EDAA20
Programmering och databaser

Föreläsning 9 – Vektorer av objekt, filer

2023-09-25, Niklas Fors

Extra resurstid tillagd torsdagar kl 10-12 i E:2116

Repetition: vektorer

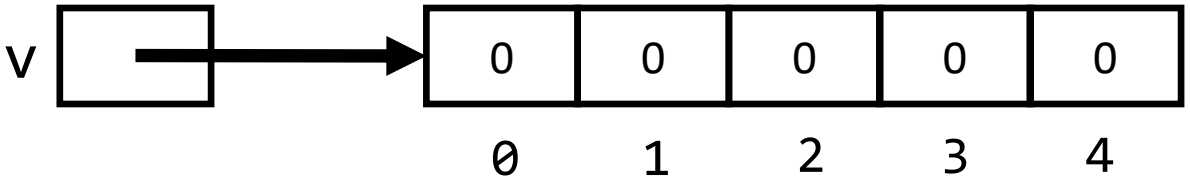
En ***vektor*** (*array*) är en följd av element av en viss typ och med en storlek.

Skapa heltalsvektor med 5 element:

```
int[] v = new int[5];
```

Elementen får 0 som startvärden (`0.0` för `double`, `false` för `boolean`, osv)

Vektorer är objekt



Referensvariabeln **v** refererar till vektorn

Repetition

Läs in heltal och lagra i vektor:

```
for (int i = 0; i < v.length; i++) {
  v[i] = scan.nextInt();
}
```

`v.length` – vektorns storlek

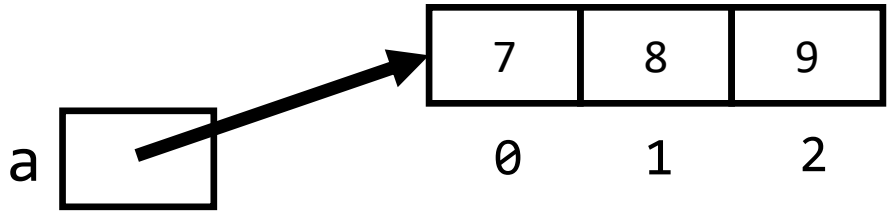
`v[i]` – uppdatera eller hämta element på index `i`

index ska vara mellan `0..v.length-1`

Övning: vektorer

Vad händer i detta exempel? Hur många vektorer skapas? Vad refererar **b** till? Komplettera bilden.

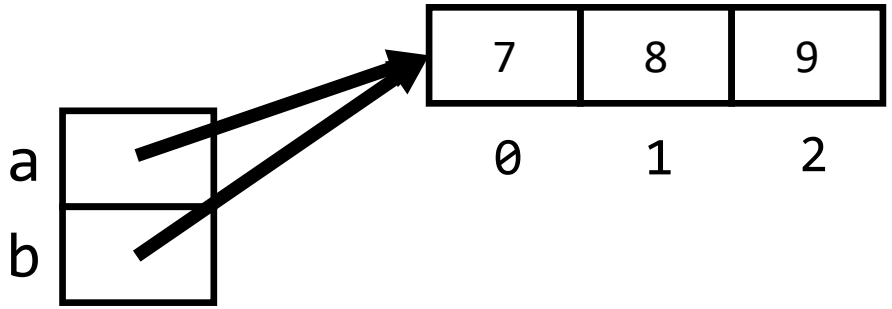
```
int[] a = new int[3];
a[0] = 7;
a[1] = 8;
a[2] = 9;
int[] b = a;
```



Övning: vektorer

Vad händer i detta exempel? Hur många vektorer skapas? Vad refererar **b** till? Komplettera bilden.

```
int[] a = new int[3];
a[0] = 7;
a[1] = 8;
a[2] = 9;
int[] b = a;
```

