Dagens föreläsning

Arrayer

och

klasser

Medan ni väntar:

Gå till m.voto.se/prog11 och svara på några gamla tentamensfrågor!

Arrayer

En *array* är en datatyp som representerar *flera* värden av samma typ.

Exempel:

värden	7	23	8	1	4	17	9	3	42
index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*

I Java är arrayer *objekt* som skapas med **new**.

Uttrycket `new int[9]`

skapar följande array-objekt

värden	0	0	0	0	0	0	0	0	0
index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*

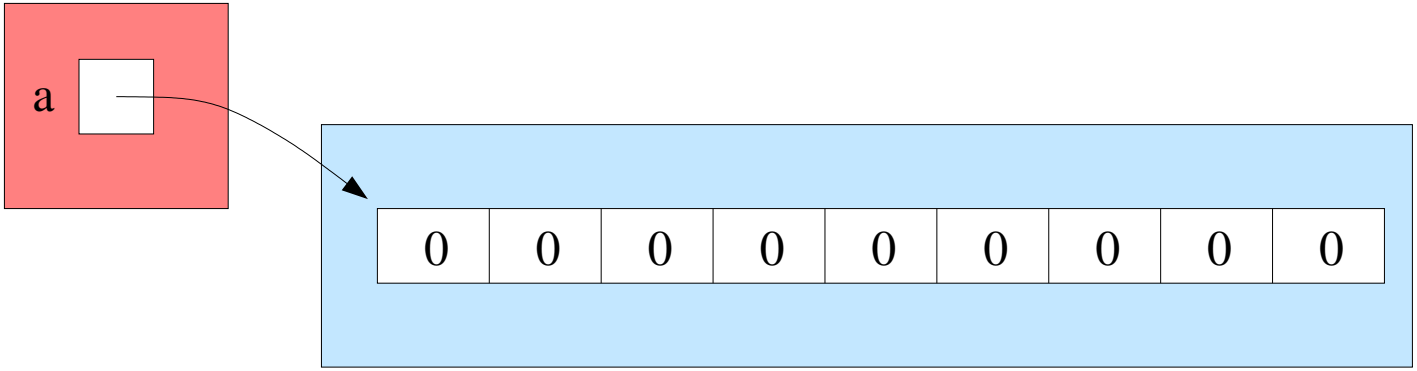
Arrayer

För att hålla reda på array-objekt användes referensvariabler.

Satsen

```
int[] a = new int[9];
```

skapar följande bild:

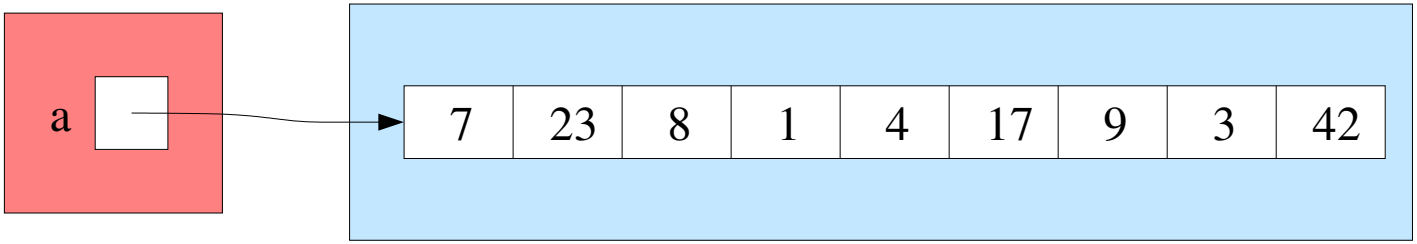


Arrayer

Det går att ge värden till elementen i samband med att det skapas. Satsen

```
int[] a = new int[]{7, 23, 8, 1, 4, 17, 9, 3, 42}
```

skapar det array-objekt och sätter `a` att referera det.



Skrivs dock vanligen (läs alltid) utan `new int[]` dvs så här

```
int[] a = {7, 23, 8, 1, 4, 17, 9, 3, 42}
```

Det är bara vid *deklarationer* av array-referenser man kan göra på detta sätt.

