**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

Курсовая работа

**по дисциплине**

**«****Объектно-ориентированное программирование»**

Тема: «Разработка приложений на основе принципов объектно-ориентированного подхода»

|  |  |  |
| --- | --- | --- |
| Студент гр. 1335 |  | Максимов Ю.Е. |

|  |  |  |
| --- | --- | --- |
| Преподаватель: |  | Новакова Н.Е. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Максимов Ю. Е. | | |
| Группа 1335 | | |
| Тема работы: Разработка приложений на основе принципов объектно-ориентированного подхода | | |
| Исходные данные:  Среда разработки: CLion  Язык программирования: C++ | | |
| Содержание пояснительной записки:  «Содержание», «Введение», «Цель работы», «Задание», «Теоретические сведения», «Формализация задачи», «Спецификация программы», «Руководство пользователя», «Руководство программиста», «Контрольный пример», «Листинг программы», «Заключение», «Список использованных источников» | | |
| Предполагаемый объем пояснительной записки:  Не менее 40 страниц. | | |
| Дата выдачи задания: 21.06.2023 | | |
| Дата сдачи курсовой работы: 26.06.2023 | | |
| Дата защиты курсовой работы: 26.06.2023 | | |
| Студент |  | Максимов Ю.Е. |
| Преподаватель |  | Новакова Н.Е. |

**Аннотация**

Курсовая работа содержит в себе решение трех задач на основе объектно-ориентированного подхода. В первой задаче рассмотрена иерархия классов, во второй – реализация алгоритма Дейкстры на графе, в третьей – разработка имитационной модели.

На основе этих моделей были разработаны приложения, включающие в себя различные пользовательские классы. Полученные результаты приведены в работе.

**Summary**

Course work contains the solution of three problems based on an object-oriented approach. In the first task, the class hierarchy is considered, in the second, the implementation of the Dijkstra algorithm on a graph, in the third, the development of a simulation model.

Based on these models, applications have been developed that include various user interfaces. The results obtained are presented in the work.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение  Цель работы | 6  6 |
| 1. | Разработка объектной модели | 7 |
| 1.1. | Задание | 7 |
| 1.2.  1.3.  1.4.  1.5.  1.6.  1.7.  1.8. | Теоретические сведения  Формализация задачи  Спецификация программы  Руководство пользователя  Руководство программиста  Контрольный пример  Листинг программы | 7  7  8  9  10  10  14 |
| 2. | Работа с графами | 18 |
| 2.1. | Задание | 18 |
| 2.2.  2.3.  2.4.  2.5.  2.6.  2.7.  2.8. | Теоретические сведения  Формализация задачи  Спецификация программы  Руководство пользователя  Руководство программиста  Контрольный пример  Листинг программы | 18  19  19  20  20  21  21 |
| 3. | Имитационное моделирование | 24 |
| 3.1. | Задание | 24 |
| 3.2.  3.3.  3.4.  3.5.  3.6.  3.7.  3.8. | Теоретические сведения  Формализация задачи  Спецификация программы  Руководство пользователя  Руководство программиста  Контрольный пример  Листинг программы | 24  25  26  28  30  31  36 |
|  | Заключение | 50 |
|  | Список использованных источников | 51 |

введение

Курсовая работа направлена на создание приложений на основе объектно-ориентированного подхода на языке С++. В ней рассматриваются иерархии классов и наследования, а также имитационные модели. Для первых двух задач реализовывается консольное приложение для взаимодействия с пользователем, для третьей используются Qt.

ЦЕЛЬ РАБОТЫ

Целью курсовой работы является закрепление теоретических знаний и получение практических навыков разработки программного обеспечения на основе объектно-ориентированного подхода.

1. Разработка объектной модели
   1. Задание

Вариант 1

Разработать программу для представления транспортных средств. Определить основные пассажирские транспортные средства: легковой автомобиль, автобус, поезд, водные транспортные средства, самолет.

* 1. Теоретические сведения

Легковой автомобиль, автобус, поезд, водные транспортные средства, самоле это все производное от транспортного средства.

* 1. Формализация задачи

Сore – основной класс где создаются различные транспортные средства.

TransportBase – базовый класс транспорта.

Bus – класс наследник TransportBase, реализующий автобус.

PassengerCar – класс наследник TransportBase, реализующий легковой автомобиль.

Plane – класс наследник TransportBase, реализующий самолет.

Train – класс наследник TransportBase, реализующий поезд.

WaterVehicles – класс наследник TransportBase, реализующий водный транспорт.

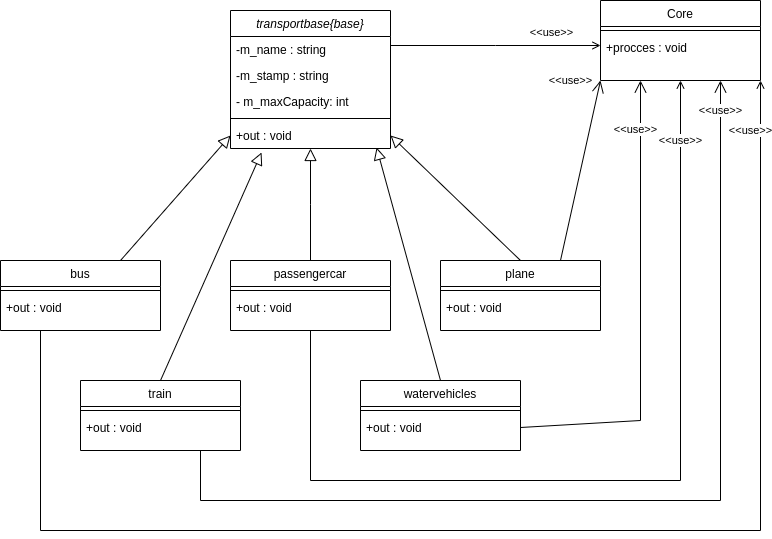


Рисунок 1.3.1. UML диаграмма классов

* 1. Спецификация программы

Таблица 1.3.1 Описание методов класса Сore

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
| --- | --- | --- | --- | --- |
| process | void | public | - | Создание трансопрта |

