

FMI Ticket System

Generated by Doxygen 1.9.3

Chapter 1

FMI Tickets System

Author

Kamen Mladenov; FMI Computer Science, Year 1, Group 6; Project 1, Topic 4

Date

12 May 2022

Copyright

GNU General Public License v3.0

Source Code https://github.com/Syndamia/FMI-OOP-P1_TicketSystem

1.1 About

The "FMI Ticket System" is a small project, designed for managing the seats for different events. You can create or cancel an event in any available hall, reserve or straight out buy tickets for any given event and create some reports.

1.2 Structure overview

The project is roughly divided into 4 main components: User Interface, Services, Models and Generic [fig1](#).

- The User Interface is the messenger between a user and the underlying application. All input and output is handled here.
- Services contain the business logic of the whole application. All requests to do anything like buying a ticket or creating an event must be sent to here.
- Models are the general classes that are used in the previous two layers.
- Generic is a place for all code that is can be used independently from the project. Stuff like [Date](#), [String](#) or [List](#).

This offers a fairly modular and flexible architecture, which should make big changes or additions to the code easier and more pain-free.

1.3 Building the project

Instructions are all for `gcc`. All commands should be executed while in the `src` folder.

1.3.1 Linux/BSD/MacOS

```
g++ *.cpp */*.cpp */*/*.cpp -o TicketSystem.out && ./TicketSystem.out
```

1.3.2 Windows (PowerShell)

```
g++ (Get-ChildItem -Recurse *.cpp) -o TicketSystem.exe && ./TicketSystem.exe
```

1.4 Running the tests

Instructions are all for `gcc`. All commands should be executed while in the `tests` folder.

1.4.1 Linux/BSD/MacOS

```
g++ Tests_ALL.cpp ../src/*.cpp ../src/*/*.cpp -o Tests_ALL.out && ./Tests_ALL.out
```

1.5 Acknowledgements

Wikipedia Leap year calculation for [Date](#)

https://en.wikipedia.org/wiki/Leap_year#Gregorian_calendar

StackOverflow Way to clear the console in [Toolbox.hpp](#)

<https://stackoverflow.com/a/52895729/12036073>

1.6 Figures

Figure 1: Overview of the "FMI Ticket System" design

Figure 1.1 Not available in LaTeX!

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Command	??
Date	??
Event	??
EventService	??
Hall	??
HallService	??
List< T >	??
OrderedList< T >	??
List< Command >	??
List< Event >	??
OrderedList< Event >	??
List< Hall >	??
OrderedList< Hall >	??
List< Reservation >	??
OrderedList< Reservation >	??
List< Ticket >	??
OrderedList< Ticket >	??
Menu	??
Reservation	??
String	??
Ticket	??

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Command	Stores a 256 character name and a function pointer to be executed when calling run()	??
Date	Stores a day, month and year	??
Event	Stores an event's hall, name, date, bought tickets and reservations	??
EventService	Each EventService stores all events and a pointer to HallService	??
Hall	Each hall contains a number, rows and seats per row	??
HallService	Each HallService stores all halls	??
List< T >	Templated class that stores an array of elements in dynamic memory	??
Menu	Handles navigation between multiple commands	??
OrderedList< T >	Inherits List , but contains it's elements in a sorted manner	??
Reservation	Each Reservations contains a ticket, a password and a note	??
String	??
Ticket	Each Ticket contains a row and a seat number	??

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

src/main.cpp	Entry point for application	??
src/Generic/ConsoleInterface/Command.h	Stores the declaration of class <code>Command</code>	??
src/Generic/ConsoleInterface/Menu.h	Stores the declaration of class <code>Menu</code>	??
src/Generic/ConsoleInterface/Toolbox.hpp	Stores a wide range of functions for simpler/more automated printing	??
src/Generic/Date/Date.h	Stores the declaration of class <code>Date</code>	??
src/Generic/List/List.hpp	Stores declaration and definition of templated class <code>List</code>	??
src/Generic/List/OrderedList.hpp	Stores declaration and definition of templated class <code>OrderedList</code>	??
src/Generic/String/String.h	Stores declaration of class <code>String</code>	??
src/Models/Event.h	Stores the declaration of class <code>Event</code>	??
src/Models/Hall.h	Stores the declaration of class <code>Hall</code>	??
src/Models/Reservation.h	Stores the declaration of class <code>Reservation</code>	??
src/Models/Ticket.h	Stores the declaration of class <code>Ticket</code>	??
src/Services/EventService.h	Stores the declaration of class <code>EventService</code>	??
src/Services/HallService.h	Stores the declaration of class <code>HallService</code>	??
src/Services/StatusCode.h	Stores declaration of enum <code>StatusCode</code>	??
src/UserInterface/FMITicketSystemConsoleUI.h	Stores declaration of <code>runUI</code> function	??

Chapter 5

Class Documentation

5.1 Command Class Reference

Stores a 256 character name and a function pointer to be executed when calling [run\(\)](#)

```
#include <Command.h>
```

Public Member Functions

- **Command ()**
Leaves name empty and exec function pointer to nullptr.
- [Command](#) (const char *nameInMenu, void(*exec)())
Copies contents of nameInMenu and stores exec.
- void [run](#) () const
Executes the stored function pointer.
- const char * **get_nameInMenu** () const

5.1.1 Detailed Description

Stores a 256 character name and a function pointer to be executed when calling [run\(\)](#)

[Menu](#) uses this class for a more generic implementation of it's navigate function.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Command()

```
Command::Command (
    const char * nameInMenu,
    void(*) () exec )
```

Copies contents of nameInMenu and stores exec.

Parameters

<i>nameInMenu</i>	C-style string, class stores at most 255 characters (last character is always terminating zero)
<i>exec</i>	Function pointer that will be executed when run() is called. Can be nullptr.

5.1.3 Member Function Documentation

5.1.3.1 run()

```
void Command::run ( ) const
```

Executes the stored function pointer.

Executes the stored function pointer, when it's not nullptr. Otherwise does nothing.

The documentation for this class was generated from the following files:

- [src/Generic/ConsoleInterface/Command.h](#)
- [src/Generic/ConsoleInterface/Command.cpp](#)

5.2 Date Class Reference

Stores a day, month and year.

```
#include <Date.h>
```

Public Member Functions

- **Date ()**
Sets date as 01 May 2022.
- **Date** (const char *str)
Parses string.
- **Date** (unsigned short day, unsigned short month, unsigned short year)
Sets day, month and year.
- unsigned short **get_day** () const
Returns the day.
- bool **set_day** (unsigned short day)
Sets the day.
- unsigned short **get_month** () const
Returns the month.
- bool **set_month** (unsigned short month)
Sets the month.
- unsigned short **get_year** () const
Returns the year.

- bool `set_year` (unsigned short year)
Sets the year.
- `String createString ()` const
Returns the date as a `String` object.
- void `read` (std::istream &istr)
Reads date from stream.
- void `write` (std::ostream &ostr)
Writes date to stream.
- int `compare` (const `Date` &other) const
Compares two dates.

5.2.1 Detailed Description

Stores a day, month and year.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 `Date()` [1/2]

```
Date::Date (  
    const char * str )
```

Parses string.

Parameters

<code>str</code>	C-style string
------------------	----------------

Values are initialized with zeroes. Then it parses the string:

Skips spaces, saves day with `atoi`, skips two characters, skips spaces, saves month with `atoi`, skips two characters, skips spaces, saves year with `atoi`.

Values are validated with setters. You should always have spaces between day, month and year and no non-numeric characters between them.

Remarks

If there are non-numeric characters, as per `atoi()` spec, the appropriate property is zero.

5.2.2.2 Date() [2/2]

```
Date::Date (
    unsigned short day,
    unsigned short month,
    unsigned short year )
```

Sets day, month and year.

Initializes values with zeroes and then uses setters to save the function arguments.

5.2.3 Member Function Documentation

5.2.3.1 compare()

```
int Date::compare (
    const Date & other ) const
```

Compares two dates.

Returns

-1 if this < other, 1 if this > other, 0 if this == other

5.2.3.2 createString()

```
String Date::createString ( ) const
```

Returns the date as a [String](#) object.

Returns

[String](#) object in the format "day.month.year"

Note

Currently there isn't any way to change the formatting

5.2.3.3 read()

```
void Date::read (
    std::istream & istr )
```

Reads date from stream.

Parameters

<i>istr</i>	An input stream
-------------	-----------------

Directly reads bytes from stream (calls [read\(\)](#) function).

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ifstream

5.2.3.4 set_day()

```
bool Date::set_day (
    unsigned short newDay )
```

Sets the day.

Returns

Whether newDay was saved or not

If newDay is equal to 0, or is more than the appropriate days in the month, the day isn't set.

5.2.3.5 set_month()

```
bool Date::set_month (
    unsigned short newMonth )
```

Sets the month.

Returns

Whether newMonth was saved or not

If newMonth is equal to 0 or is more than 12, the month isn't set.

5.2.3.6 set_year()

```
bool Date::set_year (
    unsigned short newYear )
```

Sets the year.

Returns

Whether year was saved or not

Year isn't set when newYear is zero

5.2.3.7 write()

```
void Date::write (
    std::ostream & ostr )
```

Writes date to stream.

Parameters

<i>ostr</i>	An output stream
-------------	------------------

Directly writes bytes to stream (calls [write\(\)](#) function).

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ofstream

The documentation for this class was generated from the following files:

- src/Generic/Date/[Date.h](#)
- src/Generic/Date/[Date.cpp](#)

5.3 Event Class Reference

Stores an event's hall, name, date, bought tickets and reservations.

```
#include <Event.h>
```

Public Member Functions

- **Event ()**
Calls default constructors on data.
- **Event (const [Hall](#) &hall, [String](#) name, [Date](#) date)**
Copies given hall and sets name and date.
- const [Hall](#) & **get_hall ()** const
Returns constant reference [Hall](#) in which event will be held.
- const [String](#) & **get_name ()** const
Returns constant reference to name of event.
- const [Date](#) & **get_date ()** const
Returns constant reference to date on which event will be held.
- [OrderedList](#)< [Ticket](#) > & **get_tickets ()**
Returns reference to bought tickets.
- const [OrderedList](#)< [Ticket](#) > & **get_tickets ()** const
Returns constant reference to bought tickets.
- [OrderedList](#)< [Reservation](#) > & **get_reservations ()**
Returns reference to reservations.
- const [OrderedList](#)< [Reservation](#) > & **get_reservations ()** const
Returns constant reference to reservations.
- void **read** (std::istream &istr)
Reads [Event](#) from stream.
- void **write** (std::ostream &ostr)
Writes [Event](#) to stream.
- int **compare** (const [Event](#) &other)
Compares two events.

