**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

**Кафедра автоматизації проектування  
енергетичних процесів і систем**

***КУРСОВА РОБОТА***

**з дисципліни: «Об’єктно-орієнтоване програмування - 1»**

**тема:**

**«Створення моделей об’єктів реального світу на мові C++»**

|  |  |
| --- | --- |
| **Керівник – Карпенко С.Г.** | **Виконав Пархоменко В.Р.** |
| **Допущено до захисту** | **Студент 2-го курсу** |
| **\_\_ \_\_\_\_\_\_\_\_ 2019 р.** | **Групи ТВ-з81** |
| **Захищено з оцінкою** | **залікова книжка № ТА-61350** |
| **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  |

***2019***

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Кафедра автоматизації проектування  
енергетичних процесів і систем

**КУРСОВА РОБОТА**

з дисципліни «Об’єктно-орієнтоване програмування - 1»

(назва дисципліни)

на тему:

«Створення моделей об’єктів реального світу на мові C++»

Студента групи ТВ-81

спеціальність **121 «Інженерія програмного забезпечення»**

освітня програма **«Інженерія програмного забезпечення розподілених систем»**

Пархоменко В.Р.

(прізвище та ініціали)

Керівник доцент, канд. фіз.-мат. наук Карпенко С.Г.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_ доц., к. ф.-м. н. Карпенко С.Г.

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_ доц., к.е.н. Левченко Л.О.

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_ доц., к.т.н. Шаповалова С.І.

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали

Київ - 2019 рік

**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет ТЕПЛОЕНЕРГЕТИЧНИЙ ФАКУЛЬТЕТ

( повна назва)

Кафедра автоматизації проектування енергетичних процесів та систем

(повна назва)

Рівень вищої освіти **перший (бакалавр)**

спеціальність **121 «Інженерія програмного забезпечення»**

(шифр і назва)

Освітня програма «**Інженерія програмного забезпечення розподілених систем»**

(шифр і назва)

ЗАВДАННЯ

**НА КУРСОВУ РОБОТУ СТУДЕНТУ**

Пархоменко Валентину Романовичу

(прізвище, ім’я, по-батькові)

1. Тема роботи «Створення моделей об’єктів реального світу на мові C++»

керівник курсової роботи –

Карпенко Станіслав Григорович, канд. фіз.-мат. наук, доцент

( прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи – 24 грудня 2019 р.

3. Вихідні дані до проекту (роботи): мова C++, модель об’єкту реального світу.

4. Зміст пояснювальної записки курсової роботи (перелік питань, які потрібно розробити) – Розробити та обґрунтувати ієрархічну модель класів для опису об’єкту реального світу, використовуючи інструментальні можливості мови C++, зокрема успадковування, інкапсуляцію, перевантажені операції, шаблони функцій, запис у двійкові файли та обробку виключень.

5. Перелік графічного матеріалу –

графічне зображення ієрархічної моделі класів з урахуванням як успадковування, так й включення; відображення значень основних параметрів реальних об’єктів у вигляді таблиці.

6. Дата видачі завдання – 24 вересня 2019 р.

#### КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів виконання курсової роботи | Строк виконання етапів роботи | Примітка |
|  | Побудова ієрархічної моделі класів | 15.10.2019 |  |
|  | Розробка та застосування інструментальних засобів обробки даних | 19.11.2019 |  |
|  | Написання розрахунково-пояснювальної записки | 24.12.2019 |  |

**Студент \_\_\_\_\_\_\_\_\_ Пархоменко В.Р.**

( підпис ) (прізвище та ініціали)

**Керівник курсової роботи \_\_\_\_\_\_\_\_\_ Карпенко С.Г.**

( підпис ) (прізвище та ініціали)

# АНОТАЦІЯ

Виконуючи дану роботу, я дослідив базові принципи об’єктно-орієнтованого програмування. На основі цих принципів було побудовано ієрархічну модель класів, що відображає модель реальних об’єктів. Було визначено поля і методи об’єктів розроблюваного похідного класу а також всіх прямих та опосередкованих базових класів.

