1. Description of the project

Project consists of 4 classes, namely “Warehouse”, “Employees”, “Goods” and “Equipment”.

Warehouse – consists of all the employees, goods and equipment it has. We can calculate it’s total capacity by adding up capacities of every equipment. Has it’s size limited by height, width, length and volume. We can plan shipments of large quantities of goods in which case the program will check availability of proper equipment and employees, capacity of the whole warehouse and creates simple transport flow which makes sure that there are no conflicts (for example two goods transported into the same place or two employees assigned to one equipment). It has methods which adds and removes both employees and equipment.

Employees – is a blueprint for all type of employees that can be assigned to work in a Warehouse. (i.e. truck drivers, forklift operator, repairman).

Functionalities: assign employees to jobs, hire, fire, retire and train employees.

Relations – We can assign equipment to employee provided that he knows how to use it (there are in total 10 training “certificates”). Employees can only transport goods if they have proper training connected with that type of goods.

Goods – All the items being stored in the warehouse. Goods are both the item we store and the container we store it with.

Functionalities – 4 types of hazardous materials – flammable, explosive, toxic and corrosive. Goods can be gas, liquid or solid. Every good has it’s weight, height, width, length and volume. length, height and width stored inside one vector – size, volume calculated from the method (For a sake of simplicity we are gonna assume that everything is cuboid).

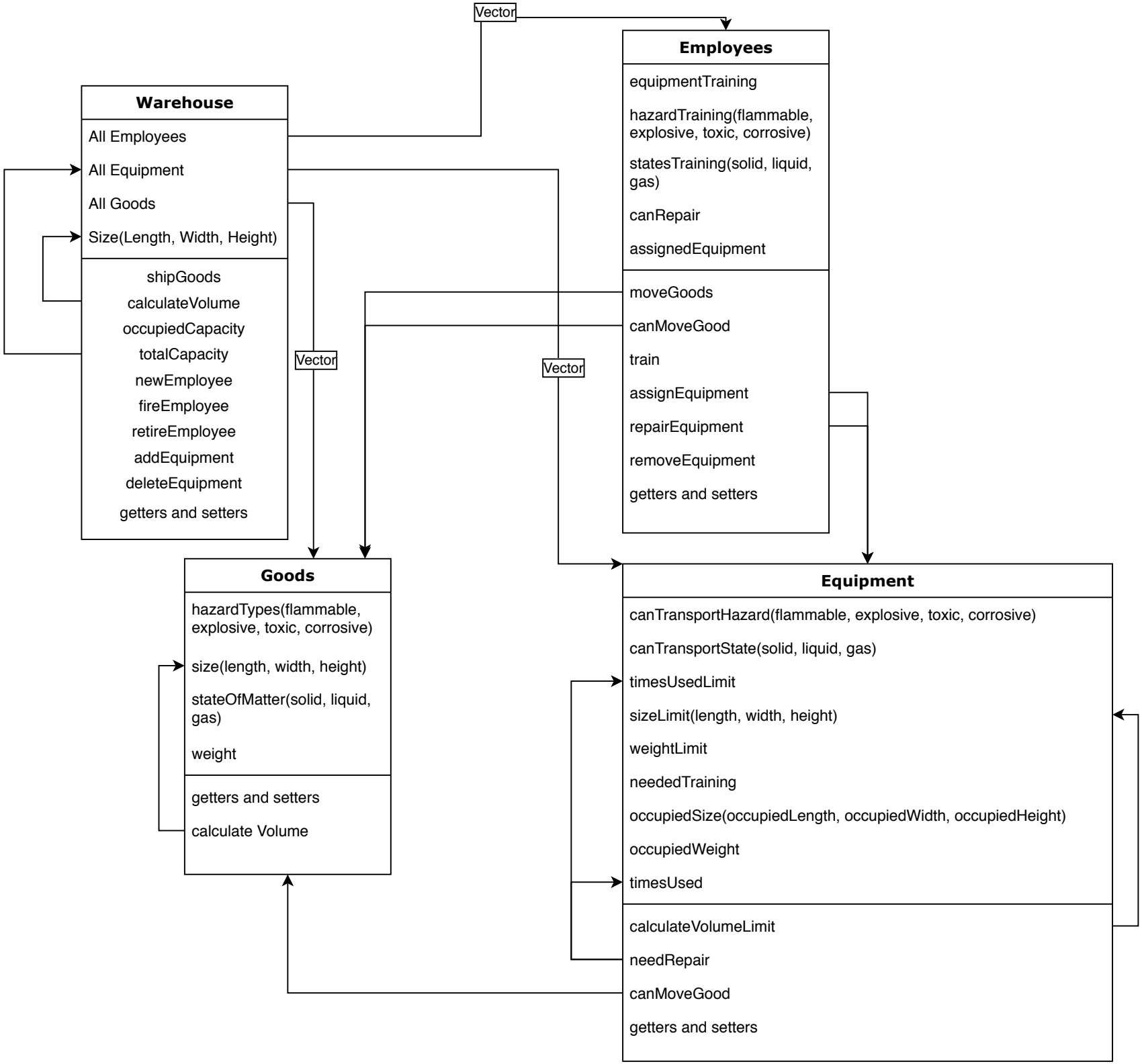
Relations – Hazardous goods or goods in certain state can only be handled by proper equipment and staff trained to handle it. Goods cannot be put on the equipment if their size is too big