5.3.1 Detailed Description

Stores an event's hall, name, date, bought tickets and reservations.

5.3.2 Member Function Documentation

5.3.2.1 compare()

```
int Event::compare (
    const Event & other )
```

Compares two events.

Returns

compare of halls, or if they are equal, compare of dates

5.3.2.2 read()

```
void Event::read (
    std::istream & istr )
```

Reads [Event](#) from stream.

Parameters

<i>istr</i>	An input stream
-------------	-----------------

Directly reads bytes from stream (calls [read\(\)](#) function). Stored types must have defined a [read\(\)](#) function.

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ifstream

5.3.2.3 write()

```
void Event::write (
    std::ostream & ostr )
```

Writes [Event](#) to stream.

Parameters

<code>ostr</code>	An output stream
-------------------	------------------

Directly writes bytes to stream (calls [write\(\)](#) function). Stored types must have defined a [read\(\)](#) function.

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ofstream

The documentation for this class was generated from the following files:

- src/Models/[Event.h](#)
- src/Models/Event.cpp

5.4 EventService Class Reference

Each [EventService](#) stores all events and a pointer to [HallService](#).

```
#include <EventService.h>
```

Public Member Functions

- **EventService** (const [HallService](#) *hs)
Sets [HallService](#) pointer and initializes events [List](#).
- **EventService** (const [HallService](#) *hs, const [Event](#) *events, unsigned eventCount)
Sets [HallService](#) pointer and copies values from events into events [List](#).
- **StatusCode createEvent** (int hallNumber, const [String](#) &name, const [Date](#) &date)
Creates an [Event](#), with the given hallNumber, name and date.
- **StatusCode cancelEvent** (const char *name, const [Date](#) &date)
Cancels an [Event](#) with the given name and date.
- **StatusCode reserveTicket** (const char *name, const [Date](#) &date, const char *password, const char *note, const [Ticket](#) &ticket)
Creates a [Reservation](#) for an [Event](#) by name and date.
- **StatusCode cancelTicketReservation** (const char *name, const [Date](#) &date, const [Ticket](#) &ticket)
Cancels a [Reservation](#) for a ticket.
- **StatusCode ticketsReserved** (const char *name, const [Date](#) &date, const [Ticket](#) &ticket)
Whether a ticket is a reserved or not.
- **StatusCode buyTicket** (const char *name, const [Date](#) &date, const [Ticket](#) &ticket, const char *password, [String](#) *reservationNoteOutput)
Buys ticket, assuming it is reserved and stores the reservation note.
- **StatusCode buyTicket** (const char *name, const [Date](#) &date, const [Ticket](#) &ticket)
Buys ticket, assuming it isn't reserved.
- **String createSeatingString** (const char *name, const [Date](#) &date, unsigned *seatsPerRow)

Creates a special [String](#) for status of all seats in given [Event](#) (by name and date) and returns seatsPerRow in given pointer.

- [List](#)< [Event](#) > [queryMostWatched](#) (unsigned topN)
- [List](#)< [Event](#) > [queryInsufficientlyVisited](#) ()
- [StatusCode](#) [reportReservations](#) (const char *name, const [Date](#) &date)

Creates a report that lists all Events (with given name and date) and their reserved tickets.

- [StatusCode](#) [reportBoughtTickets](#) (int hallNumber, const [Date](#) &start, const [Date](#) &end, bool all=false)

Creates a report that Lists all Events (with given name and date) and number of bought tickets.

- [StatusCode](#) [load](#) ()

Loads Events from a database file.

- [StatusCode](#) [save](#) ()

Saves Events to a database file.

5.4.1 Detailed Description

Each [EventService](#) stores all events and a pointer to [HallService](#).

5.4.2 Member Function Documentation

5.4.2.1 [buyTicket\(\)](#) [1/2]

```
StatusCode EventService::buyTicket (
    const char * name,
    const Date & date,
    const Ticket & ticket )
```

Buys ticket, assuming it isn't reserved.

Calls [buyTicket](#) with [Reservation](#), but sets password and noteOutput to nullptr.

Returns

[E_EventDoesNotExist](#) if No [Event](#) with the given name and date exists

[E_TicketAlreadyBought](#) if A ticket for the given row and seat has already been bought

[E_TicketAlreadyReserved](#) if The seat and row are reserved, but given password is nullptr

[E_WrongReservationPassword](#) if The seat and row are reserved, but given password is incorrect

Success otherwise

5.4.2.2 buyTicket() [2/2]

```
StatusCode EventService::buyTicket (
    const char * name,
    const Date & date,
    const Ticket & ticket,
    const char * password,
    String * reservationNoteOutput )
```

Buys ticket, assuming it is reserved and stores the reservation note.

Returns

E_EventDoesNotExist if No [Event](#) with the given name and date exists
E_TicketAlreadyBought if A ticket for the given row and seat has already been bought
E_TicketAlreadyReserved if The seat and row are reserved, but given password is nullptr
E_WrongReservationPassword if The seat and row are reserved, but given password is incorrect
Success **otherwise**

5.4.2.3 cancelEvent()

```
StatusCode EventService::cancelEvent (
    const char * name,
    const Date & date )
```

Cancels an [Event](#) with the given name and date.

Returns

E_EventDoesNotExist if No [Event](#) with the given name and date exists
Success **otherwise**

5.4.2.4 cancelTicketReservation()

```
StatusCode EventService::cancelTicketReservation (
    const char * name,
    const Date & date,
    const Ticket & ticket )
```

Cancels a [Reservation](#) for a ticket.

Canceling a reservation doesn't require password and doesn't return the [Reservation](#)'s note

Returns

E_EventDoesNotExist if No [Event](#) with the given name and date exists
W_TicketHadNotBeenReserved [Ticket](#) hadn't been reserved in the first place
Success **otherwise**

5.4.2.5 createEvent()

```
StatusCode EventService::createEvent (
    int hallNumber,
    const String & name,
    const Date & date )
```

Creates an [Event](#), with the given hallNumber, name and date.

Uses [HallService](#) pointer to get [Hall](#) from hallNumber

Returns

E_HallDoesntExist if hallNumber doesn't correspond to any [Hall](#), saved in [HallService](#)

E_EventWillOverlap if [Event](#)'s date overlaps with another event in the same [Hall](#)

Success **otherwise**

5.4.2.6 createSeatingString()

```
String EventService::createSeatingString (
    const char * name,
    const Date & date,
    unsigned * seatsPerRow )
```

Creates a special [String](#) for status of all seats in given [Event](#) (by name and date) and returns seatsPerRow in given pointer.

Each seat in the [Hall](#) of the [Event](#) is represented by a single character, where 'R' means the seat is reserved, 'B' means the seat is bought and ' ' if it's neither.

Returns

Empty [String](#) if No [Event](#) with the given name and date exists

Proper [String](#) **otherwise**

5.4.2.7 load()

```
StatusCode EventService::load ( )
```

Loads Events from a database file.

Looks for file named "eventsDatabase.fmits", which is just a binary file, and then reads it with the "read" function in events [List](#).

Returns

E_FileCouldNotBeOpened if The files couldn't be opened/Doesn't exist

Success **otherwise**

5.4.2.8 queryInsufficientlyVisited()

```
List< Event > EventService::queryInsufficientlyVisited ( )
```

Returns

A [List](#) of events which have attracted sales of less than 10% of the [Hall's](#) seat

5.4.2.9 queryMostWatched()

```
List< Event > EventService::queryMostWatched (
    unsigned topN )
```

Returns

A [List](#) of topN most watched Events

Events are sorted from index 0 to last index decreasingly by number of watchers. Watchers count for an [Event](#) is the amount of bought tickets.

If there are less events than topN, all events are returned.

5.4.2.10 reportBoughtTickets()

```
StatusCode EventService::reportBoughtTickets (
    int hallNumber,
    const Date & start,
    const Date & end,
    bool all = false )
```

Creates a report that Lists all Events (with given name and date) and number of bought tickets.

Parameters

<i>hallNumber</i>	Number of the Event's Hall
<i>start</i>	Starting Date from which to take Events
<i>end</i>	Ending Date until which to take Evnets (including)

Report is saved in a file named "boughtTickets.txt" in the current directory.

All Events are inserted into the file with the << operator and a setting of 0b01001.

Returns

E_FileCouldNotBeOpened if Report file couldn't be created

Success **otherwise**

5.4.2.11 reportReservations()

```
StatusCode EventService::reportReservations (
    const char * name,
    const Date & date )
```

Creates a report that lists all Events (with given name and date) and their reserved tickets.

Parameters

<i>name</i>	C-style string, if it is equal to "ALL", reservations from all Events on the given date will be reported
<i>date</i>	If day, month and year are equal to 0, reservations from all dates will be reported

Report is saved in a file in the current directory, named "report-NAME-DATE.txt", where NAME is the given name, and DATE is the given date or "ALL" if the date is 0/0/0

All Events are inserted into the file with the << operator and a setting of 0b00101.

Returns

E_FileCouldNotBeOpened if Report file couldn't be created

Success otherwise

5.4.2.12 reserveTicket()

```
StatusCode EventService::reserveTicket (
    const char * name,
    const Date & date,
    const char * password,
    const char * note,
    const Ticket & ticket )
```

Creates a [Reservation](#) for an [Event](#) by name and date.

Returns

E_EventDoesNotExist if No [Event](#) with the given name and date exists

E_TicketAlreadyBought if A ticket for the given row and seat has already been bought

E_TicketAlreadyReserved if A reservation has already been placed on the given row and seat

Success otherwise

5.4.2.13 save()

```
StatusCode EventService::save ( )
```

Saves Events to a database file.

Looks for file named "eventsDatabase.fmits", which is just a binary file, and then writes to it with the "write" function in events [List](#).

Returns

E_FileCouldNotBeOpened if The files couldn't be opened
Success **otherwise**

5.4.2.14 ticketIsReserved()

```
StatusCode EventService::ticketIsReserved (
    const char * name,
    const Date & date,
    const Ticket & ticket )
```

Whether a ticket is a reserved or not.

Returns

E_EventDoesNotExist if No [Event](#) with the given name and date exists
W_TicketHadNotBeenReserved [Ticket](#) hadn't been reserved in the first place
Success **otherwise**

The documentation for this class was generated from the following files:

- src/Services/[EventService.h](#)
- src/Services/EventService.cpp

5.5 Hall Class Reference

Each hall contains a number, rows and seats per row.

```
#include <Hall.h>
```


Public Member Functions

- **Hall** ()
Sets values to 0.
- **Hall** (int number)
Sets values, except number, to 0.
- **Hall** (int number, unsigned rows, unsigned seatsPerRow)
Sets values.
- int **get_number** () const
Returns a Hall's number.
- unsigned **get_rows** () const
Returns a Hall's row count.
- unsigned **get_seatsPerRow** () const
Return a Hall's seats per row count.
- void **read** (std::istream &istr)
Reads Hall from stream.
- void **write** (std::ostream &ostr) const
Writes Hall to stream.
- int **compare** (const Hall &other) const
Compare two Halls.

