

Черкаський національний університет імені Богдана Хмельницького

Кафедра програмного забезпечення автоматизованих систем

КУРСОВА РОБОТА

з дисципліни «Об'єктно-орієнтоване програмування»

НА ТЕМУ «Моделювання роботи станції швидкої допомоги»

Студента _____ 2 _____ курсу, групи _____ КС-19

спеціальності _____ «Інженерія

_____ програмного забезпечення»

_____ Джеріхова Івана Олеговича

Керівник _____ старший викладач Гребенович
Ю.Є.._____

Національна шкала _____

Кількість балів: _____ Оцінка: ECTS _____

Члени комісії

(підпис)

(прізвище та ініціали)

(підпис)

(прізвище та ініціали)

(підпис)

(прізвище та ініціали)

м. Черкаси – 2021 рік

Зміст

Вступ.....	4
Розділ 1. Огляд систем масового обслуговування. Вибір технологій для реалізації програми.	6
1.1. Теорія масового обслуговування. Системи масового обслуговування (СМО), їх види.....	6
1.2. Розгляд деяких існуючих моделей СМО	8
1.3. Вибір системи масового обслуговування для поставленої задачі	9
1.4. Вибір патерну проектування програми	10
1.5. Обґрунтування вибору мови програмування та середовища розробки	11
1.6. Висновки до розділу	11
Розділ 2. Проектування станції швидкої допомоги	13
2.1. Проектування ієрархії класів	13
2.2. Спрощений алгоритм роботи станції ШМД.....	14
2.3. Опис алгоритму надходження заявки пацієнта до системи	15
2.4. Опис алгоритму призначення бригади на виклик	16
2.5. Опис алгоритму емуляції поїздки на виклик	16
2.6. Реалізація принципів ООП, діаграма класів.....	17
2.7. Висновки до розділу	18
Розділ 3. Програмна реалізація продукту	19
3.1. Реалізація контролерів бригад та пацієнтів.....	19
3.2. Реалізація моделей	20
3.3. Реалізація файлового менеджера	20
3.4. Реалізація та огляд графічного інтерфейсу	20

3.5. Тестування програми	24
3.6. Висновки стосовно реалізації	27
Висновки	28
Список використаної літератури	29
Додаток А. Блок-схеми алгоритмів	30
Додаток Б. Програмний код реалізації продукту.....	34

Вступ

Здоров'я – найцінніша річ, що є в людини. З давніх часів підтримання свого тіла і духу у тонусі були невід'ємною частиною культури різних народів. Звичайні для наших часів хвороби лікували народними методами. Проте в ситуаціях, коли здоров'ю було завдано серйозної шкоди і потребувалась екстрена допомога, щось зробити самому було майже неможливо. Це і стало головною причиною появи станцій швидкої медичної допомоги.

Перший прототип станції з'явився ще в XI ст. в Єрусалимі ченцями при монастирі св. Іоана, з якого через декілька століть візьме свій початок орден Госпітальєрів, завдяки якому у хрестових походах була опіка над пораненими бійцями.

Першу стаціонарну станцію швидкої медичної допомоги створив віденський професор Яромир Мунді у 1883 році. Уряд Австрії виділив фінанси на цей заклад, оскільки кількома роками раніше через відсутність швидкої допомоги загинуло близько 500 глядачів “Комічної опери” у результаті пожежі.

В Україні перша станція відкрилась у 1902 році. Транспорт закладу складався з трьох пар коней з каретами, через десять років було додано 2 санітарні автомобілі. [1]

З тих часів багато чого змінилось. Зараз станції швидкої допомоги є майже у кожному населеному пункті. Диспетчерська служба викликається за єдиним номером в усій країні (103 або 112 для всіх екстрених служб).

З теоретичної точки зору, робота станції ШМД є одним з прикладів системи масового обслуговування (СМО). Існує декілька видів та критеріїв таких систем, які будуть розглянуті далі.

Тему курсової роботи було обрано через цікавість та перспективність поставленої задачі.

Мета роботи: дослідження існуючих систем масового обслуговування та власна програмна реалізація системи станції швидкої медичної допомоги.

Завдання даної роботи: проаналізувати інформаційні джерела, описати алгоритм роботи станції, написати програму емуляції роботи станції швидкої медичної допомоги, провести тестування продукту.

Розділ 1. Огляд систем масового обслуговування. Вибір технологій для реалізації програми.

1.1. Теорія масового обслуговування. Системи масового обслуговування (СМО), їх види.

Теорія масового обслуговування – це наука, що досліджує вибір структури системи обслуговування та процесів обслуговування, виходячи з параметрів потоків обслуговування, очікування, черг та ін..

Перші проблеми теорії почав розглядати Агнер Ерланг, співробітник телефонної компанії в Копенгагені на початку ХХ століття, якому доручили завдання упорядкування роботи телефонної станції.[2]

Система масового обслуговування (СМО) - це система, яка обслуговує заявки, що надходять до неї.

Характерні риси СМО (рис. 1.1):

1. Випадкове надходження потоку заявок на обслуговування.
2. Черги заявок.
3. Пристрої, які здійснюють обслуговування цих заявок.

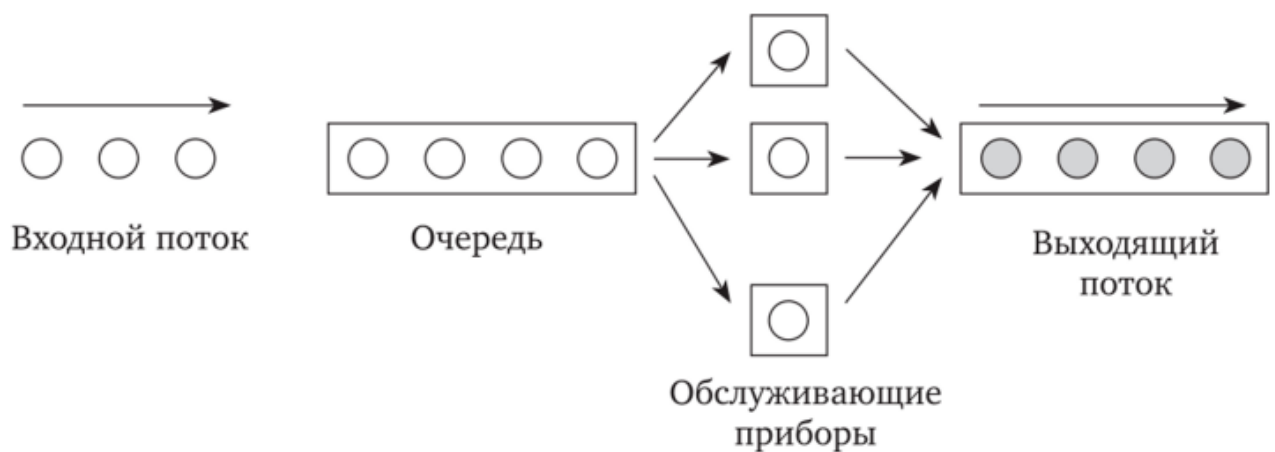


Рис 1.1. Схема роботи системи масового обслуговування

Прикладами систем масового обслуговування є станція швидкої медичної допомоги, телефонна станція, диспетчерська служба таксі.

Існує велика кількість класифікацій систем масового обслуговування за різними ознаками.

1. За кількістю каналів обслуговування:

- a. одноканальна СМО(один канал для обробки заявок).
- b. багатоканальна СМО, яка дозволяє обробляти декілька заявок одночасно.

2. За наявністю черг:

- a. СМО без черги (з відмовами). При надходженні заявки у момент зайнятості усіх каналів, заявка втрачається і не обслуговується.
- b. СМО з чергою (з очікуванням). При надходженні заявки у момент зайнятості усіх каналів, заявка стає в чергу і чекає свого часу для обслуговування.

У той час, за характеристиками черга поділяється на:

- i. за довжиною черги накопичувача: обмежена або безлімітна. При обмеженій довжині черги заявка зникає після досягнення ліміту накопичувача.
- ii. за часом очікування: обмежений або необмежений. При варіанті з обмеженим часом очікування заявка втрачається після певного проміжку, за який вона не була обслугована. Необмежений час передбачає відсутність втрати заявки незалежно від часу очікування.
- iii. за принципом обробки заявок: обслуговування першої заявки в першу чергу (принцип FIFO), обслуговування останньої заявки в першу чергу (LIFO), обслуговування заявок у випадковому порядку (FIRO), обслуговування з пріоритетами.[3]

1.2. Розгляд деяких існуючих моделей СМО

Одноканальна система з відмовами (рис. 1.2). Прикладом такої черги є, наприклад директор певної компанії, який обслуговує різного виду заявки (розмови з підлеглими, заповнення документів та ін.)

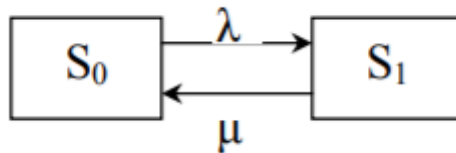


Рис 1.2. Схема роботи одноканальної СМО з відмовами

Кількість можливих станів такої СМО – 2, канал або вільний і готовий прийняти заявку, або зайнятий і заявки відхиляються.

Одноканальна система з необмеженою чергою (рис. 1.3). Прикладом такої системи можна назвати телефонну будку з одним апаратом всередині.

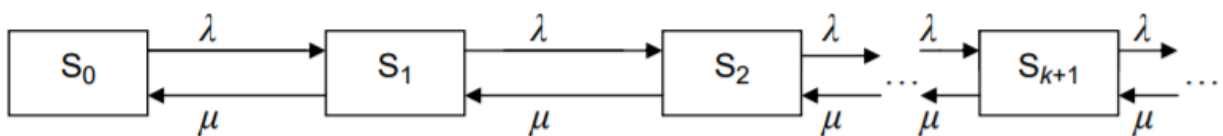


Рис 1.3. Схема роботи одноканальної СМО з необмеженою чергою

Кількість можливих станів такої СМО безліч через необмежену довжину черги.

Багатоканальна система з відмовами (рис. 1.4). Прикладом такої системи є телефонна станція (АТС).

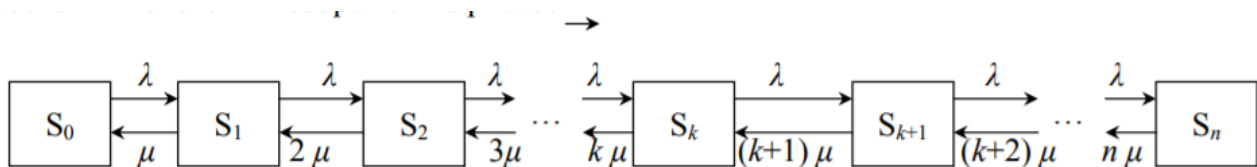


Рис 1.4. Схема роботи багатоканальної СМО з відмовами

Стани системи змінюються від усіх вільних каналів (S_0) до стану S_n з усіма зайнятими каналами.

Багатоканальна система з обмеженою чергою (рис. 1.5.). Прикладом також може слугувати АТС.

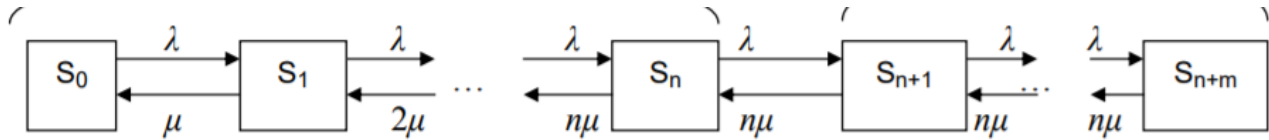


Рис 1.5. Схема роботи багатоканальної СМО з обмеженою чергою

Стани системи змінюються від усіх вільних каналів (S_0) до стану S_n з усіма зайнятими каналами. Потім починається черга при стані S_{n+1} і закінчується станом S_{n+m} , де m – довжина черги.

