COS 121 Project

Thomas Scholtz - 15015026

How to use

A makefile is included with the project. To run the code you need the ncurses library. If you are on a Debian based distro you can install ncurses by running:

```
sudo apt-get install libncurses-dev
```

If you are on a RedHat based distro you can install ncurses by running:

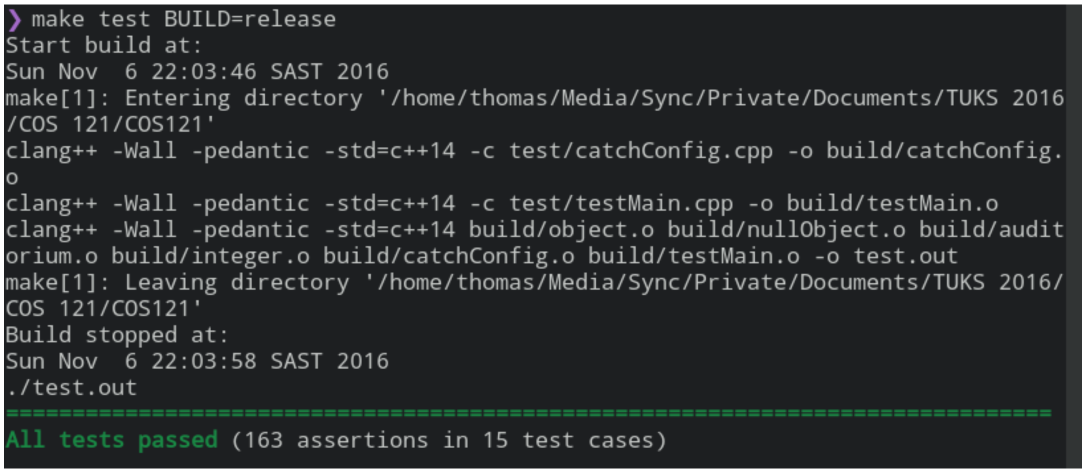
```
sudo dnf install libncurses-dev
```

The Makefile has two targets:
`make test`
and
`make run`

In order to run this project without the debug features you will need to add the variable:
```
make test BUILD=release
make run BUILD=release
```

The unit tests are not dependant on the Libraries. The default run target however is.

Expected output of `make test BUILD=release`

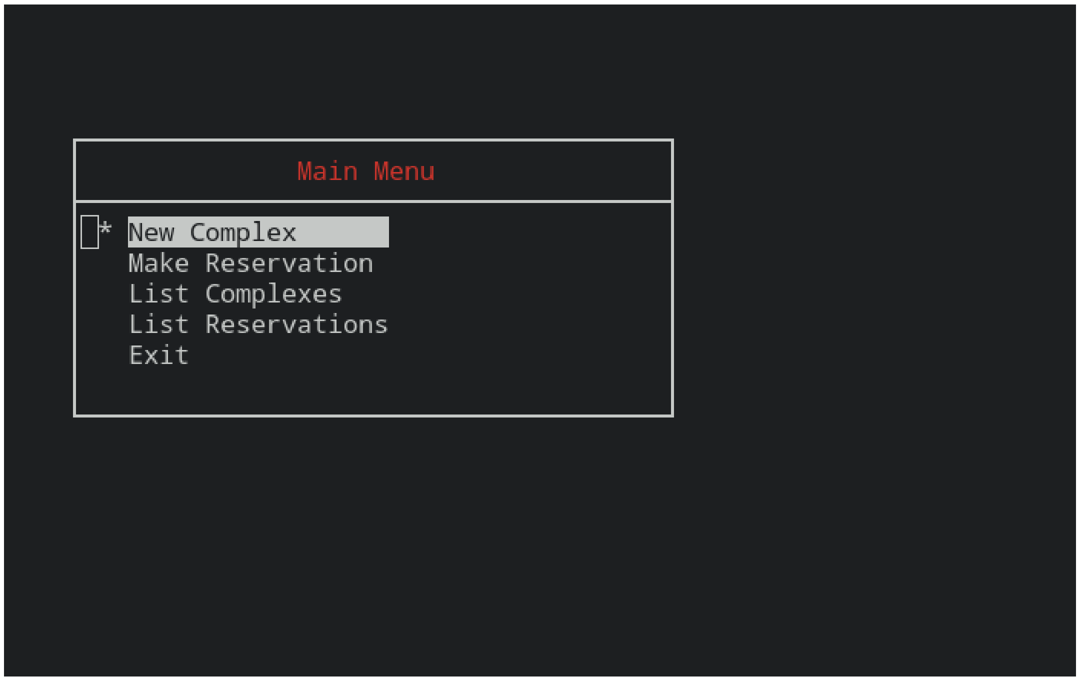


```
❯ make test BUILD=release
Start build at:
Sun Nov  6 22:03:46 SAST 2016
make[1]: Entering directory '/home/thomas/Media/Sync/Private/Documents/TUKS 2016
/COS 121/COS121'
clang++ -Wall -pedantic -std=c++14 -c test/catchConfig.cpp -o build/catchConfig.
o
clang++ -Wall -pedantic -std=c++14 -c test/testMain.cpp -o build/testMain.o
clang++ -Wall -pedantic -std=c++14 build/object.o build/nullObject.o build/audit
orium.o build/integer.o build/catchConfig.o build/testMain.o -o test.out
make[1]: Leaving directory '/home/thomas/Media/Sync/Private/Documents/TUKS 2016/
COS 121/COS121'
Build stopped at:
Sun Nov  6 22:03:58 SAST 2016
./test.out
===============================================================================
All tests passed (163 assertions in 15 test cases)
```

If all tests pass you are good to go!