5.5.1 Detailed Description

Each hall contains a number, rows and seats per row.

5.5.2 Member Function Documentation

5.5.2.1 compare()

```
int Hall::compare (  
    const Hall & other ) const
```

Compare two Halls.

Returns

-1 if number < other.number, 1 if number > other.number, 0 if number == other.number

5.5.2.2 read()

```
void Hall::read (  
    std::istream & istr )
```

Reads Hall from stream.

Parameters

<i>istr</i>	An input stream
-------------	-----------------

Directly reads bytes from stream (calls [read\(\)](#) function).

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ifstream

5.5.2.3 write()

```
void Hall::write (
    std::ostream & ostr ) const
```

Writes [Hall](#) to steam.

Parameters

<i>ostr</i>	An output stream
-------------	------------------

Directly writes bytes to stream (calls [write\(\)](#) function).

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ofstream

The documentation for this class was generated from the following files:

- src/Models/[Hall.h](#)
- src/Models/Hall.cpp

5.6 HallService Class Reference

Each [HallService](#) stores all halls.

```
#include <HallService.h>
```

Public Member Functions

- **HallService** (const [Hall](#) *halls, unsigned hallCount)
Copies halls from parameters.
- const [OrderedList](#)< [Hall](#) > & **get_halls** () const
Returns a constant reference to halls.
- [StatusCode](#) **load** ()
Loads Halls to a database file.
- [StatusCode](#) **save** ()
Saves Halls to a database file.

5.6.1 Detailed Description

Each [HallService](#) stores all halls.

5.6.2 Member Function Documentation

5.6.2.1 load()

```
StatusCode HallService::load ( )
```

Loads Halls to a database file.

Looks for file named "hallsDatabase.fmits", which is just a binary file, and then reads it with the "read" function in events [List](#).

Returns

[E_FileCouldNotBeOpened](#) **if** The files couldn't be opened/Doesn't exist
Success **otherwise**

5.6.2.2 save()

```
StatusCode HallService::save ( )
```

Saves Halls to a database file.

Looks for file named "hallsDatabase.fmits", which is just a binary file, and then writes to it with the "write" function in events [List](#).

Returns

[E_FileCouldNotBeOpened](#) **if** The files couldn't be opened
Success **otherwise**

The documentation for this class was generated from the following files:

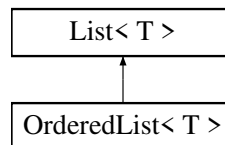
- src/Services/[HallService.h](#)
- src/Services/HallService.cpp

5.7 List< T > Class Template Reference

Templated class that stores an array of elements in dynamic memory.

```
#include <List.hpp>
```

Inheritance diagram for List< T >:



Public Member Functions

- **List** (const T *elements, unsigned elementsCount)
Copies all elements in given array.
- void **add** (const T &element)
Adds an element.
- void **insertAt** (const T &element, unsigned index)
Inserts element at given index.
- bool **removeAt** (unsigned index)
Removes element at index.
- unsigned **findIndex** (const T &element) const
Finds the index of element.
- bool **contain** (const T &element) const
- T & **operator[]** (unsigned index)
Returns reference to element at index.
- const T & **operator[]** (unsigned index) const
Returns constant reference to element at index.
- List< T > & **operator+=** (const List< T > other)
Appends elements from other list.
- std::istream & **read** (std::istream &istr)
Reads from stream.
- std::ostream & **write** (std::ostream &ostr) const
Writes to stream.
- unsigned **get_length** () const
Returns the length.
- unsigned **get_count** () const
Returns the count.
- List & **operator=** (const List &other)
- **List** (const List &other)
- **List** (List &&other)
- List & **operator=** (List &&other)

Protected Member Functions

- void **resize** ()
- void **free** ()
- void **copyFrom** (const List &other)

Protected Attributes

- T * **elements**
- unsigned **length**
- unsigned **count**

5.7.1 Detailed Description

```
template<typename T>
class List< T >
```

Templated class that stores an array of elements in dynamic memory.

Warning

[findIndex\(\)](#), [contain\(\)](#), [read\(\)](#), [write\(\)](#), [operator<<\(\)](#) and [operator>>\(\)](#) require the type to have defined a couple of functions.

See also

[findIndex\(\)](#)
[contain\(\)](#)
[read\(\)](#)
[write\(\)](#)
[operator<<\(\)](#)
[operator>>\(\)](#)

5.7.2 Member Function Documentation

5.7.2.1 add()

```
template<typename T >
void List< T >::add (
    const T & element )
```

Adds an element.

Resizes internal array if there is no space for additional elements.

5.7.2.2 contain()

```
template<typename T >
bool List< T >::contain (
    const T & element ) const
```

Returns

Whether the element is contained in the current [List](#)

Warning

Function depends on [findIndex\(\)](#), which means the same "compare" function must be defined in the type

See also

[findIndex\(\)](#)

5.7.2.3 findIndex()

```
template<typename T >
unsigned List< T >::findIndex (
    const T & element ) const
```

Finds the index of element.

Returns

Index of element. If element isn't found, returns the count of element.

Warning

The function depends on the type having a function "compare" defined, which takes two elements and returns a number <0 if elem1 < elem2, >0 if elem1 > elem2, 0 if elem1 == elem2

Note

Searching is done linearly

5.7.2.4 get_count()

```
template<typename T >
unsigned List< T >::get_count
```

Returns the count.

Count is the amount of elements that are stored.

5.7.2.5 get_length()

```
template<typename T >
unsigned List< T >::get_length
```

Returns the length.

Length is the size of the underlying array (allocated memory).

5.7.2.6 insertAt()

```
template<typename T >
void List< T >::insertAt (
    const T & element,
    unsigned index )
```

Inserts element at given index.

If index is after the last element, the element is just added. Otherwise all elements after the index are shifted right and element is put in place.

Resizes internal array if there is no space for additional elements.

5.7.2.7 read()

```
template<typename T >
std::istream & List< T >::read (
    std::istream & istr )
```

Reads from stream.

Parameters

<i>istr</i>	An input stream
-------------	-----------------

Directly reads bytes from stream (calls [read\(\)](#) function). Any stored values are deleted and replaced with those from the stream.

Warning

The function depends on the type having a function "read" defined, which takes an std::istream& and writes it's data to it. Return type doesn't matter.

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ifstream

5.7.2.8 removeAt()

```
template<typename T >
bool List< T >::removeAt (
    unsigned index )
```

Removes element at index.

Returns

Whether element could be removed

If index is after that of the last element, nothing is done and false is returned. Otherwise elements after index are shifted right and count is reduced.

5.7.2.9 write()

```
template<typename T >
std::ostream & List< T >::write (
    std::ostream & ostr ) const
```

Writes to stream.

Parameters

<i>ostr</i>	An output stream
-------------	------------------

Directly writes bytes to stream (calls [write\(\)](#) function).

Warning

The function depends on the type having a function "write" defined, which takes an std::ostream& and writes it's data to it. Return type doesn't matter.

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ofstream

The documentation for this class was generated from the following file:

- src/Generic/List/[List.hpp](#)

5.8 Menu Class Reference

Handles navigation between multiple commands.

```
#include <Menu.h>
```


Public Member Functions

- **Menu** ()
Sets name as "Menu", leaves all flags to false and creates an empty [Command](#) list.
- **Menu** (const char *title, bool backExistsApp, bool isSubMenu)
Copies title, saves flags and creates an empty [Command](#) list.
- **Menu** (const char *title, bool backExistsApp, bool isSubMenu, const [Command](#) *commands, unsigned commandCount)
Copies title, saves flags and copies commands.
- void **addCommand** (const [Command](#) &command)
Adds a command to the internal [Command](#) list.
- void **registerError** (const char *message) const
Tell [Menu](#) to show an error message next time it prints.
- void **registerWarning** (const char *message) const
Tell [Menu](#) to show a warning message next time it prints.
- void **registerSuccess** (const char *message) const
Tell [Menu](#) to show a success message next time it prints.
- void **navigate** () const
Lists commands and after user input executes an appropriate command run() function.

5.8.1 Detailed Description

Handles navigation between multiple commands.

Shows the title, then below it an Error/Warning/Success message if set, lists all command options as an enumerated list, takes command index and calls the chosen command's run() function.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 Menu() [1/2]

```
Menu::Menu (
    const char * title,
    bool backExistsApp,
    bool isSubMenu )
```

Copies title, saves flags and creates an empty [Command](#) list.

Parameters

<i>title</i>	C-style string, class stores at most 255 characters (last character is always terminating zero)
<i>backExistsApp</i>	Flag whether or not to list the 0-th option as "Exit", rather than "Go Back"
<i>isSubMenu</i>	Flag whether or not to show the title as a subTitle or not

5.8.2.2 Menu() [2/2]

```
Menu::Menu (
    const char * title,
    bool backExistsApp,
    bool isSubMenu,
    const Command * commands,
    unsigned commandCount )
```

Copies title, saves flags and copies commands.

Parameters

<i>title</i>	C-style string, class stores at most 255 characters (last character is always terminating zero)
<i>backExistsApp</i>	Flag whether or not to list the 0-th option as "Exit", rather than "Go Back"
<i>isSubMenu</i>	Flag whether or not to show the title as a subTitle or not
<i>commands</i>	Pointer to an array of Command instances
<i>commandCount</i>	Number of elements that "commands" points to

5.8.3 Member Function Documentation

5.8.3.1 navigate()

```
void Menu::navigate ( ) const
```

Lists commands and after user input executes an appropriate command run() function.

Prints the title, then an Error/Warning/Sucess message (if set), prints all command names as an ordered list (starting from 1), waits for user input to select one of those commands (by list number) and finally executes the appropriate command's run() function.

After the run() function exists, everything is reprinted. The 0 list index is always "Go Back"/"Exit" and it always stops the reprinting loop.

If there are no commands, prints "Menu is empty!". If user input doesn't correspond to any command, registers an error message and reprints.

5.8.3.2 registerError()

```
void Menu::registerError (
    const char * message ) const
```

Tell [Menu](#) to show an error message next time it prints.

Parameters

<i>message</i>	C-style string, saves to pointer for later reuse when showing the message.
----------------	--

Stores the message pointer in a global variable (in the cpp file) and sets it's type as an Error in another global variable (in the cpp file).

The next time the whole [Menu](#) is shown, the message will appear below the title.