Таблица 1.3.2 Описание методов класса TransportBase

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
| --- | --- | --- | --- | --- |
| out | void | public | - | Отображение транспорта |

Таблица 1.3.3 Описание методов класса Bus

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
| --- | --- | --- | --- | --- |
| out | void | public | - | Перегруженный метод отображения автобуса |

Таблица 1.3.4 Описание методов класса PassengerCar

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
| --- | --- | --- | --- | --- |
| out | void | public | - | Перегруженный метод отображения легковой машины |

Таблица 1.3.5 Описание методов класса Plane

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
| --- | --- | --- | --- | --- |
| out | void | public | - | Перегруженный метод отображения самолета |

Таблица 1.3.5 Описание методов класса Train

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
| --- | --- | --- | --- | --- |
| out | void | public | - | Перегруженный метод отображения поезда |

Таблица 1.3.6 Описание методов класса WaterVehicles

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
| --- | --- | --- | --- | --- |
| out | void | public | - | Перегруженный метод отображения водного транспорта |

Таблица 1.3.7 Описание полей класса TransportBase

| Поля | Тип | Модификатор доступа | Назначение |
| --- | --- | --- | --- |
| m\_stamp | std::string | private | Изготовитель |
| m\_name | std::string | private | Имя |
| m\_maxCapacity | Int | private | Максимальная вместимость пассажиров |

* 1. Руководство пользователя
     1. Назначение программы

Представления различных транспортных средств.

* + 1. Условие выполнения программы:

Для запуска программы необходим ggc/clang, cmake, Ninja.

* + 1. Выполнение программы:

1. Пользователь запускает программу.
2. В консоли выводится разный типаж транспортных средств с их максимальной вместимостью пассажиров и марками.
   1. Руководство программиста
      1. Запуск программы:

Для запуска программы необходим ggc/clang, cmake, Ninja.

* + 1. Характеристика программы

Программа имеет список команд для функционирования, позволяя продемонстрировать различные транспортные средства.

* + 1. Входные и выходные данные

Программа не предполагает взаимодействие с пользователем напрямую. Новые данные об объектах вносит программист. Результатом служит вывод на консоль.

* 1. Контрольный пример

На рисунке 1.7.1 было произведено создания различных транспортных средств :

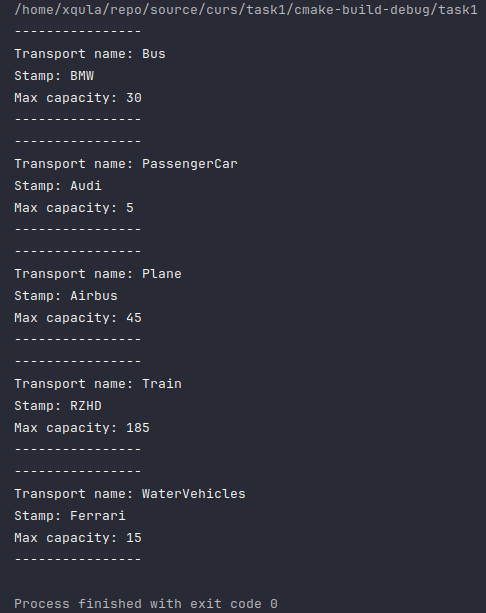


Рисунок 1.7.1 Работа программы

* 1. Листинг программы

CORE.CPP

//  
// Created by xqula on 27.06.24.  
//  
#include "core.h"  
#include "transport/transportbase.h"  
#include "transport/bus.h"  
#include "transport/passengercar.h"  
#include "transport/plane.h"  
#include "transport/train.h"  
#include "transport/watervehicles.h"  
#include "memory"  
auto Core::**process**() -> void {  
 const auto transport = std::make\_unique<Bus>("Bus", "BMW", 30);  
 transport->out();  
 const auto transport2 = std::make\_unique<PassengerCar>("PassengerCar", "Audi", 5);  
 transport2->out();  
 const auto transport3 = std::make\_unique<Plane>("Plane", "Airbus", 45);  
 transport3->out();  
 const auto transport4 = std::make\_unique<Train>("Train", "RZHD", 185);  
 transport4->out();  
 const auto transport5 = std::make\_unique<WaterVehicles>("WaterVehicles", "Ferrari", 15);  
 transport5->out();  
}

CORE.H

#pragma once  
class Core {  
public:  
 Core() = default;  
 ~Core() = default;  
  
 static auto **process**() -> void;  
  
};

BUS.CPP

#include "bus.h"  
Bus::Bus(std::string *name*, std::string *stamp*, const int *maxCapacity*)  
 : TransportBase(std::move(*name*), std::move(*stamp*), *maxCapacity*)  
{  
}  
void Bus::out() {  
 TransportBase::out();  
}

BUS.H

#pragma once  
#include "transportbase.h"  
class Bus : public TransportBase{  
public:  
 Bus(std::string *name*, std::string *stamp*, int *maxCapacity*);  
 ~Bus() override = default;  
 auto out() -> void override;  
};

PASSENGERCAR.CPP

#include "passengercar.h"  
PassengerCar::PassengerCar(std::string *name*, std::string *stamp*, const int *maxCapacity*)  
 : TransportBase(std::move(*name*), std::move(*stamp*), *maxCapacity*)  
{  
}  
void PassengerCar::out() {  
 TransportBase::out();  
}

PASSENGERCAR.H

#pragma once  
#include "transportbase.h"  
class PassengerCar : public TransportBase {  
public:  
 PassengerCar(std::string *name*, std::string *stamp*, int *maxCapacity*);  
 ~PassengerCar() override = default;  
 auto out() -> void override;  
};

PLANE.CPP

#include "plane.h"  
Plane::Plane(std::string *name*, std::string *stamp*, const int *maxCapacity*)  
 : TransportBase(std::move(*name*), std::move(*stamp*), *maxCapacity*)  
{  
}  
void Plane::out() {  
 TransportBase::out();  
}

PLANE.H

#pragma once  
#include "transportbase.h"  
class Plane : public TransportBase {  
public:  
 Plane(std::string *name*, std::string *stamp*, int *maxCapacity*);  
 ~Plane() override = default;  
 auto out() -> void override;  
  
};

TRAIN.CPP

#include "train.h"  
Train::Train(std::string *name*, std::string *stamp*, const int *maxCapacity*)  
 : TransportBase(std::move(*name*), std::move(*stamp*), *maxCapacity*)  
{  
}  
void Train::out() {  
 TransportBase::out();  
}

TRAIN.H

#pragma once  
#include "transportbase.h"  
class Train : public TransportBase {  
public:  
 Train(std::string *name*, std::string *stamp*, int *maxCapacity*);  
 ~Train() override = default;  
 auto out() -> void override;  
};

TRANSPORTBASE.CPP

#include "transportbase.h"  
#include <iostream>  
#include <utility>  
TransportBase::TransportBase(std::string *name*, std::string *stamp*, const int *maxCapacity*) :  
 m\_name(std::move(*name*)), m\_stamp(std::move(*stamp*)), m\_maxCapacity(*maxCapacity*)  
{  
}  
auto TransportBase::out() -> void {  
 std::cout << "----------------" << std::endl;  
 std::cout << "Transport name: " << m\_name << std::endl  
 << "Stamp: " << m\_stamp << std::endl  
 << "Max capacity: " << m\_maxCapacity << std::endl;  
 std::cout << "----------------" << std::endl;  
}

TRANSPORTBASE.H

#pragma once  
#include <string>  
class TransportBase {  
public:  
 explicit TransportBase(std::string *name*, std::string *stamp*, int *maxCapacity*);  
 virtual ~TransportBase() = default;  
 virtual auto out() -> void;  
private:  
 std::string m\_name;  
 std::string m\_stamp;  
 int m\_maxCapacity;  
  
};

WATERVEHICLES.CPP

#include "watervehicles.h"  
WaterVehicles::WaterVehicles(std::string *name*, std::string *stamp*, const int *maxCapacity*)  
 : TransportBase(std::move(*name*), std::move(*stamp*), *maxCapacity*)  
{  
}  
void WaterVehicles::out() {  
 TransportBase::out();  
}

WATERVEHICLES.H

#pragma once  
#include "transportbase.h"  
class WaterVehicles : TransportBase {  
public:  
 WaterVehicles(std::string *name*, std::string *stamp*, int *maxCapacity*);  
 ~WaterVehicles() override = default;  
 auto out() -> void override;  
};

2. Работа с графами

2.1. Задание

Вариант задания Г-41-4

Для заданного орграфа найти кратчайший путь между вершинами 1 и 12. Задачу решить в общем виде. На рисунке рядом с вершинами указан ее вес.

В качестве контрольного примера используются следующий граф:

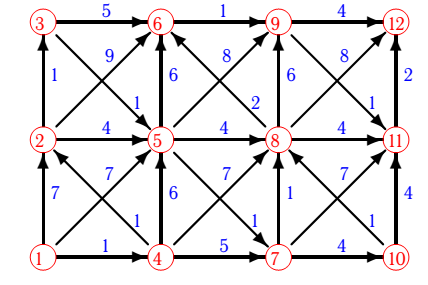


Рисунок 2.1.1 Граф для контрольного примера

2.2. Теоретические сведения

Граф (G(V,E)) – совокупность двух множеств — множества объектов V, называемого множеством вершин, и множества их парных связей E, называемого множеством рёбер. Элемент множества рёбер есть пара элементов множества вершин.

Взвешенный граф – граф, в котором каждому ребру поставлено в соответствие некоторое число, называемое весом ребра.

Матрица смежности – таблица, где как столбцы, так и строки соответствуют вершинам графа. В каждой ячейке этой матрицы записывается число, определяющее наличие связи от вершины-строки к вершине-столбцу (либо наоборот).

В теории графов транспортная сеть — ориентированный граф G = (V, E), в котором каждое ребро (u, v) ∈ E имеет неотрицательную пропускную способность c (u, v) ⩾ 0 и поток f(u,v). Выделяются две вершины: источник s и сток t такие, что любая другая вершина сети лежит на пути из s в t, при этом s ≠ t. Транспортная сеть может быть использована для моделирования, например, дорожного трафика.

Алгори́тм Де́йкстры (англ. Dijkstra’s algorithm) — алгоритм на графах, изобретённый нидерландским учёным Эдсгером Дейкстрой в 1959 году. Находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса. Алгоритм широко применяется в программировании, например, его используют протоколы маршрутизации OSPF и IS-IS.

2.3. Формализация задачи

Core – класс содержащий в себе матрицу смежности, и выводящий информацию по кратчайщему пути в консоль

Dijlstra – класс реализующий метод поиска кратчайщего пути.

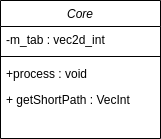


Рисунок 2.3.1 UML Диаграмма классов

2.4. Спецификация программы

Таблица 2.4.1 Описание методов класса Core

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
| procces | void | public | void | Создание матрицы смежноости и вывода информации в консоль |

2.5. Руководство пользователя

* + 1. Назначение программы

Данная программа позволяет найти кратчайщий путь в графе.

* + 1. Условие выполнения программы:

Для запуска программы необходим gcc/clang, cmake, ninja.

* + 1. Выполнение программы:

Программа разработана в виде консольного приложения. Она считывает матрицу смежности из метода Main и выводит кратчайщий путь.

* 1. **Руководство программиста**
     1. Запуск программы:

Для запуска программы необходим gcc/clang, cmake, ninja.

