PROGRAMMERING FÖRDJUPNING

JAVA

DOCENDO

Till denna bok medföljer ett antal övningsfiler som du laddar ner från vår webbplats www.docendo.se:

1. Starta webbläsaren, skriv **www.docendo.se** i adressfältet och tryck på **Retur**.
2. Skriv artikelnumret, **1260**, i sökrutan och klicka på **Sök**.
3. Klicka på titeln **Programmering Java Fördjupning**.
4. Klicka på filen **1260.zip** högst upp på sidan.
5. Klicka på **Spara**, välj var du vill spara filen, exempelvis på skrivbordet, och klicka på **Spara**.
6. När filen har hämtats stänger du dialogrutan och avslutar webbläsaren.
7. Om du har valt att spara filen på skrivbordet visas den som en ikon med namnet **1260**. Dubbelklicka på ikonen för att packa upp filerna till lämplig mapp på din hårddisk.

Tryckeri: Elanders - Fälth & Hässler, Sverige 2012
Första upplagan, första tryckningen

ISBN: 978-91-7207-967-0
Artikelnummer: 1260

Författare: Jonas Byström

Redaktör: Iréne Friberg

Omslag: Malina Andrén

Bild på omslaget ©iStock

Innehållsförteckning

4 Länkade listor

I detta kapitel

Du ska nu tillverka någonting användbart med ”referens på referens” och klasser. Nu ska du koppla ihop dessa kunskaper för att göra en mycket enkel adressbok.

Dubbelriktade länkade listor

I det tidigare exemplet med en länkad lista var listan ”enkelriktad”. Det betyder att det bara gick att leta sig igenom den i en riktning: från det första elementet till det sista. En bättre lista är den dubbelriktade, där man kan söka både framåt och bakåt:

De dubbelriktade pilarna betyder att de bägge elementen i var ända pekar på varandra. Alla element har en referens till ”föregående element” och till ”nästkommande element”. Det första elementets referens till dess föregående element är **`null`** och det sista elementets referens till dess nästkommande element är **`null`**. På så sätt kan man lätt hitta det första och det sista elementet i en lista.

Gör en adressbok

Information till uppgiften

Adressboken ska byggas upp med en dubbellänkad lista och för varje person i listan ska det gå att lagra lite information, såsom namn, telefonnummer, gatuadress och stad. Det är lämpligt att du använder ett objekt för varje person. Den sista personen i listan markeras genom att referensen till nästa person är noll. Det betyder att en lista med fyra personer i ser ut som följer:

Programmet skriver du på engelska. Vi ska ta stöd av kod som medföljer materialet för att skapa en konsoll där vi kan skriva ut och mata in text, på ungefär samma sätt som ett klassiskt konsollprogram.

Vidare ska det gå att lägga till, ta bort och skriva ut personer i adressboken. Naturligt nog är det bra om det finns någon funktion för att avsluta programmet.

Du skriver programmet

1. Starta NetBeans och skapa ett projekt som du kallar **AddressBook**.
2. Lägg till Sdb-projektet till ditt projekt (högerklicka på **Libraries**, välj **Add Projects...**, dubbelklicka på projektet **Sdb**). Memorera detta steg så att du kan göra det vid behov i andra program du skapar längre fram.
3. Skriv in följande rader efter **`public class AddressBook {`**

```
class Person {
  public Person previousPerson;
  public Person nextPerson;
  public String name;
  public String phoneNumber;
  public String streetAddress;
  public String city;
}

private Person firstPerson = null;
```

Som du ser innehåller klassen **`Person`** alla personuppgifter som krävs för att klara uppgiften. I följande bild visas ett enkelt diagram över hur objekt av klassen **`Person`** kommer att användas. I detta exempel finns det tre personer lagrade i adressboken.

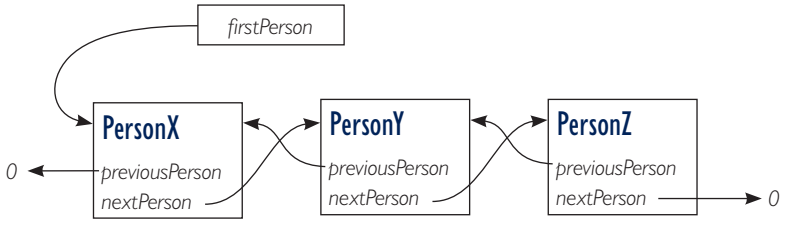


fig. 4.1

Referensen **`firstPerson`** pekar på det första objektet i listan, **`PersonX`**. Medlemsreferensen **`previousPerson`** i det första objektet, **`PersonX`**, pekar på **`null`** – det finns ju inget föregående element! Medlemsreferensen **`nextPerson`** i **`PersonX`** pekar på nästa objekt i listan, som här kallas **`PersonY`**.