Arrayer

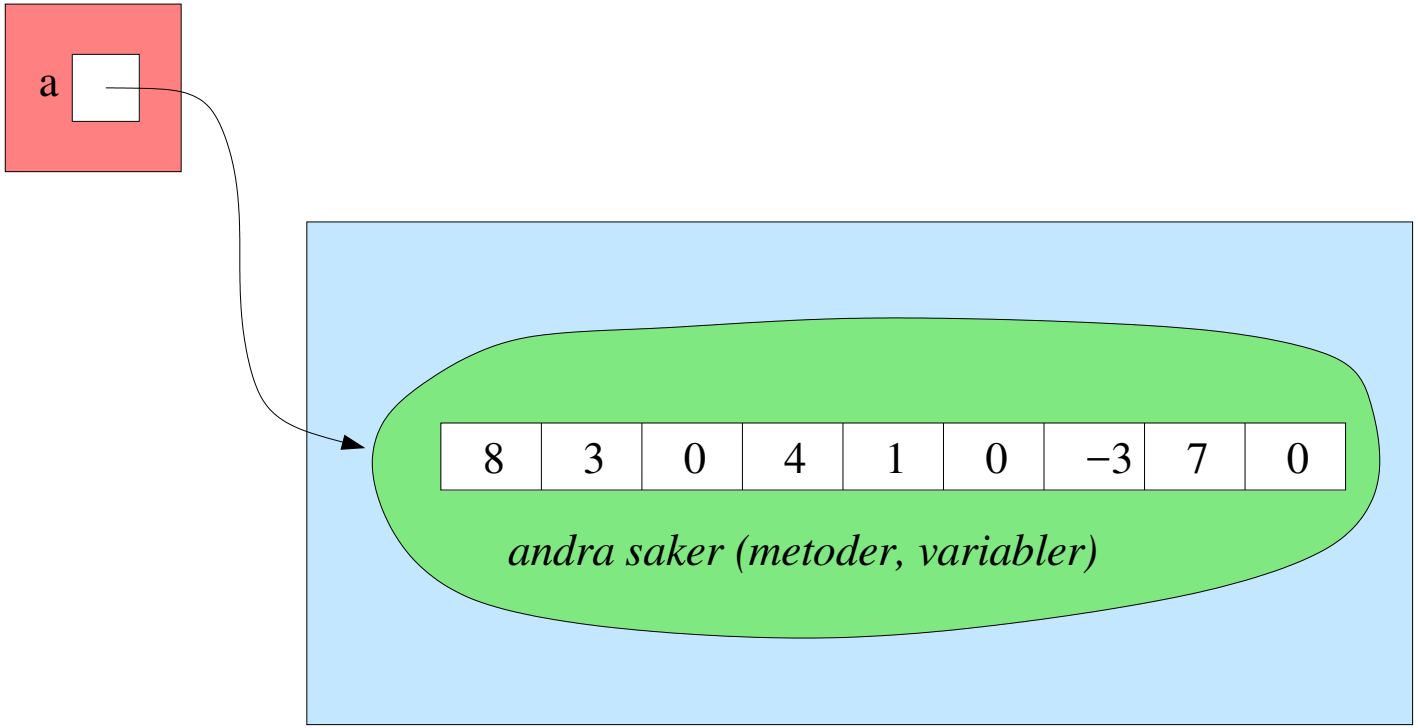
För att komma åt enskilda element i arrayen används *index* inom `[ ]`-parenteser.

Exempel:

```
a[0] = -47;
a[1] = 3;
a[2] = 5;
a[3] = a[1] + a[2];
a[a[1]] = (int)(Math.random()*1000);
a[(int)(Math.random()*9)] = 99;
char c = 'd';
a[c-'a']++;
```

Arrayer

Array-objekt innehåller fler saker än värden. En mer rättvisande bild är



Arrayer

Ett exempel på information som finns i array-objekten är attributet `length` (obs: *ej* en metod)

Exempel: Räkna antalet negativa tal i arrayen `a`:

```
int n = 0;
int i = 0;
while (i < a.length) {
   if (a[i] < 0) {
      n++;
   }
   i++
}
System.out.println("Antal negativa tal är " + n);
```

Sammanfattning av arrayers egenskaper

- En array innehåller 0 eller flera element.
- Alla element måste vara av samma typ.
- Elementen kan vara av vilken typ som helst inklusive objektreferenser och arrayer.
- Ett skapat array-objekt har en fix storlek som inte kan ändras men en array-referens kan tilldelas ett annat array-objekt med annan storlek.
- Attributet `length` innehåller arrayens storlek.
- Man använder index-operatorn `[ ]` för att referera enskilda element.
- Indexering med `[ ]` är en mycket effektiv operation.

for-satsen

För att iterera över elementen i en array används ofta **for**- i stället för **while**-satsen.

Exempel: Räkna antalet negativa värden i en array

```
int n = 0;
for (int i=0; i<a.length; i++) {
   if (a[i] < 0) {
      n++;
   }
}
```

```
int n = 0;
int i = 0;
while (i<a.length) {
   if (a[i] < 0) {
      n++;
   }
   i++;
}
```

Funktionen är identisk (men loopvariabeln `i` existerar inte efter **for**-loopen)

”for-each”-satsen

Exempel: Räkna antalet negativa element som ovan

```
int n = 0;
for (int i = 0; i < a.length; i++) {
   if (a[i] < 0) {
        n++;
   }
}
```

```
int n = 0;
for (int v : a) {
   if (v < 0) {
      n++;
   }
}
```

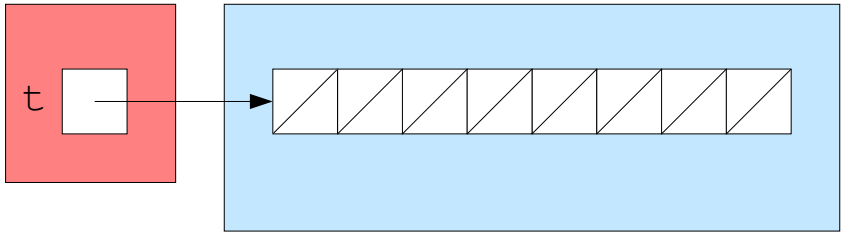
Observera att

- Variabeln `v` kommer successivt att anta *alla värden* som finns i arrayen
- Kommer inte åt några *index*

Arrayer kan lagra objektreferenser

Exempel: Array med paddor.

