CENG213 Data Structures
Fall 2008
Homework 1
Due: 26/10/2008 23:59

General Rules:

- You have a total of 7 days for late submission to submit your programming assignments. You can spend this credit for any of the assignments or distribute it for all assignments. Make sure that your total late submissions do not exceed the limit. If you have more than 7 days of late submission throughout the term, there will be a penalty of 10% for each additional day. There is no teaming up. The homework has to be done individually. Be careful when discussing the homework. Sharing too much information can be accepted as cheating if there are too many similarities between submitted works.
- In case of cheating, all involved will get zero. If you are involved in cheating all your submissions will be marked zero automatically.
- Your codes should be written in C++, and will be controlled using g++ on “inek“ machines. Be sure that your homework runs on inek machines.
- Proper use of indentation is encouraged. You can use auto-indentations of the editors, homeworks without indentation will be penalized by 5%
- Be careful about input/output specifications (do not print any unnecessary characters, white spaces.)
- Submit your homework electronically through https://cow.ceng.metu.edu.tr

Homework: Online Event Ticketing System

You will implement the backend layer of an online event ticketing system. You will be given the exact definitions of the classes that you'll implement. You are not allowed to change the header file or given class definitions. You will submit only implementations of these given classes, you will not submit a driver program.

Problem Definition

- Problem implementation consists of Events, Places, Tickets and a Database that manages these objects
- Every Place has a unique name, number of seats, and other properties like address and type
- Every Event has a unique event id (such as TEOMAN-A83), a date, and a place name. Dates will be given and stored as strings in the form YYYYMMDD (20081022)
- Events also contain additional information on event's name, price per seat, and list of performers on the stage
- Every Ticket has event id and owner's name. A ticket can span multiple seats. Owner's name is unique among other tickets in the same event. A ticket can be a reservation
- Tickets (both reservation and sold) can be cancelled. In this case the ticket will be removed without a trace.
- A ticket is first reserved (online). When the customer goes to a sales-point and gets the actual ticket, ticket data is updated as sold.
- Database manages and stores Place, Event, Ticket objects
- Database works in terms of Place, Event and Ticket objects for its inputs and outputs

Specifications

- Using the Database class, one will be able to:
  - add places
  - add events
  - add reservation tickets
  - sell an already-reserved ticket (by providing event id and customer name)

        - You will cancel the reservation ticket and create a new ticket in this homework
    - cancel a ticket (by providing event id and customer name)
    - list all Places
    - list all Events
    - list an Event's tickets (by providing event id)
    - list all Events taking place in a single Place (by providing place name)
    - learn the number of available seats left for an event (number of seats that are not reserved or sold yet)
- Using an Event object, one will be able to:
    - add performers to the Event
        - this will be done only BEFORE adding the Event to Database
- Using a Place object, one will be able to:
    - update its address
        - this will be done only BEFORE adding the Place to Database
- Only Database class is supposed to throw exceptions. It will throw the following exceptions wherever applicable:
    - ERR_PLACE_EXISTS
    - ERR_PLACE_NOT_EXISTS
    - ERR_EVENT_EXISTS
    - ERR_EVENT_NOT_EXISTS
    - ERR_PLACE_OCCUPIED
    - ERR_NOT_ENOUGH_SEATS
    - ERR_TICKET_CUSTOMER_EXISTS
    - ERR_TICKET_CUSTOMER_NOT_EXISTS
    - These errors are defined as an enumeration: SystemError
- One will be able to output Event, Place and Ticket objects using overloaded operator "<<" Header file "hw1.h" is provided to you. You will only submit Event.cpp Ticket.cpp Place.cpp and Database.cpp which will include implementations of the methods written in hw1.h file. Make sure that you don't submit your main() method that you use for testing
- You can use a Makefile to make your compilation process easier

Input/output Specs:

- For input, a Database object will be created and tested with direct calls to its methods.
- Output of `cout << SamplePlace`:

  <placeName>
  <placeType (number)>
  <maxSeats>
  <address>

```
METU STADIUM
4
3000
ODTU, Cankaya, Ankara
```

- Output of `cout << SampleEvent`:

  <eventId>
  <eventName>
  <date>
  <price>
  <placeName>
  <numberOfPerformers>
  <performer 1>