I det andra elementet pekar **`previousPerson`** på det första elementet och **`nextPerson`** på det sista.

I det tredje elementet pekar **`previousPerson`** på det andra elementet och **`nextPerson`** pekar på **`null`**, eftersom det inte finns något nästa element.

Nu är det så bra att det fungerar likadant för en godtyckligt stor lista. En tom lista är en lista helt utan element och därför är referensen till det första elementet i listan (**`firstPerson`**) lika med **`0`**.

4. Skapa en konstruktor i klassen:

```
public AddressBook() {
}
```

5. I **`main`** skapar du en instans av **`AddressBook`**. Orsaken är att **`main`** är en så kallad **`static`**-metod, vilket betyder att den inte körs mot en instans utan mot klassen, och därför har den inte tillgång till medlemsvariabler. Detta är inget du behöver känna till mer om just nu, men du bör veta att för att kunna läsa och skriva till medlemsvariablerna i **`AddressBook`** så måste du instansiera klassen. (Att instansiera en klass är en annat uttryck för att skapa en instans av en klass.) Du gör det så här:

```
new AddressBook();
```

6. Skriv in följande rader i konstruktorn:

```
Sdb.createConsole();
System.out.println(”Address book.”);

boolean quit = false;
while (quit == false) {
  System.out.println(””);
  System.out.println(”Options:”);
  System.out.println(”1. Print address book”);
  System.out.println(”2. Add person”);
  System.out.println(”3. Remove person”);
  System.out.println(”4. Quit”);
  System.out.println(”Enter your choice (1-4): ”);
  int option = Sdb.in.nextInt();
  switch (option) {
    default: System.out.println(”Bad option!”);
                                  break;
    case 1:  printAddressBook(); break;
    case 2:  addPerson();        break;
    case 3:  removePerson();     break;
    case 4:  quit = true;        break;
  }
}
Sdb.closeConsole();
```

I källkoden kan man se att huvudloopen i programmet ligger i konstruktorn. Det är en **`while`**-loop som kommer att snurra till dess att den booleska variabeln **`quit`** blir skild från **`false`**. Titta längst ner i programmet så ser du att variabeln **`quit`** sätts till **`true`** om man väljer alternativ fyra. Således kommer alternativ fyra att avsluta programmet. Alla andra alternativ än 1, 2, 3 eller 4 hamnar under **`default:`** som ger felutskriften **`”Bad option!”`**.

Nu måste du skapa ytterligare tre metoder: en för att skriva ut alla personuppgifter i adressboken (**printAddressBook**), en för att lägga till personer i adressboken (**addPerson**) och slutligen en för att ta bort personer (**removePerson**). Dessa metoder kan du lägga ovanför eller under konstruktorn.

Skapa **printAddressBook** så här:

```
void printAddressBook() {
  Person p = firstPerson;
  while (p != null) {
    System.out.println("Name:   " + p.name);
    System.out.println("Phone:  " +
                       p.phoneNumber);
    System.out.println("Street: " +
                       p.streetAddress);
    System.out.println("City:   " + p.city);
    p = p.nextPerson;
  }
}
```

Referensen **p** används för att ”hålla tag i” elementet som ska skrivas ut. I fallet med en tom lista kommer **firstPerson** att vara **null** och då kommer programexekveringen helt att hoppa över **while**-loopen. Annars kommer, i slutet av **while**-loopen, **p** att tilldelas värdet av referensen till nästa element. På detta sätt kommer **p** i varje ny loop att peka på nästa element i listan – detta är en typisk användning av loopar.

När **p** tilldelas värdet av **nextPerson** i det sista elementet, så kommer uttrycket i **while**-satsen att bli falskt och även då avslutas metoden. Detta eftersom **nextPerson** i det sista elementet ju är **null**. Se fig. 4.1 om du känner dig tveksam.

