Consider the problem of representing a filing cabinet with drawers of student records. A filing cabinet is implemented using a linked list of drawers. Each drawer is implemented using a linked list of student records. All student records in a drawer have a student ID less than or equal to the drawer's maximum student ID, and student records are stored in a drawer in ascending order by student ID.

The diagram below illustrates the structure of a filing cabinet as implemented by the class `FilingCabinet`. The data member `myDrawerList` is an instance of `java.util.LinkedList` that stores `Drawer` objects in ascending order by maximum ID.

The class `Student` is declared as follows.
```java
public class Student {
    // constructor and data members not shown
    // returns id of this student
    public int getID() {
        // not shown
    }

    // returns name of this student
    public String getName() {
        // not shown
    }

    // precondition: o is an instance of student
    // postcondition: returns true if o equals this student
    //                otherwise returns false
    public boolean equals(Object o) {
        // not shown
    }
}

The class Drawer is declared as follows.

```java
public class Drawer {
    private int myMaxID;            // maximum ID in this drawer
    private LinkedList myStudents;  // all students in this drawer

    // constructor and some methods not shown
    // returns maximal ID for this drawer
    public int getMaxID() {
        return myMaxID;
    }

    // add s to this drawer so students are in ascending order by ID
    public void addStudent(Student s) {
        // you will write this
    }

    // return an iterator for the students in this drawer
    public Iterator iterator() {
        return myStudents.iterator();
    }
}

The class FilingCabinet is declared as follows.

```
public class FilingCabinet
{
    private LinkedList myDrawerList;

    // precondition: this filing cabinet has at least one Drawer;
    //                studentID is less than or equal to maximum ID
    //                of last Drawer
    // postcondition: returns the first Drawer d such that
    //                d.getMaxID() >= studentID

    public Drawer findDrawer(int studentID)
    {
        // you will write this
    }

    // precondition: student.getID() <= maximum ID of last Drawer
    // postcondition: student added to proper Drawer

    public void addStudent(Student student)
    {
        Drawer d = findDrawer(student.getID());
        d.addStudent(student);
    }

    // precondition: student.getID() is less than or equal to
    //                maximum ID of last Drawer
    // postcondition: if there is a Student s in this filing cabinet
    //                equal to student, then s is removed from the
    //                drawer in which it is located; otherwise this
    //                FilingCabinet is unchanged

    public void removeStudent(Student student)
    {
        // you will write this
    }
}

Part A

Write FilingCabinet method findDrawer, which is described as follows. Method findDrawer returns the Drawer object in which studentID would be found. Method findDrawer returns the first Drawer in the list myDrawerList for which studentID is less than or equal to the maximum student ID number that can be filed in the drawer.

Complete findDrawer below.

// precondition: this filing cabinet has at least one Drawer;
//                studentID is less than or equal to maximum ID
//                of last Drawer
// postcondition: returns the first Drawer d such that
//                d.getMaxID() >= studentID

public Drawer findDrawer(int studentID)
{
}

**Part B**

Write the `FilingCabinet` method `removeStudent`, which is described as follows. Method `removeStudent` removes the `Student` object equal to `student` from the `FilingCabinet` if there is such an object. If there is no such object then the `FilingCabinet` is unchanged.

In writing `removeStudent`, you may call `findDrawer` specified in part (a). Assume that `findDrawer` works as specified, regardless of what you wrote in part (a).

Complete method `removeStudent` below.

```java
// precondition: student.getID() is less than or equal to
// maximum ID of last Drawer
// postcondition: if there is a Student s in this filing cabinet
// equal to student, then s is removed from the
// drawer in which it is located; otherwise this
// FilingCabinet is unchanged

public void removeStudent(Student student) {
}
```

**Part C**

Write the `Drawer` method `addStudent` which is described as follows. Method `addStudent` inserts the `Student` object `s` into `LinkedList` object `myStudents` so that the linked list is maintained in increasing order by student ID.

Assume that the `Drawer` constructor initializes `myStudents` to be a `LinkedList` containing no objects.

You may find the following algorithm helpful in implementing `Drawer` method `addStudent`.

- If the linked list `myStudents` is empty, add the new student anywhere in the linked list.

- If the new student's ID is less than the ID of the first student in `myStudents`, then add the new student at the beginning of the linked list.

- Similarly, if the new student's ID is greater than the ID of the last student in `myStudents`, then add the new student to the end of the linked list.

- (Otherwise there at least two objects in the linked list.) Use two `ListIterator` objects for `myStudents` and the `ListIterator` method `add`. The call to `add` should occur between the calls of `next` on the `ListIterator` objects.

Complete `Drawer` method `addStudent` below.
public class Drawer
{
    private LinkedList myStudents; // list of students in this drawer

    // add s to this drawer so students are in ascending order by ID

    public void addStudent(Student s)
    {
    
    }
}

Owen L. Astrachan
Last modified: Wed May 14 17:11:46 EDT 2003