При побудові ієрархічної моделі було використано як успадковування, агрегацію так і інші важливі елементи мови С++, такі як перевантаження операцій, обробка виключних ситуацій, запис до текстових файлів, зчитування з них інформаціїї.

# ЗМІСТ

[АНОТАЦІЯ 5](#_Toc30464826)

[ЗМІСТ 6](#_Toc30464829)

[ВСТУП 5](#_Toc30464830)

[1. ТЕОРЕТИЧНІ ВІДОМОСТІ 8](#_Toc30464831)

[1.1 Визначення та історія ООП 8](#_Toc30464832)

[1.2 Модифікатори доступу 10](#_Toc30464833)

[1.3 Висновки 11](#_Toc30464834)

[2. ІЄРАРХІЯ КЛАСІВ 12](#_Toc30464835)

[2.1 Опис класів 12](#_Toc30464836)

[2.2 Обґрунтування вибору перевантажених операцій 13](#_Toc30464837)

[2.3 Обґрунтування вибору встановлення обробки виключних ситуацій 13](#_Toc30464838)

[ВИСНОВКИ 14](#_Toc30464839)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 15](#_Toc30464840)

[ДОДАТОК 1. Текст програми 16](#_Toc30464841)

[ДОДАТОК 2. Опис програми 25](#_Toc30464842)

Розробив

Карпенко С.Г.

Пархоменко В.Р.

# ВСТУП

Об'є́ктно-орієнто́ване програмува́ння (ООП) — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають три основні концепції: інкапсуляція,успадкування та поліморфізм. Одною з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в 1960-тих роках, вона не мала широкого застосування до 1990-тих, коли розвиток комп'ютерів та комп'ютерних мереж дозволив писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм. Сьогодні багато мов програмування або підтримують ООП (Lua) або ж є цілком об'єктно-орієнтованими (зокрема, Java, C#, C++, Python, PHP, Ruby та Objective-C, ActionScript 3, Swift, Vala).

Об'єктно-орієнтоване програмування - це метод програмування, заснований на поданні програми у вигляді сукупності взаємодіючих об'єктів, кожен з яких є екземпляром певного класу, а класи є членами певної ієрархії наслідування. Програмісти спочатку пишуть клас, а на його основі при виконанні програми створюються конкретні об'єкти (екземпляри класів). На основі класів можна створювати нові, які розширюють базовий клас і таким чином створюється ієрархія класів.

Кожний стиль програмування має свою концептуальну базу. Для ОО стилю такою базою є об’єктна модель. Її побудова основана на використанні 3-х основних концепцій.

- інкапсуляція

- поліморфізм

- наслідування

Інкапсуляція – механізм, який поєднує дані та методи, що обробляють ці дані і захищає і те і інше від зовнішнього впливу або не вірного використання. Коли і методи і данні об’єднуються таким чином — створюється об’єкт.

В середині об’єкта дані і методи можуть мати різні ступені відкритості. Як правило відкриті члени класу використ. Для того, що б забезпечити інтерфейс, що контролюється з його закритою частиною.

Наслідування – процес, завдяки якому один об’єкт може придбати властивості іншого, тобто наслідувати властивість іншого обєкту і додавати риси характерні тільки для нього самого.

В якості прикладів можна навести класифікацію будь-яких об’єктів, яка дає відповідь на питання в чому схожість об’єкта даного класу з інш. Об’єктом і в чому різниця. Наслідування забезпечує загальність функції в той час припускаючи стільки особливостей, скільки необхідно. Таким чином наслідування виконує в ООП такі функції:

- моделює концептуальну структуру предметної області

- економить опис, дозволяючи використовувати функції багатократно для завдань різних класів.

- забезпечує покрокове програмування великих систем, шляхом багатократної конкретизації класів.

Поліморфізм – властивість, яка дозволяє одне і те саме ім’я використовувати для вирішення декількох технічно різних задач, тобто основною метою поліморфізму є використання одного імені для задання загальних класу дій. На практиці це значить спроможність об’єктів вибирати внутрішній метод або процедуру, виходячи із типу даних, прийнятих в повідомленні.

