DD1385
Programutvecklingsteknik
Några bilder till
Föreläsning 1
hösten 2023

- Kursöversikt
- Javarepetition/Javaintroduktion
- UML - klassdiagram-introduktion
  i anslutning till Java-exemplen

Kursmål, förkortat

- Använda objektorienterade tekniker vid eget programmeringsarbete
- Redogöra för och tillämpa kriterier för god objektorienterad design
- Redogöra för de vanligaste designmönstren inom objektorienterad programutveckling samt välja lämpliga mönster för enkla tillämpningsexempel.
- Använda UML-klassdiagram för att på ett överskådligt och tydligt sätt planera och dokumentera eget programmeringsarbete.
- Läsa och förstå UML-klassdiagram, t.ex. som introduktion till nya designmönster.
- Använda Javas bibliotheksklasser och ramverk

Kursinnehåll

- Processen från Informell kravspecifikation
  till Färdigt program
- Objektorienterad programmering i Java med
  UML, designkriterier, designmönster

- Processen från en informell kravspecifikation till färdigt program
  - OOD (ObjektOrienterad Design)
  - Designmönster
  - Kriterier för god design
  - Refactoring
  - UML–klassdiagram
  - Testning
  - Versionshantering

- Objektorienterad programmering i Java med
  - Klasser, objekt, konstruktorer, arv, static
  - Abstrakta klasser, interface
  - Klassbibliotek och ramverk
  - Designmönster i Java-biblioteken
  - Grafiska gränssnitt, i första hand med Swing
    - Enkla komponenter, t.ex. JLabel, JButton
    - Avancerade komponenter, t.ex.
      JList, JTree, JEditorPane
  - Parallella processer: trådar
  - Klient-server-program
    (där klient och server är på olika datorer)
- XML

Innehåll

- Arv, konstruktor vid arv
- Metoden `public String toString()`
- UML - klassdiagram-introduktion
  i anslutning till Java-exemplen
- Grafik-introduktion i Java
- Interface i Java
- Lyssnarinterface för grafisk interaktion i Java

Exemplen med
Spelkort och Patienskort
är mycket viktiga.
Studera dem noga!

Klass för Spelkort

```
class Spelkort {
  static String[] specvalor
      = {"ESS", "KNEKT", "DAM", "KUNG"};
  String farg;
  int valor;

  //Konstruktor
  Spelkort (String f, int v) {
    farg = f;    valor = v;
  }

  //Visa text-utseende
  public String toString () {...}
  }
```

toString-metoden i Spelkort

```
class Spelkort {

    // instansvariabler, konstruktor
    // som visats tidigare

    public String toString () {
        String valorString;
        if (valor == 1)
            valorString = specvalor[0];
        else if (valor >=2 && valor <=10)
            valorString = "" + valor;
        else
            valorString = specvalor[valor - 10];
        return farg + " " + valorString;
    }
    // ... som tidigare
}
```

toString-metoden i Spelkort (samma som förra bilden)

instansvariabel, klassvariabel, lokal variabel

```
class Spelkort {
    // instansvariabler, konstruktor
    // som visats tidigare
    public String toString () {
        String valorString;
        if (valor == 1)
            valorString = specvalor[0];
        else if (valor >=2 && valor <=10)
            valorString = "" + valor;
        else
            valorString = specvalor[valor - 10];
        return farg + " " + valorString;
    }
    // ... som tidigare
}
```

Testa Spelkortsklassen:

Klass med bara main-metod som skapar och visar kortlek

```
class TestaSpelkort {
  public static void main (String[] arg) {

    // Skapa vektor med plats for 52 Spelkort

    // Skapa korten

    // Visa korten
  }
}
```

Testa Spelkortsklassen

```
class TestaSpelkort {
  public static void main (String[] arg) {
    String[] farger =
      {"Hjarter", "Spader", "Ruter", "Klover"};
    // Skapa vektor med plats for 52 Spelkort.
    Spelkort[] kortlek = new Spelkort[52];

    // Skapa korten
    int kortnr = 0;
    for (String farg : farger)
      for (int valor = 1; valor<=13; valor++)
        kortlek[kortnr++] =
          new Spelkort(farg, valor);

    // Visa korten
    for (Spelkort spk : kortlek)
      System.out.println(spk);
  }
}
```

Kör testprogrammet!