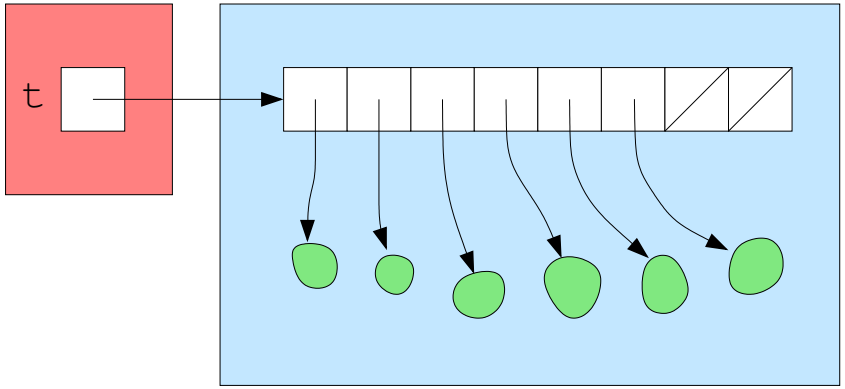
```
Turtle[] t = new Turtle[8];
```



Skapar en array men inga paddor. Ett snedstreck anger `null`-referens.

Vi kan skapa paddor på vanligt sätt och lägga in dem i världen:

```
for (int i = 0; i < 6; i++) {
   t[i] = new Turtle(w);
}
```

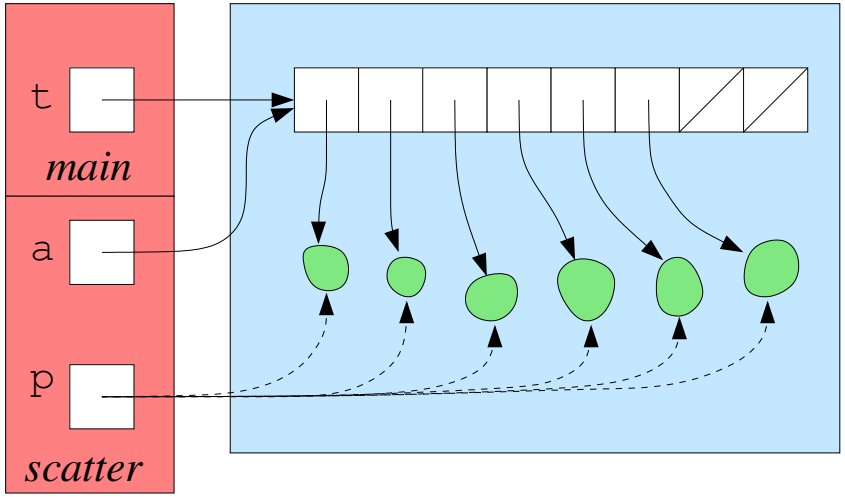


(Vi förutsätter att `w` refererar ett `World`-objekt)

Arrayer som parametrar

Antag att vi skapar en array med 6 paddor i `main` enligt ovan och anropar metoden `scatter` med arrayen `t` som parameter.

```
public void scatter(Turtle[] a) {
   int angle = 0;
   for (Turtle p : a) {
      p.turn(angle);
      p.move(150);
      angle += 60;
   }
}
```

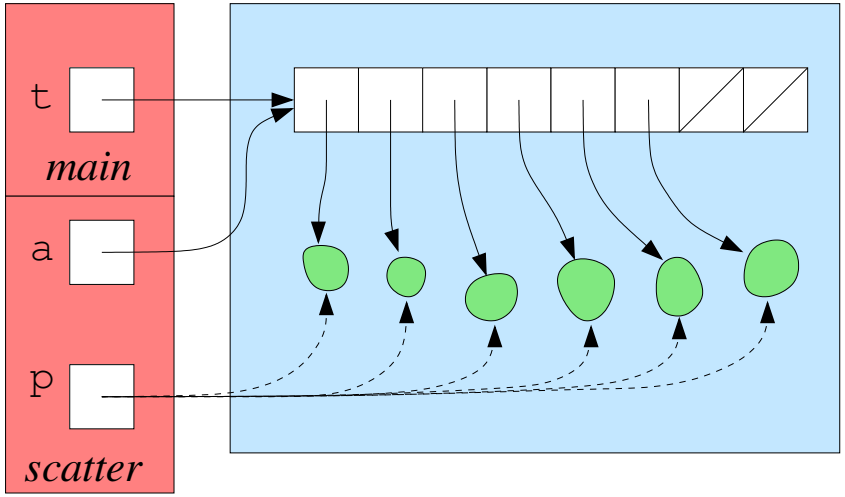


Anmärkning: Det är några problem i koden som vi återkommer till.

Arrayer som parametrar

Observera att metoden `scatter`

- skulle kunna ändra i `Turtle`-objekten,
- skulle kunna ändra i arrayen (t ex ta bort eller lägga till `Turtle`-objekt) men
- kan inte ändra på `t` eller någon annan variabel i `main`-metoden.



Vad är problemen i koden?

Testkör!

Demo TurtleDemo

- kompilerar inte

  Det går inte att anropa en *instansmetod* från en *klassmetod* utan att gå via en objektreferens.

- ”kraschar” när vi rättat kompileringsfelet.

  Går inte att anropa metoder för en `null`-referens.

