Föreläsning 11-12

Innehåll

- Mer om vektorer med objekt
- Klassen ArrayList<E>

Hantera många element

- Tidigare har vi använt vektorer för att lagra många element av samma typ. Exempel:

```
int[] nbrs = new int[6];
Point[] vertices = new Point[3];
```

- Nackdelar:
  - När vi skapar vektorn måste vi bestämma hur många element den ska innehålla.
  - Om vektorn inte är "fylld" med element finns "tomma" platser med värdet 0, null ...

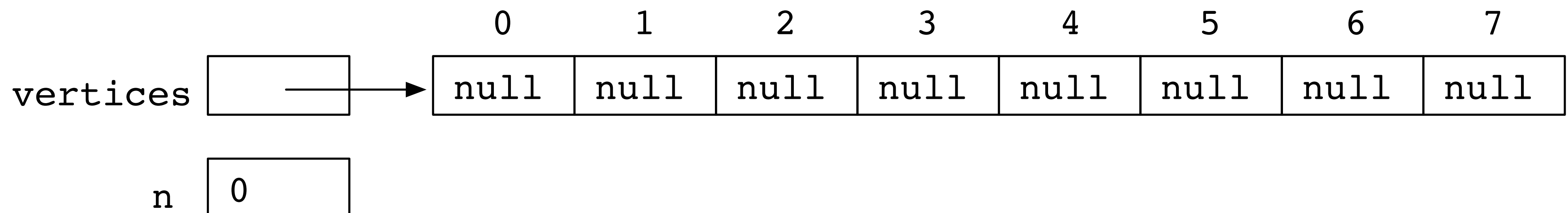
- Slutsats: En vektor passar bra när vi på förhand vet antal element.

Exempel: polygon

Godtyckligt antal element

- Antag att vi vill hantera polygoner med ett godtyckligt antal hörnpunkter.
    - Vi måste skapa en tillräckligt stor vektor för punkterna.
    - Det behövs också en variabel som håller reda på antal insatta punkter.

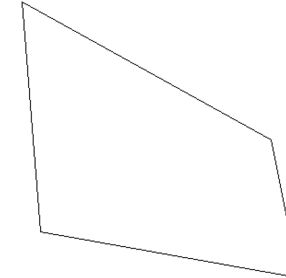
```
Point[] vertices = new Point[8];
int n = 0;
```

