**КУРСОВА РОБОТА**

**з курсу Об’єктно-орієнтоване програмування**

**тема: Симуляція Автоматичної телефонної станції**

Спеціальність 6.050103 „Програмна інженерія”

Виконав студент

групи ІПЗ16-2бд

Доленко Т.Д.

Керівник роботи

доц. Попівщий В.І.

Оцінка

Члени комісії: доц. Попівщий В.І.

доц. Скрипник І.А.

проф. Вербицький В.Г.

Запоріжжя

2017

# Реферат

Об’єм роботи: 44 сторінки. 1 діаграма та 7 screenshots.

Кількість використаних джерел: 7. Див. роз. Список літератури.

Мета роботи: Задокументувати розробку симуляції автоматичної телефонної станції.

Ключові слова роботи: АТС, телефонна мережа, абонент, користувач, адміністратор, інтерфейс, клас, ООП, Windows, C#, Visual Studio.

# Зміст

[Реферат 2](#_Toc499385592)

[Зміст 3](#_Toc499385593)

[1 Вступ 4](#_Toc499385594)

[Результат роботи 4](#_Toc499385595)

[Опис та функції АТС 4](#_Toc499385596)

[Доцільність рішення 5](#_Toc499385597)

[1.1 Глосарій 5](#_Toc499385598)

[1.1 Опис наочної області 6](#_Toc499385599)

[1.2.1 Об'єктно-орієнтоване програмування 6](#_Toc499385600)

[1.2.2 Телефонні мережі загального користування 11](#_Toc499385601)

[1.2.2.1 Опис телефонної мережі загального користування 12](#_Toc499385602)

[1.2 Постановка задачі 13](#_Toc499385603)

[1.3 План робіт 15](#_Toc499385604)

[2 Вимоги до оточення 15](#_Toc499385605)

[2.1 Вимоги до апаратного забезпечення 15](#_Toc499385606)

[2.2 Вимоги до програмного забезпечення 15](#_Toc499385607)

[2.3 Вимоги до користувачів 16](#_Toc499385608)

[3 Специфікація даних 16](#_Toc499385609)

[3.1 Опис формату файлу для зберігання хешів паролів користувачів 16](#_Toc499385610)

[3.2 Опис формату файлу для зберігання інформації сервісу ANS 16](#_Toc499385611)

[4 Функціональні вимоги 17](#_Toc499385612)

[4.1 Загальні вимоги 17](#_Toc499385613)

[5 Вимоги до інтерфейсу 18](#_Toc499385614)

[6 Проект програмної системи 18](#_Toc499385615)

[6.1 Засоби реалізації 23](#_Toc499385616)

[6.1.1 Варіанти реалізації системи 23](#_Toc499385617)

[6.1.1.1 Варіант 1 23](#_Toc499385618)

[6.1.1.2 Варіант 2 23](#_Toc499385619)

[6.1.1.3 Обґрунтування вибору 24](#_Toc499385620)

[6.2 Структури даних 24](#_Toc499385621)

[6.3 Проект інтерфейсу 35](#_Toc499385622)

[6.3.1 Користувацькі інтерфейси 36](#_Toc499385623)

[7 Реалізація і тестування 41](#_Toc499385624)

[Висновки 43](#_Toc499385625)

[Список літератури 44](#_Toc499385626)

# 1 Вступ

## Результат роботи

Результатом даної курсової роботи є створення симуляції мікропроцесора автоматичної телефонної станції у вигляді Windows застосунка. Мікропроцесор автоматичної телефонної станції повинен забезпечувати з'єднання і обмін даними між користувачами та іншими автоматичними телефонними станціями.

## Опис та функції АТС

Система автоматичних телефонних станцій забезпечує встановлення, підтримання та розрив з'єднань між апаратами, а також додаткові можливості. Це забезпечується застосуванням телефонної сигналізації.

В цілому обладнання АТС повинна забезпечувати виконання таких функцій:

1. Розпізнавання станцією сигналу ініціації виклику від абонента, якому потрібно виходить зв'язок

2. Видачу цьому абоненту сигналу про готовність до прийому керуючої інформації (історично це безперервний гудок);

3. Прийом інформації про набирається номері (спочатку набрані цифри номера передавалися відповідної послідовністю імпульсів при зворотному обертанні диска номеронабирача, в даний час зазвичай використовують більш короткі за часом багато частотні сигнали);

4. Запам'ятовування набраного номера;

5. Пошук і вибір (з багатьох доступних комбінацій «вільних» елементів АТС) варіанту підключення тракту зв'язку від АК абонента до АК абонента (якщо це абонент тієї ж АТС), або до каналу в напрямку АТС абонента. Маркування обраних елементів тракту «зайнятими» з метою неможливості їх використання для інших з'єднань, одночасно здійснюваних даної АТС. У разі відсутності вільних трактів - видача відповідного сигналу абоненту (в АТС декадно-крокової системи це був сигнал «зайнято» - часті гудки, який починав звучати прямо в середині набору номера);

6. Комутацію, тобто безпосереднє підключення обраного тракту зв'язку, що забезпечить подальшу розмову абонентів по тракту;

7. Передачу сигналу виклику абонента, якому телефонують (історично це періодичні переривчасті дзвінки телефонного апарату - дзвінки тривалістю 1 сек з інтервалами між ними по 4 сек, в даний час часто використовуються мелодії та ін. Сигнали). Одночасно абоненту надсилається сигнал «контроль посилки виклику» (аналогічний за тривалістю: гудок - 1 сек, інтервал - 4 сек). У разі, якщо телефонний апарат абонента зайнятий, тобто у нього знята трубка - видача тонального сигналу «зайнято» (часті гудки з коротким інтервалом між ними) тому, хто телефонує;

8. Підключення до тракту абонента після зняття їм трубки свого телефонного апарату;

9. Прийняття сигналу роз'єднання з'єднання від будь-якого з абонентів (вхідного або вихідного), коли буде покладена трубка телефонного апарату. Подальше роз'єднання елементів тракту зв'язку, встановленого між абонентами та маркування його елементів «вільними» (це забезпечить можливість використовувати ці елементи АТС, що складали тракт зв'язку між абонентами, в тих чи інших комбінаціях при установці інших з'єднань, що встановлюються керуючим обладнанням АТС в подальшому).

## Доцільність рішення

Необхідність даної системи і її доцільність полягає в усуненні телефоністок, тобто в роботизації та автоматизації процесів телефонної мережі для економії людських ресурсів.

## Глосарій

**Зачеплення, зчеплення** (англ. Coupling) - спосіб і ступінь взаємозалежності між програмними модулями; сила взаємозв'язків між модулями ; міра того, наскільки взаємозалежні різні підпрограми або модулі .

**Інформація** — це знання, які отримує споживач (суб'єкт) у результаті сприйняття і переробки певних відомостей.

**Клас**, в об'єктно-орієнтованому програмуванні— це спеціальна конструкція, яка використовується для групування пов'язаних змінних та функцій. При цьому, згідно з термінологією ООП, глобальні змінні класу (члени-змінні) називаються полями даних (також властивостями або атрибутами), а члени-функції називають методами класу. Створений та ініційований екземпляр класу називають об'єктом класу. На основі одного класу можна створити безліч об'єктів, що відрізнятимуться один від одного своїм станом (значеннями полів).

**Ме́тод** в об'єктно-орієнтованому програмуванні — підпрограма (процедура, функція), що використовується виключно разом із класом (методи класу) або з об'єктом (методи екземпляра).

**Об'єкт** в об'єктно-орієнтованому програмуванні (ООП) — ключове поняття об'єктно-орієнтованих технологій проектування та програмування; втілення абстрактної моделі окремої сутності (предмету або поняття), що має чітко виражене функціональне призначення в деякій області, належить до визначеного класу та характеризується своїми властивостями та поведінкою. Об'єкти є базовими елементами побудови програми — програма в ООП розглядається як сукупність об'єктів, що знаходяться у визначених відношеннях та обмінюються повідомленнями.

