Jakub Piotrowicz 318350

Keyword: Hospital

1. **Premise**
   The keyword ‘hospital’ was given. I decided to create 4 classes, in order to divide objects into sub-objects. Hospitals will be divided into wards, wards into employees, and employees into patients.
2. **Class explanation**

   Hospital class:
   - contains: the name of the hospital, a vector containing all wards in the particular hospital, a vector containing all employees within the particular hospital and a vector containing all employees within the particular hospital,
   - wards, employees and patients can be added and removed,
   - a copy constructor allows for easy copying of all data from one hospital to another.

   Ward class:
   - contains: the name of the ward, a pointer to the hospital the ward belongs to, a vector containing all employees within this particular ward, a vector containing all patients within this ward,
   - employees and patients can be added and removed.

   Employee class:

   - contains: the name of the employee, the ID of the employee, the name of the hospital this employee belongs to, the name of the ward this employee belongs to, a vector containing all patients assigned to this employee,
   - patients can be added and remove.

   Patient class:

   - contains: the name of the patient, the ID of the patient, the name of the hospital this patient belongs to, the name of the ward this patient belongs to, the name of the employee this patient is assigned to.

3. Class declarations

Hospital class:

```
#pragma once
#ifndef HOSPITAL_H
#define HOSPITAL_H

#include "Ward.h"
#include "Employee.h"
#include "Patient.h"
#include <vector>

class Hospital
{
private:
    char* name;
    std::vector <Ward*> wards;
    std::vector <Employee*> employees;
    std::vector <Patient*> patients;

public:
    Hospital() = default;
    /**
     * @brief Copy constructor, constructs a new object of this class with the
same pareameters as the given object
     *
     * @param other_obj
     */
    Hospital(const Hospital& other_obj);
    ~Hospital();

    /**
     * @brief Get this object's name
     *
     * @return char*
     */
    char* get_name();

    /**
     * @brief Set this object's name
     *
     * @param name
     */
    void set_name(char* name);

    /**
```

```
     * @brief adds a new ward to this hospital
     *
     * @param name
     * @return true if the ward is added successfully
     * @return false if the ward cannot be added
     */
    bool add_ward(char* name);

    /**
     * @brief removes a ward from this hospital
     *
     * @param name
     * @return true if the ward is removed successfully
     * @return false if the ward cannot be removed
     */
    bool remove_ward(char* name);

    /**
     * @brief adds an employee to this hospital
     *
     * @param name
     * @param ID
     * @param ward_name
     * @return true if the employee is added successfully
     * @return false if the employee cannot be added
     */
    bool add_employee(char* name, int ID, char* ward_name);

    /**
     * @brief removes an employee from this hospital
     *
     * @param name
     * @return true if the employee is removed successfully
     * @return false if the employee cannot be removed
     */
    bool remove_employee(char* name);

    /**
     * @brief adds a patient to this hospital
     *
     * @param name
     * @param ID
     * @param ward_name
     * @param employee_name
     * @return true if the patient is addeed successfully
     * @return false if the patient cannot be added
     */
    bool add_patient(char* name, int ID, char* ward_name, char* em-
ployee_name);
```

```
    /**
     * @brief removes a patient from this hospital
     *
     * @param name
     * @return true if the patient is removed successfully
     * @return false if the patient cannot be removed
     */
    bool remove_patient(char* name);

    /**
     * @brief prints all the information about this hospital
     *
     */
    void print() const;
};

#endif
```

Ward class:

```
#pragma once
#ifndef WARD_H
#define WARD_H

#include <vector>

class Hospital;
class Employee;
class Patient;

class Ward
{
private:
    char* name;
    Hospital* hospital_name;
    std::vector <Employee*> employees;
    std::vector <Patient*> patients;

public:
    Ward() = default;
    Ward(const Ward& other_obj);
    ~Ward();

    /**
     * @brief Get this object's name
     *
     * @return char*
     */
    char* get_name() const;

    /**
     * @brief Set this object's name
     *
     * @param name
     */
    void set_name(char* name);

    /**
     * @brief Get the name of the hospital this object belongs to
     *
     * @return Hospital*
     */
    Hospital* get_hospital_name() const;

    /**
     * @brief Set this object's hospital's name
     *
     * @param hospital_name
     */
    void set_hospital_name(Hospital* hospital_name);

    /**
     * @brief adds an employee to this ward
     *
     * @param name
     * @param ID
     * @return true if the employee is added successfully
     * @return false if the employee cannot be added
     */
    bool add_employee(char* name, int ID);

    /**
     * @brief removes an employee from this ward
     *
     * @param name
     * @return true if the employee is removed successfully
     * @return false if the employee cannot be removed
     */
    bool remove_employee(char* name);

    /**
     * @brief adds a patient to this ward
     *
     * @param name
     * @param ID
     * @param employee_name
     * @return true if the patient is added successfully
     * @return false if the patient cannot be added
     */
    bool add_patient(char* name, int ID, char* employee_name);

    /**
     * @brief removes a patient from this ward
     *
     * @param name
     * @return true if the patient is removed successsfully
     * @return false if the patient cannot be removed
     */
    bool remove_patient(char* name);

    /**
     * @brief prints all information about this ward
     *
     */
    void print() const;
};

#endif
```

