Mer om klasser och objekt, Animering

Vecka 4, Bildserie 2

Innehåll

- Metoder med sidoeffekter
- Likhet
- Initiering av objekt
- Array:er med objekt
- Tangentlyssnare
- Animering

Att Läsa i Boken

- 9.11
- 15.1-15.2 (översiktligt, vi gör inte som i boken men principen är densamma)

Boken använder JavaFX för grafik vi använder Java 2D

Metoder med Sidoeffekter

```
// Intuitively 0 ... (?)
out.println( o.m(0) - o.m(0) );
```

Tidigare haft uttryck med sidoeffekter

- Samma fenomen kan uppträda för instansmetoder.
- Innebär i vårt fall att metoden förändrar en instansvariabel i objektet

Referetiell transparens (? svenska begrepp saknas, referential transparency)

- Enkelt sagt: Givet samma indata, får man alltid samma utdata från metoden?
  - Om så är det lättare att resonera om program (korrekthet)
- Metoder med sidoeffekter innebär att programmet inte blir referetiellt transparent
  - Kan inte undvikas i imperativ programmering, …
  - … men man kan försöka minimera
  - Metoder med returvärden undviker att ändra instansvariabler
  - void-metoder ändrar ofta instansvariabler, men metoderna är inga uttryck, vi får inget värde..
- Försök alltid att skriva metoder som bara tar indata och returnerar utdata …
  - …om tvunget, använd instansvariabler.
- Du behöver instansvariabler om objektet måste komma ihåg något mellan metodanropen.

Likhet

```
class Person {
     final int id;
     String name;
     int age;

     public Person(int id, String name, int age) {
          this.id = id;
          this.name = name;
          this.age = age;
     }
     // This is simplified, not to use in a real application
     boolean equals(Person other) {
          if (this == other){
               return true;
          }
          if (other == null ){
               return false;
          }
          return id == other.id;
     }
}
```

Vilka objekt av en viss typ (klass) som skall räknas som “lika” måste vi själva bestämma.

- Använder vi == gäller referenssemantik (referenserna jämförs)
  - Dvs. vi får identitet
- Vill vi ha någon annan typ av likhet kan vi skapa en boolesk metod equals()
  - Ofta används ett attribut som måste vara unikt för alla objekt (ett id-nummer)
  - Förvånansvärt krångligt att få till, mer senare eller i senare kurser …
  - … bilden är bara ett förenklat exempel på hur man kan implementera likhet (för denna kurs).
- Desutom: Alla objekt har en färdig equals-metod (osynlig i koden)
  - Denna fungerar dock som ==, d.v.s identitiet.

Initiering av Objekt

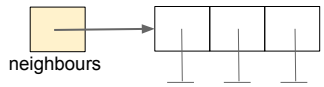
```
class MyClass {
    int i1 = 4;
    int i2 = i1;
    int i3;
    MyClass() {
        i3 = i1 + i2;
    }
}
MyClass m = new MyClass();
out.println(m.i1);    // 4
out.println(m.i2);    // 4
out.println(m.i3);    // 8
```

Objekt initieras enligt (förenklat):

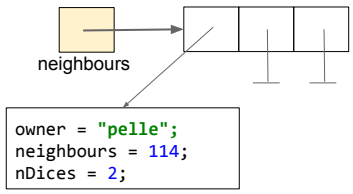
- Instansvariabler i skriven ordning …
- … därefter körs konstruktorn.

Arrayer med Objekt

```
Country[] neighbours = new Country[3];
```




```
neighbours[0] = new Country("pelle", 2, 114);
```




```
String owner = c1.neighbours[0].owner; // "pelle"
```

Arrayer kan skapas utifrån godtyckliga typer, även referenstyper (har tidigare sett String-array) …

- … och därmed typer vi själva skapat.
  - En array med Country t.ex.
- OBS! Att arrayen inte innehåller några objekt direkt efter deklarationen, den innehåller bara referensvariabler med värdet null.

För att komma åt medlemmar kombineras punktnotation och indexering

- För att komma åt ett grannlands ägare, läs v -> h (se bild)
  - c1 är ett objekt alltså "." för att komma åt alla grannar, vi får då …
  - …en array, alltså […], för att komma åt en specifik granne, vi får då ett …
  - …objekt, alltså "." för att komma åt ägaren.