The code has been tested to compile with `clang++` as well as `g++` this can be seen on the continuous integration site: https://travis-ci.org/Quantum-Sicarius/COS121

Expected output of `make run BUILD=release`



```
                    Main Menu

 * New Complex
   Make Reservation
   List Complexes
   List Reservations
   Exit
```

In case any problems arise you can contact me via:

Cell:	078 036 0680
Email:	u15015026@tuks.co.za
Alternate email:	thomas@cerberus.za.net

Declaration

I, **Thomas Scholtz**, student number u15015026, declare that the work done is mine and solely mine.

The code I wrote is Open Source and the general public is allowed to use it under the MIT Licence:

Subsystem 1

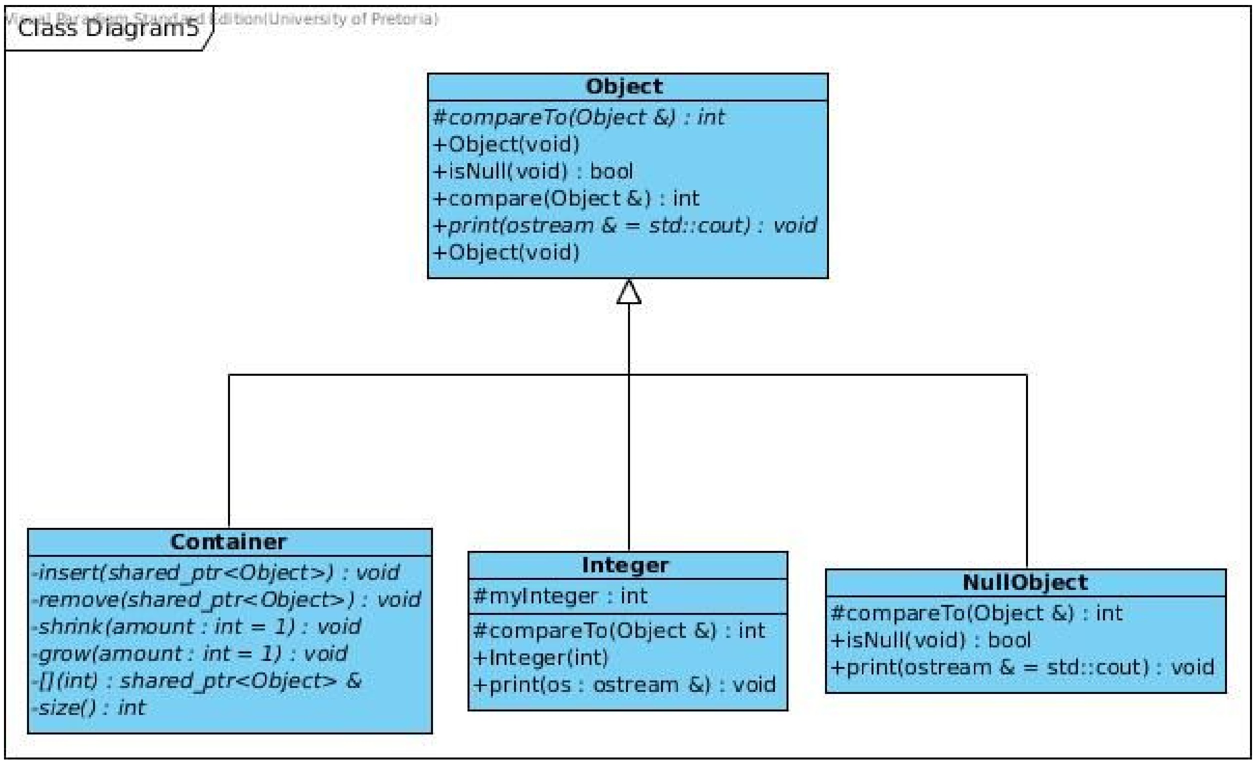
Data Structure - Object Hierarchy

Thomas Scholtz - 15015026

Subsystem Description

This subsystem comprises of the basic data structures.

UML Class Diagram


Visual Paradigm Standard Edition(University of Pretoria)
Class Diagram5
Object
#compareTo(Object &) : int
+Object(void)
+isNull(void) : bool
+compare(Object &) : int
+print(ostream & = std::cout) : void
+Object(void)
Container
-insert(shared_ptr<Object>) : void
-remove(shared_ptr<Object>) : void
-shrink(amount : int = 1) : void
-grow(amount : int = 1) : void
-[](int) : shared_ptr<Object> &
-size() : int
Integer
#myInteger : int
#compareTo(Object &) : int
+Integer(int)
+print(os : ostream &) : void
NullObject
#compareTo(Object &) : int
+isNull(void) : bool
+print(ostream & = std::cout) : void

Design pattern description and motivation

Adapter pattern is used to make the Integer class represent an int data type.