**Парадигма програмування** — система ідей і понять, які визначають стиль написання комп'ютерних програм

**Поле класу** або атрибут (змінна-член, data member, class field, instance variable) в об'єктно-орієнтованому програмуванні — змінна, зв'язана з класом або об'єктом. Всі дані об'єкта зберігаються в його полях. Доступ до полів здійснюється по імені. Зазвичай тип даних кожного поля задається в описі класу, членом якого є поле. Структурні типи, підтримувані більшістю мов програмування (які називаються структурами (structure) в C, записами (record) в Pascal і т. д.), є окремим випадком класів - а саме, класами з одних тільки полів. Вся інформація, що відноситься до полів класів, в рівній мірі відноситься і до структурних типів. Зазвичай кожному об'єкту відповідають власні значення всіх його полів. Також до полів класу відносять статичні поля(static data members, static class fields, class variables) — поля, загальні для всіх об'єктів класу.

**SHA-2** (англ. Secure Hash Algorithm Version 2 — безпечний алгоритм гешування, версія 2) — збірна назва односторонніх геш-функцій SHA-224, SHA-256, SHA-384 і SHA-512. Геш-функції призначені для створення «відбитків» або «дайджестів» повідомлень довільної бітової довжини.

## Опис наочної області

### 1.2.1 Об'єктно-орієнтоване програмування

Об'єктно-орієнтоване програмування – це парадигма програмування, заснована на концепті «об'єктів», які можуть містити інформацію у вигляді полів, часто відомих як атрибути, і коду, в формі процедур, часто відомих як методи. Однією з особливостей об'єкта є те, що процедури об'єкта можуть мати доступ і часто модифікувати поля даних об'єкта, з якими вони асоційовані (у об'єктів є нотація "this" «цей» або "self" «сам». В ООП, комп'ютерні програми сконструйовані з об'єктів, які можуть взаємодіяти між собою. Є велика різниця ООП мов, але найпопулярніші є "class-based" (заснованими на класах прим. пер.), Що означає, що об'єкти є "instances" ( «інстанціями» або «екземплярами») класів, які зазвичай визначають їх тип . Багато широко використані мови програмування (такі як С ++, Java, Python і. Т. Д.) є мовами програмування мульти-парадигми ("multi-paradigm programming languages") які підтримують об'єктно-орієнтоване програмування в більшому або меншому ступені, зазвичай в комбінації з імперативним, процедурним програмуванням. В значні об'єктно-орієнтовані мови входять: Java, C ++, C #, Python, PHP, Ruby, Perl, Object Pascal, Objective-C, Dart, Swift, Scala, Common Lisp, and Smalltalk.

**Об'єкти і класи в ООП**

Мови які підтримують об'єктно-орієнтоване програмування зазвичай використовують успадкування для повторного використання коду і розширюваності в формі або класів або прототипів. Ті які використовують класи підтримують два головних концепту:

* **Класи** - визначення для формату даних і доступні процедури для даного типу класу або об'єкта, можуть також містити дані або самі процедури (також відомі як методи).
* **Об'єкти** - екземпляри класів

Об'єкти іноді відповідають за те, що можна знайти в реальному світі. Наприклад, графічна програма може мати такі об'єкти як «коло», «квадрат», «меню». Онлайн шопінг система може мати об'єкти такі як «кошик для покупок», «покупець» і «продукт». Іноді об'єкти представляють більш абстрактні сутності, наприклад об'єкт який представляє відкритий файл, або об'єкт, який надає сервіс перекладу вимірювань за стандартом США в метричну.

**Інкапсуляція**

Інкапсуляція це концепт в об'єктно-орієнтованому програмуванні який об'єднує разом дані і функції які маніпулюють даними, і який зберігає їх у безпеці від зовнішньої інтерференції і зловживання. Інкапсуляція даних привела до важливого концепту ООП - "**data-hiding**" (прим. пер. «приховування»).

**Композиція**

Об'єкти можуть містити інші об'єкти в собі як змінні інстанції. Це відомо як композиція об'єкта. Наприклад, об'єкт в класі Працівник може містити (вказувати на) об'єкт класу Адреса, в добавок до власних змінних таких як «ім'я» і «посада». Композиція об'єктів використовується, щоб представити відносини "**has-a**" «має». Кожен працівник має адресу, тому кожен об'єкт класу Працівник має місце щоб зберігати об'єкт класу Адреса.

**Спадкування**

Мови, які підтримують класи, зазвичай підтримують успадкування. Це дозволяє класам бути розташованими в ієрархії, яка представляє відносини типу "**is-a-type-of**" «є-типом». Наприклад, клас Працівник може успадковувати від класу Особистість. Всі дані і методи доступні батьківському класу також видимі класу нащадку з тими ж іменами. Наприклад, клас Особистість може визначати змінні «ім'я» і «прізвище» і метод «getFullName ()». Вони будуть також доступні в класі Працівник, який може додати змінні «посада» і «зарплата». Ця техніка дозволяє легко повторно використовувати ті ж процедури і визначення даних, на додачу до потенційного дзеркального і інтуїтивного відображення відносин реального світу.

Принцип «відкритості-закритості» ("**open-closed principle**") відстоює, що класи і функції повинні бути «відкриті для розширення, але закриті для модифікації».

Інтуїтивно припустити, що успадкування створює семантичне відношення "**is a**" «є», і тому зробити висновок, що об'єкти ініційовані підкласами завжди можуть бути безпечно використані замість таких, які були ініційовані від суперкласу. Дане припущення, нажаль, не є істиною для більшості мов ООП, зокрема в тих, які дозволяють mutable об'єкти.

**Поліморфізм**

**Subtyping** - форма поліморфізму, це коли викликання коду може бути агностичним по відношенню до того належить об'єкт класу батькові або одному з його спадкоємців. Наприклад, функція може викликати «getFullName ()» на об'єкті, який буде працювати незважаючи на те чи буде об'єкт належати класу Особистість чи класу Працівник. Це ще один тип абстракції, який спрощує код зовнішній для ієрархії класів і дозволяє суворий поділ відповідальності.

**Відкрита рекурсія**

У мовах, які підтримують відкриту рекурсію методи об'єктів можуть викликати інші методи цього ж об'єкта (включаючи самих себе), як правило, використовуючи спеціальне ключове слово "**this**" або "**self**".

**Design patterns Банди Чотирьох**

Design Patterns: Elements of Reusable Object-Oriented Software є найвпливовішою книгою опублікованою в 1995 році Еріком Гамма, Річардом Хельм, Ральфом Джонсоном і Джоном Віссайдес, яких часто називають «Бандою Чотирьох» Разом з дослідженням можливостей і підводних каменів об'єктно-орієнтованого програмування, вона описує 23 поширені проблеми програмування і шаблони для їх вирішення.

Книга описує наступні шаблони:

* Creational patterns: Factory method pattern, Abstract factory pattern, Singleton pattern, Builder pattern, Prototype pattern
* Structural patterns: Adapter pattern, Bridge pattern, Composite pattern, Decorator pattern, Facade pattern, Flyweight pattern, Proxy pattern
* Behavioral patterns: Chain-of-responsibility pattern, Command pattern, Interpreter pattern, Iterator pattern, Mediator pattern, Memento pattern, Observer pattern, State pattern, Strategy pattern, Template method pattern, Visitor pattern

У процесі розробки програми для курсової роботи були застосовані, крім інших, такі design patterns: Singleton pattern, Facade pattern, Mediator pattern і Observer pattern. Давайте коротенько розглянемо їх.

**Singleton pattern**

В інженерії програмного забезпечення, шаблон singleton - це шаблон, який обмежує ініціалізацію класу до одного об'єкта. Це є корисним, коли саме один об'єкт потрібен для координування дій по всій системі.

**The singleton design pattern вирішує такі проблеми як:**

* Як переконатися в тому що існує тільки один екземпляр класу?
* Як єдиний екземпляр класу може бути легкодоступний?
* Як клас може контролювати власну ініціалізацію?
* Як кількість екземплярів класу може бути обмежена?

**The singleton design pattern описує як вирішити такі проблеми:**

* Приховати конструктор класу
* Визначити відкриту статичну операцію (getInstance()), яка повертає єдиний екземпляр класу.

Ключова ідея цього шаблону полягає в тому, щоб клас сам по собі був відповідальний за контроль власної ініціалізації (за те, щоб бути ініційованим тільки один раз).

Прихований конструктор (оголошений private) гарантує, що клас ніколи не буде ініційований ззовні.

