Objektsamlingar i Java

Objektorienterad programmering
Laboration 3

Syfte

Att ge träning i att använda objektsamlingar i Java.

Mål

Efter övningen skall du kunna

- använda objektsamlingsklasserna `ArrayList` och `LinkedList`.
- hantera objektsamlingar med iteratorobjekt av typen `Iterator`.
- Skaffa tillräcklig information om ovanstående i Java API Specification.

Utvecklingsmiljö

BlueJ på valfri plattform.

Litteratur

Kursboken kap 1-4.

Färdig programkod

Given programkod finns på kursens hemsida i **Laborationer->Programkod för labbarna->club.zip**.

Labbgrupper

Arbetet genomförs och redovisas i grupper om **två** personer.

Redovisning

Senaste redovisningsdag, se kurs-PM samt Fire.
Ladda upp filerna `Membership.java` och `Club.java` med dina lösningar i, samt `README.txt` med svar på frågor, i Fire.

Glöm inte att trycka på **SUBMIT***!*

Uppgift 1

Medlemskap i en klubb representeras med en instans (ett objekt) av klassen `Membership`. Varje objekt av klassen innehåller detaljer om en medlems namn och e-postadress, samt vilken månad och år medlemskapet startade. E-postadressen identifierar varje klubbmedlem unikt. Många sajter på Internet använder just e-postadressen som användar-id. I klubben får alltså inte två olika medlemsobjekt ha samma e-postadress.
I den givna koden saknas hantering av epostadress i `Membership`. Lägg till detta! Inför lämplig instansvariabel, accessmetod, samt modifiera konstruktorn och metoden `toString` på lämpligt sätt.

Uppgift 2

Skriv färdigt klassen `Club`. Ett skelett till klassen finns i projektet `club`.

Klassen skall lagra medlemsobjekt i en länkad lista (`LinkedList`). Ett nytt medlemsobjekt adderas till ett klubbobjekt med metoden `join`, som har följande beskrivning och signatur:

```
/**
 * Add a new member to the club's collection of members.
 * @param member The member object to be added.
 */
public void join(Membership member)
```

Du får anta att medlemsobjektet som skickas som parameter redan har skapats och är korrekt initierat.

Klassen `Club` har även en metod som returnerar antalet medlemmar. Den har följande beskrivning och signatur:

```
/**
 * @return The number of members (Membership objects) in
 *         the club.
 */
public int numberOfMembers()
```

Antalet medlemmar motsvarar antalet medlemsobjekt som för tillfället lagras i listan.

Uppvärmning

För att klara dessa måste du ha läst fram t.o.m. avsnitt 4.7.

Gör klart definitionen av klassen `Club`. Följande krävs:

- En instansvariabel för den länkade listan av typen `LinkedList<Membership>`.
- En parameterlös konstruktor som skapar listobjektet.
- En fullständig definition av metoden `join`.
- En fullständig definition av metoden `numberOfMembers`.

Nedan följer ytterligare uppgifter som skall redovisas, men gör ovanstående först!

Möjlig arbetsgång:
Uppgiften kan brytas ner i små separata steg. **Kompilera** och **TESTA!** efter varje steg för att verifiera att dina ändringar och tillägg är korrekta! Stegen behöver ej redovisas, bara slutresultatet.

1. Leta upp klassen `LinkedList` i API Specification. Den finns i paketet `java.util`. Enklaste sättet att nå API Specification är via Help->Java Class Libraries. Om en annan version än 1.6 kommer upp så byt till denna genom att ändra till `http://download.oracle.com/javase/6/docs/api/` i webläsarens adressfält.
2. Definera en instansvariabel som skall användas till att referera till en länkad lista. Använd en lämplig `import`–deklaration för listklassen i början av filen. Skapa ett listobjekt i konstruktorn och tilldela det till listvariabeln.
3. Gör klart metoden `numberOfMembers` som returnerar antalet element i listobjektet. Metoden kommer givetvis inte att returnera annat än 0 tills du har gjort klart metoden `join` och lagt in medlemmar i klubben. Den måste alltså testas ytterligare när detta är gjort.
4. Gör klart metoden `join`.