Filer: Spelkort.java TestaSpelkort.java

Kompilera:

```
>>>javac Spelkort.java TestaSpelkort.java
      => Spelkort.class TestaSpelkort.class
```

Kör programmet:

```
>>>java TestaSpelkort   (klass med main-metod)
```

Utskrift i terminalfönstret ...

```
Hjärter ESS
Hjärter 2
Hjärter 3
Hjärter 4
Hjärter 5
Hjärter 6
Hjärter 7
Hjärter 8
Hjärter 9
Hjärter 10
Hjärter KNEKT
Hjärter DAM
Hjärter KUNG
Spader ESS
Spader 2
Spader 3
Spader 4
Spader 5
Spader 6
Spader 7
Spader 8
Spader 9
Spader 10
Spader KNEKT
Spader DAM
Spader KUNG
```

```
Ruter ESS
Ruter 2
Ruter 3
Ruter 4
Ruter 5
Ruter 6
Ruter 7
Ruter 8
Ruter 9
Ruter 10
Ruter KNEKT
Ruter DAM
Ruter KUNG
Klöver ESS
Klöver 2
Klöver 3
Klöver 4
Klöver 5
Klöver 6
Klöver 7
Klöver 8
Klöver 9
Klöver 10
Klöver KNEKT
Klöver DAM
Klöver KUNG
```

. . . fast det blir bara en kolumn

Patienskort ärver från Spelkort

Korten har framsidan eller baksidan "synlig".

```
class Patienskort extends Spelkort {

    boolean rattvand;

    Patienskort (String f, int v, boolean rv) {
        super(f,v);
        rattvand = rv;
    }

    void vand () {
        rattvand = !rattvand;
    }

    public String toString() {...}
}
```

Konstruktorn för Patienskort

```
Patienskort (String f, int v, boolean rv) {
    super(f,v);
    rattvand = rv;
}
```

```
super(f,v);
```

$\Longrightarrow$

Konstruktorn i superklassen `Spelkort` anropas

Anrop av superklassens konstruktor görs alltid först
i konstruktorn i en subklass
Om det explicita anropet utelämnas så görs automatiskt ett anrop
av en parameterlös konstruktor: super ();

toString-metoden för Patienskort

```
public String toString() {
    if (rattvand)
        return super.toString();
    else
        return "BAKSIDA";
 }
```

```
super.toString();
```

$\Longrightarrow$

```
toString()
```

i superklassen `Spelkort` anropas

Litet kodexempel med Patienskort

```
Patienskort pk =
          new Patienskort("Spader",12,true);
System.out.println(pk);
pk.vand();
System.out.println(pk);
pk.vand();
System.out.println(pk);
```

Första utskriften ger `Spader DAM`

Andra utskriften ger `BAKSIDA`

Tredje utskriften ger `Spader DAM` igen

Testa Patienskortsklassen:

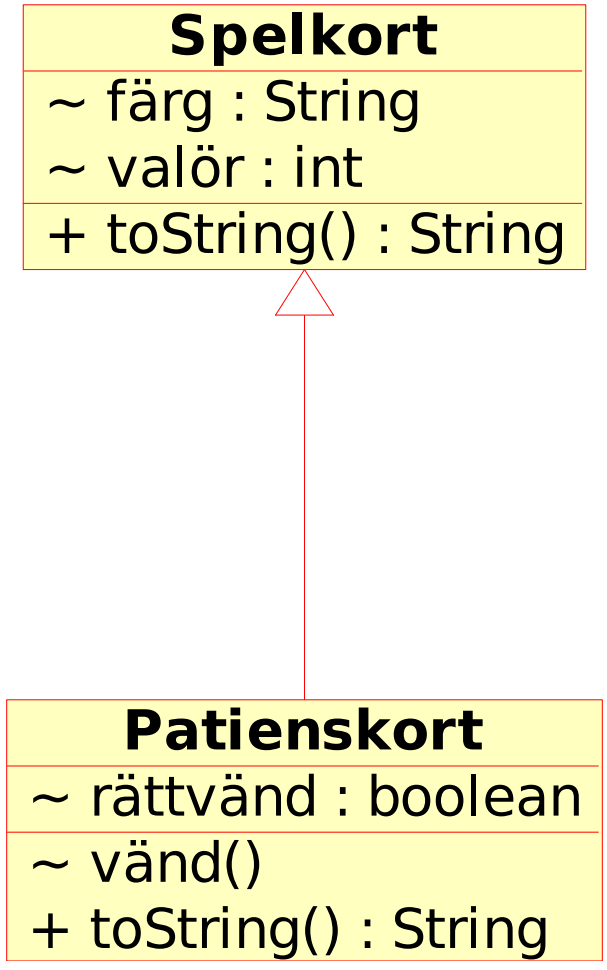
```
class TestaPatienskort {
  public static void main (String[] arg) {
    String[] farger =
      {"Hjarter", "Spader", "Ruter", "Klover"};
    // Skapa vektor med plats for 52 Patienskort.
    Patienskort[] kortlek = new Patienskort[52];

    // Skapa korten
    int kortnr = 0;
    for (String farg : farger)
      for (int valor = 1; valor<=13; valor++)
        kortlek[kortnr++] =
          new Patienskort(farg, valor, true);

    // Visa korten
    for (Patienskort pk : kortlek)
      System.out.println(pk);
  }
}
```