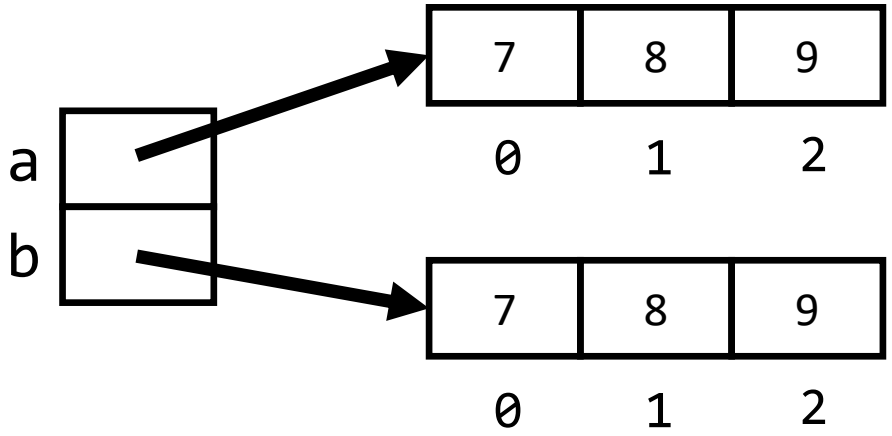


Enbart en vektor skapas. Både **a** och **b** refererar till samma vektor

Övning: två vektorer

Hur skapar man en till vektor med samma innehåll som vektorn **a**?

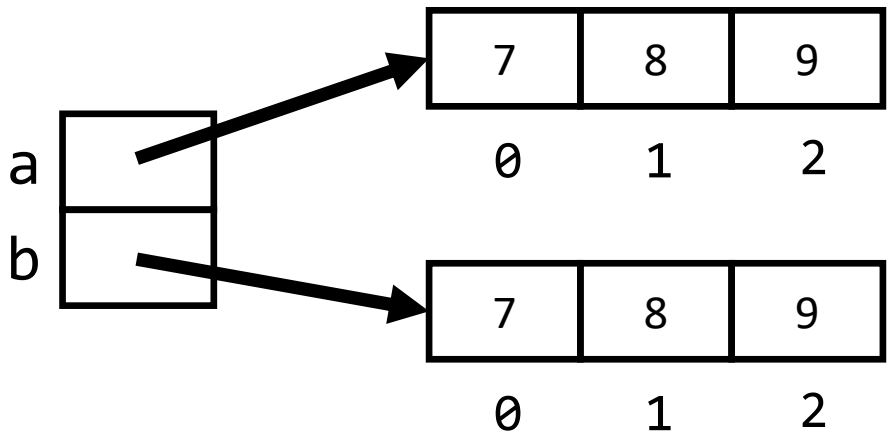
```
int[] a = new int[3];
a[0] = 7;
a[1] = 8;
a[2] = 9;

???
```



Övning: två vektorer

Hur skapar man en till vektor med samma innehåll som vektorn **a**?

```
int[] a = new int[3];
a[0] = 7;
a[1] = 8;
a[2] = 9;

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



En ny vektor med samma storlek skapas och elementen kopieras

Genvägar

Skapa vektor med givet innehåll:

```
int[] a = {7, 8, 9};
```

Skriva ut en vektors innehåll:

```
System.out.println(Arrays.toString(a));
```

Vektor av objekt

Man kan skapa ***vektorer av objekt***, exempel:

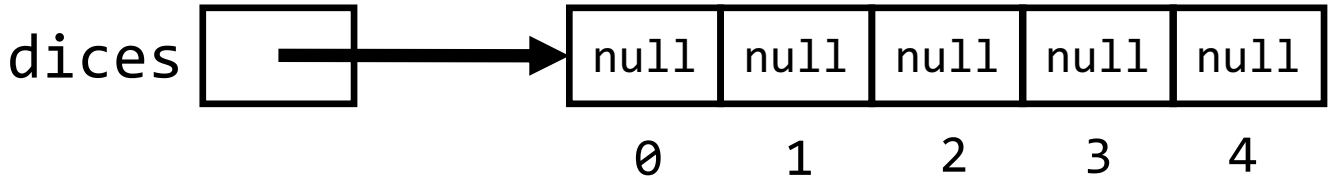
```
Dice[] dices = new Dice[5];
String[] args = new String[100];
```

Vilka startvärden får elementen?

Steg 1

Skapa vektor med **tomma platser** för tärningar:

```
Dice[] dices = new Dice[5];
```



Elementen får `null` som startvärden

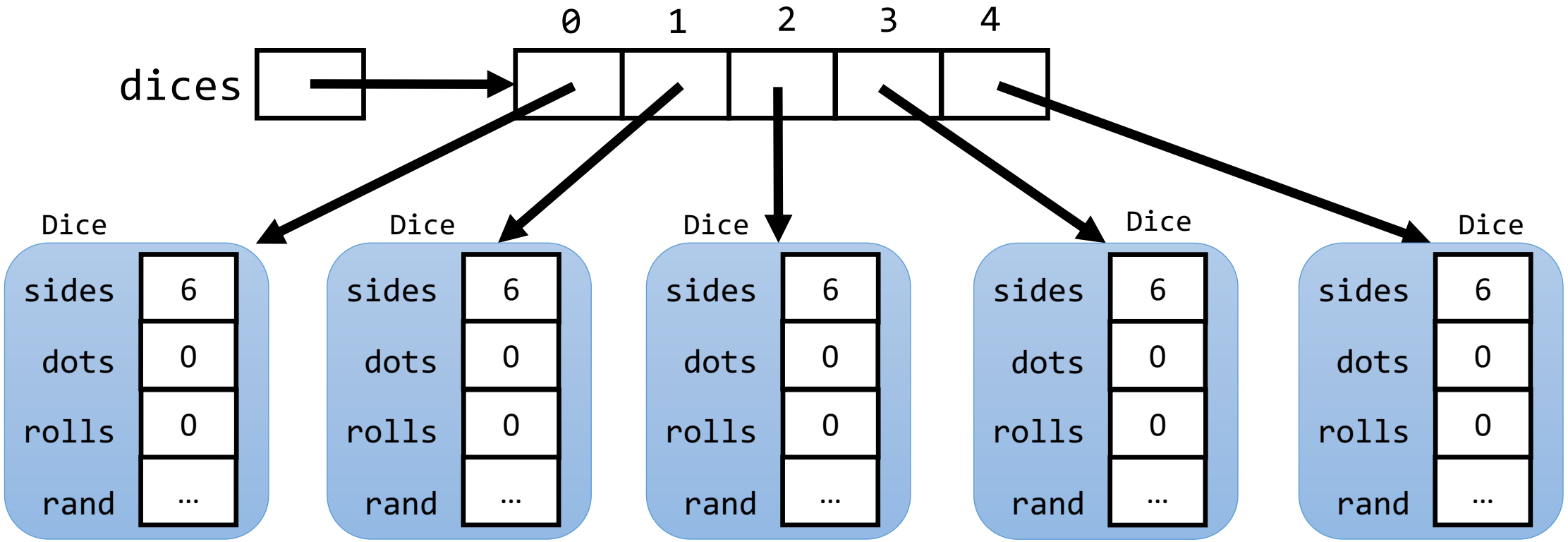
Elementen är referensvariabler av typen `Dice`

Steg 2

Därefter skapa tärningar och lagra i vektorn:

```
for (int i = 0; i < dices.length; i++) {
  dices[i] = new Dice();
}
```

Tärningar i minnet



Program med 5 tärningar

```
public class FiveDice {
  public static void main(String[] args) {
    // Steg 1: Skapa vektor för tärningar med tomma platser
    Dice[] dices = new Dice[5];

    // Steg 2: Skapa tärningsobjekt och lagra referenserna i vektorn
    for (int i = 0; i < dices.length; i++) {
      dices[i] = new Dice();
    }

    // Kasta tärningarna
    for (int i = 0; i < dices.length; i++) {
      dices[i].roll();
      System.out.println("Tärning " + i + " visar " + dices[i].getDots());
    }
  }
}
```

Programmet skapar 5 tärningar och kastar dem

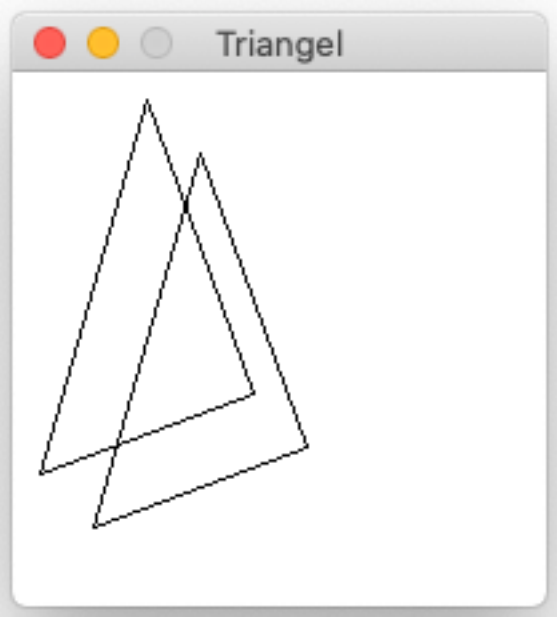
Triangel

Klassen `Triangle` har följande specifikation:

