МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет Електроніки та інформаційних технологій
(повна назва інституту/факультету)
Кафедра нананоелектроніки та модифікації поверхні
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ О.Д.Погребняк
(підпис) (Ім'я та ПРІЗВИЩЕ)

12 червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр
(бакалавр / магістр)
зі спеціальності 153 «Мікро- та наносистемна техніка»,
(код та назва)
Освітньо-професійної програми «Нанотехнології та біомедичні системи»
(освітньо-професійної / освітньо-наукової) (назва програми)
на тему: «Розробка інформаційно – довідкової системи для автоматизації медичної установи»
Здобувача (ки) групи ФЕ-91 Будка Дмитрія Романовича
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Д.Р. Будко
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник доц. О.В. Ющенко _____
(посада, науковий ступінь, вчене звання Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2023

Сумський державний університет

(назва вузу)

Факультет Електроніки та інформаційних технологій Кафедра Наноелектроніки та модифікації поверхні

Спеціальність 153 – «Мікро та наносистемна техніка»

ЗАТВЕРДЖУЮ:

Зав. кафедрою Наноелектроніки та модифікації поверхні

О.Д. Погребняк

«20» березня 2023 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Будку Дмитрію Романовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) «Розробка інформаційно – довідкової системи для автоматизації медичної установи»

затверджена наказом по університету від «_____» _____2023р. №_____

2. Термін здачі студентом закінченого проекту (роботи) 19.06.2023р.

3. Вхідні дані до проекту (роботи) 1) Java Swing; 2) База даних; 3) Розробка системи;

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) 1) Мова Java; 2) Переваги використання; 3) розгляд синтаксису мови Java; 4) Можливості розробки серидовища програмування; 5) створення плану системи для хірургічного центру; 6) Технології для створення інформаційно-довідкової системи

5. Перелік графічного матеріалу (з точним зазначення обов'язкових креслень)

План додатку; Вікно завантаження додатку; Вікно логіну в додаток; Вкладка Patient; Вкладка Doctors; Вкладка Treatment; Вкладка Prescription; Вкладка Appointments; Вкладка Dashboard

6. Дата видачі завдання 20.03.2023 р.

Керівник _____

(підпис)

Завдання прийняв до виконання _____

(підпис)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Опанування синтаксису Java	04.04 - 20.04	виконано
2	Опанування принципами ООП	21.04 - 05.05	виконано
3	Опанування середовищем розробки Apache Netbeans	06.05 - 08.05	виконано
4	Опанування технологією та элементами Java Swing	09.05 - 13.05	виконано
5	Розробка графічного інтерфейсу	14.05 - 17.05	виконано
6	Розробка класів додатку	18.05 - 21.05	виконано
7	Розробка таблиць баз даних	22.05 - 23.05	виконано
8	Побудова плану додатку	24.05 - 25.05	виконано
9	Заповнення дипломного звіту	26.05 - 01.06	виконано

Студент-дипломник _____

(підпис)

Керівник проекту _____

(підпис)

АНОТАЦІЯ

Робота містить вступ, дослідження матеріалу, в якому розглянуто, що таке мова Java, історія створення її переваги та недоліки, обгрунтовано, чому обрано саме середовище програмування Apache NetBeans, технології для розробки, а також створення інформаційно – довідкової системи медичної установи, висновків та списку використаних джерел.

Звіт містить 53 сторінки, 64 рисунка та 17 літературних джерел.

Об’єктом дослідження є довідкова система хірургічного центру.

Мета роботи – опанування набором технологій для створення інформаційно-довідкової системи хірургічного центру.

КЛЮЧОВІ СЛОВА: JAVA, APACHE NETBEANS, JAVA SWING, ХІРУРГІЧНИЙ ЦЕНТР, БАЗА ДАНИХ, DERBY DATABASE.

ЗМІСТ

ВСТУП

Сучасна сфера медицини є дуже технологічною галуззю, що використовує комп'ютерні технології для надання своєчасних та ефективних медичних послуг. Автоматизація елементів систем медичних установ є головним аспектом успіху у сфері медицини. Автоматизація медичних закладів включає в себе використання комп’ютерних систем для ефективнішої та точнішої діагностики, лікування пацієнтів та розподілу медичних ресурсів. Основна задача такої системи — це збирати, обробляти та зберігати медичну інформацію, щоб медичні установи ефективніше керували своїми ресурсами. Інформаційно-довідкові системи забезпечують легкий та швидкий доступ до інформації, що зберігається в медичних закладах, покращуючи комунікацію між лікарями та забезпечуючи швидкі та точні медичні візити.