* + 1. Характеристика программы

Программа решает задачу в общем виде, находит кратчайщий путь с помощью алгоритма Дейкстры по матрице смежности, заданной в методе Main.

* + 1. Входные и выходные данные

Программа предполагает взаимодействие с пользователем напрямую. Новые данные вносит программист (изменяет матрицу смежности и ). Результатом служит вывод кратчайщего пути в консоль.

* 1. **Контрольный пример**

На рисунке 2.7.1 – работа программы для графа, данного в качестве контрольного примера:

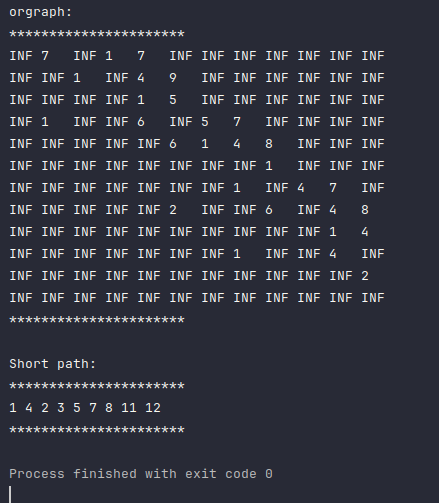


Рисунок 2.7.1 Работа программы

* 1. **Листинг программы**

**Core.cpp**

//  
// Created by xqula on 23.06.24.  
//  
#include "Core.h"  
#include "limits"  
#include "iostream"  
#include "set"  
#include "algorithm"  
using VecInt = std::vector<int>;  
using Vec2DInt = std::vector<VecInt>;  
  
namespace {  
 constexpr auto INF = std::numeric\_limits<int>::**max**( );  
  
 auto output\_path(const VecInt &*path*) -> void {  
 std::cout << "Short path:" << std::endl;  
 std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;  
 for (int i : *path*) {  
 std::cout << ++i << " ";  
 }  
 std::cout << std::endl;  
 std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;  
 }  
 auto output\_tab(const Vec2DInt &*tab*) -> void {  
 std::cout << "orgraph:" << std::endl;  
 std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;  
 for (const auto & i : *tab*) {  
 for (const int j : i) {  
 if(j == INF) std::cout << "INF" << " ";  
 else std::cout << j << " ";  
 }  
 std::cout << std::endl;  
 }  
 std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl << std::endl;  
 }  
}  
core::Core::Core() {  
 m\_tab = {  
 {INF, 7, INF, 1, 7, INF, INF, INF, INF, INF, INF, INF},  
 {INF, INF, 1, INF, 4, 9, INF, INF, INF, INF, INF, INF},  
 {INF, INF, INF, INF, 1, 5, INF, INF, INF, INF, INF, INF},  
 {INF, 1, INF, INF, 6, INF, 5, 7, INF, INF, INF, INF},  
 {INF, INF, INF, INF, INF, 6, 1, 4, 8, INF, INF, INF},  
 {INF, INF, INF, INF, INF, INF, INF, INF, 1, INF, INF, INF},  
 {INF, INF, INF, INF, INF, INF, INF, 1, INF, 4, 7, INF},  
 {INF, INF, INF, INF, INF, 2, INF, INF, 6, INF, 4, 8},  
 {INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, 1, 4},  
 {INF, INF, INF, INF, INF, INF, INF, 1, INF, INF, 4, INF},  
 {INF ,INF, INF, INF, INF, INF, INF ,INF, INF, INF, INF, 2},  
 {INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF}  
 };  
}  
auto core::Core::process() const -> void {  
 ::output\_tab(m\_tab);  
 const auto path = **getShortPath**(m\_tab);  
 ::output\_path(path);  
}  
auto core::Core::**getShortPath**(const Vec2DInt &*graph*) -> VecInt {  
 using Vec2DIntSize\_t = Vec2DInt::size\_type;  
 constexpr double INF = std::numeric\_limits<int>::**max**( );  
 const auto n = *graph*.size();  
 int s = 0; // стартовая вершина  
 std::vector<double> d(n, INF);  
 std::vector<int> p (n);  
 d[s] = 0;  
 std::set<std::pair<double,int>> q;  
 q.insert (std::make\_pair(d[s], s));  
 while (!q.empty()) {  
 const auto v = q.begin()->second;  
 q.erase (q.begin());  
 for (Vec2DIntSize\_t j=0; j<*graph*[v].size(); ++j) {  
 const auto to = j;  
 const auto len = *graph*[v][j];  
 if (d[v] + *graph*[v][j] < d[to]) {  
 q.erase (std::make\_pair(d[to], static\_cast<int>(to)));  
 d[to] = d[v] + len;  
 p[to] = v;  
 q.insert (std::make\_pair(d[to], static\_cast<int>(to)));  
 }  
 }  
 }  
 const auto t = *graph*.size()-1;  
 std::vector<int> path;  
 for (auto v=t; v!=s; v=p[v])  
 path.push\_back(static\_cast<int>(v));  
 path.push\_back (s);  
 std::ranges::reverse(path);  
 return path;  
}

**Core.h**

//  
// Created by xqula on 23.06.24.  
//  
#pragma once  
#include "vector"  
namespace core {  
 using VecInt = std::vector<int>;  
 using Vec2DInt = std::vector<VecInt>;  
  
 class Core {  
 public:  
 explicit Core();  
 auto process() -> void;  
  
 private:  
 Vec2DInt m\_tab;  
 };  
  
}

**3. Имитационное моделирование**

* 1. **Задание**

Вариант 10

Имитация работы речного порта

Речной порт имеет 4 места для причаливания (4 причала). В порт заходят различные речные суда — некоторые из них небольшие (60% от общего количества судов) и занимают только 1 причал, другие — средние (30% от общего количества судов) и занимают 2 причала, есть также большие суда (10% от общего количества судов), которые одновременно занимают три причала. Суда заходят в порт через каждые A±B часов. Небольшие суда обслуживаются в течение C±D часов, средние суда обслуживаются в течение E±F часов, большие суда обслуживаются G±H часов. При этом A<E<G. Если судну некуда причалить (не хватает нужного количества причалов), оно ждёт на рейде порта в очереди. Порт работает круглосуточно. Проанализировать процесс обслуживания 50 судов.

