DSBA Exam C++ 2024

Description

The following project structure is presented:

```
EXAM24
|   Tasks.md
|   Tasks.pdf
|   CMakeList.txt
|
└──src
|   |   main.cpp
|   |   vehicle.hpp
```

`Tasks.md` and `Tasks.pdf` are the same, they just have a different extension and contain a description of the tasks of your exam.

`CMakeList.txt` file contains a set of directives and instructions describing the project's source files and targets.

`main.cpp` source file acts as a check on your decisions from `vehicle.hpp` header file where the assigned tasks should be implemented.

Business description

You are the owner of a transport company and you have several different types of transport. You are faced with the task of digitizing your business and visualizing what your business consists of, how much cargo you can transfer.

You ~~had~~ an employee who has prepared a code for you but didn't know what to do next. Since you completed an Introduction to Programming course, you decided to take on the task of completing the goal.

> It means that you can change the code as you want but the names of the variables and functions should remain as in the task description.

> Problems must be solved in file `vehicle.hpp` and should be uploaded to contest system. `main.cpp` is present only for testing your solutions.

Task 1

Implement the Vehicle class which has the following private fields:

- name: std::string;
- load: double.

Also, you should implement:

- `Vehicle` constructor;
- `getInfo` method, which return a std::string in format `name: %name, load: %load` (where %value is placeholder for a field value).

How to test: In `main.cpp` you can find `test1` function:

```
void test1()
{
    Vehicle veh1("Cart", 325);
    std::cout << veh1.getInfo() << '\n';
}
```

Run `main.cpp` with the function:

```
int main()
{
    test1();
}
```

Output:

```
name: Cart, load: 325.500000
```

Task 2

Implement `Car`, `Truck` and `Boat` classes inherited from `Vehicle`.

`Car` private field:

- meanSpeed: double

`Car` methods:

- `Car` constructors (with mean speed as argument and default speed = 90)
- `getInfo` overloading (`name: %name, load: %load, speed: %meanSpeed`)
- `getLoad()` returns a total load (double)

`Truck` is also inherited from `Car` and has the following private field:

- extraCargo: double

`Truck` methods:

- `Truck` constructors (mean speed, default is 60 and extraCargo, default is 0);

- `getLoad()` overloading to return total load: **load + extraCargo**, you can implement **getter** methods for load and name in `Vehicle` and `getSpeed` for `Car`;
- getInfo overloading (`name: %name, load: %getLoad(), speed: %meanSpeed`).

`Boat` private fields:

- meanKnotSpeed: double

`Boat` methods:

- `Boat` constructors (with mean knot speed as argument and default speed = 30);
- `getInfo` overloading (`name: %name, load: %load, knotSpeed: %meanKnotSpeed`).

How to test:

```
void test2()
{
    Vehicle veh1("Cart", 325);
    Car car1("lada", 700, 70);
    Truck truck1("Iveco", 2000, 80, 2000);
    Boat boat1("Azimut", 1500, 60);

    Vehicle* carPtr = &car1;
    Vehicle* truckPtr = &truck1;
    Vehicle* boatPtr = &boat1;

    std::cout << veh1.getInfo() << '\n';
    std::cout << car1.getInfo() << '\n';
    std::cout << truck1.getInfo() << '\n';
    std::cout << boat1.getInfo() << '\n';
    std::cout << carPtr->getInfo() << '\n';
    std::cout << truckPtr->getInfo() << '\n';
    std::cout << boatPtr->getInfo() << '\n';
}
```

Output:

```
name: Cart, load: 325.000000
name: lada, load: 700.000000, speed: 70.000000
name: Iveco, load: 4000.000000, speed: 80.000000
name: Azimut, load: 1500.000000, knotSpeed: 60.000000
name: lada, load: 700.000000, speed: 70.000000
name: Iveco, load: 4000.000000, speed: 80.000000
name: Azimut, load: 1500.000000, knotSpeed: 60.000000
```

Task 3

Implement the static function `knot2Speed` for `Boat` class which convert knots speed to regular km/h. The function returns `double` and takes `double` as an argument.

Also, implement **getter** method for meanKnotSpeed, lets call it `getKnotSpeed()`.

> **FYI:** 1 knot = 1.852 km/h and you can find KNOT_MULTIPLIER into `vehicle.hpp` file as a global variable.