Employee class:

```
#pragma once
#ifndef EMPLOYEE_H
#define EMPLOYEE_H

#include <vector>

class Hospital;
class Ward;
class Patient;

class Employee
{
private:
    char* name;
    int ID;
    Hospital *hospital_name;
    Ward *ward_name;
    std::vector <Patient*> patients;

public:
    Employee() = default;
    Employee(const Employee& other_obj);
    ~Employee();
```

```cpp
/**
 * @brief Get this object's name
 *
 * @return char*
 */
char* get_name() const;

/**
 * @brief Set this object's name
 *
 * @param name
 */
void set_name(char* name);

/**
 * @brief Get this object's ID
 *
 * @return int
 */
int get_ID() const;

/**
 * @brief Set this object's ID
 *
 * @param ID
 */
void set_ID(int ID);

/**
 * @brief Get the name of the hospital this employee belongs to
 *
 * @return Hospital*
 */
Hospital* get_hospital_name() const;

/**
 * @brief Set this object's hospital's name
 *
 * @param hospital_name
 */
void set_hospital_name(Hospital* hospital_name);

/**
 * @brief Get the name of the ward this employee belongs to
 *
 * @return Ward*
 */
Ward* get_ward_name() const;
```

```
    /**
     * @brief Set this object's ward name
     *
     * @param ward_name
     */
    void set_ward_name(Ward* ward_name);

    /**
     * @brief adds a patient to this employee
     *
     * @param name
     * @param ID
     * @return true if the patient is added successsfully
     * @return false if the patient cannot be added
     */
    bool add_patient(char* name, int ID);

    /**
     * @brief removes a patient from this employee
     *
     * @param name
     * @return true if the patient is removed successfully
     * @return false if the patient cannot be removed
     */
    bool remove_patient(char* name);

    /**
     * @brief prints all information about this employee
     *
     */
    void print() const;
};

#endif
```

Patient class:

```
#pragma once
#ifndef PATIENT_H
#define PATIENT_H

class Hospital;
class Ward;
class Employee;

class Patient
{
```

```
private:
    char* name;
    int ID;
    Hospital *hospital_name;
    Ward *ward_name;
    Employee *doctor_name;

public:
    Patient() = default;
    Patient(const Patient& other_obj);
    ~Patient();

    /**
     * @brief Get this object's name
     *
     * @return char*
     */
    char* get_name() const;

    /**
     * @brief Set this object's name
     *
     * @param name
     */
    void set_name(char* name);

    /**
     * @brief Get this object's ID
     *
     * @return int
     */
    int get_ID() const;

    /**
     * @brief Set this object's ID
     *
     * @param ID
     */
    void set_ID(int ID);

    /**
     * @brief Get the name of the hospital this patient belongs to
     *
     * @return Hospital*
     */
    Hospital* get_hospital_name() const;

    /**
     * @brief Set this object's hospital's name
```

```cpp
     *
     * @param hospital_name
     */
    void set_hospital_name(Hospital* hospital_name);

    /**
     * @brief Get the name of the ward this patient belongs to
     *
     * @return Ward*
     */
    Ward* get_ward_name() const;

    /**
     * @brief Set this object's ward's name
     *
     * @param ward_name
     */
    void set_ward_name(Ward* ward_name);

    /**
     * @brief Get the name of the employee this patient belongs to
     *
     * @return Employee*
     */
    Employee* get_doctor_name() const;

    /**
     * @brief Set this object's employee's name
     *
     * @param employee_name
     */
    void set_doctor_name(Employee* employee_name);

    /**
     * @brief print all information about this patient
     *
     */
    void print() const;
};

#endif
```

4. Testing

```
//trying to add the same Ward again
    cout << "Test 1: ";
    if(h1.add_ward("Oncology") == false)
    {
        cout << "Failed to add the Ward\n";
    }
```

Trying to add an already existing Ward will result in a failure

```
      //trying to add an Employee to a Ward that doesn't exist

    cout << "Test 2: ";
    h1.add_employee("John Swanson", 1, "ER");
```

Trying to add an employee or a patient to a non-existing Ward will result in a failure

```
//trying to assign a Patient to an Employee who doesn't exist
    cout << "Test 3: ";
    h1.add_patient("Teo", 1, "Oncology","Babe Ruth");

    //trying to assign a Patient to a Ward that doesn't exist
    cout << "Test 4: ";
    h1.add_patient("Teo", 1, "ER", "John Swanson");
```

Similarly, trying to assign patients to employees or wards which do not exist will result in a failure.

Analogically, trying to remove or move non-existing Wards, Employees or Patients will result in failures as well.