5.8.3.3 `registerSuccess()`

```
void Menu::registerSuccess (
    const char * message ) const
```

Tell [Menu](#) to show a success message next time it prints.

Parameters

<code>message</code>	C-style string, saves to pointer for later reuse when showing the message.
----------------------	--

Stores the message pointer in a global variable (in the cpp file) and sets it's type as Success in another global variable (in the cpp file).

The next time the whole [Menu](#) is shown, the message will appear below the title.

5.8.3.4 `registerWarning()`

```
void Menu::registerWarning (
    const char * message ) const
```

Tell [Menu](#) to show a warning message next time it prints.

Parameters

<code>message</code>	C-style string, saves to pointer for later reuse when showing the message.
----------------------	--

Stores the message pointer in a global variable (in the cpp file) and sets it's type as a Warning in another global variable (in the cpp file).

The next time the whole [Menu](#) is shown, the message will appear below the title.

The documentation for this class was generated from the following files:

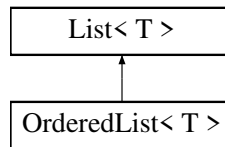
- `src/Generic/ConsoleInterface/Menu.h`
- `src/Generic/ConsoleInterface/Menu.cpp`

5.9 `OrderedList< T >` Class Template Reference

Inherits [List](#), but contains it's elements in a sorted manner.

```
#include <OrderedList.hpp>
```

Inheritance diagram for `OrderedList< T >`:



Public Member Functions

- **OrderedList** (const T *elements, unsigned elementsCount)
- void [add](#) (T element)=delete
- void [insertAt](#) (T element, unsigned index)=delete
- void [insert](#) (T element)

Inserts an element at a sorted position.

Additional Inherited Members

5.9.1 Detailed Description

```
template<typename T>
class OrderedList< T >
```

Inherits [List](#), but contains it's elements in a sorted manner.

Warning

Outside those defined in [List](#), [insert\(\)](#) requires the type to have defined a "compare" of functions.

See also

[insert\(\)](#)

5.9.2 Member Function Documentation

5.9.2.1 add()

```
template<typename T >
void OrderedList< T >::add (
    T element ) [delete]
```

Warning

Function is deleted, use [insert\(\)](#) instead

See also

[insert\(\)](#)

5.9.2.2 insert()

```
template<typename T >
void OrderedList< T >::insert (
    T element )
```

Inserts an element at a sorted position.

Warning

The function depends on the type having a function "compare" defined, which takes two elements and returns a number <0 if elem1 < elem2, >0 if elem1 > elem2, 0 if elem1 == elem2

Note

Insertion is done linearly

5.9.2.3 insertAt()

```
template<typename T >
void OrderedList< T >::insertAt (
    T element,
    unsigned index ) [deleted]
```

Warning

Function is deleted, use [insert\(\)](#) instead

See also

[insert\(\)](#)

The documentation for this class was generated from the following file:

- [src/Generic/List/OrderedList.hpp](#)

5.10 Reservation Class Reference

Each Reservations contains a ticket, a password and a note.

```
#include <Reservation.h>
```

Public Member Functions

- **Reservation** ()
Sets ticket as default value and leaves password and note blank.
- **Reservation** (const [Ticket](#) &ticket)
Sets ticket and leaves password and note blank.
- **Reservation** (const [Ticket](#) &ticket, const char *password, const char *note)
Sets ticket, password and note.
- const [Ticket](#) & **get_ticket** () const
Returns constant reference to ticket.
- const char * **get_password** () const
Returns constant C-style string to password.
- const char * **get_note** () const
Returns constant C-style string to note.
- void **read** (std::istream &istr)
Reads [Reservation](#) from stream.
- void **write** (std::ostream &ostr) const
Writes [Reservation](#) to stream.
- int **compare** (const [Reservation](#) &other) const
Compares two Reservations.

Friends

- std::istream & **operator>>** (std::istream &istr, [Reservation](#) &reservation)
Reads [Reservation](#) from stream with >> operator.
- std::ostream & **operator<<** (std::ostream &ostr, const [Reservation](#) &reservation)
Writes [Reservation](#) to stream with << operator.

5.10.1 Detailed Description

Each Reservations contains a ticket, a password and a note.

5.10.2 Member Function Documentation

5.10.2.1 compare()

```
int Reservation::compare (
    const Reservation & other ) const
```

Compares two Reservations.

Returns

Compares tickets

5.10.2.2 read()

```
void Reservation::read (
    std::istream & istr )
```

Reads [Reservation](#) from stream.

Parameters

<i>istr</i>	An input stream
-------------	-----------------

Directly reads bytes from stream (calls [read\(\)](#) function). Uses ticket's [read\(\)](#) function.

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ifstream

5.10.2.3 write()

```
void Reservation::write (
    std::ostream & ostr ) const
```

Writes [Reservation](#) to stream.

Parameters

<i>ostr</i>	An output stream
-------------	------------------

Directly writes bytes to stream (calls [write\(\)](#) function). Uses ticket's [write\(\)](#) function.

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ofstream

5.10.3 Friends And Related Function Documentation

5.10.3.1 operator<<

```
std::ostream & operator<< (
    std::ostream & ostr,
    const Reservation & reservation ) [friend]
```

Writes [Reservation](#) to stream with << operator.

Uses the stream's << operator to write the ticket only.

Note

Best used with `std::cout` or text `std::ofstream`

5.10.3.2 operator>>

```
std::istream & operator>> (
    std::istream & istr,
    Reservation & reservation ) [friend]
```

Reads [Reservation](#) from stream with `>>` operator.

Uses the stream's `>>` operator to read and parse the ticket. Password and note are left blank.

Note

Best used with `std::cin` or text `std::ifstream`

The documentation for this class was generated from the following files:

- `src/Models/Reservation.h`
- `src/Models/Reservation.cpp`

5.11 String Class Reference**Public Member Functions**

- [String](#) (unsigned length)
The underlying array is allocated with length (+ 1) size.
- [String](#) (const char *str)
Copies str.
- const char * **get_cstr** () const
Returns internal C-style string.
- unsigned **get_length** () const
Returns number of characters in string.
- char & **operator[]** (unsigned index)
Returns a character reference (from the underlying C-style string) at the given index.
- [String](#) & **operator+=** (const char *str)
Appends a C-style string.
- [String](#) & **operator+=** (unsigned number)
Appends a number.
- **String** (const [String](#) &other)
- [String](#) & **operator=** (const [String](#) &other)
- **String** ([String](#) &&other)
- [String](#) & **operator=** ([String](#) &&other)
- void **read** (std::istream &istr)
Reads [String](#) from stream.
- void **write** (std::ostream &ostr) const
Writes [String](#) to stream.
- int **compare** (const [String](#) &other) const
Compares two strings.

Friends

- `std::istream & operator>> (std::istream &istr, String &event)`
Reads [String](#) from stream with >> operator.
- `std::ostream & operator<< (std::ostream &ostr, const String &event)`
Writes [String](#) to stream with << operator.

5.11.1 Constructor & Destructor Documentation

5.11.1.1 [String\(\)](#) [1/2]

```
String::String (  
    unsigned length )
```

The underlying array is allocated with length (+ 1) size.

Allocated with length + 1 size, so there is a terminating zero at the end.

5.11.1.2 [String\(\)](#) [2/2]

```
String::String (  
    const char * str )
```

Copies str.

\params str C-style string

5.11.2 Member Function Documentation

5.11.2.1 [compare\(\)](#)

```
int String::compare (  
    const String & other ) const
```

Compares two strings.

\params other C-style string

Returns

`strcmp` between underlying C-style string and "other"

5.11.2.2 `operator+=()` [1/2]

```
String & String::operator+= (
    const char * str )
```

Appends a C-style string.

\params *str* C-style string

5.11.2.3 `operator+=()` [2/2]

```
String & String::operator+= (
    unsigned number )
```

Appends a number.

Converts the number to a C-style string and then uses += to append it.

5.11.2.4 `read()`

```
void String::read (
    std::istream & istr )
```

Reads [String](#) from stream.

Parameters

<i>istr</i>	An input stream
-------------	-----------------

Directly reads bytes from stream (calls [read\(\)](#) function). First reads the string length, then the underlying C-style string (including terminating zero).

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ifstream

5.11.2.5 `write()`

```
void String::write (
    std::ostream & ostr ) const
```

Writes [String](#) to stream.

Parameters

<i>ostr</i>	An output stream
-------------	------------------

Directly writes bytes to stream (calls [write\(\)](#) function). First writes the string length, then the underlying C-style string (including terminating zero).

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ofstream

5.11.3 Friends And Related Function Documentation

5.11.3.1 operator<<

```
std::ostream & operator<< (  
    std::ostream & ostr,  
    const String & event ) [friend]
```

Writes [String](#) to stream with << operator.

Uses the stream's << operator to write the underlying C-style string

Note

Best used with std::cout or text std::ofstream

5.11.3.2 operator>>

```
std::istream & operator>> (  
    std::istream & istr,  
    String & event ) [friend]
```

Reads [String](#) from stream with >> operator.

Uses the stream's getline function to read the data.

Warning

It takes at most 1024 characters from the stream!

Note

Best used with std::cin or text std::ifstream

The documentation for this class was generated from the following files:

- src/Generic/String/[String.h](#)
- src/Generic/String/String.cpp

5.12 Ticket Class Reference

Each [Ticket](#) contains a row and a seat number.

```
#include <Ticket.h>
```

Public Member Functions

- **[Ticket](#)** ()
Sets row and seat to zeroes.
- **[Ticket](#)** (unsigned row, unsigned seat)
Sets row and seat.
- unsigned **[get_row](#)** () const
Returns row.
- unsigned **[get_seat](#)** () const
Returns seat.
- void **[read](#)** (std::istream &istr)
Reads [Ticket](#) from stram.
- void **[write](#)** (std::ostream &ostr) const
Writes [Ticket](#) to stream.
- int **[compare](#)** (const [Ticket](#) &other) const
Compares two [Tickets](#).

5.12.1 Detailed Description

Each [Ticket](#) contains a row and a seat number.

5.12.2 Member Function Documentation

5.12.2.1 [compare\(\)](#)

```
int Ticket::compare (  
    const Ticket & other ) const
```

Compares two [Tickets](#).

Returns

If rows are equal, -1 if seat < other.seat, 1 if seat > other.seat, 0 if seat == other.seat, else the same, but for rows

5.12.2.2 [read\(\)](#)

```
void Ticket::read (  
    std::istream & istr )
```

Reads [Ticket](#) from stram.

Parameters

<i>istr</i>	An input stream
-------------	-----------------

Directly reads bytes from stream (calls [read\(\)](#) function).

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ifstream

5.12.2.3 write()

```
void Ticket::write (
    std::ostream & ostr ) const
```

Writes [Ticket](#) to stream.