Багатоканальна система з необмеженою чергою (рис. 1.6). Прикладом слугують ті сервіси, у яких відмовити в обслуговуванні заявки не можна.

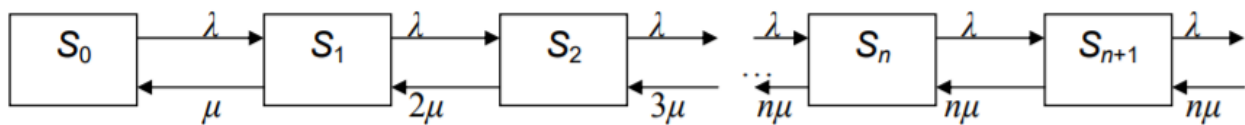


Рис 1.4. Схема роботи багатоканальної СМО з обмеженою чергою

Така система має безліч станів через необмежену довжину черги. змінюються від усіх вільних каналів (S_0) до стану S_n з усіма зайнятими каналами.

1.3. Вибір системи масового обслуговування для поставленої задачі

Для вибору СМО для моделювання роботи станції швидкої допомоги слід розглянути описані в розділах 1.1 та 1.2 критерії.

За кількістю каналів обслуговування підходить багатоканальний варіант. Одноканальна система була відкинута через специфіку роботи швидкої допомоги, яка мусить обслуговувати всіх клієнтів в декілька паралельних потоків.

За наявністю черги слід обрати варіант з наявністю такої, оскільки через те, що мова йде про життя людей, відмови в обслуговуванні вимоги не може бути. Це означає, що втрати заявок є недопустимими.

Довжина черги накопичувача має бути умовно безлімітною, знову ж таки, через неможливість відмови.

Був обраний принцип роботи із заявками під назвою FIRO з деякими поправками. Заявка вибирається не стовідсотково випадково, а на розгляд диспетчера системи.

Команда медиків являтиме собою канал СМО, а всі об'єкти бригад, відповідно, складають усю множину каналів. В свою чергу, пацієнт – це заявка, яка поступає до СМО. Його «обробкою», тобто лікуванням, і займається кожна з команд-каналів.

Кількість каналів регулюватиметься зміною кількості бригад медиків.

1.4. Вибір патерну проектування програми

При проектуванні програми орієнтиром буде патерн MVC (Model-View-Controller). MVC – схема розділу додатку на три частини: модель надає данні та виконує команди контролера, змінюючи свій стан, представлення(вид) – відображення даних моделей користувачу, реагуючи на зміни моделей, контролер – сповіщує модель про зміни при певних діях користувача з частиною представлення[4]. Рис. 1.5. схематично пояснює вище описаний текст.

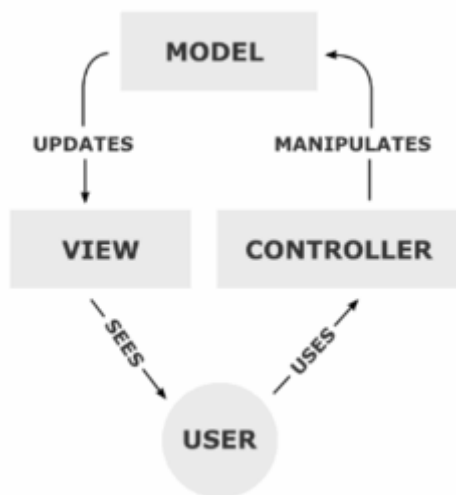


Рис. 1.5. Схема роботи програм з патерном MVC

1.5. Обґрунтування вибору мови програмування та середовища розробки

Вибір мови програмування, яку буду використовувати для написання програмного продукту, пав на C#, оскільки інші не вивчав.

Для створення графічного інтерфейсу планується використання бібліотеки Windows Forms. Вона є простою у освоєнні, її ресурсів цілком вистачить для створення MDI-додатку з патерном MVC. Іншим варіантом було використання фреймворку ASP.NET MVC, який призначений для створення проектів з патерном MVC, але більше підходить для веб-додатків.

За середовище розробки було обране Microsoft Visual Studio 2019 Community. Перевагою є безкоштовність, що ідеально підходить для розробки проектів у навчальних цілях.

Середовище JetBrains Rider відпало через відсутність нормальної роботи з Windows Forms.

1.6. Висновки до розділу

В даному розділі розглянуті теоретичні відомості щодо систем масового обслуговування. Розглянуто основні моделі систем та обґрунтований вибір однієї з них, а саме багатоканальної СМО з нескінченною чергою. Виконаний вибір патерна проектування програми.

Також обрано та обґрунтовано мову програмування, бібліотеку для створення графічного інтерфейсу та середовище розробки.

Розділ 2. Проектування станції швидкої допомоги

2.1. Проектування ієрархії класів

Для моделювання станції швидкої допомоги за патерном MVC треба спроектувати ієрархію класів в залежності від типу класу.

Моделі:

Клас Brigade. Являтиме собою абстрактний клас з базовим конструктором властивостей, таких як id бригади, спеціалізація, статус доступності, швидкість карети швидкої допомоги та кількість вилікуваних пацієнтів. Крім того, матимуться абстрактні методи емуляції поїздки та лікування хворого, які поліморфно реалізоватимуться у нащадках.

Класи HQBrigade та LQBrigade. Існуватиме два типи спеціалізацій бригад для двох типів заявок: для хвороб високої складності або екстрених випадків – висококваліфікована бригада (далі – HQ), для рядових викликів – звичайна бригада (далі – LQ). Кожна з бригад може обслуговувати лише заявку своєї складності. Класи HQ та LQ-бригад унаслідуються від абстрактного класу Brigade.

Patient, PatientGenerator. У першому будуть описані властивості пацієнта, такі як ПІБ, час виклику, дистанція від нього до станції ШМД, хвороба, з якої визначиться пріоритет виклику. Клас PatientGenerator являтиме собою генератор рандомних пацієнтів, у якому випадково будуть надані значення властивостям, що описані в класі Patient. Також тут буде задане часове значення появи нової заявки пацієнта.

Клас **Illness** являтиме собою модель хвороби, що матиме поля назви, пріоритетності та часу лікування цієї хвороби.

Клас **BaseObjectInizializer** відповідатиме за створення базових списків бригад та хвороб, якщо такі відсутні.

Контролери:

BrigadeController – клас-контролер, який здійснюватиме різні дії над об'єктами бригад, такі як створення або розформування, підвищення рангу бригади, відправлення бригади на виклик.

PatientController відповідатиме за надходження заявок пацієнтів до станції, викликаючи метод створення пацієнта з класу **PatientGenerator**.

Представлення:

1. Форма зі списком об'єктів пацієнтів
2. Форма зі списком об'єктів бригад
3. Форма призначення бригади на виклик

Окремі класи:

Необхідно передбачити серіалізацію списку об'єктів бригад з їх властивостями, оскільки продукт передбачатиме створення нових об'єктів, інформація про які не повинна зникнути. За це відповідатиме клас **FileManager**, у якому будуть методи збереження та загрузки списку бригад з файлу.

2.2. Спрощений алгоритм роботи станції ШМД

На рис. 2.1. зображений спрощений алгоритм роботи станції ШМД. На блок-схемі не враховані тупикові ситуації, як-то нестача бригад, але в програмі буде реалізована черга очікування бригади.

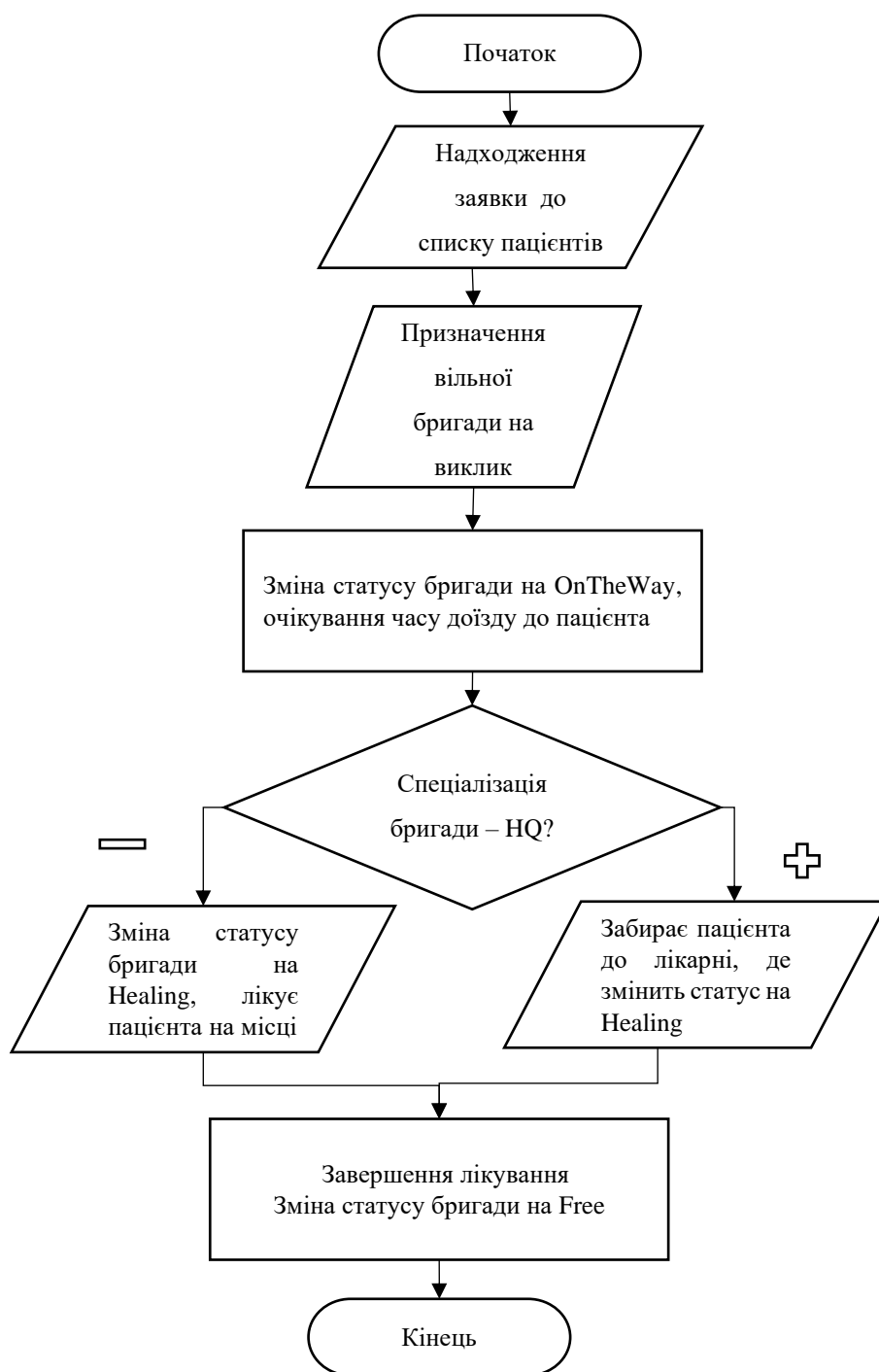


Рис 2.1. Блок-схема спрощеного алгоритму роботи станції

2.3. Опис алгоритму надходження заявки пацієнта до системи

Для емуляції надходження заявки пацієнта до станції ШМД було придумано такий алгоритм:

1. З заздалегідь створеного списку перелічень імен та прізвищ випадковим чином обирається по одному.
2. Випадковим чином задається хвороба пацієнта та дистанція до нього.
3. Задається інтервал появи нового пацієнта методом випадкового вибору (від 15 до 30 секунд).
4. Після закінчення часового інтервалу очікування заявка потрапляє до списку пацієнтів.