Det andra felet dvs `NullPointerException` är ett av de vanligaste problemen ni kommer att råka ut för!

Metoder kan returnera arrayer

Exempel: Andragradsekvationen från förra föreläsningen

```
public static double maxRoot(double p, double q) {
  double d = (p*p - 4*q)/4;
  if (d < 0) {
    System.out.println("Complex roots");
    return Double.NaN;
  } else {
    d = Math.sqrt(d);
    double x1 = -p/2 + d;
    double x2 = -p/2 - d;
    if (x1 > x2) {
      return x1;
    } else {
      return x2;
    }
  }
}
```

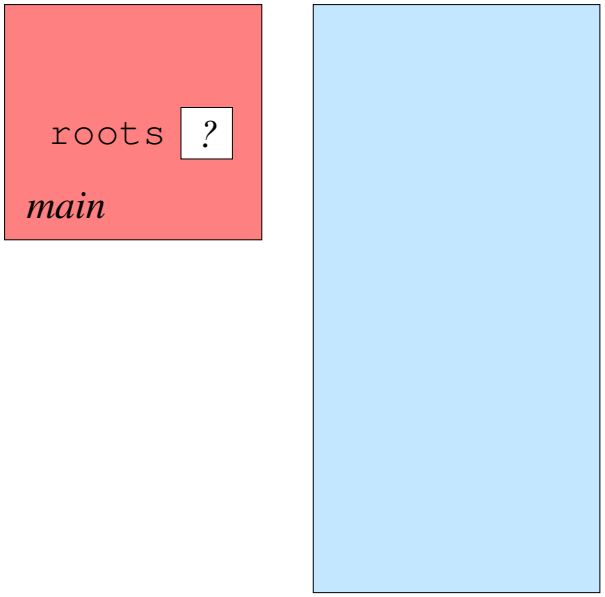
Nu kan vi låta metoden skapa och returnera en array med de två rötterna som element.

Kan returnera båda rötterna i en array

```
public static double[] quadEquation(double p, double q) {
  double d = (p*p - 4*q)/4;
  if (d < 0) {
    System.out.println("Complex roots");
    return null;
  } else {
    d = Math.sqrt(d);
    double[] r = new double[2];
    r[0] = -p/2 + d;
    r[1] = -p/2 - d;
    return r;
  }
}
```

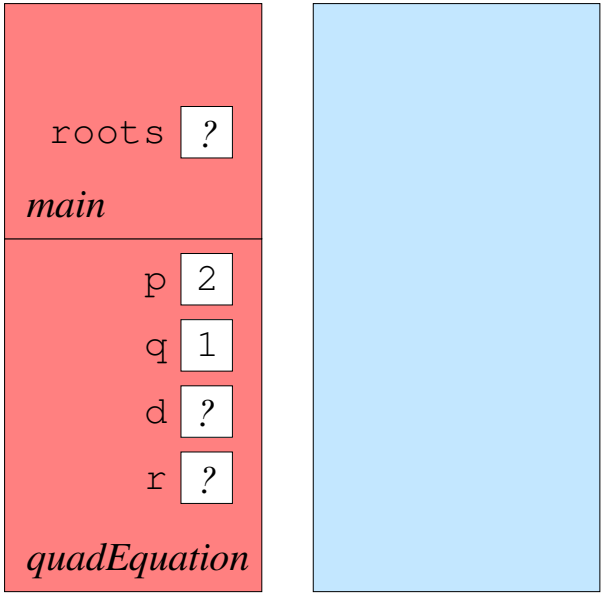
Vad händer vid anrop av quadEquation?

```
public static void main(String[] a) {
    double[] roots;
→
    roots = quadEquation(2, 1);
    ...
}
```



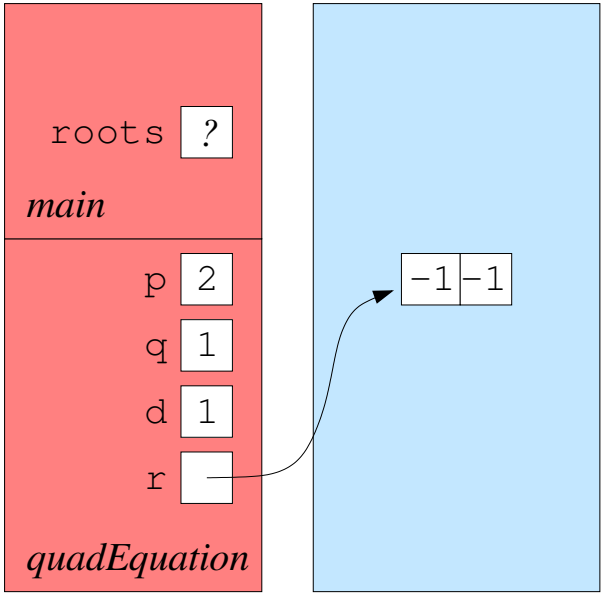
Vad händer vid anrop av quadEquation?

```
public static void main(String[] a) {
    double[] roots;

    roots = quadEquation(2, 1);
    ...
}


public static double[] quadEquation(double p,
                                    double q) {
→    double d = (p*p - 4*q)/4;
     if (d < 0) {
       System.out.println("Complex roots");
       return null;
     } else {
       d = Math.sqrt(d);
       double[] r = new double[2];
       r[0] = -p/2 + d;
       r[1] = -p/2 - d;
       return r;
     }
}
```