Відкрита статична операція може бути легкодоступною при використанні імені класу і імені даної операції (Singleton. GetInstance ()).

**Де його використовувати?**

Коли дозволені тільки один або кілька екземплярів класу. Об'єкти Façade часто є singleton, тому що потрібен тільки один об'єкт Façade. Саме це реалізовано в цій курсовій роботі. Клас ATCService має відкриту статичну змінну singleton, яка є одночасно і Façade об'єктом.

**Façade pattern**

The façade pattern є шаблоном розробки програмного забезпечення і часто використовується в ООП.

**Фасад** - це об'єкт, який надає спрощений інтерфейс більшого тілу коду, такого як бібліотека класів. фасад може

* Робити бібліотеку програмного забезпечення легше для використання, розуміння і тестування, так як фасад має зручні методи для поширених завдань
* Робити бібліотеку більш читабельною, з тих же причин
* Зменшити кількість залежностей зовнішнього коду для внутрішніх процесів бібліотеки, так як більша частина коду використовує фасад, що приводить до більшої гнучкості при розробці системи.
* Об'єднати погано спроектовану колекцію APIs в одну добре спроектовану API.

The Façade design pattern часто використовується, коли система дуже складна і важка для розуміння, так як система має велику кількість взаємозалежних класів, або їх вихідний код недоступний. Цей шаблон приховує складності великих систем і надає спрощений інтерфейс клієнта. Зазвичай це включає в себе один клас-обгортку, який містить набір членів необхідних клієнтом. Члени мають доступ до системи через фасад і приховують деталі імплементації.

**Mediator pattern**

The meditator pattern визначає об'єкт, який інкапсулює в собі процес взаємодії об'єктів між собою. Цей шаблон вважається **behavioral pattern** (шаблоном поведінки), тому що він змінює поведінку програми.

Зазвичай програма складається з великого числа класів. Логіка і обчислення розподілені між цими класами. Проте, чим більше класів додаються в програму, особливо в процесі обслуговування і / або рефакторізаціі, проблема комунікації між цими класами стає все більш і більш складною. Це робить програму складнішою в обслуговуванні. Більш того, може стати дуже складно змінити програму, так як будь-яка зміна може вплинути на код в декількох класах.

З шаблоном посередника (mediator patttern), комунікація між об'єктами інкапсульована в об'єкті посередника. Об'єкти більше не взаємодіють безпосередньо один з одним, але замість цього мають комунікацію через посередника. Це зменшує кількість залежностей між об'єктами комунікації, і тому зменшує зчеплення (**coupling**).

**Які проблеми може вирішити шаблон посередника?**

* Необхідно уникнути міцного зчеплення (coupling) між набором взаємодіючих об'єктів.
* Слід зробити можливим незалежну зміну взаємодії між набором об'єктів.

**Яке рішення описує шаблон посередника?**

* Визначити окремий об'єкт (посередник), який інкапсулює взаємодію між наборами об'єктів.
* Об'єкти делегують свою взаємодію посереднику, замість того щоб безпосередньо взаємодіяти між собою.

Це робить зчеплення між об'єктами слабким. Вони звертаються тільки до посередника, і не мають явного знання про один одного.

**Observer pattern**

The observer pattern - це шаблон проектування програмного забезпечення, в якому об'єкт, званий **subject**, обслуговує список його утриманців, званих **observers**, і автоматично оповіщає їх про будь-яку зміну стану, зазвичай за допомогою виклику одного з їх методів.

В основному цей шаблон використовується для імплементації розподіленої системи обслуговування подій. Сучасні мови, такі як Java і C # мають вбудовану конструкцію події - **event**, які виконують роль компонентів шаблону спостерігача, для легкості програмування і стислості коду.

Шаблон спостерігача також є ключовою частиною широко відомого архітектурного шаблону **model-view-controller** (MVC). Шаблон спостерігача імплементований в численних програмних бібліотеках і системах, включаючи майже всі GUI інструментарії.

**Які проблеми може вирішити шаблон спостерігача?**

• Залежність один-до-багатьох між об'єктами повинна бути визначена, без того, щоб зробити об'єкти сильно зчепленими.

• Повинно бути забезпечено те, що при зміні в стані одного об'єкта відкрите число залежних об'єктів оновлювалися автоматично.

• Повинно бути можливо, щоб один об'єкт міг повідомляти відкрите число інших об'єктів.

**Яке рішення описує шаблон спостерігача?**

• Визначити об'єкти Subject і Observer

• Таким чином, що коли Subject змінює стан, всі зареєстровані спостерігачі сповіщаються і оновлюються автоматично.

**SOLID and GRASP guidelines**

SOLID - це мнемонічний акронім, п'ять програмних принципів описаних Michael Feathers:

* Single responsibility principle
* Open / closed principle
* Liskov substitution principle
* Interface segregation principle
* Dependency inversion principle

GRASP (General Responsibility Assignment Software Patterns) є набором іншим принципів і шаблонів розподілу відповідальності описані Craig Larman.

### 1.2.2 Телефонні мережі загального користування

З тих часів як людська цивілізація почала займати територію більшу ніж невелике селище перед людиною завжди стояла задача швидкої, надійної і дешевої передачі інформації. По цей час людина знаходить і продовжує знаходити доцільні рішення даної проблеми. Одним з таких рішень є телефонні мережі. Перші комерційні телефонні мережі були створені в 1876 році як інструмент для передачі інформації між двома локаціями. І зараз, в 2017 році люди все ще користуються телефонним зв'язком, що показує, що дане рішення є доцільним та витримало перевірку часом.

#### 1.2.2.1 Опис телефонної мережі загального користування

Телефонна мережа загального користування, ТМЗК, ТМЗК (англ. PSTN, Public Switched Telephone Network) - це абонентська мережа зв'язку, для доступу до якої використовуються телефонні апарати, АТС та обладнання передачі даних.

Надання послуги доступу кінцевих користувачів до телефонної мережі загального користування знаходиться у веденні операторів телефонного зв'язку. У переважній більшості випадків, кожен абонент ТМЗК отримує певний унікальний (глобально або в рамках конкретного сегмента мережі) ідентифікатор - телефонний номер. Однак, існують інші способи організації доступу, наприклад: оренда таксофона, колективний доступ з одного і того ж «зовнішнього» номери для абонентів УВАТС з внутрішніми номерами з однієї організації, а також телефонні переговорні пункти.

У сучасному світі доступ до телефонної мережі загального доступу може надаватися у вигляді таких технологій:

* Провідна (або фіксована) телефонний зв'язок
* DECT
* стільниковий зв'язок
* Супутниковий зв'язок
* IP-телефонія
* ТМЗК - секція телефонної інфраструктури, що вела від софтсвічей Class-5 (PBX) офісів і здійснювана IXC (англ. Interexchange carriers).

У ТМЗК передача сигналів (в тому числі і настройка з'єднання) і сама розмова здійснюється через одну і ту ж універсальну лінію зв'язку (магістраль) від системи комутації (СК) джерела до СК адресата. Цей процес займає канали зв'язку всіх задіяних при з'єднанні СК, тобто, якщо викликається адресат зайнятий, всі ці сполуки виявляться марними.

Зазвичай ТМЗК використовують зіркоподібну топологію (головний елемент з'єднаний з безліччю другорядних). Але це не єдиний метод. Наприклад, CATV використовують деревоподібну топологію.

