1.204 Quiz 1 Solutions

Spring 2009

Name ____

Exam guidelines:

1) 80 minutes are allowed to complete the quiz.
2) Open notes; open book.
3) There are 4 questions (100 points) and 7 pages (including this one) in the exam booklet.
4) No laptop computers, calculators, cell phones or messaging devices are allowed. Please turn off any that you have brought.
5) Please write legibly – you are welcome to use both sides of the paper; we can provide additional paper if necessary.

Question 1. Data modeling (25 points)

Create a model of a railway reservation system. A passenger can make a reservation for a seat on a train on a date between an origin station and a destination station. A single reservation can cover up to 6 people traveling together; one seat is reserved for each person in the group. Each reservation has an ID and one passenger's name; the remaining passengers in the group are not identified in any way. The reservation system contains all trains and all stations that they serve (and other information, such as departure and arrival times, that you do not have to consider in this problem). Train numbers are unique on a given day. Seats on a train are numbered uniquely. Station names are unique. You do not need to model whether seats are available or not; assume the requested reservation (train, date, number of passengers) can be made.

Draw a data model that corresponds to this set of system rules. You only need to create one drawing that includes all the elements listed in steps a-e.

a. **Draw a box for each entity: give each an appropriate name**

b. **List the attributes in the box for each entity**

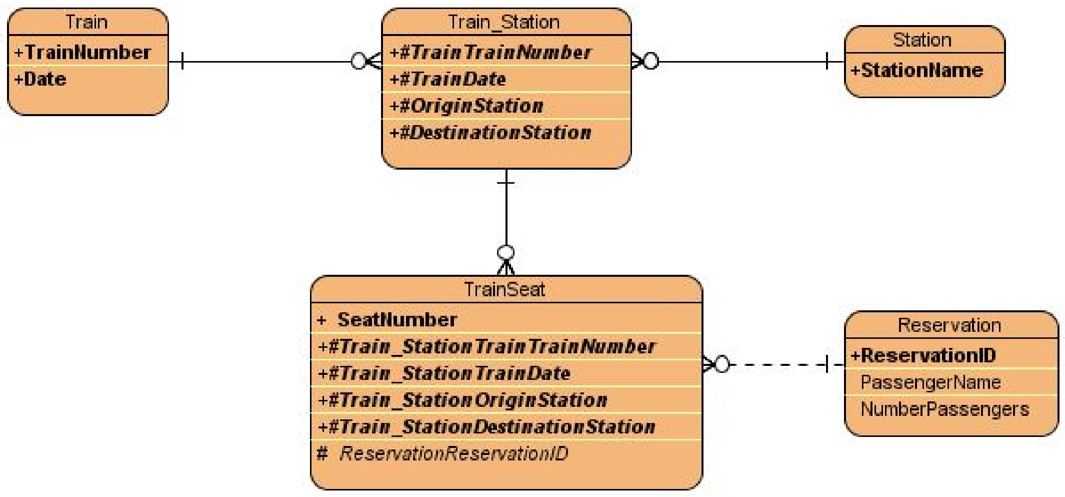
c. **Indicate the primary key for each entity by placing the phrase (PK) next to its name.**

d. **Draw all relationships between the entities in the model. Indicate foreign keys by placing the phrase (FK) next to attributes that are foreign keys.**

e. **Indicate the cardinality of the relationship: many-many, many-one or one-one. Use crows-foot notation; if you use another notation, define it.**

To repeat: You only need to create one drawing that includes all the elements listed in steps a-e above.

Please draw your data model on this page.



2. SQL (20 points)

Assume that the data model you built in the previous question has been implemented exactly as you drew it in a relational database management system such as SQL Server. Each entity is stored as a table, and attributes, keys and relationships are as you specified them. Write the SQL query to give the number of seats reserved on every train on every date.

```
SELECT TrainSeat.TrainNumber, TrainSeat.TrainDate, COUNT(*) AS
NumberReservedSeats
FROM Reservation INNER JOIN TrainSeat
ON Reservation.ReservationID = TrainSeat.ReservationID
GROUP BY TrainSeat.TrainNumber, TrainSeat.TrainDate
```

3. Data structures. (25 points)

a. The method `public Object remove(int k) throws NoSuchElementException`, which deletes the kth element, is added to the Queue class**.** What is its worst case running time? Give a brief justification. The key parts of the Queue class code are given at the end of the quiz for reference.

Deleting the kth element requires the following steps:

- *Select the kth element; this is done by finding queue[(front+k) % capacity]. This is just one instruction, regardless of the number of items in the queue*
- *Every element after the kth element must then be moved up one position in the queue array. On average, n/2 items will be moved. This is the critical step, so remove(int k) is an O(n) operation*

b. The method `public Object removeLast() throws NoSuchElementException`, which removes the last element of the queue, is added to the Queue class. What is its worst case running time? Give a brief justification.

Deleting the last element requires returning queue[rear] and decrementing rear, being careful at the wraparound point in the queue array. This is a constant time operation, independent of the size of the queue. The key code steps are

```
    Object ret= queue[rear];
    if (rear>0) rear--; else rear= capacity-1;
```

This is O(1).

4. Algorithm design (30 points)

a. Consider the greedy job scheduling algorithm we covered in lecture 10. Modify it to solve the problem when, instead of just one machine, there are N identical machines. Briefly describe your algorithm in words, pseudocode or approximate Java syntax, as you prefer.

One way to model this is to assume we still have just one machine but it works N times faster. We can cut each time interval i into N intervals, within which a job can be done, and use the original algorithm. Any job done in any of the N mini-intervals within an original unit-length interval is done before its deadline. We can arbitrarily say that the job in the first mini-interval is on machine 0, the next is on machine 1, and so on.

The pseudocode multiplies the job deadlines instead of cutting the machine interval; the effect is the same:

```
1. Take each job and multiply its deadline by N, the number of machines
2. Run the original job scheduling algorithm
3. Take all jobs done in times 1 through N and output them as being complete in time 1; take all jobs done in time slots N+1 through 2N and report them as being complete it time 2; etc.
```

b. Does your modified algorithm give a true optimal solution? Why or why not?

The algorithm is the same; only the input is manipulated, so it gives the same solution, and thus the optimal solution.

Queue class code:

```
public class Queue {
    private Object[] queue;
    private int front;
    private int rear;
    private int capacity;
    private int size = 0;

    public Queue(int cap) {
        capacity = cap;
        front = 0;
        rear = capacity - 1;
        queue= new Object[capacity];
    }

    public void add(Object o) {
        if ( size == capacity )
            grow();
        rear = ( rear + 1 ) % capacity;
        queue[ rear ] = o;
        size++;
    }

    public Object remove() throws NoSuchElementException {
        if ( isEmpty() )
            throw new NoSuchElementException();
        else {
            Object ret = queue[ front ];
            front = (front + 1) % capacity;
            size--;
            return ret;
        }
    }

    public boolean isEmpty() {
        return ( size == 0 );
    }

    public void clear() {
        size = 0;
        front = 0;
        rear = capacity - 1;
    }
    // grow() method not shown
}
```