Блок-схема алгоритму знаходиться в додатку А.

2.4. Опис алгоритму призначення бригади на виклик

Після того, як до станції надійшла заявка пацієнта, треба призначити бригаду на виклик.

Для цього був розроблений алгоритм, який обирає бригаду в залежності від хвороби пацієнта.

Відсилається запит на виведення списку доступних бригад. Якщо зараз вільних бригад потрібної спеціалізації немає, діалогове вікно повідомить про це.

При наявності екіпажів відкривається форма зі списком вільних команд. Після підтвердження вибору бригади в окремому потоці запускається метод емуляції поїздки до пацієнта.

Блок-схема алгоритму знаходиться в додатку А.

2.5. Опис алгоритму емуляції поїздки на виклик

Абстрактний метод емуляції поїздки заданий в абстрактному класі Brigade, його реалізація в класах HQBrigade та LQBrigade дещо відрізняється.

Бригада призначена на виклик, починається емуляція поїздки екіпажа до пацієнта. Спочатку вираховується час подорожі екіпажу в одну та дві сторони, після чого у MessageBox для LQBrigade виводиться повідомлення про приблизний час зайнятості бригади.

Потім викликається затримка виконання асинхронного методу лікування пацієнта. Часовий інтервал затримки дорівнює часу подорожі до пацієнта для

об'єкту LQBrigade та часу подорожі туди-назад для об'єкту HQBrigade. Після цього для об'єкту HQBrigade виводиться MessageBox з приблизним часом лікування пацієнта. Після завершення лікування пацієнта виводиться відповідне повідомлення, змінюється статус бригади на Free, а об'єкту пацієнта присвоюється значення null.

Різниця методів об'єктів HQBrigade та LQBrigade полягає у тому, що у першому випадку лікування пацієнта відбувається у лікарні, а для другого об'єкта процес лікування відбувається вдома у пацієнта. Відповідно, затримки для емуляції подорожі або лікування у цих методах викликаються в різні моменти.

Блок-схеми для об'єкту HQBrigade розташована в додатку А.

2.6. Реалізація принципів ООП, діаграма класів

Одним з прикладів інкапсуляції (приховання деталей реалізації методу) у даному програмному продукті є надання полям класу FileManager модифікатора доступу private (рис. 2.2.)

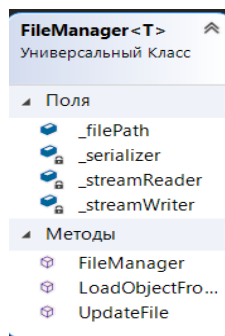


Рис. 2.2. Приклад інкапсуляції

Наслідування (успадкування класами методів чи полів іншого класу) в цьому проекті представлено абстрактним батьківським класом Brigade та його нащадками HQBrigade та LQBrigade, які використовують базовий конструктор.

Поліморфізм в цій програмі використовується для перевизначення методів класу Brigade в бригадах-нащадках.

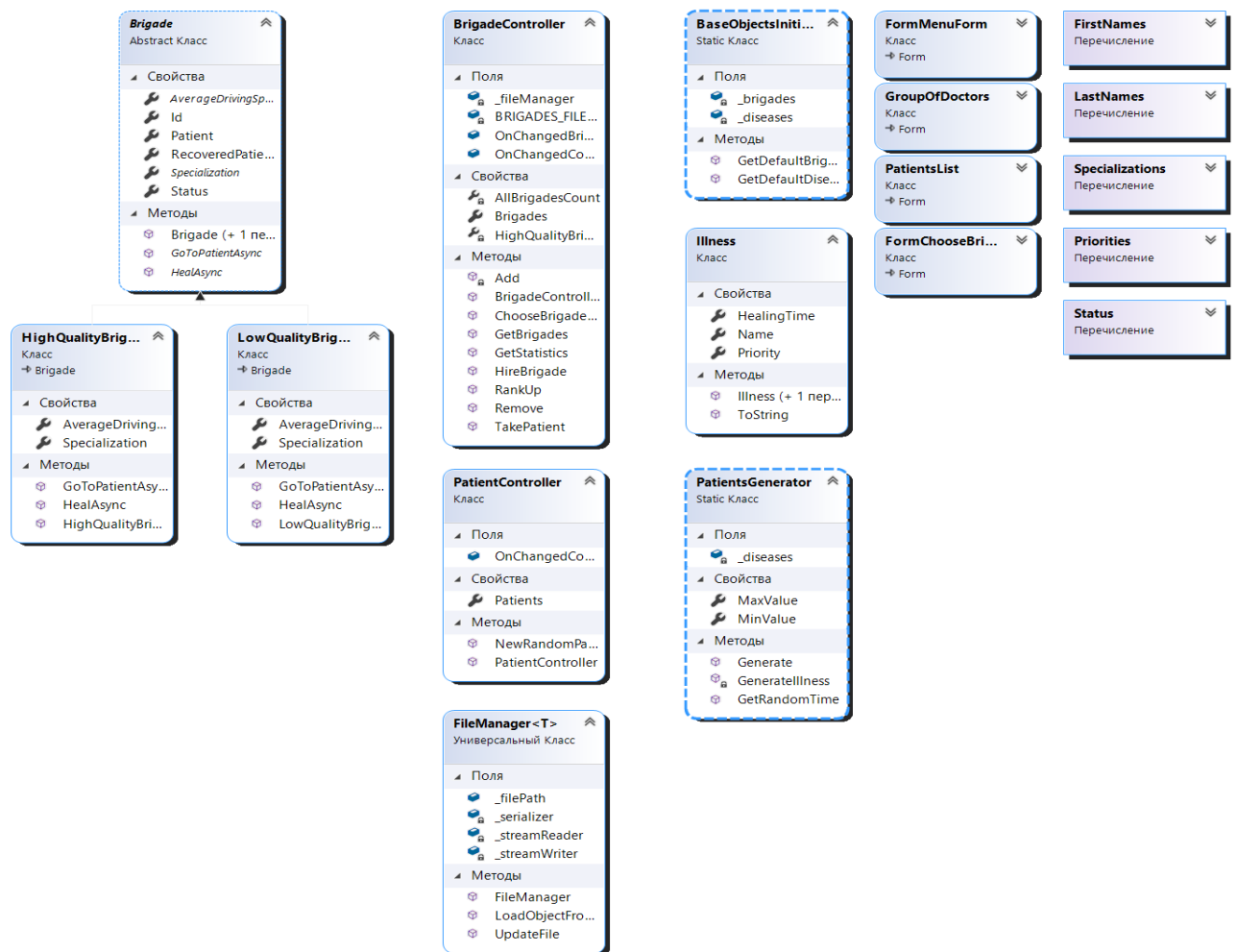


Рис. 2.3. Діаграма класів проекту

Всю ієрархію класів можна побачити на діаграмі класів (рис. 2.3.)

2.7. Висновки до розділу

В даному розділі було спроектовано ієрархію класів програми, описані основні алгоритми, пояснена реалізація принципів ООП та побудована діаграма класів.

Розділ 3. Програмна реалізація продукту

3.1. Реалізація контролерів бригад та пацієнтів

У контролерах прописана логіка керування об'єктами-моделями.

Клас `BrigadeController` дозволяє виконувати різні операції над об'єктами бригад. Наявні методи підвищення рангу бригади, створення та розформування бригади, метод призначення бригади на виклик, метод емуляції поїздки бригади. Розглянемо детальніше деякі з них.

Метод `Add`, що відповідає за додання нової бригади до списку, в якості параметра приймає будь-яку кількість об'єктів бригад. Також в якості параметра можна передати як `HQBrigade`, так і `LQBrigade`, які зміняться на `Brigade`. У методі додається нова бригада до списку, оновлюється файл зі списком та викликається делегат `OnChangedCount`, який сповіщає усіх підписників про зміни в кількості бригад.

Асинхронний метод `TakePatient` відповідає за емуляцію поїздки бригади на виклик. Всередині нього викликаються методи бригад `GoToPatientAsync` та `HealAsync`, описані у розділі 2.5.

Лістинг класу в додатку Б.1.

Контролер пацієнтів `PatientController` відповідає за дії з об'єктами пацієнтів, а саме на рандомне надходження заявок та додання до списку. Розглянемо метод `NewRandomPatient`. Відбувається звернення до генератору рандомних пацієнтів, робота якого була описана в розділі 2.3., після додавання пацієнту до списку видається звуковий сигнал, що сповіщує про нову заявку, а делегат `OnChangedCount` сповіщає підписників про зміну кількості пацієнтів.

Лістинг класу в додатку Б.2.

3.2. Реалізація моделей

Класи моделей (Brigade, HQBrigade, LQBrigade, Patient, PatientGenerator, BaseObjectGenerator) були реалізовані за проектувальним описом з розділу 2.1, деякі з них також і за описами алгоритмів з розділів 2.3-2.5.

Лістинги цих класів розташовані у додатку Б.3-9.

3.3. Реалізація файлового менеджера

FileManager являє собою параметризований клас, який відповідає за збереження та подальше зчитування списку бригад з XML-файлу. Метод UpdateFile оновлює об'єкти у файлі за допомогою серіалізатора, а метод LoadObjectFromFile десеріалізує об'єкти з файлу. До файлу серіалізуються такі атрибути бригад, як id, спеціалізація, статус та кількість вилікуваних пацієнтів. Вигляд запису об'єкту бригади у файлі зображено на рис. 2.1.

```
<Brigade xsi:type="HighQualityBrigade">
  <Id>1</Id>
  <Specialization>HighQualityBrigade</Specialization>
  <Status>Free</Status>
  <RecoveredPatients>3</RecoveredPatients>
  <AverageDrivingSpeed>100</AverageDrivingSpeed>
</Brigade>
```

Рис. 2.1. Вигляд запису бригади в XML-файлі.

Лістинг класу можна побачити у додатку Б.10.

3.4. Реалізація та огляд графічного інтерфейсу

Користувацький інтерфейс продукту складається з чотирьох форм.

Стартове вікно – форма FormMenuForm (рис. 3.2).