Приклад, об’єкт з імям “print” отримує повідомлення в якому виконуються дії по друку таких типів данних, як двійкове ціле число, двійкове число з плаваючою комою і символьний рядок. В залежності від того типу даних, які працюють в команді буде підібраний тип (даних) формату для друку.

Перевагою поліморфізму є те, що:

- він допомагає зменшити складність програм

- дозволяє використання одного інтерфейсу для єдиного класу дій, а вибір конкретної дії передається компілятору. Таким чином, поліморфізм означає присвоювання певній дії одної дії або позначення, яке спільно використовується об’єктами різних типів. При цьому кожний об’єкт реалізує дії способом, що відповідає його типу.

# 1. ТЕОРЕТИЧНІ ВІДОМОСТІ

## 1.1 Визначення та історія ООП

Об'єктно-орієнтоване програмування — це метод програмування, заснований на поданні програми як сукупності взаємодіючих об'єктів, кожен з яких є екземпляром певного класу, а класи є членами певної ієрархії наслідування. Програмісти спочатку пишуть клас, а на його основі під час виконання програми створюються конкретні об'єкти (екземпляри класів). На основі класів можна створювати нові, які розширюють базовий клас і таким чином створюється ієрархія класів.

На думку Алана Кея, розробника мови Smalltalk, якого вважають одним з «батьків-засновників» ООП, об'єктно-орієнтований підхід полягає в наступному наборі основних принципів:

* Все є об'єктами.
* Всі дії та розрахунки виконуються шляхом взаємодії (обміну даними) між об'єктами, під час якої один об'єкт потребує, щоб інший об'єкт виконав деяку дію. Об'єкти взаємодіють, надсилаючи і отримуючи повідомлення. Повідомлення — це запит на виконання дії, доповнений набором аргументів, які можуть знадобитися під час виконання дії.
* Кожен об'єкт має незалежну пам'ять, яка складається з інших об'єктів.
* Кожен об'єкт є представником (екземпляром, примірником) класу, який виражає загальні властивості об'єктів.
* У класі задається поведінка (функціональність) об'єкта. Таким чином усі об'єкти, які є екземплярами одного класу, можуть виконувати одні й ті ж самі дії.
* Класи організовані у єдину деревоподібну структуру з загальним корінням, яка називається ієрархією успадкування. Пам'ять та поведінка, зв'язані з екземплярами деякого класу, автоматично доступні будь-якому класу, розташованому нижче в ієрархічному дереві.

Таким чином, програма є набором об'єктів, що мають стан та поведінку. Об'єкти взаємодіють використовуючи повідомлення. Будується ієрархія об'єктів: програма в цілому — це об'єкт, для виконання своїх функцій вона звертається до об'єктів що містяться у ньому, які у свою чергу виконують запит шляхом звернення до інших об'єктів програми. Звісно, щоб уникнути нескінченної рекурсії у зверненнях, на якомусь етапі об'єкт трансформує запит у повідомлення до стандартних системних об'єктів, що даються мовою та середовищем програмування. Стійкість та керованість системи забезпечуються за рахунок чіткого розподілу відповідальності об'єктів (за кожну дію відповідає певний об'єкт), однозначного означення інтерфейсів міжоб'єктної взаємодії та повної ізольованості внутрішньої структури об'єкта від зовнішнього середовища (інкапсуляції).

ООП виникло в результаті розвитку ідеології процедурного програмування, де дані і підпрограми (процедури, функції) їх обробки формально не пов'язані. Для подальшого розвитку об'єктно-орієнтованого програмування велике значення мають поняття події (так зване подієво-орієнтоване програмування) і компоненти (компонентне програмування, КОП).

Формування КОП від ООП відбулося, так само як формування модульного від процедурного програмування: процедури сформувалися в модулі - незалежні частини коду до рівня збірки програми, так об'єкти сформувалися в компоненти - незалежні частини коду до рівня виконання програми. Взаємодія об'єктів відбувається за допомогою повідомлень. Результатом подальшого розвитку ООП, мабуть, буде агентно-орієнтоване програмування, де агенти - незалежні частини коду на рівні виконання. Взаємодія агентів відбувається за допомогою зміни середовища, в якій вони знаходяться.