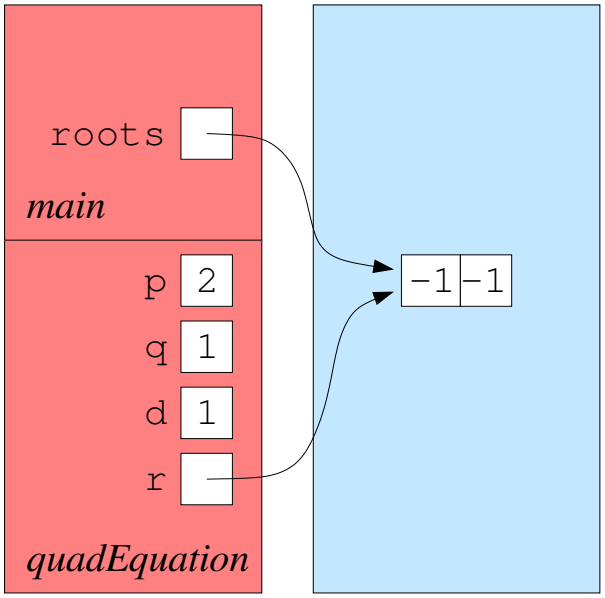
Vad händer vid anrop av quadEquation?

```
public static void main(String[] a) {
    double[] roots;

    roots = quadEquation(2, 1);
    ...
}


public static double[] quadEquation(double p,
                                    double q) {
      double d = (p*p - 4*q)/4;
      if (d < 0) {
        System.out.println("Complex roots");
        return null;
      } else {
        d = Math.sqrt(d);
        double[] r = new double[2];
        r[0] = -p/2 + d;
        r[1] = -p/2 - d;
→       return result;
      }
}
```



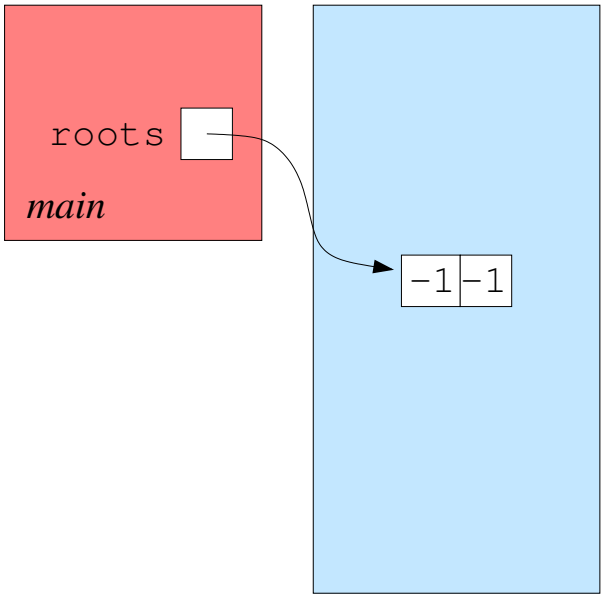
Vad händer vid anrop av quadEquation?

```
public static void main(String[] a) {
    double[] roots;
    roots = quadEquation(2, 1);
    ...
}


public static double[] quadEquation(double p,
                                    double q) {
      double d = (p*p - 4*q)/4;
      if (d < 0) {
        System.out.println("Complex roots");
        return null;
      } else {
        d = Math.sqrt(d);
        double[] r = new double[2];
        r[0] = -p/2 + d;
        r[1] = -p/2 - d;
        return result;
      }
}
```



Vad händer vid anrop av quadEquation?

```
public static void main(String[] a) {
    double[] roots;
    roots = quadEquation(2, 1);
→    ...
}
```



En anmärkning

De 4 sista raderna i `quadEquation`:

```
double[] r = new double[2];
r[0] = -p/2 + d;
r[1] = -p/2 - d;
return r;
```

kan ersättas med

```
return new double[]{-p/2+d, -p/2-d};
```

Typiska fel

Glömma skapa arrayen:

```
int[] a;
a[i] = 4;
```

NullPointerException

Felaktigt index:

```
int[] a = new int[5];
a[5] = 4;
```

ArrayIndexOutOfBoundsException

Glömma att skapa objekten
i en "objektarray":

```
Turtle[] t = new Turle[5];
t[0].move(40);
```

NullPointerException

Observera att felmeddelandet säger *var* felet har inträffat!

Introduktion till **klasser**

Ett Javaprogram består av en eller flera *klasser*.

Klassen är den minsta självständiga enheten i ett Javaprogram.

Vilka klasser har ni sett/använt?

Klasser har *namn*. Hur ser man om ett namn står för en klass?

En klass kan representera

- En konkret *fysisk* enhet. Exempel: fordon, tärning, person, hus, kondensator, sköldpadda, biljardboll, ...
- En mer *abstrakt* enhet. Exempel: triangel, bankkonto ...
- En *programmeringsteknisk* enhet. Exempel: program, teckenström, `Scanner`, `Math` ...

Vad är en klass?

- En *textfil* med typen `.java`.
- En *programmodul*.
- En abstrakt *beskrivning* av någon typ objekt.
- En *mall*.
- En *datatyp*.

