


Bridge

Alternativ till observer
(model/view)

UI – data



- n Särskilt viktigt att hålla isär tre huvuddelar i ett program


1. Hantering av input från användaren
2. Presentation av output för användaren
3. Hantering av interna data

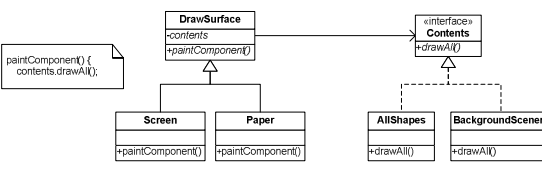
} UI

- T.ex. lista av alla figurer som har skapats

Objektorienterad programmering och Java 2


Bridge (UI – Modell)





Objektorienterad programmering och Java 3

Model-View-Controller



Controller

```

updateModel() {
  contents.updateContents();
  displayOutput.repaint();
}

GetInput() {
  contents
  displayOutput
  updateModel()
}

GetRemoteInput() {
  updateModel()
}
            
```

Model

```

interface Contents {
  updateContents()
  drawAll()
}

AllComponents() {
  drawAll()
}

BackgroundScenery() {
  drawAll()
}
            
```

View

```


paintComponent() {
  contents.drawAll();
}

DisplayOutput() {
  contents
  paintComponent()
}

Screen() {
  paintComponent()
}

Paper() {
  paintComponent()
}
            
```


Objektorienterad programmering och Java 4



State

Objektorienterad programmering och Java 5

Dynamisk "ändring av typ"



- n Java starkt typat
 - Objekt kan inte byta typ under programkörning
 - Kan inte börja bete sig radikalt annorlunda
- n Men, det vore användbart att kunna ändra typ:
 - Spelare
 - § Kan skadas
 - § Få extra krafter
 - Spelet
 - § Kan komma in i en annan fas
 - § Komma in på högre nivå (högre tempo)

Objektorienterad programmering och Java 6

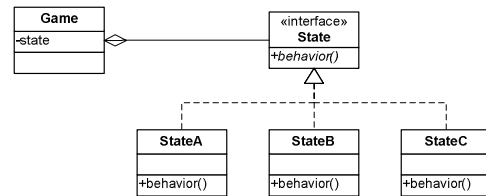
Designprincip

- n Skilj ut den dynamiska delen i objektet och lägg det i särskilt roll/state-objekt
- n Kan sedan byta objektets egenskaper genom att byta referens till ett nytt roll/state-objekt

Objektorienterad programmering och Java

7

State



Objektorienterad programmering och Java

8

Strategy

Objektorienterad programmering och Java

9

Dynamisk styrning

- n `Collection.sort();`
- n Förutsätter att elementen är naturligt jämförbara
 - Klassen `MinaElement` (eller vad man kallar sina element) måste implementera gränssnittet `Comparable<MinaElement>`

Objektorienterad programmering och Java

10

Exempel Comparable

```

public class Fil implements Comparable<Fil> {
    String namn;
    int storlek;

    public Fil(String namn, int storlek) { ... }

    public int compareTo(Fil f) {
        if (this.storlek > f.storlek) {
            return 1;
        } else if (this.storlek < f.storlek) {
            return -1;
        } else {
            return 0;
        }
    }
}
  
```

Objektorienterad programmering och Java

11

Sortering

- n Vill dynamiskt kunna ändra hur sorteringsfunktionen utförs
- n T.ex. enligt vilka kriterier en lista av studenter ska sorteras
 - I vissa delar av programmet vill jag ha listan sorterad på namn
 - Ibland sorterad efter studiepoäng

Objektorienterad programmering och Java

12

Ful lösning

- Sorteringsalgoritm där vi styr jämförelsen med en flagga sortKriterium

```

if (sortKriterium.equals("namn")) {
    c = compString(eA.hämtaNamn(), eB.hämtaNamn());
} else if (sortKriterium.equals("poäng")) {
    c = compInt(eA.hämtaPoäng(), eB.hämtaPoäng());
}
if (c > 0) { // A måste läggas före B i sorterade listan
    
```

Objektorienterad programmering och Java 13

Varför ful?

- Varje gång vi vill ändra sorteringskriterium måste vi ändra djupt inne i sorteringsalgoritmen
- Sorteringskriteriet tillhör egentligen det som ska sorteras (inte sorteringsalgoritmen)
 - Man kan t.ex. inte bestämma ett nytt kriterium utan att veta hur det som ska sorteras ser ut

Objektorienterad programmering och Java 14

Varför ful (forts)?

