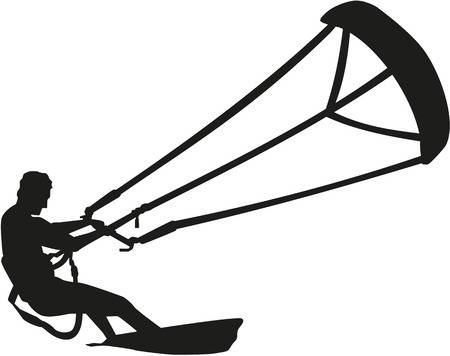
**Warsaw, 28.11.2018**

**Object Oriented Programing Project**

**Kitesurfing School**



## 

## Wojciech Marosek

## Warsaw University of Technology

## ID: 295818

Contents

[1. Goal 3](#_Toc531124592)

[2. Story 3](#_Toc531124593)

[3. Limits and restriction 4](#_Toc531124594)

[4. Memory mapping 5](#_Toc531124595)

[5. Class definitions 5](#_Toc531124596)

[Person.hpp 6](#_Toc531124597)

[Client.hpp 7](#_Toc531124598)

[Instructor.hpp 7](#_Toc531124599)

[Kite.hpp 8](#_Toc531124600)

[Activites.hpp 8](#_Toc531124601)

[KitesurfingSchool.hpp 9](#_Toc531124602)

[6. Tests 10](#_Toc531124603)

[Tests idea: 10](#_Toc531124604)

[Test example: 10](#_Toc531124605)

# Goal

My topic of project on Objected-Oriented Classes is kitesurfing school. Main target of it is creating a simulation of kitesurfing school by building system of management of it. System allows us to optimize daily activities of school. It creates activities, collects list of all clients, instructors. and data about available kites in school. Also we can see a schedule of activities for some day or the whole list of activities. In addition we can calculate final salary for instructor or final price for lesson completed for client.

To proper operation of school we have to set up some assumptions and prevent some mistakes and error.

# Story

Kitesurfing school is created. It have a instructors and sufficient equipment ( number of kites have to be greater or equal number of instructor ). Client come to school and want to start course of kitesurfing. System collect information from him, matching to him the appropriate instructor and adding him to list of client.

Client:

* Client can sign up to course in some term, if his instructor is free in this time and if client don’t have lesson in this time already, and if this lesson doesn’t’ t already exist
* Client can check schedule ( System print client’s list of activities )
* Client can pay for lesson, if he has already had some lesson. System calculate final price for lesson completed
* Client can cancel lesson, if the lesson already exist.

These are possible actions that client can do in my system. If they don’t be possible program will notify us about errors.

School:

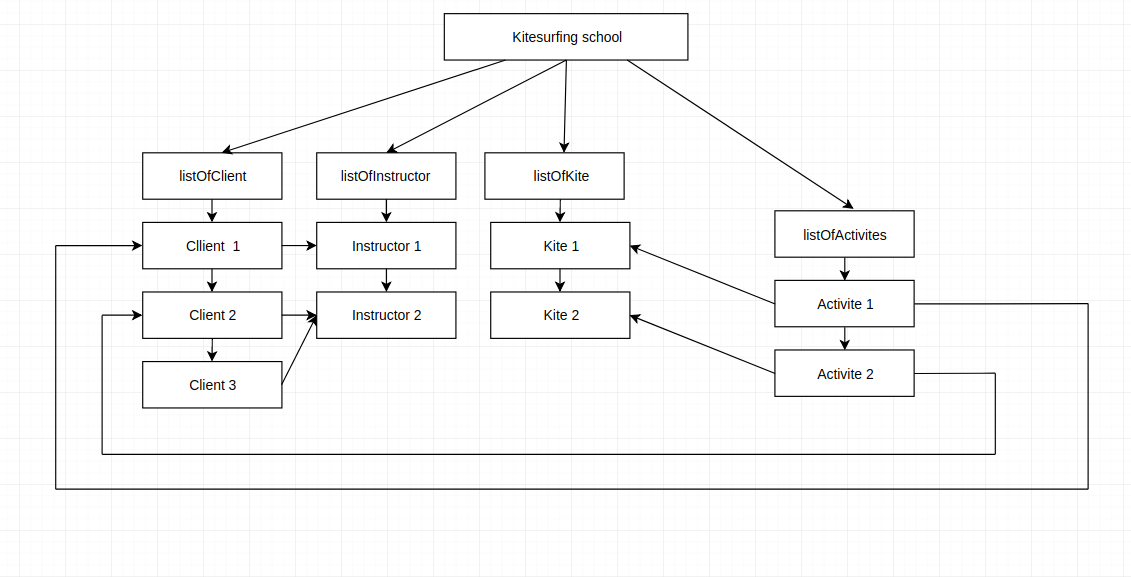
* School can be create and hire new instructors (only if number of instructor is smaller than number of kites) and fire instructors.
* School can buy new kite and sell old one ( only if number of kites is sufficient )
* School can create a schedule for clients, instructors and for the whole school.(only if clients and instructors existing)
* School can matching appropriate instructor to new client by number of instructor’s clients and weight. (Instructors should have similar number of clients)
* School can calculate final salary for instructor by number of lesson.
* School can change level of client depending on the number of lessons