Тому розробка автоматизованих інформаційно-довідкових систем для медичних установ з використанням мови програмування Java є дуже актуальною та перспективною темою. Java — об’єктно-орієнтована мова програмування, що має широкі можливості розробки програмного забезпечення у багатьох сферах нашого життя. Вона забезпечує зручну та ефективну розробку програмного забезпечення, також має високий ступінь безпеки та відмовостійкість.

Використання мови Java для розробки інформаційно-довідкової системи медичних закладів має купу переваг: зручний та швидкий доступ до медичної інформації, оптимізація робочих процесів та підвищення якості медичного обслуговування. Крім того, Java дозволяє розробникам легко створювати багатофункціональні системи, які можна розширювати та підтримувати.

РОЗДІЛ 1

МОВА ПРОГРАМУВАННЯ JAVA

1.1. Історія створення мови Java

Мова програмування Java була створена Джеймсом Артуром Гослінгом в компанії Sun Microsystems (зараз Oracle) в 1995 році. Головна мета створення мови була розробка кросплатформного програмного забезпечення, тобто програми, які можуть працювати на будь-якій операційній системі без змін вихідного коду.

Головні особливості мови: має високий рівень абстракції, що дозволяє розробникам працювати на вищому рівні абстракції, не займаючись деталями роботи з пам'яттю. Мова має автоматичне управління пам'яттю, що означає, що розробник не має контролювати життєвий цикл об'єктів. Java є повністю об'єктно-орієнтованою мовою програмування, що означає, що все в ній є об'єктом. Мова Java є безпечною мовою програмування, що означає, що вона має вбудовану систему захисту від небезпечних операцій, таких як витік пам'яті або збій програми.

1.2. Переваги використання мови Java

Java має широке застосування у багатьох сферах, таких як розробка веб-додатків, мобільних додатків, розробка ігор та інші. Один з найбільших плюсів використання мови Java полягає в тому, що вона є кросплатформною. Крім того, вона має велику кількість бібліотек та фреймворків, що значно полегшує розробку програмного забезпечення.

1.3. Синтаксис мови програмування Java

Синтаксис класів: у мові Java все є об'єктом. Для створення класу використовується ключове слово "class", яке потім слідує ім'я класу

```
public class NewClass {
    public static void main(String[] args) {
        System.out.println("Hello world!");

    }

}
```

```
compile-single:
run-single:
Hello world!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Рис.1.1 —Код з реалізацією прикладу об'єкту класу

У цьому прикладі ми створили класс "NewClass" і вивели текст на екран. Об'єкти створюються шляхом використання конструктора класу.

Ключове слово "new": використовується для створення нового об'єкта.

```
public class Basket {
    private int quantityOfGoods;

    public Basket(int quantityOfGoods) {
        this.quantityOfGoods = quantityOfGoods;
    }

    public int getQuantityOfGoods() {
        return quantityOfGoods;
    }

    public void setQuantityOfGoods(int quantityOfGoods) {
        this.quantityOfGoods = quantityOfGoods;
    }

}
public class Main {
    public static void main(String[] args) {

        Basket basket = new Basket(5);
        System.out.println("Кількість товарів у кошику: " + basket.getQuantityOfGoods());
    }
}
```

```
Кількість товарів у кошику: 5
BUILD SUCCESSFUL (total time: 0 seconds)
```

Рис.1.2 — Код з реалізацією прикладу створення нового об'єкта

У цьому прикладі ми створили об'єкт класу "Basket" за допомогою конструктора з параметром "quantityOfGoods", який повідомляє кількість товарів у

кошику. Після створення об'єкту ми використали метод "getQuantityOfGoods()" для отримання значення кількості товарів і вивели його на екран.

Змінні та типи даних: щоб оголосити змінну, потрібно використати ключове слово "int"(цілі числа), "double"(дійсні числа), "char"(символи) або "String"(рядки), а потім вказати ім'я змінної.