В программе должен быть предусмотрен вывод подсчитанной статистики. Результат сбора статистики должен быть выведен в текстовый файл.

* 1. **Теоретические сведения**

Имитационное моделирование заключается в создании процессов близких к реальным.

* 1. **Формализация задачи**

Factory – класс-фабрика создающий различные каробли.

Port – класс реализующий порт.

QueueShip – класс реализующий очередь на вход в порт.

Ship – класс реализующий корабль.

Random – класс реализующий генерация пвсевдослучайных чисел.

Writer – класс реализующий запись в файл.

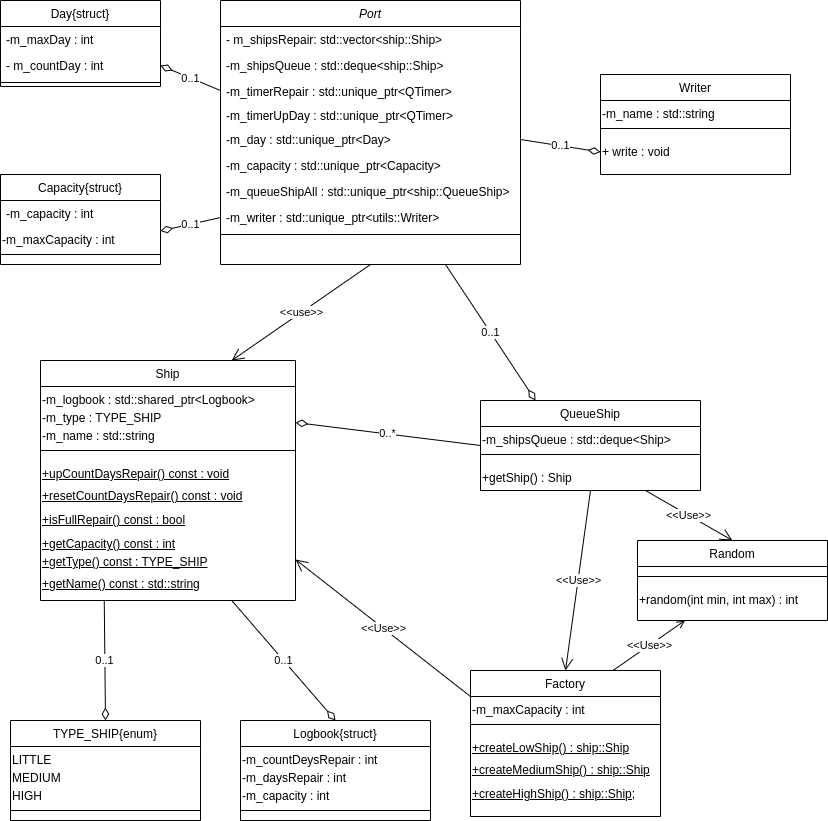


Рисунок 3.3.1 UML Диаграмма классов

* 1. **Спецификация программы**

Таблица 3.4.1 Описание методов класса Factory

| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
| --- | --- | --- | --- | --- |
| createHighShip | ship::Ship | public | void | Создание большого коробля |
| createMediumShip | ship::Ship | public | void | Создание среденго коробля |
| createLowShip | ship::Ship | public | void | Создание маленько коробля |

Таблица 3.4.2 Описание методов классов QueueShip

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
| getShip | Ship | public | void | Возвращет из очереди первый корабль |

Таблица 3.4.3 Описание методов класса Writer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
| write | void | public | std::string\_view txt | Запись данных в файл |

Таблица 3.4.4 Описание методов класса Random

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
| random | void | public | int min, int max | Генерация пвседвослучайных числел |

Таблица 3.4.4 Описание методов класса Ship

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Методы | Возвращаемый тип | Модификатор доступа | Входные параметры | Назначение |
| upCountDaysRepair | Void | Public | Void | Добавляет время |
| resetCountDaysRepair | void | Public | void | Обнуляет время |
| isFullRepair | Void | Public | void | Проверяет полностью починили |
| getCapacity | int | Public | Void | Возвращает размер |
| getType | TYPE\_SHIP | Public | Void | Возвращает тип |
| getName | Std::string | public | void | Возвращает имя |

* 1. **Руководство пользователя**
     1. **Назначение программы**

Данная программа позволяет моделировать работу речного порта.

* + 1. **Условие выполнения программы:**

Для запуска программы необходим gcc/clang, cmake, ninja, qt5.15

* + 1. **Выполнение программы:**

Программа разработана с использованием Qt5.15. Пользователь запускает программу и види работу проектного отдела в лог файле output.txt.

* 1. **Руководство программиста**
     1. **Запуск программы:**

Для запуска программы необходим gcc/clang, cmake, ninja, qt5.15

* + 1. **Характеристика программы**

Программа выполняется с использованием Qt5.15. Программа имеет интуитивно понятный интерфейс, проста и доступна в использовании.

* + 1. **Входные и выходные данные**

Выходные данные записываются в лог файл output.txt.