Мовні конструкції, конструктивно не пов'язані безпосередньо з об'єктами, але необхідні їм для їх безпечної (виняткові ситуації, перевірки) та ефективної роботи, інкапсулюються від них в аспекти (в аспектно - орієнтованому програмуванні). Суб'єктно-орієнтоване програмування розширює поняття об'єктів шляхом забезпечення більш уніфікованого і незалежної взаємодії об'єктів. Може бути перехідною стадією між ООП та агентним програмуванням в частині самостійної їх взаємодії.

Першою мовою програмування, в якій були запропоновані принципи об'єктної орієнтованості, була Симула. На момент своєї появи (в 1967 році), ця мова програмування запропонувала революційні ідеї: об'єкти, класи, віртуальні методи тощо, однак це все не було сприйнято сучасниками як щось грандіозне. Тим не менше, більшість концепцій були розвинені Аланом Кеєм та Деном Інгаллсом у мові Smalltalk. Саме вона стала першою широко поширеною об'єктно - орієнтованою мовою програмування.

Наразі кількість прикладних мов програмування (список мов), що реалізують об'єктно-орієнтовану парадигму, є найбільшим по відношенню до інших парадигм. В області системного програмування досі застосовується парадигма процедурного програмування, і загальноприйнятою мовою програмування є мова C. Хоча при взаємодії системного і прикладного рівнів операційних систем стали помітно впливати мови об'єктно-орієнтованого програмування. Наприклад, однією з найпоширеніших бібліотек мультиплатформового програмування є об'єктно-орієнтована бібліотека Qt, написана мовою C++.

## 1.2 Модифікатори доступу

В С ++ члени класу класифікуються відповідно до прав доступу на наступні три категорії: публічні (public), приватні (private) і захищені (protected). Будь-яка функція програми має доступ до публічних членам. Доступ до приватного члену мають тільки функції-члени класу або функції-друзі класу. Захищені члени аналогічні приватним членам. Різниця між ними з'являється тільки при спадкуванні класів.

Коли один клас успадковує інший, всі публічні члени базового класу стають публічними членами похідного класу. На противагу цьому приватні члени базового класу не доступні всередині похідного класу.

Public - доступ відкритий усім, хто бачить визначення даного класу.

Protected - доступ відкритий класів, похідних від даного. Тобто, похідні класи отримують вільний доступ до таких властивостей або методу. Всі інші класи такого доступу не мають.

Private - доступ відкритий самому класу (тобто функцій-членів даного класу) і друзям (friend) даного класу - як функцій, так і класами. Однак похідні класи не отримують доступу до цих даних зовсім. І всі інші класи такого доступу не мають.

Захищений член подібний приватному, за винятком механізму наслідування. При спадкуванні захищеного члена похідний клас також має до нього доступ. Таким чином, вказавши специфікатор доступу protected, можна дозволити використовувати член всередині ієрархії класів і заборонити доступ до нього ззовні цієї ієрархії.

## 1.3 Висновки

В цьому розділі описані загальні теоретичні відомості про об’єктно-орієнтоване програмування та його історію.

# 2. ІЄРАРХІЯ КЛАСІВ

## 2.1 Опис класів

У роботі реалізовано шєрархію класів згідно з наступним завданням:

Таблиця 2.1 Завдання

|  |  |
| --- | --- |
| 6 | Дата {Рік, Місяць, День};  Особа {Прізвище, Ім’я, Дата (народження)};  Назва посади {Лаборант, старший лаборант, інженер, старший інженер, молодший науковий співробітник, старший науковий співробітник, начальник};  Посада {Назва посада, оклад посади};  Співробітник {Особа, Посада, Дата (зарахування)};  Лабораторія {Назва (лабораторії), Завідувач лабораторії, Список співробітників};  Відділ {Назва, Керівник відділу, Список лабораторій}.  Середнє значення кількості співробітників в лабораторіях та середнє значення окладів співробітників в лабораторіях. |

Було стоврено такі класи відповідно до завдання.

Клас **Date** є базовим, має три значення типу **int.**

Клас **Person** є базовим, має два значення типу **string**, та об’єкт класу **Date.**

Клас **PositionNames** має три значення **int** та вектор значень **string**.

