

## Omkontrollskrivning, Programmeringsteknik för D/C

**2012-04-11, 14.00-19.00**

*Anvisningar:* Preliminärt ger uppgifterna  $5 + 17 + 13 + 5 = 40$  poäng. För godkänt betyg krävs ungefär 25 poäng. Tillåtet hjälpmedel: Java-snabbreferens.

Betyget på denna skrivning utgör betyget på kontrollskrivningarna, oavsett resultatet på tidigare skrivningar (dock sänks inte ett tidigare godkänt betyg). Resultatlistan anslås på institutionens anslagstavla när rättningen är klar.

---

1. I uppgift 1–2 ska du skriva delar av ett program som skulle kunna användas av ett antal samverkande bibliotek. Varje bok har ett unikt id-nummer och tillhör ett visst bibliotek. Man kan lämna tillbaka en lånad bok på vilket bibliotek som helst, och då skickas den tillbaka till rätt bibliotek. Lånetiden för böcker är normalt 3 veckor. Om en bok är reserverad (dvs om det finns flera personer som vill låna den på en väntelista) är lånetiden dock endast 10 dagar.

Följande klasser kan tänkas ingå i ett sådant program (bara de operationer som är intressanta för uppgiften finns med):

Book (*färdigskriven*)

```
/** Skapar en bok med titeln title och id-nummer id som tillhör
    biblioteket libraryId */
Book(String title, int id, int libraryId);

/** Ger bokens id-nummer */
int getId();

/** Ger bokens titel */
String getTitle();

/** Ger biblioteket som boken tillhör */
int getLibraryId();

/** Ger true om boken är reserverad, annars false */
boolean isReserved();
```

Person (*färdigskriven*)

```
/** Ger personens namn */
String getName();
```

Klassen BorrowObject beskriver ett boklån. Alla datum är strängar på formen "2012-04-11".

BorrowObject

```
/** Skapar ett låneobjekt som beskriver att personen person lånat boken
    book idag */
BorrowObject(Person person, Book book);

/** Ger personen */
Person getPerson();

/** Ger boken */
Book getBook();

/** Ger utlåningsdatum */
String getBorrowDate();

/** Ger sista återlämningsdatum */
String getLastReturnDate();

/** Ger det datum boken återlämnades. Om boken inte är återlämnad returneras
    "Not returned" */
String getReturnedDate();

/** Markerar boken som återlämnad idag */
void setReturned();
```

(Fortsättning nästa sida.)

### DateKeeper (färdigskriven)

```
/** Ger dagens datum på formen 2012-04-11 */
static String today();

/** Ger datumet n dagar från dagens datum (framåt i tiden om n > 0,
    bakåt om n < 0) */
static String getDate(int n);
```

Implementera klassen BorrowObject.

2. Biblioteket håller reda på böckerna och lånen i ett objekt av klassen Register. Man håller reda på både aktuella lån (dvs av böcker som ej är återlämnade) och gamla lån (dvs av böcker som är återlämnade).

### Register

```
/** Skapar ett biblioteksregister */
Register();

/** Noterar boken med id-nummer bookId som lånad av person idag */
void borrowedBook(int bookId, Person person);

/** Noterar boken med id-nummer bookId som återlämnad på biblioteket libraryId
    idag. Returnerar -1 om boken ej fanns noterad som utlånad, 0 om boken
    lämnades tillbaka på sitt eget bibliotek, annars biblioteks-id för det
    bibliotek som boken tillhör */
int returnedBook(int bookId, int libraryId);

/** Ger en lista på alla lån som ska återlämnas om days dagar */
ArrayList<BorrowObject> getWarning(int days);

// Här finns operationer för att till exempel lägga till och ta bort böcker,
// hantera reservationer, m.m. Dessa operationer ska du inte implementera!
```

Implementera klassen Register.

Anvisningar:

- Du ska använda tre listor i lösningen: en för böckerna, en för aktuella lån och en för gamla lån.
- För att underlätta uppletningen av böcker i boklistan ska du implementera och använda följande privata hjälpmetod i klassen:

```
/** Letar upp boken med id-nummer bookId, null om boken inte finns */
private Book findBook(int bookId) {
    ...
}
```

3. Vi ska testa om det finns någon effektivitetsskillnad om man använder en vanlig vektor eller en lista av typen `ArrayList` för att lagra heltal. Testet består av att vi mäter hur lång tid det tar att:

1. Skapa vektorn eller listan och fylla den med ett antal slumpmässiga heltal. Alla talen är olika.
2. Ta bort en slumpmässig delmängd av heltalen.

För att jämförelsen ska bli rättvis måste man använda samma testfall för både vektorn och listan. Klassen `TestCase` beskriver ett sådant testfall.

*TestCase (färdigskriven)*

```
/** Skapar ett testfall bestående av totalSize slumpmässiga positiva olika
    heltal, samt en slumpmässig delmängd av dessa (partSize stycken,
    partSize <= totalSize) i slumpmässig ordning */
TestCase(int totalSize, int partSize);

/** Returnerar alla heltalen i testfallet */
int[] getAllNumbers();

/** Returnerar delmängden av heltalen i testfallet */
int[] getPartNumbers();
```

Skriv ett huvudprogram som genomför ett effektivitetstest enligt ovan.

Anvisningar:

- Det totala antalet tal och antalet tal i delmängden ska läsas in i början av programmet. Du ska naturligtvis använda klassen `TestCase` för testfallet.
- Vid borttagning av tal ska alltid de aktuella talen finnas i början av vektorn/listan. Ordningen mellan de kvarvarande talen ska bibehållas.
- Programmet ska skriva ut hur lång tid respektive test tog (i millisekunder). För att mäta tiden ska du använda metoden `currentTimeMillis()` i klassen `System`:

```
/** Returnerar tidsskillnaden, mätt i millisekunder, mellan aktuell tid och
    midnatt 1970-01-01 */
static long currentTimeMillis();
```

4. I klassen `TestCase` i uppgift 3 ska man dra många slumpstal som alla är olika. Slumptalen ska lagras i en vektor (den vektor som returneras av `getAllNumbers`). Man måste för varje nytt tal kontrollera om det har dragits tidigare. Det kan man göra genom att undersöka om det dragna talet redan finns i vektorn, men det tar lång tid om antalet tal är stort. Bättre är att utnyttja standardklassen `HashSet`, som beskriver en mängd av objekt. Klassen har en operation `contains` som *snabbt* kontrollerar om ett objekt finns i mängden.

Klassen har följande (förenklade och något förkortade) specifikation:

```
/** Skapar en tom mängd för objekt av typen E */
HashSet<E>();

/** Lägger in objektet e i mängden, om det inte redan finns där */
void add(E e);

/** Undersöker om objektet e finns i mängden (då returneras true) */
boolean contains(E e);
```

Skriv programrader som 1) skapar en heltalsvektor `allNumbers` med `totalSize` element, 2) fyller vektorn med slumpmässiga positiva heltal som alla är olika. Gör kontrollen av att talen är olika med hjälp av ett `HashSet`-objekt.