Klasser och objekt

En typisk klass konstrueras med

- *instansvariabler*
- *metoder*

Instansvariablerna står ofta för någon *egenskap* (ex: *längd*, *vikt*, *färg*, *personnummer*, *segelyta*, ...

Metoderna står för någon form av operation (`move`, `turn`, ...) eller används för att hämta information (`getXPos`, `getWorld`, ...)

Fram t o m nätlektion 4 har ni sett metoder men inga instansvariabler.

De variabler ni sett/använt har varit *lokala* variabler eller *formella parametrar*.

Metoder

Klasser *instansieras* till *objekt*. Detta görs med operatorn **new**.

Referensvariabler (pekare, adresser) används för att hålla reda på objekt.

Operatorn **new** returnerar en referens till det skapade objektet.

För att komma åt attribut och metoder i används *punktnotation*.

Instansvariabler och metoder har en *synlighet* som anges med **public** eller `private` (två till finns).

Metoderna är ofta publika och instansvariablerna privata.

Exempel: En klass för att representera en tärning

Vilka egenskaper har en tärning?

- *Färg*?
- *Vikt*?
- *Material*?
- *Storlek*?
- *Antal sidor*?
- *Aktuellt värde*?
- ...

Beror på vad vi vill göra!

Nöjer oss med *Antal sidor* och *Aktuellt värde*

Vilka operationer vill vi ha?

- *Skapa* en tärning
- *Slå* en tärning
- *Avläsa* tärningens värde
- *Ändra* tärningens värde? Nej, knappast.
- *Ändra* antalet sidor? Möjligen.
- ...

En första skiss av klassen Dice

```
public class Dice {
    private int nSides;
    private int value;

    public Dice() { ... }            // Konstruktor

    public int getValue() { ... }    // Avläsa värde

    public void roll() { ... }       // Slå tärningen
}
```

Två privata *instansvariabler* (`nSides` och `value`).

En parameterlös *konstruktor* (`Dice`) och två publika *metoder* (`getValue` och `roll`).

Konstruktorer

Har samma namn som klassen. I detta fall `Dice`.

Används när ett objekt ska skapas.

Ger instansvariabler värden.

```
public Dice() {
    this.nSides = 6;
}
```

Fråga: Hur gör man för att få en tärning med ett annat antal sidor?

Alternativ konstruktor

För att skapa en tärning med ett annat antal sidor än 6:

```
public Dice(int nSides) {
    this.nSides = nSides;
}
```

Obs: Här är det nödvändigt att använda `this`!

Nu kan man skapa tärningar:

```
Dice d1 = new Dice();
Dice d2 = new Dice(17);
Dice d3 = new Dice(40396);
```

Måste man skriva konstruktorer?

Dice: Metoderna

Definition av metoderna:

```
public int getValue() {
   return value;
}


public void roll() {
   value = (int)(Math.random()*nSides) + 1;
}
```

Nu kan vi testa! Demo Dice

Förbättringar av konstruktorerna

Vad får `value` för värde när en tärning skapas?

Defaultvärde är på instansvariabler är 0 (eller `null` för referensvariabler).

Fråga: Vilket värde borde `value` få?

Så här då?

```
public Dice() {
    this.nSides = 6;
    this.roll();
}

public Dice(int nSides) {
   this.nSides = nSides;
   this.roll()
}
```

Fler förbättringar av konstruktorerna

Vad händer om man gör `new Dice(-3)`?

Konstruktorerna bör kontrollera att den får vettiga värden!

```
public Dice(int nSides) {
   if (nSides < 2) {
      System.out.println("Dice: Wrong number of sides");
      this.nSides = 6;
   }
   this.nSides = nSides;
   this.roll()
}
```

Dice: Ännu fler förbättringar

Upprepning av identisk kod är en styggelse!

Den parameterlösa konstruktorn skrivs snyggare så här:

```
public Dice() {
   this(6);
}
```

Uttrycket `this( ... )` i en konstruktor anropar alltså en annan konstruktor.

Flexiblare användning med modifierad `roll`-metod:

```
public int roll() {
   this.value = (int)(Math.random()*nSides) + 1;
   return this.value;
}
```

Dice: toString-metod

Metoden toString definierar en konvertering från ett objekt till ett String-objekt. Den vill man så gott som alltid ha!

```
public String toString()  {
   return  "" + value;
}
```

Öh?

