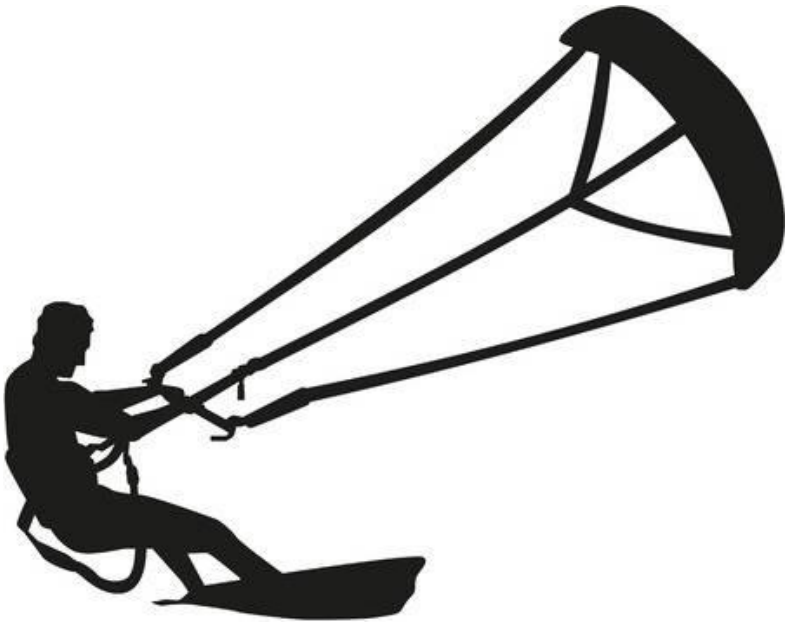
Warsaw, 23.01.2019

Object Oriented Programing Project
Kitesurfing School



Wojciech Marosek
Warsaw University of Technology
ID: 295818

Contents

1. 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.

2. Story

Kitesurfing school is created. It have a instructors and sufficient equipment. 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:

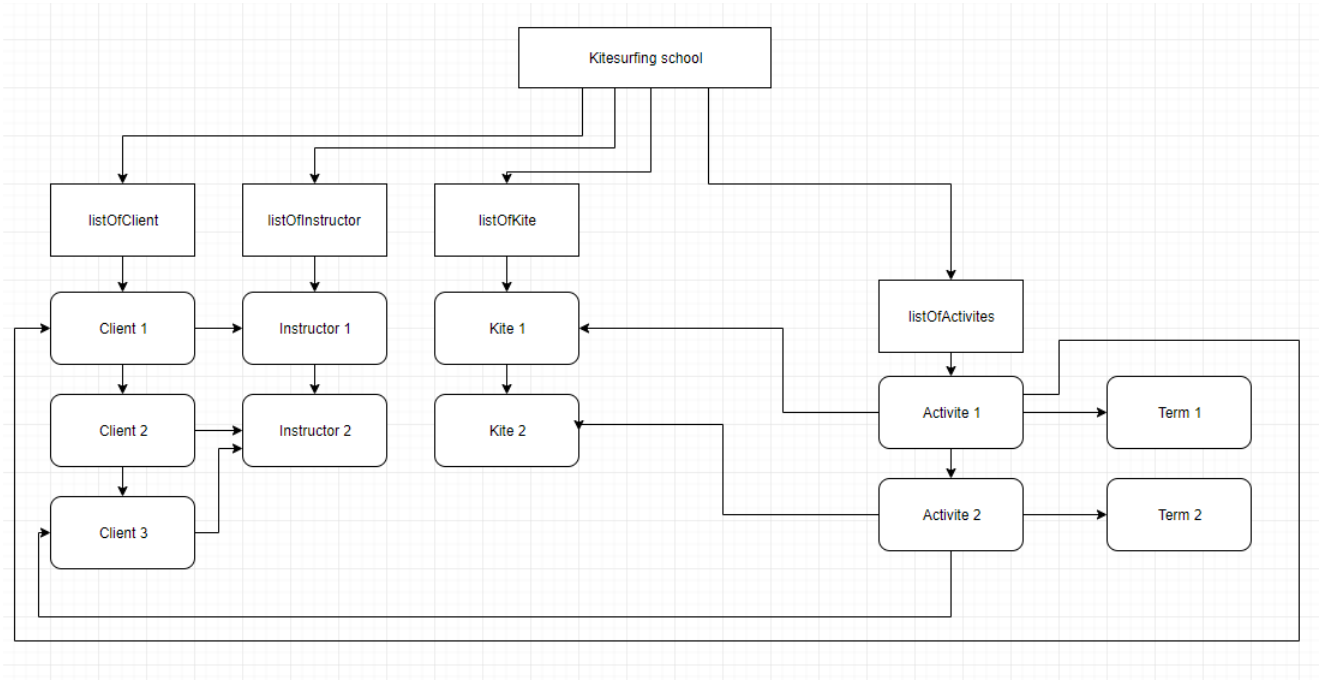
- School can be create and hire new instructors and also fire theirs.
- School can find the best Instructor for Client.
- School can create a schedule for clients, instructors and for the whole school. (only if clients and instructors existing)
- School can matching appropriate instructor to new client by number of instructor's clients. (Instructors should have similar number of clients)
- School can calculate final salary for instructor by product of number of lesson and money per hour.