```
/** Skapar triangel med hörnpunkterna x1 y1, x2 y2 och x3 y3. */
Triangle(int x1, int y1, int x2, int y2, int x3, int y3);

/** Flyttar triangeln relativt med avstånden dx, dy */
void move(int dx, int dy);

/** Ritar triangeln i fönstret w. */
void draw(SimpleWindow w);
```

Användning

```
public class TriangleExample {
  public static void main(String[] args) {
    SimpleWindow w = new SimpleWindow(200, 200, "Triangel");
    Triangle t = new Triangle(10, 150, 50, 10, 90, 120);
    t.draw(w);
    t.move(20, 20);
    t.draw(w);
  }
}
```



Klassen `Triangle`

Vi ska implementera klassen `Triangle`:

```
/** Skapar triangel med hörnpunkterna x1 y1, x2 y2 och x3 y3. */
Triangle(int x1, int y1, int x2, int y2, int x3, int y3);

/** Flyttar triangeln relativt med avstånden dx, dy */
void move(int dx, int dy);

/** Ritar triangeln i fönstret w. */
void draw(SimpleWindow w);
```

ÖVNING: Vilka attribut ska klassen ha?

Vilka attribut ska klassen `Triangle` ha?

Olika lösningar för hörnpunkterna:

- `private int x1, y1, x2, y2, x3, y3;`
- `private Point p1, p2, p3;`
- `private Point[] vertices;`

Vilka attribut ska klassen `Triangle` ha?

Olika lösningar för hörnpunkterna:

- `private int x1, y1, x2, y2, x3, y3;`
- `private Point p1, p2, p3;`
- `private Point[] vertices;`

Vi väljer alternativ tre,
där en färdigimplementerad klass `Point` används

Klassen Point

Exempel på användning av Point:

```
Point p = new Point(10, 20);
p.move(5, 10);
System.out.println(p.getX());  // 15
System.out.println(p.getY());  // 30
```

Specifikation av klassen Point:

```
/** Skapar en punkt med koordinaterna x och y */
Point(int x, int y);

/** Returnerar x-koordinaten */
int getX();

/** Returnerar y-koordinaten */
int getY();

/** Flyttar punkten relativt med avstånden dx, dy */
void move(int dx, int dy);
```

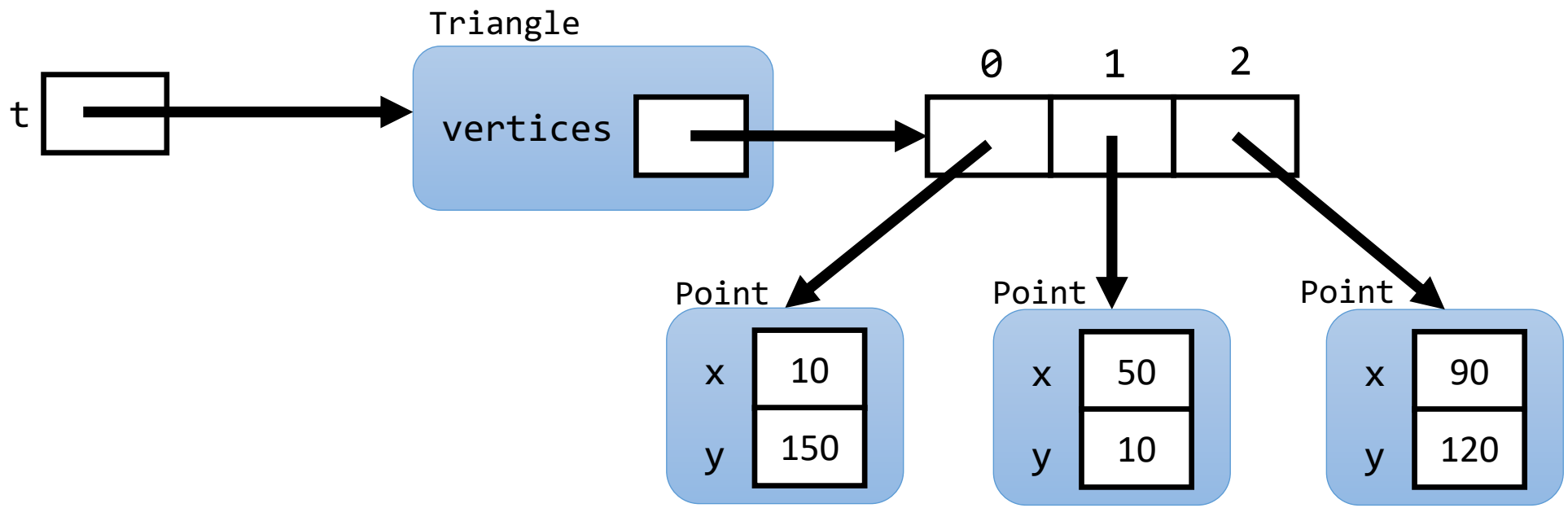
```java
public class Point {
  private int x, y;