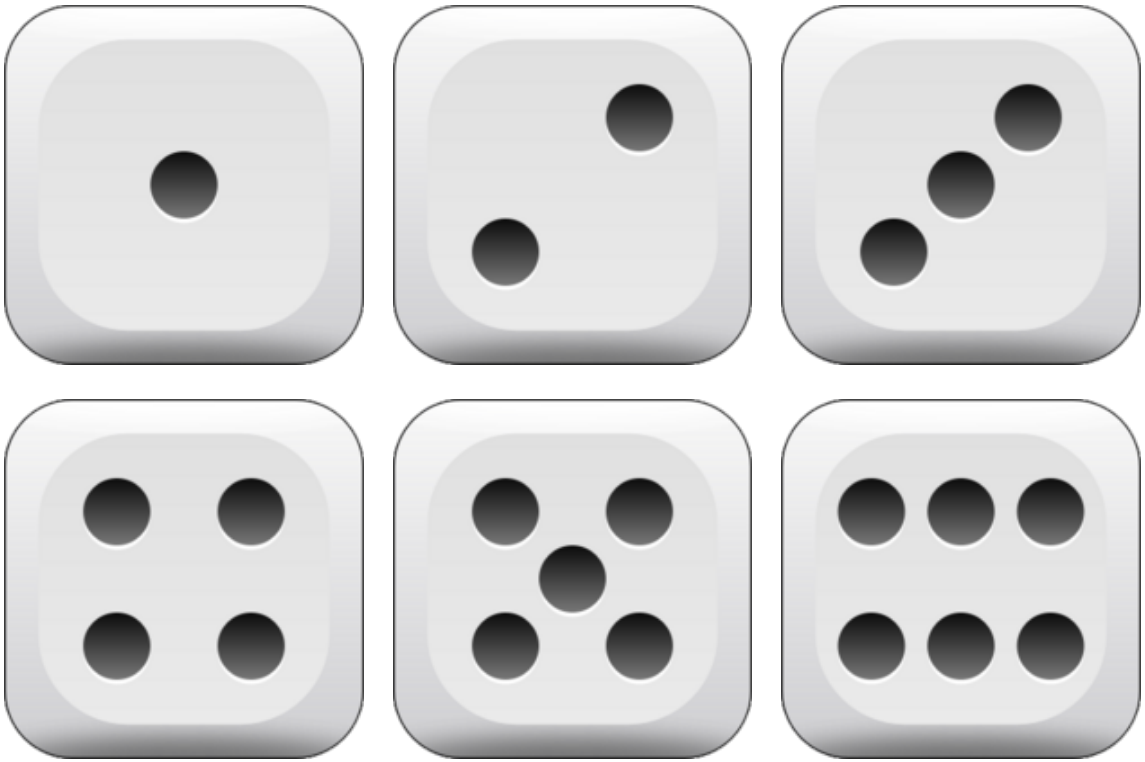
eller

```
public String toString()  {
   return "Dice(" + nSides + ", " + value + ")";
}
```

Råd: Gör inte toString-metoderna så pratiga!

Snyggare

Ett tärningsvärden ser ju ut så här:



Snyggare

eller så här:

Först: Hur kan man visa en bild?*

```
import java.awt.*;
import javax.swing.*;

public class ImageDisplay extends JFrame {

      public ImageDisplay(String imageFile, int width, int height,
                          int x, int y)
      {
          setLocation(x, y);
          ImageIcon icon = new ImageIcon(imageFile);
          JLabel label = new JLabel(icon);
          add(label);
          setPreferredSize(new Dimension(width, height));
          pack();
          setDefaultCloseOperation(EXIT_ON_CLOSE);
          setVisible(true);
      }
}
```

Demo ImageDisplay

Resultat

Med `main`-programmet

```
public static void main(String[] args) {
   new ImageDisplay("images/Matterhorn-800.jpg", 850, 600, 0, 0);
}
```

får man fönstret:

Länk till källkoden.

Hur gör man tärningsbilder?

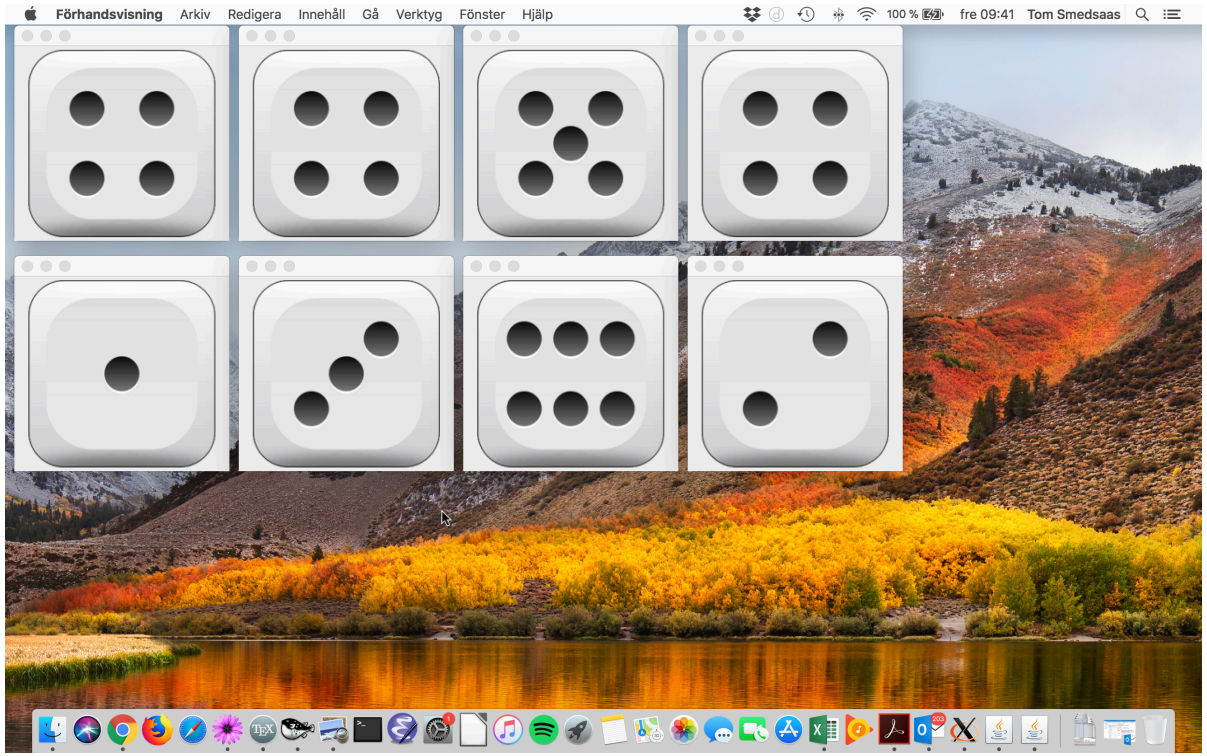
```
public class DiceDisplay {
   private String[] faces = new String[6];    // instansvariabel

   public DiceDisplay() {                     // konstruktor
      faces[0] = "images/Die1.png";
      faces[1] = "images/Die2.png";
      faces[2] = "images/Die3.png";
      faces[3] = "images/Die4.png";
      faces[4] = "images/Die5.png";
      faces[5] = "images/Die6.png";
   }

   void display(Dice d, int x, int y) {        // metod
      int index = d.getValue() - 1;
      new ImageDisplay(faces[index], 230, 230, x, y);
   }
}
```