Equipment – Objects representing stuff that deals with moving and storing the goods, for example forklift (used to move goods) is equipment and pallet (used to store goods) is also an equipment.

Functionalities – Equipment has weight limit and storage capacity limited by weight, height and width. Weight, height and width stored inside one vector – size, volume calculated from the method. (For a sake of simplicity we are gonna assume that everything is cuboid) Making chain of equipments working together, for example truck → forklift → pallet, where truck extracts goods to forklift which then extracts them on pallet.
Equipment has it’s limit of times it can be used, after which it has to be checked by the repairman. Weight, height, width, length and volume occupied by goods already on the equipment, (starting with 0 and increasing as we put more goods on the equipment, and decreasing after removing goods).

Relations – Can only transport goods within equipment limit, can only transport hazardous goods if it is prepared for it. Can only transport goods if they are in the proper state. Can only be used by trained employees (there are in total 10 training “certificates”). We can assign equipment to the employee.

2. Case study


Vector
Employees
equipmentTraining
hazardTraining(flammable, explosive, toxic, corrosive)
statesTraining(solid, liquid, gas)
canRepair
assignedEquipment
moveGoods
canMoveGood
train
assignEquipment
repairEquipment
removeEquipment
getters and setters
Warehouse
All Employees
All Equipment
All Goods
Size(Length, Width, Height)
shipGoods
calculateVolume
occupiedCapacity
totalCapacity
newEmployee
fireEmployee
retireEmployee
addEquipment
deleteEquipment
getters and setters
Vector
Vector
Goods
hazardTypes(flammable, explosive, toxic, corrosive)
size(length, width, height)
stateOfMatter(solid, liquid, gas)
weight
getters and setters
calculate Volume
Equipment
canTransportHazard(flammable, explosive, toxic, corrosive)
canTransportState(solid, liquid, gas)
timesUsedLimit
sizeLimit(length, width, height)
weightLimit
neededTraining
occupiedSize(occupiedLength, occupiedWidth, occupiedHeight)
occupiedWeight
timesUsed
calculateVolumeLimit
needRepair
canMoveGood
getters and setters