* 1. **Контрольный пример**

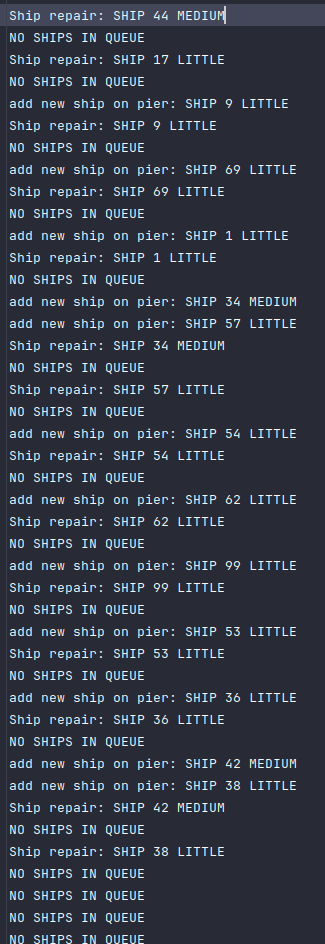
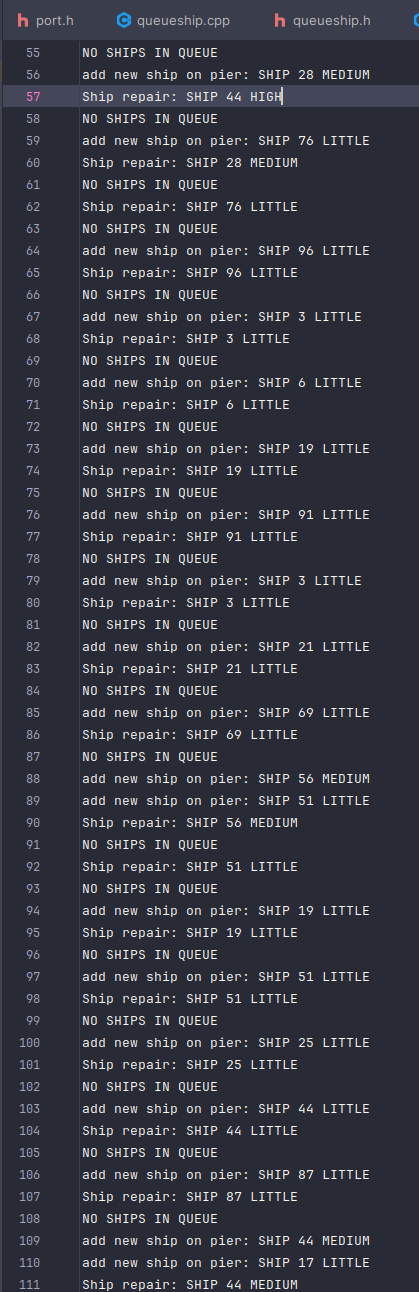
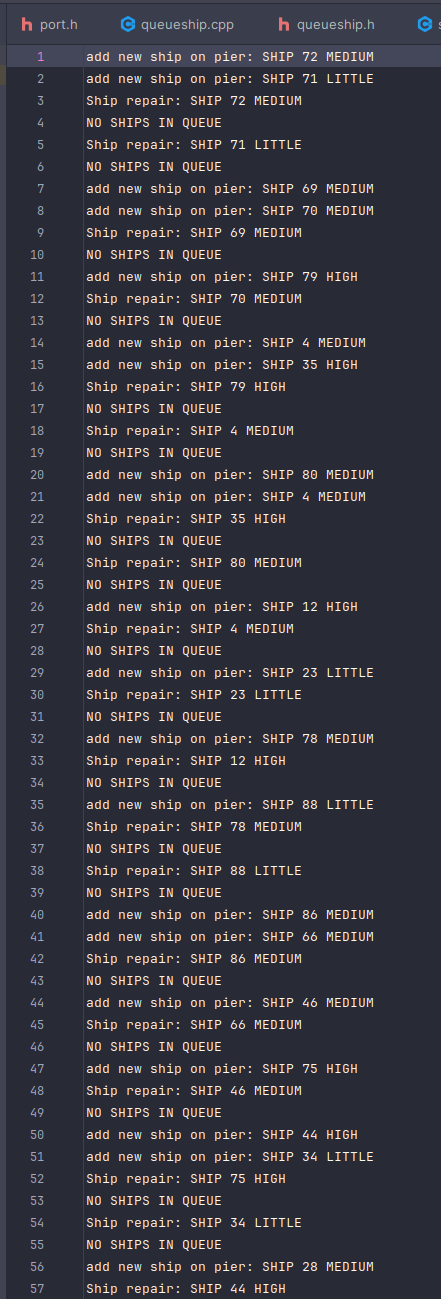


Рисунок 3.7.1 Работа программы

* 1. **Листинг программы**

Factory.cpp

#include "factory.h"  
#include "utils/random.h"  
auto factory::Factory::**createLowShip**() -> ship::Ship {  
 return {5, 3, ship::TYPE\_SHIP::LITTLE, std::to\_string(utils::Random::**random**(1, 100))};  
}  
auto factory::Factory::**createMediumShip**() -> ship::Ship {  
 return {10, 6, ship::TYPE\_SHIP::MEDIUM, std::to\_string(utils::Random::**random**(1, 100))};  
}  
auto factory::Factory::**createHighShip**() -> ship::Ship{  
 return {15, 9, ship::TYPE\_SHIP::HIGH, std::to\_string(utils::Random::**random**(1, 100))};  
}

Factroy.h

#pragma once  
#include "factory/factory.h"  
#include "ship/ship.h"  
namespace factory {  
 class Factory {  
 public:  
 static auto **createLowShip**() -> ship::Ship;  
  
 static auto **createMediumShip**() -> ship::Ship;  
  
 static auto **createHighShip**() -> ship::Ship;  
 };  
}

Port.h

//  
// Created by xqula on 27.06.24.  
//  
#include "port.h"  
#include "../ship/queueship.h"  
#include "iostream"  
namespace {  
 auto typeShipToString(const ship::TYPE\_SHIP *type*) -> std::string {  
 if(*type* == ship::TYPE\_SHIP::LITTLE) {  
 return "LITTLE";  
 } else if(*type* == ship::TYPE\_SHIP::MEDIUM) {  
 return "MEDIUM";  
 } else {  
 return "HIGH";  
 }  
 }  
}  
port::Port::Port(QObject \**parent*)  
 : QObject{*parent*},  
 m\_timerRepair(std::make\_unique<QTimer>(nullptr)),  
 m\_timerUpDay(std::make\_unique<QTimer>(nullptr)),  
 m\_day(std::make\_unique<Day>()),  
 m\_capacity(std::make\_unique<Capacity>()),  
 m\_queueShipAll(std::make\_unique<ship::QueueShip>()),  
 m\_writer(std::make\_unique<utils::Writer>("../output.txt"))  
{  
 m\_timerRepair->setInterval(100);  
 m\_timerUpDay->setInterval(100);  
 m\_day->m\_maxDay = 7;  
 m\_capacity->m\_maxCapacity = 9;  
  
