**Листинг кода**

**Класс Сar**

class Car {

friend class Database;

protected:

int id; // ID

bool deleteMark; // метка, удалена ли машина

wchar\_t brand[30] = {L"\0"}, model[30] = {L"\0"}, country[30] = {L"\0"}, bodyType[30] = {L"\0"};

int manufacturedYear; // год производства

int numberOfSeats; // количество мест

float maxSpeed; // максимальная скорость

struct interior {

wchar\_t color[30] = {L"\0"}, material[30] = {L"\0"};

} interior;

wchar\_t color[30] = {L"\0"};

int costPerDay; // стоимость аренды

public:

Car();

virtual ~Car() = default;

virtual void SetInfo();

bool IsDelete() { return deleteMark; } // при удалении цена аренды 0

void DeleteCar() { deleteMark = true; costPerDay = 0;}

int GetID() { return id; };

int GetManufacturedYear() { return manufacturedYear; };

wchar\_t\* GetBrand() { return brand; };

wchar\_t\* GetModel() { return model; };

wchar\_t\* GetCountry() { return country; };

wchar\_t\* GetBodyType() { return bodyType; };

int GetNumberOfSeats() { return numberOfSeats; };

float GetMaxSpeed() { return maxSpeed; };

wchar\_t\* GetInteriorColor() { return interior.color; };

wchar\_t\* GetInteriorMaterial() { return interior.material; };

wchar\_t\* GetColor() { return color; };

int GetCostPerDay() { return costPerDay; };

};

Car :: Car() {

id = 0; // присваем ID

deleteMark = false;

manufacturedYear = 0;

numberOfSeats = 0;

maxSpeed = 0;

costPerDay = 0;

}

void Car :: SetInfo() {

int choice;

id = IdGenerator::GetIdForCar(); // присваем ID

CarBrands carinfo;

carinfo.GetCarBrands(); // выводит список марок

cout << "Выберите марку автомобиля: " << endl; // выбор

while (1) {

try {

cin >> choice;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (choice < 1 || choice > 30)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

wcscpy(brand, carinfo.GetBrand(choice).c\_str());

wcscpy(country, carinfo.GetCountry(choice).c\_str());

cout << "Модель автомобиля: " << endl;

fflush(stdin);

while (1) {

try {

fgetws(model, 30, stdin);

model[wcscspn(model, L"\n")] = L'\0';

if (model[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(model, L" ") < wcscspn(model, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

model[0] = towupper(model[0]);

cout << "Тип кузова: " << endl;

fflush(stdin);

while (1) {

try {

fgetws(bodyType, 30, stdin);

bodyType[wcscspn(bodyType, L"\n")] = L'\0';

if (bodyType[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(bodyType, L" ") < wcscspn(bodyType, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

bodyType[0] = towupper(bodyType[0]);

cout << "Год выпуска автомобиля: " << endl;

while (1) {

try {

cin >> manufacturedYear;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (manufacturedYear < 1930 || manufacturedYear > 2022)

throw Exception("Неверное значение");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

cout << "Количество мест: " << endl;

while (1) {

try {

cin >> numberOfSeats;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (numberOfSeats < 2 || numberOfSeats > 7)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

cout << "Материал салона" << endl;

fflush(stdin);

while (1) {

try {

fgetws(interior.material, 30, stdin);

interior.material[wcscspn(interior.material, L"\n")] = L'\0';

if (interior.material[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(interior.material, L" ") < wcscspn(interior.material, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

interior.material[0] = towupper(interior.material[0]);

cout << "Цвет салона" << endl;

fflush(stdin);

while (1) {

try {

fgetws(interior.color, 30, stdin);

interior.color[wcscspn(interior.color, L"\n")] = L'\0';

if (interior.color[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(interior.color, L" ") < wcscspn(interior.color, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

interior.color[0] = towupper(interior.color[0]);

cout << "Цвет автомобиля" << endl;

fflush(stdin);

while (1) {

try {

fgetws(color, 30, stdin);

color[wcscspn(color, L"\n")] = L'\0';

if (color[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(color, L" ") < wcscspn(color, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

color[0] = towupper(color[0]);

cout << "Максимальная скорость: " << endl;

while (1) {

try {

cin >> maxSpeed;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (maxSpeed < 10 || maxSpeed > 500)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

cout << "Стоимость аренды на день: " << endl;

while (1) {

try {

cin >> costPerDay;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (costPerDay < 0)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

}

**Класс OilCar**

class OilCar : virtual public Car

{

friend class Database;

protected:

float consumption; // расход

float volume; // объем

public:

OilCar();

virtual ~OilCar() = default;

void SetInfo();

float GetConsumption() { return consumption; };

float GetVolume() { return volume; };

};

OilCar :: OilCar () : Car() {

volume = 0.0;

consumption = 0.0;

}

void OilCar :: SetInfo() {

Car::SetInfo();

cout << "Объем двигателя: " << endl;

while (1) {

try {

cin >> volume;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (volume < 0 || volume > 8.5)

throw Exception("Введено неверное значение");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

cout << "Расход топлива на 100км: " << endl;

while (1) {

try {

cin >> consumption;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (consumption < 0)

throw Exception("Введено неверное значение");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

}

**Класс ElectricCar**

class ElectricCar : virtual public Car

{

friend class Database;

protected:

int batteryCapacity;

public:

ElectricCar();

virtual ~ElectricCar() = default;

void SetInfo();

int GetBatteryCapacity() { return batteryCapacity; };

};

ElectricCar :: ElectricCar() {

batteryCapacity = 0;

}

void ElectricCar :: SetInfo() {

Car::SetInfo();

cout << "Объем батареи: " << endl;

while (1) {

try {

cin >> batteryCapacity;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (batteryCapacity < 0)

throw Exception("Введено неверное значение");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

}

**Класс PetrolCar**

class PetrolCar : public OilCar

{

friend class Database;

protected:

int petrolType;

public:

PetrolCar();

virtual ~PetrolCar() = default;

void SetInfo();

int GetPetrolType() { return petrolType; };

};

PetrolCar :: PetrolCar() : OilCar() {

petrolType = 0;

}

void PetrolCar :: SetInfo() {

OilCar::SetInfo();

int choice;

cout << "Марка бензина: " << endl;

cout << "1 - 80\n"

"2 - 92\n"

"3 - 95\n"

"4 - 98\n";

while (1) {

try {

cin >> choice;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (choice < 1 || choice > 4)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

switch (choice) {

case 1: petrolType = 80; break;

case 2: petrolType = 92; break;

case 3: petrolType = 95; break;

case 4: petrolType = 98; break;

}

}

**Класс DieselCar**

class DieselCar : public OilCar

{

friend class Database;

public:

DieselCar() { };

~DieselCar() = default;

void SetInfo() { OilCar::SetInfo(); };

};

**Класс HybridCar**

class HybridCar : public ElectricCar, public OilCar

{

friend class Database;

protected:

wchar\_t fuelType[8] = {L"\0"};

public:

HybridCar() : Car(), OilCar(), ElectricCar() { }

~HybridCar() = default;

void SetInfo();

wchar\_t\* GetFuelType() { return fuelType; };

};

void HybridCar :: SetInfo() {

Car::SetInfo();

int choice;

cout << "Тип топлива: " << endl;

cout << "1 - Бензин\n"

"2 - Дизель\n";

while (1) {

try {

cin >> choice;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (choice > 2 || choice < 1)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

switch (choice) {

case 1: wcscpy(fuelType, L"Бензин\0"); break;

case 2: wcscpy(fuelType, L"Дизель\0"); break;

}

cout << "Объем двигателя: " << endl;

while (1) {

try {

cin >> volume;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (volume < 0 || volume > 8.5)

throw Exception("Введено неверное значение");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

cout << "Расход топлива на 100км: " << endl;

while (1) {

try {

cin >> consumption;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (consumption < 0)

throw Exception("Введено неверное значение");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

cout << "Объем батареи: " << endl;

while (1) {

try {

cin >> batteryCapacity;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (batteryCapacity < 0)

throw Exception("Введено неверное значение");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

}

**Класс CarBrands**

struct CarBrands

{

friend class Car;

private:

int num;

wchar\_t brand[30] = {L"\0"};

wchar\_t country[30] = {L"\0"};

CarBrands() = default;

~CarBrands() = default;

void GetCarBrands();

wstring GetCountry(int);

wstring GetBrand(int);

};

void CarBrands :: GetCarBrands() {

ifstream file("/Users/max/Desktop/CarRent/Files/CarBrands.bin", ios::binary);

if (!file.is\_open())

cout << "Error";

cout << left << setw(15) << "Номер" << setw(20) << "Марка" << setw(20) << "Страна" << endl;

while (file.read((char \*) this, sizeof(CarBrands))) {

wcout << left << setw(10) << this->num << setw(15) << this->brand << setw(20) << this->country << endl;

}

file.close();

}

wstring CarBrands :: GetBrand(int choice) {

ifstream file("../Files/CarBrands.bin", ios::binary);

wstring brandTmp;

if (!file.is\_open())

cout << "Error";

else {

while (file.read((char \*) this, sizeof(CarBrands))) {

if (this->num == choice) {

brandTmp = this->brand;

return brandTmp;

}

}

}

}

wstring CarBrands :: GetCountry(int choice) {

ifstream file("../Files/CarBrands.bin", ios::binary);

wstring countryTmp;

if (!file.is\_open())

cout << "Error";

else {

while (file.read((char \*) this, sizeof(CarBrands))) {

if (this->num == choice) {

countryTmp = this->country;

return countryTmp;

}

}

}

}

**Класс Database**

class Database

{

public:

static void AddInDatabase(DieselCar&);

static void AddInDatabase(PetrolCar&);

static void AddInDatabase(ElectricCar&);

static void AddInDatabase(HybridCar&);

static void GetFromDatabase(vector<DieselCar>&);

static void GetFromDatabase(vector<PetrolCar>&);

static void GetFromDatabase(vector<ElectricCar>&);

static void GetFromDatabase(vector<HybridCar>&);

static void DeleteFromDatabase(DieselCar&);

static void DeleteFromDatabase(PetrolCar&);

static void DeleteFromDatabase(ElectricCar&);

static void DeleteFromDatabase(HybridCar&);

};

void Database :: AddInDatabase(DieselCar &obj) {

struct Data {

int id;

bool deleteMark;

wchar\_t brand[30], model[30], country[30], bodyType[30];

int manufacturedYear;

int numberOfSeats;

float maxSpeed;

float consumption;

float volume;

struct interior {

wchar\_t color[30], material[30];

} interior;

wchar\_t color[30];

int costPerDay;

} data;

data.id = obj.id;

data.deleteMark = obj.deleteMark;

wcscpy(data.brand, obj.brand);

wcscpy(data.model, obj.model);

wcscpy(data.country, obj.country);

wcscpy(data.bodyType, obj.bodyType);

data.manufacturedYear = obj.manufacturedYear;

data.numberOfSeats = obj.numberOfSeats;

data.maxSpeed = obj.maxSpeed;

data.consumption = obj.consumption;

data.volume = obj.volume;

wcscpy(data.interior.color, obj.interior.color);

wcscpy(data.interior.material, obj.interior.material);

wcscpy(data.color, obj.color);

data.costPerDay = obj.costPerDay;

ofstream file;

file.open("/Users/max/Desktop/CarRent/Files/DieselCars.bin", ios::binary | ios::app);

if (!file.is\_open())

cout << "Error";

file.write((char \*) &data, sizeof(data));

file.close();

}

void Database :: GetFromDatabase(vector<DieselCar>&dieselCars) {

DieselCar obj;

struct Data {

int id;

bool deleteMark;

wchar\_t brand[30], model[30], country[30], bodyType[30];

int manufacturedYear;

int numberOfSeats;

float maxSpeed;

float consumption;

float volume;

struct interior {

wchar\_t color[30], material[30];

} interior;

wchar\_t color[30];

int costPerDay;

} data;

ifstream file;

file.open("/Users/max/Desktop/CarRent/Files/DieselCars.bin", ios::binary);

if (!file.is\_open())

cout << "Error";

while (file.read((char \*) &data, sizeof(data))) {

obj.deleteMark = data.deleteMark;

obj.id = data.id;

wcscpy(obj.brand, data.brand);

wcscpy(obj.model, data.model);

wcscpy(obj.country, data.country);

wcscpy(obj.bodyType, data.bodyType);

obj.manufacturedYear = data.manufacturedYear;

obj.numberOfSeats = data.numberOfSeats;

obj.maxSpeed = data.maxSpeed;

obj.consumption = data.consumption;

obj.volume = data.volume;

wcscpy(obj.interior.color, data.interior.color);

wcscpy(obj.interior.material, data.interior.material);

wcscpy(obj.color, data.color);

obj.costPerDay = data.costPerDay;

dieselCars.emplace\_back(obj);

}

file.close();

}

void Database :: AddInDatabase(PetrolCar &obj) {

struct Data {

int id;

bool deleteMark;

wchar\_t brand[30], model[30], country[30], bodyType[30];

int manufacturedYear;

int numberOfSeats;

float maxSpeed;

float consumption;

float volume;

struct interior {

wchar\_t color[30], material[30];

} interior;

wchar\_t color[30];

int costPerDay;

int petrolType;

} data;

data.id = obj.id;

data.deleteMark = obj.deleteMark;

wcscpy(data.brand, obj.brand);

wcscpy(data.model, obj.model);

wcscpy(data.country, obj.country);

wcscpy(data.bodyType, obj.bodyType);

data.manufacturedYear = obj.manufacturedYear;

data.numberOfSeats = obj.numberOfSeats;

data.maxSpeed = obj.maxSpeed;

data.consumption = obj.consumption;

data.volume = obj.volume;

wcscpy(data.interior.color, obj.interior.color);

wcscpy(data.interior.material, obj.interior.material);

wcscpy(data.color, obj.color);

data.costPerDay = obj.costPerDay;

data.petrolType = obj.petrolType;

ofstream file;

file.open("/Users/max/Desktop/CarRent/Files/PetrolCars.bin", ios::binary | ios::app);

if (!file.is\_open())

cout << "Error";

file.write((char \*) &data, sizeof(data));

file.close();

}

void Database :: GetFromDatabase(vector<PetrolCar>&petrolCars) {

PetrolCar obj;

struct Data {

int id;

bool deleteMark;

wchar\_t brand[30], model[30], country[30], bodyType[30];

int manufacturedYear;

int numberOfSeats;

float maxSpeed;

float consumption;

float volume;

struct interior {

wchar\_t color[30], material[30];

} interior;

wchar\_t color[30];

int costPerDay;

int petrolType;

} data;

ifstream file;

file.open("/Users/max/Desktop/CarRent/Files/PetrolCars.bin", ios::binary);

if (!file.is\_open())

cout << "Error";

while (file.read((char \*) &data, sizeof(data))) {

obj.deleteMark = data.deleteMark;

obj.id = data.id;

wcscpy(obj.brand, data.brand);

wcscpy(obj.model, data.model);

wcscpy(obj.country, data.country);

wcscpy(obj.bodyType, data.bodyType);

obj.manufacturedYear = data.manufacturedYear;

obj.numberOfSeats = data.numberOfSeats;

obj.maxSpeed = data.maxSpeed;

obj.consumption = data.consumption;

obj.volume = data.volume;

wcscpy(obj.interior.color, data.interior.color);

wcscpy(obj.interior.material, data.interior.material);

wcscpy(obj.color, data.color);

obj.costPerDay = data.costPerDay;

obj.petrolType = data.petrolType;

petrolCars.emplace\_back(obj);

}

file.close();

}

void Database :: AddInDatabase(ElectricCar &obj) {

struct Data {

int id;

bool deleteMark;

wchar\_t brand[30], model[30], country[30], bodyType[30];

int manufacturedYear;

int numberOfSeats;

float maxSpeed;

struct interior {

wchar\_t color[30], material[30];

} interior;

wchar\_t color[30];

int costPerDay;

int batteryCapacity;

} data;

data.id = obj.id;

data.deleteMark = obj.deleteMark;

wcscpy(data.brand, obj.brand);

wcscpy(data.model, obj.model);

wcscpy(data.country, obj.country);

wcscpy(data.bodyType, obj.bodyType);

data.manufacturedYear = obj.manufacturedYear;

data.numberOfSeats = obj.numberOfSeats;

data.maxSpeed = obj.maxSpeed;

wcscpy(data.interior.color, obj.interior.color);

wcscpy(data.interior.material, obj.interior.material);

wcscpy(data.color, obj.color);

data.costPerDay = obj.costPerDay;

data.batteryCapacity = obj.batteryCapacity;

ofstream file;

file.open("/Users/max/Desktop/CarRent/Files/ElectricCars.bin", ios::binary | ios::app);

if (!file.is\_open())

cout << "Error";

file.write((char \*) &data, sizeof(data));

file.close();

}

void Database :: GetFromDatabase(vector<ElectricCar>&electricCars) {

ElectricCar obj;

struct Data {

int id;

bool deleteMark;

wchar\_t brand[30], model[30], country[30], bodyType[30];

int manufacturedYear;

int numberOfSeats;

float maxSpeed;

struct interior {

wchar\_t color[30], material[30];

} interior;

wchar\_t color[30];

int costPerDay;

int batteryCapacity;

} data;

ifstream file;

file.open("/Users/max/Desktop/CarRent/Files/ElectricCars.bin", ios::binary);

if (!file.is\_open())

cout << "Error";

while (file.read((char \*) &data, sizeof(data))) {

obj.deleteMark = data.deleteMark;

obj.id = data.id;

wcscpy(obj.brand, data.brand);

wcscpy(obj.model, data.model);

wcscpy(obj.country, data.country);

wcscpy(obj.bodyType, data.bodyType);

obj.manufacturedYear = data.manufacturedYear;

obj.numberOfSeats = data.numberOfSeats;

obj.maxSpeed = data.maxSpeed;

wcscpy(obj.interior.color, data.interior.color);

wcscpy(obj.interior.material, data.interior.material);

wcscpy(obj.color, data.color);

obj.costPerDay = data.costPerDay;

obj.batteryCapacity = data.batteryCapacity;

electricCars.emplace\_back(obj);

}

file.close();

}

void Database :: AddInDatabase(HybridCar &obj) {

struct Data {

int id;

bool deleteMark;

wchar\_t brand[30], model[30], country[30], bodyType[30];

int manufacturedYear;

int numberOfSeats;

float maxSpeed;

float consumption;

float volume;

struct interior {

wchar\_t color[30], material[30];

} interior;

wchar\_t color[30];

int costPerDay;

int batteryCapacity;

wchar\_t fuelType[8];

} data;

data.id = obj.id;

data.deleteMark = obj.deleteMark;

wcscpy(data.brand, obj.brand);

wcscpy(data.model, obj.model);

wcscpy(data.country, obj.country);

wcscpy(data.bodyType, obj.bodyType);

data.manufacturedYear = obj.manufacturedYear;

data.numberOfSeats = obj.numberOfSeats;

data.maxSpeed = obj.maxSpeed;

data.consumption = obj.consumption;

data.volume = obj.volume;

wcscpy(data.interior.color, obj.interior.color);

wcscpy(data.interior.material, obj.interior.material);

wcscpy(data.color, obj.color);

data.costPerDay = obj.costPerDay;

data.batteryCapacity = obj.batteryCapacity;

wcscpy(data.fuelType, obj.fuelType);

ofstream file;

file.open("/Users/max/Desktop/CarRent/Files/HybridCars.bin", ios::binary | ios::app);

if (!file.is\_open())

cout << "Error";

file.write((char \*) &data, sizeof(data));

file.close();

}

void Database :: GetFromDatabase(vector<HybridCar>&hybridCars) {

HybridCar obj;

struct Data {

int id;

bool deleteMark;

wchar\_t brand[30], model[30], country[30], bodyType[30];

int manufacturedYear;

int numberOfSeats;

float maxSpeed;

float consumption;

float volume;

struct interior {

wchar\_t color[30], material[30];

} interior;

wchar\_t color[30];

int costPerDay;

int batteryCapacity;

wchar\_t fuelType[8];

} data;

ifstream file;

file.open("/Users/max/Desktop/CarRent/Files/HybridCars.bin", ios::binary);

if (!file.is\_open())

cout << "Error";

while (file.read((char \*) &data, sizeof(data))) {

obj.deleteMark = data.deleteMark;

obj.id = data.id;

wcscpy(obj.brand, data.brand);

wcscpy(obj.model, data.model);

wcscpy(obj.country, data.country);

wcscpy(obj.bodyType, data.bodyType);

obj.manufacturedYear = data.manufacturedYear;

obj.numberOfSeats = data.numberOfSeats;

obj.maxSpeed = data.maxSpeed;

obj.consumption = data.consumption;

obj.volume = data.volume;

wcscpy(obj.interior.color, data.interior.color);

wcscpy(obj.interior.material, data.interior.material);

wcscpy(obj.color, data.color);

obj.costPerDay = data.costPerDay;

obj.batteryCapacity = data.batteryCapacity;

wcscpy(obj.fuelType, data.fuelType);

hybridCars.emplace\_back(obj);

}

file.close();

}

void Database :: DeleteFromDatabase(DieselCar &obj) {

struct Data {

int id;

bool deleteMark;

wchar\_t brand[30], model[30], country[30], bodyType[30];

int manufacturedYear;

int numberOfSeats;

float maxSpeed;

float consumption;

float volume;

struct interior {

wchar\_t color[30], material[30];

} interior;

char color[60];

int costPerDay, costPerWeek, costPerMonth;

} data;

ifstream file1; // для чтения

ofstream file2; // для записи

file1.open("/Users/max/Desktop/CarRent/Files/DieselCars.bin", ios::binary);

if (!file1.is\_open())

cout << "Error";

file2.open("/Users/max/Desktop/CarRent/Files/tmp.bin", ios::binary | ios::app);

while (file1.read((char \*) &data, sizeof(data))) {

if (obj.id == data.id) {

data.deleteMark = obj.deleteMark = true;

file2.write((char \*) &data, sizeof(data));

} else

file2.write((char \*) &data, sizeof(data));

}

file1.close();

remove("/Users/max/Desktop/CarRent/Files/DieselCars.bin");

file2.close();

rename("/Users/max/Desktop/CarRent/Files/tmp.bin", "/Users/max/Desktop/CarRent/Files/DieselCars.bin");

}

void Database :: DeleteFromDatabase(PetrolCar &obj) {

struct Data {

int id;

bool deleteMark;

wchar\_t brand[30], model[30], country[30], bodyType[30];

int manufacturedYear;

int numberOfSeats;

float maxSpeed;

float consumption;

float volume;

struct interior {

wchar\_t color[30], material[30];

} interior;

wchar\_t color[30];

int costPerDay, costPerWeek, costPerMonth;

int petrolType;

} data;

ifstream file1; // для чтения

ofstream file2; // для записи

file1.open("/Users/max/Desktop/CarRent/Files/PetrolCars.bin", ios::binary);

if (!file1.is\_open())

cout << "Error";

file2.open("/Users/max/Desktop/CarRent/Files/tmp.bin", ios::binary | ios::app);

while (file1.read((char \*) &data, sizeof(data))) {

if (obj.id == data.id) {

data.deleteMark = obj.deleteMark = true;

file2.write((char \*) &data, sizeof(data));

} else

file2.write((char \*) &data, sizeof(data));

}

file1.close();

remove("/Users/max/Desktop/CarRent/Files/PetrolCars.bin");

file2.close();

rename("/Users/max/Desktop/CarRent/Files/tmp.bin", "/Users/max/Desktop/CarRent/Files/PetrolCars.bin");

}

void Database :: DeleteFromDatabase(ElectricCar &obj) {

struct Data {

int id;

bool deleteMark;

wchar\_t brand[30], model[30], country[30], bodyType[30];

int manufacturedYear;

int numberOfSeats;

float maxSpeed;

struct interior {

wchar\_t color[30], material[30];

} interior;

char color[60];

int costPerDay, costPerWeek, costPerMonth;

int batteryCapacity;

} data;

ifstream file1; // для чтения

ofstream file2; // для записи

file1.open("/Users/max/Desktop/CarRent/Files/ElectricCars.bin", ios::binary);

if (!file1.is\_open())

cout << "Error";

file2.open("/Users/max/Desktop/CarRent/Files/tmp.bin", ios::binary | ios::app);

while (file1.read((char \*) &data, sizeof(data))) {

if (obj.id == data.id) {

data.deleteMark = obj.deleteMark = true;

file2.write((char \*) &data, sizeof(data));

} else

file2.write((char \*) &data, sizeof(data));

}

file1.close();

remove("/Users/max/Desktop/CarRent/Files/ElectricCars.bin");

file2.close();

rename("/Users/max/Desktop/CarRent/Files/tmp.bin", "/Users/max/Desktop/CarRent/Files/ElectricCars.bin");

}

void Database :: DeleteFromDatabase(HybridCar &obj) {

struct Data {

int id;

bool deleteMark;

wchar\_t brand[30], model[30], country[30], bodyType[30];

int manufacturedYear;

int numberOfSeats;

float maxSpeed;

float consumption;

float volume;

struct interior {

wchar\_t color[30], material[30];

} interior;

char color[60];

int costPerDay, costPerWeek, costPerMonth;

int batteryCapacity;

char fuelType[60];

} data;

ifstream file1; // для чтения

ofstream file2; // для записи

file1.open("/Users/max/Desktop/CarRent/Files/HybridCars.bin", ios::binary);

if (!file1.is\_open())

cout << "Error";

file2.open("/Users/max/Desktop/CarRent/Files/tmp.bin", ios::binary | ios::app);

while (file1.read((char \*) &data, sizeof(data))) {

if (obj.id == data.id) {

data.deleteMark = obj.deleteMark = true;

file2.write((char \*) &data, sizeof(data));

} else

file2.write((char \*) &data, sizeof(data));

}

file1.close();

remove("/Users/max/Desktop/CarRent/Files/HybridCars.bin");

file2.close();

rename("/Users/max/Desktop/CarRent/Files/tmp.bin", "/Users/max/Desktop/CarRent/Files/HybridCars.bin");

}

**Класс Exeption**

class Exception

{

protected:

string message;

public:

Exception() = default;

Exception(string message) {

this->message = message;

}

void Show() {

cout << message << endl;

}

virtual void Fix() {

cin.clear();

fflush(stdin);

this->Show();

cout << "Введите еще раз: ";

}

};

**Класс BadInputException**

class BadInputException : public Exception

{

public:

BadInputException(string message) : Exception() {

this->message = message;

}

};

**Класс OverflowException**

class OverflowException : public Exception

{

public:

OverflowException(string message) : Exception() {

this->message = message;

}

};

**Класс EmptyInputException**

class EmptyInputException : public Exception

{

public:

EmptyInputException(string message) : Exception() {

this->message = message;

}

void Fix() {

fflush(stdin);

this->Show();

cout << "Введите еще раз: ";

}

};

**Класс SpaceException**

class SpaceException : public Exception

{

public:

SpaceException(string message) : Exception() {

this->message = message;

}

};

**Класс IdGenerator**

class IdGenerator

{

friend class Car;

friend class User;

friend class Admin;

private:

static int GetIdForCar();

static int GetIdForPerson();

};

int IdGenerator :: GetIdForCar() {

fstream file; // файл для записи и чтения

file.open("/Users/max/Desktop/CarRent/ID/CarId.bin", ios::binary | ios::in | ios::ate); // открытие файла

int id = 1;

if (file.tellg() == 0) // если файла пустой, то ID = 1

{

ofstream file2("/Users/max/Desktop/CarRent/ID/CarId.bin", ios::binary | ios::app);

file2.write((char \*) &id, sizeof(int));

file2.close();

return id;

}

file.seekp(0, ios::beg); // если существует

file.read((char \*) &id, sizeof(int)); // то считываем значение

file.close(); // закрываем

id++; // прибавляем 1

file.open("/Users/max/Desktop/CarRent/ID/CarId.bin", ios::binary | ios::out);

file.write((char \*) &id, sizeof(int)); // открываем файл для записи и перезаписываем новое значение

file.close();

return id; // возвращаем значение

}

int IdGenerator :: GetIdForPerson() // все абсолютно аналогично

{

fstream file;

file.open("/Users/max/Desktop/CarRent/ID/PersonID.bin", ios::binary | ios::in | ios::ate);

int id = 1;

if (file.tellg() == 0) {

ofstream file2("/Users/max/Desktop/CarRent/ID/PersonID.bin", ios::binary | ios::app);

file2.write((char \*) &id, sizeof(int));

file2.close();

return id;

}

file.seekp(0, ios::beg);

file.read((char \*) &id, sizeof(int));

file.close();

id++;

file.open("/Users/max/Desktop/CarRent/ID/PersonID.bin", ios::binary | ios::out);

file.write((char \*) &id, sizeof(int));

file.close();

return id;

}

**Класс Authentication**

сlass Authentication

{

friend class Interface;

private:

Userdata account;

Authentication() = default;

~Authentication() = default;

bool AdminAuthentication(Admin&); // аутентификация администратора

bool UserAuthentication(User&); // аутентификация пользователя

};

bool Authentication :: AdminAuthentication(Admin& admin) {

wchar\_t login[30], password[30];

int mode = 1;

cout << "Введите логин учетной записи" << endl;

fflush(stdin);

fgetws(login, 30, stdin);

login[wcscspn(login, L"\n")] = L'\0';

cout << "Введите пароль от учетной записи" << endl;

fflush(stdin);

fgetws(password, 30, stdin);

password[wcscspn(password, L"\n")] = L'\0';

account.SetData(login, password);

while (account.CheckAccess(mode) == false) {

char tmp;

fflush(stdin);

cout << "Ошибка! Введите логин и пароль заново" << endl;

cout << "Если хотите выбрать другой режим входа, введите 1\nЕсли хотите продолжить, нажмите любую другую кнопку"

<< endl;

tmp = cin.get();

if (tmp == '1')

return false;

cout << "Логин" << endl;

fflush(stdin);

fgetws(login, 30, stdin);

login[wcscspn(login, L"\n")] = L'\0';

cout << "Пароль" << endl;

fflush(stdin);

fgetws(password, 30, stdin);

password[wcscspn(password, L"\n")] = L'\0';

account.SetData(login, password);

}

cout << "Вход выполнен!" << endl;

ifstream file;

file.open("/Users/max/Desktop/CarRent/Files/Admin.bin", ios::binary);

if (!file.is\_open())

cout << "Error";

while (file.read((char \*) &admin, sizeof(Admin))) {

if (admin.GetID() == account.GetID())

break;

}

file.close();

return true;

}

bool Authentication :: UserAuthentication(User& user)

{

wchar\_t login[30], password[30];

int mode = 2;

cout << "Введите логин учетной записи" << endl;

fflush(stdin);

fgetws(login, 30, stdin);

login[wcscspn(login, L"\n")] = L'\0';

cout << "Введите пароль от учетной записи" << endl;

fflush(stdin);

fgetws(password, 30, stdin);

password[wcscspn(password, L"\n")] = L'\0';

account.SetData(login, password);

while(!account.CheckAccess(mode))

{

char tmp;

fflush(stdin);

cout << "Ошибка! Введите логин и пароль заново" << endl;

cout << "Если хотите выбрать другой режим входа, введите 1\nЕсли хотите продолжить, нажмите любую другую кнопку" << endl;

tmp = cin.get();

if (tmp == '1')

return false;

cout << "Логин" << endl;

fflush(stdin);

fgetws(login, 30, stdin);

login[wcscspn(login, L"\n")] = L'\0';

cout << "Пароль" << endl;

fflush(stdin);

fgetws(password, 30, stdin);

password[wcscspn(password, L"\n")] = L'\0';

account.SetData(login, password);

}

cout << "Вход выполнен!" << endl;

ifstream file;

file.open("/Users/max/Desktop/CarRent/Files/User.bin", ios::binary);

if (!file.is\_open()) {

cout << "Ошибка открытия файла";

return false;

}

while(file.read((char\*)&user, sizeof(User)))

{

if (user.GetID() == account.GetID()){

file.close();

break;

}

}

return true;

};

**Класс CarInUsage**

struct CarInUsage

{

private:

int lastDay;

int carID;

public:

CarInUsage() = default;

CarInUsage(int, int);

~CarInUsage() = default;

void WriteInFile();

void RefreshFile();

void GetCarsInUsage(vector<int>&);

};

CarInUsage :: CarInUsage(int lastDay, int carID) {

this->lastDay = lastDay;

this->carID = carID;

}

void CarInUsage :: WriteInFile() {

ofstream file;

file.open("/Users/max/Desktop/CarRent/Files/CarInUsage.bin", ios::binary | ios::app);

if (!file.is\_open())

cout << "Error";

file.write((char \*) this, sizeof(CarInUsage));

file.close();

this->RefreshFile();

}

void CarInUsage :: RefreshFile() {

int nowDay;

time\_t seconds = time(NULL);

tm \*currentTime = localtime(&seconds);

int a = (14 - currentTime->tm\_mon + 1) / 12;

int y = currentTime->tm\_year + 1900 + 4800 - a;

int m = currentTime->tm\_mon + 1 + 12 \* a - 3;

nowDay = currentTime->tm\_mday + ((153 \* m + 2) / 5) + 365 \* y + y / 4 - y / 100 + y / 400 - 32045;

ifstream file1; // для чтения

ofstream file2; // для записи

file1.open("/Users/max/Desktop/CarRent/Files/CarInUsage.bin", ios::binary);

if (!file1.is\_open())

cout << "Error";

file2.open("/Users/max/Desktop/CarRent/Files/tmp.bin", ios::binary | ios::app);

while (file1.read((char \*) this, sizeof(CarInUsage))) {

if (this->lastDay >= nowDay)

file2.write((char \*) this, sizeof(CarInUsage));

else

continue;

}

file1.close();

remove("/Users/max/Desktop/CarRent/Files/CarInUsage.bin");

file2.close();

rename("/Users/max/Desktop/CarRent/Files/tmp.bin", "/Users/max/Desktop/CarRent/Files/CarInUsage.bin");

}

void CarInUsage :: GetCarsInUsage(vector<int> &carIdInUsage) {

this->RefreshFile();

ifstream file;

file.open("/Users/max/Desktop/CarRent/Files/CarInUsage.bin", ios::binary);

if (!file.is\_open())

cout << "Error";

while (file.read((char \*) this, sizeof(CarInUsage)))

carIdInUsage.push\_back(this->carID);

file.close();

}

**Класс Interface**

class Interface

{

friend class Session;

protected:

vector<PetrolCar> petrolCars; // все авто

vector<DieselCar> dieselCars;

vector<ElectricCar> electricCars;

vector<HybridCar> hybridCars;

vector<int> carIdInUsage; // id машин, которые недоступны

Session newSession;

vector<Session> allSessions;

User user;

Admin admin;

public:

Interface();

~Interface() = default;

void FirstMenu(); // главное меню, открывается при запуске

void UserMainMenu(); // главное меню пользователя

void AdminMainMenu(); // главное меню администратора

void Registration(); // регистрация

void UserAccountInfo(); // выводит информацию об аккауте пользователя

void SeeCars();

void CarOrder(int);

void SeeUserHistory(int);

void ShowDetailedInfo(int, int); // о пользователе

bool ShowCarsForChoosing(int); // выводит автомобили

void AddCount(); // пополнение баланса

int FindCost(int, int); // ищет стоимость аренды по его ID

void RefreshUserInFile(); // записывает обновленные данные пользователя в файл

void SetAccountSettings(); // изменение данных аккаунта

void ShowAllSessions(); // показывает все заказы

void ShowAllUsers(); // показывает всех пользователей

void ShowAllCars();

void ShowSpecificCarSessions();

void ShowSpecificUserSessions();

void DeleteSpecificCar();

void RefreshAllCars();

void AddNewCar();

};

Interface::Interface() {

Database::GetFromDatabase(petrolCars);

Database::GetFromDatabase(dieselCars);

Database::GetFromDatabase(electricCars);

Database::GetFromDatabase(hybridCars);

CarInUsage tmp;

tmp.GetCarsInUsage(carIdInUsage);

Session::GetAllSessions(allSessions);

}

void Interface :: FirstMenu() {

int choice;

while (1) {

cout << "Выберите режим входа: \n"

"1 - Администратор\n"

"2 - Пользователь\n"

"3 - Регистрация нового аккаунта пользователя\n"

"4 - Завершить работу" << endl;

while (1) {

try {

cin >> choice;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (choice < 1 || choice > 4)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

Authentication auth;

switch (choice) {

case 1:

if (auth.AdminAuthentication(this->admin))

this->AdminMainMenu();

break;

case 2:

if (auth.UserAuthentication(this->user))

this->UserMainMenu();

break;

case 3:

this->Registration(); break;

case 4:

exit(0);

}

}

}

void Interface :: Registration() {

user.SetInfo();

}

void Interface :: AdminMainMenu() {

int choice;

while (1) {

cout << "===== Личный кабинет ===== \n"

"1 - Просмотреть базу данных пользователей\n"

"2 - Просмотреть базу данных автомобилей\n"

"3 - Просмотреть все заказы\n"

"4 - Просмотреть все заказы конкретного пользователя\n"

"5 - Просмотреть все заказы конкретного автомобиля\n"

"6 - Добавление автомобиля\n"

"7 - Удаление автомобиля\n"

"8 - Выход из аккаунта\n"

"9 - Завершить работу" << endl;

while (1) {

try {

cin >> choice;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (choice < 1 || choice > 9)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

switch (choice) {

case 1:

this->ShowAllUsers(); break;

case 2:

this->ShowAllCars(); break;

case 3:

this->ShowAllSessions(); break;

case 4:

this->ShowSpecificUserSessions(); break;

case 5:

this->ShowSpecificCarSessions(); break;

case 6:

this->AddNewCar(); break;

case 7:

this->DeleteSpecificCar(); break;

case 8:

this->RefreshAllCars(); return;

case 9:

this->RefreshAllCars(); exit(0);

}

}

}

void Interface :: ShowAllUsers() {

ifstream file;

file.open("/Users/max/Desktop/CarRent/Files/User.bin", ios::binary);

if (!file.is\_open()) // кетч

cout << "Error";

int i = 0;

cout

<< "----------------------------------------------------------------------------------------------------------------------------------"

<< endl;

wcout << '|' << setw(10) << left << L"ID" << '|' << setw(15) << L"Фамилия" << '|' << setw(15) << L"Имя" << '|'

<< setw(10) << L"Возраст" << '|'

<< setw(15) << L"Баланс" << '|' << setw(15) << L"Потрачено" << '|' << setw(10) << L"Статус" << '|' << setw(15)

<< L"Номер телефона" << '|' << setw(15) << L"Номер паспорта" << '|' << endl;

cout

<< "----------------------------------------------------------------------------------------------------------------------------------"

<< endl;

while (file.read((char \*) &user, sizeof(User))) {

cout << '|' << setw(10) << left << user.GetID();

wcout << '|' << setw(15) << left << user.GetName() << '|' << setw(15) << user.GetSurname() << '|' << setw(10)

<< user.GetAge() << '|'

<< setw(15) << user.GetCount() << '|' << setw(15) << user.GetSpendMoney() << '|' << setw(10)

<< user.GetStatus() << '|' << setw(15) << user.GetPhoneNumber() << '|' << setw(15)

<< user.GetPassportNum() << '|' << endl;

cout

<< "----------------------------------------------------------------------------------------------------------------------------------"

<< endl;

}

}

void Interface :: ShowAllSessions() {

ifstream file;

file.open("/Users/max/Desktop/CarRent/Files/User.bin", ios::binary);

if (!file.is\_open())

cout << "Error";

vector<User> users;

while (file.read((char \*) &user, sizeof(User)))

users.push\_back(user);

if (allSessions.size() == 0) {

cout << "Нет заказов!" << endl;

return;

}

int i = 1;

cout

<< "-----------------------------------------------------------------------------------------------------------------------"

<< endl;

wcout << '|' << setw(10) << left << L"№" << '|' << setw(15) << L"Фамилия" << '|' << setw(15) << L"Имя" << '|'

<< setw(15) << L"Марка" << '|' << setw(15) << L"Модель" << '|'

<< setw(15) << L"Дата начала" << '|' << setw(15) << L"Дата окончания" << '|' << setw(10) << L"Стоимость"

<< '|' << endl;

cout

<< "-----------------------------------------------------------------------------------------------------------------------"

<< endl;

for (auto it = users.begin(); it != users.end(); ++it) {

for (auto it1 = allSessions.begin(); it1 != allSessions.end(); ++it1) {

for (auto it2 = petrolCars.begin(); it2 != petrolCars.end(); ++it2) {

if ((it2->GetID() == it1->carID) && (it->GetID() == it1->userID)) {

wcout << '|' << setw(10) << left << i << '|' << setw(15) << it->GetName() << '|' << setw(15)

<< it->GetSurname() << '|' << setw(15) << it2->GetBrand() << '|' << setw(15)

<< it2->GetModel() << '|';

cout << setw(2) << left << it1->DateFrom.day << '.' << setw(2) << it1->DateFrom.month << '.'

<< setw(9) << it1->DateFrom.year << '|' << setw(2) << it1->DateTo.day << '.' << setw(2)

<< it1->DateTo.month << '.' << setw(9) << it1->DateTo.year << '|' << setw(10) << it1->cost

<< '|' << endl;

cout

<< "-----------------------------------------------------------------------------------------------------------------------"

<< endl;

i++;

}

}

for (auto it2 = dieselCars.begin(); it2 != dieselCars.end(); ++it2) {

if ((it2->GetID() == it1->carID) && (it->GetID() == it1->userID)) {

wcout << '|' << setw(10) << left << i << '|' << setw(15) << it->GetName() << '|' << setw(15)

<< it->GetSurname() << '|' << setw(15) << it2->GetBrand() << '|' << setw(15)

<< it2->GetModel() << '|';

cout << setw(2) << left << it1->DateFrom.day << '.' << setw(2) << it1->DateFrom.month << '.'

<< setw(9) << it1->DateFrom.year << '|' << setw(2) << it1->DateTo.day << '.' << setw(2)

<< it1->DateTo.month << '.' << setw(9) << it1->DateTo.year << '|' << setw(10) << it1->cost

<< '|' << endl;

cout

<< "-----------------------------------------------------------------------------------------------------------------------"

<< endl;

i++;

}

}

for (auto it2 = electricCars.begin(); it2 != electricCars.end(); ++it2) {

if ((it2->GetID() == it1->carID) && (it->GetID() == it1->userID)) {

wcout << '|' << setw(10) << left << i << '|' << setw(15) << it->GetName() << '|' << setw(15)

<< it->GetSurname() << '|' << setw(15) << it2->GetBrand() << '|' << setw(15)

<< it2->GetModel() << '|';

cout << setw(2) << left << it1->DateFrom.day << '.' << setw(2) << it1->DateFrom.month << '.'

<< setw(9) << it1->DateFrom.year << '|' << setw(2) << it1->DateTo.day << '.' << setw(2)

<< it1->DateTo.month << '.' << setw(9) << it1->DateTo.year << '|' << setw(10) << it1->cost

<< '|' << endl;

cout

<< "-----------------------------------------------------------------------------------------------------------------------"

<< endl;

i++;

}

}

for (auto it2 = hybridCars.begin(); it2 != hybridCars.end(); ++it2) {

if ((it2->GetID() == it1->carID) && (it->GetID() == it1->userID)) {

wcout << '|' << setw(10) << left << i << '|' << setw(15) << it->GetName() << '|' << setw(15)

<< it->GetSurname() << '|' << setw(15) << it2->GetBrand() << '|' << setw(15)

<< it2->GetModel() << '|';

cout << setw(2) << left << it1->DateFrom.day << '.' << setw(2) << it1->DateFrom.month << '.'

<< setw(9) << it1->DateFrom.year << '|' << setw(2) << it1->DateTo.day << '.' << setw(2)

<< it1->DateTo.month << '.' << setw(9) << it1->DateTo.year << '|' << setw(10) << it1->cost

<< '|' << endl;

cout

<< "-----------------------------------------------------------------------------------------------------------------------"

<< endl;

i++;

}

}

}

}

}

void Interface :: ShowAllCars() {

cout << "----------------------------------------------------------------------------------" << endl;

wcout << '|' << setw(10) << left << L"Номер" << '|' << setw(15) << L"Марка" << '|' << setw(15) << L"Модель" << '|'

<< setw(15) << L"Год выпуска" << '|' << setw(10) << L"Стоимость" << '|' << setw(10) << L"Удален ли" << '|'

<< endl;

cout << "----------------------------------------------------------------------------------" << endl;

for (auto it = petrolCars.begin(); it != petrolCars.end(); ++it) {

wcout << '|' << setw(10) << left << it->GetID() << '|' << setw(15) << it->GetBrand() << '|' << setw(15)

<< it->GetModel();

cout << '|' << setw(15) << left << it->GetManufacturedYear() << '|' << setw(10) << left << it->GetCostPerDay()

<< '|';

if (it->IsDelete())

wcout << setw(10) << left << L"Да" << '|' << endl;

else

wcout << setw(10) << left << L"Нет" << '|' << endl;

cout << "----------------------------------------------------------------------------------" << endl;

}

for (auto it = dieselCars.begin(); it != dieselCars.end(); ++it) {

wcout << '|' << setw(10) << left << it->GetID() << '|' << setw(15) << it->GetBrand() << '|' << setw(15)

<< it->GetModel();

cout << '|' << setw(15) << left << it->GetManufacturedYear() << '|' << setw(10) << left << it->GetCostPerDay()

<< '|';

if (it->IsDelete())

wcout << setw(10) << left << L"Да" << '|' << endl;

else

wcout << setw(10) << left << L"Нет" << '|' << endl;

cout << "----------------------------------------------------------------------------------" << endl;

}

for (auto it = electricCars.begin(); it != electricCars.end(); ++it) {

wcout << '|' << setw(10) << left << it->GetID() << '|' << setw(15) << it->GetBrand() << '|' << setw(15)

<< it->GetModel();

cout << '|' << setw(15) << left << it->GetManufacturedYear() << '|' << setw(10) << left << it->GetCostPerDay()

<< '|';

if (it->IsDelete())

wcout << setw(10) << left << L"Да" << '|' << endl;

else

wcout << setw(10) << left << L"Нет" << '|' << endl;

cout << "----------------------------------------------------------------------------------" << endl;

}

for (auto it = hybridCars.begin(); it != hybridCars.end(); ++it) {

wcout << '|' << setw(10) << left << it->GetID() << '|' << setw(15) << it->GetBrand() << '|' << setw(15)

<< it->GetModel();

cout << '|' << setw(15) << left << it->GetManufacturedYear() << '|' << setw(10) << left << it->GetCostPerDay()

<< '|';

if (it->IsDelete())

wcout << setw(10) << left << L"Да" << '|' << endl;

else

wcout << setw(10) << left << L"Нет" << '|' << endl;

cout << "----------------------------------------------------------------------------------" << endl;

}

}

void Interface :: ShowSpecificUserSessions() {

this->ShowAllUsers();

int userID;

ifstream file;

cout << "Введите ID интересующего пользователя: ";

while (1) {

try {

cin >> userID;

if (cin.fail())

throw BadInputException("Введена не цифра");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

}

file.open("/Users/max/Desktop/CarRent/Files/User.bin", ios::binary); // кетч

if (!file.is\_open())

cout << "Error";

while (file.read((char \*) &user, sizeof(User))) {

if (user.GetID() == userID) {

this->SeeUserHistory(userID);

return;

}

}

cout << "Введен несуществующий ID" << endl;

return;

}

void Interface ::DeleteSpecificCar() {

int choice;

while (1) {

this->ShowAllCars();

cout << "Введите ID автомобиля для удаления: \n"

"Для выхода введите '0'" << endl;

while (1) {

try {

cin >> choice;

if (cin.fail())

throw BadInputException("Введена не цифра");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

}

if (choice == 0)

return;

cout << "Для подтверждения удаления введите '1'\n"

"Для отмены введите '0'" << endl;

int tmp;

while (1) {

try {

cin >> tmp;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (tmp < 0 || tmp > 1)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

if (tmp == 0)

return;

for (auto it = petrolCars.begin(); it != petrolCars.end(); ++it) {

if (it->GetID() == choice) {

it->DeleteCar();

cout << "Автомобиль успешно удален" << endl;

return;

}

}

for (auto it = dieselCars.begin(); it != dieselCars.end(); ++it) {

if (it->GetID() == choice) {

it->DeleteCar();

cout << "Автомобиль успешно удален" << endl;

return;

}

}

for (auto it = electricCars.begin(); it != electricCars.end(); ++it) {

if (it->GetID() == choice) {

it->DeleteCar();

cout << "Автомобиль успешно удален" << endl;

return;

}

}

for (auto it = hybridCars.begin(); it != hybridCars.end(); ++it) {

if (it->GetID() == choice) {

it->DeleteCar();

cout << "Автомобиль успешно удален" << endl;

return;

}

}

cout << "Введен неверный ID" << endl;

}

}

void Interface :: RefreshAllCars() {

ofstream file;

file.open("/Users/max/Desktop/CarRent/Files/PetrolCars.bin", ios::out | ios::binary | ios::trunc);

file.close();

for (auto it = petrolCars.begin(); it != petrolCars.end(); ++it)

Database::AddInDatabase(\*it);

file.open("/Users/max/Desktop/CarRent/Files/DieselCars.bin", ios::out | ios::binary | ios::trunc);

file.close();

for (auto it = dieselCars.begin(); it != dieselCars.end(); ++it)

Database::AddInDatabase(\*it);

file.open("/Users/max/Desktop/CarRent/Files/ElectricCars.bin", ios::out | ios::binary | ios::trunc);

file.close();

for (auto it = electricCars.begin(); it != electricCars.end(); ++it)

Database::AddInDatabase(\*it);

file.open("/Users/max/Desktop/CarRent/Files/HybridCars.bin", ios::out | ios::binary | ios::trunc);

file.close();

for (auto it = hybridCars.begin(); it != hybridCars.end(); ++it)

Database::AddInDatabase(\*it);

}

void Interface :: ShowSpecificCarSessions() {

ifstream file;

file.open("/Users/max/Desktop/CarRent/Files/User.bin", ios::binary);

if (!file.is\_open()) {

cout << "Ошибка открытия файла";

return;

}

vector<User> users;

while (file.read((char \*) &user, sizeof(User)))

users.push\_back(user);

this->ShowAllCars();

cout << "Введите ID интересуещего автомобиля: ";

int carID;

while (1) {

try {

cin >> carID;

if (cin.fail())

throw BadInputException("Введена не цифра");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

}

for (auto it = allSessions.begin(); it != allSessions.end(); ++it) {

if (it->carID == carID)

break;

cout << "Нет заказов или введен несуществующий номер!" << endl;

return;

}

int i = 0;

cout << "---------------------------------------------------------------------------------------" << endl;

wcout << '|' << setw(10) << left << L"№" << '|' << setw(15) << L"Фамилия" << '|' << setw(15) << L"Имя" << '|'

<< setw(15) << L"Дата начала" << '|' << setw(15) << L"Дата окончания" << '|' << setw(10) << L"Стоимость"

<< '|' << endl;

cout << "---------------------------------------------------------------------------------------" << endl;

for (auto it = allSessions.begin(); it != allSessions.end(); ++it)

{

if (it->carID == carID)

{

for (auto it1 = users.begin(); it1 != users.end(); ++it1) {

if (it->userID == it1->GetID()) {

wcout << '|' << setw(10) << left << i << '|' << setw(15) << it1->GetName() << '|' << setw(15)

<< it1->GetSurname() << '|';

cout << setw(2) << left << it->DateFrom.day << '.' << setw(2) << it->DateFrom.month << '.'

<< setw(9) << it->DateFrom.year << '|' << setw(2) << it->DateTo.day << '.' << setw(2)

<< it->DateTo.month << '.' << setw(9) << it->DateTo.year << '|' << setw(10) << it->cost << '|'

<< endl;

cout << "---------------------------------------------------------------------------------------"

<< endl;

i++;

}

}

}

}

return;

}

void Interface :: AddNewCar() {

cout << "Тип автомобиля тип для добавления\n"

"1 - Бензиновые автомобили\n"

"2 - Дизельные автомобили\n"

"3 - Электрические автомобили\n"

"4 - Гибридные автомобили\n"

"5 - Выход в главное меню" << endl;

int choice;

while (1) {

try {

cin >> choice;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (choice < 1 || choice > 5)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

PetrolCar pcar;

DieselCar dcar;

ElectricCar ecar;

HybridCar hcar;

switch (choice) {

case 1:

pcar.SetInfo();

Database::AddInDatabase(pcar);

petrolCars.emplace\_back(pcar);

break;

case 2:

dcar.SetInfo();

Database::AddInDatabase(dcar);

dieselCars.emplace\_back(dcar);

break;

case 3:

ecar.SetInfo();

Database::AddInDatabase(ecar);

electricCars.emplace\_back(ecar);

break;

case 4:

hcar.SetInfo();

Database::AddInDatabase(hcar);

hybridCars.emplace\_back(hcar);

break;

case 5:

return;

}

}

void Interface :: UserMainMenu() {

int choice;

while (1) {

cout << "===== Личный кабинет ===== \n"

"1 - Информация об аккаунте\n"

"2 - Автомобили для заказа\n"

"3 - История поездок\n"

"4 - Пополнение счета\n"

"5 - Настройка аккаунта\n"

"6 - Выйти из аккаунта\n"

"7 - Завершить работу" << endl;

while (1) {

try {

cin >> choice;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (choice < 1 || choice > 7)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

switch (choice) {

case 1:

this->UserAccountInfo(); break;

case 2:

this->SeeCars(); break;

case 3:

this->SeeUserHistory(user.GetID()); break;

case 4:

this->AddCount(); break;

case 5:

this->SetAccountSettings(); break;

case 6:

this->RefreshUserInFile(); return;

case 7:

this->RefreshUserInFile();

exit(0); // сохранять измененный объект юзера, заменяя его в файле

}

}

}

void Interface :: UserAccountInfo() {

cout << "Имя: ";

wcout << user.GetName() << ' ' << user.GetSurname() << endl;

cout << "Возраст: ";

cout << user.GetAge() << endl;

cout << "Баланс: ";

cout << user.GetCount() << endl;

cout << "Потрачено на платформе: ";

cout << user.GetSpendMoney() << endl; // не выводит дробную часть

cout << "Текущий статус: ";

wcout << user.GetStatus() << endl;

cout << "Номер паспорта, привязанного к аккаунту: ";

wcout << user.GetPassportNum() << endl;

cout << "Номер телефона: ";

wcout << user.GetPhoneNumber() << endl;

cout << "Адрес: ";

wcout << user.GetCountry() << ", " << user.GetCity() << ", " << user.GetStreet() << ' ' << user.GetHouse() << '-'

<< user.GetFlat() << endl;

}

void Interface :: SeeCars() {

while (1) {

int choice;

cout << "Какой тип автомобилей вас интересует?\n"

"1 - Бензиновые автомобили\n"

"2 - Дизельные автомобили\n"

"3 - Электрические автомобили\n"

"4 - Гибридные автомобили\n"

"5 - Выход в главное меню" << endl;

while (1) {

try {

cin >> choice;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (choice < 1 || choice > 5)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

if (choice == 5)

return;

while (1) {

if (this->ShowCarsForChoosing(choice)) {

cout << "1 - Выход в меню пользователя\n"

"2 - Выбор автомобиля\n"

"3 - Просмотр подробной информации об автомобиле\n"

"4 - Выбор в другого типа автомобилей" << endl;

} else

return;

int tmp;

while (1) {

try {

cin >> tmp;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (tmp < 0 || tmp > 4)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

if (tmp == 4)

break;

switch (tmp) {

case 1: return;

case 2:

this->CarOrder(choice); return;

case 3:

int id;

cout << "Введите номер автомобиля для просмотра: ";

cin >> id;

this->ShowDetailedInfo(choice, id); // логика как и в case 2

break;

}

}

}

}

bool Interface :: ShowCarsForChoosing(int choice) {

cout << "-----------------------------------------------------------------------" << endl;

wcout << '|' << setw(10) << left << L"Номер" << '|' << setw(15) << L"Марка" << '|' << setw(15) << L"Модель" << '|'

<< setw(15) << L"Год выпуска" << '|' << setw(10) << L"Стоимость" << '|' << endl;

cout << "-----------------------------------------------------------------------" << endl;

switch (choice) {

case 1:

if (petrolCars.empty()) {

cout << "Нет доступных автомобилей!" << endl;

return false;

}

for (auto it = petrolCars.begin(); it != petrolCars.end(); ++it) {

if (it->IsDelete())

continue;

wcout << '|' << setw(10) << left << it->GetID() << '|' << setw(15) << it->GetBrand() << '|' << setw(15)

<< it->GetModel();

cout << '|' << setw(15) << left << it->GetManufacturedYear() << '|' << setw(10) << left

<< it->GetCostPerDay() << '|' << endl;

cout << "-----------------------------------------------------------------------" << endl;

}

break;

case 2:

if (dieselCars.empty()) {

cout << "Нет доступных автомобилей!" << endl;

return false;

}

for (auto it = dieselCars.begin(); it != dieselCars.end(); ++it) {

if (it->IsDelete())

continue;

wcout << '|' << setw(10) << left << it->GetID() << '|' << setw(15) << it->GetBrand() << '|' << setw(15)

<< it->GetModel();

cout << '|' << setw(15) << left << it->GetManufacturedYear() << '|' << setw(10) << left

<< it->GetCostPerDay() << '|' << endl;

cout << "-----------------------------------------------------------------------" << endl;

}

break;

case 3:

if (electricCars.empty()) {

cout << "Нет доступных автомобилей!" << endl;

return false;

}

for (auto it = electricCars.begin(); it != electricCars.end(); ++it) {

if (it->IsDelete())

continue;

wcout << '|' << setw(10) << left << it->GetID() << '|' << setw(15) << it->GetBrand() << '|'

<< setw(15) << it->GetModel();

cout << '|' << setw(15) << left << it->GetManufacturedYear() << '|' << setw(10) << left

<< it->GetCostPerDay() << '|' << endl;

cout << "-----------------------------------------------------------------------" << endl;

}

break;

case 4:

if (hybridCars.empty()) {

cout << "Нет доступных автомобилей!" << endl;

return false;

}

for (auto it = hybridCars.begin(); it != hybridCars.end(); ++it) {

if (it->IsDelete())

continue;

wcout << '|' << setw(10) << left << it->GetID() << '|' << setw(15) << it->GetBrand() << '|' << setw(15)

<< it->GetModel();

cout << '|' << setw(15) << left << it->GetManufacturedYear() << '|' << setw(10) << left

<< it->GetCostPerDay() << '|' << endl;

cout << "-----------------------------------------------------------------------" << endl;

}

break;

}

return true;

}

void Interface :: CarOrder(int choice) {

cout << "Введите номер желаемого автомобиля: " << endl;

int carId;

int tmp;

while (1) {

try {

cin >> carId;

if (cin.fail())

throw BadInputException("Введена не цифра");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

}

sort(carIdInUsage.begin(), carIdInUsage.end());

// проверяем, доступен ли этот автомобиль

if (binary\_search(carIdInUsage.begin(), carIdInUsage.end(), carId)) {

cout << "К сожалению, данный автомобиль на данный момент недоступен!\n"

"Для ввода другого номера введите '1'\n"

"Для выхода в меню введите '2'" << endl;

while (1) {

try {

cin >> tmp;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (tmp < 1 || tmp > 2)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

if (tmp == 1) {

this->ShowCarsForChoosing(choice);

cout << "Введите номер желаемого автомобиля: " << endl;

cin >> carId;

} else

return;

}

while (!newSession.CreateSession(user.GetID(), carId, FindCost(choice, carId), user)) {

cout << "Желаете продолжить?\n"

"Для выбора другого автомобиля введите '1'\n"

"Для выхода в меню введите '2'" << endl;

while (1) {

try {

cin >> tmp;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (tmp < 1 || tmp > 2)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

if (tmp == 1) {

this->ShowCarsForChoosing(choice);

cout << "Введите номер желаемого автомобиля: " << endl;

cin >> carId;

} else

return;

}

carIdInUsage.push\_back(carId);

allSessions.push\_back(newSession);

}

int Interface :: FindCost(int choice, int carId) {

if (choice == 1) {

for (auto it = petrolCars.begin(); it != petrolCars.end(); ++it) {

if (it->GetID() == carId)

return it->GetCostPerDay();

}

}

if (choice == 2) {

for (auto it = dieselCars.begin(); it != dieselCars.end(); ++it) {

if (it->GetID() == carId)

return it->GetCostPerDay();

}

}

if (choice == 3) {

for (auto it = electricCars.begin(); it != electricCars.end(); ++it) {

if (it->GetID() == carId)

return it->GetCostPerDay();

}

}

if (choice == 4) {

for (auto it = hybridCars.begin(); it != hybridCars.end(); ++it) {

if (it->GetID() == carId)

return it->GetCostPerDay();

}

}

return 0;

}

void Interface :: ShowDetailedInfo(int choice, int id) {

switch (choice) {

case 1:

for (auto it = petrolCars.begin(); it != petrolCars.end(); ++it) {

if (it->IsDelete())

break;

if (it->GetID() == id) {

wcout << L"Марка: " << it->GetBrand() << endl;

wcout << L"Модель: " << it->GetModel() << endl;

wcout << L"Тип кузова: " << it->GetBodyType() << endl;

cout << "Год выпуска: " << it->GetManufacturedYear() << endl;

cout << "\nКоличество мест: " << it->GetNumberOfSeats() << endl;

wcout << L"Материал и цвет салона: " << it->GetInteriorMaterial() << ',' << it->GetInteriorColor()

<< endl;

wcout << L"Цвет автомобиля: " << it->GetColor() << endl;

cout << "Максимальная скорость: " << it->GetMaxSpeed() << " км/ч" << endl;

cout << "Объем двигателя: " << it->GetVolume() << " л." << endl;

cout << "Расход на 100 км: " << it->GetConsumption() << " л." << endl;

cout << "Минимальный тип бензина: " << it->GetPetrolType() << endl;

cout << "Стоимость аренды на день без учета скидок: " << it->GetCostPerDay() << " BYN" << endl;

cout << "Стоимость аренды на неделю без учета скидок: " << it->GetCostPerDay() \* 6 << " BYN"

<< endl;

cout << "Стоимость аренды на месяц без учета скидок: " << it->GetCostPerDay() \* 26 << " BYN"

<< endl;

return;

}

}

cout << "Введен неверный ID" << endl;

break;

case 2:

for (auto it = dieselCars.begin(); it != dieselCars.end(); ++it) {

if (it->IsDelete())

break;

if (it->GetID() == id) {

wcout << L"Марка: " << it->GetBrand() << endl;

wcout << L"Модель: " << it->GetModel() << endl;

wcout << L"Тип кузова: " << it->GetBodyType() << endl;

cout << "Год выпуска: " << it->GetManufacturedYear() << endl;

cout << "Количество мест: " << it->GetNumberOfSeats() << endl;

wcout << L"Материал и цвет салона: " << it->GetInteriorMaterial() << ", " << it->GetInteriorColor()

<< endl;

wcout << L"Цвет автомобиля: " << it->GetColor() << endl;

cout << "Максимальная скорость: " << it->GetMaxSpeed() << " км/ч" << endl;

cout << "Объем двигателя: " << it->GetVolume() << " л." << endl;

cout << "Расход на 100 км: " << it->GetConsumption() << " л." << endl;

cout << "Стоимость аренды на день без учета скидок: " << it->GetCostPerDay() << " BYN" << endl;

cout << "Стоимость аренды на неделю без учета скидок: " << it->GetCostPerDay() \* 6 << " BYN"

<< endl;

cout << "Стоимость аренды на месяц без учета скидок: " << it->GetCostPerDay() \* 26 << " BYN"

<< endl;

return;

}

}

cout << "Введен неdерный ID" << endl;

break;

case 3:

for (auto it = electricCars.begin(); it != electricCars.end(); ++it) {

if (it->IsDelete())

break;

if (it->GetID() == id) {

wcout << L"Марка: " << it->GetBrand() << endl;

wcout << L"Модель: " << it->GetModel() << endl;

wcout << L"Тип кузова: " << it->GetBodyType() << endl;

cout << "Год выпуска: " << it->GetManufacturedYear() << endl;

cout << "Количество мест: " << it->GetNumberOfSeats() << endl;

wcout << L"Материал и цвет салона: " << it->GetInteriorMaterial() << ", " << it->GetInteriorColor()

<< endl;

wcout << L"Цвет автомобиля: " << it->GetColor() << endl;

cout << "Максимальная скорость: " << it->GetMaxSpeed() << " км/ч" << endl;

cout << "Объем батареи: " << it->GetMaxSpeed() << " кВт/ч" << endl;

cout << "Стоимость аренды на день без учета скидок: " << it->GetCostPerDay() << " BYN" << endl;

cout << "Стоимость аренды на неделю без учета скидок: " << it->GetCostPerDay() \* 6 << " BYN"

<< endl;

cout << "Стоимость аренды на месяц без учета скидок: " << it->GetCostPerDay() \* 26 << " BYN"

<< endl;

return;

}

}

cout << "Введен неdерный ID" << endl;

break;

case 4:

for (auto it = hybridCars.begin(); it != hybridCars.end(); ++it) {

if (it->IsDelete())

break;

if (it->GetID() == id) {

wcout << L"Марка: " << it->GetBrand() << endl;

wcout << L"Модель: " << it->GetModel() << endl;

wcout << L"Тип кузова: " << it->GetBodyType() << endl;

cout << "Год выпуска: " << it->GetManufacturedYear() << endl;

cout << "Количество мест: " << it->GetNumberOfSeats() << endl;

wcout << L"Материал и цвет салона: " << it->GetInteriorMaterial() << ", " << it->GetInteriorColor()

<< endl;

wcout << L"Цвет автомобиля: " << it->GetColor() << endl;

wcout << L"Тип топлива: " << it->GetFuelType() << endl;

cout << "Максимальная скорость: " << it->GetMaxSpeed() << " км/ч" << endl;

cout << "Объем двигателя: " << it->GetVolume() << " л." << endl;

cout << "Расход на 100 км: " << it->GetConsumption() << " л." << endl;

cout << "Объем батареи: " << it->GetMaxSpeed() << " кВт/ч" << endl;

cout << "Стоимость аренды на день без учета скидок: " << it->GetCostPerDay() << " BYN" << endl;

cout << "Стоимость аренды на неделю без учета скидок: " << it->GetCostPerDay() \* 6 << " BYN"

<< endl;

cout << "Стоимость аренды на месяц без учета скидок: " << it->GetCostPerDay() \* 26 << " BYN"

<< endl;

return;

}

}

cout << "Введен неверный ID" << endl;

break;

}

}

void Interface :: SeeUserHistory(int userID) {

int i = 0;

cout << "---------------------------------------------------------------------------------------" << endl;

wcout << '|' << setw(10) << left << L"Номер" << '|' << setw(15) << L"Марка" << '|' << setw(15) << L"Модель" << '|'

<< setw(15) << L"Дата начала" << '|'

<< setw(15) << L"Дата окончания" << '|' << setw(10) << L"Стоимость" << '|' << endl;

cout << "---------------------------------------------------------------------------------------" << endl;

for (auto it = allSessions.begin(); it != allSessions.end(); ++it) {

i++;

for (auto it1 = petrolCars.begin(); it1 != petrolCars.end(); ++it1) {

if ((it1->GetID() == it->carID) && (userID == it->userID)) {

wcout << '|' << setw(10) << left << i << '|' << setw(15) << it1->GetBrand() << '|' << setw(15)

<< it1->GetModel() << '|';

cout << setw(2) << left << it->DateFrom.day << '.' << setw(2) << it->DateFrom.month << '.' << setw(9)

<< it->DateFrom.year << '|' << setw(2) << it->DateTo.day << '.' << setw(2) << it->DateTo.month

<< '.' << setw(9) << it->DateTo.year << '|' << setw(10) << it->cost << '|' << endl;

cout << "---------------------------------------------------------------------------------------"

<< endl;

}

}

for (auto it1 = dieselCars.begin(); it1 != dieselCars.end(); ++it1) {

if ((it1->GetID() == it->carID) && (userID == it->userID)) {

wcout << '|' << setw(10) << left << i << '|' << setw(15) << it1->GetBrand() << '|' << setw(15)

<< it1->GetModel() << '|';

cout << setw(2) << left << it->DateFrom.day << '.' << setw(2) << it->DateFrom.month << '.' << setw(9)

<< it->DateFrom.year << '|' << setw(2) << it->DateTo.day << '.' << setw(2) << it->DateTo.month

<< '.' << setw(9) << it->DateTo.year << '|' << setw(10) << it->cost << '|' << endl;

cout << "---------------------------------------------------------------------------------------"

<< endl;

}

}

for (auto it1 = electricCars.begin(); it1 != electricCars.end(); ++it1) {

if ((it1->GetID() == it->carID) && (userID == it->userID)) {

wcout << '|' << setw(10) << left << i << '|' << setw(15) << it1->GetBrand() << '|' << setw(15)

<< it1->GetModel() << '|';

cout << setw(2) << left << it->DateFrom.day << '.' << setw(2) << it->DateFrom.month << '.' << setw(9)

<< it->DateFrom.year << '|' << setw(2) << it->DateTo.day << '.' << setw(2) << it->DateTo.month

<< '.' << setw(9) << it->DateTo.year << '|' << setw(10) << it->cost << '|' << endl;

cout << "---------------------------------------------------------------------------------------"

<< endl;

}

}

for (auto it1 = hybridCars.begin(); it1 != hybridCars.end(); ++it1) {

if ((it1->GetID() == it->carID) && (userID == it->userID)) {

wcout << '|' << setw(10) << left << i << '|' << setw(15) << it1->GetBrand() << '|' << setw(15)

<< it1->GetModel() << '|';

cout << setw(2) << left << it->DateFrom.day << '.' << setw(2) << it->DateFrom.month << '.' << setw(9)

<< it->DateFrom.year << '|' << setw(2) << it->DateTo.day << '.' << setw(2) << it->DateTo.month

<< '.' << setw(9) << it->DateTo.year << '|' << setw(10) << it->cost << '|' << endl;

cout << "---------------------------------------------------------------------------------------"

<< endl;

}

}

}

}

void Interface :: AddCount() {

cout << "Введите сумму для пополнения" << endl;

float plus;

while (1) {

try {

cin >> plus;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (plus > FLT\_MAX)

throw OverflowException("Введено слишком большое значение");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (OverflowException exp) {

exp.Fix();

}

}

user.SetCount(plus);

}

void Interface :: SetAccountSettings() {

while (1) {

cout << "1 - Cменить номер паспорта\n"

"2 - Сменить номер телефона\n"

"3 - Сменить адрес\n"

"4 - Сменить логин\n"

"5 - Сменить пароль\n"

"6 - Выход в меню" << endl; // смена логина и пароля

int choice;

while (1) {

try {

cin >> choice;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (choice < 1 || choice > 6)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

switch (choice) {

case 1:

user.SetNewPassport(); break;

case 2:

user.SetNewNumber(); break;

case 3:

user.SetNewAdress(); break;

case 4:

Userdata::ChangeLogin(user.GetID()); break;

case 5:

Userdata::ChangePassword(user.GetID()); break;

case 6:

return;

}

}

}

void Interface :: RefreshUserInFile() {

ifstream file1; // для чтения

ofstream file2; // для записи

if (user.GetSpendMoney() >= 500 && user.GetSpendMoney() < 1000)

user.SetStatus(L"Silver");

else if (user.GetSpendMoney() >= 1000 && user.GetSpendMoney() < 2000)

user.SetStatus(L"Gold");

else if (user.GetSpendMoney() >= 2000)

user.SetStatus(L"Platinum");

file1.open("/Users/max/Desktop/CarRent/Files/User.bin", ios::binary);

if (!file1.is\_open())

cout << "Error";

file2.open("/Users/max/Desktop/CarRent/Files/tmp.bin", ios::binary | ios::app);

User tmp;

while (file1.read((char \*) &tmp, sizeof(User))) {

if (tmp.GetID() == user.GetID())

file2.write((char \*) &user, sizeof(User));

else

file2.write((char \*) &tmp, sizeof(User));

}

file1.close();

remove("/Users/max/Desktop/CarRent/Files/User.bin");

file2.close();

rename("/Users/max/Desktop/CarRent/Files/tmp.bin", "/Users/max/Desktop/CarRent/Files/User.bin");

}

**Класс Person**

class Person

{

protected:

int id;

wchar\_t name[30] = {L"\0"};

wchar\_t surname[30] = {L"\0"};

int age;

public:

Person();

~Person() = default;

int GetID() { return id; }

wchar\_t\* GetName() { return name; }

wchar\_t\* GetSurname() { return surname; };

int GetAge() { return age; }

virtual void SetInfo();

};

Person :: Person()

{

id = 0;

age = 18;

}

void Person :: SetInfo() {

cout << "Введите фамилию: ";

fflush(stdin);

while (1) {

try {

fgetws(surname, 30, stdin);

surname[wcscspn(surname, L"\n")] = L'\0';

if (surname[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(surname, L" ") < wcscspn(surname, L"\0")) // считает до \0

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

surname[0] = towupper(surname[0]);

cout << "Введите имя: ";

fflush(stdin);

while (1) {

try {

fgetws(name, 30, stdin);

name[wcscspn(name, L"\n")] = L'\0';

if (name[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(name, L" ") < wcscspn(name, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

name[0] = towupper(name[0]);

cout << "Введите возраст: ";

while (1) {

try {

cin >> age;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (age < 18 || age > 100)

throw Exception("Вам не может быть меньше 18 и больше 100 лет!");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

}

**Класс Admin**

class Admin : public Person

{

public:

Admin() { };

~Admin() = default;

void SetInfo();

};

void Admin :: SetInfo() {

id = IdGenerator::GetIdForPerson();

Person::SetInfo();

Userdata userdata(id);

userdata.SetInfo(1);

ofstream file;

file.open("/Users/max/Desktop/CarRent/Files/Admin.bin", ios::binary | ios::app);

if (!file.is\_open())

cout << "Error";

file.write((char \*) this, sizeof(Admin));

file.close();

}

**Класс User**

class User : public Person

{

private:

wchar\_t passportNum[15] = {L"\0"}; // номер паспорта

wchar\_t status[10] = {L"\0"}; // classic -0%, silver -5%, gold -10%, platinum -15%

float count; // счет

float spendMoney; // потраченные средства

wchar\_t phoneNumber[15] = {L"\0"};

struct Adress

{

wchar\_t country[30] = {L"\0"};

wchar\_t city[30] = {L"\0"};

wchar\_t street[30] = {L"\0"};

int house;

int flat;

} adress;

public:

User();

~User() = default;

void SetInfo(); // заполнение полей класса

void SetNewNumber(); // установка нового номера телефона

void SetNewAdress(); // установка нового номера паспорта

void SetNewPassport();

void SetCount(float plus) { count += plus; } // пополнение счета

void SetSpendMoney(float spend) { spendMoney += spend; } // изменение количества потраченных средств

void SetStatus(wchar\_t \*newStatus) { wcscpy(status, newStatus); } // установка нового статуса

wchar\_t\* GetPassportNum() { return passportNum; } // получение номера паспорта

wchar\_t\* GetStatus() { return status; } // получение текущего статуса

float GetCount() { return count; } // получение текущего баланса

float GetSpendMoney() { return spendMoney; } // получение потраченных средств

wchar\_t\* GetPhoneNumber() { return phoneNumber; } // получение номера телефона

wchar\_t\* GetCountry() { return adress.country; } // получение страны

wchar\_t\* GetCity() { return adress.city; } // получение города

wchar\_t\* GetStreet() { return adress.street; } // получение улицы

int GetHouse() { return adress.house; } // получение дома

int GetFlat() { return adress.flat; } // получение квартиры

};

User :: User() {

count = 0.0;

spendMoney = 0.0;

}

void User :: SetInfo() {

id = IdGenerator::GetIdForPerson();

Person::SetInfo();

Userdata userdata(id);

userdata.SetInfo(2);

wcscpy(status, L"Classic");

count = 0;

cout << "Введите номер паспорта: ";

fflush(stdin);

while (1) {

try {

fgetws(passportNum, 15, stdin);

passportNum[wcscspn(passportNum, L"\n")] = L'\0';

if (passportNum[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(passportNum, L" ") < wcscspn(passportNum, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

cout << "Введите номер телефона в формате <<КОД CТРАНЫ-ХХХХХХХХХХ>>: ";

fflush(stdin);

while (1) {

try {

fgetws(phoneNumber, 15, stdin);

phoneNumber[wcscspn(phoneNumber, L"\n")] = L'\0';

if (phoneNumber[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(phoneNumber, L" ") < wcscspn(phoneNumber, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

cout << "Введите страну проживания: ";

fflush(stdin);

while (1) {

try {

fgetws(adress.country, 30, stdin);

adress.country[wcscspn(adress.country, L"\n")] = L'\0';

if (adress.country[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

}

adress.country[0] = towupper(adress.country[0]);

cout << "Введите город проживания: ";

fflush(stdin);

while (1) {

try {

fgetws(adress.city, 30, stdin);

adress.city[wcscspn(adress.city, L"\n")] = L'\0';

if (adress.city[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(adress.city, L" ") < wcscspn(adress.city, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

adress.city[0] = towupper(adress.city[0]);

cout << "Введите улицу: ";

fflush(stdin);

while (1) {

try {

fgetws(adress.street, 30, stdin);

adress.street[wcscspn(adress.street, L"\n")] = L'\0';

if (adress.street[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(adress.street, L" ") < wcscspn(adress.street, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

adress.street[0] = towupper(adress.street[0]);

cout << "Введите дом: ";

while (1) {

try {

cin >> adress.house;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (adress.house <= 0)

throw Exception("Неверный ввод. ");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

cout << "Введите номер квартиры: ";

while (1) {

try {

cin >> adress.flat;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (adress.flat <= 0)

throw Exception("Неверный ввод. ");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

cout << "Аккаунт успешно создан!" << endl;

ofstream file;

file.open("../Files/User.bin", ios::binary | ios::app);

if (!file.is\_open())

cout << "Error";

file.write((char \*) this, sizeof(User));

file.close();

}

void User :: SetNewNumber() {

cout << "Введите номер телефона: ";

fflush(stdin);

wchar\_t tmpPhoneNumber[15];

while (1) {

try {

fgetws(tmpPhoneNumber, 15, stdin);

tmpPhoneNumber[wcscspn(tmpPhoneNumber, L"\n")] = L'\0';

if (tmpPhoneNumber[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(tmpPhoneNumber, L" ") < wcscspn(tmpPhoneNumber, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

cout << "Подтвердить изменения?\n"

"1 - Да\n"

"2 - Нет" << endl;

int choice;

while (1) {

try {

cin >> choice;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (choice < 1 || choice > 2)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

if (choice == 1)

wcscpy(phoneNumber, tmpPhoneNumber);

else

return;

}

void User :: SetNewAdress() {

wchar\_t country[30], city[30], street[30];

int house, flat;

cout << "Введите страну проживания: ";

fflush(stdin);

while (1) {

try {

fgetws(country, 30, stdin);

country[wcscspn(country, L"\n")] = L'\0';

if (country[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

}

country[0] = towupper(country[0]);

cout << "Введите город проживания: ";

fflush(stdin);

while (1) {

try {

fgetws(city, 30, stdin);

city[wcscspn(city, L"\n")] = L'\0';

if (city[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(city, L" ") < wcscspn(city, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

city[0] = towupper(city[0]);

cout << "Введите улицу: ";

fflush(stdin);

while (1) {

try {

fgetws(street, 30, stdin);

street[wcscspn(street, L"\n")] = L'\0';

if (street[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(street, L" ") < wcscspn(street, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

street[0] = towupper(street[0]);

cout << "Введите дом: ";

while (1) {

try {

cin >> house;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (house <= 0)

throw Exception("Неверный ввод. ");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

cout << "Введите номер квартиры: ";

while (1) {

try {

cin >> flat;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (flat <= 0)

throw Exception("Неверный ввод. ");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

cout << "Подтвердить изменения?\n"

"1 - Да\n"

"2 - Нет" << endl;

int choice;

while (1) {

try {

cin >> choice;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (choice < 1 || choice > 2)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

if (choice == 1) {

wcscpy(adress.country, country);

wcscpy(adress.city, city);

wcscpy(adress.street, street);

adress.house = house;

adress.flat = flat;

} else

return;

}

void User :: SetNewPassport() {

wchar\_t tmpPassportNum[15];

cout << "Введите новый номер паспорта: ";

fflush(stdin);

while (1) {

try {

fgetws(tmpPassportNum, 15, stdin);

tmpPassportNum[wcscspn(tmpPassportNum, L"\n")] = L'\0';

if (tmpPassportNum[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(tmpPassportNum, L" ") < wcscspn(tmpPassportNum, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

cout << "Подтвердить изменения?\n"

"1 - Да\n"

"2 - Нет" << endl;

int choice;

while (1) {

try {

cin >> choice;

if (cin.fail())

throw BadInputException("Введена не цифра");

if (choice < 1 || choice > 2)

throw Exception("Введен неверный пункт");

break;

}

catch (BadInputException exp) {

exp.Fix();

}

catch (Exception exp) {

exp.Fix();

}

}

if (choice == 1)

wcscpy(passportNum, tmpPassportNum);

else

return;

}

**Класс Userdata**

class Userdata

{

friend class Interface;

private:

wchar\_t login[30] = {L"\0"}; // ограничить длину

wchar\_t password[30] = {L"\0"};

int id;

public:

Userdata();

Userdata(int);

int GetID() { return id; }

void SetInfo(int);

void SetData(wchar\_t\*, wchar\_t\*);

wchar\_t\* GetLogin() { return login; }

wchar\_t\* GetPassword() { return password; }

bool CheckAccess(int); // admin - 1, user - 2

static bool CheckLogin(wchar\_t\*); // admin - 1, user - 2

static void ChangeLogin(int);

static void ChangePassword(int);

};

Userdata :: Userdata() {

id = 0;

}

Userdata :: Userdata(int id) {

this->id = id;

}

void Userdata :: SetInfo(int mode)

{

cout << "Введите логин (пробелы недопустимы): ";

fflush(stdin);

while (1)

{

try

{

fgetws(login, 30, stdin);

login[wcscspn(login, L"\n")] = L'\0';

if (login[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(login, L" ") < wcscspn(login, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp)

{

exp.Fix();

}

catch (SpaceException exp)

{

exp.Fix();

}

}

while (!this->CheckLogin(login))

{

cout << "Данный логин недоступен! " << endl;

cout << "Введите другой логин: " << endl;

fflush(stdin);

while (1)

{

try

{

fgetws(login, 30, stdin);

login[wcscspn(login, L"\n")] = L'\0';

if (login[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(login, L" ") < wcscspn(login, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp)

{

exp.Fix();

}

catch (SpaceException exp)

{

exp.Fix();

}

}

}

cout << "Введите пароль (пробелы недопустимы): ";

fflush(stdin);

while (1)

{

try

{

fgetws(password, 30, stdin);

password[wcscspn(password, L"\n")] = L'\0';

if (login[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(password, L" ") < wcscspn(password, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp)

{

exp.Fix();

}

catch (SpaceException exp)

{

exp.Fix();

}

}

ofstream file;

switch (mode)

{

case 1:

file.open("/Users/max/Desktop/CarRent/Files/AdminAuthentication.bin", ios::binary | ios::app);

if (!file.is\_open())

cout << "Error";

file.write((char\*)this, sizeof(Userdata));

file.close();

break;

case 2:

file.open("/Users/max/Desktop/CarRent/Files/UserAuthentication.bin", ios::binary | ios::app);

if (!file.is\_open())

cout << "Error";

file.write((char\*)this, sizeof(Userdata));

file.close();

break;

}

}

void Userdata :: SetData(wchar\_t login[], wchar\_t password[])

{

wcscpy(this->login, login);

wcscpy(this->password, password);

}

bool Userdata :: CheckAccess(int mode) {

Userdata tmp;

ifstream file;

if (mode == 1) {

file.open("/Users/max/Desktop/CarRent/Files/AdminAuthentication.bin", ios::binary);

if (!file.is\_open())

cout << "Error";

// добавить исключение

while (file.read((char \*) &tmp, sizeof(tmp))) {

if (!wcscmp(tmp.login, login) && !wcscmp(tmp.password, password)) {

this->id = tmp.id;

file.close();

return true;

}

}

file.close();

return false;

}

if (mode == 2) {

file.open("/Users/max/Desktop/CarRent/Files/UserAuthentication.bin", ios::binary);

if (!file.is\_open())

cout << "Error";

// добавить искючения

while (file.read((char \*) &tmp, sizeof(tmp))) {

if (!wcscmp(tmp.login, login) && !wcscmp(tmp.password, password)) {

this->id = tmp.id;

file.close();

return true;

}

}

file.close();

return false;

}

}

bool Userdata :: CheckLogin(wchar\_t login[]) {

Userdata tmp;

ifstream file;

file.open("/Users/max/Desktop/CarRent/Files/AdminAuthentication.bin", ios::binary);

if (!file.is\_open())

cout << "Error";

while (file.read((char \*) &tmp, sizeof(tmp))) {

if (!wcscmp(tmp.login, login)) {

file.close();

return false;

}

}

file.close();

file.open("/Users/max/Desktop/CarRent/Files/UserAuthentication.bin", ios::binary);

if (!file.is\_open())

cout << "Error";

while (file.read((char \*) &tmp, sizeof(tmp))) {

if (!wcscmp(tmp.login, login)) {

file.close();

return false;

}

}

return true;

}

void Userdata :: ChangeLogin(int id) {

Userdata newUserdata;

ifstream file;

file.open("/Users/max/Desktop/CarRent/Files/UserAuthentication.bin", ios::binary);

if (!file.is\_open())

cout << "Error";

while (file.read((char \*) &newUserdata, sizeof(Userdata))) {

if (newUserdata.GetID() == id)

break;

}

file.close();

wchar\_t login[30];

cout << "Введите новый логин (пробелы недопустимы): ";

fflush(stdin);

while (1) {

try {

fgetws(login, 30, stdin);

login[wcscspn(login, L"\n")] = L'\0';

if (login[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(login, L" ") < wcscspn(login, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

while (CheckLogin(login) == false) {

cout << "Данный логин недоступен! " << endl;

cout << "Введите другой логин: " << endl;

fflush(stdin);

while (1) {

try {

fgetws(login, 30, stdin);

login[wcscspn(login, L"\n")] = L'\0';

if (login[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(login, L" ") < wcscspn(login, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

}

newUserdata.SetData(login, newUserdata.GetPassword());

ifstream file1; // для чтения

ofstream file2; // для записи

file1.open("/Users/max/Desktop/CarRent/Files/UserAuthentication.bin", ios::binary);

if (!file1.is\_open())

cout << "Error";

file2.open("/Users/max/Desktop/CarRent/Files/tmp.bin", ios::binary | ios::app);

Userdata tmp;

while (file1.read((char \*) &tmp, sizeof(Userdata))) {

if (tmp.GetID() == newUserdata.GetID())

file2.write((char \*) &newUserdata, sizeof(newUserdata));

else

file2.write((char \*) &tmp, sizeof(newUserdata));

}

file1.close();

remove("/Users/max/Desktop/CarRent/Files/UserAuthentication.bin");

file2.close();

rename("/Users/max/Desktop/CarRent/Files/tmp.bin", "/Users/max/Desktop/CarRent/Files/UserAuthentication.bin");

}

void Userdata :: ChangePassword(int id) {

Userdata newUserdata;

ifstream file;

file.open("/Users/max/Desktop/CarRent/Files/UserAuthentication.bin", ios::binary);

if (!file.is\_open())

cout << "Error";

while (file.read((char \*) &newUserdata, sizeof(Userdata))) {

if (newUserdata.GetID() == id)

break;

}

file.close();

wchar\_t password[30];

cout << "Введите новый пароль (пробелы недопустимы): ";

fflush(stdin);

while (1) {

try {

fgetws(password, 30, stdin);

password[wcscspn(password, L"\n")] = L'\0';

if (password[0] == L'\0')

throw EmptyInputException("Вы ничего не ввели");

if (wcscspn(password, L" ") < wcscspn(password, L"\0"))

throw SpaceException("Введена строка с пробелом");

break;

}

catch (EmptyInputException exp) {

exp.Fix();

}

catch (SpaceException exp) {

exp.Fix();

}

}

newUserdata.SetData(newUserdata.GetLogin(), password);

ifstream file1; // для чтения

ofstream file2; // для записи

file1.open("/Users/max/Desktop/CarRent/Files/UserAuthentication.bin", ios::binary);

if (!file1.is\_open())

cout << "Error";

file2.open("/Users/max/Desktop/CarRent/Files/tmp.bin", ios::binary | ios::app);

Userdata tmp;

while (file1.read((char \*) &tmp, sizeof(Userdata))) {

if (tmp.GetID() == newUserdata.GetID())

file2.write((char \*) &newUserdata, sizeof(newUserdata));

else

file2.write((char \*) &tmp, sizeof(newUserdata));

}

file1.close();

remove("/Users/max/Desktop/CarRent/Files/UserAuthentication.bin");

file2.close();

rename("/Users/max/Desktop/CarRent/Files/tmp.bin", "/Users/max/Desktop/CarRent/Files/UserAuthentication.bin");

}

**Класс Session**

class Session

{

friend class Interface;

private:

int userID;

int carID;

float cost;

struct DateFrom

{

int day;

int month;

int year;

} DateFrom;

struct DateTo

{

int day;

int month;

int year;

} DateTo;

public:

Session();

~Session() = default;

bool CreateSession(int, int, int, User&); // создание заказа

static void GetAllSessions(vector<Session>&);

bool CheckDate(int);

bool CheckDate(int, int);

int JDDate(int, int, int);

int NowDays();

bool InputDateCheck(int, int, int);

};

Session :: Session() {

userID = 0;

carID = 0;

cost = 0.0;

DateFrom.day = 0;

DateFrom.month = 0;

DateFrom.year = 0;

DateTo.day = 0;

DateTo.month = 0;

DateTo.year = 0;

}

bool Session :: CreateSession(int userID, int carID, int costPerDay, User &user) {

if (costPerDay == 0) {

cout << "Введен неверный номер автомобиля" << endl;

return false;

}

this->userID = userID;

this->carID = carID;

int term;

int JDNDay[2];

cout << "С какой даты начинается аренда? Введите день, месяц и год через пробел" << endl;

cin >> DateFrom.month;

cin >> DateFrom.year;

JDNDay[0] = JDDate(DateFrom.day, DateFrom.month, DateFrom.year);

while (!this->CheckDate(JDNDay[0]) && !this->InputDateCheck(DateFrom.day, DateFrom.month, DateFrom.year)) {

cout << "Ошибка! Введите дату еще раз!" << endl;

cin >> DateFrom.day;

cin >> DateFrom.month;

cin >> DateFrom.year;

JDNDay[0] = JDDate(DateFrom.day, DateFrom.month, DateFrom.year);

}

if (JDNDay[0] - NowDays() > 3) {

cout << "Ошибка! Максимальный срок до начала аренды - 3 дня" << endl;

return false;

}

cout << "До какой даты аренда? Введите день, месяц и год через пробел" << endl;

cin >> DateTo.day;

cin >> DateTo.month;

cin >> DateTo.year;

JDNDay[1] = JDDate(DateTo.day, DateTo.month, DateTo.year);

while (!this->CheckDate(JDNDay[0], JDNDay[1]) && !this->InputDateCheck(DateFrom.day, DateFrom.month, DateFrom.year)) {

cout << "Ошибка! Введите дату еще раз!" << endl;

cin >> DateTo.day;

cin >> DateTo.month;

cin >> DateTo.year;

JDNDay[1] = JDDate(DateFrom.day, DateFrom.month, DateFrom.year);

}

term = JDNDay[1] - JDNDay[0] + 1;

// механизм расчета стоимости поездки

if (term == 0)

cost = costPerDay;

if (term > 0 || term <= 6) {

cost = costPerDay \* term;

}

if (term > 6 && term < 30) {

int week = (term - (term % 7)) / 7;

cost = week \* costPerDay \* 6 + (term - week \* 7) \* costPerDay;

}

if (term > 30) {

int month = (term - (term % 30)) / 30;

int week = ((term - month \* 30) - ((term - month \* 30) % 7)) / 7;

cost = month \* costPerDay \* 26 + week \* costPerDay \* 6 + (term - month \* 30 - week \* 7) \* costPerDay;

}

if (cost > user.GetCount()) {

cout << "Недостаточно средств" << endl; // прерывание, если недостаточно средств

return false;

}

if (!wcscmp(user.GetStatus(), L"Silver"))

cost = cost - (cost \* 0.05);

if (!wcscmp(user.GetStatus(), L"Gold"))

cost = cost - (cost \* 0.1);

if (!wcscmp(user.GetStatus(), L"Platinum"))

cost = cost - (cost \* 0.15);

cout << "Стоимость аренды составит " << cost << "BYN "<< endl;

cout << "Для подтверждения заказа введите '1'\n"

"Для отмены заказа введите '2'" << endl;

int tmp;

cin >> tmp;

while (tmp > 2 || tmp < 1) {

cout << "Ошибка. Введите еще раз:" << endl;

cin >> tmp;

}

if (tmp == 2)

return false;

user.SetCount(-1 \* cost);

user.SetSpendMoney(cost);

CarInUsage object(JDNDay[1], carID);

object.WriteInFile();

ofstream file;

file.open("/Users/max/Desktop/CarRent/Files/Session.bin", ios::binary | ios::app);

if (!file.is\_open())

cout << "Error";

file.write((char \*) this, sizeof(Session));

file.close();

cout << "Заказ успешно оформлен!" << endl;

return true;

}

int Session :: NowDays() {

int nowDays;

time\_t seconds = time(NULL);

tm \*currentTime = localtime(&seconds);

int a = (14 - currentTime->tm\_mon + 1) / 12;

int y = currentTime->tm\_year + 1900 + 4800 - a;

int m = currentTime->tm\_mon + 1 + 12 \* a - 3;

nowDays = currentTime->tm\_mday + ((153 \* m + 2) / 5) + 365 \* y + y / 4 - y / 100 + y / 400 - 32045;

return nowDays;

}

bool Session :: CheckDate(int daysNum) {

int nowDays = NowDays();

if (nowDays <= daysNum)

return true;

else

return false;

}

bool Session :: CheckDate(int daysNumFrom, int daysNumTo) {

if (daysNumFrom > daysNumTo)

return false;

else

return true;

}

int Session :: JDDate(int day, int month, int year) {

int a = (14 - month) / 12;

int y = year + 4800 - a;

int m = month + 12 \* a - 3;

int result = day + ((153 \* m + 2) / 5) + 365 \* y + y / 4 - y / 100 + y / 400 - 32045;

return result;

}

void Session :: GetAllSessions(vector<Session>&allSessions) {

Session session;

ifstream file;

file.open("../Files/Session.bin", ios::binary);

if (!file.is\_open())

cout << "Error";

while (file.read((char \*) &session, sizeof(Session)))

allSessions.push\_back(session);

}

bool Session :: InputDateCheck(int day, int month, int year) {

while (year > 2022)

return false;

while (month > 12)

return false;

while (month == 1 || month == 3 || month == 5 || month == 7 ||

month == 8 || month == 10 || month == 12 && day > 31)

return false;

while (month == 4 || month == 6 ||

month == 9 || month == 11 && day > 30)

return false;

while (month == 2 && year % 4 == 0 && day > 29)

return false;

while (month == 2 && year % 4 != 0 && day > 28)

return false;

return true;

}