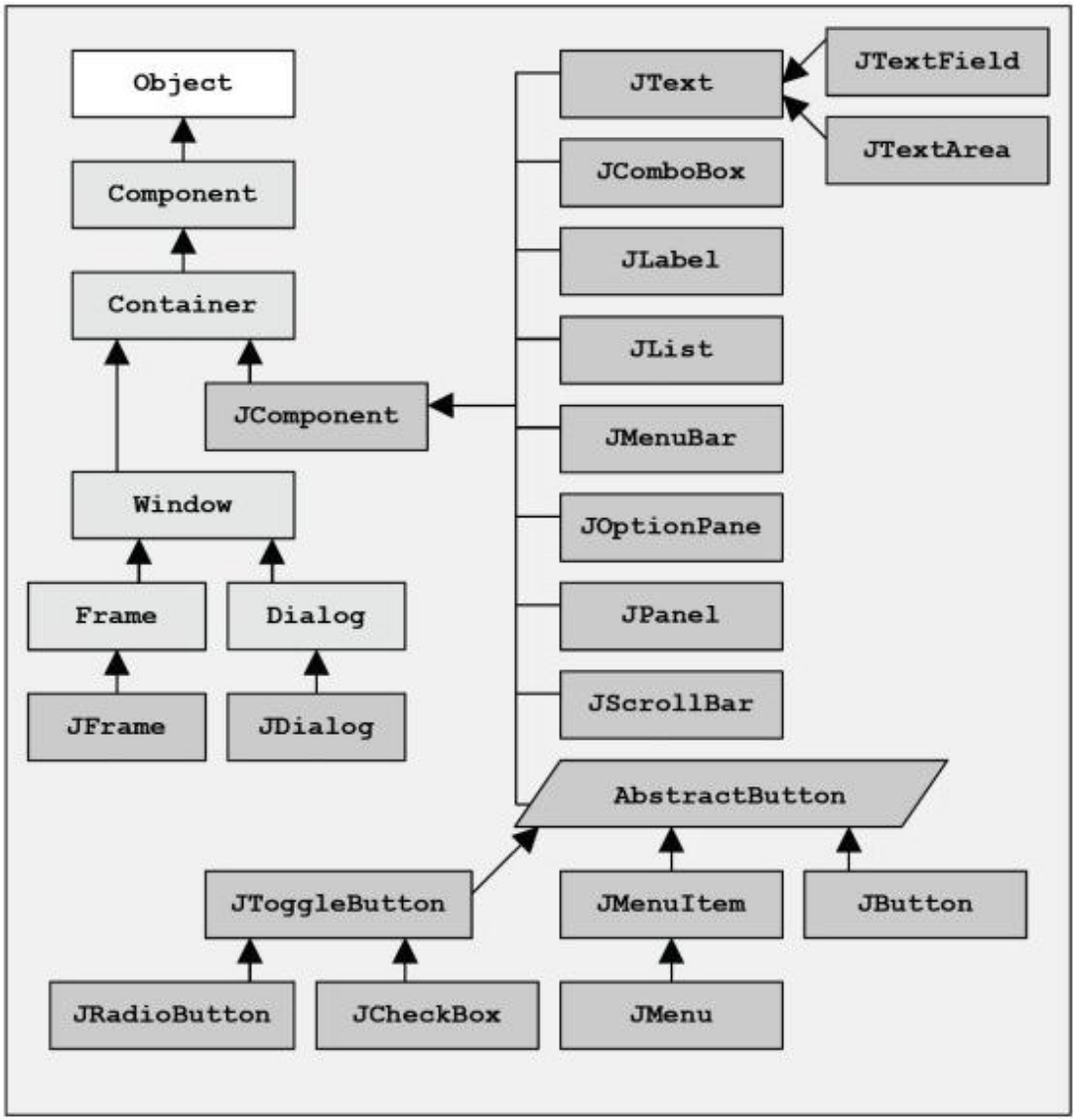
- Metoder

final	kan inte överskuggas i subklass
abstract	måste implementeras i subklass
static	finns bara en i varje klass

- Klasser

final	kan instansieras, kan inte ärvas
abstract	kan ärvas, kan inte instansieras

Klasshierarki, swing

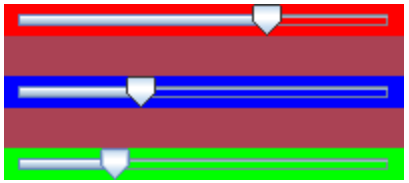


This diagram shows a *subset* of the Swing components (in dark gray) and how they extend the AWT (in light gray) components.

