EECS268:Lab4

Contents

Due time

This lab is due 7 days from the start of your lab.

Restrictions

- All solutions must be entirely C++.
- Do NOT use existing standard library data structure to implement this lab. (e.g. don't use vector to implement your stack or queue interfaces)

Program Overview

Elevator Action!

You are being placed in charge of running an elevator in an office building. People enter your building on the ground floor to wait in line for the elevator.

Ground floor Rules:

- The first person in line will be the first to get on the elevator.
- There is no cap on how many people can be waiting for the elevator

Elevator Rules:

- The first person to get into the elevator will be last one off
- To avoid violating fire codes, only 7 people may occupy the elevator

You will read in from file what is happening to the elevator. You can assume the file will be well formatted, but you **may not assume** that the commands will be given in a logical order or have safe values. For example, the command "DROP_OFF 10" could be in the file when no one is in the elevator. When problems like this arise, **you must handle the thrown exception!** Do not simply disallow such actions to occur; you need to get practice catching and handling exceptions.

Commands from file:

Command	Description
WAIT <name>	▪ Someone is waiting in line for the elevator ▪ It will be a single word (no whitespace)
PICK_UP	▪ As many people that can fit get on the elevator
DROP_OFF <#>	▪ The elevator goes up into the building ▪ N number of people get out ▪ Then the elevator returns to the ground floor

INSPECTION	▪ The status of the following is printed to the screen
	    ▪ Is the elevator empty?
	    ▪ Who will be the next to get off the elevator?
	    ▪ Who will be the next to get on the elevator?
	`Elevator status:`
	`The elevator is not empty.`
	`Mark will be the next person to leave the elevator.`
	`Stacy will be the next person to get on the elevator.`

Note, there is no exit command. You'll have to read until you reach the end of file.

Here's one way of reading until you reach the end of file:

```
while (inFileObject >> someVar)
{
  //you just successfully read something into someVar
  //so you can proceed or perhaps even continue reading into other variables
}
```

Example File

```
WAIT Bart
WAIT Homer
WAIT Marge
WAIT Lisa
WAIT Maggie
WAIT Fred
INSPECTION
WAIT Wilma
WAIT Betty
WAIT Barney
WAIT George
WAIT Jane
PICK_UP
INSPECTION
DROP_OFF 3
PICK_UP
INSPECTION
```

File overview:

- Six people get in line for the elevator
- Inspection occurs. The following is printed:

```
Elevator status:
The elevator is empty.
No one is in the elevator.
Bart will be the next person to get on the elevator.
```

- Five more people get in line for the elevator
- The elevator is filled
- Inspection occurs. The following is printed:

```
Elevator status:
The elevator is not empty.
Wilma will be the next person to leave the elevator.
Betty will be the next person to get on the elevator.
```

- Three are dropped off
- The elevator filled
- Inspection occurs. The following is printed:

```
Elevator status:
The elevator is not empty.
George will be the next person to leave the elevator.
Jane will be the next person to get on the elevator.
```

We will test your code with files that will cause exceptions.

Example:

```
WAIT Kirk
WAIT Spock
DROP_OFF 15
```

Notable departures from reality:

- Once people have been dropped off by the elevator, they cannot get back on the elevator (in other you don't have keep track of all people in the building)
- We don't care what floor the elevator is going to

Inheritance Review

Inheritance is a way to make an extension to, or a more specialized version of, a class. An interface, in C++, is a class that has all pure virtual methods, except for the destructor. In this lab you will have one interface, StackInterface, which will contain the pure virtual methods - again, minus the destructor - that form a obligation to the inheriting class to implement.

```
         StackInterface
              |
            Stack
```

Some important notes about interfaces:

- Any method that accepts a base class as a parameter, can be passed any of its descendants.
- You **cannot** create an instance of an Interface nor any abstract class (a class with one or more pure virtual methods); it has no implementation so you cannot create an instance of it; you can only declare and pass around instances of subclasses
- A class that inherits from an interface or abstract class **must** implement all pure virtual methods. Otherwise this subclass will be considered abstract as well.
- We define a virtual destructor with an empty body in the interface to ensure that the destructor for all classes derived from the interface will be called when appropriate
- Since no instance of the interface itself will ever be created, we do not define any constructor for interfaces

```
class ExampleInterface
```

```
{
    public:
    virtual ~ExampleInterface() {}; //virtual destructor with empty definition. It's not pure virtual.

    virtual someFunction() = 0; //pure virtual function, must be implemented by subclass
};
```

Classes

StackInterface

Note You do not have a StackInterface.hpp. This file acts as a contract that any subclass must fulfill. The descriptions are to be used in your documentation for this class and the Stack class.