3. Declaration of the classes

```
class Warehouse
{
    private:
        std::vector<Employees> allEmployes;
        std::vector<Equipment> allEquipment;
        std::vector<Goods> allGoods;

        std::vector<unsigned int> size_ = {3};
        // vector[0] - length; 1 - width; 2 - height

    public:
        Warehouse(); // constructor
        ~Warehouse(); // destructor
        Warehouse(const Warehouse& originalWarehouse); // copy constructor
        bool shipGoods(std::vector<Goods> shippedGoods);
        // returns true if the operation was successful or false if operation
        // was unsuccessful

        void newEmployee(Employees newEmployee);

        void fireEmployee(Employees employeeToFire);

        void retireEmployee(Employees employeeToRetire);

        unsigned int calculateVolume() const;
        // from formula height*width*length

        std::vector<unsigned int> occupiedCapacity() const;
        // takes the occupied size of every equipment and returns
        // total occupiedLength, occupiedWidth and occupiedHeight

        std::vector<unsigned int> totalCapacity() const;
        // takes the total capacity of every equipment and returns
        // total length, width and height

        void addEquipment(Equipment newEquipment);

        void deleteEquipment(Equipment equipmentToDelete);

        std::vector<unsigned int> getSize() const;
        std::vector<unsigned int> setSize(std::vector<unsigned int> newSize(3));

};
```

```
class Employees
{
    private:
        std::vector<bool> equipmentTraining(10);
        // there are 10 certificates in total

        std::vector<bool> hazardTraining(4);
        // flammable, explosive, toxic and corrosive

        std::vector<bool> statesTraining(3);
        // solid, liquid, gas

        bool canRepair;
        // whether the employee can or can not repair equipment.

        std::vector<Equipment> assignedEquipment;

    public:
        Employees(); // constructor
        ~Employees(); // destructor
        Employees(const Employees& originalEmployee); // copy constructor

        bool moveGoods(std::vector<Goods> goodsToMove,
                       std::vector<Equipment> equipmentToUse) const;
        // returns true if the operation was successful or false if it was not

        bool canMoveGood(Goods goodChecked) const;
        // returns true if the employee can move good, or false if not

        void train(unsigned int numberOfCertificate);
        // train the employee the specific certificate

        void assignEquipment(Equipment equipmentName);
        // adds equipment to assignedEquipment vector

        void removeEquipment(Equipment equipmentToRemove);

        bool repairEquipment(Equipment equipmentToRepair) const;
        // returns true if the equipment was repaired and false if not

        std::vector<bool> getEquipmentTraining() const;
        std::vector<bool> getHazardTraining() const;
        std::vector<bool> getStatesTraining() const;
        bool getCanRepair() const;
        std::vector<Equipment> getAssignedEquipment() const;

        bool setCanRepair(bool numberToUse) const;


};
```

```cpp
class Equipment
{
    private:
        std::vector<bool> canTransportHazard(4);
        // vector[0] - flammable; 1 - explosive, 2 - toxic, 3 - corrosive

        std::vector<bool> canTransportState(3);
        // vector[0] - solid; 1 - liquid; 2 - gas

        unsigned int timesUsedLimit;
        // after used too many times, the equipment cannot be used under repaired

        std::vector<unsigned int> sizeLimit(3);
        // vector[0] - length; 1 - width; 2 - height

        unsigned int weightLimit;

        std::vector<bool> neededTraining(10);

        std::vector<unsigned int> occupiedSize(3);

        // vector[0] - length; 1 - width; 2 - height

        unsigned int occupiedWeight;

        unsigned int timesUsed;

    public:
        Equipment(); // constructor
        ~Equipment(); // destructor
        Equipment(const Equipment& originalEquipment); // copy constructor

        unsigned int calculateVolumeLimit() const;
        // from equation length * width * height

        bool needRepair() const;

        bool canMoveGood(Goods goodInQuestion) const;
        // returns true if can move good, and false if it cannot

        std::vector<bool> getCanTransportHazard() const;
        std::vector<bool> getCanTransportState() const;
        unsigned int getTimesUsedLimit const;
        std::vector<unsigned int> getSizeLimit() const;
        unsigned int getWeightLimit() const;
        std::vector<bool> getNeededTraining() const;
        std::vector<unsigned int> getOccupiedSize() const;
        unsigned int getOccupiedWeight() const;
        unsigned int getTimesUsed() const;

        std::vector<bool> setCanTransportState(std::vector<bool> newCanTranportState(3));
        std::vector<bool> setCanTransportHazard(std::vector<bool> newCanTransportHazard(4));
        std::vector<bool> setNeededTraining(std::vector<bool> newNeededTraining);
        std::vector<unsigned int> setSizeLimit(std::vector<unsigned int> newSizeLimit(3));
        unsigned int setTimesUsedLimit(unsigned int newTimesUsed);
        unsigned int setWeightLimit(unsigned int newWeightLimit);
        unsigned int setOccupiedWeight(unsigned int newOccupiedWeight);
        unsigned int setTimesUsed(unsigned int newTimesUsed);
};
```

```cpp
class Goods
{
    private:
        std::vector<bool> hazardTypes(4);
        // 0 - flammable; 1 - explosive; 2 - toxic; 3 - corrosive

        std::vector<unsigned int> size_(3);
        // 0 - length; 1 - width; 2 - height

        unsigned int stateOfMatter;
        // 0 - solid; 1 - liquid; 2 - gas;

        unsigned int weight;

    public:
        Goods();
        ~Goods();
        Goods(const Goods& originalGoods); // copy constructor

        unsigned int calculateVolume() const;

        std::vector<bool> getHazardTypes() const;
        std::vector<unsigned int> getSize() const;
        unsigned int getStateOfMatter() const;
        unsigned int getWeight() const;

        std::vector<unsigned int> setSize(std::vector<unsigned int> newSize(3));
        unsigned int setStateOfMatter(unsigned int newStateOfMatter);
        std::vector<bool> setHazardTypes(std::vector<bool> newHazardTypes(4));
        unsigned int setWeight(unsigned int weightToBeSet);

};
```