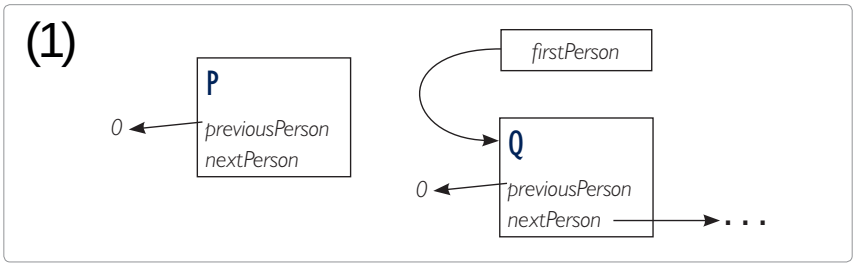
7. Skriv **addPerson** så här:

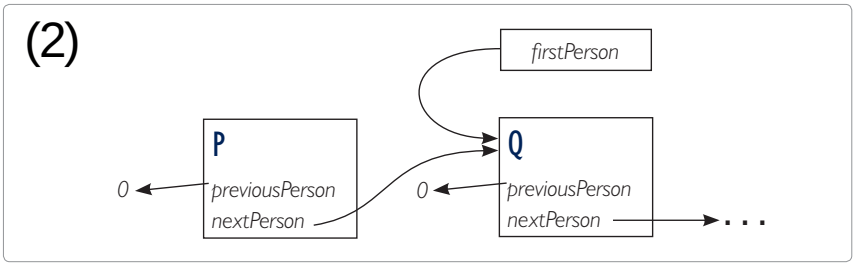
```
void addPerson() {
  Person p = new Person();
  Person q = firstPerson;
  p.previousPerson = null;
  p.nextPerson = q;
  if (q != null) {
    q.previousPerson = p;
  }
  firstPerson = p;

  System.out.println("Enter name: ");
```

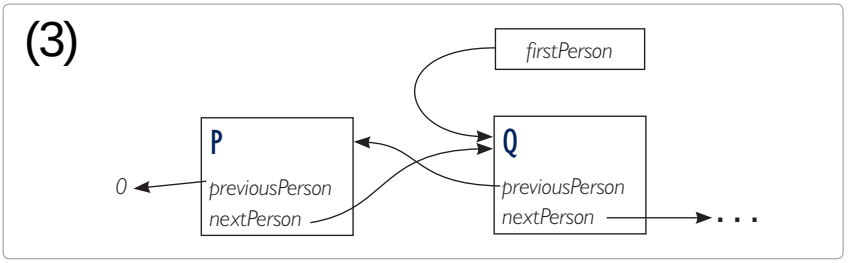
```
        p.name = Sdb.in.nextLine();
        System.out.println("Enter phone: ");
        p.phoneNumber = Sdb.in.nextLine();
        System.out.println("Enter street: ");
        p.streetAddress = Sdb.in.nextLine();
        System.out.println("Enter city: ");
        p.city = Sdb.in.nextLine();
    }
```

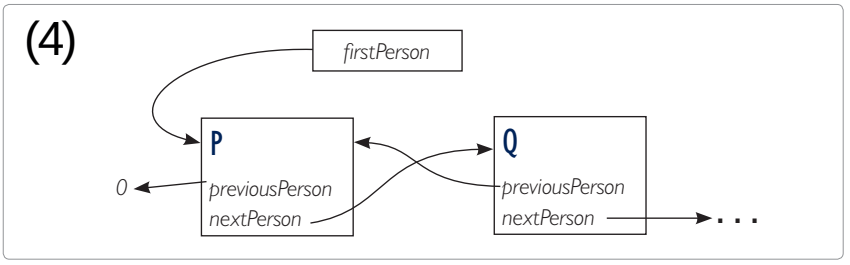
I **addPerson** skapas ett nytt objekt, **p**, med operatorn **new**, och en annan referens, **q**, tilldelas referensen till det första elementet i listan.

Nu ska **p** stoppas in först i listan. Se följande figurer. Referensen till det föregående elementet i **p** sätts till **null**, eftersom det nya objektet ska bli det första elementet i listan (1). Nästa element i listan kommer att bli **q**, och därför sätts referensen till nästa element i **p** att peka till (2). Sedan kontrolleras om listan tidigare var tom, men om listan inte var tom så måste det element som tidigare var det första peka på det element som nu blir dess föregångare (3). Slutligen i insättningen låter vi **firstPerson** peka på det nya elementet (4), och därmed är listan uppdaterad.









Efter att det nya objektet är instoppat i listan är det bara kvar att fylla i personuppgifterna. Personuppgifterna hamnar direkt i det nya objektet och sedan är metoden färdig.