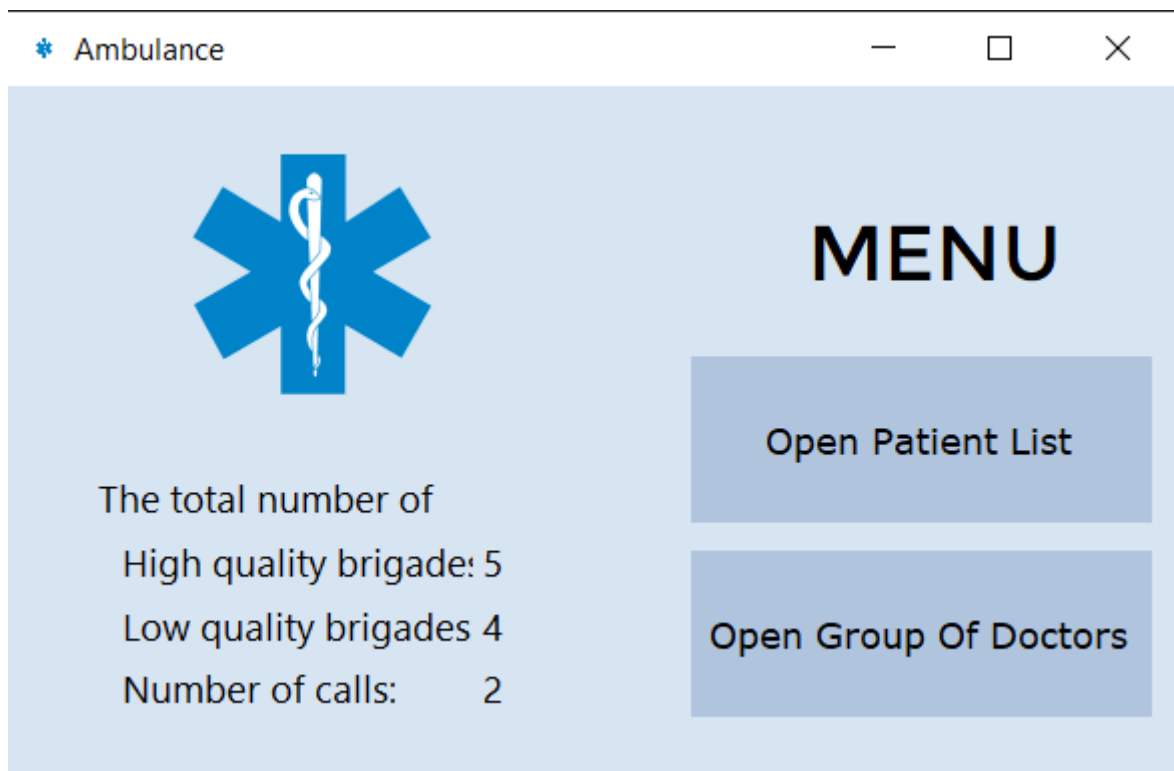


Рис. 3.2. Стартова форма програми

Вікно містить кнопки з написами «Open Patient List» та «Open Group Of Doctors», які відповідають за відкриття форм зі списками пацієнтів та бригад. Крім того, наявне так зване інформаційне табло, на якому виводиться загальна кількість бригад кожного типу, а також кількість непризначених заявок. Табло працює за рахунок змін значень делегату `OnChangedCount`, які відбуваються у контролерах бригад та пацієнтів при створенні або видаленні нових об'єктів.

Форма `FormPatientList` зображає інформацію щодо заявок пацієнтів, що надходять до програми (рис. 3.3):

List of receipts					Set Brigade
CallTime	FullName	Distance	Illness	Priority	
25.05 16:38:22	Олійник Богдан	9	Грип	High	
25.05 16:38:22	Савченко Давид	34	Онко	Low	

Рис. 3.3. Форма заявок пацієнтів

Список заявок реалізований за допомогою вбудованого до Windows Forms класу DataGridView, що дозволяє виводити дані зі списку пацієнтів BindingList у кастомізовану таблицю.

Сам BindingList – це не простий список, а список, до якого прив'язаний DataGridView, що дозволяє при кожній зміні в BindingList автоматично додавати\видаляти об'єкти.

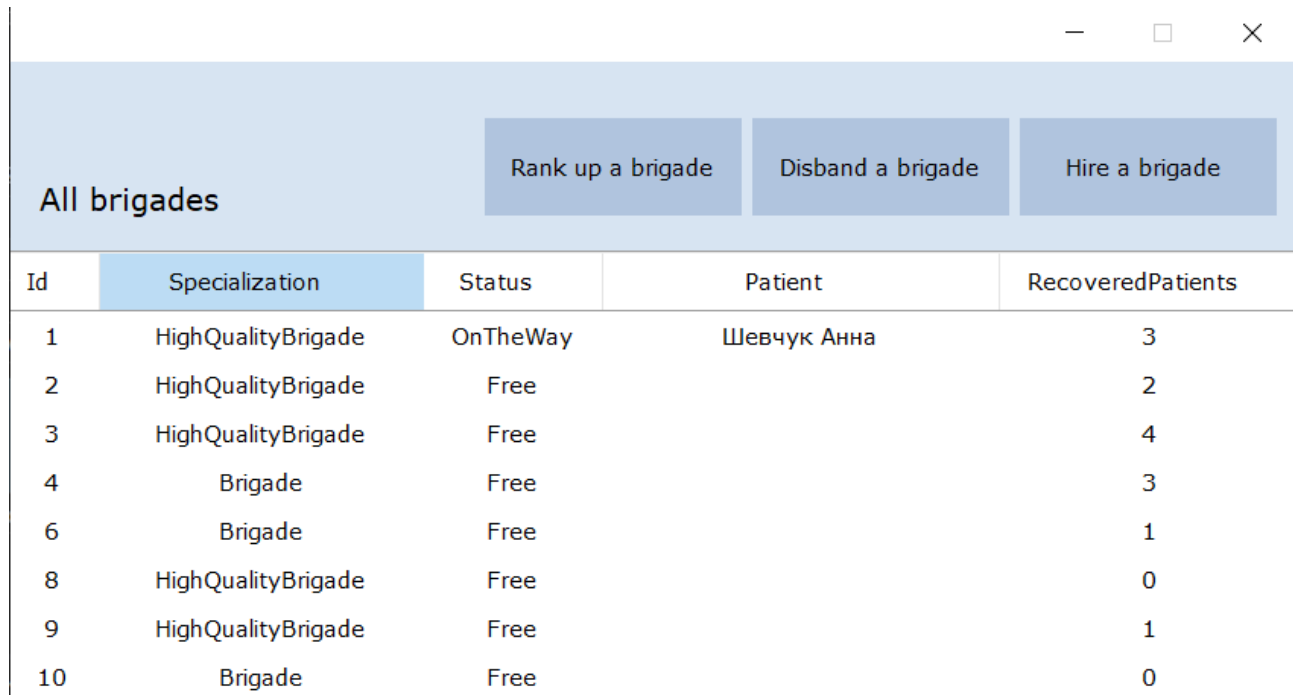
При відкритті форми ініціалізуються контролери бригад та пацієнтів, визначається джерело даних для таблиці, а також оформлюється підписка на подію SelectionChanged. Ця подія відповідає за виклик методу, що буде показувати або приховувати кнопку Set Brigade, при зміні виділення рядку таблиці. Також задається параметр CancellationTokenSource, який відповідає за призупинення потоку пацієнтів при закритті даної форми. Після відкриття форми запускається асинхронний метод, який починає генерацію пацієнтів через контролер. Лістинг методів класу зображений в додатку Б.11.

При натисненні на кнопку Set Brigade постає форма вибору бригади (рис. 3.4), на якій у вигляді таблиці зображений список вільних на цей момент бригад і кнопка призначення бригади на виклик, яка повертає до форми з пацієнтами сповістку про те, що бригада була призначена на виклик, відповідно, дані про пацієнта можна прибирати з таблиці.

Choose a free brigade:			To appoint
Id	Specialization	Status	RecoveredPatients
1	HighQualityBrigade	Free	3
2	HighQualityBrigade	Free	2
3	HighQualityBrigade	Free	4
8	HighQualityBrigade	Free	0
9	HighQualityBrigade	Free	1

Рис 3.4. Форма з вибором бригади для призначення на виклик

Остання форма (рис. 3.5.) GroupOfDoctors являє собою вікно контролю над бригадами. Вже згаданий вище клас DataGridView дозволив вивести з файлу список бригад у вигляді таблиці. Метод UpdateGrid задає параметри вигляду таблиці та пропонує створити першу бригаду, якщо вони взагалі відсутні.



All brigades				
Rank up a brigade Disband a brigade Hire a brigade				
Id	Specialization	Status	Patient	RecoveredPatients
1	HighQualityBrigade	OnTheWay	Шевчук Анна	3
2	HighQualityBrigade	Free		2
3	HighQualityBrigade	Free		4
4	Brigade	Free		3
6	Brigade	Free		1
8	HighQualityBrigade	Free		0
9	HighQualityBrigade	Free		1
10	Brigade	Free		0

Рис. 3.5. Вигляд форми контролю за бригадами

Над таблицею розташовані три кнопки, що дозволяють виконувати різні дії над бригадами. Кнопка Rank up викликає метод підвищення кваліфікації бригади, який перевіряє, чи є нинішня кваліфікація бригади звичайною, а також статус команди. Для підвищення бригада повинна бути вільною в цей момент, інакше виведуться відповідні MessageBox з помилкою виконання операції.

Кнопка Disband a brigade активується при вибраному рядку певної бригади в таблиці, при натисненні звертається до контролера бригад та видаляє її зі списку.

Кнопка Hire a brigade також звертається до контролера, але з протилежною причиною – створення і занесення до списку нової бригади.

Лістинги кнопок розташовані у додатку Б.12.

3.5. Тестування програми

Протестуємо програму (рис. 3.6 – рис. 3.14) у режимі роботи диспетчера станції ШМД: прийняти заявки пацієнтів, виконати різні дії над бригадами.

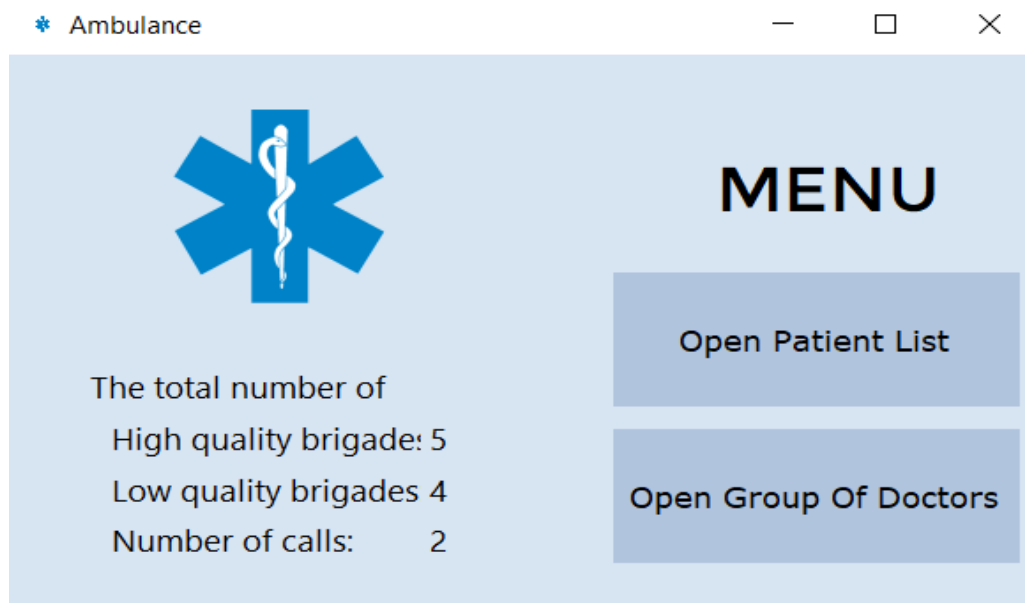


Рис. 3.6. З головного меню перехід до списку пацієнтів

List of receipts					Set Brigade
CallTime	FullName	Distance	Illness	Priority	
31.05 3:44:05	Ковальчук Василь	29	Опик	Low	
31.05 3:44:05	Шевчук Марія	36	Перелом	High	

Рис. 3.7. Вибір пацієнта та натиснення кнопки призначення бригади

Choose a free brigade:			To appoint
Id	Specialization	Status	RecoveredPatient
4	Brigade	Free	3
6	Brigade	Free	1
10	Brigade	Free	0
11	Brigade	Free	0