How to test:

```
void test3()
{
    Boat boat1("Reka", 500);
    std::cout << Boat::knot2Speed(30) << '\n';
    std::cout << boat1.knot2Speed(boat1.getKnotSpeed()) << '\n';
}
```

Output:

```
55.56
55.56
```

Task 4

Sudden business requirements

Unexpected regulatory requirements appeared that all vehicles must be certified as *Landcraft* or *Seacraft* and also has a mean speed in km/h.

You decided to make `Vehicle` class as **abstract** and add pure virtual methods `getType()` which returns "Landcraft" for cars and trucks and "Seacraft" for boats and `getSpeed` which returns speed in km/h.

Enter pure virtual method `getType()` which returns std::string, `getSpeed` which returns double and implement overloadings.

How to test:

```
void test4()
{
    Car car1("lada", 700, 70);
    Truck truck1("Mercedes", 1000);
    Boat boat1("Reka", 500);

    Vehicle* carPtr = &car1;
    Vehicle* truckPtr = &truck1;
    Vehicle* boatPtr = &boat1;
```

```
    std::cout << carPtr->getType() << " " << carPtr->getSpeed() << '\n';
    std::cout << truckPtr->getType() << " " << truckPtr->getSpeed() <<
'\n';
    std::cout << boatPtr->getType() << " " << boatPtr->getSpeed() << '\n';
}
```

Output:

```
Landcraft 70
Landcraft 60
Seacraft 55.56
```

> Important note, due to changes in class type previous tests may stop working.

Task 5

Other business requirements

You should store your vehicles in some storage. Lets implement the class `Warehouse` with your vehicles.

`Warehouse` private field:

- vehicles: std::vector<Vehicle*>

`Warehouse` methods:

- default constructor (it means that you don't need to implement it);
- `addVehicle(Vehicle*)` adds a vehicle pointer into Warehouse;
- `getTotalLoad()` returns total possible load for your company;
- overloading `operator[]` returns pointer to i-th element from Warehouse and item can not be modified. If i-th element is not exist throw std::out_of_range with message `%i is a wrong index`;
- overloading `operator<<` returns the ostream with `getInfo()` vehicles. The output should be sorted descending by `total load`, if the loads equal descending by `meanSpeed` in km/h, if the loads and meanSpeeds equal by ascending name. After returning vehicles the order in the container should not be changed.

How to test:

```
void test5()
{
    Warehouse w;
    Truck truck1("Mercedes", 1000, 60, 6000);
    Truck truck2("NotMercedes", 7000);
    Boat boat1("Reka", 500, 50);
    Boat boat2("Azimut", 500, 60);
    Boat boat3("Kazanka", 500, 50);
    Car car1("Lada", 500, 112);
```

```
    Car car2("Renault", 500, 90);
    Car car3("KIA", 600, 60);
    w.addVehicle(&truck1);
    w.addVehicle(&truck2);
    w.addVehicle(&boat1);
    w.addVehicle(&boat2);
    w.addVehicle(&boat3);
    w.addVehicle(&car1);
    w.addVehicle(&car2);
    w.addVehicle(&car3);

    // get total load
    std::cout << "Total cargo is " <<  w.getTotalLoad() << '\n';
    std::cout << "----------------" << '\n';
    // if there is a wrong index
    try
    {
        const Vehicle* ptr = w[100];
    } catch (std::out_of_range& error)
    {
        std::cout << error.what() << '\n';
    }
    std::cout << "----------------" << '\n';

    // check 0-th item
    std::cout << w[0]->getInfo() << '\n';
    std::cout << "----------------" << '\n';

    // print warehouse
    std::cout << w;
    std::cout << "----------------" << '\n';

    // check 0-th item not changed
    std::cout << w[0]->getInfo() << '\n';
}
```

Output:

```
Total cargo is 17100
----------------
100 is a wrong index
----------------
name: Mercedes, load: 7000.000000, speed: 60.000000
----------------
name: NotMercedes, load: 7000.000000, speed: 60.000000
name: Mercedes, load: 7000.000000, speed: 60.000000
name: KIA, load: 600.000000, speed: 60.000000
name: Lada, load: 500.000000, speed: 112.000000
name: Azimut, load: 500.000000, knotSpeed: 60.000000
name: Kazanka, load: 500.000000, knotSpeed: 50.000000
name: Reka, load: 500.000000, knotSpeed: 50.000000
name: Renault, load: 500.000000, speed: 90.000000
```

```
----------------
name: Mercedes, load: 7000.000000, speed: 60.000000
```