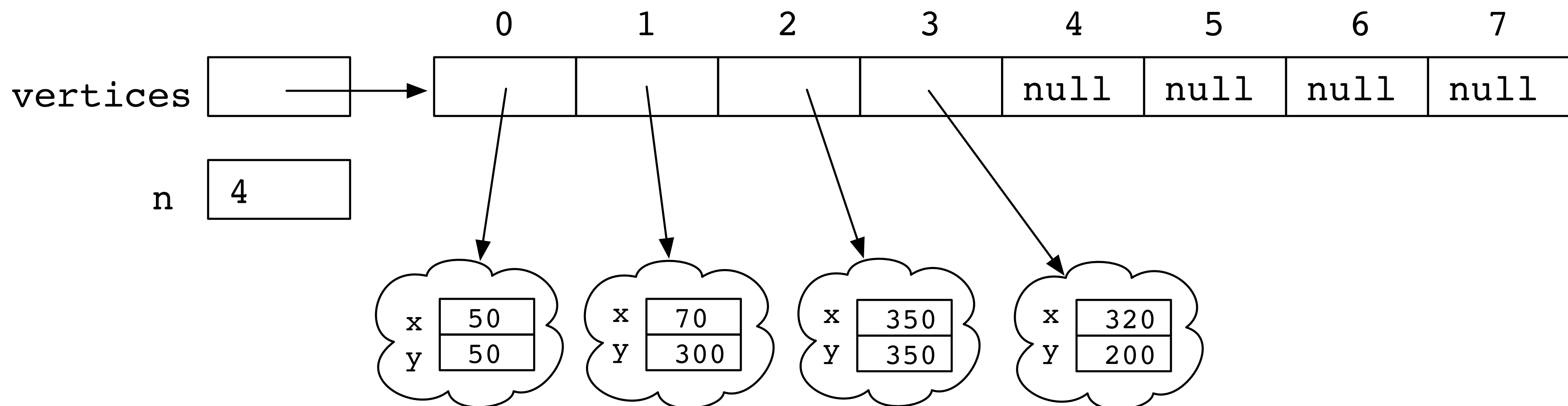


Exempel: polygon

forts

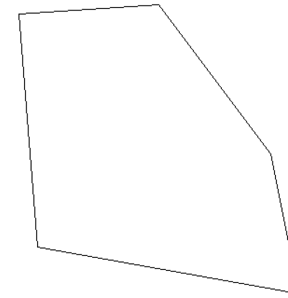
- Exempel: Polygon med fyra hörnpunkter





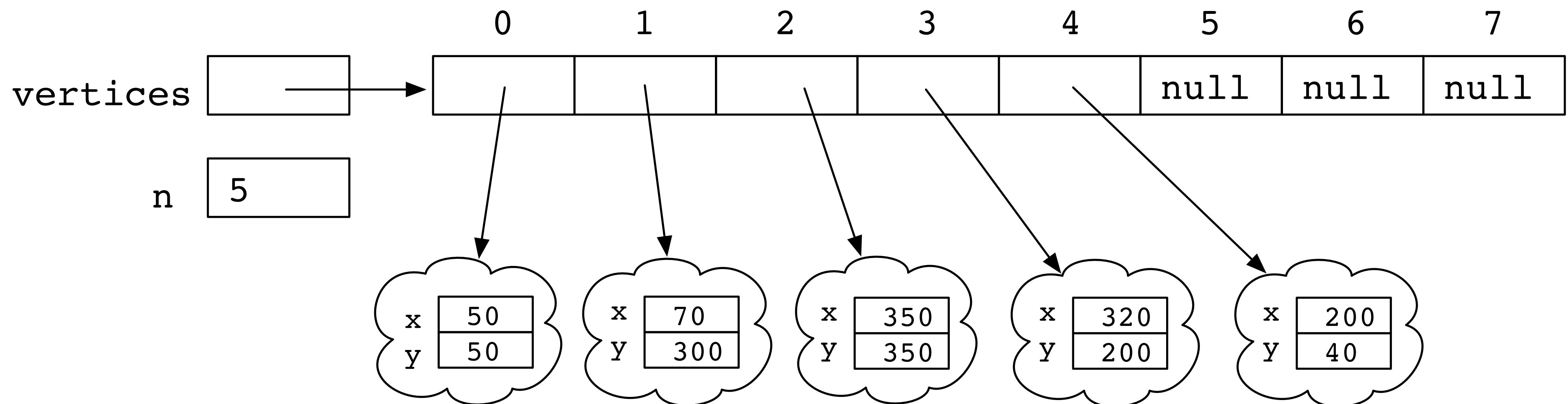
Exempel: polygon

Sätta in en ny punkt



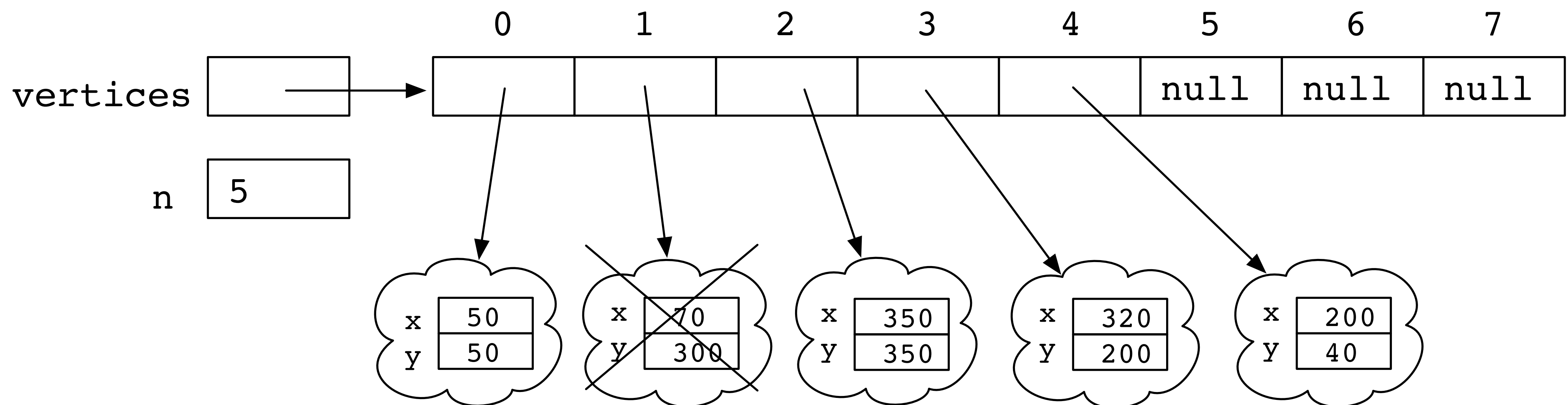
- Insättning av en ny punkt:

```
vertices[n] = new Point(200, 40);
n++;
```