Рис. 3.8. Вибір будь-якої вільної бригади, натиснення кнопки призначення бригади

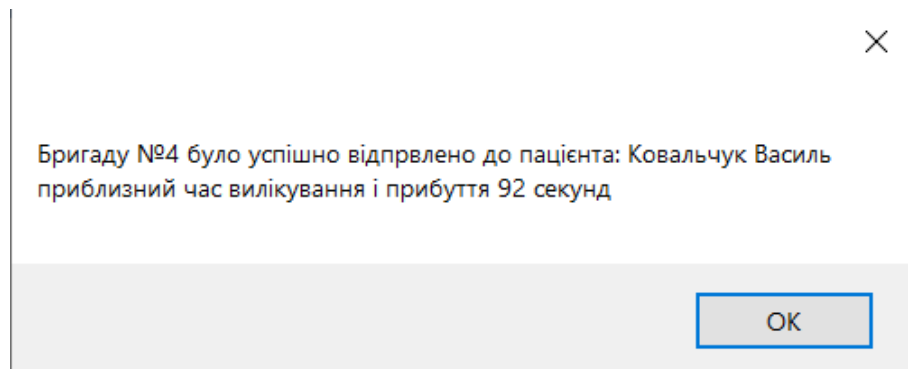


Рис. 3.9. Діалогове вікно підтвердження відправки бригади

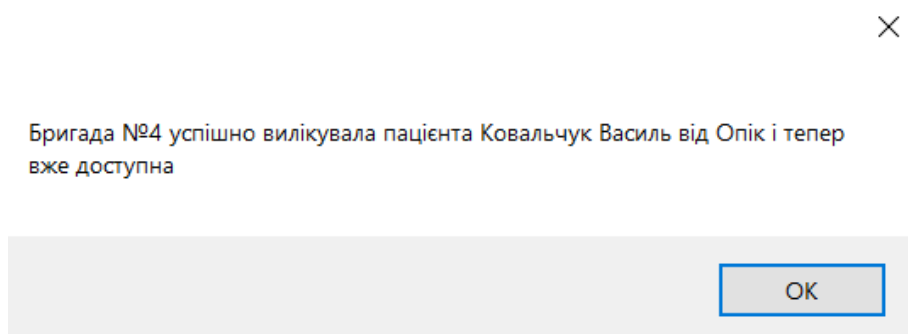


Рис. 3.10. Виведення діалогового вікна завершення процесу лікування

All brigades					Rank up a brigade	Disband a brigade	Hire a brigade
Id	Specialization	Status	Patient		RecoveredPatients		
1	HighQualityBrigade	Free			3		
2	HighQualityBrigade	Free			2		
3	HighQualityBrigade	Free			4		
4	Brigade	HealsThePati...	Ковальчук Василь		3		
6	Brigade	Free			1		
8	HighQualityBrigade	Free			0		
9	HighQualityBrigade	Free			1		
10	Brigade	Free			0		
11	Brigade	Free			0		

Рис. 3.11. Вибір бригади при натисненні на рядок

6	Brigade	Free
8	HighQualityBrigade	Free
9	HighQualityBrigade	Free
10	Brigade	Free
11	HighQualityBrigade	Free

Бригада була підвищена
 OK

Рис. 3.12. Підвищення бригади з id 11

6	Brigade	Free
8	HighQualityBrigade	Free
9	HighQualityBrigade	Free
11	HighQualityBrigade	Free

Видалення бригади
 Бригаду №10 розформовано
 OK

Рис. 3.13. Розформування бригади з id 10

6	Brigade	Free
8	HighQualityBrigade	Free
9	HighQualityBrigade	Free
11	HighQualityBrigade	Free
12	Brigade	Free

Увага!
 Нову бригаду створено
 OK

Рис. 3.14. Створення нової бригади з id 12

3.6. Висновки стосовно реалізації

У цьому розділі була описана реалізація контролерів і моделей продукту, проведений ретельний розбір реалізації графічного інтерфейсу з використанням Windows Forms, а також проведене тестування програми за сценарієм роботи диспетчера станції ШМД.

Висновки

Після аналізу технічного завдання, були проведені пошуки інформації щодо поняття систем масового обслуговування та їх видів, був обраний підходящий варіант.

При проектуванні застосунку була розроблена ієрархія класів, яка відповідає схемі обраного патерну проектування (MVC). Наявність ієрархії класів полегшила створення як загальних, так і конкретних алгоритмів реалізації деяких методів.

При розробці курсового проекту було покращене розуміння принципів об'єктно-орієнтованого програмування, у результаті чого вони були успішно інтегровані до програмного продукту.

Придумані алгоритми були реалізовані з використанням мови C# та бібліотеки Windows Forms, що дозволило створити швидкий, зручний і зрозумілий MDI-додаток.

Продукт має місце для подальшого розвитку його можливостей. Варіанти розширень функціоналу:

- розширена кастомізація бригад;
- звільнення або призначення кожного члена бригади окремо;
- реалізація відмов пацієнтів від обслуговування;
- створення рейтингової системи для кожної з бригад та ін.

Список використаної літератури

1. Стаття на тему «Швидка медична допомога». [Електронний ресурс] - Режим доступу: https://uk.wikipedia.org/wiki/Швидка_медична_допомога Дата доступу 22.05.2021
2. Стаття на тему «Теорія масового обслуговування». [Електронний ресурс] - Режим доступу: https://uk.wikipedia.org/wiki/Теорія_масового_обслуговування Дата доступу 22.05.2021
3. Лекція на тему «Основні поняття теорії масового обслуговування». - [Електронний ресурс] Режим доступу: https://er.nau.edu.ua/bitstream/NAU/21000/23/Лекція_CA_12.pdf Дата доступу 23.05.2021
4. Стаття на тему «Model-View- Controller» . - [Електронний ресурс] Режим доступу: <https://ru.wikipedia.org/wiki/Model-View-Controller> Дата доступу 24.05.2021
5. Супруненко О. О. Методичні вказівки до виконання та оформлення курсової роботи з дисципліни «Об'єктно-орієнтоване програмування» для студентів, які навчаються за напрямом підготовки 050101 – «Комп'ютерні науки», 050103 – «Програмна інженерія» та 040303 – «Системний аналіз» усіх форм навчання. / О. О. Супруненко, Ю. Є. Гребенович. – Черкаси: ЧНУ імені Богдана Хмельницького, 2013. – 40 с.

Додаток А. Блок-схеми алгоритмів

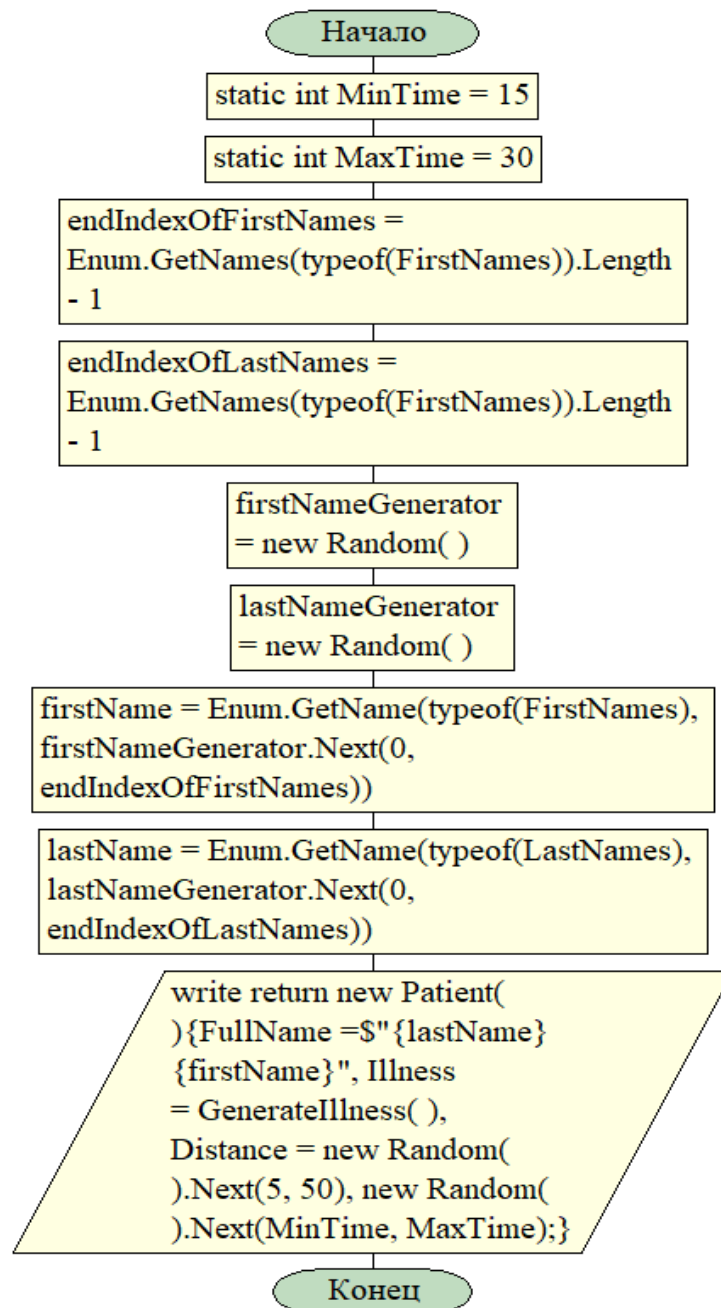


Рис. А.1. Блок-схема алгоритму генерації об'єкта пацієнта.