- virtual ~StackInterface() {};
- virtual bool isEmpty() const = 0;
  - returns true if the stack is empty, false otherwise
- virtual void push(const T value) = 0;
  - Entry added to top of stack
  - throws std::runtime_error if push cannot occur (e.g. stack full)
- virtual void pop() = 0;
  - assumes the stack is not empty
  - top of the stack is removed
  - throws std::runtime_error when a pop is attempted on an empty stack. Does not return a value in this case.
- virtual T peek() const = 0;
  - assumes the stack is not empty
  - returns the value at the top of the stack
  - throws a std::runtime_error is attempted on an empty stack. Does not return a value in this case.

QueueInterface

Note You do not have a QueueInterface.cpp. This file acts as a contract that any subclass must fulfill. The descriptions are to be used in your documentation for this class and the Queue class.

- virtual ~QueueInterface() {}
- virtual bool isEmpty() const = 0;
  - returns true if the stack is empty, false otherwise
- virtual void enqueue(const T value) = 0;
  - Entry added to back of queue
- virtual void dequeue() = 0;
  - assumes the queue is not empty
  - front of the front is removed
  - throws std::runtime_error ifattempted on an empty queue. Does not return a value in this case.
- virtual T peekFront() const = 0;
  - assumes the queue is not empty
  - returns the value at the front of the stack
  - throws a std::runtime_error if attempted on an empty queue. Does not return a value in this case.

Stack

- Implements all methods from the StackInterface class.
- Put the precondition, postcondition, return, and throws comments in this class also
- Your stack implementation should complement the problem at hand. HINT: Is there anything we know

that could help guide our stack implementation?

Any method that throws an exception needs to pass a meaningful message to into the constructor of the exception object.

- Use the following messages
  - "Pop attempted on an empty stack"
  - "Peek attempted on an empty stack"

Queue

- Implements all methods in the QueueInterface
- Put the precondition, postcondition, return, and throws comments in this class also
- Your Queue implementation should complement the problem at hand. HINT: Is there anything we know that could help guide our Queue implementation?
- As with Stack, the exceptions thrown should contain meaningful messages

std::runtime_error

This class inherits from ~~std::runtime_error~~ std::exception. Since it inherits from std::exception, and std::runtime_error implements the std::exception class, we can catch basic exceptions:

```
try
{
  something that could throw a std::runtime_error
}
catch(std::exception& e)
{
   std::cout << e.what(); //print what happened
}
```

Rubric

- 60pts Main program
  - [20pts] Stack and Queue implementation makes sense given the problem requirements
  - [10pts] All exceptions are handled
  - [30pts] General functionality of the program matches requirements
- 20pts Modularity
  - Code is well designed with a logical separation of responsibilities among classes
- 10pts Memory Management:
  - There should be no memory leaks and a equal number of allocations and deallocations. Use valgrind ./YourProgramName to verify
  - Any memory leaks or unequal number of allocations to deallocations (barring the bug mentioned in previous labs) will result in a loss of all 10 points
- 10pts Documentation
  - Comments: Pre conditions, Post conditions, Return descriptions in header files. Not require for hpp/cpp files
  - Formatting: Rational use of white space and indentation. Header and implementation files easily readable

Emailing Your Submission

Once you have created the tarball with your submission files, email it to your TA. The email subject line **must** look like "[EECS 268] *SubmissionName*":

> [EECS 268] Lab 0#

Note that the subject should be *exactly* like the line above. Do not leave out any of the spaces, or the bracket characters ("[" and "]"). In the body of your email, include your name and student ID.

Retrieved from "https://wiki.ittc.ku.edu/ittc_wiki/index.php?title=EECS268:Lab4&oldid=23526"