Kontrollskrivning 3, Programmeringsteknik för D/C

2012–12–17, 8.00–13.00

Anvisningar: Preliminärt ger uppgifterna $10 + 10 + 10 + 10 = 40$ poäng. Tillåtet hjälpmedel: Java-snabbreferens.

Vi meddelar på kurshemsidan, `cs.lth.se/eda016`, när rättningen är klar. Vi hoppas vara klara före jul så att resultatet kan registreras i Ladok fredag 21 december.

Omkontrollskrivning på hela kursen äger rum tisdag 8 januari 2013 kl 14–19 i Vic 1A. Det behövs inte någon föranmälan till omkontrollskrivningarna. Det är tillåtet att försöka höja sitt betyg (”plussa”) vid en omkontrollskrivning.

---

1. Ett biljettbokningssystem som ska användas av en konsertarrangör ska skrivas. I systemet ska man kunna boka biljetter till olika föreställningar samt få biljetter utskrivna. Biljetterna är onumrerade (det innebär att alla biljetterna till en viss föreställning ser likadana ut). Det finns bara ett begränsat antal biljetter till varje föreställning.

   Den som har hand om bokning och utskrift av biljetter sitter vid en dator. Varje kund får vid bokning ett bokningsnummer som bekräftelse på bokningen. När kunden hämtar biljetterna uppger han eller hon sitt bokningsnummer, och då skrivs biljetterna ut. Man kan bara skriva ut biljetterna en gång.

   En föreställning beskrivs av ett objekt av klassen `Performance`:

   `Performance`

```
// ... här finns konstruktor och metoder som inte behövs för uppgiften

/** Tar reda på tiden för föreställningen. Tiden anges med datum och
    klockslag, till exempel "201212171900" */
String getTime();

/** Tar reda på antalet kvarvarande (obokade) biljetter */
int getTicketsLeft();

/** Bokar tickets biljetter (före anrop har man kontrollerat att det finns
    tillräckligt antal biljetter kvar) */
void reserveTickets(int tickets);
```

   I ett objekt av klassen `PerformanceList` finns uppgifter om samtliga föreställningar:

   `PerformanceList`

```
// ... här finns konstruktor och metoder som inte behövs för uppgiften

/** Letar upp föreställningen som ges vid tiden time. Returnerar null om det
    inte finns någon sådan föreställning (det förutsätts att det inte ges två
    föreställningar vid samma tid) */
Performance findPerformance(String time);
```

(fortsättning nästa sida)

En bokning, som kan gälla flera biljetter till samma föreställning, beskrivs av klassen `Reservation`. Varje bokning har ett bokningsnummer, som är 1 för den första bokningen och som ökas med 1 för varje ny bokning.

`Reservation`

```
/** Skapar en bokning med bokningsnumret reservationNbr, som beskriver att
    tickets biljetter är bokade till föreställningen perf */
Reservation(int reservationNbr, int tickets, Performance perf);

/** Tar reda på bokningsnumret */
int getReservationNbr();

/** Tar reda på antalet biljetter som är bokade */
int getNbrTickets();

/** Skriver ut en biljett */
void printOneTicket();
```

Alla bokningarna har samlats i ett objekt av klassen `ReservationList`:

`ReservationList`

```
/** Skapar en tom lista för bokningar */
ReservationList();

/** Tar reda på bokningen med bokningsnummer reservationNbr. Returnerar null
    om det inte finns någon sådan bokning */
Reservation find(int reservationNbr);

/** Skapar en bokning med nummer = 1 + högsta hittills använda nummer, ökar
    detta nummer, lägger in bokningen i listan, returnerar bokningsnumret.
    Det förutsätts att det finns tillräckligt antal biljetter */
int insertReservation(int tickets, Performance perf);

/** Tar bort bokningen r ur listan */
void delete(Reservation r);
```

Klasserna `Performance`, `PerformanceList` och `Reservation` är färdigskrivna. Implementera klassen `ReservationList`.

2. Implementera följande klass (se uppgift 1 för en beskrivning av klasserna som används):

`ReservationManager`

```
/** Skapar ett objekt som hanterar bokningar till föreställningarna i
    listan pl. Bokningarna läggs in i listan rl */
ReservationManager(PerformanceList pl, ReservationList rl);

/** Försöker boka tickets biljetter till föreställningen vid tiden time:
      1) Om en föreställning med tiden time inte finns, skriv ut texten
         "Ingen sådan föreställning"
      2) Om föreställningen finns men antalet kvarvarande biljetter är
         för litet, skriv ut texten "För få biljetter kvar"
      3) Annars: reservera biljetterna till föreställningen, lägg in
         en bokning i bokningslistan, skriv ut bokningsnumret */
void makeReservation(String time, int tickets);

/** Skriver ut biljetterna för bokningen med bokningsnummer reservationNbr:
      1) Om bokningen inte finns, skriv ut texten "Ingen sådan bokning"
      2) Annars: skriv ut rätt antal biljetter, tag bort bokningen */
void printTickets(int reservationNbr);
```

3. I en del lekar, till exempel kurragömma, väljs en person ut för en särskild uppgift. Ett sätt att göra valet är att personerna ställer sig i en ring varefter man räknar med en ramsa, t ex Ole-dole-doff-. . . , fram till en person som tas bort. Sedan upprepar man ramsan med början från nästa person och en annan person tas bort. Detta fortsätter tills det bara finns en person i ringen.

   Antag att det finns fem personer `A-E` i ringen och att räkneramsans längd är 3. Med start på `A` tas personerna bort i följande ordning:

`A B C D E`	start på `A` ⇒ `C` tas bort
`A B D E`	start på `D` ⇒ `A` tas bort
`B D E`	start på `B` ⇒ `E` tas bort
`B D`	start på `B` ⇒ `B` tas bort
`D`	`D` är utvald

   Skriv en `main`-metod som utför valet enligt följande beskrivning:

   1. Läs in antalet personer (`nbrPersons`) och räkneramsans längd (`n`) från tangentbordet.
   2. Läs in namnen på de `nbrPersons` personerna (varje namn står på en egen rad) och lägg in personerna i en ring.
   3. Låt räknandet börja på personen på plats 1 i ringen.
   4. Så länge det finns mer än en person kvar i ringen: räkna till `n` och tag bort personen på aktuell plats.
   5. Skriv ut namnet på den utvalde personen.

   Nedanstående klass[1] *ska* utnyttjas i lösningen. Du ska inte implementera klassen i denna uppgift.

```
/** Skapar en tom ring */
Ring();

/** Sätter in personen p "sist i ringen" (efter senast insatta objekt) */
void insert(Person p);

/** Sätter aktuell plats till "början av ringen" (det objekt som sattes
    in först) */
void reset();

/** Räknar till n i ringen med början på aktuell plats (aktuell plats
    flyttas) */
void countTo(int n);

/** Tar bort objektet på aktuell plats, returnerar det borttagna objektet.
    Aktuell plats flyttas till efterföljaren */
Person removeCurrent();

/** Tar reda på antalet objekt i ringen */
int size();
```

   Klassen Person har följande utseende:

```
public class Person {
    private String name;
    public Person(String name) { this.name = name; }
    public String getName() { return name; }
}
```

4. Implementera klassen `Ring` från uppgift 3.

[1] Klassen beskriver en ring av personer – i fördjupningskursen kommer du att lära dig hur man skriver en klass som beskriver en ring av godtyckliga objekt, en ”generisk” ringklass.