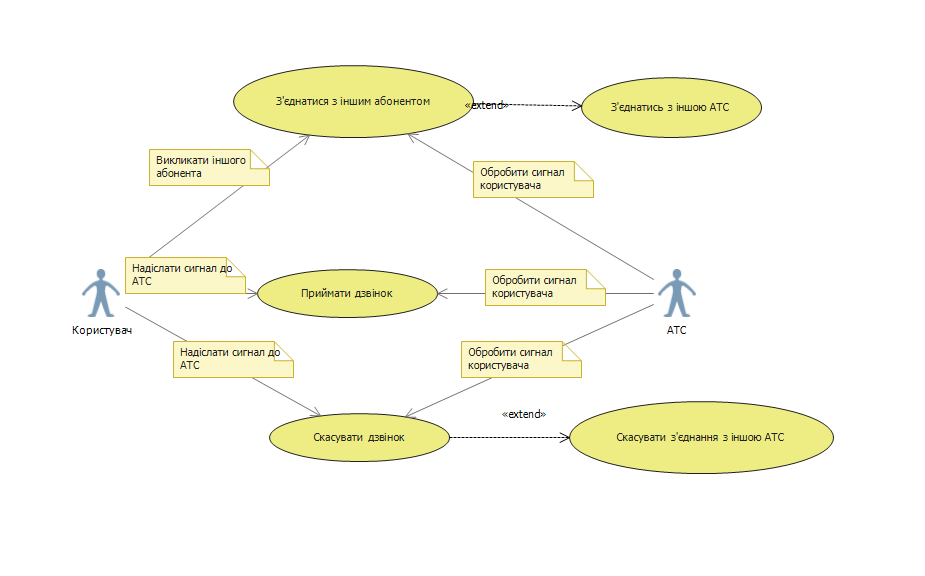
## Постановка задачі

Потрібно розробити модель програмного забезпечення вбудованого мікропроцесора засновницької міні-АТС (автоматичної телефонної станції).

Міні-АТС здійснює зв'язок між службовцями установи. кожен абонент підключений до неї лінією зв'язку. Міні-АТС з'єднує лінії абонентів (здійснює комутацію ліній).

Абоненти мають номери, що складаються з трьох цифр. Телефонні станції мають ідентифікаторі, що складаються з двох цифр.

Діаграму варіантів використання дивись на рис. 1.3.1.



*Рис.1.3.1 Діаграма варіантів використання*

## План робіт

1. Розробка структури програмного забезпечення – 2 години
2. Розробка програмного забезпечення – 10 годин
3. Тестування – 2 години
4. Написання курсової роботи – 6 годин

# 2 Вимоги до оточення

## 2.1 Вимоги до апаратного забезпечення

**Мінімальна апаратна конфігурація**

* Процесор 800 МГц
* 128 МБ оперативної пам'яті
* Відеокарта з обсягом пам'яті 32 МБ
* Миша, клавіатура

**Рекомендована апаратна конфігурація:**

* Процесор 3 ГГц
* 4 ГБ оперативної пам'яті
* Відеокарта з обсягом пам'яті 512 МБ
* Миша, клавіатура

## 2.2 Вимоги до програмного забезпечення

Базове програмне забеспечення:

* Windows 7 і вище
* Інтегроване середовище розробки Visual Studio 2014 і вище

## 2.3 Вимоги до користувачів

**Вимоги до абонента**

Роль: Абонент

Функції: з’єднання з іншими абонентами

Можливості: Отримати ідентифікатор з трьох цифр, мати пароль для свого облікового запису, з’єднання з іншими абонентами.

Вимоги: Базові знання англійської мови.

**Вимоги до адміністратора**

Роль: Адміністратор АТС

Функції: Контроль роботи АТС

Можливості: Перегляд з’єднаних користувачів та АТС, перегляд поточних з’єднань, та журналу подій.

Вимоги: Базові знання англійської мови.

# 3 Специфікація даних

## 3.1 Опис формату файлу для зберігання хешів паролів користувачів

Програма в процесі роботи може створювати двійкові файли, які зберігають хеші паролів користувачів якоїсь даної АТС. Програма перед зберіганням отримує хеш паролю використовуючи алгоритм MD5, у наступному форматі: файл зберігає цю інформацію у вигляді бібліотеки, де ключі – це ідентифікатори користувача даної АТС (трьох значний номер), а значення – це хеш паролю. Назва файлу – це ідентифікатор АТС, її двозначний номер. Формат файлу – ‘.dat’.

## 3.2 Опис формату файлу для зберігання інформації сервісу ANS

Програма використовує сервіс ANS для зберігання пар «ідентифікатор» - «Ім’я» для кожної АТС. Сервіс ANS зберігає цю інформацію у двійковому файлі, у вигляді бібліотеки, де ключі – ідентифікатори АТС (двозначні номери), а значення – імена АТС. Програма створює тільки один файл такого формату, та він має назву ‘ANS.dat’.

# 4 Функціональні вимоги

## 4.1 Загальні вимоги

* Система повинна ініціалізувати АТС
* Система у разі, коли користувач не задав ідентифікатор АТС повинна автоматично надавати двоцифровий ідентифікатор
* Система повинна зберігати імена АТС у парі з їх ідентифікатором
* Система повинна у разі введення користувачем ідентифікатору АТС та імені, яке не збігається з поточним ім’ям АТС. перейменувати існуючу АТС
* Система повинна з’єднувати між собою все поточні ініціалізовані АТС.
* Система повинна включати АТС з ім’ям “Mini” та ідентифікатором 00 при натисканні на кнопку “Mini” на головному вікні.
* Система повинна включати АТС з ім’ям “City” та ідентифікатором 09 при натисканні на кнопку “ City ” на головному вікні.
* Система повинна зберігати облікові записи користувачів
* Система повинна реєструвати новий обліковий запис користувача, якщо облікова запис за наданим трьох значним ідентифікатором не існує
* Система повинна перевіряти пароль, та якщо він не правильний відкривати вікно з інформацією про помилку
* Система повинна відкривати вікно з інформацією про помилку, якщо введені дані є некоректними або сталася помилка при обробці
* Система повинна відображати список з’єднаних користувачів та АТС, список поточних з’єднань, та список журналу подій, при натисканні на кнопку “Settings” на вікні будь якої АТС.
* Система повинна з’єднувати користувача АТС з іншим користувачем цієї ж АТС при натисканні користувачем зеленої кнопки та введенні трьох цифр.
* Система повинна з’єднувати користувача АТС з іншим користувачем іншої АТС при натисканні користувачем зеленої кнопки та введенні п’яти цифр.
* Система повинна інформувати користувача АТС, якщо він намагається з’єднатися с користувачем який не підключений до мережі, або вже має зв’язок з іншими користувачами.
* Система повинна здійснювати з’єднання між двома користувачами або користувачем та іншою АТС, якщо після того як, проінформований користувач АТС, про те, що деякий користувач бажає здійснити з ним з’єднання, натиснув на зелену кнопку.
* Система повинна з’єднувати деякого користувача, якщо користувач деякої АТС, який вже має з’єднання з іншими користувачами, натиснув зелену кнопку та ввів номер даного користувача, до вже існуючого з’єднання тим самим створюючи з’єднання трьох або більше користувачів

# 5 Вимоги до інтерфейсу

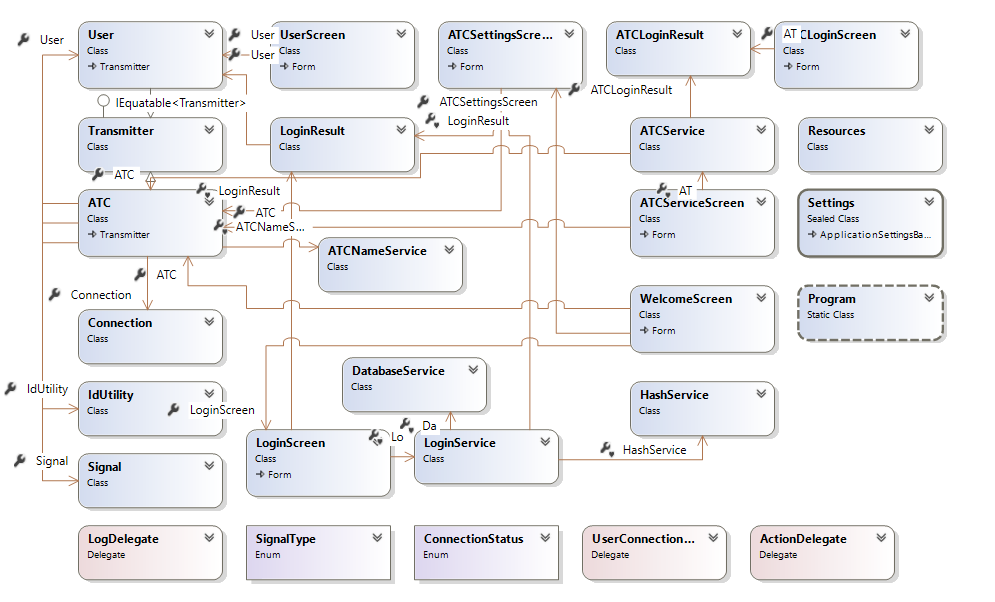
Програма має інтерфейс Windows застосунка. Основні елементи управління – це кнопки, та текстові поля.