Diskutera

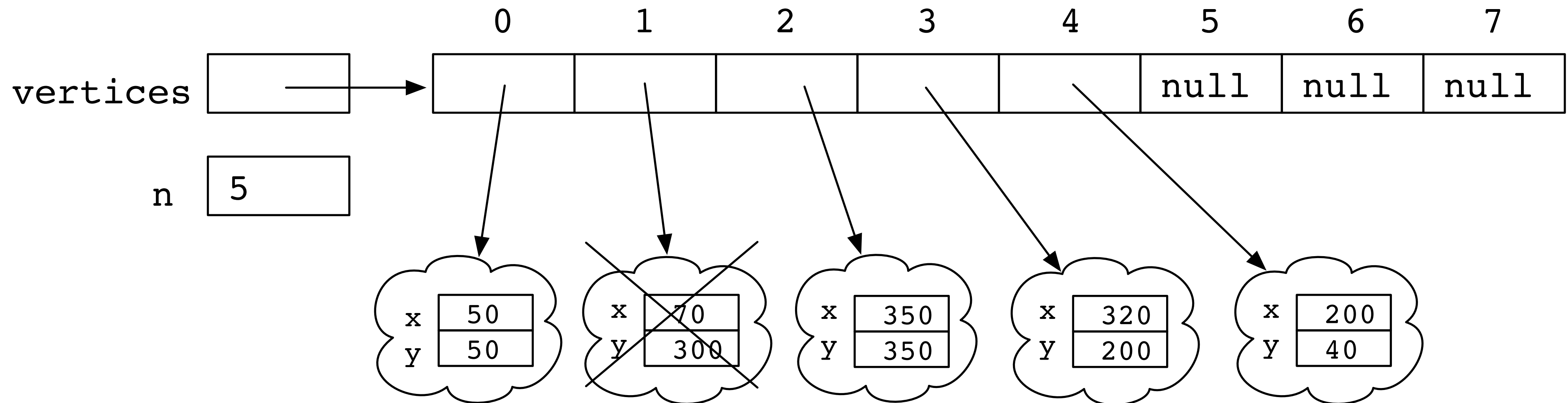
- Antag att vi ska ta bort en punkt från polygonen, t ex andra punkten.



- Hur ska det gå till? Skissa på kod för detta.

Exempel: polygon

Ta bort en punkt

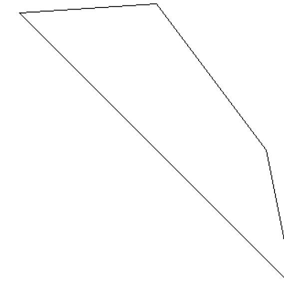


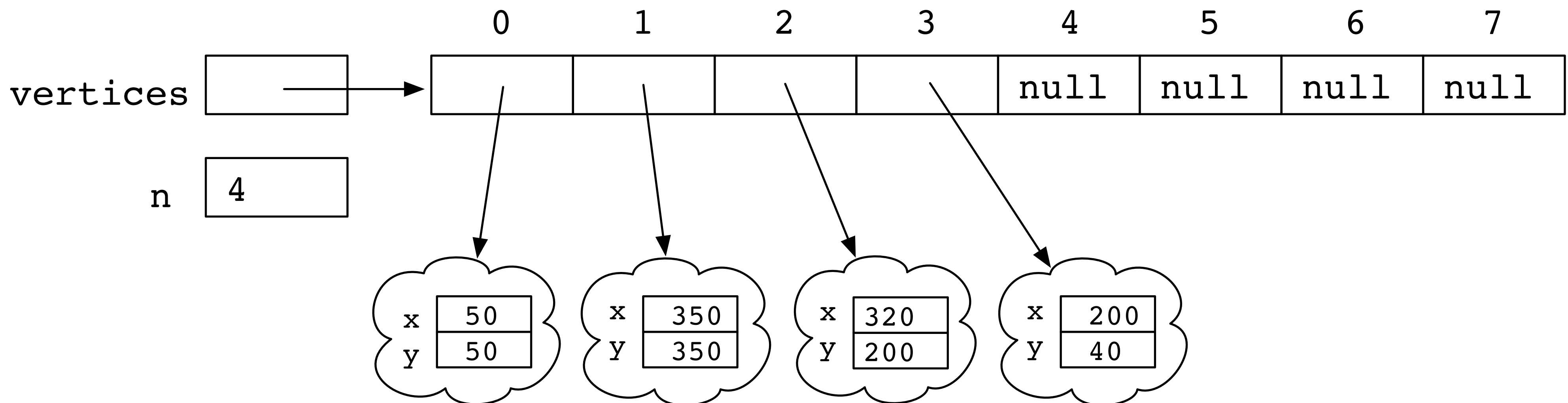
- Tag bort andra punkten (finns på position 1 i vektorn):
  - Täpp till "hålet" i vektorn genom att flytta alla efterföljande element ett steg till vänster.
  - Uppdatera `n` eftersom antal punkter nu minskat med 1.