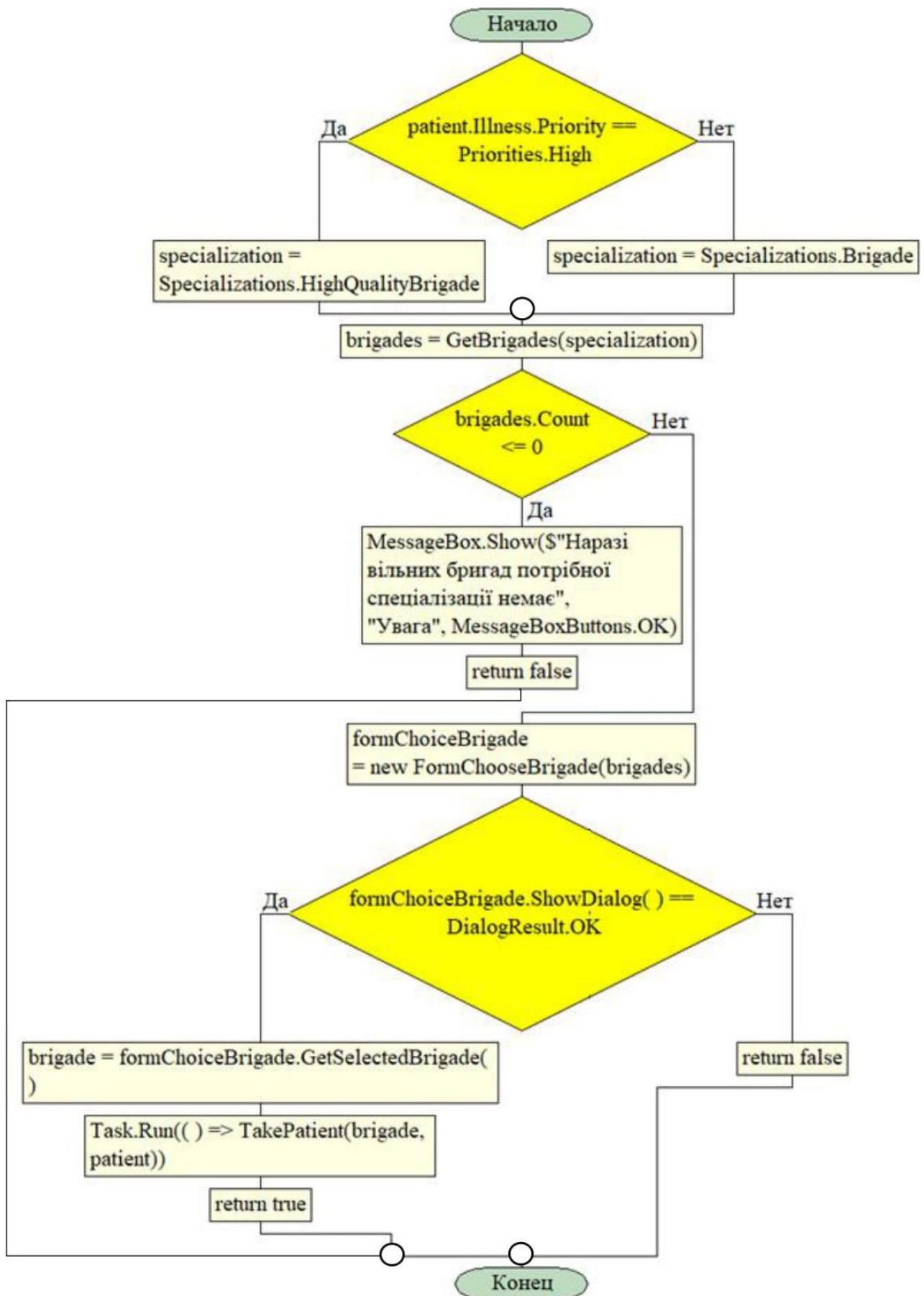


Рис. А.2. Блок-схема алгоритму призначення бригади медиків на виклик

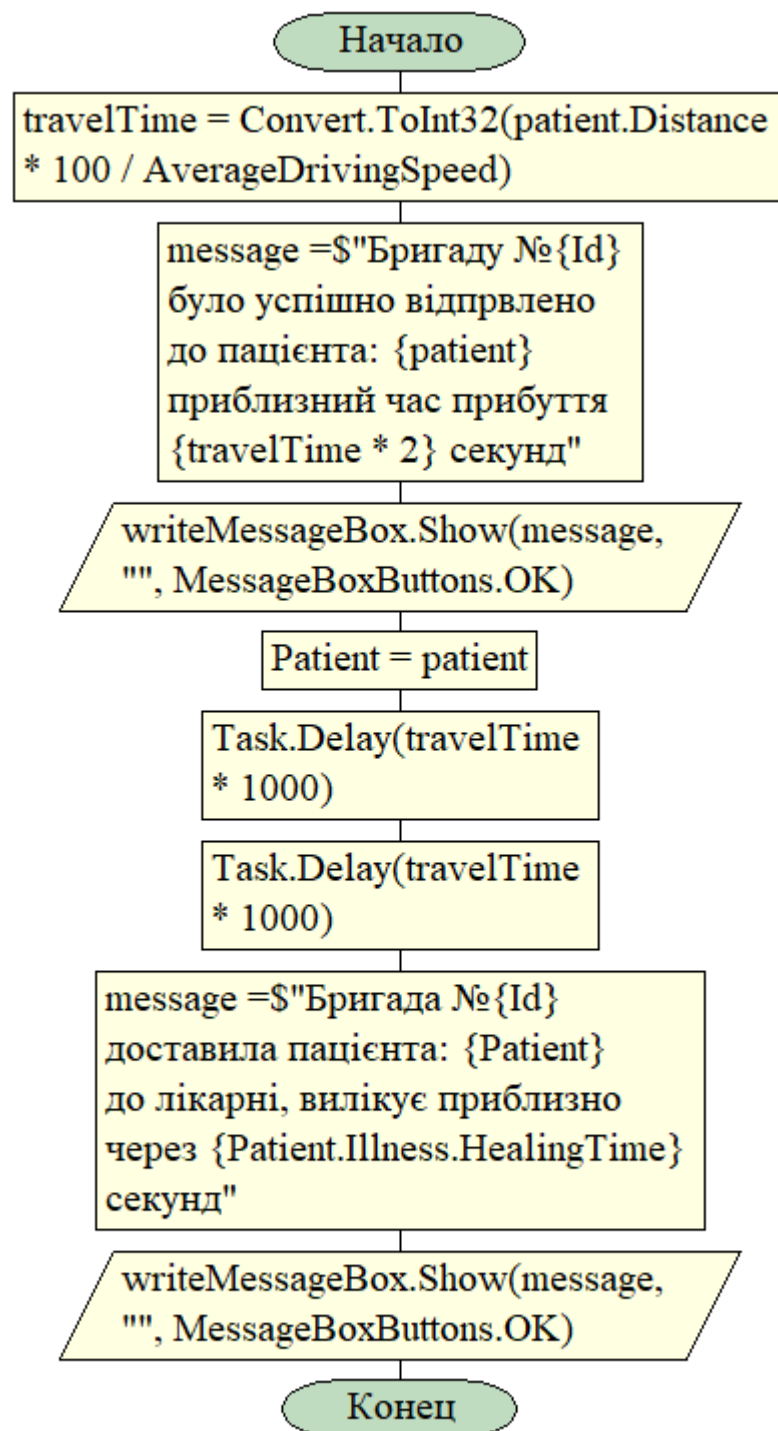


Рис. А.3. Блок-схема алгоритму емуляції поїздки бригади

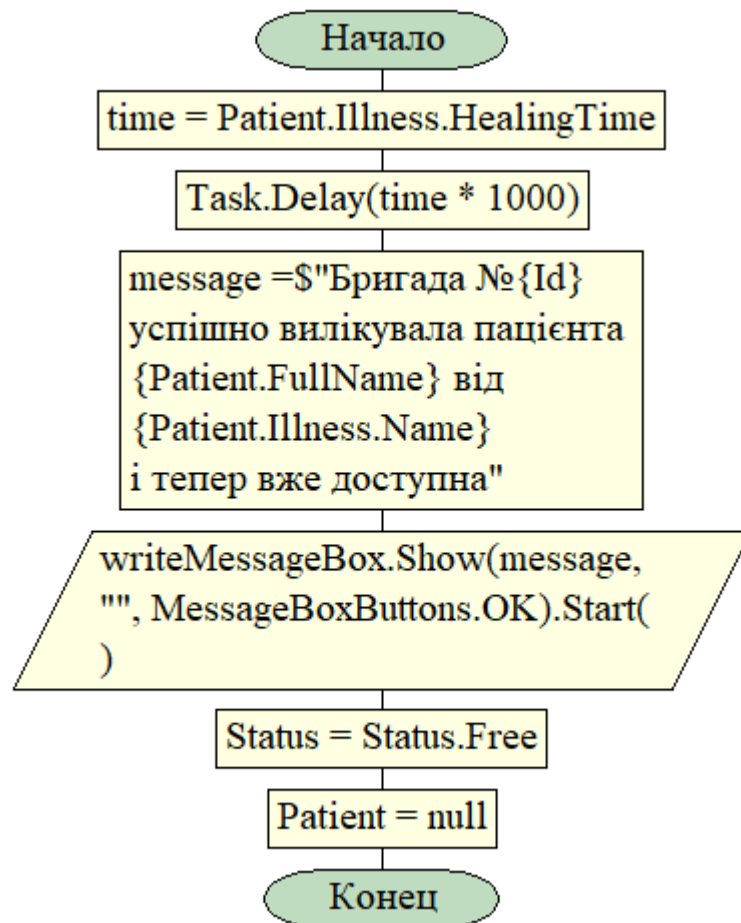


Рис. А.4. Блок-схема емуляції лікування пацієнта.

Додаток Б. Програмний код реалізації продукту

Лістинг 1. Клас BrigadeController

```
public class BrigadeController
{
    private readonly string BRIGADES_FILEPATH = "brigades.xml";
    //екземпляр файлового менеджера, створений для того щоб серіалізувати файли,
    //оновляти і тд
    //BindingList<Brigade> - це об'єкт, який буде серіалізуватись
    //якщо файла з вище згаданою назвою немає, то він його створить
    private readonly FileManager<BindingList<Brigade>> _fileManager;

    public BindingList<Brigade> Brigades { get; set; } //список бригад якими ми будем
    контролювати
    //делегат, який викликається, коли кількість бригад змінюється(для того, щоб
    FormMenu дізнавався про зміни і оновляв статистику)
    public Action<int, int> OnChangedCount;
    //делегат, який викликається, коли в бригади змінюється статус
    public Action OnChangedBrigadeStatus;

    public BrigadeController()
    {
        OnChangedCount = new Action<int, int>((int a, int b) => { });
        OnChangedBrigadeStatus = new Action(() => { });

        bool isNewFile = !File.Exists(BRIGADES_FILEPATH);

        _fileManager = new FileManager<BindingList<Brigade>>(BRIGADES_FILEPATH);
        Brigades = _fileManager.LoadObjectFromFile();
        var nonFreeBrigades = Brigades.Where(b => b.Status != Status.Free).ToArray();
        foreach (var brigade in nonFreeBrigades)
        {
            brigade.Patient = null;
            brigade.Status = Status.Free;
        }

        if (isNewFile)
            Add(BaseObjectsInitializer.GetDefaultBrigades().ToArray());
    }

    //логіка найму нової бригади
    public void HireBrigade()
    {
        int id;
        if (Brigades.Count > 0)
        {
            int lastNumber = Brigades.Count - 1;
            int lastId = Brigades[lastNumber].Id;
            id = ++lastId;
        }
        //якщо бригад нема, то ID нової бригади = 1
        else id = 1;
        //створюється нова бригада
        LowQualityBrigade brigade = new LowQualityBrigade(id);
        //одразу ж додаємо її, метод який добавить в список бригаду і оновить файл
        //за допомогою файлового менеджера
        Add(brigade);
    }
}
```

```

private void Add(params Brigade[] brigades)
{
    foreach (var brigade in brigades)
        Brigades.Add(brigade);
    _fileManager.UpdateFile(Brigades);
    OnChangedCount.Invoke(HighQualityBrigadeCount, AllBrigadesCount);
}

//видалення бригади
public void Remove(Brigade brigade)
{
    //видаляємо з списку і оновлюємо файл
    Brigades.Remove(brigade);
    _fileManager.UpdateFile(Brigades);
    //виклик делегату, який сповістить всіх "підписників" про зміни
    OnChangedCount.Invoke(HighQualityBrigadeCount, AllBrigadesCount);
}

//метод, який викликає форму вибору бригади для пацієнта
public bool ChooseBrigadeForPatient(Patient patient)
{
    //тут ми шукаємо всі бригади потрібної спеціалізації: для різних пріоритетів -
    різні бригади
    Specializations specialization;
    if (patient.Illness.Priority == Priorities.High)
        specialization = Specializations.HighQualityBrigade;
    else
        specialization = Specializations.Brigade;
    //тут за допомогою метода GetBrigades ми вибираємо бригади тільки підходящої
    спеціалізації
    BindingList<Brigade> brigades = GetBrigades(specialization);

    //якщо немає вільних бригад
    if (brigades.Count <= 0)
    {
        MessageBox.Show($"Наразі вільних бригад потрібної спеціалізації немає",
        "Увага", MessageBoxButtons.OK);
        return false;
    }
    //якщо бригади потрібної спеціалізації є, то викликається форма вибору бригади
    FormChooseBrigade formChoiceBrigade = new FormChooseBrigade(brigades);
    if (formChoiceBrigade.ShowDialog() == DialogResult.OK)//якщо ми вибрали і нажали
    OK
    {
        //в формі FormChooseBrigade є метод GetSelectedBrigade який вертає нам
        вибрану бригаду
        Brigade brigade = formChoiceBrigade.GetSelectedBrigade();
        //тут запускається метод TakePatient в новому потоці, по суті в ньому ми
        відправляємо бригаду
        Task.Run(() => TakePatient(brigade, patient));
        return true;
    }
    else
        return false;
}

//асинхронний метод, в якому ми відправляємо бригаду взяти пацієнта
public async void TakePatient(Brigade brigade, Patient patient)
{
    //змінюємо статус бригади і викликаємо делегат
    brigade.Status = Status.OnTheWay;
    OnChangedBrigadeStatus.Invoke();
}