Ärva JPanel

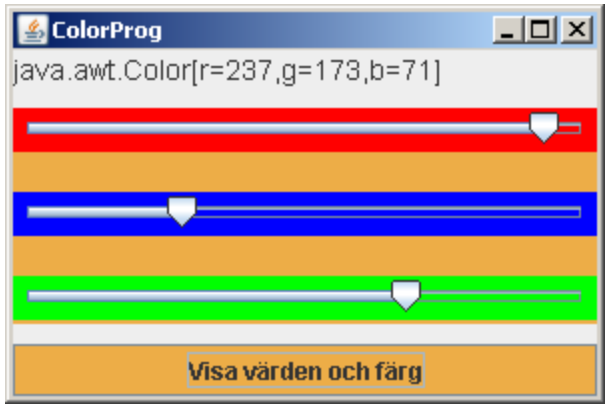
En klass kan ärva JPanel. Den nya klassen är då en JPanel. Eftersom en JPanel är en containerklass så kan man placera komponenter på panelen. När man är nöjd med panelens utseende och funktion kan den användas i grafiska program.

```
public class ColorPanel extends JPanel implements ChangeListener {
    private JSlider red = new JSlider(0, 255, 128);
    private JSlider green = new JSlider(0, 255, 128);

    :
}
```



Objekt av panel-klassen kan sedan användas i grafiska program.

```
public class DemoColorPanel extends JPanel {
    private ColorPanel pnlColor = new ColorPanel();
    :
    public DemoColorPanel() {
        :
        add(pnlColor);
        :
     }
}
```



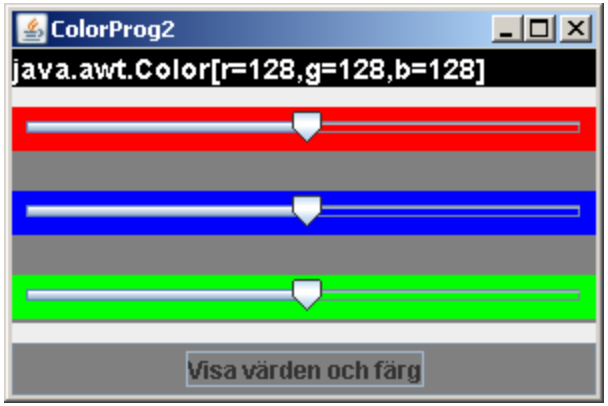
ColorPanel | DemoColorPanel | ColorProg1

Ärva JLabel

En klass kan ärva JLabel. Man kan t.ex. ge den nya JLabel-komponenten några defult-värden:

```
public class BlackAndWhite extends JLabel {
    public BlackAndWhite(String str) {
        setForeground(Color.WHITE);
        setBackground(Color.BLACK);
        setOpaque(true);
        setFont(new Font("SansSerif",Font.BOLD,14));
        setText(str);
    }
}
```

Sedan kan man använda BlackAndWhite-objekt där man vill visa vit text mot svart bakgrund i program.




```
public class ColorDemo2Panel implements ActionListener {
    private ColorPanel pnlColor = new ColorPanel();
    private BlackAndWhite lblValues = new BlackAndWhite(" ");
```

BlackAndWhite | DemoBlackAndWhite | ColorProg2

Ärva JFrame

En klass kan ärva JFrame. Då är klassen ett fönster

```
public class ColorProg2Frame extends JFrame implements ActionListener {
   public constructGUI() {
      this.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      this.setResizable( false );
      this.add(new DemoBlackAndWhite() );
      this.pack();
      this.setVisible( true );
   }

   public static void main(String[] args) {
      SwingUtilities.invokeLater(new Runnable() {
         public void run() {
            ColorProg2Frame prog = new ColorProg2Frame();
            prog.constructGUI();
         }
      });
   }
}
```

ColorProg2Frame

JApplet

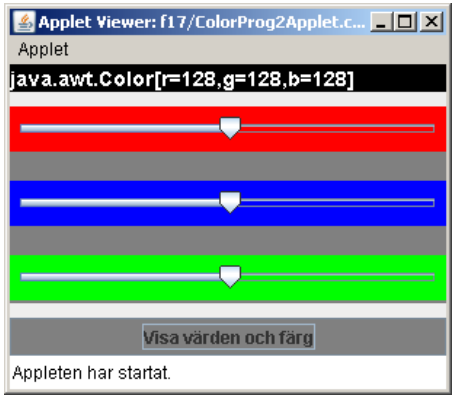
Om man har ett program som ärver JFrame är det som regel enkelt att göra om programmet till en applet, till ett program som *exekveras i en browser*.

- Klassen ska ärva JApplet i stället för JFrame.
- Fönsterinställningar ska tas bort från constructGUI-metoden (ex **this.setVisible(true);**)
- Programmet startas av browsern genom anrop till metoden ***init***.
  * Ta bort eventuell main-metod ur klassen.
  * Lägg till init-metod:

```
public void init() {
    try {
        SwingUtilities.invokeAndWait(new Runnable() {
            public void run() {
                constructGUI();
            }
        });
    } catch (Exception e) {}
}
```

När du kör appleten kommer appleten visas med programmet AppletViewer.