Exempel: polygon

Efter borttagning av punkten

```
int pos = 1;
for (int i = pos; i < n - 1; i++) {
    vertices[i] = vertices[i + 1];
}
vertices[n - 1] = null;
n--;
```





Passar en vektor för detta?

- I exemplet på föregående bilder måste vi gissa hur stor vektor vi behöver. Vad händer om vi vill sätta in fler än 8 punkter? Då måste vi skapa en ny större vektor.
- För problem där antal element varierar är det enklare att använda den färdiga klassen `ArrayList`.
- Ett `ArrayList`-objekt använder internt en vektor och har färdiga metoder för att sätta in element i listan, ta bort element etc.

Klassen ArrayList<E>

Klassen `ArrayList<E>`

- används för att lagra element i en lista.
- är en standardklass (i paketet `java.util`).
- innehåller alltid objekt (inte `int`, `double`, ...).
- lagrar internt sina element i en vektor.
  - utökar vektorns storlek vid behov.
  - har metoder för att sätta in, ta bort, ...

Några metoder i klassen ArrayList<E>

```
/** Skapar en tom lista. */
ArrayList();

/** Returnerar elementet på plats pos. */
E get(int pos);

/** Lägger in obj sist. */
void add(E obj);

/** Tar bort elementet på plats pos, returnerar det borttagna
    elementet.  */
E remove(int pos);

/** Returnerar antalet element. */
int size();
...
```

Generisk klass

- ArrayList<E> är en generisk klass vilket innebär att den kan innehålla objekt av godtycklig typ.
- En generisk klass har en eller flera typparametrar.
  - Anges inom $< >$ efter klassnamnet.
- När en generisk klass används ska man ange ett typargument – ett klassnamn.
  - Exempel där vi skapar en lista som kan innehålla `String`-objekt:

```
ArrayList<String> words = new ArrayList<String>();
words.add("en");
words.add("lista");
words.add("med");
words.add("ord");
```

Generisk klass

Typkontroll

- Generik hindrar oss att göra fel.
  - I en lista av typen `ArrayList<String>` kan man bara sätta in strängar.
- Kompilatorn kommer att upptäcka typfel. Ex:

```
ArrayList<String> words = new ArrayList<String>();
words.add(42); // Går inte!
```

(Raden `words.add(42);` är överstruken på sidan.)

Detta ger kompileringsfel. Listan `words` får enligt sin deklaration endast innehålla objekt av typen `String`.

Generisk klass

Restriktioner för typargument

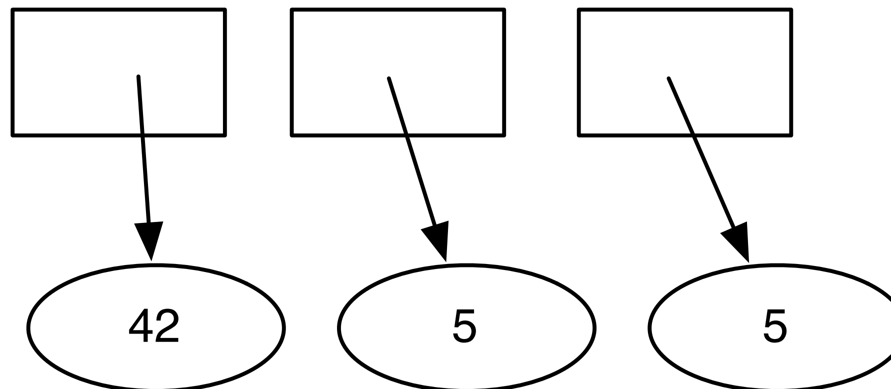
- Typargumentet får *inte* vara av primitiv typ:

  ~~`ArrayList<int> list = new ArrayList<int>();`~~ // Går inte!