These are possible capabilities that school can do in my system. If they don’t be possible program will notify us about errors.

# Limits and restriction

* Number of instructor has to be smaller than number of kites
* If number of instructor equal zero , we can’t add new client
* The instructor in one time can have only one lesson
* The client in one time can have only one lesson
* Client , instructor, kite , activities ID is unique, so can’t add two objects to one list in this same time
* Instructors should have similar number of clients
* List of clients, instructors , activities, kites will be sorted alphabetically

# Memory mapping



# Class definitions

| -- src

| |--Activites.cpp

| |--Client.cpp

| |--Instructor.cpp

| |--Kite.cpp

| |--KitesurfingSchool.cpp

| |--Person.cpp

| -- include

| |--Activites.hpp

| |--Client.hpp

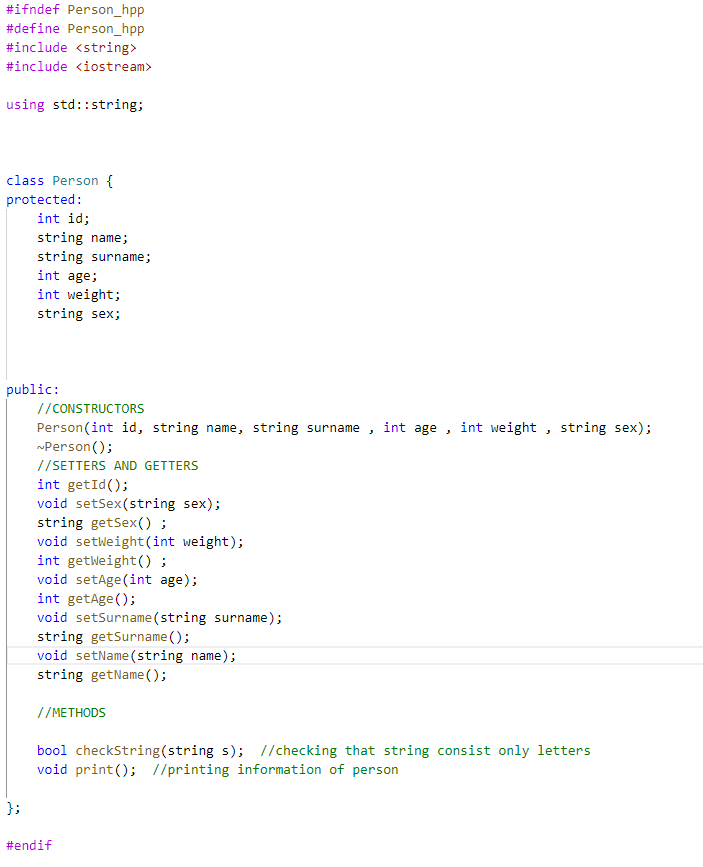
| |--Instructor.hpp

| |--Kite.hpp

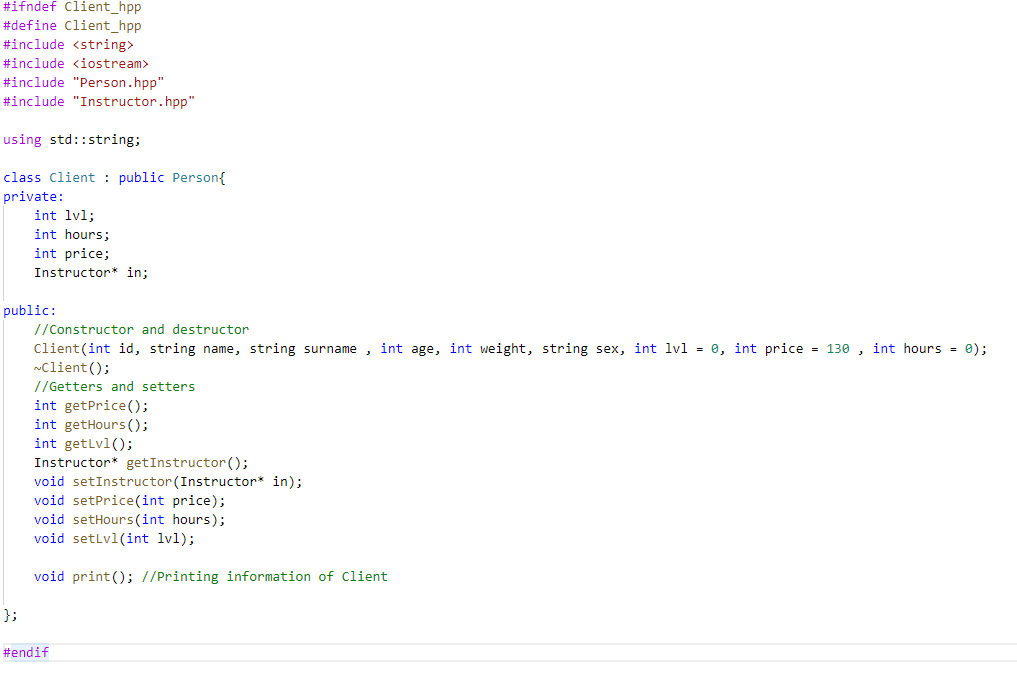
| |--KitesurfingSchool.hpp

| |--Person.hpp

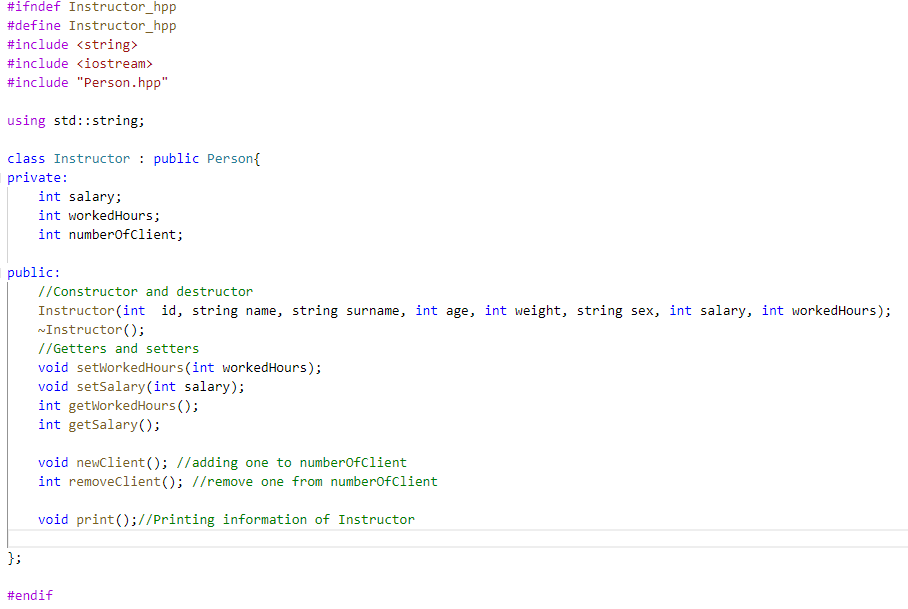