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

3. Limits and restriction

- 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 by surname.

4. Memory mapping



5. Class definitions

```
| -- src
|    |--Activites.cpp
|    |--Client.cpp
|    |--Instructor.cpp
|    |--Kite.cpp
|    |--KitesurfingSchool.cpp
|    |--Person.cpp
|    |--Term.cpp
| -- include
|    |--Activites.hpp
|    |--Client.hpp
|    |--Instructor.hpp
|    |--Kite.hpp
|    |--KitesurfingSchool.hpp
|    |--Person.hpp
|    |--Term.hpp
```

Person.hpp

```cpp
#ifndef Person_hpp
#define Person_hpp
#include <string>
#include <iostream>

using std::string;


class Person {
protected:
    int id;
    string name;
    string surname;
    int age;
    int weight;
    string sex;


public:
    //CONSTRUCTORS
    Person(int id, string name, string surname , int age , int weight , string sex);
    Person();
    ~Person();
    //SETTERS AND GETTERS
    int getId() const;
    string getSex() ;
    void setSex(string sex);
    void setWeight(int weight);
    int getWeight() ;
    void setAge(int age);
    int getAge();
    void setSurname(string surname);
    string getSurname();
    void setName(string name);
    string getName();

    friend std::ostream& operator<<(std::ostream &os, const Person &p);
    bool operator==(const Person &in) const;
    bool operator!=(const Person &in) const;
    bool operator <(const Person &a) const;
};

#endif
```

Client.hpp

```cpp
#ifndef Client_hpp
#define Client_hpp
#include <string>
#include <iostream>
#include "Person.hpp"
#include "Instructor.hpp"

using std::string;


class Client : public Person{
private:
    int hours;
    int price;
    Instructor *in;


public:
    //Constructor and destructor
    Client();
    Client(int id, string name, string surname , int age, int weight, string sex, int price = 130 , int hours = 0);
    ~Client();
    //Getters and setters
    int getPrice();
    int getHours();
    void setPrice(int price);
    void setHours(int hours);
    void setInstructor(Instructor *in);
    Instructor* getInstructor();
    void getPay(); //Pay by lessons

    friend std::ostream& operator<<(std::ostream &os, const Client &c);

};

#endif
```

Instructor.hpp

```cpp
#ifndef Instructor_hpp
#define Instructor_hpp
#include <string>
#include <iostream>
#include "Person.hpp"

using std::string;

class Instructor : public Person{
private:
    int salary;
    int workedHours;
    int numberOfClient;



public:
    //Constructor and destructor
    Instructor(int  id, string name, string surname, int age, int weight, string sex, int salary, int workedHours = 0);
    ~Instructor();
    Instructor();
    Instructor(const Instructor& in);
    //Getters and setters
    void setWorkedHours(int workedHours);
    void setSalary(int salary);
    int getWorkedHours();
    int getSalary();
    int getNumberOfClient();
    void setNumberOfClient(int numberOfClient);

    Instructor& operator=(const Instructor& in);

    friend std::ostream& operator<<(std::ostream &os, const Instructor &in);

};

#endif
```

Kite.hpp

```cpp
#ifndef Kite_hpp
#define Kite_hpp
#include <string>

using std::string;

class Kite
{
  private:
    int id;
    string brand;
    int size;

  public:
    //Constructor and destructor
    Kite(int id, string brand, int size);
    ~Kite();

    //Getters and setters
    int getId() const;
    void setSize(int size);
    int getSize();
    void setBrand(string brand);
    string getBrand();

    friend std::ostream& operator<<(std::ostream &os, const Kite &k);
    bool operator==(const Kite &k) const;
    bool operator<(const Kite &a) const;
};

#endif
```

Term.hpp

```cpp
#ifndef Term_hpp
#define Term_hpp
#include <string>
#include <iostream>

using std::ostream;

class Term {
private:
    int start;
    int end;
    int day;
    int month;

public:
    Term(int start, int end,  int day , int month);
    ~Term();
    int getMonth() const;
    int getDay() const;
    int getEnd() const;
    int getStart() const;
    friend ostream& operator<<(ostream &os, const Term &t);
    bool operator== (const Term& t2) const;


};

#endif
```