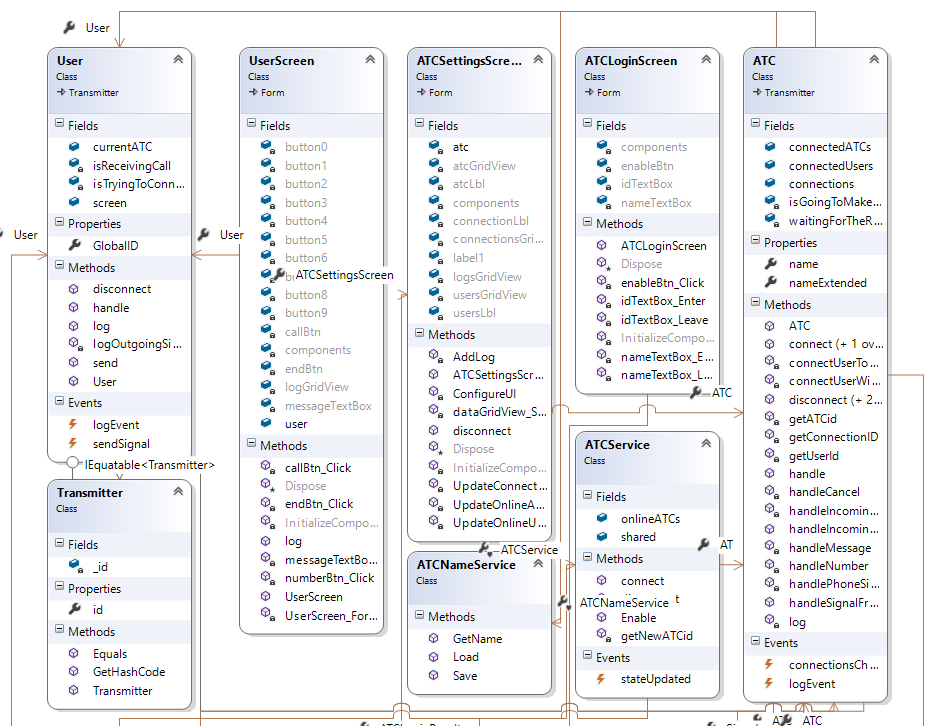
# 6 Проект програмної системи

Система складається з двадцяти класів, з яких два – основні, та шість – це елементи користувацького інтерфейсу. Система має ієрархічний вид. Найвищим рівень ієрархії виступає клас “ATCService”, який відповідає за вмикання та вимикання АТС в програмі та за зв’язок між ними. Далі йде клас “ATC”. Він відповідає за користувачів з’єднаних з нею та за обробку сигналів від цих користувачів, або інших АТС. Та найнижчим є клас “User”, який тільки відправляє сигнали та приймає сигнали зі своєї АТС. При цьому при розробці класу “ATCService” використовувались Façade та Singleton design patterns, а при розробці класу “ATC” – Mediator та Observer design patterns.

При розробці системи розробник керувався принципами SOLID. Наприклад, кожен клас системи здійснює одну і тільки одну функцію. Наприклад клас-сервіс “ANSService” зберігає та надає інформацію про імена АТС, а “DataService” серіалізує та десеріалізує хеші паролів. Це э Single responsibility principle.

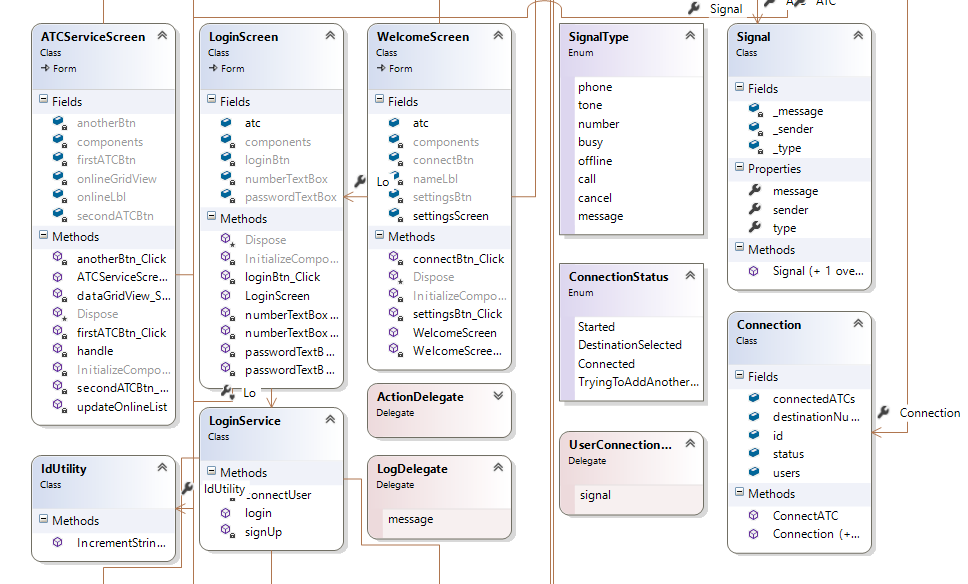
Діаграму класів дивись на рис. 6.0.1, та детальну діаграму класів на рисунках 6.0.2, 6.0.3 та 6.0.4.

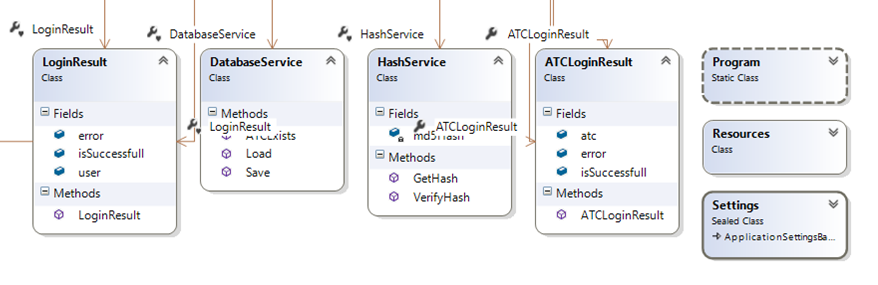
*Рис. 6.0.1 Діаграма класів*



*Рис. 6.0.2 Деталізована діаграма класів, частина 1*

*Рис. 6.0.3 Деталізована діаграма класів, частина 2*





*Рис. 6.0.4 Деталізована діаграма класів, частина 3*

## 6.1 Засоби реалізації

### 6.1.1 Варіанти реалізації системи

Розглядалися два варіанти реалізації системи.

#### 6.1.1.1 Варіант 1

Мова програмування: JavaScript

Середовища розробки: Atom, Branch

При розробці даного варіанту, був би створений сервер на Node JS, та front-end на JavaScript, HTML та CSS. Сервер виконував би функції системи, яка керує АТС, а front-end був би крос-платформним користувацьким інтерфейсом. Результатом розробки даного варіанту був би надійний та ефективний кінцевий програмний продукт, який би був по факту відносно корисним messenger.

Переваги:

* Ефективність
* Надійність
* Корисність

Недоліки:

* Витрачений час. На розробку даного продукту знадобилось би щонайменше 20-30 робочих годин.

#### 6.1.1.2 Варіант 2

Мова програмування: C#

Середовище розробки: Visual Studio 2014

При розробці даного варіанту був створений Windows-застосунок, який одночасно виконує функції серверу та користувацького інтерфейсу. Результатом розробки даного варіанту є симуляція АТС.

Переваги:

* Швидкість розробки
* Достатність для виконання умов завдання

Недоліки

* Марність кінцевого продукту

#### 6.1.1.3 Обґрунтування вибору

У зв’язку з браком часу, було прийнято рішення, що краще «готово» ніж «ідеально», був обраний варіант 2. Цей варіант, незважаючи на очевидну перевагу першого, все ж таки, задовольняє умові завдання.

## 6.2 Структури даних

**Клас АТС**

Даний клас описує об’єкти, які виконують чи симулюють функції АТС. При розробці класу використовувались Mediator та Observer design patterns. Даний клас наслідує клас Transmitter.

public event LogDelegate logEvent;

Даний event типу LogDelegate передає інформацію to subscribers про зміни в стані, або операції АТС.

public event ActionDelegate connectionsChanged;

Даний event типу ActionDelegate передає інформацію to subscribers про зміни поточних з’єднань.

public ATC(string id, string name) : base(id)

Даний конструктор класу, приймає два параметри типу string, ініціалізує базовий клас Transmitter та передає запит до ATCNameService на зберігання назви поточної АТС.

public List<ATC> connectedATCs = new List<ATC>();

Дана зміна зберігає список приєднаних до поточної АТС інших АТС.

private List<User> connectedUsers = new List<User>();

Дана зміна зберігає список приєднаних до поточної АТС абонентів.

private List<Connection> connections = new List<Connection>();

Дана зміна зберігає список поточних з’єднань поточної АТС.

private List<User> waitingForTheResponceFrom = new List<User>();

Це допоміжний список, який зберігає в себе абонентів від яких зараз очікуєтеся відповідь.

private Dictionary<string, bool> isGoingToMakeOutgoingCall = new Dictionary<string, bool>();

Це допоміжна бібліотека, яка зберігає в себе ідентифікатори абонентів, які зараз набирають номер, та можуть вирішити подзвонити абоненту який не належить даній АТС.

public string name

Це computed property, яка має getter, який повертає строку яка містить в собі результат запиту до методу класу ATCNameService, який поверне ім’я поточної АТС, якщо ATCNameService є запис про поточну АТС.

public string nameExtended

Це computed property, яка має getter, який повертає строку яка містить в собі змінну name цього класу та поточну кількість з’єднань поточної АТС в квадратних скобках.

private void log(string message)

Це метод який викликає event logEvent поточного класу якщо він не є null.

public void disconnect()

Це метод який роз’єднує поточну АТС зі всіма з’єднаними до неї АТС, абонентами та subscribers of events logEvent та connectionsChanged.

public void disconnect(ATC atc)

Це метод який роз’єднує поточну АТС та АТС, передану як параметр даної функції

public bool connect(User user)

Це метод який з’єднує поточну АТС та User, переданого як параметр даної функції. Цей User вважається абонентом даної АТС.

public void disconnect(User user)

Це метод який роз’єднує поточну АТС та User, переданого як параметр даної функції.

public void connect(ATC atc)

Це метод який з’єднує поточну АТС та АТС, передану як параметр даної функції.

void connectUserToATC(User user, Connection connection)

Це метод який з’єднує User, переданого як параметр даної функції, до нового з’єднання яке встановлено з іншою АТС.

public string handleIncomingConnection(ATC atc, string destinationNumber, string connectionID, string callerID)

Це метод який обробляє запит на вхідне з’єднання з іншої АТС.

string getConnectionID()

Це метод який повертає новий, не зайнятий ідентифікатор з’єднання.

string getATCid(string GlobalID)

Це метод який повертає ідентифікатор АТС із номеру “GlobalID”, яке містить ідентифікатор абонента, та АТС до якої він належить. Наприклад, абонент з ідентифікатором 017 АТС з ідентифікатором 09 буде мати “GlobalID” 09017.

string getUserId(string GlobalID)

Це метод який повертає ідентифікатор абоненту із номеру “GlobalID”, яке містить ідентифікатор абонента, та АТС до якої він належить.

public void handleSignalFromATC(Signal signal, string connectionID)

Це метод який обробляє сигнал з іншої АТС.

void handleIncomingBusyOffline(Signal signal, Connection connection)

Це метод який обробляє “Busy” and “Offline” сигнал з іншої АТС.

public void handle(Signal signal)

Це метод який обробляє сигнали абонентів з поточної АТС.

void handlePhoneSignalWithExistingConnection(User sender, Signal signal, Connection connection)

Це метод який обробляє сигнал “Phone” від абонента, який вже має з’єднання. Це дозволяє додавати до з’єднання одночасно більше двох абонентів.

void connectUserWithIncommingCaller(Connection connection, ATC atc, string callerID)

Це метод який з’єднує локального абонента з іншою АТС.

async void handleNumber(Signal signal, Connection connection)

Це асинхронний метод який обробляє сигнал “Number”. Асинхронність цього методу дозволяю «почекати» дві секунди після вже набраних трьох цифр номеру призначення, для того щоб перевірити, хоче абонент здійснити вихідний дзвінок, чи ні.

void handleMessage(Signal signal, Connection connection)

Це метод який обробляє сигнал “Message”.

void handleCancel(Signal signal, Connection connection)

Це метод який обробляє сигнал “Cancel”.

**Клас User**

Даний клас описує об’єкти, які виконують функції абонента.

public UserScreen screen;

Це змінна яка зберігає посилання на форму UserScreen поточного абонента.

public ATC currentATC;

Це змінна яка зберігає посилання на АТС до якої під’єднаний даний абонент.

public event LogDelegate logEvent;

Даний event типу LogDelegate передає інформацію to subscribers про зміни в стані, або операції User.

public event UserConnectionDelegate sendSignal;

Даний event типу UserConnectionDelegate посилає сигнали на під’єднану АТС.

bool isTryingToConnect = false;

Це змінна яка зберігає інформацію про те, намагається поточний абонент здійснити з’єднання чи ні.

public User(string id, ATC atc) : base(id)

Даний конструктор класу, приймає параметри типу string та параметр типу АТС, ініціалізує базовий клас Transmitter та реєструє АТС, як ту, до якої належить даний абонент.

public string GlobalID

Це змінна є computed property, та має getter, який повертає “GlobalID”, який містить ідентифікатор абонента, та АТС до якої він належить. Наприклад, абонент з ідентифікатором 017 АТС з ідентифікатором 09 буде мати “GlobalID” 09017.

public void send(SignalType type, string message)

Це метод який надсилає сигнал до поточної АТС.

public void log(string message)

Це метод який викликає event logEvent поточного класу якщо він не є null.

public void handle(Signal signal)

Це метод який обробляє отриманий з поточної АТС сигнал.

void logOutgoingSignal(Signal signal)

Це метод викликає функцію log, повідомленням для кожного типу сигналу.

public void disconnect()

Це метод роз’єднує абонента з поточною АТС.

**Клас Transmitter**

Даний клас є базовим для User та ATC, та зберігає ідентифікатор трансмітеру.

private string \_id;

Це приватна змінна яка зберігає інформацію про ідентифікатор трансмітеру.

public string id

Це змінна є computed property, та має getter, який повертає значення змінної \_id.

public Transmitter(string id)

Даний конструктор класу, приймає параметр типу string та ініціалізує змінну \_id.

public bool Equals(Transmitter other)

Це метод який порівнює ідентифікатор поточного трансмітеру та іншого трансмітеру.

public override int GetHashCode()

Це перевантажений метод який повертає результат виклику методу GetHashCode на id.

**Клас Signal**

Даний клас зберігає інформацію про сигнал в застосунку.

public enum SignalType

Даний enum зберігає перелік типів сигналів.

private SignalType \_type;

Це приватна змінна яка зберігає інформацію про тип сигналу.

private string \_message;

Це приватна змінна яка зберігає інформацію про повідомлення сигналу.

private Transmitter \_sender;

Це приватна змінна яка зберігає інформацію про відправника сигналу.

public SignalType type

Це змінна є computed property, та має getter, який повертає значення змінної \_type.

public string message

Це змінна є computed property, та має getter, який повертає значення змінної \_message.

public Transmitter sender

Це змінна є computed property, та має getter, який повертає значення змінної \_sender.

public Signal(Transmitter sender, SignalType type)

Даний конструктор класу, ініціалізує змінну \_sender та \_type. Об’єкт який ініціалізує даний конструктор використовується в застосунку як сигнал без повідомлення, наприклад “Phone”.

public Signal(Transmitter sender, SignalType type, string message)

Даний конструктор класу, ініціалізує змінну \_sender, \_type та \_message.