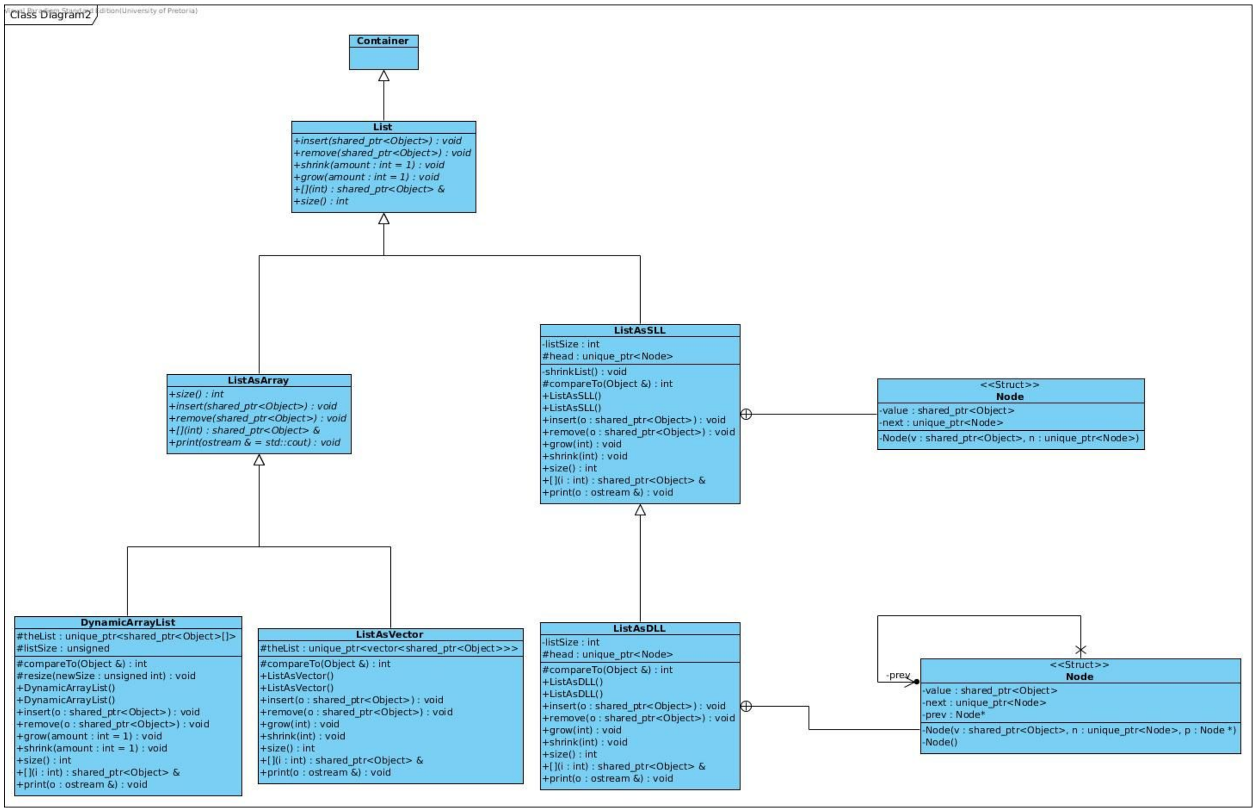
Subsystem 2
Data Structure - List Hierarchy

Thomas Scholtz - 15015026

Subsystem Description

The list hierarchy subsystem is the basic list based data structures.

UML Class Diagram


Visual Paradigm Standard Edition(University of Pretoria)
Class Diagram2
Container
List
+insert(shared_ptr<Object>) : void
+remove(shared_ptr<Object>) : void
+shrink(amount : int = 1) : void
+grow(amount : int = 1) : void
+[](int) : shared_ptr<Object> &
+size() : int
ListAsArray
+size() : int
+insert(shared_ptr<Object>) : void
+remove(shared_ptr<Object>) : void
+[](int) : shared_ptr<Object> &
+print(ostream & = std::cout) : void
ListAsSLL
-listSize : int
#head : unique_ptr<Node>
-shrinkList() : void
#compareTo(Object &) : int
+ListAsSLL()
+ListAsSLL()
+insert(o : shared_ptr<Object>) : void
+remove(o : shared_ptr<Object>) : void
+grow(int) : void
+shrink(int) : void
+size() : int
+[](i : int) : shared_ptr<Object> &
+print(o : ostream &) : void
<<Struct>>
Node
-value : shared_ptr<Object>
-next : unique_ptr<Node>
-Node(v : shared_ptr<Object>, n : unique_ptr<Node>)
DynamicArrayList
#theList : unique_ptr<shared_ptr<Object>[]>
#listSize : unsigned
#compareTo(Object &) : int
#resize(newSize : unsigned int) : void
+DynamicArrayList()
+DynamicArrayList()
+insert(o : shared_ptr<Object>) : void
+remove(o : shared_ptr<Object>) : void
+grow(amount : int = 1) : void
+shrink(amount : int = 1) : void
+size() : int
+[](i : int) : shared_ptr<Object> &
+print(o : ostream &) : void
ListAsVector
#theList : unique_ptr<vector<shared_ptr<Object>>>
#compareTo(Object &) : int
+ListAsVector()
+ListAsVector()
+insert(o : shared_ptr<Object>) : void
+remove(o : shared_ptr<Object>) : void
+grow(int) : void
+shrink(int) : void
+size() : int
+[](i : int) : shared_ptr<Object> &
+print(o : ostream &) : void
ListAsDLL
-listSize : int
#head : unique_ptr<Node>
#compareTo(Object &) : int
+ListAsDLL()
+ListAsDLL()
+insert(o : shared_ptr<Object>) : void
+remove(o : shared_ptr<Object>) : void
+grow(int) : void
+shrink(int) : void
+size() : int
+[](i : int) : shared_ptr<Object> &
+print(o : ostream &) : void
-prev
<<Struct>>
Node
-value : shared_ptr<Object>
-next : unique_ptr<Node>
-prev : Node*
-Node(v : shared_ptr<Object>, n : unique_ptr<Node>, p : Node *)
-Node()