- Just nu har vi dubbelriktat beroende
 - Sorteringsalgoritmen måste känna till det som sorteras
 - Vilka kriterier som kan tillämpas
 - Det som ska sorteras måste kunna anropa sorteringsalgoritmen

Objektorienterad programmering och Java 15

Snygg lösning

- Skapa en arvshierarki med de olika sorteringskriterierna
- Klienten (i vårt fall sorteringsalgoritmen) bryr sig inte om typ av kriterium
 - Olika instanser kan läggas i en generell variabel
 - Rätt sorteringskriterium kommer att tillämpas utifrån vilket objekt som ligger i variabeln

Objektorienterad programmering och Java 16

Dynamiskt val av strategi

```

Strategy str = new ConcreteStrategyA();
Collection.sort(myList, str);

public void sort(List l, str) {
    if (str.algorithm(eA, eB) > 0) {
    
```

Objektorienterad programmering och Java 17

Exempel: Comparator

```

Comparator comparator = new CompareName();
sort(klassLista, comparator);
    
```

Objektorienterad programmering och Java 18

Exempel Comparator



```
public class JfrFilnamn implements Comparator<Fil> {
    //konstruktor, returnerar ett objekt med inkapslad metod public JfrFilnamn() {
    }

    public int compare(Fil f1, Fil f2) {
        if (f1.hämtaNamn() > f2.hämtaNamn()) {
            return 1;
        } else if (f1.hämtaNamn() < f2.hämtaNamn()) {
            return -1;
        } else {
            return 0;
        }
    }
}
```

Objektorienterad programmering och Java

19

Spel i Java

Översikt



- n Övergripande design
- n Användning av state
- n Observer
 - Timer
- n Måla upp egna komponenter
- n Att ta tiden
- n Kollisionsdetektion
- n Demo av exempelkod

Objektorienterad programmering och Java

21

Dataspel



- n Värld
- n Aktörer
 - Mer eller mindre självständiga
- n Tidsflöde

Objektorienterad programmering och Java

22

Dataspellets delar



- n Grafik
- n Intern representation av alla aktörer
- n Uppdateringsloop
 - Dvs. rita upp alla aktörer (precis som i ritverktyget)
- n Inmatning

Objektorienterad programmering och Java

23

Aktörerna: Sprites



- n Sprites betyder "älva"
- n 2D-figurer
- n T.ex. transparent .gif-bild
- n Använd t.ex. Gimp för att skapa bilderna
- n Använd ImageIcon för bildinläsning

Objektorienterad programmering och Java

24

Exempel: imageIcon.paintIcon()



```
public class Tree extends Sprite {
    private ImageIcon icon = new ImageIcon("C:/Users/ritko/Documents/figures/tree.gif");

    public Tree(int x, int y) {
        super(x, y);
        myState = new Stationary();
    }

    public void draw(Graphics g) {
        icon.paintIcon(null, g, (int) myPos.getX(), (int) myPos.getY());
    }
}
```

Exempel: g.drawImage()



```
public class Tree extends Sprite {
    private ImageIcon icon = new ImageIcon("C:/Users/ritko/Documents/figures/cactus.gif");
    private Image img = icon.getImage();

    public Tree(int x, int y) {
        super(x, y);
        myState = new Stationary();
    }

    public void draw(Graphics g) {
        g.drawImage(img, (int) myPos.getX(), (int) myPos.getY(), null);
    }
}
```

Exempel: Egna figurer



```
public class Tree extends Sprite {

    public Tree(int x, int y) {
        super(x, y);
        myState = new Stationary();
    }

    public void draw(Graphics g) {
        g.setColor(GameColors.TREE_TRUNK);
        g.fillRect((int) myPos.getX(), (int) myPos.getY(), width, height);
        g.setColor(GameColors.TREE_CROWN);
        g.fillOval((int) (myPos.getX()-(height/2-width/2)), (int) (myPos.getY()-height+10),
            height, height);
    }
}
```

Graphics2D



- n Detta är egentligen subclass till Graphics
- n Måste göra downcast för att få tillgång till Graphics2D protokollet

```
public void draw(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    g2.rotate(...);
}
```

Exempel Graphics2D



```
public void draw(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    //spara nuvarande transformation
    AffineTransform transf = g2.getTransform();
    // Ange rotation för utritning
    g2.rotate(Math.toRadians(45), (int) myPos.getX(), (int) myPos.getY());
    // rita roterat
    g2.setColor(Color.RED);
    g2.fillRect((int) myPos.getX(), (int) myPos.getY(), width, height);
    // återställ transformen
    g2.setTransform(transf);
}
```

Kollisionsdetektering

Kollisionsdetektering



- n Använd omgivande rektanglar
- n Klassen Rectangle

```
Rectangle rectangle = new Rectangle();
if (rectangle.intersects(rectangle2) {
    ...
}

if (r.contains(point) {
    ...
}
```