 **connect**(m\_timerUpDay.get(), &QTimer::timeout, this, [this]() {  
 m\_day->m\_countDay++;  
 if(m\_day->m\_countDay == m\_day->m\_maxDay) {  
 m\_day->m\_countDay = 0;  
 try {  
 if(const auto ship = m\_queueShipAll->getShip();  
 ship.getCapacity() + m\_capacity->m\_capacity <= m\_capacity->m\_maxCapacity && m\_shipsQueue.empty()) {  
 m\_writer->write("add new ship on pier: " + ship.getName() + " " + typeShipToString(ship.getType()));  
 m\_shipsRepair.emplace\_back(ship);  
 }else {  
 m\_writer->write("add new ship on queue: " + ship.getName() + " " + typeShipToString(ship.getType()));  
 m\_shipsQueue.emplace\_back(ship);  
 }  
 }catch (std::logic\_error &e) {  
 m\_writer->write(e.what());  
 }  
 }  
 });  
  
 **connect**(m\_timerRepair.get(), &QTimer::timeout, this, [this]() {  
 for(const auto &ship : m\_shipsRepair) {  
 ship.upCountDaysRepair();  
 if(ship.isFullRepair()) {  
 m\_writer->write( "Ship repair: " + ship.getName() + " " + typeShipToString(ship.getType()));  
 m\_capacity->m\_capacity -= ship.getCapacity();  
 std::erase\_if(m\_shipsRepair, [](const ship::Ship &*shipTmp*) {  
 return *shipTmp*.isFullRepair();  
 });  
 if(m\_shipsQueue.empty()) {  
 m\_writer->write( "NO SHIPS IN QUEUE");  
 return;  
 }  
 if(m\_shipsQueue.front().getCapacity() + m\_capacity->m\_capacity <= m\_capacity->m\_maxCapacity) {  
 m\_writer->write( "add new ship on pier: " + ship.getName() + " " + typeShipToString(m\_shipsQueue.front().getType()));  
 m\_shipsRepair.emplace\_back(m\_shipsQueue.front());  
 m\_shipsQueue.pop\_front();  
 }  
 }  
 }  
 });  
  
 m\_timerRepair->start();  
 m\_timerUpDay->start();  
}

Port.cpp

#pragma once  
#include "ship/ship.h"  
#include "ship/queueship.h"  
#include "utils/writer.h"  
#include "deque"  
#include "vector"  
#include "memory"  
#include "QTimer"  
#include "QObject"  
namespace port {  
 struct Day {  
 int m\_maxDay;  
 int m\_countDay;  
 };  
 struct Capacity {  
 int m\_capacity;  
 int m\_maxCapacity;  
 };  
 class Port final : QObject {  
 Q\_OBJECT  
 public:  
 explicit Port(QObject \**parent* = nullptr);  
 private:  
 std::vector<ship::Ship> m\_shipsRepair;  
 std::deque<ship::Ship> m\_shipsQueue;  
 std::unique\_ptr<QTimer> m\_timerRepair;  
 std::unique\_ptr<QTimer> m\_timerUpDay;  
 std::unique\_ptr<Day> m\_day;  
 std::unique\_ptr<Capacity> m\_capacity;  
 std::unique\_ptr<ship::QueueShip> m\_queueShipAll;  
 std::unique\_ptr<utils::Writer> m\_writer;  
 };  
}

QueueShip.cpp

//  
// Created by xqula on 27.06.24.  
//  
#include "queueship.h"  
#include "../factory/factory.h"  
#include "../utils/random.h"  
namespace {  
 auto createShip(  
 int &*countShip*,  
 const int *maxCountChip*,  
 const ship::TYPE\_SHIP *type*,  
 std::deque<ship::Ship> &*shipsQueue*,  
 bool &*isAdd*)  
 -> void  
 {  
 if(*countShip* < *maxCountChip*) {  
 if(*type* == ship::TYPE\_SHIP::LITTLE) {  
 *shipsQueue*.push\_back(factory::Factory::**createLowShip**());  
 }else if(*type* == ship::TYPE\_SHIP::MEDIUM) {  
 *shipsQueue*.push\_back(factory::Factory::**createMediumShip**());  
 }else {  
 *shipsQueue*.push\_back(factory::Factory::**createHighShip**());  
 }  
 *countShip*++;  
 *isAdd* = true;  
 }  
 }  
 auto createQueueShip() -> std::deque<ship::Ship> {  
 std::deque<ship::Ship> shipsQueue;  
 constexpr int maxShips = 50;  
 constexpr int maxLittleShip = maxShips \* 0.6;  
 constexpr int maxMediumShip = maxShips \* 0.3;  
 constexpr int maxHighShip = maxShips \* 0.1;  
 int countLittleShip = 0;  
 int countMediumShip = 0;  
 int countHighShip = 0;  
 for(int i = 0; i < maxShips; i++) {  
 bool isAdd = false;  
 if(const int randNum = utils::Random::**random**(1, 3);  
 randNum == 1)  
 {  
 createShip(  
 countLittleShip,  
 maxLittleShip,  
 ship::TYPE\_SHIP::LITTLE,  
 shipsQueue,  
 isAdd);  
 } else if(randNum == 2) {  
 createShip(  
 countMediumShip,  
 maxMediumShip,  
 ship::TYPE\_SHIP::MEDIUM,  
 shipsQueue,  
 isAdd);  
 } else if(randNum == 3) {  
 createShip(  
 countHighShip,  
 maxHighShip,  
 ship::TYPE\_SHIP::HIGH,  
 shipsQueue,  
 isAdd);  
 }  
 if(not isAdd) {  
 createShip(  
 countLittleShip,  
 maxLittleShip,  
 ship::TYPE\_SHIP::LITTLE,  
 shipsQueue,  
 isAdd);  
 if(isAdd) continue;  
 createShip(  
 countMediumShip,  
 maxMediumShip,  
 ship::TYPE\_SHIP::MEDIUM,  
 shipsQueue,  
 isAdd);  
 if(isAdd) continue;  
 createShip(  
 countHighShip,  
 maxHighShip,  
 ship::TYPE\_SHIP::HIGH,  
 shipsQueue,  
 isAdd);  
 }  
 }  
 return shipsQueue;  
 }  
}  
ship::QueueShip::QueueShip():  
 m\_shipsQueue(createQueueShip())  
{  
}  
auto ship::QueueShip::getShip() -> Ship {  
 if(m\_shipsQueue.empty()) {  
 throw std::logic\_error("NO SHIPS IN QUEUE");  
 }  
 const auto ship = m\_shipsQueue.front();  
 m\_shipsQueue.pop\_front();  
 return ship;  
}