UML

Standard för grafiska beskrivningar av olika aspekter av objektorienterade program.

- class diagram
- use case diagram
- statechart diagram
- sequence diagram
- activity diagram
- collaboration diagram
- component diagram
- deployment diagram

UML för Spelkort och Patienskort

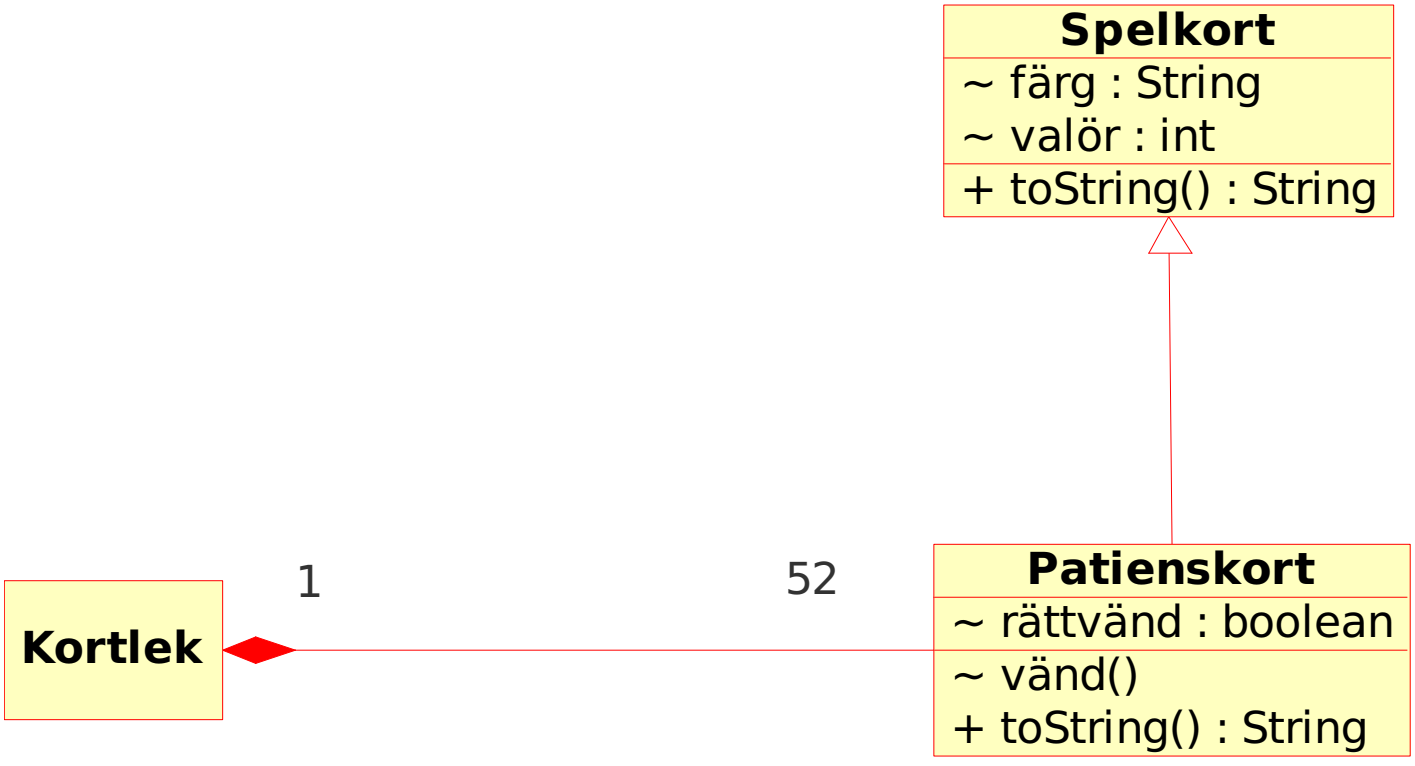



```
class Patienskort extends Spelkort {
    ...
}
```