Клас **Position** наслідує **PositionNames**, і має власне поле **int**.

Клас **Employee** наслідує класи **Person**, **Position** та **Date**.

Клас **Laboratory** є базовим, має значення типу **string**, об’єкт класу **Person** та вектор обє’ктів класу **Employee.**

Клас **Department** є базовим, має значення типу **string**, об’єкт класу **Person** та вектор обє’ктів класу **Employee.**

Для всіх внутрішнів значень класів встановлені модифікатори доступу **private**, доступ до яких може бути отриманий лише за допомогою відкритих методів класу з модифікатором **public**.

## 2.2 Обґрунтування вибору перевантажених операцій

При написанні курсової роботи було використано перевантаження операторів у класах **Date, Person, PositionNames, Laboratory**, що дозволяють отримати простіший доступ до певних даних класів або привести їх до іншого типу без використання додаткових методів.

## 2.3 Обґрунтування вибору встановлення обробки виключних ситуацій

У виконанні даної курсової роботи обробка виключних ситуацій була використана для обробки власноруч створеного виключення при спробі встановити від’ємний оклад посади.

# ВИСНОВКИ

В даній роботі була наведена теорія і реалізовано на практиці основи об’єктно-орієнтованого програмування, а саме: інкапсуляція, успадковування, поліморфізм, абстракція.

Згідно з поставленим завданням було реалізовано:

* наслідування класів;
* використання закритих модифікаторів доступу для внутрішніх полів класів;
* публічні методи змінення закритих полів класів;
* перевантаження операторів;
* застосування перевантажених операторів;
* визначення перевантаженого оператора індексування для внутрішнього динамічного масиву;
* обробка виключних ситуацій;
* застосування виключень;
* виведення даних у вигляді таблиці;
* запис даних у текстовий файл;
* зчитування даних з текстового файлу;

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Галузевий стандарт вищої освіти: ОКХ бакалавра напрямку підготовки 0804 – Комп’ютерні науки. – К.: МОН України, 2004.
2. Кендалл С. Унифицированный процесс. Основные концепции.–М.;–СПб. Киев.–2002..
3. Гради Буч, Роберт А. Максимчук, Майкл У. Энгл, Бобби Дж. Янг, Джим Коналлен, Келли А. Хьюстон. Объектно-ориентированный анализ и проектирование с примерами приложений. М.: Вильямс, 2008.- 720с.
4. Лафоре Р. Объектно-ориентированное программирование в С++. Классика Computer Science. 4 изд.. – СПб.: Питер, 2003.
5. Бенджамин Пирс. Типы в языках программирования. — Добросвет, 2012. — 680 с.
6. Лаптев В.В. C++. Экспресс-курс. – СПб.: БХВ-Петербург, 2004. – 512 с.: ил.
7. Сидорова Н. М. Навчання інженерії програмного забезпечення – систематичний огляд літератури/ Н. М. Сидорова // Матеріали міжнародної науково- практичної конференції аспірантів і студентів «Інженерія програмного забезпечення 2011». [Електрон.ресурс].-Спосіб доступу: [http://www.nbuv.gov.ua/portal/ natural/Ipz/2011\_2/Sidorova.pdf].
8. Armstrong, Deborah J. (February 2006). The Quarks of Object-Oriented Development. Communications of the ACM 49 (2): 123–128. ISSN 0001-0782

# ДОДАТОК 1. Текст програми

#ifndef KURS\_CLASSES\_H

#define KURS\_CLASSES\_H

#include <string>

#include <vector>

#include "TextTable.h"

#include <fstream>

using namespace std;

class Date {

private:

int day, month, year;

public:

Date() {day = 01; month = 01; year = 2000;};

Date (int day, int month, int year) {

this->day = day;

this->month = month;

this->year = year;

}

void setDate(Date date) {

this->day = date.day;

this->month = date.month;

this->year = date.year;

}

int getDay() {

return day;

}

int getMonth() {

return month;

}

int getYear() {

return year;

}

void setDate(int day, int month, int year) {

this->day = day;

this->month = month;

this->year = year;