Programmering

```
Complex[] ca1 = {new Complex(1, 2), new Complex(6, -1)};
Complex[] ca2 = {new Complex(1, -1), new Complex(2, -1)};

Complex[] result = add( ca1, ca2 );

// Output  [ 2 + i, 8 - 2i ]
out.println(Arrays.toString(result));
```

Skriv metoden add!

Objektorientering

```
class Country {
    Country[] neighbours;
    ...
    Country(String name, Player owner, int nDices) {
        this.name = name;
        this.owner = owner;
        ...
    }
    ...
}

// Usage
Country c1 = new Country( …, new Player(...), 6 );
Country c2 = new Country( … );
Country c3 = new Country( … );
c1.neighbours = new Country[]{c2, c3};
```

En (bättre) mer objektorienterad version av Country.

- En land har (en lista med) grannar (inte en kodning av grannarna i from av ett heltal som vi haft tidigare)
- En stryka med objektorienterad programmering är att “koden” avbildar “verkligheten”.
    - Gör det lättare (mer naturligt) att strukturera och lösa problem

.

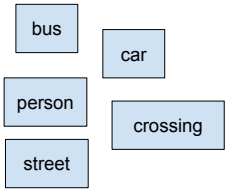
Ansvarsområden för klasser

En klass skall fånga ett koncept

- En klass skall ha ett ansvarsområde
- … på samma sätt som för metoder
- … annars blir de svåra att (åter)använda, felsöka, … det blir rörigt!
- Bättre att kombinera flera (små) tydliga klasser

Bilden: Så här vill vi inte att en klass skall fungera (för mycket olika saker)

Metodik



För att lösa problem m.h.a. objektorientering jobbar man **bottom-up**

- Man försöker hitta objekt som finns i problemet
  - Försöker avgöra vilka attribut och metoder objekten har.
- Man skapar klasser för objekten
- Utifrån klasserna skapar man Java-objekt som man försöker få att samverka
  - Hur detta skall gå till är inte helt klart från början
- D.v.s. vi börjar med delarna och försöker sätta samman dessa till en lösning, precis tvärtom emot funktionell nedbrytning (där vi börjar med helheten).
  - … men i praktiken använder man båda metoderna, beroende på situation.

Tangentordslyssnare

```
// Add a key listener
void initGraphics() {
   setPreferredSize(new Dimension(width, height));
     ...
     // Will create an objekt with method keyPressed
     window.addKeyListener(new KeyAdapter() {
       public void keyPressed(KeyEvent e) {
           handleKeys(e.getKeyCode());
       }
   });
}
void handleKeys(int keyCode) {    // Our listener
   if (keyCode == KeyEvent.VK_UP) {
       s.y++;
   }
   ...
}
```

Våra grafiska program kan behöva känna av att användaren trycker på någon tangent

- sc.nextInt() o.s.v. fungerar inte så bra i detta sammanhang.

Detta är lite tekniskt, detaljerna är inget man behöver kunna utantill

- Principen, vad som behövs och att det måste kopplas ihop skall du känna till.

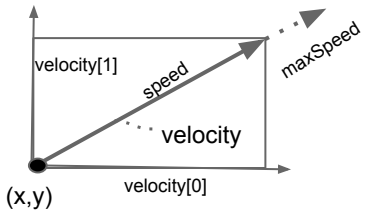
Analys av koden

- Vi skapar ett lyssnarobjekt (ett keyAdapter-objekt), med uttrycket new KeyAdapter(){ ….}
- Vi kopplar ihop objektet med window (addKeyListener), d.v.s. då vi klickar på fönstret kommer händelsesystemet att anrops lyssnarmetod keyPressed
    - Som parameter till metoden kommer det att skickas ett händelseobjekt, e.
    - Vi kan fråga objektet efter t.ex. tangentkoden för tryckt tangent
        - Koderna finns fördefinierade i en klass KeyEvent som Java tillhandahåller.
- Så fort vi fått tangentkoden anropar vi en egen metod.
- Nu är hela händelsestyrningen uppkopplad och klar, när vi trycker en

- tangent kommer programmet att “hoppa” till keyPressed() och köra koden i metoden.