Klassen Kortlek

```
class Kortlek {
    Patienskort[] lek = new PatiensKort[52];

    Kortlek () {
        // Skapa 52 kort som vi gjorde tidigare
        // i testprogrammet
    }

    void blanda () { ... }  // ej i
                            // UML-diagrammet!
    // fler metoder
}
```

UML för Kortlek, Spelkort och Patienskort




```
class Kortlek {
   Patienskort[] kortlek = new PatiensKort[52];
      ...
}
```

Grafiska program i Java

- Använd grafik-bibliotek (awt, swing)
- Välj en ”fönsterklass” som superklass
- Skriv subklass
- Fyll ev. med andra grafiska objekt
  (ev. objekt av subklasser)

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, avvecklad
  - ScrollPane
  - Window fristående fönster
    - Dialog används i
    - Frame fristående program
- Scrollbar

Första grafiska programmet

- Vi använder awt med Frame som fönsterklass
- Awt används för att programmen ska bli korta och enkla.
- Från nästa föreläsning och på labbarna kommer vi att använda swing för grafiken.

Visa fönster på skärmen, ingen subklass

```
import java.awt.*;
class AwtDemo {
  public static void main (String[] args) {
    Frame f = new Frame();
    f.setSize(300 ,300);
    f.setVisible(true) ;
    f.setBackground(Color.magenta);
  }
}
```

- Utan f.setSize(h,w) blir fönstret pyttelitet men kan dras till önskad storlek
- Utan f.setVisible(true) blir fönstret osynligt
- Utan f.setBackground(...) blir fönstret grått

Likadant fönster, med subklass

```
import java.awt.*;
class AwtDemo1 extends Frame {
  AwtDemo1 ( ) {
    setSize (300 ,300);
    setVisible(true);
    setBackground(Color.green);
  }

  public static void main (String[] u) {
    AwtDemo1 window = new AwtDemo1();
  }
}
```

Lägg till en knapp

```
import java.awt.*;
class AwtDemo2 extends Frame {
  Button b = new Button("PLEASE PRESS");
  AwtDemo2 ( ) {
    setSize (300 ,300);
    setVisible(true);
    setBackground(Color.cyan);
    add(b);
  }

  public static void main (String[] u) {
    AwtDemo2 window = new AwtDemo2();
  }
}
```

