EDA016 Programmeringsteknik för D

Läsvecka 10: Listor

Björn Regnell

Datavetenskap, LTH

Lp1-2, HT 2015

10 Listor

- Att göra denna vecka
- Repetition: Vektorer: utöka, stoppa in, ta bort, etc.
- ArrayList

Att göra i Vecka 10: Förstå ArrayList och skillnader/likheter jämfört med primitiva vektorer.

1. Läs följande kapitel i kursboken: 8.3-8.5, 12
   Begrepp: ArrayList, generisk datastruktur, automatisk in- och upp-packning (auto-boxing/-unboxing), typklasser, utökad for-sats (for-each).
2. Gör övning 9: ArrayList
3. Träffas i samarbetsgrupper och hjälp varandra
4. Gör Lab 8: vektorer, simulering av patiens

Repetition: Primitiva vektorer

- Primitiva vektorer (Array med []) i Java har **fördelar**:
  - Det är den snabbaste indexerbara datastrukturen i JVM: att läsa och uppdatera ett element på en viss plats är mycket effektiv om man vet platsens index.
  - Fungerar lika bra med både primitiva värden och objektreferenser
- ... men också **nackdelar**:
  - Man måste bestämma sig för antalet element vid new.
  - Man kan ta i lite extra när man allokerar om man behöver plats för fler senare, men då måste man hålla reda på hur många platser man använder och veta var nästa lediga plats finns.
  - Det är krångligt att stoppa in (eng. *insert*) och ta bort (eng. *delete*) element.
  - Vill man ha fler platser måste man allokera en helt ny, större vektor och kopiera över alla befintliga element.

Polygon med primitiv vektorer

```
1  package week10.vector;
2
3  public class Polygon {
4      private Point[] vertices; // vektor med hörnpunkter
5      private int n;            // antalet hörnpunkter
6
7      /** Skapar en polygon */
8      public Polygon() {
9          vertices = new Point[1];
10         n = 0;
11     }
12
13     ...
```

Polygon med primitiv vektorer: stoppa in sist och vid behov skapa mer plats

Metoden `addVertex` i klassen `Polygon`
med attributet: `private Point[] vertices`

```
1     private void extend(){
2         Point[] oldVertices = vertices;
3         vertices = new Point[2 * vertices.length]; // skapa dubbel plats
4         for (int i = 0; i < oldVertices.length; i++) {  // kopiera
5             vertices[i] = oldVertices[i];
6         }
7     }
8
9     /** Definierar en ny punkt med koordinaterna x,y */
10    public void addVertex(int x, int y) {
11        if (n == vertices.length) extend();
12        vertices[n] = new Point(x, y);
13        n++;
14    }
```

Polygon med primitiv vektorer: stoppa in mitt i på angiven plats

Metoden `insertVertex` i klassen `Polygon` med attributet: **`private`** `Point[] vertices`

```
1     public void insertVertex(int pos, int x, int y) {
2         if (n == vertices.length) extend();   // utöka vid behov
3         for (int i = n; i > pos; i--) {       // flytta element bakifrån
4             vertices[i] = vertices[i - 1];
5         }
6         vertices[pos] = new Point(x, y);
7         n++;
8     }
```

Varför ArrayList?

En betydande nackdel med primitiva vektorer är att vi kan behöva ”uppfinna hjulet” upprepade gånger:

- För varje ny klass med vektor-attribut (vektor av Point, Person, Turtle, ...) som vi vill ska klara insert och append, blir det en hel del att implementera och testa...

Det vore smidigt med en datastruktur ...

- som inte kräver att vi känner antalet element från början,
- där vi enkelt kan lägga till och ta bort element,
- som kan hantera element av olika typ (likt vektorer).

Från och med version 5 av Java (2004) så introducerades **generics** vilket möjliggör skapandet av klasser som kan erbjuda generell behandling av olika typer av objekt. Generiska klasser känns igen med syntaxen `Klassnamn<Typ>`, till exempel `ArrayList<Point>`

Fördjupning: se javase tutorial, mer om detta i fördjupningskursen.

Vad är ArrayList?

`ArrayList` är en standardklass i paketet `java.util` med många fördelar:

- Lagrar sina element i en snabbindexerad primitiv vektor.
- Fungerar för alla typer av objekt.
- Utökar vektorns storlek av sig själv vid behov.

Det finns också vissa nackdelar:

- Fungerar **inte** med primitiva typer **`int`**, **`double`**, **`char`**, ... (men det finns sätt komma runt detta)
- Kräver visst onödigt minnesutrymme om vi vet antalet från början och inte behöver automatisk utökning.
- Likt primitiva vektorer tar det tid att göra insert och delete.

Polygon med ArrayList

Klassen `Polygon`, nu med ett attribut av typen `ArrayList<Point>` som håller reda på hörnpunkterna:

```
public class Polygon {
    private ArrayList<Point> vertices; // lista med hörnpunkter

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

    ...
```

Det behövs inget attribut n eftersom vi inte själva behöver hålla reda på antalet allokerade platser: allokering, insättning, och utökning av antalet platser sköts helt automatiskt av `ArrayList`-klassen vid behov.

Viktiga operationer på ArrayList (Urval)

ArrayList

```
/** Skapar en ny lista */
ArrayList<E>();

/** Tar reda på elementet på plats pos */
E get(int pos);

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

/** Lägger in obj på plats pos; efterföljande flyttas */
void add(int pos, E obj);

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

/** Tar reda på antalet element i listan */
int size();
```

Lär dig vad som finns om ArrayList i java snabbreferens!

Läs mer om ArrayList i javadoc.

Överkurs för den nyfikne: kolla implementation av ArrayList här.

ArrayList är en *generisk* klass

- `ArrayList` är en så kallad **generisk** klass. Se t.ex. wikipedia.
- Namnet **E** är en **typparameter** till klassen.
  (Mer om detta i Programmeringsteknik – fördjupningskurs.)
- Typparameterns namn kan användas i implementationen av en generisk klass och kompilatorn kommer att *ersätta* typparametern med den *egentliga* typen vid kompilering.
- I fallet `ArrayList`: **E** ersätts med typen på de objekt som egentligen lagras i listan.

Exempel:

```
ArrayList<Point> vertices = new ArrayList<Point>();
vertices.add(new Point(50, 50));
vertices.add(new Point(50, 10));
vertices.add(new Point(30, 40));
```

Polygon med ArrayList: metoderna blir enklare

```
public void addVertex(int x, int y) {
    vertices.add(new Point(x, y));
}

public void move(int dx, int dy) {
    for (int i = 0; i < vertices.size(); i++) {
     vertices.get(i).move(dx, dy);
    }
}

public void insertVertex(int pos, int x, int y) {
 vertices.add(pos, new Point(x, y));
}

public void removeVertex(int pos) {
 vertices.remove(pos);
}
```

Se hela lösningen här: