ИСТ-АДМО-202Б

Трубин В.Д.

Орлов К.О.

3241.406215.000 ПЗ

Геоинформационных систем

Языки программирования

Уфа – 2025

Основы проектирования классов. Наследование.

Содержание

[Введение 3](#_Toc198797429)

[1. Задание 1 4](#_Toc198797430)

[2. Задание 2 7](#_Toc198797431)

[3. Задание 3 10](#_Toc198797432)

[Заключение 13](#_Toc198797433)

[Список литературы 14](#_Toc198797434)

Введение

Цель практических занятий: закрепление и углубление знаний, полученных в теоретической части дисциплины, развитие навыков объектно-ориентированной технологии проектирования алгоритмов и программ, развитие творческих способностей в области программирования.

# Задание 1

Описать родительский класс для объектов в соответствии с вариантом задания. Для создаваемого класса предусмотреть четкое выполнение принципов Инкапсуляции (все данные объекта недоступны для прямого обращения). Все данные задаются и выводятся посредством вызова интерфейсных методов.

#include <iostream>

#include <string>

class Car {

private:

std::string brand;

double curbWeight;

double payloadCapacity;

double fuelTankVolume;

double mileage;

double fuelPercentage;

public:

// Default constructor

Car() {

brand = "";

curbWeight = 0.0;

payloadCapacity = 0.0;

fuelTankVolume = 0.0;

mileage = 0.0;

fuelPercentage = 0.0;

}

// Setters

void setBrand(const std::string& b) {

brand = b;

}

void setCurbWeight(double w) {

if (w >= 0)

curbWeight = w;

}

void setPayloadCapacity(double p) {

if (p >= 0)

payloadCapacity = p;

}

void setFuelTankVolume(double v) {

if (v >= 0)

fuelTankVolume = v;

}

void setMileage(double m) {

if (m >= 0)

mileage = m;

}

void setFuelPercentage(double percent) {

if (percent >= 0 && percent <= 100)

fuelPercentage = percent;

}

// Add mileage

void addMileage(double km) {

if (km > 0)

mileage += km;

}

// Refuel (increase percentage)

void refuel(double percent) {

if (percent > 0) {

fuelPercentage += percent;

if (fuelPercentage > 100)

fuelPercentage = 100;

}

}

// Getters

std::string getBrand() const {

return brand;

}

double getCurbWeight() const {

return curbWeight;

}

double getPayloadCapacity() const {

return payloadCapacity;

}

double getMileage() const {

return mileage;

}

// Calculate remaining fuel in liters

double getRemainingFuelLiters() const {

return (fuelTankVolume \* fuelPercentage) / 100.0;

}

};

int main() {

Car truck;

// Set initial data

truck.setBrand("KAMAZ");

truck.setCurbWeight(8000); // kg

truck.setPayloadCapacity(12000); // kg

truck.setFuelTankVolume(350); // liters

truck.setMileage(50000); // km

truck.setFuelPercentage(40); // %

// Update values

truck.addMileage(150); // Add 150 km

truck.refuel(30); // Refuel by 30%

// Display information

std::cout << "=== Vehicle Information ===" << std::endl;

std::cout << "Brand: " << truck.getBrand() << std::endl;

std::cout << "Curb weight: " << truck.getCurbWeight() << " kg" << std::endl;

std::cout << "Payload capacity: " << truck.getPayloadCapacity() << " kg" << std::endl;

std::cout << "Mileage: " << truck.getMileage() << " km" << std::endl;

std::cout << "Remaining fuel: " << truck.getRemainingFuelLiters() << " liters" << std::endl;

return 0;

}

# Задание 2

Описать производный (дочерний) класс объектов в соответствии с вариантом. В производном классе предусмотреть описание дополнительных атрибутов, а также методов, которые необходимы для выполнения операций над новым типом объектов с преемственностью к методам родительского класса, т.е. методы, доступные классу родителю и не меняющие алгоритм работы не должны переопределяться заново, в то же время методы объектов класса наследника, не доступные объектам класса родителя должны быть определены.

#include <iostream>

#include <string>

#include <iomanip>

// === Base Class ===

class Car {

private:

std::string brand;

double curbWeight;

double payloadCapacity;

double fuelTankVolume;

double mileage;

double fuelPercentage;

public:

Car() {

brand = "";

curbWeight = 0.0;

payloadCapacity = 0.0;

fuelTankVolume = 0.0;

mileage = 0.0;

fuelPercentage = 0.0;

}

void setBrand(const std::string& b) {

brand = b;

}

void setCurbWeight(double w) {

if (w >= 0)

curbWeight = w;

}

void setPayloadCapacity(double p) {

if (p >= 0)

payloadCapacity = p;

}

void setFuelTankVolume(double v) {

if (v >= 0)

fuelTankVolume = v;

}

void setMileage(double m) {

if (m >= 0)

mileage = m;

}

void setFuelPercentage(double percent) {

if (percent >= 0 && percent <= 100)

fuelPercentage = percent;

}

void addMileage(double km) {

if (km > 0)

mileage += km;

}

void refuel(double percent) {

if (percent > 0) {

fuelPercentage += percent;

if (fuelPercentage > 100)

fuelPercentage = 100;

}

}

std::string getBrand() const {

return brand;

}

double getCurbWeight() const {

return curbWeight;

}

double getPayloadCapacity() const {

return payloadCapacity;

}

double getMileage() const {

return mileage;

}

double getFuelTankVolume() const {

return fuelTankVolume;

}

double getFuelPercentage() const {

return fuelPercentage;

}

double getRemainingFuelLiters() const {

return (fuelTankVolume \* fuelPercentage) / 100.0;

}

};

// === Derived Class ===

class Truck : public Car {

private:

std::string modelName;

std::string inventoryNumber;

double standardFuelConsumption; // liters per ton-kilometer

double cargoTonnokm; // ton-kilometers

double fuelUsed; // total fuel used (liters)

public:

// Default constructor

Truck() {

modelName = "";

inventoryNumber = "";

standardFuelConsumption = 0.0;

cargoTonnokm = 0.0;

fuelUsed = 0.0;

}

// Setters for unique fields

void setModelName(const std::string& name) {

modelName = name;

}

void setInventoryNumber(const std::string& number) {

inventoryNumber = number;

}

void setStandardFuelConsumption(double sfc) {

if (sfc >= 0)

standardFuelConsumption = sfc;

}

// Transport cargo

void transportCargo(double tonnokm) {

if (tonnokm > 0) {

cargoTonnokm += tonnokm;

double fuelForTrip = tonnokm \* standardFuelConsumption;

fuelUsed += fuelForTrip;

// Reduce fuel percentage based on how much fuel was used

double currentFuel = getRemainingFuelLiters();

double newFuel = currentFuel - fuelForTrip;

if (newFuel < 0) newFuel = 0;

double newFuelPercent = (newFuel / getFuelTankVolume()) \* 100;

setFuelPercentage(newFuelPercent);

}

}

// Refuel (liters)

void refuelLiters(double liters) {

double currentFuel = getRemainingFuelLiters();

double total = currentFuel + liters;

if (total > getFuelTankVolume())

total = getFuelTankVolume();

double newPercent = (total / getFuelTankVolume()) \* 100;

setFuelPercentage(newPercent);

}

// Get full vehicle info

void getFullInfo() const {

std::cout << "=== Truck Full Info ===" << std::endl;

std::cout << "Model name: " << modelName << std::endl;

std::cout << "Inventory number: " << inventoryNumber << std::endl;

std::cout << "Brand: " << getBrand() << std::endl;

std::cout << "Curb weight: " << getCurbWeight() << " kg" << std::endl;

std::cout << "Payload capacity: " << getPayloadCapacity() << " kg" << std::endl;

std::cout << "Mileage: " << getMileage() << " km" << std::endl;

std::cout << "Fuel tank: " << getFuelTankVolume() << " liters" << std::endl;

std::cout << "Fuel left: " << getRemainingFuelLiters() << " liters" << std::endl;

std::cout << "Standard fuel consumption: " << standardFuelConsumption << " l/ton-km" << std::endl;

std::cout << "Total fuel used: " << fuelUsed << " liters" << std::endl;

std::cout << "Total transported cargo: " << cargoTonnokm << " ton-kilometers" << std::endl;

}

// Get total transported cargo

double getTotalCargoTonnokm() const {

return cargoTonnokm;

}

// Estimate remaining range (in ton-kilometers)

double getRemainingRange() const {

double remainingLiters = getRemainingFuelLiters();

if (standardFuelConsumption == 0)

return 0;

return remainingLiters / standardFuelConsumption;

}

};

int main() {

Truck truck;

// Set base and unique fields

truck.setBrand("KAMAZ");

truck.setModelName("KAMAZ-65117");

truck.setInventoryNumber("INV-10245");

truck.setCurbWeight(8000);

truck.setPayloadCapacity(12000);

truck.setFuelTankVolume(350);

truck.setMileage(100000);

truck.setFuelPercentage(60);

truck.setStandardFuelConsumption(0.3); // l/ton-km

// Transport some cargo

truck.transportCargo(200); // 200 ton-kilometers

// Refuel with 50 liters

truck.refuelLiters(50);

// Show full info

truck.getFullInfo();

// Show cargo and remaining range

std::cout << "\nTotal transported cargo: " << truck.getTotalCargoTonnokm() << " ton-km" << std::endl;

std::cout << "Estimated remaining range: " << truck.getRemainingRange() << " ton-km" << std::endl;

return 0;

}

Заключение

Входе данной практической работы были изучены основы наследования и иерархии классов в объектно-ориентированной парадигме программирования. В результате работы была создана программа демонстрирующая принцип наследования на примере двух классов Car (родительский класс) и Truck (дочерний класс). Класс Truck заимствует все публичные методы класса Car.

Список литературы

1. Кубенский, А. А. Функциональное программирование : учебник и практикум для академического бакалавриата / А. А. Кубенский. — М. : Издательство Юрайт, 2019. — 348 с.
2. Трофимов, В. В. Основы алгоритмизации и программирования : учебник для СПО / В. В. Трофимов, Т. А. Павловская ; под ред. В. В. Трофимова. — М. : Издательство Юрайт, 2019. — 137 с