  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public int getX() {
    return x;
  }

  public int getY() {
    return y;
  }

  public void move(int dx, int dy) {
    x += dx;
    y += dy;
  }
}
```

Implementation av klassen `Point`

Vi behöver egentligen inte veta hur klassen `Point` är implementerad för att använda den.
Det räcker med specifikationen.

Triangel i minnet

```
Triangle t = new Triangle(10, 150, 50, 10, 90, 120);
```



Attribut & konstruktor

Specifikation av Point:

```
Point(int x, int y);
int getX();
int getY();
void move(int dx, int dy);
```

```
public class Triangle {
  private Point[] vertices;

  /** Skapar en triangel med hörnpunkterna x1 y1, x2 y2 och x3 y3. */
  public Triangle(int x1, int y1, int x2, int y2, int x3, int y3) {
    // Steg 1: Skapa Point-vektor med 3 tomma platser
    vertices = new Point[3];
    // Steg 2: Skapa Point-objekten och lagra i vektorn
    vertices[0] = new Point(x1, y1);
    vertices[1] = new Point(x2, y2);
    vertices[2] = new Point(x3, y3);
  }

  ...
}
```

Övning: metoden **move**

Specifikation av Point:

```
Point(int x, int y);
int getX();
int getY();
void move(int dx, int dy);
```

```
public class Triangle {
  private Point[] vertices;
  ...

  /** Flyttar triangeln relativt med avstånden dx, dy */
  public void move(int dx, int dy) {
    // ÖVNING: Implementera metoden och flytta hörnen.
    // Vilken metod är lämplig att anropa på punkterna?

  }

  ...
}
```

Övning: metoden **move**

Specifikation av Point:

```
Point(int x, int y);
int getX();
int getY();
void move(int dx, int dy);
```

```
public class Triangle {
  private Point[] vertices;
  ...

  /** Flyttar triangeln relativt med avstånden dx, dy */
  public void move(int dx, int dy) {
    for (int i = 0; i < vertices.length; i++) {
      vertices[i].move(dx, dy);
    }
  }
  ...
}
```

Fördel med vektor som attribut:
möjliggör for-satser – oavsett hur många punkter som finns!

Gå igenom vektorn av punkter och anropa metoden move på varje punkt

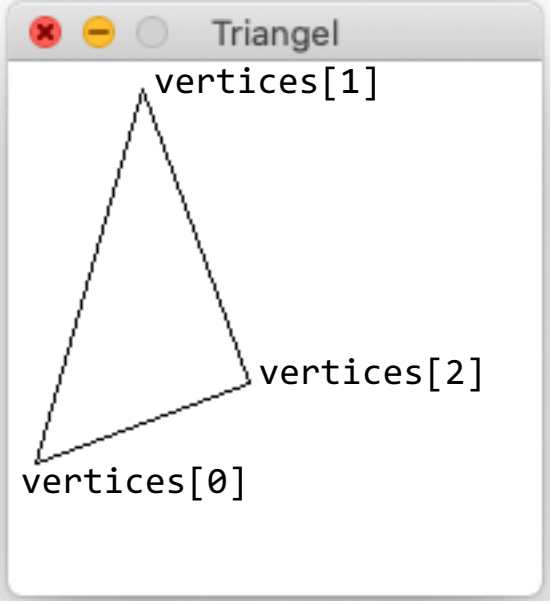
Metoden `draw`

Specifikation av `Point`:

```
Point(int x, int y);
int getX();
int getY();
void move(int dx, int dy);
```

```
public class Triangle {
  private Point[] vertices;
  ...