### Person.hpp



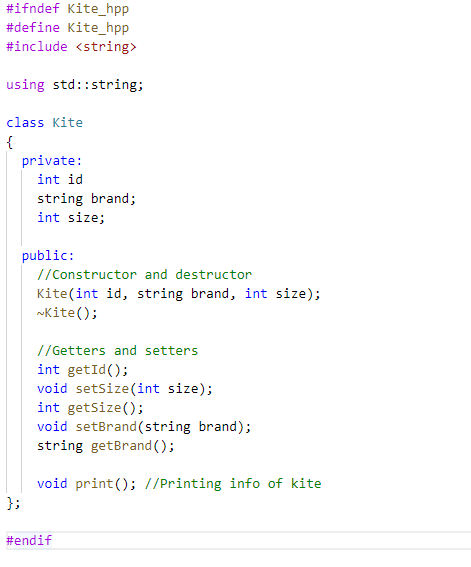
### Client.hpp



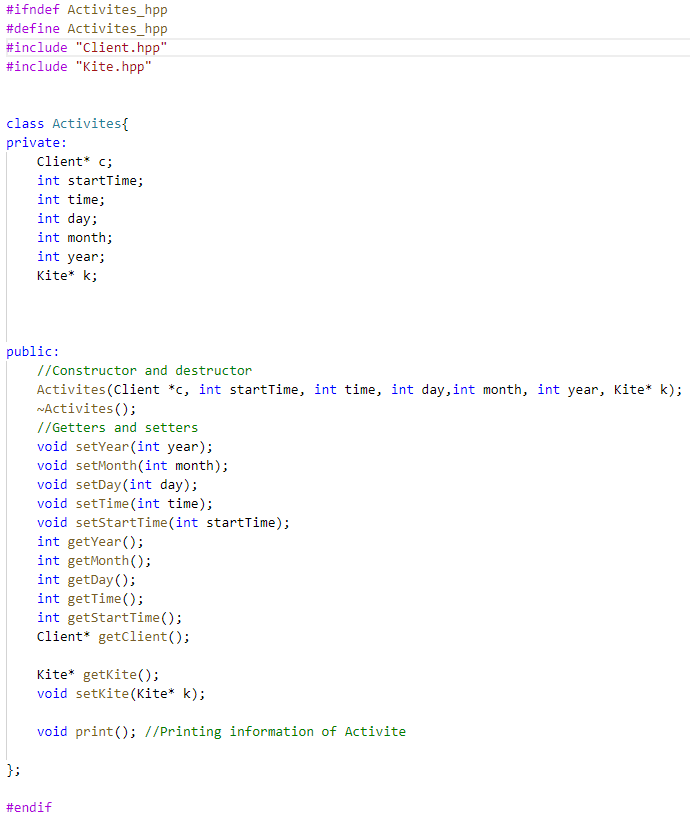
### Instructor.hpp



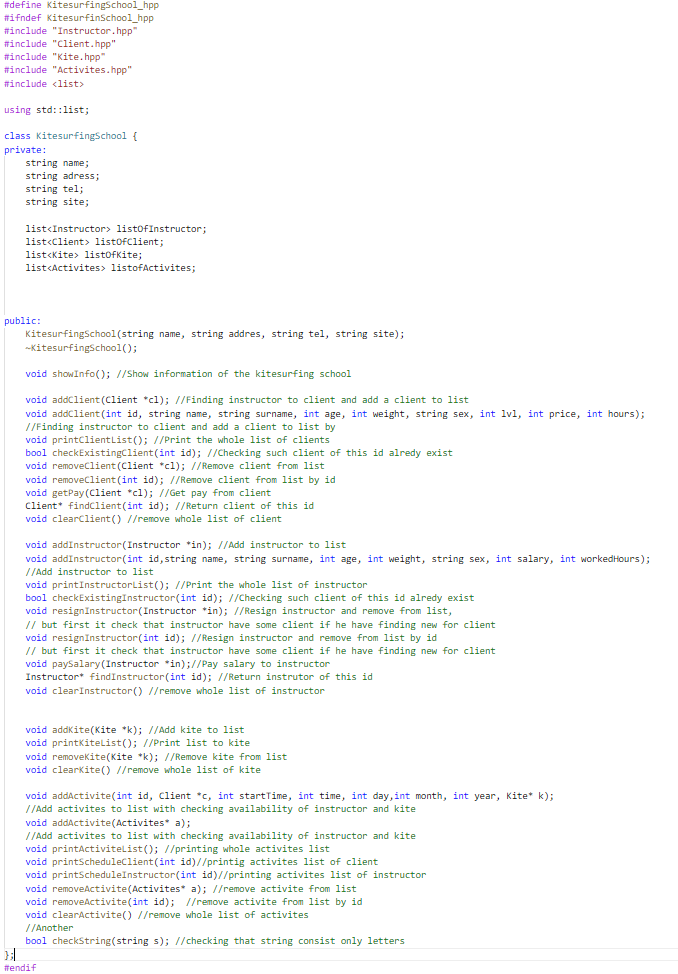
### Kite.hpp



### Activites.hpp



### KitesurfingSchool.hpp

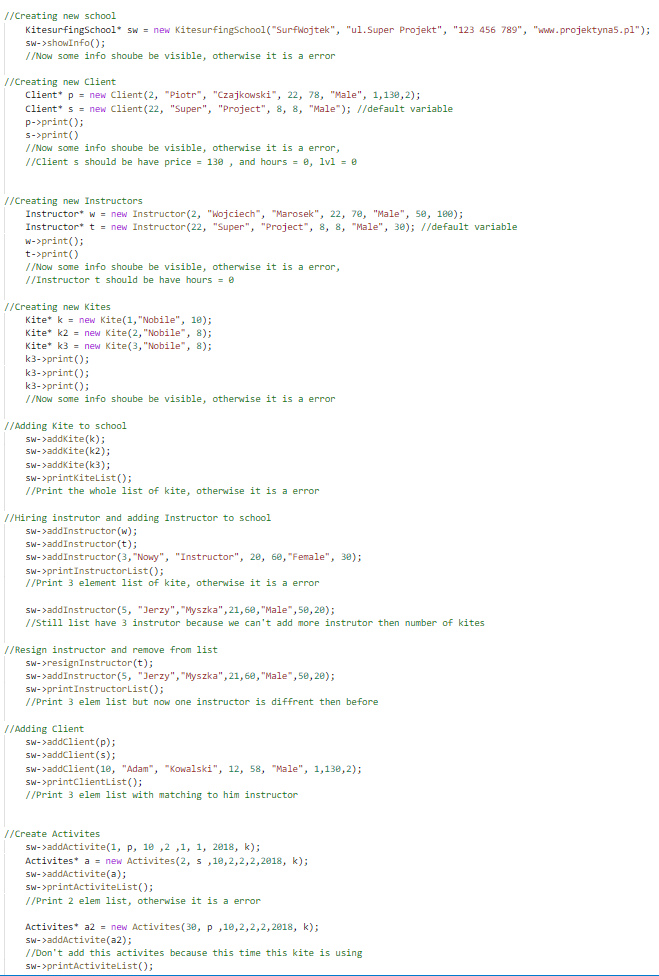


# Tests

## Tests idea:

1. Creating all objects and using methods on them.
2. Calling variables that went out of scope after deletion to see if everything works
3. Creating a hundreds clients, instructor, kites and simulate it and check for exception
4. Adding clients to list, after remove theirs instructor
5. Adding two clients of this same id
6. Paying a salary to not existing instructor
7. Trying make a activate with client and instructor that have already course of this time
8. Paying salary to no existing instructor and getting pay from no existing client.
9. Trying add instructor, when number of kite is equal to number of instructor
10. Trying add client , when number of instructor is zero

## Test example:



## 