

## 2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (a) Write the constructor for the `SeatingChart` class. The constructor initializes the `seats` instance variable to a two-dimensional array with the given number of rows and columns. The students in `studentList` are copied into the seating chart in the order in which they appear in `studentList`. The students are assigned to consecutive locations in the array `seats`, starting at `seats[0][0]` and filling the array column by column. Empty seats in the seating chart are represented by `null`.

For example, suppose a variable `List<Student> roster` contains references to `Student` objects in the following order.

"Karen" 3	"Liz" 1	"Paul" 4	"Lester" 1	"Henry" 5	"Renee" 9	"Glen" 2	"Fran" 6	"David" 1	"Danny" 3
--------------	------------	-------------	---------------	--------------	--------------	-------------	-------------	--------------	--------------

A `SeatingChart` object created with the call `new SeatingChart(roster, 3, 4)` would have `seats` initialized with the following values.

	0	1	2	3
0	"Karen" 3	"Lester" 1	"Glen" 2	"Danny" 3
1	"Liz" 1	"Henry" 5	"Fran" 6	null
2	"Paul" 4	"Renee" 9	"David" 1	null

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

Part (a) continues on page 9.

## 2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

Complete the `SeatingChart` constructor below.

```
/** Creates a seating chart with the given number of rows and columns from the students in
 * studentList. Empty seats in the seating chart are represented by null.
 * @param rows the number of rows of seats in the classroom
 * @param cols the number of columns of seats in the classroom
 * Precondition: rows > 0; cols > 0;
 *                 rows * cols >= studentList.size()
 * Postcondition:
 *   - Students appear in the seating chart in the same order as they appear
 *     in studentList, starting at seats[0][0].
 *   - seats is filled column by column from studentList, followed by any
 *     empty seats (represented by null).
 *   - studentList is unchanged.
 */
public SeatingChart(List<Student> studentList,
                     int rows, int cols)
```

Part (b) begins on page 10.

## 2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (b) Write the `removeAbsentStudents` method, which removes students who have more than a given number of absences from the seating chart and returns the number of students that were removed. When a student is removed from the seating chart, a `null` is placed in the entry for that student in the array `seats`. For example, suppose the variable `SeatingChart introCS` has been created such that the array `seats` contains the following entries showing both students and their number of absences.

	0	1	2	3
0	"Karen" 3	"Lester" 1	"Glen" 2	"Danny" 3
1	"Liz" 1	"Henry" 5	"Fran" 6	null
2	"Paul" 4	"Renee" 9	"David" 1	null

After the call `introCS.removeAbsentStudents(4)` has executed, the array `seats` would contain the following values and the method would return the value 3.

	0	1	2	3
0	"Karen" 3	"Lester" 1	"Glen" 2	"Danny" 3
1	"Liz" 1	null	null	null
2	"Paul" 4	null	"David" 1	null

Class information repeated from the beginning of the question:

```

public class Student

public String getName()
public int getAbsenceCount()

public class SeatingChart

private Student[][] seats
public SeatingChart(List<Student> studentList,
                    int rows, int cols)
public int removeAbsentStudents(int allowedAbsences)

```

## **2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Complete method `removeAbsentStudents` below.

```
/** Removes students who have more than a given number of absences from the
 * seating chart, replacing those entries in the seating chart with null
 * and returns the number of students removed.
 * @param allowedAbsences an integer >= 0
 * @return number of students removed from seats
 * Postcondition:
 * - All students with allowedAbsences or fewer are in their original positions in seats.
 * - No student in seats has more than allowedAbsences absences.
 * - Entries without students contain null.
 */
public int removeAbsentStudents(int allowedAbsences)
```

## 2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

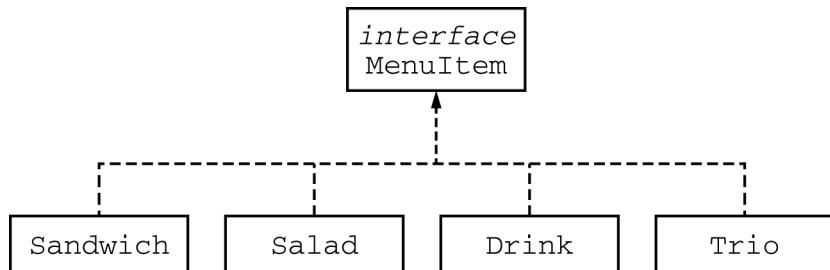
4. The menu at a lunch counter includes a variety of sandwiches, salads, and drinks. The menu also allows a customer to create a "trio," which consists of three menu items: a sandwich, a salad, and a drink. The price of the trio is the sum of the two highest-priced menu items in the trio; one item with the lowest price is free.

Each menu item has a name and a price. The four types of menu items are represented by the four classes Sandwich, Salad, Drink, and Trio. All four classes implement the following MenuItem interface.

```
public interface MenuItem
{
    /** @return the name of the menu item */
    String getName();

    /** @return the price of the menu item */
    double getPrice();
}
```

The following diagram shows the relationship between the MenuItem interface and the Sandwich, Salad, Drink, and Trio classes.



For example, assume that the menu includes the following items. The objects listed under each heading are instances of the class indicated by the heading.

Sandwich	Salad	Drink
"Cheeseburger" 2.75	"Spinach Salad" 1.25	"Orange Soda" 1.25
"Club Sandwich" 2.75	"Coleslaw" 1.25	"Cappuccino" 3.50

Question 4 continues on page 13

# AP® COMPUTER SCIENCE A 2014 CANONICAL SOLUTIONS

## Question 4: Trio

```
public class Trio implements MenuItem {  
    private Sandwich sandwich;  
    private Salad salad;  
    private Drink drink;  
  
    public Trio(Sandwich s, Salad sal, Drink d){  
        sandwich = s;  
        salad = sal;  
        drink = d;  
    }  
  
    public String getName(){  
        return sandwich.getName() + "/" + salad.getName() + "/" +  
            drink.getName() + " Trio";  
    }  
  
    public double getPrice(){  
        double sandwichPrice = sandwich.getPrice();  
        double saladPrice = salad.getPrice();  
        double drinkPrice = drink.getPrice();  
        if (sandwichPrice <= saladPrice && sandwichPrice <= drinkPrice)  
            return saladPrice + drinkPrice;  
        else if (saladPrice <= sandwichPrice && saladPrice <= drinkPrice)  
            return sandwichPrice + drinkPrice;  
        else  
            return sandwichPrice + saladPrice;  
    }  
}
```

### Alternate

```
public class Trio implements MenuItem {  
    private String name;  
    private double price;  
  
    public Trio(Sandwich s, Salad sal, Drink d){  
        double sandwichPrice = s.getPrice();  
        double saladPrice = sal.getPrice();  
        double drinkPrice = d.getPrice();  
        if (sandwichPrice <= saladPrice && sandwichPrice <= drinkPrice)  
            price = saladPrice + drinkPrice;  
        else if (saladPrice <= sandwichPrice && saladPrice <= drinkPrice)  
            price = sandwichPrice + drinkPrice;  
        else  
            price = sandwichPrice + saladPrice;  
        name = s.getName() + "/" + sal.getName() + "/" + d.getName() + " Trio";  
    }  
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.

# **AP<sup>®</sup> COMPUTER SCIENCE A 2014 CANONICAL SOLUTIONS**

## **Question 4: Trio continued**

```
public String getName() {  
    return name;  
}  
  
public double getPrice() {  
    return price;  
}  
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.