Parameters

<i>ostr</i>	An output stream
-------------	------------------

Directly writes bytes to stream (calls [write\(\)](#) function).

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ofstream

The documentation for this class was generated from the following files:

- src/Models/[Ticket.h](#)
- src/Models/Ticket.cpp

Chapter 6

File Documentation

6.1 src/Generic/ConsoleInterface/Command.h File Reference

Stores the declaration of class [Command](#).

Classes

- class [Command](#)

Stores a 256 character name and a function pointer to be executed when calling [run\(\)](#)

6.1.1 Detailed Description

Stores the declaration of class [Command](#).

6.2 Command.h

[Go to the documentation of this file.](#)

```
1 #ifndef HEADER_CONSOLEINTERFACE_COMMAND
2 #define HEADER_CONSOLEINTERFACE_COMMAND
3
12 class Command {
13     char nameInMenu[256];
14     void (*exec)();
15
16 public:
17     Command();
20     Command(const char* nameInMenu, void (*exec)());
22     void run() const;
23
24     const char* get_nameInMenu() const;
25 };
26
27 #endif
```

6.3 src/Generic/ConsoleInterface/Menu.h File Reference

Stores the declaration of class [Menu](#).

```
#include "Command.h"
#include "../List/List.hpp"
```

Classes

- class [Menu](#)

Handles navigation between multiple commands.

6.3.1 Detailed Description

Stores the declaration of class [Menu](#).

6.4 Menu.h

[Go to the documentation of this file.](#)

```
1 #ifndef HEADER_CONSOLEINTERFACE_MENU
2 #define HEADER_CONSOLEINTERFACE_MENU
3
4 #include "Command.h"
5 #include "../List/List.hpp"
6
7 class Menu {
8     char title[256];
9     List<Command> menuOptions;
10
11     bool backExistsApp;
12     bool isSubMenu;
13
14 public:
15     Menu();
16     Menu(const char* title, bool backExistsApp, bool isSubMenu);
17     Menu(const char* title, bool backExistsApp, bool isSubMenu, const Command* commands, unsigned
18         commandCount);
19
20     void addCommand(const Command& command);
21
22     void registerError(const char* message) const;
23     void registerWarning(const char* message) const;
24     void registerSuccess(const char* message) const;
25
26     void navigate() const;
27 };
28
29 #endif
```

6.5 src/Generic/ConsoleInterface/Toolbox.hpp File Reference

Stores a wide range of functions for simpler/more automated printing.

```
#include <iostream>
```

Macros

- #define **MAX_LINE_WIDTH** 1024

Functions

- void `clear` ()
Clears the console screen.
- void `titleBox` (const char *title)
Prints a title box.
- void `subTitleBox` (const char *title)
Prints a title sub box.
- void `successBox` (const char *message)
Prints a success box.
- void `successSubBox` (const char *message)
Prints a success box as a sub box.
- void `warningBox` (const char *message)
Prints a warning box.
- void `warningSubBox` (const char *message)
Prints a warning box as a sub box.
- void `errorBox` (const char *message)
Prints an error box.
- void `errorSubBox` (const char *message)
Prints an error box as a sub box.
- void `inputLineBox` (const char *label, char *output, unsigned maxWidth, bool ignore=true)
Prints label, gets a whole line of input and stores it to output.
- void `inputLineSubBox` (const char *label, char *output, unsigned maxWidth, bool ignore=true)
Prints label as a sub box, gets a whole line of input and stores it to output.
- void `pressEnterToContinue` (bool ignore=true)
Waits for user to press enter.
- void `resetOrderedList` (int starter=1)
Resets the ordered list starting number.
- void `table` (unsigned startNumber, unsigned columns, const char *items)
Prints a string as a table.
- void `_printSubBoxSpacing` ()
Semi-internal function that prints the spacing for sub boxes.
- void `_printInputBoxLabel` (const char *label)
Semi-internal function that prints the label of an input box.
- void `_printOrderedListBeginning` ()
Semi-internal function that prints the latest ordered list index.
- template<typename T >
void `read` (T *storage)
Reads user input and stores it.
- template<typename T >
void `read` (T &storage)
Reads user input and stores it.
- template<typename T >
void `print` (const T *item)
Prints given item.
- template<typename T >
void `print` (const T &item)
Prints given item.
- template<typename T >
void `printLine` (const T *item)
Prints given item and an endline character.

- `template<typename T >`
void **printLine** (const T &item)
Prints given item and an newline character.
- `template<typename T >`
void **printOrderedListElem** (const T &elem)
Prints the element as the latest list item.
- `template<typename T >`
void **inputBox** (const char *label, T *output)
Prints a label and then reads user input and stores it.
- `template<typename T >`
void **inputSubBox** (const char *label, T *output)

6.5.1 Detailed Description

Stores a wide range of functions for simpler/more automated printing.

Adds a lot of functions for printing.

Remarks

iostream is included by necessity (templated functions), you should only use the provided functions, if you can.

6.5.2 Function Documentation

6.5.2.1 `_printInputBoxLabel()`

```
void _printInputBoxLabel (
    const char * label )
```

Semi-internal function that prints the label of an input box.

Parameters

<i>label</i>	C-style string, there are no size checks so it could wrap
--------------	---

Prints "(+)" and then the given label. It doesn't create spacing after label, so it should exist in the label itself.

Remarks

Semi-internal means the function is defined in the header out of necessity (for templated functions)

6.5.2.2 `_printOrderedListBeginning()`

```
void _printOrderedListBeginning ( )
```

Semi-internal function that prints the latest ordered list index.

Prints the latests ordered list index, surrounded by square brackets, and increments the ordered list index.

Remarks

There are no size checks, so list index could overflow.

Semi-internal means the function is defined in the header out of necessity (for templated functions)

6.5.2.3 `_printSubBoxSpacing()`

```
void _printSubBoxSpacing ( )
```

Semi-internal function that prints the spacing for sub boxes.

Sub boxes are like normal boxes, but indented with 4 spaces.

6.5.2.4 `clear()`

```
void clear ( )
```

Clears the console screen.

Erases display and moves cursor to top left corner via ANSI escape sequences: `ESC[2J ESC[1;1H`

Warning

Not all terminals support any or all ANSI escape sequences

Note

Source: <https://stackoverflow.com/a/52895729/12036073>

6.5.2.5 `errorBox()`

```
void errorBox (
    const char * message )
```

Prints an error box.

Parameters

<i>message</i>	C-style string, there are no size checks so it could wrap
----------------	---

Error boxes start with "<E>" and print the message.

Remarks

Success/Warning/Error boxes start with "<C>" where C is an appropriate character.

6.5.2.6 errorSubBox()

```
void errorSubBox (
    const char * message )
```

Prints an error box as a sub box.

Parameters

<i>message</i>	C-style string, there are no size checks so it could wrap
----------------	---

Error boxes start with "<E>" and print the message.

Remarks

Success/Warning/Error boxes start with "<C>" where C is an appropriate character.

See also

[_printSubBoxSpacing\(\)](#)

6.5.2.7 inputLineBox()

```
void inputLineBox (
    const char * label,
    char * output,
    unsigned maxWidth,
    bool ignore )
```

Prints label, gets a whole line of input and stores it to output.

Parameters

<i>label</i>	C-style string, there are no size check so it could wrap
<i>output</i>	Pointer to a char array
<i>maxWidth</i>	Maximum count of characters to read from user input. output MUST be able to hold that many characters!
<i>ignore</i>	Whether or not to ignore the first new-line delimiter. true by default, should be set to false only when an inputLineBox/SubBox has been issued directly prior or when it's the very first issued box command.

Prints "(+)" before the label.

6.5.2.8 inputLineSubBox()

```
void inputLineSubBox (
    const char * label,
    char * output,
    unsigned maxWidth,
    bool ignore )
```

Prints label as a sub box, gets a whole line of input and stores it to output.

Parameters

<i>label</i>	C-style string, there are no size check so it could wrap
<i>output</i>	Pointer to a char array
<i>maxWidth</i>	Maximum count of characters to read from user input. output MUST be able to hold that many characters!
<i>ignore</i>	Whether or not to ignore the first new-line delimiter. true by default, should be set to false only when an inputLineBox/SubBox has been issued directly prior or when it's the very first issued box command.

Prints "(+)" before the label.

See also

[_printSubBoxSpacing\(\)](#)

6.5.2.9 inputSubBox()

```
template<typename T >
void inputSubBox (
    const char * label,
    T * output )
```

Prints a label with sub box spacing and then reads user input and stores it

See also

[_printSubBoxSpacing\(\)](#)

6.5.2.10 pressEnterToContinue()

```
void pressEnterToContinue (
    bool ignore )
```

Waits for user to press enter.

Parameters

<i>ignore</i>	Whether or not to ignore the first new-line delimiter. true by default, should be set to false only when an inputLineBox/SubBox has been issued directly prior or when it's the very first issued box command.
---------------	--

Prints a "Press enter to continue" message and waits for the user

6.5.2.11 resetOrderedList()

```
void resetOrderedList (
    int starter )
```

Resets the ordered list starting number.

Parameters

<i>starter</i>	Sets the starting number by which ordered list will enumerate
----------------	---

Remarks

Although starter can be a negative number, ordered list will always increment up the number.

6.5.2.12 subTitleBox()

```
void subTitleBox (
    const char * title )
```

Prints a title sub box.

Parameters

<i>title</i>	C-style string, there are no size checks so it could wrap
--------------	---

Prints the title, surrounded by "---".

6.5.2.13 successBox()

```
void successBox (
    const char * message )
```

Prints a success box.

Parameters

<i>message</i>	C-style string, there are no size checks so it could wrap
----------------	---

Success boxes start with "<S>" and print the message.

Remarks

Success/Warning/Error boxes start with "<C>" where C is an appropriate character.

6.5.2.14 successSubBox()

```
void successSubBox (
    const char * message )
```

Prints a success box as a sub box.

Parameters

<i>message</i>	C-style string, there are no size checks so it could wrap
----------------	---

Success boxes start with "<S>" and print the message.

Remarks

Success/Warning/Error boxes start with "<C>" where C is an appropriate character.

See also

[_printSubBoxSpacing\(\)](#)

6.5.2.15 table()

```
void table (
    unsigned startNumber,
    unsigned columns,
    const char * items )
```

Prints a string as a table.

Parameters

<i>startNumber</i>	The number by which column and row enumeration begins
<i>columns</i>	How many columns the table should have
<i>items</i>	C-style string, each cell is a single character from the string

Prints a string as a grid/table of characters from top to bottom, left to right. The first character is on the top left, the last one is on the bottom right.

Table is printed until a terminating zero is encountered. The rows are "calculated" from the columns count and the items length.