**Клас Connection**

Даний клас зберігає інформацію про сигнал в застосунку.

enum ConnectionStatus

Даний enum зберігає перелік статусів з’єднань.

public string id;

Це змінна яка зберігає інформацію про ідентифікатор з’єднання.

public ConnectionStatus status;

Це змінна яка зберігає поточний статус з’єднання.

public string destinationNumber;

Це змінна яка зберігає поточний номер призначення з’єднання.

public List<User> users = new List<User>();

Це змінна яка зберігає список абонентів які під’єднані до з’єднання.

public Dictionary<ATC, string> connectedATCs = new Dictionary<ATC, string>();

Це змінна яка зберігає список АТС які під’єднані до з’єднання.

public Connection(string id)

Даний конструктор класу, надає з’єднанню ідентифікатор, та статус “Started”.

public Connection(string id, User user)

Даний конструктор класу, надає з’єднанню ідентифікатор, та статус “Started”, та добавляє абонента user до списку під’єднаних абонентів.

public void ConnectATC(ATC atc, string connectionID)

Даний метод добавляє АТС atc до з’єднання.

**Клас DatabaseService**

Даний клас зберігає серіалізує та десералізує хеші паролів користувачів.

public static bool Save(string atcId, string id, string passwordHash)

Це статичний метод який серіалізує хеш паролю для користувача з ідентифікатором id та АТС atcId.

public static Dictionary<string, string> Load(string atcId)

Це статичний метод який десеріалізує бібліотеку хешів паролів для користувачів який належить АТС з ідентифікатором atcId.

public static bool ATCExists(string atcId)

Це статичний метод який перевіряє наявність інформації з хешамі паролів для даної АТС з ідентифікатором atcId.

**Клас HashService**

Даний клас отримує та перевіряє SHA256 хеші паролів користувачів.

public static string GetSHA256Hash(string input)

Це статичний метод, який отримує SHA256 хеш зі строки input.

public static bool VerifySHA256Hash (string input, string hash)

Це статичний метод, який перевіряє SHA256 хеш зі строки input, та хеш hash.

**Клас LoginResult**

Даний клас зберігає інформацію про результат авторизації.

public string error;

Ця змінна зберігає інформацію про помилку, яка відбулася під час авторизації. Коли авторизація була успішною вона дорівнює null.

public bool isSuccessfull;

Ця змінна зберігає інформацію про результат авторизації.

public User user;

Ця змінна зберігає об’єкт користувача user, якщо авторизація була успішною.

public LoginResult(bool isSuccessfull, string error, User user)

Це конструктор який ініціалізує всі поля класу.

**Клас LoginService**

Даний клас відповідає за реєстрацію та авторизацію абонентів.

public static LoginResult login(string id, string password, ATC atc)

Це публічний статичний метод, який виконує авторизацію з заданими ідентифікатором, паролем та АТС, повертає об’єкт типу LoginResult з інформацією про результат авторизації, та інформацією про помилку, якщо вона відбулася.

static LoginResult signUp(ATC atc, string id, string password)

Це публічний статичний метод, який виконує реєстрацію з заданими ідентифікатором, паролем та АТС, повертає об’єкт типу LoginResult з інформацією про результат авторизації, та інформацією про помилку, якщо вона відбулася.

static User connectUser(string id, ATC atc)

Це статичний метод, який відправляє АТС atc запит на з’єднання абонента user. Повертає null, якщо відбулася помилка.

**Клас IdUtility**

Даний допоміжний клас декларує функцію для інкрементування ідентифікатору.

public static string IncrementStringID(string id)

Це статичний метод, який інкрементує та повертає ідентифікатор id. Наприклад, якщо передати параметром ідентифікатор «0526» функція поверне строку «0527».

**Клас ATCLoginResult**

Даний клас зберігає інформацію про результат АТС авторизації.

public string error;

Ця змінна зберігає інформацію про помилку, яка відбулася під час авторизації. Коли авторизація була успішною вона дорівнює null.

public bool isSuccessfull;

Ця змінна зберігає інформацію про результат авторизації.

public ATC atc;

Ця змінна зберігає об’єкт АТС atc, якщо авторизація була успішною.

public ATCLoginResult(bool isSuccessfull, string error, ATC atc)

Це конструктор який ініціалізує всі поля класу.

**Клас ATCService**

Даний клас відповідає за вмикання та вимикання АТС в програмі та за зв’язок між ними. При розробці цього класу використовувались Façade та Singleton design patterns.

public event ActionDelegate stateUpdated;

Ця змінна – event типу ActionDelegate повідомляє subscribers про зміни в стані та процеси сервісу.

public static ATCService shared = new ATCService();

Ця статична змінна – сінглтон, який містить посилання на статичний об’єкт цього класу.

public List<ATC> onlineATCs = new List<ATC>();

Ця змінна зберігає список поточних включених АТС.

public ATCLoginResult Enable(string name, string id)

Це метод, який «включає» АТС з заданим ім’ям та ідентифікатором.

public void connect(ATC atc)

Це метод, який з’єднує АТС atc з усіма включеними АТС.

public void disconnect(ATC atc)

Це метод, який видаляє АТС atc зі списку включених АТС.

string getNewATCid()

Це метод, який повертає новий, не зайнятий ідентифікатор АТС.

**Клас ATCNameService**

Даний клас відповідає за зберігання пар ім’я – ідентифікатор для АТС.

public static bool Save(string atcName, string id)

Це статичний метод, який зберігає пару ім’я atcName та ідентифікатор id.

public static string GetName(string id)

Це статичний метод, який повертає ім’я для переданого ідентифікатору, або null, якщо потрібного запису не існує.

public static Dictionary<string, string> Load()

Це статичний метод, який повертає бібліотеку, яка зберігає всі існуючі пари ім’я – ідентифікатор, або null, якщо її не існує.

## 6.3 Проект інтерфейсу

При проектуванні інтерфейсу програми, використовувалися стандартні елементи для Windows застосунків. Було вирішено не ускладнювати процес розробки кастомними формами та елементами інтерфейсу. Стандартні елементи інтерфейсу виглядають знайомими користувачам, та є досить мінімалістичними та мають свій стиль.

Стандартним шрифтом проекту був обраний Segoe UI. У застосунку були використані наступні варіації даного шрифта:

* Segoe UI Light, 24pt, для заголовків
* Segoe UI Light, 20.25pt, для великих кнопок
* Segoe UI, 14.25pt, для маленьких кнопок
* Segoe UI, 12pt, для звичайного тексту

Даний шрифт був обраний, тому що він є один із стандартних, виглядає досить сучасно, та на нього приємно дивитися.

Основні кольори проекту:

* Стандартний DarkCyan, #008B8B, для головних кнопок
* Білий, #000000, для звичайних кнопок, та фону форм

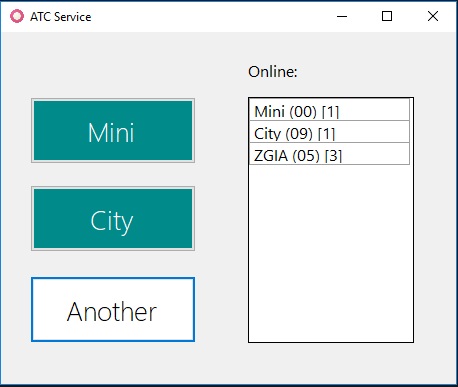
Кольори для кнопок виклику та відміни для інтерфейсу абонента:

* Зелений, #4CD964
* Красний, #F53333

### 6.3.1 Користувацькі інтерфейси

**Інтерфейс “ATC Service”**

Дивись screenshot цього інтерфейсу на рис. 6.3.1.1



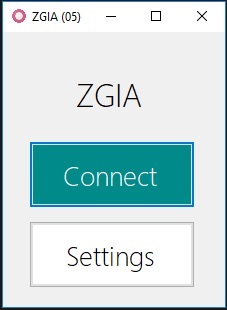
*Рис.6.3.1.1 Screenshot інтерфейсу ‘ATC Service’*

Елементи інтерфейсу:

* Блакитна кнопка “Mini”.
  + При натисканні на цю кнопку система відкриває інтерфейс для АТС “Mini”
* Блакитна кнопка “City”.
  + При натисканні на цю кнопку система відкриває інтерфейс для АТС “City”
* Біла кнопка “Another”.
  + При натисканні на цю кнопку система відкриває інтерфейс “ATC Login”
* Список “Online”
  + Відображає список поточних з’єднаних АТС.