Användning

```
public static void main(String[] args)    {
   Dice d = new Dice();
   DiceDisplay dd = new DiceDisplay();
   for (int i =0; i<12; i++) {
      d.roll();
      System.out.print(d.getValue() + "  ");
      //dd.display(d, 0, 0);
      dd.display(d, i%4*240 + 10, i/4*280 + 10);
   }
   System.out.println();
}
```

Demo DiceDisplay
Demo PokerDice
Demo DiceWindow

Resulterande skärm av `DiceDisplay`



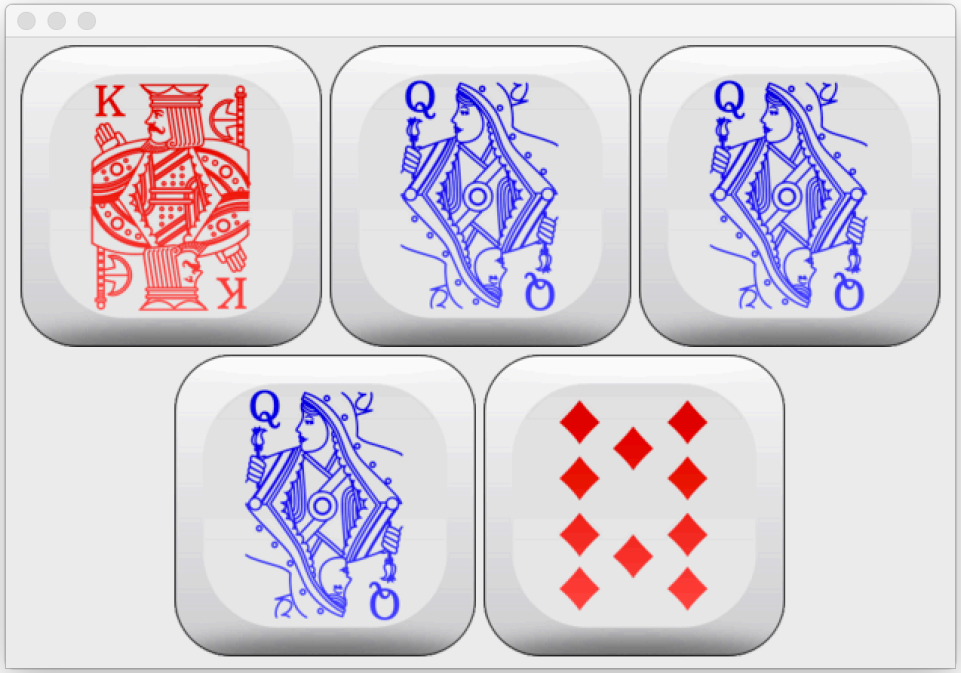
Länk till källkoden.

Pokertärningar

```
public class PokerDice  extends JFrame {
   /**
    * Opens a window and displays a number of dice
    * with poker faces
    * @param dice the set of dice
    */
   public PokerDice(Dice[] dice) {
      ImageIcon[] icons = new ImageIcon[6];
      icons[0] = new ImageIcon("images/Die9.png");
      icons[1] = new ImageIcon("images/Die10.png");
      icons[2] = new ImageIcon("images/DieJack.png");
      icons[3] = new ImageIcon("images/DieQueen.png");
      icons[4] = new ImageIcon("images/DieKing.png");
      icons[5] = new ImageIcon("images/DieAce.png");
      for (Dice d: dice) {
         add(new JLabel(icons[d.getValue()-1]));
      }
      setLayout(new FlowLayout());
      setPreferredSize(new Dimension(630, 440));
      pack();
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      setVisible(true);
   }
```

Pokertärningar forts

```
public static void main(String[] args) {
   Dice[] poker = new Dice[5];
   for (int i=0; i<5; i++) {
      poker[i] = new Dice();
   }
   new PokerDice(poker);
}
```



Länk till källkoden.

Länk till källkoden för klickbar pokertärning.