Interface

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

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

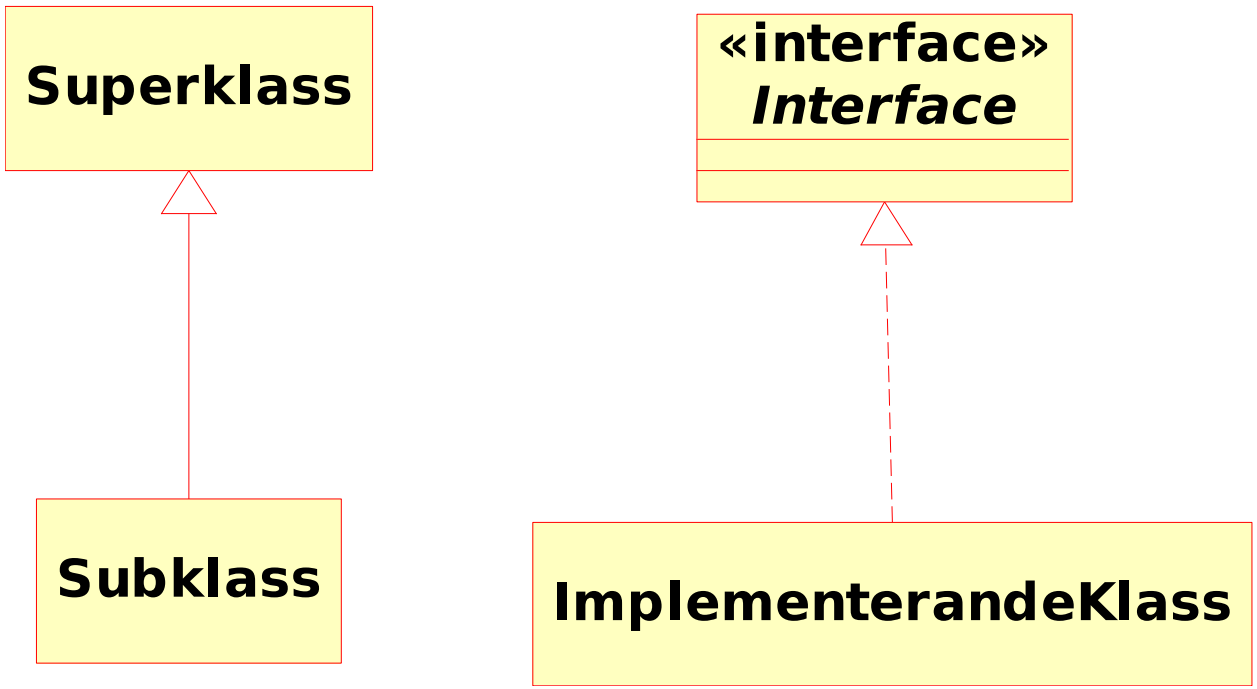
Ytterligare en sorts Monster: ScaryMonster

```
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

UML för arv och interface



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`

Alla som implementerar Monster har typen Monster

Alla som implementerar Monster passar där typen Monster krävs

Objekt av CookieMonster har typerna CookieMonster och Monster

Objekt av ScaryMonster har typerna ScaryMonster och Monster

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 $\Rightarrow$ lyssnarklass $\Rightarrow$ *metod*
- Koppla objekt av lyssnarklassen till *b*

Ett tryck på knappen medför att *metod* anropas

Javas lyssnarinterface

Flera lyssnarinterface finns

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.

Koppla lyssnare

Grafiska objekt som kan detektera händelser har metod för att koppla lyssnare. Om b är en knapp:

b.addActionListener (...);

Som parameter ges ett objekt som implementerar lyssnarinterfacet, här

ActionListener

Java-systemet detekterar händelse och anropar interfacets metod, här

actionPerformed()

Metoden actionPerformed()är definierad av programmeraren

Fönstret med knapp men utan händelse

```
import java.awt.*;
class AwtDemo2 extends Frame {
  Button b = new Button("PLEASE PRESS");

  AwtDemo2 ( ) {
    setSize (300 ,300);
    setVisible( true ) ;
    setBackground ( Color.cyan ); add(b);
  }

  public static void main (String[] u) {
    AwtDemo2 window = new AwtDemo2();
  }
}
```

Ge knapptryckningen betydelse

- Gör en klass till lyssnarklass implements Actionlistener
  implements ActionListener
- Definiera interface-metoden public void actionPerformed (...)
  här skrivs vad som ska ske vid knapptryckning.
- Koppla lyssnarobjekt till knappen b.addActionListener(...)

Vid knapptryckning kommer Javasystemet att anropa
`actionPerfromed()`

Vi väljer här att göra fönsterklassen till lyssnarklass

Knapptryckningen byter bakgrundsfärg till blå så endast första trycket har effekt.

```
import java.awt.*;
import java.awt.event.*;
class AwtDemo3 extends Frame implements ActionListener {
  Button b = new Button("PLEASE PRESS");
  AwtDemo3 ( ) {
    setSize (300 ,300);
    setVisible (true );
    setBackground ( Color.cyan );
    add(b);
    b.addActionListener(this);
  }

  public void actionPerformed(ActionEvent e) {
    b.setBackground(Color.blue);
  }

  public static void main (String[] u) {
    AwtDemo3 window = new AwtDemo3();
  }
```

Vi bygger upp programmet
stegvis under föreläsningen.