Design pattern description and motivation

Subsystem 3

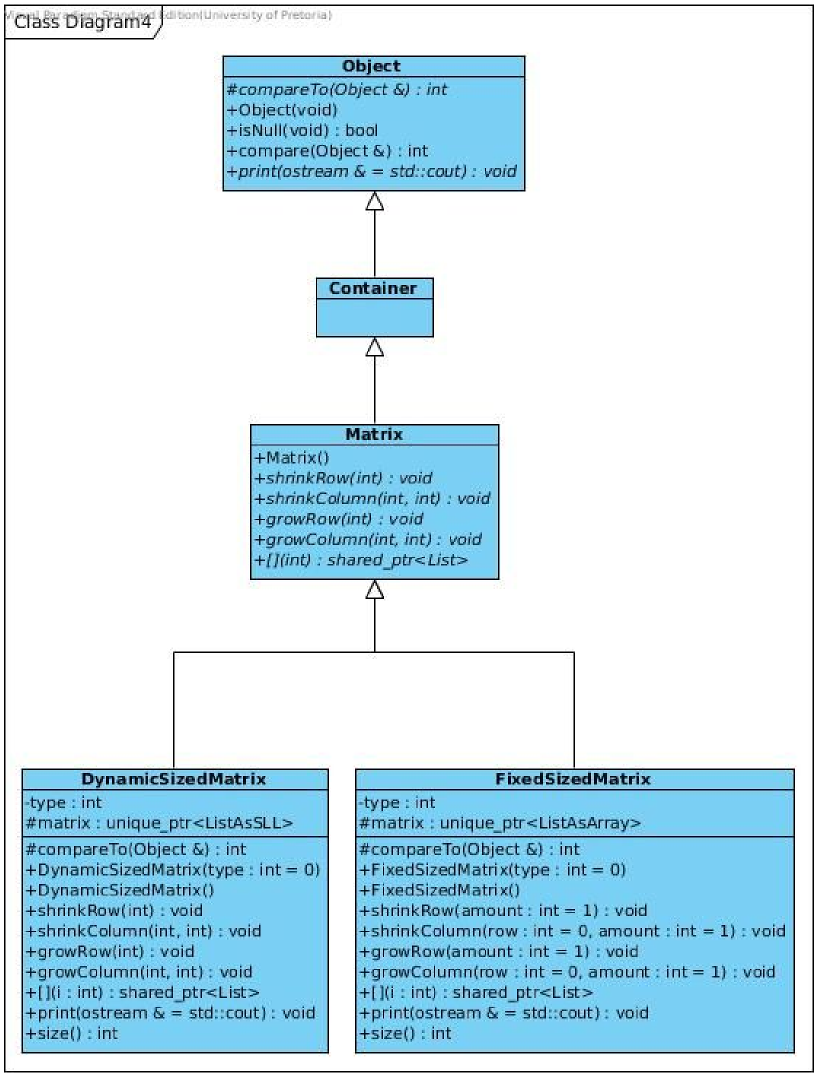
Data Structure - Matrix Hierarchy

Thomas Scholtz - 15015026

Subsystem Description

The matrix hierarchy subsystem comprises of lists of lists. Using the data structures previously created we can nou create matrixes.

UML Class Diagram


Visual Paradigm Standard Edition(University of Pretoria)
Class Diagram4
Object
#compareTo(Object &) : int
+Object(void)
+isNull(void) : bool
+compare(Object &) : int
+print(ostream & = std::cout) : void
Container
Matrix
+Matrix()
+shrinkRow(int) : void
+shrinkColumn(int, int) : void
+growRow(int) : void
+growColumn(int, int) : void
+[](int) : shared_ptr<List>
DynamicSizedMatrix
-type : int
#matrix : unique_ptr<ListAsSLL>
#compareTo(Object &) : int
+DynamicSizedMatrix(type : int = 0)
+DynamicSizedMatrix()
+shrinkRow(int) : void
+shrinkColumn(int, int) : void
+growRow(int) : void
+growColumn(int, int) : void
+[](i : int) : shared_ptr<List>
+print(ostream & = std::cout) : void
+size() : int
FixedSizedMatrix
-type : int
#matrix : unique_ptr<ListAsArray>
#compareTo(Object &) : int
+FixedSizedMatrix(type : int = 0)
+FixedSizedMatrix()
+shrinkRow(amount : int = 1) : void
+shrinkColumn(row : int = 0, amount : int = 1) : void
+growRow(amount : int = 1) : void
+growColumn(row : int = 0, amount : int = 1) : void
+[](i : int) : shared_ptr<List>
+print(ostream & = std::cout) : void
+size() : int