```

```

        //запускається асинхронний метод бригади GoToPatientAsync
        //емуляція того, що бригада їде до пацієнта, і залежно від того, яка це бригада,
        або лікує на місці, або відвозить до лікарні
        await brigade.GoToPatientAsync(patient);

        //тоді коли бригада приїхала, змінюємо їй статус і викликаємо делегат
        brigade.Status = Status.HealsThePatient;
        OnChangedBrigadeStatus.Invoke();

        //HealAsync - метод, який починає лікувати пацієнта
        //лікуємо і оновлюємо файл, також викликаємо делегат
        await brigade.HealAsync();
        _fileManager.UpdateFile(Brigades);
        OnChangedBrigadeStatus.Invoke();
    }

    //логіка підвищення бригади
    public void RankUp(LowQualityBrigade brigade)
    {
        //створюємо новий об'єкт HighQualityBrigade на основі LowQualityBrigade
        HighQualityBrigade highQualityBrigade = new HighQualityBrigade()
        {
            Id = brigade.Id,
            Status = brigade.Status,
            Patient = brigade.Patient,
            RecoveredPatients = brigade.RecoveredPatients
        };
        //вставляємо його назад у список під тим індексом, під яким він був
        int index = Brigades.IndexOf(brigade);
        Brigades.Remove(brigade);
        Brigades.Insert(index, highQualityBrigade);

        //оновлюємо файл і викликаємо делегат
        _fileManager.UpdateFile(Brigades);
        OnChangedCount.Invoke(HighQualityBrigadeCount, AllBrigadesCount);
    }

    //в цьому методі ми можемо взяти бригаду потрібної спеціалізації
    public BindingList<Brigade> GetBrigades(Specializations specialization)
    {
        return new BindingList<Brigade>(Brigades.Where(b =>
            b.Specialization == specialization &&
            b.Status == Status.Free).ToList());
    }

    //автоматичні властивості для інформації, скільки бригад є
    //використовуються делегатом для оповіщення своїх підписників про кількість бригад
    різних спеціалізації
    private int HighQualityBrigadeCount { get => Brigades.Where(b => b.Specialization ==
        Specializations.HighQualityBrigade).Count(); }
    private int AllBrigadesCount { get => Brigades.Where(b => b.Specialization ==
        Specializations.Brigade).Count(); }
    public (string, string) GetStatistics()
    {
        return (HighQualityBrigadeCount.ToString(), AllBrigadesCount.ToString());
    }

```

Лістинг 2. Контролер PatientController

```
public class PatientController
{
    public Action<int> OnChangedCount; //делегат, який викликається при зміні кількості
    пацієнтів
    public BindingList<Patient> Patients { get; set; } //список пацієнтів
    public PatientController()
    {
        OnChangedCount = new Action<int>((int a) => { });

        Patients = new BindingList<Patient>()
        {
            //генеруються два рандомних об'єкта пацієнтів в початковий список
            PatientsGenerator.Generate(),
            PatientsGenerator.Generate()
        };
    }
    //метод, який дозволяє створити нового рандомного пацієнта і одразу додати його до
    списку всіх пацієнтів
    public void NewRandomPatient()
    {
        Patients.Add(PatientsGenerator.Generate());
        //звук
        Console.Beep();
        //виклик делегату
        OnChangedCount.Invoke(Patients.Count);
    }
}
```

Лістинг 3. Brigade

```
public abstract class Brigade
{
    public int Id { get; set; }
    public abstract Specializations Specialization { get; set; }
    public Status Status { get; set; }

    public Patient Patient { get; set; }
    //кількість вилікуваних пацієнтів
    public int RecoveredPatients { get; set; } = 0;

    //середня швидкість, абстрактна автоматична властивість, в конкретних класах бригад
    вона буде перевизначатись
    public abstract double AverageDrivingSpeed { get; set; }

    //абстрактні методи без тіла, будуть перевизначатись в конкретних об'єктах бригад
    public abstract Task HealAsync();
    public abstract Task GoToPatientAsync(Patient patient);

    public Brigade() {}
    //перегружений конструктор, приклад поліморфізму
    public Brigade(int id)
    {
        Id = id;
    }
}
```

Лістинг 4. HQBrigade

```
public override Specializations Specialization { get; set; } =
Specializations.HighQualityBrigade;
public override double AverageDrivingSpeed { get; set; } = 100;
```

```

public HighQualityBrigade() { }
public HighQualityBrigade(int id) : base(id) { }

//перевизначений асинхронний метод, в якому відбувається емуляція поїздки до
пацієнта
public async override Task GoToPatientAsync(Patient patient)
{
    //визначається час подорожі в одну сторону
    int travelTime = Convert.ToInt32(patient.Distance * 100 / AverageDrivingSpeed);

    //запускаємо MessageBox в інший потік, щоб іконка не ступорила нам програму
    string message = $"Бригаду №{Id} було успішно відправлено до пацієнта: {patient}
приблизний час прибуття {travelTime * 2} секунд";
    new Thread(() => MessageBox.Show(message, "", MessageBoxButtons.OK)).Start();

    Patient = patient;

    //емуляція подорожі до хворого
    await Task.Delay(travelTime * 1000);

    //емуляція подорожі назад
    await Task.Delay(travelTime * 1000);

    //запускаємо MessageBox в інший потік, щоб іконка не ступорила нам програму
    message = $"Бригада №{Id} доставила пацієнта: {Patient} до лікарні, вилікує
приблизно через {Patient.Illness.HealingTime} секунд";
    new Thread(() => MessageBox.Show(message, "", MessageBoxButtons.OK)).Start();
}

//метод емуляції лікування хворого
public async override Task HealAsync()
{
    //час лікування
    int time = Patient.Illness.HealingTime;

    //емуляція лікування
    await Task.Delay(time * 1000);
    //запускаємо MessageBox в інший потік, щоб іконка не ступорила нам програму
    string message = $"Бригада №{Id} успішно виликувала пацієнта {Patient.FullName}
від {Patient.Illness.Name} і тепер вже доступна";
    new Thread(() => MessageBox.Show(message, "", MessageBoxButtons.OK)).Start();

    //змінюємо статус
    Status = Status.Free;
    //звільняємо пацієнта
    Patient = null;
    //добавляємо +1 до виликаних пацієнтів
    ++RecoveredPatients;
}

```

Лістинг 5. LQBrigade

```

public override Specializations Specialization { get; set; } = Specializations.Brigade;
public override double AverageDrivingSpeed { get; set; } = 70;

//перегружений конструктор

public LowQualityBrigade() : base() { }
public LowQualityBrigade(int id) : base(id) { }

//перевизначений асинхронний метод, в якому відбувається емуляція поїздки до
пацієнта

```

```

public async override Task GoToPatientAsync(Patient patient)
{
    int travelTime = Convert.ToInt32(patient.Distance * 100 / AverageDrivingSpeed);
    int waitingTime = travelTime * 2 + patient.Illness.HealingTime;

    string message = $"Бригаду №{Id} було успішно відправлено до пацієнта: {patient}
приблизний час вилікування і прибуття {waitingTime} секунд";
    new Thread(() => MessageBox.Show(message, "", MessageBoxButtons.OK)).Start();

    //емуляція подорожі до хворого
    Patient = patient;
    await Task.Delay(travelTime * 1000);
}

//метод емуляції лікування хворого
public async override Task HealAsync()
{
    //час лікування
    int time = Patient.Illness.HealingTime;

    //емуляція лікування
    await Task.Delay(time * 1000);
    //запускаємо MessageBox в інший потік, щоб іконка не ступорила нам програму
    string message = $"Бригаду №{Id} успішно вилікувала пацієнта {Patient.FullName}
від {Patient.Illness.Name} і тепер вже доступна";
    new Thread(() => MessageBox.Show(message, "", MessageBoxButtons.OK)).Start();

    //змінюємо статус
    Status = Status.Free;
    //звільняємо пацієнта
    Patient = null;
    //добавляємо +1 до виликаних пацієнтів
    ++RecoveredPatients;
}

```

Лістинг 6. Patient

```

public class Patient
{
    public string CallTime { get; set; } //час виклику
    public string FullName { get; set; } //повне ім'я
    public double Distance { get; set; } //дистанція до пацієнта
    public Illness Illness { get; set; } //хвороба
    public Priorities Priority { get => Illness.Priority; } //приоритет пацієнта, беремо
з приорітету хвороби

    public Patient() //дата виклику буде братись з часу ініціалізації конструктора
    {
        CallTime = DateTime.Now.ToString("dd/MM H:mm:ss");
    }

    public override string ToString()
    {
        return $"{FullName}";
    }
}

```

Лістинг 7. PatientGenerator

```

public static class PatientsGenerator
{
    //часовий діапазон в якому буде появлятись новий пацієнт
    public static int MinValue { get; set; } = 15;
    public static int MaxValue { get; set; } = 30;
    //всі можливі хвороби
}

```

```

        private static readonly BindingList<Illness> _diseases =
BaseObjectsInitializer.DefaultDiseases();

        //статичний метод, який генерує нового пацієнта
        public static Patient Generate()
        {
            //беремо індекс останнього елементу з усіх імен
            int endIndexOfFirstNames = Enum.GetNames(typeof(FirstNames)).Length - 1;
            //беремо індекс останнього елементу з усіх прізвищ
            int endIndexOfLastNames = Enum.GetNames(typeof(FirstNames)).Length - 1;

            var firstNameGenerator = new Random();
            var lastNameGenerator = new Random();

            //генеруємо ім'я та прізвище
            string firstName = Enum.GetName(typeof(FirstNames), firstNameGenerator.Next(0,
endIndexOfFirstNames));
            string lastName = Enum.GetName(typeof(FirstNames), lastNameGenerator.Next(0,
endIndexOfLastNames));

            return new Patient()
            {
                FullName = $"{lastName} {firstName}",
                Illness = GenerateIllness(), //генеруєм пацієнту нову хворобу
                Distance = new Random().Next(5, 50), //генеруєм дистанцію до пацієнта
            };
        }

        //метод, який генерує хворобу
        private static Illness GenerateIllness()
        {
            Random random = new Random();
            int randomValue = random.Next(0, _diseases.Count);
            return _diseases[randomValue];
        }

        //генерує час з діапазону, визначає, коли з'явиться новий хворий
        public static int GetRandomTime()
        {
            return new Random().Next(0, MaxValue);
        }
    }

```

Лістинг 8. Illness

```

public class Illness
{
    public int HealingTime { get; set; } //потрібний час для того, щоб вилікувати
хворобу
    public string Name { get; set; } //назва хвороби
    public Priorities Priority { get; set; } //пріоритет хвороби

    public Illness() { } //конструктори
    public Illness(string name, Priorities priority, int healingTime)
    {
        Name = name;
        Priority = priority;
        HealingTime = healingTime;
    }

    public override string ToString()
    {
        return Name;
    }
}

```


Лістинг 9. BaseObjectsInitializer

```
public static class BaseObjectsInitializer
{
    private static BindingList<Illness> _diseases;
    private static BindingList<Brigade> _brigades;

    //повертає базовий список можливих хвороб
    public static BindingList<Illness> GetDefaultDiseases()
    {
        if (_diseases == null)
            return _diseases = new BindingList<Illness>()
            {
                new Illness(name: "Covid-19", Priorities.High, healingTime: 35),
                new Illness(name: "Перелом", Priorities.High, healingTime: 15),
                new Illness(name: "Грип", Priorities.High, healingTime: 40),

                new Illness(name: "Опік", Priorities.Low, healingTime: 10),
                new Illness(name: "Простуда", Priorities.Low, healingTime: 5),
            };
        else
            return _diseases;
    }

    //повертає базовий список бригад, вони будуть появлятися, якщо файл новий
    public static BindingList<Brigade> GetDefaultBrigades()
    {
        if (_brigades == null)
            return _brigades = new BindingList<Brigade>()
            {
                new HighQualityBrigade { Id = 1, Status = Status.Free },
                new HighQualityBrigade { Id = 2, Status = Status.Free },
                new HighQualityBrigade { Id = 3, Status = Status.Free },

                new LowQualityBrigade { Id = 4, Status = Status.Free },
                new LowQualityBrigade { Id = 5, Status = Status.Free },
                new LowQualityBrigade { Id = 6, Status = Status.Free }
            };
        else
            return _brigades;
    }
}
```