QueueShip.h

#pragma once  
#include <deque>  
#include "ship.h"  
namespace ship{  
 class QueueShip {  
 public:  
 explicit QueueShip();  
 auto getShip() -> Ship;  
 private:  
 std::deque<Ship> m\_shipsQueue;  
 };  
}

Ship.cpp

//  
// Created by xqula on 27.06.24.  
//  
#include "ship.h"  
ship::Ship::Ship(int *daysRepair*, int *capacity*, const TYPE\_SHIP *type*, const std::string &*name*)  
 : m\_logbook(std::make\_shared<Logbook>(*daysRepair*, 0, *capacity*)),  
 m\_type(*type*),  
 m\_name(std::move("SHIP " + *name*))  
{  
}  
auto ship::Ship::upCountDaysRepair() const -> void {  
 m\_logbook->m\_countDeysRepair++;  
}  
auto ship::Ship::resetCountDaysRepair() const -> void {  
 m\_logbook->m\_countDeysRepair = 0;  
}  
auto ship::Ship::isFullRepair() const -> bool {  
 return m\_logbook->m\_countDeysRepair >= m\_logbook->m\_daysRepair;  
}  
auto ship::Ship::getCapacity() const -> int {  
 return m\_logbook->m\_capacity;  
}  
auto ship::Ship::getType() const -> TYPE\_SHIP {  
 return m\_type;  
}  
auto ship::Ship::getName() const -> std::string {  
 return m\_name;  
}

Ship.h

#pragma once  
#include <memory>  
namespace ship {  
 struct Logbook {  
 int m\_daysRepair;  
 int m\_countDeysRepair;  
 int m\_capacity;  
 };  
  
 enum class TYPE\_SHIP {  
 LITTLE = 1,  
 MEDIUM = 2,  
 HIGH = 3  
 };  
  
 class Ship {  
 public:  
 Ship(int *daysRepair*, int *capacity*, TYPE\_SHIP *type*, const std::string &*name*);  
 ~Ship() = default;  
 auto upCountDaysRepair() const -> void;  
 auto resetCountDaysRepair() const -> void;  
 [[nodiscard]] auto isFullRepair() const -> bool;  
 [[nodiscard]] auto getCapacity() const -> int;  
 [[nodiscard]] auto getType() const -> TYPE\_SHIP;  
 [[nodiscard]] auto getName() const -> std::string;  
 private:  
 std::shared\_ptr<Logbook> m\_logbook;  
 TYPE\_SHIP m\_type;  
 std::string m\_name;  
 };  
}

Random.cpp

#include "random.h"  
#include "random"  
auto utils::Random::**random**(const int *min*, const int *max*) -> int {  
 std::random\_device rd; // non-deterministic generator  
 std::mt19937 gen(rd()); // to seed mersenne twister.  
 std::uniform\_int\_distribution<> dist(*min*,*max*); // define the range  
 return dist(gen); // generate numbers  
}

Random.h

#pragma once  
namespace utils {  
 class Random {  
 public:  
 static auto **random**(int *min*, int *max*) -> int;  
 };  
}

Write.h

#pragma once  
#include "string"  
namespace utils {  
 class Writer {  
 public:  
 explicit Writer(std::string *name*);  
 ~Writer() = default;  
 auto write(std::string\_view *txt*) const -> void;  
  
 private:  
 std::string m\_name;  
 };  
}

Write.cpp

#include "writer.h"  
#include <fstream>  
#include <utility>  
utils::Writer::Writer(std::string *name*) : m\_name(std::move(*name*)) {}  
auto utils::Writer::write(const std::string\_view *txt*) const -> void {  
 std::ofstream out;  
 out.open(m\_name, std::ios\_base::**app**);  
 if (out.is\_open())  
 {  
 out << *txt* << std::endl;  
 }else {  
 throw std::runtime\_error("Can't open file" + m\_name);  
 }  
 out.close();  
}

**заключение**

В результате выполнения курсовой работы были разработаны 3 программы, основанные на принципах объектно-ориентированного подхода. Были закреплены навыки работы с файлами, Qt, интерфейсами, абстрактными классами. Использовались принципы инкапсуляции, наследования, полиморфизма, агрегации. Получен опыт в разработке приложений, моделирующих непостоянные процессы.

**список использованных источников**

1. Поздняков, Рыбин – Дискретная математика (дата обращения: 24.06.2024)
2. Moodle – источник заданий URL: vec.etu.ru/moodle (дата обращения 24.06.2024)
3. Документация по языку C++ URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 24.05.2024)
4. Статья с информацией о UML-диаграммах классах URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-aggregation-vs-composition/
5. Документация по UML-диаграммам URL: https://www.uml-diagrams.org/