Objektorienterad programmering och Java

31

Löpa igenom alla sprites



- n Kan vara problematiskt om man vill ta bort element i en lista som man håller på att löpa igenom
 - ConcurrentModificationException
- n Använd iteratorn för att lägga till och ta bort från listan (t.ex iter.add() och iter.remove())
 - **Ändra inte på något annat sätt i listan efter att du fått en iterator i handen!**

Objektorienterad programmering och Java

32

Exempelkod iterator



```
Iterator < Sprite> itr = allTrees.iterator();
while (itr.hasNext()) {
    if (itr.next().getX() < -100) {
        itr.remove(); // destruction of trees outside the playArea
    }
}
```

Objektorienterad programmering och Java

33

Slumptal



```
import java.util.Random;

Random rand = new Random();
width = (int) (rand.nextDouble()*20 + 15);
height = (int) (rand.nextDouble()*100 + 50);
```

Objektorienterad programmering och Java

34

Demo av exempelkod

Timer

Uppdatering



- n Ser till att alla aktörer (sprites) agerar (t.ex. rör sig; ett snäpp för varje timersignal)
- n **Använd inte while-loop!**
 - While loop bara för att löpa igenom alla figurer
 - Inte för att vänta på rätt ögonblick att rita upp!
- n Använd klocksignal
 - Tidtagarur (timer)
 - Lyssnare som sedan anropar `model.update();`
`display.repaint();`

Objektorienterad programmering och Java

37

Timer



- n Ett objekt kan registrera sig hos en timer som kommer att generera händelser av typen `ActionEvent` (jfr `MouseEvent`)

Objektorienterad programmering och Java

38

Timer



```
public class ListenToTimer extends Listen implements ActionListener {
    private javax.swing.Timer timer;

    public ListenToTimer(World world, Display display, int timestep) {
        ...
        timer = new javax.swing.Timer(timestep, this);
        timer.start();
    }

    public void actionPerformed(ActionEvent e) {
        myWorld.updateWorld();
        myDisplay.repaint();
    }
}
```

Objektorienterad programmering och Java

39

Att ta tiden

Hur länge en knapp hålls nertryckt



```
public void keyPressed(KeyEvent ev) {

    switch (key) {
    case KeyEvent.VK_UP:
        timePressed = System.currentTimeMillis();
        break;
    }
}
```

Objektorienterad programmering och Java

41

Hur länge en knapp hålls nertryckt



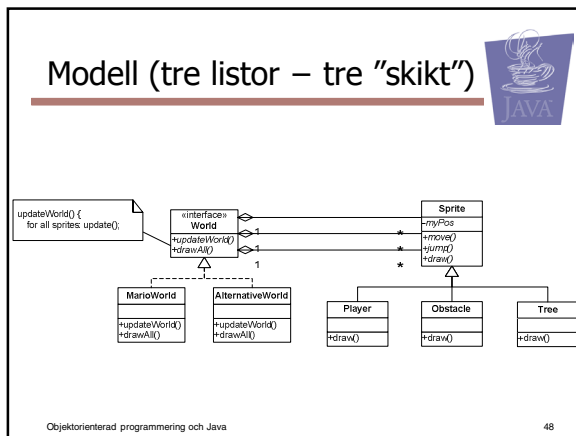
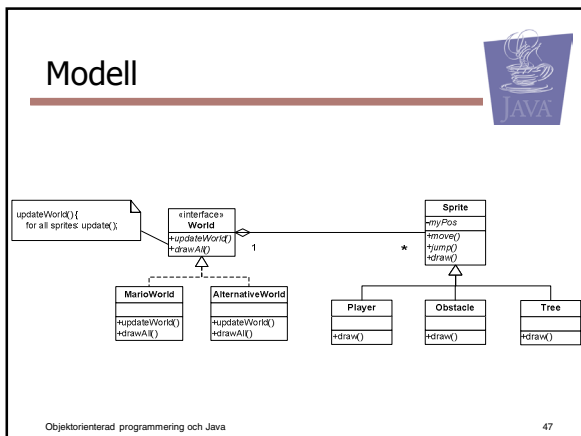
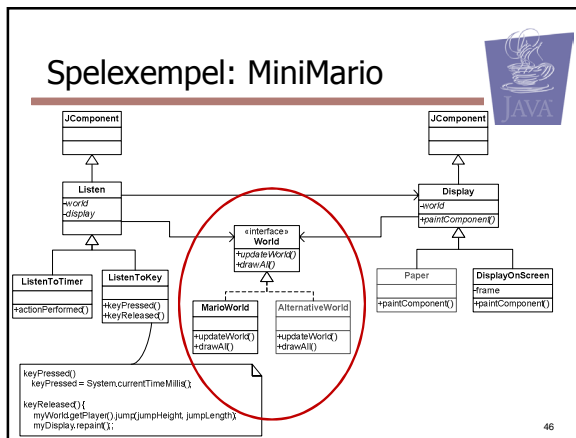
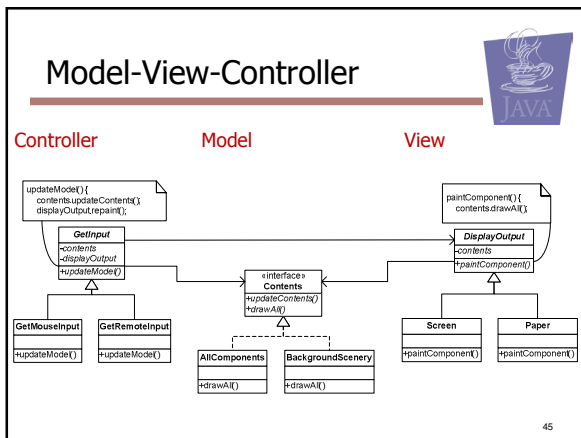
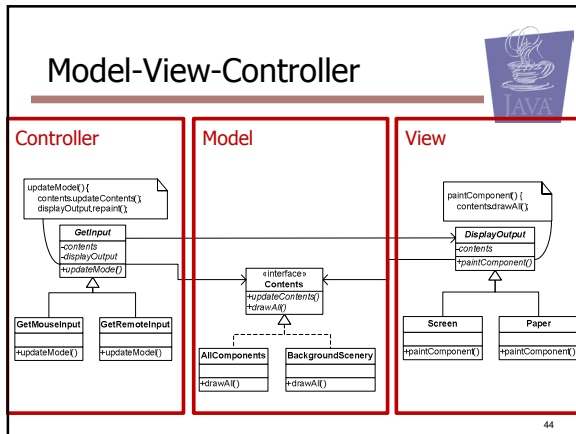
```
public void keyReleased(KeyEvent ev) {

    switch (key) {
    case KeyEvent.VK_UP:
        timeReleased = System.currentTimeMillis();
        timeHeld = (int) timeReleased - timePressed;
        break;
    }
}
```