ColorProg2Applet



JApplet

I HTML-filen som läses av browsern krävs en APPLET-tagg för att starta appleten.

<APPLET code="f22/ColorProg2Applet.class" width=300 height=200></APPLET>

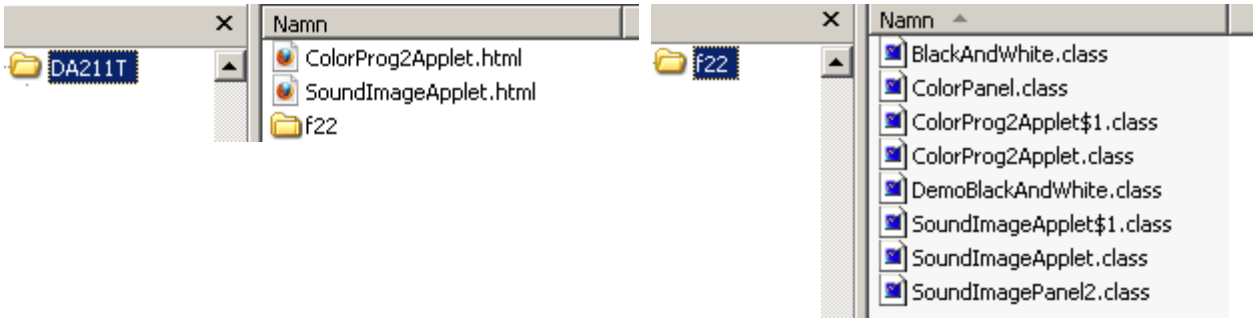
I APPLET-taggen anger man:

Applet-klassens namn inklusive paket, **code = f22/ColorProg2Applet.class**

Hur brett fönster appleten ska visas i: **width = 300**

Hur högt fönster appleten ska visas i: **height = 200**

På web-servern ska katalogen f22 finnas tillsammans med HTML-filen. I katalogen f22 ska alla class-filer som används av programmet finnas. Dock inte javas standardklasser (ska finnas på datorn där programmet exekveras). I det här fallet är det ***ColorProg2Applet.class***, ***ColorProg2Applet$1.class***, ***DemoBlackAndWhite.class***, ***ColorPanel.class*** och ***BlackAndWhite.class***.



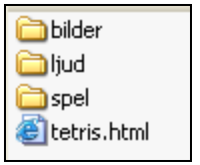
ColorProg2Applet.html

Bild och ljud i applet

Filen JAppletMall.java innehåller metoden ***getImageIcon*** för att hämta en bildfil och ***getAudioClip*** för att hämta en ljudfil. Kopiera metoderna till den klass som ärver JApplet. Glöm inte att importera paketen java.applet och java.net.

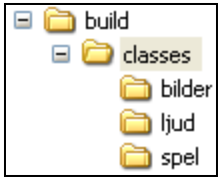
```
public ImageIcon getImageIcon(String path) // hämta bild
public AudioClip getAudioClip(String path)  // hämta ljud
```

Bild- och ljud-filer ska placeras relativt applet-filen. I nedanstående exempel finns bildfilerna i katalogen ***bilder***, ljudfilerna i katalogen ***ljud*** och appleten i paketet ***spel***. Html-filen med APPLET-tagg heter ***tetris.html***.



På webservern (t.ex. M:\public_html) ska katalogerna *bilder*, *ljud* och *spel* lagras på samma ställe som *tetris.html*.

I eclipse ska katalogerna *bilder* och *ljud* placeras i katalogen ***bin***.



Hämta bildfilen spegel.jpg:

```
ImageIcon im = applet.getImageIcon( "bilder/spegel.jpg" );
```

Hämta ljudfilen kvack.wav:

```
AudicClip clip = applet.getAudioClip( "ljud/kvack.wav" );
```

JAppletMall

SoundImagePanel2

SoundImageApplet

Inre klasser

Det är tillåtet att skriva en klass inuti en annan.

```
public class OuterClass {
   private int a = 20;
   private InnerClass cls = new InnerClass();

   public OuterClass(…) {
      cls.b = cls.b + 2;
      cls.innerMethod1();
   }

   private class InnerClass {
      private b = 10;

      private void innerMethod1() {
         a = a + 1;
      }
   }
}
```

Den inre klassen har tillgång till attribut och metoder i huvudklassen. Huvudklassen har tillgång till inre klassers attribut och metoder (via instans av den inre klassen).

Den kompilerade inre klassen får namnet **OuterClass$InnerClass.class**

OuterClass.java

Events1, 2, 3