Activites.hpp

```cpp
#ifndef Activites_hpp
#define Activites_hpp
#include "Client.hpp"
#include "Kite.hpp"
#include "Term.hpp"

class Activites{
private:
    int id;
    Client* c;
    Kite* k;
    Term* t;

public:
    //Constructor and destructor
    Activites(int id, Client *c, int startTime, int endTime, int day, int month, Kite* k);
    Activites(int id, Client *c, Term* t, Kite* k);
    ~Activites();
    //Getters and setters
    Client* getClient() const;
    Term* getTerm() const;
    Kite* getKite() const;
    int getId() const;
    void setKite(Kite* k);

    friend ostream& operator<<(ostream &os, const Activites &a);
    bool operator<(const Activites &a) const;
};

#endif
```

KitesurfingSchool.hpp

```
#define KitesurfingSchool_hpp
#ifndef KitesurfinSchool_hpp
#include "Instructor.hpp"
#include "Client.hpp"
#include "Kite.hpp"
#include "Activites.hpp"
#include <list>
#include <algorithm>

using std::list;

class KitesurfingSchool {
private:
    string name;
    string adress;
    string tel;
    string site;

    list<Instructor>listOfInstructor;
    list<Client> listOfClient;
    list<Kite> listOfKite;
    list<Activites> listofActivites;


public:
    KitesurfingSchool(string name, string addres, string tel, string site);
    ~KitesurfingSchool();

    void showInfo();  //Show information of the kitesurfing school


    void addClient(Client *cl); //Finding instructor to client and add a client to list
    void addClient(int id, string name, string surname, int age, int weight, string sex, int price, int hours);
    //Finding instructor to client and add a client to list by
    void printClientList(); //Print the whole list of clients
    void printClientOfInstructor(int id); //Print list of clients of instutor of this id
    bool checkExistingClient(int id); //Checking such client of this id alredy exist
    void removeClient(Client *cl); //Remove client from list
    void removeClient(int id); //Remove client from list by id
    Client* findClient(int id); //Return client of this id
    void clearClient(); //remove whole list of client


    void addInstructor(Instructor *in); //Add instructor to list
    void addInstructor(int id,string name, string surname, int age, int weight, string sex, int salary, int workedHours);
    //Add instructor to list
    void printInstructorList(); //Print the whole list of instructor
    bool checkExistingInstructor(int id); //Checking such client of this id alredy exist
    void resignInstructor(Instructor *in); //Resign instructor and remove from list,
    // but first it check that instructor have some client if he have finding new for client
    void resignInstructor(int id); //Resign instructor and remove from list by id
    // but first it check that instructor have some client if he have finding new for client
    Instructor* findInstructor(int id); //Return instrutor of this id
    Instructor* findTheBest();
    Instructor* findTheBestWithout(Instructor* in);
    void clearInstructor(); //remove whole list of instructor
    void paySalary(int id);//Pay salary to instructor


    void addKite(Kite *k); //Add kite to list
    void printKiteList(); //Print list to kite
    bool checkExistingKite(int id);
    void removeKite(int id); //Remove kite from list
    void clearKite(); //remove whole list of kite

    void addActivite(int id, Client *c, Kite* k, int startTime, int endTime, int day,int month);
    //Add activites to list with checking availability of instructor and kite
    void addActivite(Activites* a);
    //Add activites to list with checking availability of instructor and kite
    void printActiviteList(); //printing whole activites list
    void printScheduleClient(int id); //printig activites list of client
    void printScheduleInstructor(int id);  //printing activites list of instructor
    void removeActivite(Activites* a); //remove activite from list
    void removeActivite(int id);  //remove activite from list by id
    bool checkExistingActivites(int id);
    void clearActivite(); //remove whole list of activites
    Activites* findActivite(int id);



};
#endif
```

6. Changes

During writing a code of my project I realized that I have to add some new feature:

1. I added a term, which include a start and end hours and simple date (day, month) so now each activity collect information about term , client , kite and instructor.
2. In addition, I replaced a methods like printing info for operator overloading (`operator<<`)
3. Finally, I add some operators to each class.