6.5.2.16 titleBox()

```
void titleBox (
    const char * title )
```

Prints a title box.

Parameters

<i>title</i>	C-style string, there are no size checks so it could wrap
--------------	---

Prints the title, surrounded by "===".

6.5.2.17 warningBox()

```
void warningBox (
    const char * message )
```

Prints a warning box.

Parameters

<i>message</i>	C-style string, there are no size checks so it could wrap
----------------	---

Warning boxes start with "<W>" and print the message.

Remarks

Success/Warning/Error boxes start with "<C>" where C is an appropriate character.

6.5.2.18 warningSubBox()

```
void warningSubBox (
    const char * message )
```

Prints a warning box as a sub box.

Parameters

<i>message</i>	C-style string, there are no size checks so it could wrap
----------------	---

Warning boxes start with "<W>" and print the message.

Remarks

Success/Warning/Error boxes start with "<C>" where C is an appropriate character.

See also

[_printSubBoxSpacing\(\)](#)

6.6 Toolbox.hpp

[Go to the documentation of this file.](#)

```

1 #ifndef HEADER_CONSOLEINTERFACE_TOOLBOX
2 #define HEADER_CONSOLEINTERFACE_TOOLBOX
3
4 #include <iostream>
5
14 #define MAX_LINE_WIDTH 1024
15
17 void clear();
19 void titleBox(const char* title);
21 void subTitleBox(const char* title);
22
24 void successBox(const char* message);
26 void successSubBox(const char* message);
28 void warningBox(const char* message);
30 void warningSubBox(const char* message);
32 void errorBox(const char* message);
34 void errorSubBox(const char* message);
35
37 void inputLineBox(const char* label, char* output, unsigned maxWidth, bool ignore = true);
39 void inputLineSubBox(const char* label, char* output, unsigned maxWidth, bool ignore = true);
41 void pressEnterToContinue(bool ignore = true);
42
44 void resetOrderedList(int starter = 1);
45
47 void table(unsigned startNumber, unsigned columns, const char* items);
48
50 void _printSubBoxSpacing();
52 void _printInputBoxLabel(const char* label);
54 void _printOrderedListBeginning();
55
57 template <typename T>
58 void read(T* storage) {
59     std::cin » storage;
60 }
61
63 template <typename T>
64 void read(T& storage) {
65     std::cin » storage;
66 }
67
69 template <typename T>
70 void print(const T* item) {
71     std::cout « item;
72 }
73
75 template <typename T>
76 void print(const T& item) {
77     std::cout « item;
78 }
79
81 template <typename T>
82 void printLine(const T* item) {
83     std::cout « item « std::endl;
84 }
85
87 template <typename T>
88 void printLine(const T& item) {
89     std::cout « item « std::endl;
90 }
91
93 template <typename T>
94 void printOrderedListElem(const T& elem) {
95     _printOrderedListBeginning();
96     printLine(elem);
97 }
98
100 template <typename T>

```

```

101 void inputBox(const char* label, T* output) {
102     _printInputBoxLabel(label);
103     std::cin » *output;
104 }
105
106 template <typename T>
107 void inputSubBox(const char* label, T* output) {
108     _printSubBoxSpacing();
109     _printInputBoxLabel(label);
110     std::cin » *output;
111 }
112
113 #endif

```

6.7 src/Generic/Date/Date.h File Reference

Stores the declaration of class [Date](#).

```

#include <istream>
#include <ostream>
#include "../String/String.h"

```

Classes

- class [Date](#)
Stores a day, month and year.

Functions

- `std::istream & operator>> (std::istream &istr, Date &dt)`
Reads date from stream with >> operator.
- `std::ostream & operator<< (std::ostream &ostr, const Date &dt)`
Writes date to stream with << operator.

6.7.1 Detailed Description

Stores the declaration of class [Date](#).

6.7.2 Function Documentation

6.7.2.1 operator<<()

```

std::ostream & operator<< (
    std::ostream & ostr,
    const Date & dt )

```

Writes date to stream with << operator.

Uses the stream's << operator to write the date in the format "day.month.year"

Note

Best used with `std::cout` or text `std::ofstream`

6.7.2.2 operator>>()

```
std::istream & operator>> (
    std::istream & istr,
    Date & dt )
```

Reads date from stream with >> operator.

Uses the stream's >> operator to read and parse the day, month and year.

Note

Best used with std::cin or text std::ifstream

6.8 Date.h

[Go to the documentation of this file.](#)

```
1 #ifndef HEADER_DATE
2 #define HEADER_DATE
3
4 #include <istream>
5 #include <ostream>
6 #include "../String/String.h"
7
8 class Date {
9     unsigned short day;
10    unsigned short month;
11    unsigned short year;
12
13    bool isLeapYear();
14    unsigned short daysInMonth();
15
16 public:
17     Date();
18     Date(const char* str);
19     Date(unsigned short day, unsigned short month, unsigned short year);
20
21     unsigned short get_day() const;
22     bool set_day(unsigned short day);
23     unsigned short get_month() const;
24     bool set_month(unsigned short month);
25     unsigned short get_year() const;
26     bool set_year(unsigned short year);
27
28     String createString() const;
29
30     void read(std::istream& istr);
31     void write(std::ostream& ostr);
32     int compare(const Date& other) const;
33 };
34
35 std::istream& operator>>(std::istream& istr, Date& dt);
36 std::ostream& operator<<(std::ostream& ostr, const Date& dt);
37
38 #endif
```

6.9 src/Generic/List/List.hpp File Reference

Stores declaration and definition of templated class [List](#).

```
#include <istream>
#include <ostream>
```

Classes

- class [List< T >](#)

Templated class that stores an array of elements in dynamic memory.

Functions

- `template<typename T >`
`std::istream & operator>> (std::istream &istr, List< T > &obj)`
Reads [List](#) from stream with >> operator.
- `template<typename T >`
`std::ostream & operator<< (std::ostream &ostr, const List< T > &obj)`
Writes [List](#) to stream with << operator.

6.9.1 Detailed Description

Stores declaration and definition of templated class [List](#).

6.9.2 Function Documentation

6.9.2.1 `operator<<()`

```
template<typename T >
std::ostream & operator<< (
    std::ostream & ostr,
    const List< T > & obj )
```

Writes [List](#) to stream with << operator.

Uses the stream's << operator to write the count and then all objects.

Warning

The function depends on the type having the operator << defined, which takes an std::ostream& and writes it's data to it. Return type doesn't matter.

Note

Best used with std::cout or text std::ofstream

6.9.2.2 operator>>()

```
template<typename T >
std::istream & operator>> (
    std::istream & istr,
    List< T > & obj )
```

Reads [List](#) from stream with >> operator.

Uses the stream's >> operator to read and parse the elements. The first item in the stream should be the count.

Warning

The function depends on the type having the operator >> defined, which takes an std::istream& and writes it's data to it. Return type doesn't matter.

Note

Best used with std::cin or text std::ifstream

6.10 List.hpp

[Go to the documentation of this file.](#)

```
1 #ifndef HEADER_LIST
2 #define HEADER_LIST
3
4 #include <istream>
5 #include <ostream>
6
7 template <typename T>
8 class List {
9     protected:
10         T* elements;
11         unsigned length;
12         unsigned count;
13
14         void resize();
15         void free();
16         void copyFrom(const List& other);
17
18     public:
19         List(const T* elements, unsigned elementsCount);
20         void add(const T& element);
21         void insertAt(const T& element, unsigned index);
22         bool removeAt(unsigned index);
23         unsigned findIndex(const T& element) const;
24         bool contain(const T& element) const;
25         T& operator[](unsigned index);
26         const T& operator[](unsigned index) const;
27
28         List<T>& operator+=(const List<T> other);
29
30         std::istream& read(std::istream& istr);
31         std::ostream& write(std::ostream& ostr) const;
32
33         unsigned get_length() const;
34         unsigned get_count() const;
35
36         List();
37         List& operator=(const List& other);
38         List(const List& other);
39         ~List();
40
41         List(List&& other);
42         List& operator=(List&& other);
43 };
44
45 template <typename T>
46 std::istream& operator>>(std::istream& istr, List<T>& obj);
47
48 template <typename T>
```

```

76 std::ostream& operator<<(std::ostream& ostr, const List<T>& obj);
77
78 /* Private */
79
80 template <typename T>
81 void List<T>::resize() {
82     length = (length == 0) ? 8 : length << 1;
83     T* temp = new T[length];
84     for (int i = 0; i < count; i++)
85         temp[i] = elements[i];
86     delete[] elements;
87     elements = temp;
88 }
89
90 template <typename T>
91 void List<T>::free() {
92     delete[] elements;
93 }
94
95 template <typename T>
96 void List<T>::copyFrom(const List& other) {
97     elements = new T[other.length];
98     for (int i = 0; i < other.count; i++)
99         elements[i] = other.elements[i];
100     length = other.length;
101     count = other.count;
102 }
103
104 /* Public */
105
106 template <typename T>
107 List<T>::List(const T* elements, unsigned elementsCount) {
108     length = 8;
109     count = elementsCount;
110     this->elements = new T[length];
111     for (unsigned i = 0; i < count; i++)
112         this->elements[i] = elements[i];
113 }
114
115 template <typename T>
116 void List<T>::add(const T& element) {
117     if (length == count) resize();
118     elements[count++] = element;
119 }
120
121 template <typename T>
122 void List<T>::insertAt(const T& element, unsigned index) {
123     if (index >= count) {
124         add(element);
125         return;
126     }
127     if (length == count) resize();
128     for (unsigned i = count; i > index; i--)
129         elements[i] = elements[i - 1];
130     elements[index] = element;
131     count++;
132 }
133
134 template <typename T>
135 bool List<T>::removeAt(unsigned index) {
136     if (index >= count) return false;
137     for (int i = index; i < count; i++)
138         elements[i] = elements[i + 1];
139     count--;
140     return true;
141 }
142
143 template <typename T>
144 unsigned List<T>::findIndex(const T& element) const {
145     unsigned ind = 0;
146     while (ind < count && elements[ind].compare(element) != 0)
147         ind++;
148     return ind;
149 }
150
151 template <typename T>
152 bool List<T>::contain(const T& element) const {
153     return findIndex(element) < count;
154 }
155
156 template <typename T>
157 T& List<T>::operator[](unsigned index) {
158     return elements[index];
159 }