Objektorienterad programmering och Java

42

Övergripande design



MarioWorld.updateWorld()

```

public void updateWorld() {
    updateObstacles();
    updateTrees();
    for (Iterator<Tree> itr = allTrees.iterator(); itr.hasNext();) {
        itr.next().move();
    }
    for (Iterator<Obstacle> itr = allObstacles.iterator(); itr.hasNext();) {
        itr.next().move();
    }
    for (Iterator<Sprite> itr = allSprites.iterator(); itr.hasNext();) {
        itr.next().move();
    }
}
    
```

Objektorienterad programmering och Java 49

MarioWorld.drawAll()

```

public void drawAll(Graphics g) {
    for (Iterator<Tree> itr = allTrees.iterator(); itr.hasNext();) {
        itr.next().draw(g);
    }
    for (Iterator<Obstacle> itr = allObstacles.iterator(); itr.hasNext();) {
        itr.next().draw(g);
    }
    for (Iterator<Sprite> itr = allSprites.iterator(); itr.hasNext();) {
        itr.next().draw(g);
    }
}
    
```

Objektorienterad programmering och Java 50

Hus State kan användas

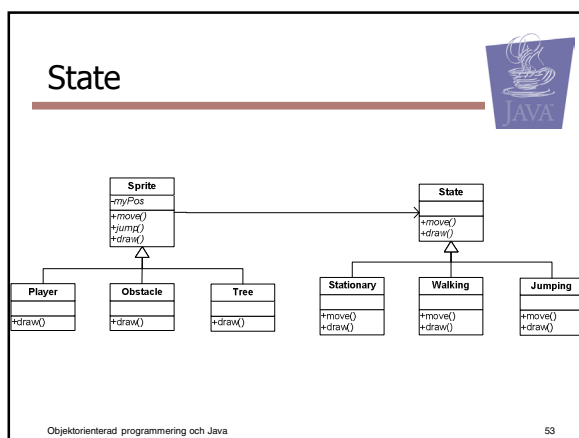
Delegering till State

- n Alla sprites har ett tillstånd
 - T.ex Walking
 - T.ex. Jumping

Dessa funktioner sköts av State:

- n Förflyttning
 - move()
- n Uppritning
 - draw()

Objektorienterad programmering och Java 52



Kodexempel

```

public void jump(double height, int duration) {
    if (!(myState instanceof Jumping)) {
        myState = new Jumping(myPos.getY(), height, duration);
        myState.move(myPos);
    }
}
    
```

Objektorienterad programmering och Java 54