}

string getDate() {

return (to\_string(day).length()<2?("0"+to\_string(day)):(to\_string(day))) + "-" + (to\_string(month).length()<2?("0"+to\_string(month)):(to\_string(month))) + "-" + to\_string(year);

}

operator string() {

return getDate();

}

};

class Person {

private:

string name, surname;

Date birthDate;

public:

void setName(string name) {

this->name = name;

}

void setSurname(string surname) {

this->surname = surname;

}

void setBirthDate(Date birthDate) {

this->birthDate.setDate(birthDate);

}

string getPerson() {

return name + " " + surname + " " + birthDate.getDate();

}

string getName() {

return name;

}

string getSurname() {

return surname;

}

Date getBirthDate() {

return birthDate;

}

Person() {

name = "Sam";

surname = "Sepiol";

birthDate.setDate(9,5,2015);

}

Person(string name, string surname, Date birthDate) {

this->name = name;

this->surname = surname;

this->birthDate.setDate(birthDate);

}

Person(string name, string surname) {

this->name = name;

this->surname = surname;

}

operator string() {

return getPerson();

}

};

class PositionNames {

private:

int posIndex;

vector<string> posNames;

public:

void savePosNamesToFile() {

ofstream file;

file.open("posNames.txt");

for (int i = 0; i < posNames.size(); i++) {

file << posNames[i] << "\n";

}

file.close();

}

static void loadPosNamesFromFile() {

string line;

ifstream file ("posNames.txt");

if (file.is\_open())

{

while ( getline (file, line) )

{

cout << line << '\n';

}

file.close();

}

}

void printPosNames() {

TextTable t( '-', '|', '+' );

t.add( "index" );

t.add( "Pos name" );

t.endOfRow();

for (int i = 0; i < posNames.size(); i++) {

t.add(std::to\_string(i));

t.add(posNames[i]);

t.endOfRow();

}

t.setAlignment( 2, TextTable::Alignment::RIGHT );

std::cout << t;

}

vector<string> getPosNames() {

return posNames;

}

string getPos() {

return posNames[posIndex];

}

void setPos(int posIndex) {

this->posIndex = posIndex;

}

PositionNames() {

posIndex = 0;

posNames.emplace\_back("Лаборант");

posNames.emplace\_back("старший лаборант");

posNames.emplace\_back("iнженер");

posNames.emplace\_back("cтарший iнженер");

posNames.emplace\_back("молодший науковий спiвробiтник");

posNames.emplace\_back("cтарший науковий спiвробiтник");

posNames.emplace\_back("начальник");

}

string& operator[] (int index) {

return posNames[index];

}

};

struct InvalidSalary : public exception {

const char \* msg () const throw () {

return "Invalid employee salary exception (below zero).";

}

};

class Position : public PositionNames {

private:

int salary;

public:

int getSalary() {

return salary;

}

void setSalary(int salary) {

if(salary < 0) throw InvalidSalary();

this->salary = salary;

}

Position() {

setPos(0);

salary = 100;

}

Position(int salary, int posIndex) {

this->salary = salary;

setPos(posIndex);

}

};

class Employee : public Person , public Position , public Date {

public:

explicit Employee(Person person) {

setName(person.getName());

setSurname(person.getSurname());

setBirthDate(person.getBirthDate());

}

};

class Laboratory {

private:

string name;

Person zavLaboratory;

vector<Employee> Employees;

public:

void setName(string name) {

this->name = name;

}

string getName() {

return name;

}

void setZav(Person zav) {

zavLaboratory = zav;

}

Person getZav() {

return zavLaboratory;

}

vector<Person> getEmployees;

void addEmployee(const Employee& employee) {

Employees.emplace\_back(employee);

}

int avgSalary() {

int sumSalaries = 0;

for (auto i : Employees) {

sumSalaries += i.getSalary();

}

return sumSalaries / Employees.size();

}

int counEmployees() {

return Employees.size();

}

operator string() {

return name;

};

operator Person() {

return zavLaboratory;

}

Laboratory() {

name = "Lab name";

}

};

class Department {

private:

string name;

Person manager;

vector<Laboratory> labs;

public:

void setName(string name) {

this->name = name;

}

string getName() {

return name;

}

Person getManager() {

return manager;