- Istället får man använda motsvarande **wrapperklass**:

```
List<Integer> list = new ArrayList<Integer>();
```

- I en lista av typen `ArrayList<Integer>` lagras alltså elementen i en vektor med `Integer`-objekt:



Wrapperklasser

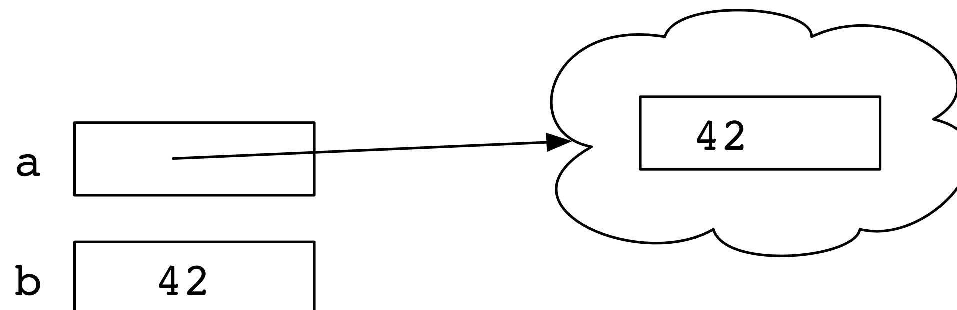
Primitiva typer i Java:	Motsvarande wrapperklasser:
`boolean`	`Boolean`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`char`	`Character`
`byte`	`Byte`
`float`	`Float`
`double`	`Double`

Wrapperklasser

forts.

- Wrapperklasser används när man behöver ”packa in” ett värde av en primitiv datatyp i ett objekt.
  - T.ex. när man vill lagra heltal i en Arraylist.
- Exempel på användning av klassen `Integer`

```
Integer a = new Integer(42);  // skapar ett Integer-objekt

int b = a.intValue();         // hämtar heltalet i objektet a
```



Wrapperklasser

fler metoder

- I wrapperklasserna finns även en hel del andra metoder.
- Exempel: statisk metod som konverterar från sträng till heltal:

```
int n  = Integer.parseInt("42");
```

och från heltal till sträng:

```
String s = Integer.toString(42);
```

Autoboxing - unboxing

- Automatisk konvertering mellan primitiv datatyp och objekt.
- Exempel:

```
Integer i = 42;     // autoboxing till Integer-objekt
```

- Exempel:

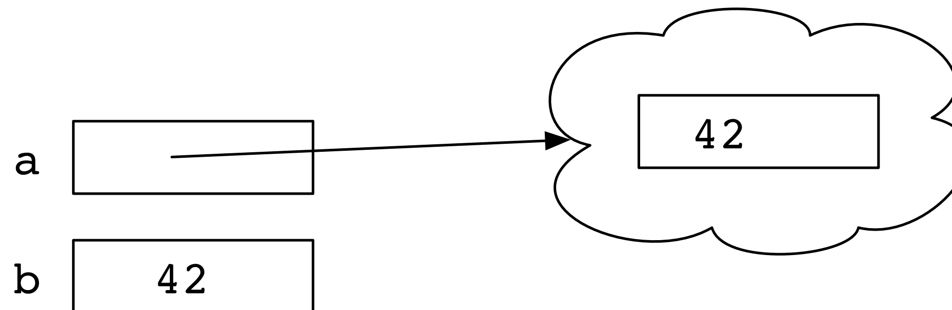
```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add(42);              // autoboxing
int n = list.get(0);       // unboxing
```

  - Praktiskt! När man använder en lista av typen `ArrayList<Integer>` behöver man alltså inte själv tänka på att konvertera från vanligt heltal till Integer-objekt och tvärtom.

int vs Integer

- Observera skillnaden mellan typerna `int` (primitiv datatyp) och `Integer` (klass).
- Exempel:

```
Integer a = 42;  // autoboxing, därför samma resultat som
                 // Integer a = new Integer(42);
int b = 42;
```



Traversera elementen i en lista

Mönster

- *Uppgift*: Behandla alla elementen i en lista.
- *Algoritm*:

```
för varje element i listan {
    använd elementet
}
```

- Det finns olika sätt att traversera listan:
  - ”for each”-sats
  - använda metoderna `get` och `size` i en for-sats
  - ...

Behandla alla elementen i listan - ”for each”

Alternativ 1

```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add(53);
list.add(-11);
...

int sum = 0;
for (int n : list) {   // för varje heltal n i list
   sum = sum + n;
}
```

- Fungerar bra när man ska gå igenom listan från början. (Svårare om man inte ska starta från position 0.)
- Fungerar även för att traversera andra samlingar av element i Java.
- Man får inte ändra i listan med `add` eller `remove` inuti for-each-satsen.