```

```

187 }
188
189 template <typename T>
190 const T& List<T>::operator[](unsigned index) const {
191     return elements[index];
192 }
193
194 template <typename T>
195 List<T>& List<T>::operator+=(const List<T> other) {
196     for (unsigned i = 0; i < other.length; i++)
197         add(other[i]);
198     return *this;
199 }
200
201 template <typename T>
202 std::istream& List<T>::read(std::istream& istr) {
203     istr.read((char*)&length, sizeof(length));
204     istr.read((char*)&count, sizeof(count));
205
206     delete[] elements;
207     elements = new T[length];
208
209     for (int i = 0; i < count; i++)
210         elements[i].read(istr);
211
212     return istr;
213 }
214
215 template <typename T>
216 std::ostream& List<T>::write(std::ostream& ostr) const {
217     ostr.write((const char*)&length, sizeof(length));
218     ostr.write((const char*)&count, sizeof(count));
219
220     for (int i = 0; i < count; i++)
221         elements[i].write(ostr);
222
223     return ostr;
224 }
225
226 template <typename T>
227 unsigned List<T>::get_length() const {
228     return length;
229 }
230
231 template <typename T>
232 unsigned List<T>::get_count() const {
233     return count;
234 }
235
236 // Rule of 4
237
238 template <typename T>
239 List<T>::List() : List(nullptr, 0) {}
240
241 template <typename T>
242 List<T>& List<T>::operator=(const List& other) {
243     if (this != &other) {
244         free();
245         copyFrom(other);
246     }
247     return *this;
248 }
249
250 template <typename T>
251 List<T>::List(const List& other) {
252     copyFrom(other);
253 }
254
255 template <typename T>
256 List<T>::~~List() {
257     free();
258 }
259
260 // Move semantics
261
262 template <typename T>
263 List<T>::List(List&& other) {
264     length = other.length;
265     count = other.count;
266     elements = other.elements;
267     other.elements = nullptr;
268 }
269
270 template <typename T>
271 List<T>& List<T>::operator=(List&& other) {
272     if (this != &other) {
273         free();

```

```

297         length = other.length;
298         count = other.count;
299         elements = other.elements;
300         other.elements = nullptr;
301     }
302     return *this;
303 }
304
305 /* Outside of class */
306
307 template <typename T>
308 std::istream& operator<<(std::istream& istr, List<T>& obj) {
309     List<T> newObj;
310     unsigned count;
311     istr >> count;
312
313     T temp;
314     for (int i = 0; i < count; i++) {
315         istr >> temp;
316         obj.add(temp);
317     }
318
319     return istr;
320 }
321
322 template <typename T>
323 std::ostream& operator<<(std::ostream& ostr, const List<T>& obj) {
324     ostr << obj.get_count() << std::endl;
325     for (int i = 0; i < obj.get_count(); i++)
326         ostr << obj[i];
327
328     return ostr;
329 }
330
331 #endif

```

6.11 src/Generic/List/OrderedList.hpp File Reference

Stores declaration and definition of templated class [OrderedList](#).

```
#include "List.hpp"
```

Classes

- class [OrderedList< T >](#)
Inherits [List](#), but contains it's elements in a sorted manner.

6.11.1 Detailed Description

Stores declaration and definition of templated class [OrderedList](#).

6.12 OrderedList.hpp

[Go to the documentation of this file.](#)

```

1 #ifndef HEADER_ORDEREDLIST
2 #define HEADER_ORDEREDLIST
3
4 #include "List.hpp"
5
6 template <typename T>
7 class OrderedList : public List<T> {
8 public:

```



```

18     OrderedList();
19     OrderedList(const T* elements, unsigned elementsCount);
20
24     void add(T element) = delete;
28     void insertAt(T element, unsigned index) = delete;
29
31     void insert(T element);
32 };
33
34 template <typename T>
35 OrderedList<T>::OrderedList() : OrderedList(nullptr, 0) {}
36
37 template <typename T>
38 OrderedList<T>::OrderedList(const T* elements, unsigned elementsCount) : List<T>::List(elements,
    elementsCount) {}
39
44 template <typename T>
45 void OrderedList<T>::insert(T element) {
46     if (List<T>::length == List<T>::count) List<T>::resize();
47
48     unsigned insertionInd = 0;
49     while (insertionInd < List<T>::count && List<T>::elements[insertionInd].compare(element) < 0)
50         insertionInd++;
51
52     List<T>::insertAt(element, insertionInd);
53 }
54
55 #endif

```

6.13 src/Generic/String/String.h File Reference

Stores declaration of class [String](#).

```

#include <istream>
#include <ostream>

```

Classes

- class [String](#)

6.13.1 Detailed Description

Stores declaration of class [String](#).

6.14 String.h

[Go to the documentation of this file.](#)

```

1 #ifndef HEADER_STRING
2 #define HEADER_STRING
3
4 #include <istream>
5 #include <ostream>
6
7 class String {
8     char* str;
9     unsigned length;
10
11     void free();
12     void copyFrom(const String& other);
13
14 public:
15     String(unsigned length);
16     String(const char* str);

```

```

24     const char* get_cstr() const;
25     unsigned get_length() const;
26     char& operator[](unsigned index);
27
28     String& operator+=(const char* str);
29     String& operator+=(unsigned number);
30
31     String();
32     String(const String& other);
33     String& operator=(const String& other);
34     ~String();
35
36     String(String&& other);
37     String& operator=(String&& other);
38
39     void read(std::istream& istr);
40     void write(std::ostream& ostr) const;
41     int compare(const String& other) const;
42
43     friend std::istream& operator>(std::istream& istr, String& event);
44     friend std::ostream& operator<(std::ostream& ostr, const String& event);
45 };
46 #endif

```

6.15 src/main.cpp File Reference

Entry point for application.

```

#include "Services/EventService.h"
#include "Services/HallService.h"
#include "Models/Hall.h"
#include "UserInterface/FMITicketSystemConsoleUI.h"

```

Functions

- int [main](#) ()

6.15.1 Detailed Description

Entry point for application.

Contains the [main\(\)](#) function.

6.15.2 Function Documentation

6.15.2.1 main()

```
int main ( )
```

Creates instances of [HallService](#) and [EventService](#) and executes [runUI\(\)](#) function.

6.16 src/Models/Event.h File Reference

Stores the declaration of class [Event](#).

```
#include "../Generic/List/OrderedList.hpp"
#include "../Generic/Date/Date.h"
#include "../Generic/String/String.h"
#include "Hall.h"
#include "Ticket.h"
#include "Reservation.h"
#include <istream>
#include <ostream>
```

Classes

- class [Event](#)
Stores an event's hall, name, date, bought tickets and reservations.

Functions

- std::istream & [operator>>](#) (std::istream &istr, [Event](#) &event)
Reads [Event](#) from stream with >> operator.
- void [configureEventInsertionOp](#) (unsigned setting)
Configures how [Event](#) should be shown, when written to stream with << operator.
- std::ostream & [operator<<](#) (std::ostream &ostr, const [Event](#) &event)
Writes [Event](#) to stream with << operator.

6.16.1 Detailed Description

Stores the declaration of class [Event](#).

6.16.2 Function Documentation

6.16.2.1 [configureEventInsertionOp\(\)](#)

```
void configureEventInsertionOp (
    unsigned setting )
```

Configures how [Event](#) should be shown, when written to stream with << operator.

Parameters

<i>setting</i>	A number, defining the different options. It's best if it is passed as binary number.
----------------	---

Sets a configuration setting for how `operator<<` should behave. Setting is stored as a global variable in the cpp file, so it's independent of `Event` instance.

Each bit in the number is a certain setting. Each bit from left to right represents one of these options, from top to bottom:

1. Write count of bought tickets (available only if 5. is set)
2. Write count of reserved tickets (available only if 5. is set)
3. Write reservations
4. Write bought tickets
5. Write `Hall` information

Example

To configure printing the `Hall` information, bought tickets count and the bought tickets themselves, you would do:

```
configureEventInsertionOp(0b10011);
```

6.16.2.2 `operator<<()`

```
std::ostream & operator<< (
    std::ostream & ostr,
    const Event & event )
```

Writes `Event` to stream with `<<` operator.

Uses the stream's `<<` operator to write all internal data, following setting from `configureEventInsertionOp()`. Stored types must have defined operator `<<`.

Note

Best used with `std::cout` or text `std::ofstream`

6.16.2.3 `operator>>()`

```
std::istream & operator>> (
    std::istream & istr,
    Event & event )
```

Reads `Event` from stream with `>>` operator.

Uses the stream's `>>` operator to read and parse all internal values. Stored types must have defined operator `>>`.

Note

Best used with `std::cin` or text `std::ifstream`

6.17 Event.h

[Go to the documentation of this file.](#)

```

1 #ifndef HEADER_MODEL_EVENT
2 #define HEADER_MODEL_EVENT
3
4
5
6
7
8 #include "../Generic/List/OrderedList.hpp"
9 #include "../Generic/Date/Date.h"
10 #include "../Generic/String/String.h"
11 #include "Hall.h"
12 #include "Ticket.h"
13 #include "Reservation.h"
14 #include <istream>
15 #include <ostream>
16
17
18
19 class Event {
20     Hall hall;
21     String name;
22     Date date;
23     OrderedList<Ticket> tickets;
24     OrderedList<Reservation> reservations;
25
26 public:
27     Event();
28     Event(const Hall& hall, String name, Date date);
29
30
31
32     const Hall& get_hall() const;
33     const String& get_name() const;
34     const Date& get_date() const;
35
36     OrderedList<Ticket>& get_tickets();
37     const OrderedList<Ticket>& get_tickets() const;
38     OrderedList<Reservation>& get_reservations();
39     const OrderedList<Reservation>& get_reservations() const;
40
41     void read(std::istream& istr);
42     void write(std::ostream& ostr);
43     int compare(const Event& other);
44 };
45
46
47
48
49
50
51
52
53
54
55
56
57 std::istream& operator>>(std::istream& istr, Event& event);
58 void configureEventInsertionOp(unsigned setting);
59 std::ostream& operator<<(std::ostream& ostr, const Event& event);
60
61
62
63 #endif

```

6.18 src/Models/Hall.h File Reference

Stores the declaration of class [Hall](#).

```

#include <istream>
#include <ostream>

```

Classes

- class [Hall](#)

Each hall contains a number, rows and seats per row.

Functions

- std::istream & [operator>>](#) (std::istream &istr, [Hall](#) &hall)
Reads [Hall](#) from stream with >> operator.
- std::ostream & [operator<<](#) (std::ostream &ostr, const [Hall](#) &hall)
Writes [Hall](#) to stream with << operator.

6.18.1 Detailed Description

Stores the declaration of class [Hall](#).

6.18.2 Function Documentation

6.18.2.1 `operator<<()`

```
std::ostream & operator<< (  
    std::ostream & ostr,  
    const Hall & hall )
```

Writes [Hall](#) to stream with << operator.

Uses the stream's << operator to write all internal data.