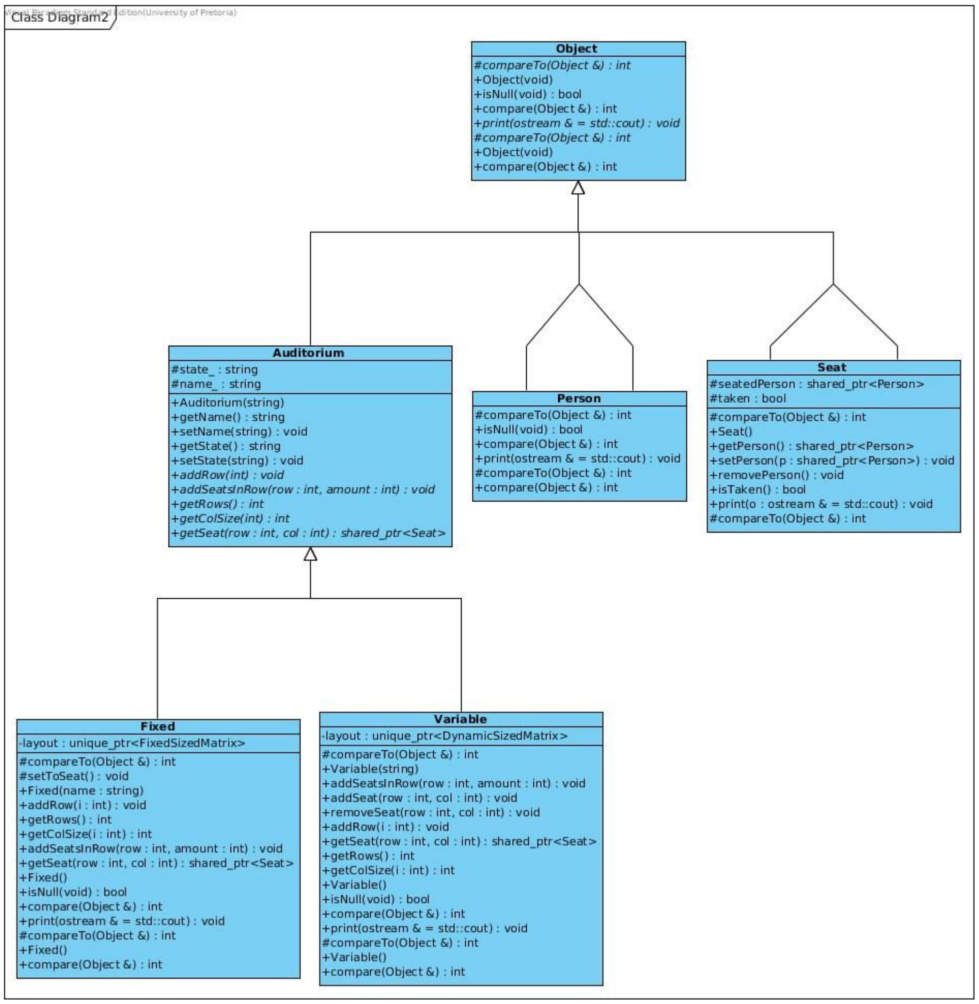
Subsystem 4
Auditorium Modeller

Thomas Scholtz - 15015026

Subsystem Description

This subsystem is responsible for defining different types of auditoriums and how they are designed.

UML Class Diagram


Class Diagram2
Visual Paradigm Standard Edition(University of Pretoria)
Object
#compareTo(Object &) : int
+Object(void)
+isNull(void) : bool
+compare(Object &) : int
+print(ostream & = std::cout) : void
#compareTo(Object &) : int
+Object(void)
+compare(Object &) : int
Auditorium
#state_ : string
#name_ : string
+Auditorium(string)
+getName() : string
+setName(string) : void
+getState() : string
+setState(string) : void
+addRow(int) : void
+addSeatsInRow(row : int, amount : int) : void
+getRows() : int
+getColSize(int) : int
+getSeat(row : int, col : int) : shared_ptr<Seat>
Person
#compareTo(Object &) : int
+isNull(void) : bool
+compare(Object &) : int
+print(ostream & = std::cout) : void
#compareTo(Object &) : int
+compare(Object &) : int
Seat
#seatedPerson : shared_ptr<Person>
#taken : bool
#compareTo(Object &) : int
+Seat()
+getPerson() : shared_ptr<Person>
+setPerson(p : shared_ptr<Person>) : void
+removePerson() : void
+isTaken() : bool
+print(o : ostream & = std::cout) : void
#compareTo(Object &) : int
Fixed
-layout : unique_ptr<FixedSizedMatrix>
#compareTo(Object &) : int
#setToSeat() : void
+Fixed(name : string)
+addRow(i : int) : void
+getRows() : int
+getColSize(i : int) : int
+addSeatsInRow(row : int, amount : int) : void
+getSeat(row : int, col : int) : shared_ptr<Seat>
+Fixed()
+isNull(void) : bool
+compare(Object &) : int
+print(ostream & = std::cout) : void
#compareTo(Object &) : int
+Fixed()
+compare(Object &) : int
Variable
-layout : unique_ptr<DynamicSizedMatrix>
#compareTo(Object &) : int
+Variable(string)
+addSeatsInRow(row : int, amount : int) : void
+addSeat(row : int, col : int) : void
+removeSeat(row : int, col : int) : void
+addRow(i : int) : void
+getSeat(row : int, col : int) : shared_ptr<Seat>
+getRows() : int
+getColSize(i : int) : int
+Variable()
+isNull(void) : bool
+compare(Object &) : int
+print(ostream & = std::cout) : void
#compareTo(Object &) : int
+Variable()
+compare(Object &) : int

Design pattern description and motivation

Template: All derived classes of Auditorium, template is used to create an abstract interface of auditoriums.
Iterator: One can iterate through a auditorium(Row) as well as iterate through columns by calling the relevant list iterator of the matrix.
Prototype: Person class is an abstract class. Minor, Adult and Pensioner all inherit from Person.