4. Testing

```cpp
#include "warehouse.hpp"
#include "employees.hpp"
#include "equipment.hpp"
#include "goods.hpp"

#include <iostream>


int main()
{
    // Tests ordered from those that require least compatibility between classes, to ones that require the most compatibility.
  // This is the main function, the rest 4 functions are in next 4 pictures
    checkGoods();
    checkEquipment();
    checkEmployees();
    checkWarehouse();
    std::cout << "End of tests" << std::endl;
    return 0;
}
```

```cpp
void error_message(const std::string& message)
{
    std::cerr << "Error - " << message << "!" << std::endl;
}

void checkGoods
{
    Goods testGoods;
    testGoods.setSize({2, 2, 2});
    if(testGoods.calculateVolume() != 8)
    {
        error_message("Volume of good calculated wrong");
    }
}
```

```
void checkEquipment
{
    Equipment testEquipment;
    testEquipment.setSizeLimit({2, 2, 2});

    if(testEquipment.calculateVolumeLimit() != 8)
    {
        error_message("Volume limit calculated wrong");
    }

    testEquipment.setTimesUsedLimit(10);
    testEquipment.setTimesUsed(11);

    if(!testEquipment.needRepair())
    {
        error_message("needRepair returns 0 despite equipment needing to be repaired");
    }

    testEquipment.setTimesUsed(2);
    if(testEquipment.needRepair())
    {
        error_message("needRepair returns 1 despite equipment not needing to be repaired");
    }

    testEquipment.setWeightLimit(2);
    Goods testGood;
    testGood.setWeight(3);

    if(testEquipment.canMoveGood(testGood))
    {
        error_message("Equipment can move good despite it being too heavy");
    }

    testGood.setWeight(1);
    testGood.setSize({3, 3, 3});
    if(testEquipment.canMoveGood(testGood))
    {
        error_message("Equipment can move good despite it being too big");
    }

    testGood.setWeight(1);
    testGood.setStateOfMatter(0);
    testEquipment.setSizeLimit({5, 5, 5});

    if(testEquipment.canMoveGood(testGood))
    {
        error_message("Equipment can move good despite not being able to move it's state");
    }

    testEquipment.setCanTransportState({1, 1, 1});
    testGood.setHazardTypes({1, 0, 0, 0});

    if(testEquipment.canMoveGood(testGood))
    {
        error_message("Equipment can move good despite not being able to move this type of hazard");

    }
}
```

```
void checkEmployees
{
    Employees testEmployee;
    Goods gold;
    gold.setHazardTypes({1, 0, 0, 0});
    Equipment forklift;
    forklift.setCanTransportHazard({1, 0, 0, 0});
    forklift.setNeededTraining({1, 0, 0, 0, 0, 0, 0, 0, 0})
    forklift.setTimesUsedLimit(10);
    forklift.setTimesUsed(0);

    if(testEmployee.moveGoods({gold}, {forklift}))
    {
        error_message("Employee can move goods despite not having proper training");
    }

    testEmployee.train(0);

    if(!testEmployee.moveGoods({gold}, {forklift}))
    {
        error_message("Employee can not move goods despite having proper training");
    }

    testEmployee.assignEquipment(forklift);
    if(testEmployee.assignedEquipment.size() != 1)
    {
        error_message("Employee was not assigned any equipment");
    }

    testEmployee.removeEquipment(forklift);

    if(testEmployee.assignedEquipment.size() != 0)
    {
        error_message("Employee equipment was not removed");
    }

    forklift.setTimesUsed(15);
    testEmployee.setCanRepair(false);

    if(testEmployee.repairEquipment(forklift))
    {
        error_message("Employee repaired equipment without knowing how");
    }

    testEmployee.setCanRepair(true);

    if(!testEmployee.repairEquipment(forklift))
    {
        error_message("Employee did not repair equipment despite knowing how");
    }
}
```

```
void checkWarehouse
{
    Warehouse testWarehouse;

    Employees testEmployee;
    testWarehouse.newEmployee(testEmployee);
    if(testWarehouse.allEmployes.size() != 1)
    {
        error_message("No employee was added");
    }

    Warehouse newWarehouse(testWarehouse);
    if(newWarehouse.allEmployes.size() != 1)
    {
        error_message("Problem with copy constructor");
    }

    testWarehouse.fireEmployee(testEmployee);
    if(testWarehouse.allEmployes.size() != 0)
    {
        error_message("No employee was fired");
    }

    Goods gold;
    gold.setSize({1, 2, 3});
    if(testWarehouse.shipGoods(gold))
    {
        error_message("Error in shipGoods method");
    }

    testWarehouse.setSize({2, 2, 2});
    if(testWarehouse.calculateVolume() != 8)
    {
        error_message("Error in calculateVolume method");
    }

    Equipment forklift;
    testWarehouse.addEquipment(forklift);
    if(testWarehouse.allEquipment.size() != 1)
    {
        error_message("No equipment was added");
    }

    testWarehouse.deleteEquipment(forklift);
    if(testWarehouse.allEquipment.size() != 0)
    {
        error_message("No equipment was deleted");
    }

    Equipment pallet;
    pallet.setSizeLimit({2, 2, 2});
    if(testWarehouse.totalCapacity() != 8)
    {
        error_message("Error in totalCapacity method");
    }

    Goods testGoods;
    testGoods.setSize({1, 1, 1});
    testWarehouse.shipGoods({testGoods});
    if(testWarehouse.occupiedCapacity() != 1)
    {
        error_message("Error in occupiedCapacity method");
    }
}
```