}

void setManager(Person manager) {

this->manager = manager;

}

vector<Laboratory> getLabs() {

return labs;

}

void addLab(Laboratory lab) {

labs.emplace\_back(lab);

}

Department() {

name = "Dep name";

}

};

#endif //KURS\_CLASSES\_H

#include <iostream>

#include "Classes.h"

#include <fstream>

int main() {

cout << "Initialising new Date class object 'date' 9/5/1975\n";

Date date(9, 5,1975);

cout << "Printing 'date'\n";

cout << date.getDate() << endl;

cout << "Init 'pers1' and 'pers2' of class Person, 'lab' of class Laboratory\n";

Person pers1, pers2, pers3;

Laboratory lab;

cout << "Creatine 'emp1' and 'emp2' of class Employee with 'pers1' and 'pers2'\n";

Employee emp1(pers1);

Employee emp2(pers2);

Employee emp3(pers3);

cout << "Setting birth date for 'pers1' with 'date' object\n";

pers1.setBirthDate(date);

cout << "Printing 'pers1' executing a 'getPerson()' method of Person class\n";

cout << pers1.getPerson() << endl;

cout << "Trying to set salary '-150' for Employee 'emp1'\n";

try {

emp1.setSalary(-150);

cout << "Successful salary setting for Employee object\n";

} catch (InvalidSalary e) {

cout << "Caught exception: " << e.msg() << endl;

}

cout << "Trying to set salary '1000' for Employee 'emp2'\n";

try {

emp2.setSalary(1000);

cout << "Successful salary setting for Employee object\n";

} catch (InvalidSalary e) {

cout << "Caught exception: " << e.msg() << endl;

}

cout << "Trying to set salary '758' for Employee 'emp3'\n";

try {

emp3.setSalary(758);

cout << "Successful salary setting for Employee object\n";

} catch (InvalidSalary e) {

cout << "Caught exception: " << e.msg() << endl;

}

cout << "Adding 'emp1', 'emp2' and 'emp3' to 'lab' object\n";

lab.addEmployee(emp1);

lab.addEmployee(emp2);

lab.addEmployee(emp3);

cout << "Get average salary for 'lab' with executing 'avgSalary()' method\n";

cout << lab.avgSalary() << endl;

cout << "Executing 'printPosNames()' for 'emp1' ti print a table-view for position names array\n";

emp1.printPosNames();

cout << "Executing 'savePosNamesToFile()' for 'emp1' to save position names to text file\n";

try {

emp1.savePosNamesToFile();

cout << "Successful save data to text file\n";

} catch (exception e) {

cout << "Caught some exception during saving a file. Exception: " << e.what() << endl;

}

cout << "Executing 'loadPosNamesFromFile()' for 'emp1' to load position names from text file\n";

emp1.loadPosNamesFromFile();

return 0;

}

#

pragma once

#include class TextTable{ public : enum class Alignment{ LEFT, RIGHT }; typedef std::vector<std::string > Row; TextTable(char horizontal = '-', char vertical = '|', char corner = '+') : \_horizontal(horizontal), \_vertical(vertical), \_corner(corner){} void setAlignment(unsigned i, Alignment alignment){ \_alignment[i] = alignment;

}

Alignment alignment(unsigned i) const

{

return \_alignment[i];

}

char vertical() const

{

return \_vertical;

}

char horizontal() const

{

return \_horizontal;

}

void add(std::string

const &content)

{

\_current.push\_back(content);

}

void endOfRow()

{

\_rows.push\_back(\_current);

\_current.assign(0, "");

}

template void addRow(Iterator begin, Iterator end)

{

for (auto i = begin; i != end; ++i)

{

add(\*i);

}

endOfRow();

}

template void addRow(Container

const &container)

{

addRow(container.begin(), container.end());

}

std::vector < Row>

const &rows() const

{

return \_rows;

}

void setup() const

{

determineWidths();

setupAlignment();

}

std::string ruler() const

{

std::string result;

result += \_corner;

for (auto width = \_width.begin(); width != \_width.end(); ++width)

{

result += repeat(\*width, \_horizontal);

result += \_corner;

}

return result;

}

int width(unsigned i) const

{

return \_width[i];