Subsystem 5

Auditorium Developer

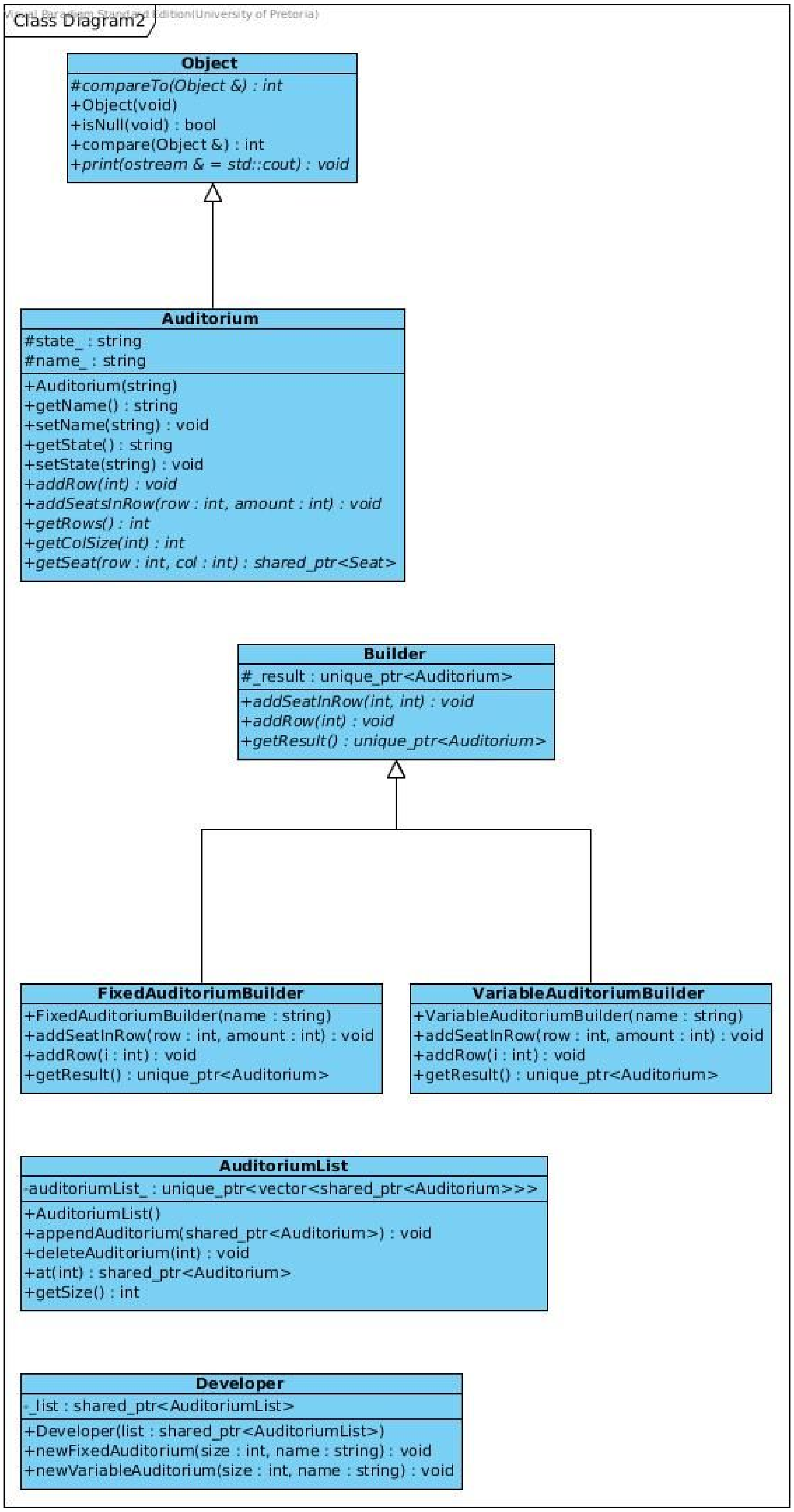
Thomas Scholtz - 15015026

Subsystem Description

This subsystem is responsible for creating advanced auditoriums models.

Design pattern description and motivation

Builder was used to create specific auditoriums.
Factory was used to create abstract auditoriums.
Memento was used to keep the auditorium persistent.


Visual Paradigm Standard Edition(University of Pretoria)
Class Diagram2
Object
#compareTo(Object &) : int
+Object(void)
+isNull(void) : bool
+compare(Object &) : int
+print(ostream & = std::cout) : void
Auditorium
#state_ : string
#name_ : string
+Auditorium(string)
+getName() : string
+setName(string) : void
+getState() : string
+setState(string) : void
+addRow(int) : void
+addSeatsInRow(row : int, amount : int) : void
+getRows() : int
+getColSize(int) : int
+getSeat(row : int, col : int) : shared_ptr<Seat>
Builder
#_result : unique_ptr<Auditorium>
+addSeatInRow(int, int) : void
+addRow(int) : void
+getResult() : unique_ptr<Auditorium>
FixedAuditoriumBuilder
+FixedAuditoriumBuilder(name : string)
+addSeatInRow(row : int, amount : int) : void
+addRow(i : int) : void
+getResult() : unique_ptr<Auditorium>
VariableAuditoriumBuilder
+VariableAuditoriumBuilder(name : string)
+addSeatInRow(row : int, amount : int) : void
+addRow(i : int) : void
+getResult() : unique_ptr<Auditorium>
AuditoriumList
-auditoriumList_ : unique_ptr<vector<shared_ptr<Auditorium>>>
+AuditoriumList()
+appendAuditorium(shared_ptr<Auditorium>) : void
+deleteAuditorium(int) : void
+at(int) : shared_ptr<Auditorium>
+getSize() : int
Developer
-_list : shared_ptr<AuditoriumList>
+Developer(list : shared_ptr<AuditoriumList>)
+newFixedAuditorium(size : int, name : string) : void
+newVariableAuditorium(size : int, name : string) : void