Лістинг 10. FileManager

```
public class FileManager<T> where T : new()
{
    public readonly string _filePath;

    //інкапсульовані поля, доступні тільки в межах цього класу
    private readonly XmlSerializer _serializer; //серіалізатор
    private StreamWriter _streamWriter;
    private StreamReader _streamReader;

    public FileManager(string path)
    {
        //перевіряємо вхідні дані

        if (string.IsNullOrEmpty(path))
            throw new ArgumentNullException("File path is empty or null", nameof(path));

        _filePath = path;
        _serializer = new XmlSerializer(typeof(T));
    }
}
```

```

//метод, який оновлює об'єкт у файлі
public virtual void UpdateFile(T fileObject)
{
    try
    {
        _streamWriter = new StreamWriter(_filePath);
        _serializer.Serialize(_streamWriter, fileObject);
        _streamWriter.Dispose();
    }
    catch (SerializationException ex)
    {
        throw new FileLoadException(ex.Message, _filePath);
    }
}

//метод, який загрузає об'єкт з файлу, якщо його не існує, то створює його
public virtual T LoadObjectFromFile()
{
    if (!File.Exists(_filePath))
    {
        T anyObject = new T();
        UpdateFile(anyObject);
        return anyObject;
    }
    try
    {
        _streamReader = new StreamReader(_filePath);
        T xmlObject = (T)_serializer.Deserialize(_streamReader);
        _streamReader.Dispose();
        return xmlObject;
    }
    catch (SerializationException ex)
    {
        throw new FileLoadException(ex.Message, _filePath);
    }
}
}

```

Лістинг 11. FormPatientList

```

public partial class PatientsList : Form
{
    private readonly BrigadeController _brigadeController;
    private readonly PatientController _patientController;

    //Cancellation token створені для того, щоб зупинити дію в іншому потоці
    private readonly CancellationTokenSource cancellationTokenSource;
    private readonly CancellationToken token;

    public PatientsList(BrigadeController brigadeController, PatientController
patientController)
    {
        //визначаємо CancellationTokenSource
        cancellationTokenSource = new CancellationTokenSource();
        token = cancellationTokenSource.Token;

        _brigadeController = brigadeController;
        _patientController = patientController;

        //якщо форму буде закрито, то ми зупиняємо потік нових клієнтів
        //можна забрати, тоді нові клієнти будуть появлятися тоді, коли форма
PatientsList закрыта
        FormClosed += StopReceive;
    }
}

```

```

InitializeComponent();

//визначаємо джерело даних для PatientsGridView
PatientsGridView.DataSource = _patientController.Patients;
//підписуємся на подію SelectionChanged, при зміні виділення буде викликатись
метод PatientsGridView_SelectionChanged
PatientsGridView.SelectionChanged += PatientsGridView_SelectionChanged;

//встановлюємо ширину для колонок
PatientsGridView.Columns["CallTime"].Width = 100;
PatientsGridView.Columns["FullName"].Width = 145;
PatientsGridView.Columns["Distance"].Width = 60;
PatientsGridView.Columns["Priority"].Width = 65;

//асинхронно (щоб не блокувало інші процеси) запускаємо метод, який стартує
генерацію нових пацієнтів
Task.Run(() => StartAsync());
}

public async Task StartAsync()
{
    while (true)
    {
        if (token.IsCancellationRequested)
            return;
        await Task.Delay(PatientsGenerator.GetRandomTime() * 1000);
        _patientController.NewRandomPatient();
    }
}

private void PatientsGridView_SelectionChanged(object sender, EventArgs e)
{
    //якщо не вибрано жодного пацієнта, то кнопка, яка дозволяє задати бригаду для
пацієнта, зникає
    if (_patientController.Patients.Count <= 0)
        ButtonSetGroup.Visible = false;
    else
        ButtonSetGroup.Visible = true;
}

//кнопка, яка задає бригаду для пацієнта
private void ButtonSetGroup_Click(object sender, EventArgs e)
{
    //обираємо виділену колонку
    var row = PatientsGridView.SelectedRows[0];
    //беремо з неї об'єкт пацієнта
    if(row.DataBoundItem is Patient patient)
    {
        //викликаємо з контроллера метод вибору бригади для пацієнта
        if (_brigadeController.ChooseBrigadeForPatient(patient) == true)
        {
            //якщо вибір бригади пройшов успішно, відбувається видалення пацієнта зі
списку
            _patientController.Patients.Remove(patient);
            //якщо більше немає пацієнтів, то відключаємо кнопку "Вибрати групу(для
пацієнта)"
            if (_patientController.Patients.Count <= 0)
                ButtonSetGroup.Visible = false;
        }
    }
}

//метод, що зупиняє потік зі StartAsync(); і створення нових пацієнтів зупиняється
public void StopReceive(object sender, EventArgs e) => cancellationTokenSource.Cancel();
}

```

Лістинг 12. FormGroupOfDoctors

```
public partial class FormGroupOfDoctors : Form
{
    private readonly BrigadeController _brigadeController;

    public FormGroupOfDoctors(BrigadeController brigadeController)
    {
        _brigadeController = brigadeController;
        //ініціалізуємо вікно потрібними властивостями
        InitializeComponent();
        CheckForIllegalCrossThreadCalls = false;
        //підписуєм на делегат SelectionChanged метод GridSelectionChanged
        //коли буде якесь виділення в формі, то буде викликаний метод
        GridSelectionChanged
        BrigadesGridView.SelectionChanged += GridSelectionChanged;

        //визначаємо джерело даних для BrigadesGridView, звідти будуть братись моделі
        для колонок і даних
        BrigadesGridView.DataSource = _brigadeController.Brigades;
        //коли буде змінюватись статус бригади, ми будемо оновляти саму таблицьку
        _brigadeController.OnChangedBrigadeStatus += UpdateGrid;
        UpdateGrid();
    }

    private void UpdateGrid()
    {
        //оновлюється
        BrigadesGridView.DataSource = null;
        BrigadesGridView.DataSource = _brigadeController.Brigades;

        if (BrigadesGridView.Columns["Id"] == null)
            return;

        //визначається розмір форм
        BrigadesGridView.Columns["Id"].Width = 40;
        BrigadesGridView.Columns["Status"].Width = 75;
        BrigadesGridView.Columns["Specialization"].Width = 140;
        BrigadesGridView.Columns["Patient"].Width = 172;
        BrigadesGridView.Columns["RecoveredPatients"].Width = 130;
        BrigadesGridView.Columns["AverageDrivingSpeed"].Visible = false;

        //якщо бригад немає, то пропонуємо користувачеві найняти нову бригаду
        if (_brigadeController.Brigades.Count <= 0)
        {
            ButtonRemove.Visible = false;
            ButtonRankUp.Visible = false;
            if (MessageBox.Show($"Бригад немає, бажаєте створити нову бригаду?",
                "Увага", MessageBoxButtons.YesNo) == DialogResult.Yes)
            {
                _brigadeController.HireBrigade();
                MessageBox.Show($"Нову бригаду створено", "Увага!",
                    MessageBoxButtons.OK);
            }
        }

        //якщо не вибрана жодна з бригад, то кнопки "Видалити бригаду" і "Підвищити бригаду"
        стануть недоступними
        private void GridSelectionChanged(object sender, EventArgs e)
        {
            if (BrigadesGridView.SelectedCells.Count > 0)
            {
                ButtonRemove.Visible = true;
                ButtonRankUp.Visible = true;
            }
        }
    }
}
```

```

    }
    if (_brigadeController.Brigades.Count <= 0)
    {
        ButtonRemove.Visible = false;
        ButtonRankUp.Visible = false;
    }
}

//метод видалення бригади
private void RemoveBrigade_Click(object sender, EventArgs e)
{
    var selectedBrigade = TryToGetSelectedBrigade();//метод, який обирає об'єкт
    виділеної/вибраної бригади

    if (MessageBox.Show($"Ви дійсно хочете розформувати бригаду
    №{selectedBrigade.Id}", "Видалення бригади", MessageBoxButtons.YesNo) == DialogResult.Yes)
    {
        //якщо так, то розформує бригаду
        _brigadeController.Remove(selectedBrigade);
        MessageBox.Show($"Бригаду №{ selectedBrigade.Id} розформовано", "Видалення
    бригади", MessageBoxButtons.OK);
    }
}

//метод, який звертається до контроллера і створює нову бригаду
private void ButtonCreateBrigade_Click(object sender, EventArgs e)
{
    _brigadeController.HireBrigade();
    MessageBox.Show($"Нову бригаду створено", "Увага!", MessageBoxButtons.OK);
}

//метод, який підвищує бригаду
private void ButtonRankUp_Click(object sender, EventArgs e)
{
    //беремо виділену бригаду
    var selectedBrigade = TryToGetSelectedBrigade();

    if (selectedBrigade.Specialization == Specializations.Brigade //якщо
    спеціалізація бригади звичайна
    && selectedBrigade.Status == Status.Free)// і якщо бригада зараз вільна
    {
        _brigadeController.RankUp((LowQualityBrigade)selectedBrigade);// то підвищує
        MessageBox.Show($"Бригада була підвищена", "", MessageBoxButtons.OK);
    }
    else if(selectedBrigade.Status != Status.Free) //якщо бригада зараз не вільна,
    то викидаємо помилку
    {
        MessageBox.Show($"Бригада зараз в дорозі", "Помилка!",
        MessageBoxButtons.OK);
    }
    else if(selectedBrigade.Specialization != Specializations.Brigade)//якщо бригада
    і так максимального рангу, то викидаємо помилку
    {
        MessageBox.Show($"Бригада максимального рангу", "Помилка!",
        MessageBoxButtons.OK);
    }
}

//інкапсульований метод, який повертає об'єкт виділеної бригади
private Brigade TryToGetSelectedBrigade()
{
    var row = BrigadesGridView.SelectedRows[0];
    if (row.DataBoundItem is Brigade brigade)

```

```

        return brigade;
    else throw new InvalidCastException($"{row.DataBoundItem.GetType()} isn't a
Brigade");
    }
}

```

Лістинг 13. FormChooseBrigade

```

public partial class FormChooseBrigade : Form
{
    //форма вибору бригади
    //приймає список доступних бригад
    public FormChooseBrigade(BindingList<Brigade> brigades)
    {
        InitializeComponent();

        //при зміні виділення викликає метод GridSelectionChanged, який перевіряє, чи
        //виділена зараз якась бригада
        BrigadesGridView.SelectionChanged += GridSelectionChanged;

        //визначає видимість і розміри колонок
        BrigadesGridView.DataSource = brigades;
        BrigadesGridView.Columns["Patient"].Visible = false;
        BrigadesGridView.Columns["AverageDrivingSpeed"].Visible = false;
        BrigadesGridView.Columns["RecoveredPatients"].Width = 150;
        BrigadesGridView.Columns["Specialization"].Width = 150;
    }

    //метод, який дозволяє взяти об'єкт виділеної бригади
    public Brigade GetSelectedBrigade()
    {
        var row = BrigadesGridView.SelectedRows[0];
        if (row.DataBoundItem is Brigade brigade)
            return brigade;
        else
            throw new InvalidCastException($"{row.DataBoundItem.GetType()} isn't a
Brigade");
    }

    private void GridSelectionChanged(object sender, EventArgs e)
    {
        if (BrigadesGridView.SelectedCells.Count > 0)
            ButtonChoice.Visible = true;
        else
            ButtonChoice.Visible = false;
    }
}

```