}

private:

char \_horizontal;

char \_vertical;

char \_corner;

Row \_current;

std::vector<Row> \_rows;

std::vector<unsigned> mutable \_width;

std::map<unsigned, Alignment> mutable \_alignment;

static std::string repeat(unsigned times, char c)

{

std::string result;

for (; times > 0; --times)

result += c;

return result;

}

unsigned columns() const

{

return \_rows[0].size();

}

void determineWidths() const

{

\_width.assign(columns(), 0);

for (auto rowIterator = \_rows.begin(); rowIterator != \_rows.end(); ++rowIterator)

{

Row

const &row = \*rowIterator;

for (unsigned i = 0; i < row.size(); ++i)

{

\_width[i] = \_width[i] > row[i].size() ? \_width[i] : row[i].size();

}

}

}

void setupAlignment() const

{

for (unsigned i = 0; i < columns(); ++i)

{

if (\_alignment.find(i) == \_alignment.end())

{

\_alignment[i] = Alignment::LEFT;

}

}

}

};

inline std::ostream &operator<<<table.ruler() << "\n";

for (auto rowIterator = table.rows().begin(); rowIterator != table.rows().end(); ++rowIterator)

{

TextTable::Row

const &row = \*rowIterator;

stream << table.vertical();

for (unsigned i = 0; i < row.size(); ++i)

{

auto alignment = table.alignment(i) == TextTable::Alignment::LEFT ? std::left : std::right;

stream << std::setw(table.width(i)) << alignment << row[i];

stream << table.vertical();

}

stream << "\n";

stream << table.ruler() << "\n";

}

return stream;

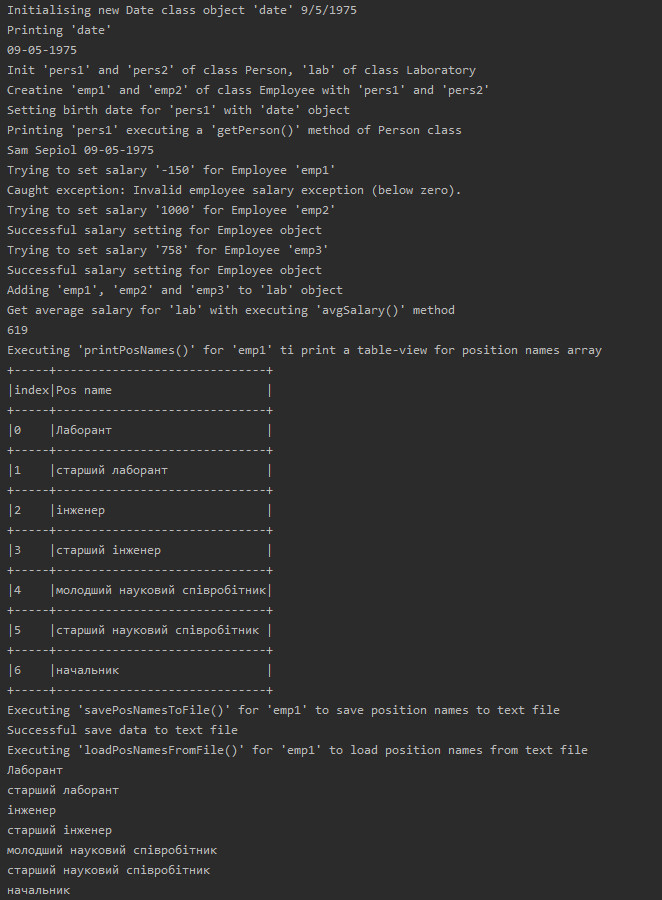
}

# ДОДАТОК 2. Опис програми

Програма була розроблена на мові С++ в програмному середовищі JetBrains CLion. Створений програмний код дозволяє отримувати дані з клавіатури абл з текстового файлу, дає можливість виводити інформацію на екран або в текстовий.

В програмі була реалізована базова демонстрація можливостей роботи з класами, розроблених у програмі.

Результатом вибору в програмі значень за замовчуванням отримуємо наступне віведення на екран (рисунок 1).

**

**Рисунок 1 – Результат роботи програми**