**Інтерфейс для ATC**

Дивись screenshot цього інтерфейсу на рис. 6.3.1.2



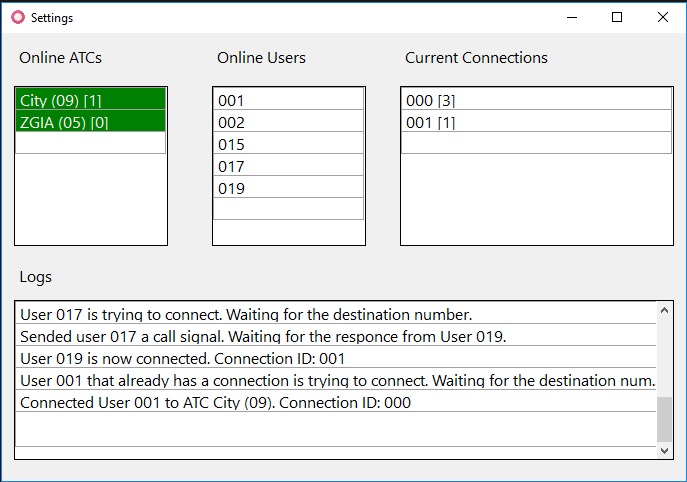
*Рис.6.3.1.2 Screenshot інтерфейсу ‘ATC Screen’*

Елементи інтерфейсу:

* Блакитна кнопка “Connect”.
  + При натисканні на цю кнопку система відкриває інтерфейс “LoginScreen”
* Біла кнопка “Settings”.
  + При натисканні на цю кнопку система відкриває інтерфейс “Settings”

**Інтерфейс “Settings”**

Дивись screenshot цього інтерфейсу на рис. 6.3.1.2



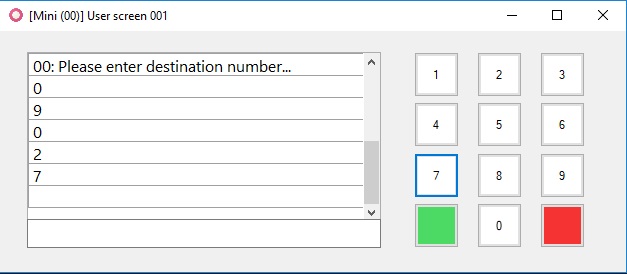
*Рис.6.3.1.2 Screenshot інтерфейсу ‘ATC Screen’*

Елементи інтерфейсу:

* Список “Online ATCs”
  + Відображає список поточних з’єднаних АТС з поточную АТС.
* Список “Online Users”
  + Відображає список поточних з’єднаних користувачів з поточную АТС.
* Список “Current Connections”
  + Відображає список поточних з’єднань поточної АТС.
* Список “Logs”
  + Відображає журнал подій поточної АТС.

**Інтерфейс “User Screen”**

Дивись screenshot цього інтерфейсу на рис. 6.3.1.3



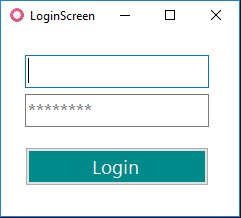
*Рис.6.3.1.3 Screenshot інтерфейсу ‘User Screen’*

Елементи інтерфейсу:

* Список “Messages”
  + Відображає список повідомлень.
* Текст-бокс “Message”
  + Користувач може вводити повідомлення в даний текст-бокс, та відправляти натиснувши кнопку Enter.
* Кнопки з цифрами 0-9
  + Кнопки для набору номеру.
* Красна кнопка
  + Кнопки «відмінити», для скасування поточного зв’язку.
* Зелена кнопка
  + Кнопки «подзвонити», для передавання сигналу “Phone” поточній АТС, для виклику іншого абоненту, або прийняття вхідного дзвінка.

**Інтерфейс “Login Screen”**

Дивись screenshot цього інтерфейсу на рис. 6.3.1.4



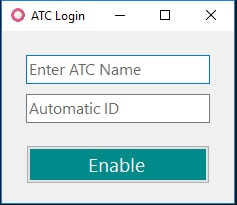
*Рис.6.3.1.4 Screenshot інтерфейсу ‘Login Screen’*

Елементи інтерфейсу:

* Текст-бокс “Ідентифікатор”
  + Користувач може вводити свій ідентифікатор в даний текст-бокс.
* Текст-бокс “Пароль”
  + Користувач може вводити свій пароль в даний текст-бокс.
* Блакитна кнопка “Login”
  + Натиснувши кнопку, користувач може запросити авторизацію, використовуючи введені дані.

**Інтерфейс “ATC Login Screen”**

Дивись screenshot цього інтерфейсу на рис. 6.3.1.5



*Рис.6.3.1.5 Screenshot інтерфейсу ‘ATC Login Screen’*

Елементи інтерфейсу:

* Текст-бокс “ATC Name”
  + Користувач може вводити ім’я бажаної АТС в даний текст-бокс.
* Текст-бокс “Ідентифікатор”
  + Користувач може вводити ідентифікатор бажаної АТС в даний текст-бокс, якщо користувач на введе ніякого ідентифікатора, система створить нову АТС з новим ідентифікатором.
* Блакитна кнопка “Enable”
  + Натиснувши кнопку, користувач може запросити вмикання АТС, використовуючи введені дані.

# 7 Реалізація і тестування

Об’єм коду:

С#:

* 95.7 KB
* 2589 рядків

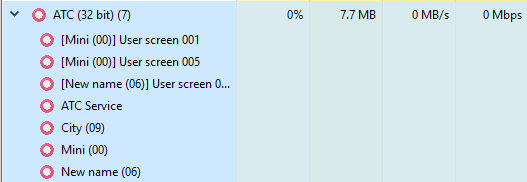
Кількість екранів: 6. Див. розділ 6.3.1

Кількісні дані файлів проекту Visual Studio:

* 1960 КБ
* 65 файлів
* 14 папок

Кількісні дані готової програми:

* 663 КБ
* 14 файлів



*Рис.7.1 Screenshot Task Manager Windows при роботі застосунку*

Як видно на рис. 7.1, витрати оперативної пам’яті задовільнять вимогам пункту 7.2.

Тестування застосунку проводилося вручну на протязі однієї години. Як результат було знайдено та виправлено декілька багів.

Висновок: система реалізована успішно та відповідає наданим вимогам та завданню.

# Висновки

Таким чином в процесі курсової роботи мною було

1. закріплено навички володіння мовою C#
2. закріплені навички використання парадигми програмування ООП
3. закріплені знання деяких design patterns
4. вивчено основи технологій телефонних мереж
5. розроблено програмний Windows-застосунок на мові С# використовуючи інтегроване середовище розробки Visual Studio 2014
6. написано дану курсову роботу, що підвищило мої навички документування розроблених програмних застосунків

# Список літератури

1. Абилов А. В., Сети связи и системы коммутации. — Ижевск: Изд-во ИжГТУ, 2002 г.
2. Г.П. Башарин, А.Д. Харкевич, М.А. Шнепс-Шнеппе. Массовое обслуживание в телефонии. — М.: Наука, 1968. — 246 с.
3. Гольдштейн. Сигнализация в сетях связи. Протоколы сети доступа. — Москва: «Радио и связь»
4. Пышкин Е.В. Основные концепции и механизмы объектно-ориентированного программирования СПб.: БХВ-Петербург, 2005. — 640 с. — ISBN:5-94157-554-8
5. Роберт С. Мартин, Мика Мартин. Принципы, паттерны и методики гибкой разработки на языке C#. — СПб.: Символ-плюс, 2011. — С. 8. — 768 с. — ISBN 978-5-93286-197-4.
6. Kindler, E.; Krivy, I. (2011). "Object-Oriented Simulation of systems with sophisticated control". International Journal of General Systems: 313–343.
7. Lewis, John; Loftus, William (2008). Java Software Solutions Foundations of Programming Design 6th ed. Pearson Education Inc. [*ISBN*](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [*0-321-53205-8*](https://en.wikipedia.org/wiki/Special:BookSources/0-321-53205-8)., section 1.6 "Object-Oriented Programming"