Att lägga till medlemsobjekt till ett klubbobjekt från ”objektbänken” kan göras på två sätt:

5. Skapa ett nytt medlemsobjekt på objektbänken, anropa metoden `join` för klubbobjektet, och ange den aktuella parametern till `join` genom att klicka på medlemsobjektet.

- Anropa `join` för klubbobjektet och skriv in:

  `new` `Membership("`*medlemens namn*`","`*e-postadress*`",`*månad*`,`*år*`)`

  i dialogrutan för `join`-metodens parameter.

När du har lagt in en ny medlem skall du testa metoden `numberOfMembers` för att kontrollera a) att `join` verkligen lagt in ett nytt objekt i listan, b) att `numberOfMembers` fungerar.

Fler uppgifter

För att klara dessa måste du ha läst fram t.o.m. avsnitt 4.10.
All listhantering i de tre uppgifterna nedan skall ske med iteratorer.

- Leta upp `Iterator` i API Specification. Den finns i paketet `java.util` under rubriken Interfaces.
- Förbättra metoden `join` i `Club` så att den kontrollerar att den nya medlemmen inte redan är medlem i klubben. I så fall skall det nya medlemsobjektet inte läggas in i listan utan ett lämpligt felmeddelande skrivas ut. För att kunna avgöra om medlemmen finns i listan eller ej måste du undersöka samtliga element i den. Använd ett iteratorobjekt av typen `Iterator<Membership>` och lämplig loop-konstruktion.
  Att fundera på: Det kommer att behövas iteratorobjekt på flera ställen i lösningen. Bör dessa deklareras lokalt i resp. metod, eller räcker det med ett gemensamt iteratorobjekt i klassen som kan användas i de olika metoderna? Motivera! *Besvara frågan i README!*
- Definiera en metod i klassen `Club` med följande beskrivning och signatur:

```
/**
 * Return how many members joined in the given month.
 * @param month The month we are interested in.
 * @param year The year of the Membership.
 * @return How many members joined in the given month.
 */
public int joinedInMonth(int month, int year)
```

Om parametern `month` har ett värde utanför det giltiga intervallet 1..12 skall metoden skriva ut ett felmeddelande samt returnera 0.

- Definiera metoden `search`

```
/**
 * Search for a member with a given email address.
 * @return A matching membership object if found,
 *         null otherwise.
 */
public Membership search(String email)
```

- Definiera metoden `getEmailAddresses`

```
/**
 * Return the email addresses of all the club members.
 * @return A comma separated string of email addresses.
 */
public String getEmailAdresses()
```

Istället för att upprepade gånger bygga en sträng av e-postadresser med uttryck av formen `s = s + "text";` så kan du använda klassen `StringBuffer` (eller `StringBuilder`). Studera dem i API spec. Objekt av klassen `StringBuffer` är *muterbara*, till skillnad från `String`. Om en strängbuffert blir full allokeras automatiskt mer plats. Eftersom `String` ej är muterbar kan vi inte ändra strängen som `s` refererar till ovan, utan måste för varje ny e-postadress som skall läggas till, skapa ett helt nytt strängobjekt med uttrycket `s + "text"` som sedan tilldelas till referensvariabeln `s`. Om vi i stället deklarerar

```
StringBuffer sb = new StringBuffer(256);  // startkapacitet 256
```

så kan vi addera text utan att skapa nya objekt vid varje addition:

```
sb.append("TEXT");
sb.append("mertext");
String s = sb.toString(); // s innehåller nu "TEXTmertext"
```

- Leta upp klassen `ArrayList` i API Specification.
- Definiera en metod i klassen `Club` med följande beskrivning och signatur:

```
/**
 * Remove from the club's collection all members who joined
 * in the given month, and return them stored in a separate
 * collection object.
 * @param month The month of the Membership.
 * @param year The year of the Membership.
 */
public ArrayList<Membership> purge(int month,int year)
```

Om parametern `month` har ett värde utanför det giltiga intervallet 1..12 skall metoden skriva ut ett felmeddelande samt returnera **`null`** . Denna metod är något svårare att skriva än ovanstående. *Tips*: Utnyttja metoden `remove`. OBS, metoden finns både i listklassen och i iteratorklassen. Vilken bör du använda? **Vad skulle kunna hända om du blandar?** Läs om dem i API specification. *Besvara den sista frågan i README!*

Lycka Till!