<performer 2>
<performer 3>
<performer ...>
<performer n>

```
TEOMAN-METU08
Teoman Odtu Konseri
20081022
25
METU STADIUM
1
Teoman
```

- Output of cout << SampleTicket:
  <eventId>
  <customerName>
  <numberOfSeats>
  <isReservation>

```
TEOMAN-METU08
Ali Veli
2
RESERVATION (or SOLD)
```

- All lines in above examples end with newlines, INCLUDING last lines
- Output of `list*` calls will output the objects in question one by one
- Make sure you add a newline at the end of all lines output by the above rules. When multiple copies of the same object are printed, they should all start on new lines, with no empty lines between them.

Follow the newsgroup for possible changes, or your questions.

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

/* Event */
class Event
{
        private:
                string  eventId;
                string placeName;
                string eventName;
                vector<string> performers;
                string date;
                int price;

        public:
```

```
                Event(string id, string placeName, string eventName, string date,
int price);
                ~Event();
                void addPerformer(string performer);

                string getEventId();
                string getPlaceName();
                string getDate();

                friend ostream& operator<<(ostream& out, const Event &event);
};

ostream& operator<<(ostream &out, const Event &event);


/* Place */
class Place
{
        public:
                enum Type {
                        PLACE_CONCERT_HALL = 1,
                        PLACE_THEATRE = 2,
                        PLACE_PUB = 3,
                        PLACE_STADIUM = 4
                };

        private:
                string placeName;
                int maxSeats;
                string address;
                Place::Type type;

        public:
                Place(string name, Place::Type type, int seats, string address);
                ~Place();
                void setAddress(string newAddress);

                int getMaxSeats();
                string getPlaceName();

                friend ostream& operator<<(ostream& out, const Place &place);
};

ostream& operator<<(ostream& out, const Place &place);


/* Ticket */
class Ticket
{
        private:
                int numSeats;
                string ownerName;
                string eventId;
                bool isReservation;

        public:
                Ticket(string eventId, string ownerName, int numSeats, bool
isReservation);
                ~Ticket();
```

```
                int getNumberOfSeats();
                string getEventId();
                string getOwnerName();
                bool getIsReservation();

                friend ostream& operator<<(ostream& out, const Ticket &ticket);
};

ostream& operator<<(ostream& out, const Ticket &ticket);


/* Database */

/*
   only Database class will throw exceptions

   ERR_PLACE_EXISTS
   ERR_PLACE_NOT_EXISTS
   ERR_EVENT_EXISTS
   ERR_EVENT_NOT_EXISTS
   ERR_PLACE_OCCUPIED
   ERR_NOT_ENOUGH_SEATS
   ERR_TICKET_CUSTOMER_EXISTS
   ERR_TICKET_CUSTOMER_NOT_EXISTS

   will be thrown in all methods whereever applicable.

*/
enum SystemError
{
        ERR_PLACE_EXISTS = 0,
        ERR_PLACE_NOT_EXISTS = 1,
        ERR_EVENT_EXISTS = 2,
        ERR_EVENT_NOT_EXISTS = 3,
        ERR_PLACE_OCCUPIED = 4,
        ERR_NOT_ENOUGH_SEATS = 5,
        ERR_TICKET_CUSTOMER_EXISTS = 6,
        ERR_TICKET_CUSTOMER_NOT_EXISTS = 7,
};

class Database
{
        private:
                vector<Place> places;
                vector<Event> events;
                vector<Ticket> tickets;

        public:
                Database();
                ~Database();

                void addPlace(Place place);
                void addEvent(Event event);

                void reserveTicket(string eventId, string personName, int
numSeats);
                void cancelTicket(string eventId, string personName);
```

```
                // hint: cancel reservation and add ticket again
                void sellTicket(string eventId, string personName);

                Place getPlace(string placeName);
                Event getEvent(string eventId);
                vector<Ticket> getEventTickets(string eventId);
                vector<Event> getEventsByPlace(string placeName);

                int getNumberOfAvailableSeats(string eventId);

                void listPlaces();
                void listEvents();
                void listEventTickets(string eventId);
                void listEventsByPlace(string placeName);
};
```