  /** Ritar triangeln i fönstret w. */
  public void draw(SimpleWindow w) {

    // använd w.moveTo och w.lineTo



  }

  ...
}
```

Flytta först till sista punkten, därefter ritar vi ett sträck till punk 0, till punkt 1, till punkt 2.



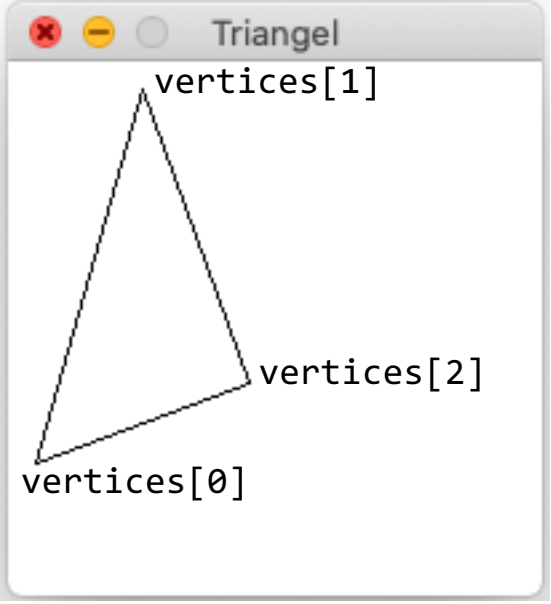
Metoden `draw`

Specifikation av `Point`:

```
Point(int x, int y);
int getX();
int getY();
void move(int dx, int dy);
```

```
public class Triangle {
  private Point[] vertices;
  ...

  /** Ritar triangeln i fönstret w. */
  public void draw(SimpleWindow w) {
    w.moveTo(vertices[2].getX(), vertices[2].getY());
    w.lineTo(vertices[0].getX(), vertices[0].getY());
    w.lineTo(vertices[1].getX(), vertices[1].getY());
    w.lineTo(vertices[2].getX(), vertices[2].getY());
  }

  ...
}
```

Flytta först till sista punkten, därefter ritar vi ett sträck till punk 0, till punkt 1, till punkt 2.



Metoden `draw`

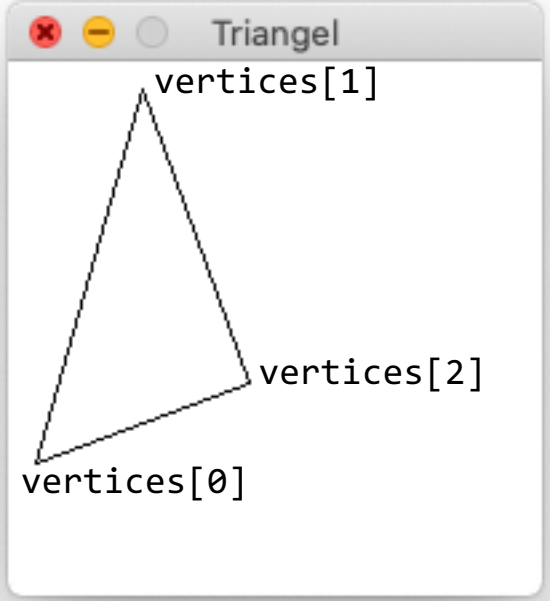
Specifikation av `Point`:

```
Point(int x, int y);
int getX();
int getY();
void move(int dx, int dy);
```

```
public class Triangle {
  private Point[] vertices;
  ...