7. 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:

```
void test(){

    cout << "\n====== TEST 1 =======\n" << endl;
    //Creating new school
    KitesurfingSchool* sw = new KitesurfingSchool("SurfWojtek", "ul.Super Projekt", "123 456 789", "www.projektyna5.pl");
    sw->showInfo();
    //Now some info shoube be visible, otherwise it is a error


    //Creating new Client
    cout << "\n====== Creating new Client =======\n" << endl;
    Client* p = new Client(2782, "Piotr", "Czajkowski", 22, 78, "Male",130,2);
    Client* s = new Client(288, "Super", "Project", 80, 80, "Male");
    cout << *p << endl;
    cout << *s << endl;
    //Now some info shoube be visible, otherwise it is a error,
    //Client s should be have price = 130 , and hours = 0

    //Creating new Instructors
    cout << "\n====== Creating new Instructors =======\n" << endl;
    Instructor* w = new Instructor(2, "Wojciech", "Marosek", 22, 70, "Male", 50, 100);
    Instructor* t = new Instructor(28, "Super", "Project", 80, 80, "Male", 30);
    cout << *w << endl;
    cout << *t << endl;
    //Now some info shoube be visible, otherwise it is a error,
    //Instructor t should be have hours = 0


    //Creating new Kites
    cout << "\n====== Creating new Kites =======\n" << endl;
    Kite* k = new Kite(1,"Nobile", 10);
    Kite* k2 = new Kite(2,"Nobile", 8);
    Kite* k3 = new Kite(3,"Nobile", 8);
    cout << *k << endl;
    cout << *k2 << endl;
    cout << *k3 << endl;
    //Now some info shoube be visible, otherwise it is a error

    //Adding Kite to school
    cout << "\n====== Adding Kite to school =======\n" << endl;
    sw->addKite(k);
    sw->addKite(k2);
    sw->addKite(k3);
    sw->printKiteList();
    //Print the whole list of kite, otherwise it is a error

    //Hiring instrutor and adding Instructor to school
    cout << "\n====== Adding Instructor to school =======\n" << endl;

    sw->addInstructor(t);
    sw->addInstructor(3,"Nowy", "Instructor", 20, 60,"Female", 30,20);
    sw->addInstructor(5, "Jerzy","Myszka",21,60,"Male",50,20);
    sw->addInstructor(w);
    sw->printInstructorList();
    //Print 3 element list of kite, otherwise it is a error

    //Resign instructor and remove from list
    cout << "\n====== Resign Instructor from school =======\n" << endl;

    sw->resignInstructor(t);
    sw->printInstructorList();

    cout << "\n====== Adding Client to school =======\n" << endl;
    //Adding Client
    sw->addClient(p);
    sw->addClient(s);
    sw->addClient(10, "Adam", "Kowalski", 12, 58, "Male", 130,2);
    sw->addClient(102, "Robet", "Musial", 12, 58, "Male", 130,2);
    sw->addClient(100, "Michal", "Kaniuk", 20, 58, "Male",130,2);
    sw->printClientList();

    cout << "\n====== Adding Activities to school =======\n" << endl;
    //Create Activites
    sw->addActivite(3,p,k,12,15,1,1);
    Activites* a = new Activites(2,s,10,12,1,1,k2);
    sw->addActivite(a);
    sw->printActiviteList();

    sw->addActivite(23,p,k3,13,15,1,1);
    sw->printActiviteList();
    //Don't add this activites because this time this kite is using
```

```
Client* u = sw->findClient(102);
if(u!=NULL){
    sw->addActivite(35,u,k,15,18,1,1);
    sw->addActivite(36,u,k,15,18,2,1);
    sw->addActivite(37,u,k,15,18,5,1);
    sw->addActivite(38,u,k,10,12,4,1);

    sw->printActiviteList();
}

sw->printInstructorList();

cout << "\n====== Printing Activities of client =======\n" << endl;

//Schedule client
sw->printScheduleClient(102);

cout << "\n====== Printing Activities of instrutor =======\n" << endl;

//Schedule instructor
sw->printClientOfInstructor(5);
sw->printClientOfInstructor(3);
sw->printScheduleClient(288);
Instructor* in3 = sw->findInstructor(3);
cout << *in3 << endl;

cout << "\n====== Paying salary  =======\n" << endl;

//Paying salarry
sw->paySalary(5);
Instructor* in = sw->findInstructor(5);
cout << *in << endl;

cout << "\n====== Getting pay  =======\n" << endl;

//Getting pay
cout << *u << endl;
u->getPay();
cout << *u << endl;

cout << "\n====== Removing Activities  =======\n" << endl;

//Remove Activite
sw->removeActivite(35);
sw->removeActivite(23);
sw->printActiviteList();

//Remove all activite
sw->clearActivite();
sw->printActiviteList();
//List is empty

cout << "\n====== Removing Clients  =======\n" << endl;

//Remove all client
sw->clearClient();
sw->printClientList();
    //List is empty

cout << "\n====== Removing Kites  =======\n" << endl;

//Remove all kites
sw->clearKite();
sw->printKiteList();
//List is empty


//Remove all instrutor
//sw->clearInstructor();
sw->printInstructorList();
//List is empty
```