Animering

```
class SpaceShip {
   final double maxSpeed;
   final double[] velocity;
   double x;
   double y;
  void move() {
     this.x += velocity[0];
     this.y += velocity[1];
  }

}
```



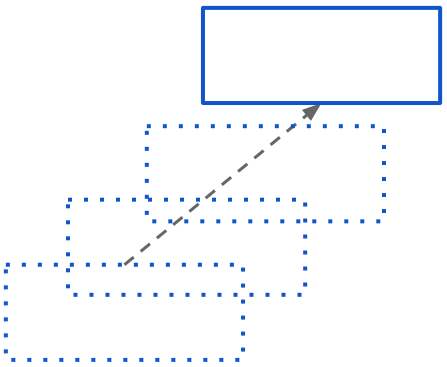
Objektet som skall röra sig i en 2d värld behöver instansvariabler för

- Position, x och y.
- Hastighet (velocity), d.v.s. en vektor som beskriver hastighet (= längden av vektorn) och riktningen
  - Om objektet bara rör sig i x- eller y-led behövs troligen inte velocity.
- Ofta används en maxhastighet som inte får överskridas av längden på velocity
  - Om objekt har konstant hastighet behövs inte detta.
- För att flytta objektet finns en metod move() som uppdaterar x och y utifrån hastigheten.

Animeringen drivs som tidigare av en Timer

- För varje “tick” (diskret tidssteg) uppdaterar vi världen och ritar om bilden av världen.

Programmering



Skriv ett grafiskt program som rör en rektangel över skärmen.

- Använd Java 2D.

OBS! Detta blir inte en optimalt snygg animering.

- Om man vill ha professionell kvalitet får man använda JavaFX eller lägga till icke-standard bibliotek

Kollisionsdetektering

```
class RainDrop {
   final double x;     // x and y for center
   double y;
   final double radius;
     ...
   Rectangle getBoundingBox() {
       int x = (int) (this.x - radius);
       int y = (int) (this.y + radius);
       int h = (int) (2 * radius);
       int w = (int) (2 * radius);
       return new Rectangle(x, y, w, h);
   }
}
// Usage
RainDrop rd1 ..., rd2 = ...;
if ( rd1.getBoundingBox().intersects(rd2.getBoundingBox()) {...}
```

Finns färdig användbar klass i Java, Rectangle.

- Låt klasser som skall ritas ut returnera en “bounding box” i form av en Rectangle.
- Använd metoden intersects()
- OBS! Måste anpassa så att man jämför skärmkoordinater annar stämmer inte bilden och
- logiken.

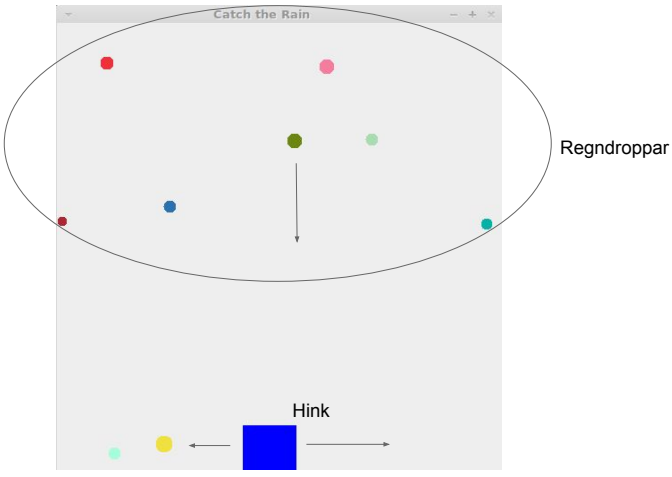
Mallen igen...

```
public class MyProgram {
     public static void main(String[] args) {
        new MyProgram().program();
    }

    void program() {
        int result = add( 1,2 ); // = this.add(1,2)
    }

    int add(int a, int b) {
        …
    }
```

Ytterligare ännu en analys av mallen

- Mallen är egentligen en klass med instansmetoderna program() och add()
  - main är inte en instansmetod, mer senare …
- I main metoden skapas ett objekt “av Mallen”
  - Hmm: Skapas main i main ?!?! …
  - … förklaring: main-metoden ligger inte i objektet (den finns i klass-objektet, se tidigare bild + mer senare)
- Därefter anropas metoden program() direkt på den returnerade referensen
  - Vi spar inte undan referensen vi får
  - Så fort metoden program() är klar kommer objektet att skräpsamlas (eftersom ingen referens finns sparad)

Inför Övning 4



Vi gör ett grafiskt program.

- Ett spel, … CatchTheRain
- Rör hinken höger/vänster för att fånga regndroppar.