  /** Ritar triangeln i fönstret w. */
  public void draw(SimpleWindow w) {
    w.moveTo(vertices[2].getX(), vertices[2].getY());
    for (int i = 0; i < vertices.length; i++) {
      w.lineTo(vertices[i].getX(), vertices[i].getY());
    }
  }

  ...
}
```

Använd en `for`-sats istället för att upprepa koden!

Flytta först till sista punkten, därefter ritar vi ett sträck till punk 0, till punkt 1, till punkt 2.



Klassen Scanner för inläsning

```
public class SumNumbers {
  public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    int sum = 0;
    while (scan.hasNextInt()) {
      sum = sum + scan.nextInt();
    }
    System.out.println("Summa: " + sum);
  }
}
```

Vi har hittills använt Scanner-klassen för att läsa in värden från terminalen

Klassen `Scanner` för inläsning

Från terminalfönstret:

```
Scanner scan = new Scanner(System.in);
```

Från fil:

```
Scanner scan = new Scanner(new File("filnamn"));
```

Från sträng:

```
Scanner scan = new Scanner("1 23 34 56");
```

Scanner för att läsa in från fil

```
public class SumNumbersFromFile {
  public static void main(String[] args) {
    Scanner scan = null;
    try {
      scan = new Scanner(new File("numbers.txt"));
    } catch (FileNotFoundException e) {
      System.out.println("Filen kunde inte öppnas");
      System.exit(1);
    }
    int sum = 0;
    while (scan.hasNextInt()) {
      sum = sum + scan.nextInt();
    }
    System.out.println("Summa: " + sum);
    scan.close();
  }
}
```

Scanner-klassen kan också användas för inläsning från fil.
Då måste man hantera om filen inte finns.

Scanner för att läsa in från fil

```
public class SumNumbersFromFile {
  public static void main(String[] args) {
    Scanner scan = null;
    try {
      scan = new Scanner(new File("numbers.txt"));
    } catch (FileNotFoundException e) {
      System.out.println("Filen kunde inte öppnas");
      System.exit(1);
    }
    int sum = 0;
    while (scan.hasNextInt()) {
      sum = sum + scan.nextInt();
    }
    System.out.println("Summa: " + sum);
    scan.close();
  }
}
```

1) Öppna filen, skriv ut felmeddelande om filen ej finns och avsluta programmet

2) Läs in heltal och summera, precis som tidigare. Därefter skriv ut summan

3) Stäng filen

Scanner-klassen kan också användas för inläsning från fil.
Då måste man hantera om filen inte finns.

Scanner för att läsa in från fil

```
public class SumNumbersFromFile {
  public static void main(String[] args) {
    Scanner scan = null;
    try {
      scan = new Scanner(new File("numbers.txt"));
    } catch (FileNotFoundException e) {
      System.out.println("Filen kunde inte öppnas");
      System.exit(1);
    }
    int sum = 0;
    while (scan.hasNextInt()) {
      sum = sum + scan.nextInt();
    }
    System.out.println("Summa: " + sum);
    scan.close();
  }
}
```

Variabeln måste deklareras innan try-satsen, så att den kan användas efter try-satsen om filen öppnas.

Java kräver i detta fall att vi tilldelar variabeln ett värde, även om vi kan se att det inte behövs i detta fall (då programmet stängs av om vi inte lyckas öppna filen).

Scanner-klassen kan också användas för inläsning från fil.
Då måste man hantera om filen inte finns.

Inläsning från sträng

```
public class SumNumbersFromString {
  public static void main(String[] args) {
    Scanner scan = new Scanner("1 23 34 56");
    int sum = 0;
    while (scan.hasNextInt()) {
      sum = sum + scan.nextInt();
    }
    System.out.println("Summa: " + sum);     // Skriver ut summan 114
  }
}
```

Programmet läser in flera heltal från en sträng och skriver ut deras summa

Detta används i labb 8, där man läser alla tal för varje rad i en fil

Felhantering: `try-catch`-sats

Ibland inträffar fel och då kan man hantera det med en `try-catch`-sats:

```
try {
  // Försöker göra något där fel kan inträffa
} catch (<Typen på felet> e) {
  // Felhantering om felet inträffar,
  // exempelvis skriv ut felmeddelande och avbryt programmet.
  // Använd variabeln e för att få fram information om felet.
}
```

Om felet inträffar avbryts exekveringen i `try`-satsen och exekveringen övergår till `catch`-satsen. Mer om detta i fortsättningskursen.

Detta kallas ***Exceptions / Exception-hantering***