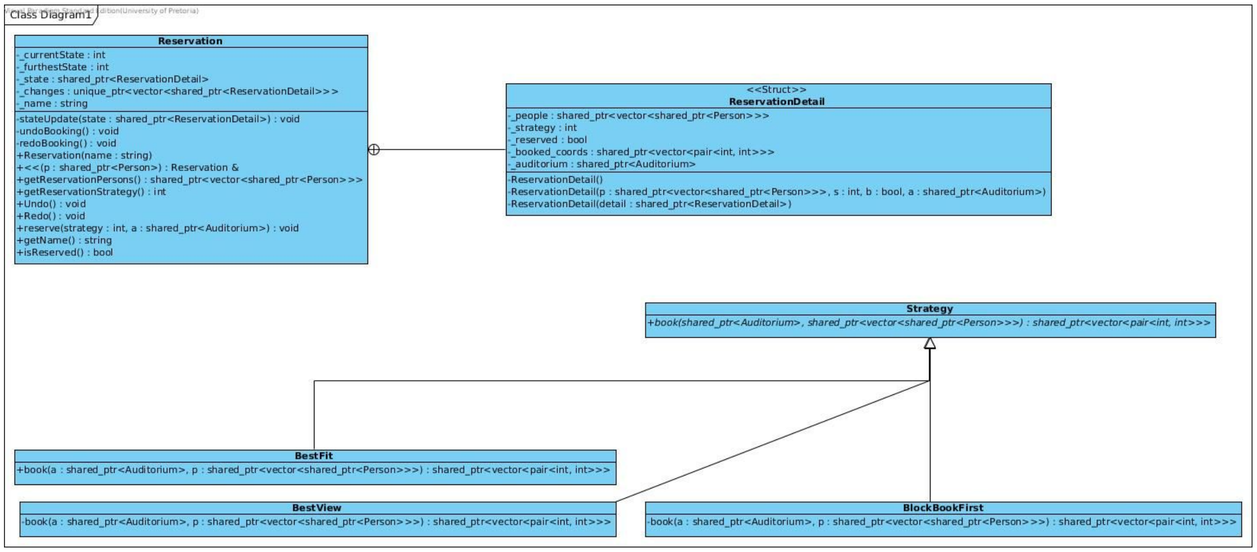
Subsystem 6
Reservation

Thomas Scholtz - 15015026

Subsystem Description

This subsystem handles reservations. A user can create a new reservation add persons and select seats to reserve. Reservations can be undone and redone.

UML Class Diagram


Class Diagram1
Visual Paradigm Standard Edition(University of Pretoria)
Reservation
-_currentState : int
-_furthestState : int
-_state : shared_ptr<ReservationDetail>
-_changes : unique_ptr<vector<shared_ptr<ReservationDetail>>>
-_name : string
-stateUpdate(state : shared_ptr<ReservationDetail>) : void
-undoBooking() : void
-redoBooking() : void
+Reservation(name : string)
+<<(p : shared_ptr<Person>) : Reservation &
+getReservationPersons() : shared_ptr<vector<shared_ptr<Person>>>
+getReservationStrategy() : int
+Undo() : void
+Redo() : void
+reserve(strategy : int, a : shared_ptr<Auditorium>) : void
+getName() : string
+isReserved() : bool
<<Struct>>
ReservationDetail
-_people : shared_ptr<vector<shared_ptr<Person>>>
-_strategy : int
-_reserved : bool
-_booked_coords : shared_ptr<vector<pair<int, int>>>
-_auditorium : shared_ptr<Auditorium>
-ReservationDetail()
-ReservationDetail(p : shared_ptr<vector<shared_ptr<Person>>>, s : int, b : bool, a : shared_ptr<Auditorium>)
-ReservationDetail(detail : shared_ptr<ReservationDetail>)
Strategy
+book(shared_ptr<Auditorium>, shared_ptr<vector<shared_ptr<Person>>>) : shared_ptr<vector<pair<int, int>>>
BestFit
+book(a : shared_ptr<Auditorium>, p : shared_ptr<vector<shared_ptr<Person>>>) : shared_ptr<vector<pair<int, int>>>
BestView
-book(a : shared_ptr<Auditorium>, p : shared_ptr<vector<shared_ptr<Person>>>) : shared_ptr<vector<pair<int, int>>>
BlockBookFirst
-book(a : shared_ptr<Auditorium>, p : shared_ptr<vector<shared_ptr<Person>>>) : shared_ptr<vector<pair<int, int>>>

Design pattern description and motivation

Momento and Command is used to undo and redo reservations.
Strategy pattern is used to simplify the booking strategies.