Behandla alla elementen i listan - for-sats med get och size

Alternativ 2

```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add(53);
list.add(-11);
...

int sum = 0;
for (int i = 0; i < list.size(); i++) {
   sum = sum + list.get(i);
}
```

- Fungerar bra med klassen `ArrayList`. Men intre med liknande klasser i Java.
- Tänk på att `list.size()` levererar ett nytt värde om listans antal element ändras.

Övning

En lista `numbers` av typen `ArrayList<Double>` innehåller ett antal tal.
Räkna antal positiva tal i `numbers`.

Diskutera

Vad skrivs ut när man kör programmet?

```
public class LengthExample {
  public static void main(String[] args) {
      ArrayList<Integer> list = new ArrayList<Integer>();
      System.out.println("Antal element i listan: " + list.size());

      int[] v = new int[10];
      System.out.println("Antal element i vektorn " + v.length);
  }
}
```

`Arraylist<E>` vs vektor

- `ArrayList<E>` passar bra när antal element inte är bestämt. Klassen har färdiga metoder för att lägga till element, ta bort element . . .
- En vektor passar bra när man på förhand vet antal element. Exempel:
  - `int[] nbrs = new int[6];`
- Observera att ett `ArrayList`-objekt utifrån sett är en lista (och inte en bekvämare variant av en vektor).
  - En nyskapad lista är tom från början.
  - En nyskapad vektor har däremot redan från början det antal element som angavs vid skapandet av vektorn.

Exempel: Klassen Polygon

- Vi ska skriva en klass som representerar ett polygon som kan ritas i ett fönster. Vi ska kunna lägga till och ta bort punkter samt flytta polygonen.
- Inuti klassen behöver vi ett attribut som håller reda på polygonens hörnpunkter.

```
private ArrayList<Point> vertices;
```

- Det finns en färdig klass `Point` i Java som kan användas. Men här väljer vi att skriva en egen `Point`-klass som håller reda på koordinaterna i form av heltal.

Klassen Point

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

    /** Skapar en punkt med koordinaterna x och y. */
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    /** Tar reda på x-koordinaten. */
    public int getX() {
        return x;
    }

    /** Tar reda på y-koordinaten. */
    public int getY() {
        return y;
    }
    ...
```

Klassen Polygon

```
public class Polygon {
    private ArrayList<Point> vertices;

    /** Skapar en polygon med 0 hörnpunkter. */
    public Polygon() {
        vertices = new ArrayList<Point>();
    }

    /** Lägger till en ny punkt med koordinaterna x, y. */
    public void addVertex(int x, int y) {
        vertices.add(new Point(x, y));
    }

    /** Tar bort punkten på position pos. Punkterna numreras
        från 0 och uppåt i den ordning de lagts till. */
    public void removeVertex(int pos) {
        vertices.remove(pos);
    }

    ...
```

Klassen Polygon

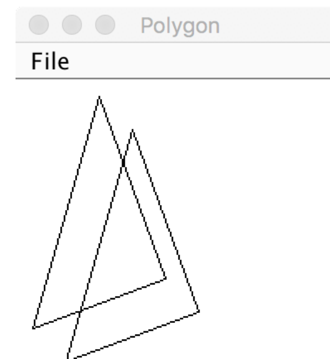
forts

```
...

/** Flyttar polygonen avståndet dx i x-led, dy i y-led. */
public void move(int dx, int dy) {
    for (Point p : vertices) {
        p.move(dx, dy);
    }
}

/** Ritar polygonen i fönstret w. */
public void draw(SimpleWindow w) {
    if (vertices.size() == 0) {
        return;
    }
    Point last = vertices.get(vertices.size() - 1);
    w.moveTo(last.getX(), last.getY());
    for (Point p : vertices) {
        w.lineTo(p.getX(), p.getY());
    }
}
```

Klassen Polygon

forts

```
public class PolygonExample {
    public static void main(String[] args) {
        Polygon p = new Polygon();
        p.addVertex(10, 150);
        p.addVertex(50, 10);
        p.addVertex(90, 120);
        SimpleWindow w = new SimpleWindow(200, 200, "Polygon");
        p.draw(w);
        p.move(20,  20);
        p.draw(w);
    }
}
```



Övning
Sökning

Lägg till följande metod i klassen `Polygon`:

```
/** Undersöker om det finns någon hörnpunkt
    med koordinaterna x, y. */
public boolean hasVertex(int x, int y) {









}
```

Sökmetoder i ArrayList<E>

- Det finns färdiga metoder för att söka i listan:

```
/** Söker upp ett element som matchar obj. Returnerar
    true om sådant element element finns, annars false. */
boolean contains(Object obj);

/** Söker upp ett element som matchar obj. Returnerar
    index för första förekomsten av elementet, -1 om
    elementet inte finns. */
int indexOf(Object obj);

/** Tar bort första förekomsten av objektet obj, om det
    finns. Returnerar true om ett element togs bort. */
boolean remove(Object obj);
```

- Klassen `Object` i Java kan användas som typ för alla slags objekt.
- Det ingår *inte* i kursen att använda dessa metoder.

Sökmetoder i `ArrayList<E>`

forts.

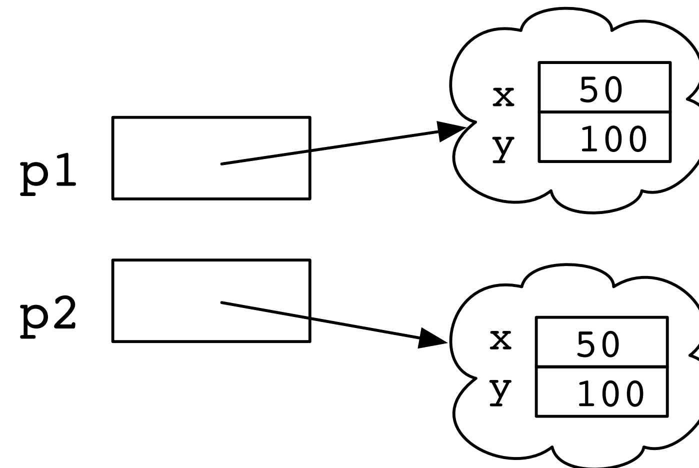
- Inuti `ArrayLists` sökmetoder (de med en parameter av typen `Object`) söks alla element igenom och jämförs med parametern `obj`.
- Vid jämförelsen används en metod `boolean equals(Object obj)`. Om den inte är implementerad på rätt sätt i elementens klass fungerar kanske inte sökningen som avsett.
- I Javas färdiga klasser finns metoden `equals` implementerad. Därför fungerar det att använda sökmetoderna om man använder en lista av typen ArrayList<Integer>, ArrayList<Double>, ArrayList<String>, ...
- Om man däremot har skrivit en egen klass `C` och lagrar objekt av `C` i en `ArrayList<C>` måste man själv implementera metoden `equals`.
  - Det är lite besvärligt och ingår inte i grundkursen.

Diskutera

Jämföra objekt - samma eller lika

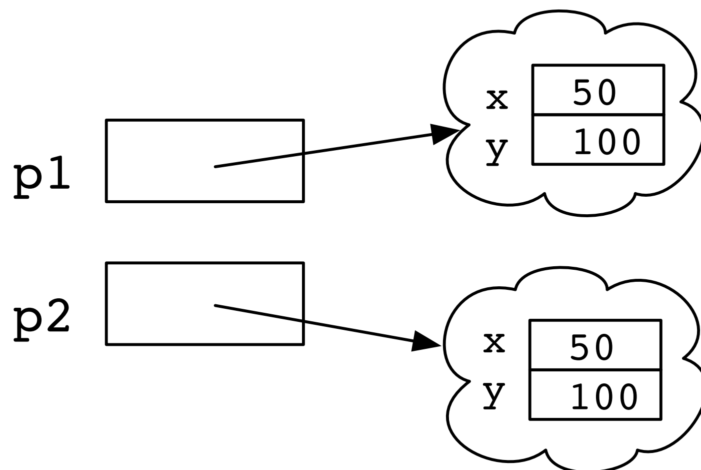
- Vad skrivs ut?