Note

Best used with `std::cout` or text `std::ofstream`

6.18.2.2 `operator>>()`

```
std::istream & operator>> (  
    std::istream & istr,  
    Hall & hall )
```

Reads [Hall](#) from stream with >> operator.

Uses the stream's >> operator to read and parse all internal values.

Note

Best used with `std::cin` or text `std::ifstream`

6.19 Hall.h

[Go to the documentation of this file.](#)

```
1 #ifndef HEADER_MODEL_HALL
2 #define HEADER_MODEL_HALL
3
4 #include <istream>
5 #include <ostream>
6
7 class Hall {
8     int number;
9     unsigned rows;
10    unsigned seatsPerRow;
11
12 public:
13    Hall();
14    Hall(int number);
15    Hall(int number, unsigned rows, unsigned seatsPerRow);
16
17    int get_number() const;
18    unsigned get_rows() const;
19    unsigned get_seatsPerRow() const;
20
21    void read(std::istream& istr);
22    void write(std::ostream& ostr) const;
23    int compare(const Hall& other) const;
24
25 };
26
27 std::istream& operator>>(std::istream& istr, Hall& hall);
28 std::ostream& operator<<(std::ostream& ostr, const Hall& hall);
29
30 #endif
```

6.20 src/Models/Reservation.h File Reference

Stores the declaration of class [Reservation](#).

```
#include "Ticket.h"
#include <istream>
#include <ostream>
```

Classes

- class [Reservation](#)

Each Reservations contains a ticket, a password and a note.

Macros

- #define **PASSWORD_LEN** 8
- #define **NOTE_LEN** 32

6.20.1 Detailed Description

Stores the declaration of class [Reservation](#).

6.21 Reservation.h

[Go to the documentation of this file.](#)

```

1 #ifndef HEADER_MODEL_RESERVATION
2 #define HEADER_MODEL_RESERVATION
3
4
5
6
7
8 #include "Ticket.h"
9 #include <istream>
10 #include <ostream>
11
12 #define PASSWORD_LEN 8
13 #define NOTE_LEN 32
14
15
16
17 class Reservation {
18     Ticket ticket;
19     char password[PASSWORD_LEN];
20     char note[NOTE_LEN];
21
22 public:
23     Reservation();
24     Reservation(const Ticket& ticket);
25     Reservation(const Ticket& ticket, const char* password, const char* note);
26
27     const Ticket& get_ticket() const;
28     const char* get_password() const;
29     const char* get_note() const;
30
31     void read(std::istream& istr);
32     void write(std::ostream& ostr) const;
33     int compare(const Reservation& other) const;
34
35     friend std::istream& operator>>(std::istream& istr, Reservation& reservation);
36     friend std::ostream& operator<<(std::ostream& ostr, const Reservation& reservation);
37 };
38
39 #endif

```

6.22 src/Models/Ticket.h File Reference

Stores the declaration of class [Ticket](#).

```

#include <istream>
#include <ostream>

```

Classes

- class [Ticket](#)
Each [Ticket](#) contains a row and a seat number.

Functions

- std::istream & [operator>>](#) (std::istream &istr, [Ticket](#) &ticket)
Reads [Ticket](#) from stream with >> operator.
- std::ostream & [operator<<](#) (std::ostream &ostr, const [Ticket](#) &ticket)
Writes [Ticket](#) to stream with << operator.

6.22.1 Detailed Description

Stores the declaration of class [Ticket](#).

6.22.2 Function Documentation

6.22.2.1 operator<<()

```
std::ostream & operator<< (
    std::ostream & ostr,
    const Ticket & ticket )
```

Writes `Ticket` to stream with << operator.

Uses the stream's << operator to write row and seat, separated by a space.

Note

Best used with `std::cout` or text `std::ofstream`

6.22.2.2 operator>>()

```
std::istream & operator>> (
    std::istream & istr,
    Ticket & ticket )
```

Reads `Ticket` from stream with >> operator.

Uses the stream's >> operator to read and parse row and seat.

Note

Best used with `std::cin` or text `std::ifstream`

6.23 Ticket.h

[Go to the documentation of this file.](#)

```
1 #ifndef HEADER_HEADER_TICKET
2 #define HEADER_HEADER_TICKET
3
4 #include <istream>
5 #include <ostream>
6
7 class Ticket {
8     unsigned row;
9     unsigned seat;
10
11 public:
12     Ticket();
13     Ticket(unsigned row, unsigned seat);
14
15     unsigned get_row() const;
16     unsigned get_seat() const;
17
18     void read(std::istream& istr);
19     void write(std::ostream& ostr) const;
20     int compare(const Ticket& other) const;
21 };
22
23 std::istream& operator>>(std::istream& istr, Ticket& ticket);
24 std::ostream& operator<<(std::ostream& ostr, const Ticket& ticket);
25
26 #endif
```

6.24 src/Services/EventService.h File Reference

Stores the declaration of class [EventService](#).

```
#include "../Generic/List/OrderedList.hpp"
#include "../Generic/Date/Date.h"
#include "../Models/Event.h"
#include "HallService.h"
#include "StatusCode.h"
```

Classes

- class [EventService](#)
Each [EventService](#) stores all events and a pointer to [HallService](#).

6.24.1 Detailed Description

Stores the declaration of class [EventService](#).

6.25 EventService.h

[Go to the documentation of this file.](#)

```
1 #ifndef HEADER_SERVICES_EVENTSERVICE
2 #define HEADER_SERVICES_EVENTSERVICE
3
4
5
6
7
8 #include "../Generic/List/OrderedList.hpp"
9 #include "../Generic/Date/Date.h"
10 #include "../Models/Event.h"
11 #include "HallService.h"
12 #include "StatusCode.h"
13
14
15
16 class EventService {
17     OrderedList<Event> events;
18
19     const HallService* hs;
20
21     unsigned indexOfEvent(const char* name, const Date& date);
22
23 public:
24     EventService(const HallService* hs);
25     EventService(const HallService* hs, const Event* events, unsigned eventCount);
26
27     StatusCode createEvent(int hallNumber, const String& name, const Date& date);
28     StatusCode cancelEvent(const char* name, const Date& date);
29
30     StatusCode reserveTicket(const char* name, const Date& date, const char* password, const char* note,
31                             const Ticket& ticket);
32     StatusCode cancelTicketReservation(const char* name, const Date& date, const Ticket& ticket);
33     StatusCode ticketIsReserved(const char* name, const Date& date, const Ticket& ticket);
34
35     StatusCode buyTicket(const char* name, const Date& date, const Ticket& ticket, const char* password,
36                         String* reservationNoteOutput);
37     StatusCode buyTicket(const char* name, const Date& date, const Ticket& ticket);
38
39     String createSeatingString(const char* name, const Date& date, unsigned* seatsPerRow);
40
41     List<Event> queryMostWatched(unsigned topN);
42     List<Event> queryInsufficientlyVisited();
43
44     StatusCode reportReservations(const char* name, const Date& date);
45     StatusCode reportBoughtTickets(int hallNumber, const Date& start, const Date& end, bool all = false);
46
47     StatusCode load();
48     StatusCode save();
49 };
50
51 #endif
```

6.26 src/Services/HallService.h File Reference

Stores the declaration of class [HallService](#).

```
#include "../Generic/List/OrderedList.hpp"
#include "../Models/Hall.h"
#include "StatusCode.h"
```

Classes

- class [HallService](#)
Each [HallService](#) stores all halls.

6.26.1 Detailed Description

Stores the declaration of class [HallService](#).

6.27 HallService.h

[Go to the documentation of this file.](#)

```
1 #ifndef HEADER_SERVICES_HALLSERVICE
2 #define HEADER_SERVICES_HALLSERVICE
3
4 #include "../Generic/List/OrderedList.hpp"
5 #include "../Models/Hall.h"
6 #include "StatusCode.h"
7
8
9
10
11
12
13
14 class HallService {
15     OrderedList<Hall> halls;
16
17 public:
18     HallService(const Hall* halls, unsigned hallCount);
19
20     const OrderedList<Hall>& get_halls() const;
21
22     StatusCode load();
23     StatusCode save();
24 };
25
26 #endif
```

6.28 src/Services/StatusCode.h File Reference

Stores declaration of enum `StatusCode`.

Enumerations

- enum [StatusCode](#) {
Success , **W_TicketHadNotBeenReserved** , **E_FileCouldNotBeOpened** , **E_HallDoesntExist** ,
E_EventWillOverlap , **E_EventDoesNotExist** , **E_TicketAlreadyBought** , **E_TicketAlreadyReserved** ,
E_ReservationDoesNotExist , **E_WrongReservationPassword** }
Used for communication between Services and what uses them.

6.28.1 Detailed Description

Stores declaration of enum StatusCode.

6.28.2 Enumeration Type Documentation

6.28.2.1 StatusCode

enum `StatusCode`

Used for communication between Services and what uses them.

Most functions in Services return a StatusCode.

Those starting with W should be considered Warning, and those starting with E should be considered Errors.

6.29 StatusCode.h

[Go to the documentation of this file.](#)

```
1 #ifndef HEADER_SERVICES_STATUSCODE
2 #define HEADER_SERVICES_STATUSCODE
3
14 enum StatusCode {
15     Success,
16
17     W_TicketHadNotBeenReserved,
18
19     E_FileCouldNotBeOpened,
20
21     E_HallDoesntExist,
22
23     E_EventWillOverlap,
24     E_EventDoesNotExist,
25
26     E_TicketAlreadyBought,
27     E_TicketAlreadyReserved,
28     E_ReservationDoesNotExist,
29     E_WrongReservationPassword,
30 };
31
34 #endif
```

6.30 src/UserInterface/FMITicketSystemConsoleUI.h File Reference

Stores declaration of runUI function.

```
#include "../Services/EventService.h"
#include "../Services/HallService.h"
```

Functions

- void `runUI` (`EventService` *eventService, `HallService` *hallService)
Runs the console interface for the ticket system.

6.30.1 Detailed Description

Stores declaration of runUI function.

6.30.2 Function Documentation

6.30.2.1 runUI()

```
void runUI (  
    EventService * eventService,  
    HallService * hallService )
```

Runs the console interface for the ticket system.

Stores `EventService` and `HallService` pointers in internal global variables, calls `EventService` and `HallService` load() functions, initializes all menus, navigates the main menu and finally calls `EventService` and `HallService` save() functions

6.31 FMITicketSystemConsoleUI.h

[Go to the documentation of this file.](#)

```
1 #ifndef HEADER_FMITICKETSYSTEMCONSOLEUI  
2 #define HEADER_FMITICKETSYSTEMCONSOLEUI  
3  
8 #include "../Services/EventService.h"  
9 #include "../Services/HallService.h"  
10  
12 void runUI(EventService* eventService, HallService* hallService);  
13  
14 #endif
```