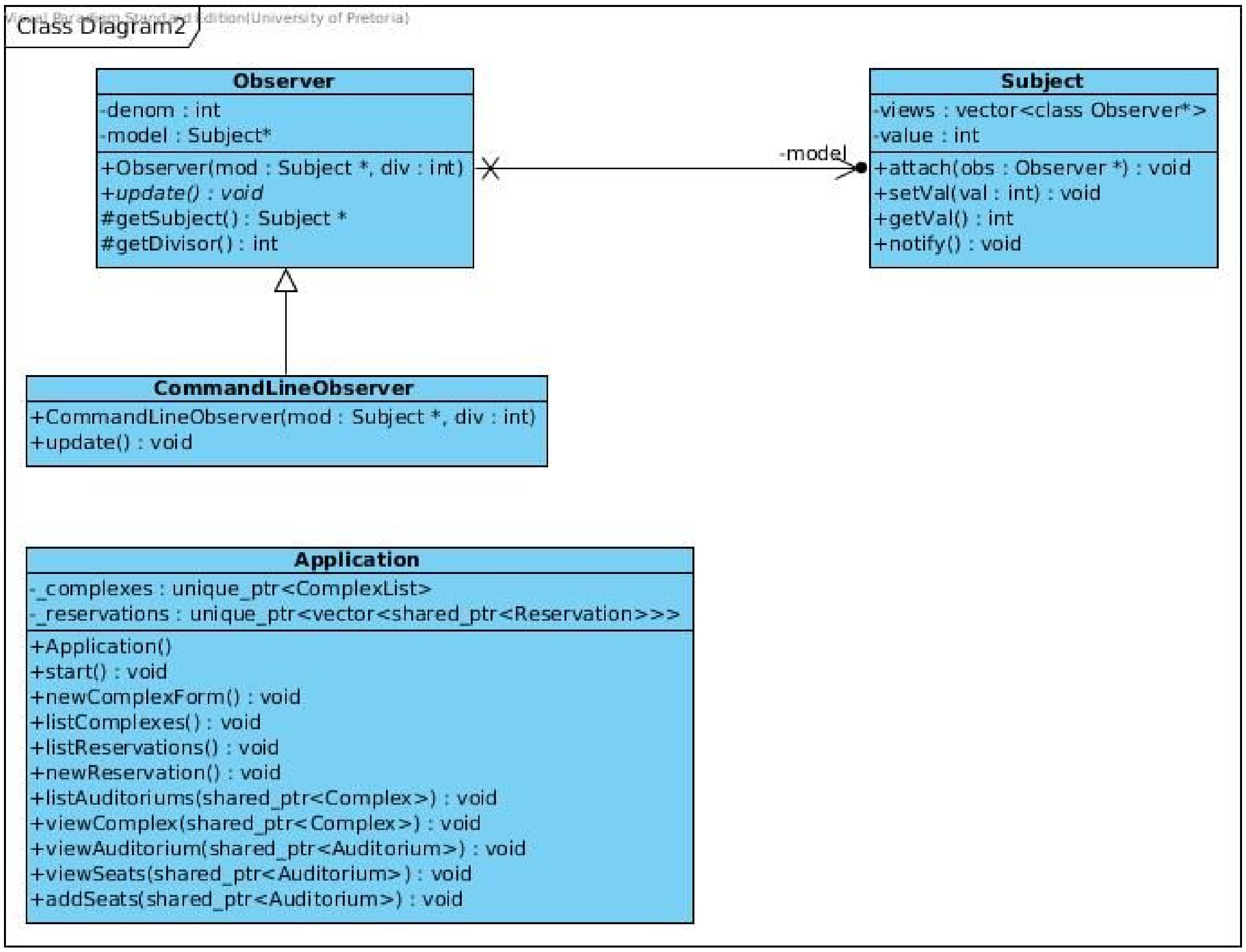
Subsystem 7
User Interface

Thomas Scholtz - 15015026

Subsystem Description

This subsystem comprises of the code used to run the usable program. The user can use either the terminal interface or the web interface.

UML Class Diagram


Class Diagram2
Visual Paradigm Standard Edition(University of Pretoria)
Observer
-denom : int
-model : Subject*
+Observer(mod : Subject *, div : int)
+update() : void
#getSubject() : Subject *
#getDivisor() : int
-model
Subject
-views : vector<class Observer*>
-value : int
+attach(obs : Observer *) : void
+setVal(val : int) : void
+getVal() : int
+notify() : void
CommandLineObserver
+CommandLineObserver(mod : Subject *, div : int)
+update() : void
Application
-_complexes : unique_ptr<ComplexList>
-_reservations : unique_ptr<vector<shared_ptr<Reservation>>>
+Application()
+start() : void
+newComplexForm() : void
+listComplexes() : void
+listReservations() : void
+newReservation() : void
+listAuditoriums(shared_ptr<Complex>) : void
+viewComplex(shared_ptr<Complex>) : void
+viewAuditorium(shared_ptr<Auditorium>) : void
+viewSeats(shared_ptr<Auditorium>) : void
+addSeats(shared_ptr<Auditorium>) : void

Design pattern description and motivation

Observer will be used to keep all corresponding interfaces in sync in the event of data changes.

Chosen interface library

For the terminal interface I used ncurses (https://www.gnu.org/software/ncurses/). I used ncurses because I wanted to rapidly develop an usable interface.