```
Point p1 = new Point(50, 100);
Point p2 = new Point(50, 100);

System.out.println(p1 == p2);
System.out.println(p1.getX() == p2.getX() &&
                   p2.getY() == p2.getY());
```



Jämföra objekt - samma eller lika

- När man jämför objekt måste man ha klart för sig vad man vill jämföra.
- Vi har två *olika* objekt med samma innehåll.
- Uttrycket `p1 == p2` undersöker om variablerna `p1` och `p2` har samma värde, dvs refererar till samma objekt.
- Uttrycket `p1.getX() == p2.getX() && p1.getY() == p2.getY()` undersöker om `p1` och `p2` har samma innehåll.



Sökmetoder i `ArrayList<E>` och metoden `equals`

Överkurs

- Antag att du skrivit en klass `C` och vill lagra objekt i en lista av typen `ArrayList<C>`.
- Antag också att du med likhet menar att två `C`-objekt är lika när de har samma värde på ett eller flera attribut.
  Om du vill använda sökmetoderna (`contains`, ...) måste du då skriva en egen `equals`-metod i klassen `C`.
- Finns det ingen metod `boolean equals(Object obj)` i klassen `C` är det istället referenser som jämförs.
- I läroboken finns ibland metoder `equals` som är implementerade på ett förenklat sätt och inte fungerar som avsett i sökmetoderna.
  - Metoden `equals` måste ha en parameter av typen `Object`.

Sökmetoder i `ArrayList<E>` och metoden `equals`

Överkurs, forts.

- Exempel på `equals`-metod i klassen `Point`:

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

    // konstruktor och metoder

    public boolean equals(Object obj) {
        if (! obj instanceOf Point) {
            return false;
        }
        Point p = (Point) obj;
        return x == p.x && y == p.y;
    }
}
```

Operatorn `instanceOf` ger `true` om `obj` refererar till ett objekt av rätt typ (här `Point`).

Insättning i sorterad följd

Exempel:

```
ArrayList<String> wordList= new ArrayList<String>();
wordList.add("apelsin");
wordList.add("banan");
wordList.add("druva");
wordList.add("enbär");
```

Uppgift: Lägg till ordet `word` i den redan sorterade listan.

`wordList` | "apelsin" | "banan" | "druva" | "enbär"

`word` | "citron"

Insättning i sorterad följd

Lösning: Gå framåt i listan tills det dyker upp ett ord som är ''för stort''. Sätt in `word` före detta ord:

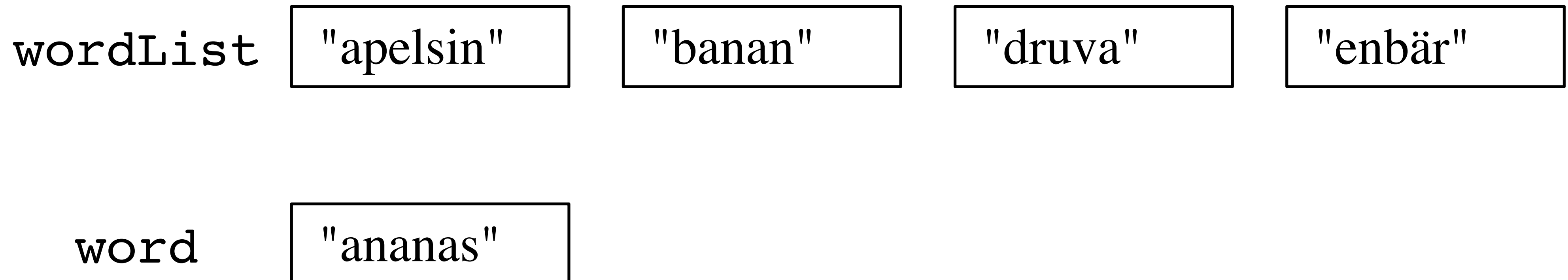
```
int pos = 0;
while (pos < wordList.size() &&
    wordList.get(pos).compareTo(word) < 0) {
    pos++;
}
wordList.add(pos, word);
```

wordList	"apelsin"	"banan"	"citron"	"druva"	"enbär"

Diskutera

Insättning i sorterad följd

- Vad händer `word` ska in först i listan?
- Vad händer `word` ska in sist i listan?
- Vad händer om listan är tom?



Insättning i sorterad följd

Alternativ

Lösning: Gå framåt i listan tills det dyker upp ett ord som är "för stort". Sätt in `word` före detta ord:

```
int pos;  // Måste deklareras utanför for-satsen för att
          // kunna användas i anropet av add.
for (pos = 0; pos < wordList.size(); pos++) {
    if (wordList.get(pos).compareTo(word) >= 0) {
        break;   // avbryter loopen
    }
}
wordList.add(pos, word);
```

wordList | "apelsin" | "banan" | "citron" | "druva" | "enbär"

Checklista

Exempel på vad du ska kunna

- Förklara begreppet generisk klass.
- Använda klassen `ArrayList<E>`.
  - deklarera och skapa en lista
  - sätta in element
  - ta bort element
  - traversera en lista och behandla alla element
  - söka efter ett element i listan
  - sätta in ett element i en sorterad följd