8. **`removePerson`** ska se ut så här:

```
void removePerson() {
  System.out.println(”Enter person’s name: ”);
  String nameOfPerson = Sdb.in.nextLine();
  Person p = firstPerson;
  boolean deleted = false;
  while (p != null) {
    if (p.name.equals(nameOfPerson)) {
      Person prev = p.previousPerson;
      Person next = p.nextPerson;
      if (prev != null) {
        prev.nextPerson = p.nextPerson;
      }
      if (next != null) {
        next.previousPerson = p.previousPerson;
      }
      if (firstPerson == p) {
        firstPerson = next;
      }
      deleted = true;
    }
    p = p.nextPerson;
  }
```

```
        if (deleted != true) {
          System.out.println("Could not remove \"" +
              nameOfPerson +
              "\", person not found in address book!");
        }
      }
```

Först får användaren mata in personens namn, sedan kommer **`while`**-loopen att söka efter alla personer i listan med *exakt* det namnet. För varje person med det namnet kommer det elementet att kopplas från listan. Frånkoppling sker genom att det föregående elementet sätts att peka på nästa element och vise versa; dessutom kontrolleras om elementet som tas bort är det första elementet – i så fall måste den globala referensen **`firstPerson`** sättas att peka på nästa element.

Sist i metoden kontrolleras om någon har tagits bort ur adressboken. Om inget namn kunde matchas exakt så skrivs ett felmeddelande ut.

Nu är adressboken färdig! Den är flexibel i och med att du kan ta bort personer ifrån vilken plats som helst i listan utan att det blir ”glapp” eller andra problem.

9. Kompilera och testkör.

10. Mata in uppgifter för fem personer, ta bort några personer (från olika delar av listan) och lägg till några nya. Verifiera att programmet verkligen fungerar som det är tänkt!

11. Spara och avsluta NetBeans.

Övningsuppgifter

Övning 4.1

Skapa en klass för lagring av matvaror till en dubbelriktad länkad lista. Klassen ska innehålla minst tre publika medlemsvariabler, till exempel för pris.

Övning 4.2

Gör en valfri applikation med dubbelriktade länkade listor. Inget facit ges för denna uppgift. Några förslag på tänkbara program (rangordnade efter svårighetsgrad från enkel till svår):

1. En klasslista som kan lagra information om elevers namn, betyg och födelsedata.
2. En loggbok eller dagbok som bara lagrar datum, tid och text. Man ska kunna söka på datum för att skriva ut gamla noteringar. Om du vill kan du även lägga till sökning på fritext.
3. En applikation för parkeringshus som lagrar info om bilarna som parkeras där, till exempel ankomsttid och registreringsnummer. Priset är tidbaserat och ska anges när ett fordon åker ut ur parkeringshuset. Alla tider ska införas automatiskt, till exempel med metoden **`System.currentTimeMillis`**.
4. En applikation för varuhus. Info om varor såväl som anställda ska lagras. Vinst eller förlust ska presenteras med avseende på hur många varor som sålts och hur mycket pengar som utgått i lön. Om du vill kan du lägga till automatiska ”simuleringssteg” som slumpar fram varuförsäljning under en månad och sedan betalar ut månadslöner för alla anställda.

Övning 4.3

Utöka programmet i 4.2: lägg till funktioner för att spara och ladda data i de länkade listorna. Använd klasserna **`File`** för att referera en fil. Använd **`FileReader`** och **`BufferedReader`** för att läsa, **`PrintWriter`** för att spara och **`File.close`** för att stänga filen igen. Detta är en mycket svår uppgift, och tänkt för dig som kanske kan lite programmering sedan tidigare och ligger före. Det enklaste sättet att lösa uppgiften är att använda ett skiljetecken mellan typ och värde (till exempel kolon, **`Name:Urban`**) och att enbart lagra ett värde per rad (så att radslutsteckenet **`\n`** blir skiljetecken mellan värden).

PROGRAMMERING FÖRDJUPNING JAVA

Den här boken lämpar sig för fördjupningskurser i Java. Successivt kommer du att få prova på objektorienterad pogrammering med Java. Efter att ha gått igenom boken kan du skapa avancerade projekt med de snabbaste sorterings- och sökalgoritmerna. Du får också en kort introduktion till nätverksprogrammering (med TCP/IP) som du kan använda för att skapa program som kan kommunicera över hela världen. I slutet av boken får du också prova på programmering för OpenGL (3D).

I boken varvas teoridelar med steg för steg-beskrivningar som är lätta att följa. I tillämpningsuppgifterna får du arbeta självständigt och pröva dina nyvunna kunskaper. Övningsfilerna till boken laddar du ner utan kostnad från vår webbplats docendo.se.

Vår serie På rätt kurs är grund- och fördjupningsböcker där du genom att följa instruktioner lär dig viktiga funktioner i programmen. Teoridelar och övningar är sammanvävda och böckerna fungerar både för lärarledd undervisning och självstudier.

DOCENDO

ISBN 978-91-7207-967-0
9 789172 079670