```
public class variable {
    public static void main(String[] args) {
        int x = 10; // оголошення цілочисельної змінної x і присвоєння їй значення 10
        double y = 3.14; // оголошення дійсної змінної y і присвоєння їй значення 3.14
        char symbol = 'a'; // оголошення символьної змінної symbol і присвоєння їй значення 'a'
        String name = "John"; // оголошення рядкової змінної name і присвоєння їй значення "John"

        System.out.println("Значення змінних: ");
        System.out.println("x = " + x);
        System.out.println("y = " + y);
        System.out.println("symbol = " + symbol);
        System.out.println("name = " + name);
    }
}
Значення змінних:
x = 10
y = 3.14
symbol = a
name = John
BUILD SUCCESSFUL (total time: 1 second)
```

Рис.1.3 — Код з реалізацією прикладу створення різних типів даних

У цьому прикладі ми оголосили змінні з різними типами даних, а саме: ціле число "x", дійсне число "y", символ "symbol" та рядок "name". Після оголошення змінних ми присвоїли їм певні значення і вивели їх на екран за допомогою методу "println()" з класу "System".

Умовні конструкції та цикли: мова програмування Java має умовні конструкції, такі як "if-else", "switch-case", які дозволяють виконувати певні дії залежно від певних умов. Також в мові присутні цикли, такі як "for", "while", "do-while", що дозволяють виконувати дії певну кількість разів.

```
public class Cycle {
    public static void main(String[] args) {
        int number = 5;
        // Умовна конструкція if-else
        if (number < 0) {
            System.out.println("Число від'ємне");
        } else if (number == 0) {
            System.out.println("Число дорівнює нулю");
        } else {
            System.out.println("Число додатнє");
        }
```

Рис.1.4 — Код з реалізацією прикладу створення конструкції "if-else"

```
// Умовна конструкція switch-case
switch (number) {
    case 0:
        System.out.println("Число дорівнює нулю");
        break;
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        System.out.println("Число менше або дорівнює 5");
        break;
    default:
        System.out.println("Число більше 5");
        break;
}
```

Рис.1.5 — Код з реалізацією прикладу створення конструкції "switch-case"

```
        // Цикл for
        System.out.println("Цикл for:");
        for (int i = 0; i < 5; i++) {
            System.out.println("i = " + i);
        }
```

Рис.1.6 — Код з реалізацією прикладу створення циклу "for"

```
// Цикл while
System.out.println("Цикл while:");
int j = 0;
while (j < 5) {
    System.out.println("j = " + j);
    j++;
}
```

Рис.1.7 — Код з реалізацією прикладу створення циклу "while"

```
// Цикл do-while
System.out.println("Цикл do-while:");
int k = 0;
do {
    System.out.println("k = " + k);
    k++;
} while (k < 5);
```

Рис.1.8 — Код з реалізацією прикладу створення циклу "do-while"

У цій програмі ми використали умовні конструкції "if-else" та "switch-case" для визначення певних дій залежно від значення змінної "number". Також ми використали цикли "for", "while" та "do-while" для виведення певної кількості повідомлень на екран:

```
Число додатне
Число менше або дорівнює 5
Цикл for:
i = 0
i = 1
i = 2
i = 3
i = 4
Цикл while:
j = 0
j = 1
j = 2
j = 3
j = 4
Цикл do-while:
k = 0
k = 1
k = 2
k = 3
k = 4
```

Рис.1.9 — Вивід тексту з програми

Масиви: для створення масиву потрібно вказати тип даних, а потім вказати кількість елементів масиву. Доступ до елементів масиву здійснюється за допомогою індексів.

```
public class Array {
    public static void main(String[] args) {
        // Оголошення та ініціалізація масиву
        int[] numbers = {1, 2, 3, 4, 5};

        // Доступ до елементів масиву за допомогою індексів
        System.out.println("Елемент з індексом 0: " + numbers[0]);
        System.out.println("Елемент з індексом 2: " + numbers[2]);

        // Зміна значень елементів масиву
        numbers[0] = 10;
        numbers[2] = 30;

        // Виведення всіх елементів масиву за допомогою циклу for
        System.out.println("Елементи масиву:");
        for (int i = 0; i < numbers.length; i++) {
            System.out.println(numbers[i]);
        }

        // Оголошення та ініціалізація двовимірного масиву
        int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

        // Доступ до елементів двовимірного масиву за допомогою індексів
        System.out.println("Елемент з індексами [1][1]: " + matrix[1][1]);

        // Зміна значень елементів двовимірного масиву
        matrix[0][2] = 30;
        matrix[2][0] = 70;

        // Виведення всіх елементів двовимірного масиву за допомогою вкладених циклів for
        System.out.println("Елементи двовимірного масиву:");
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

```
Елемент з індексом 0: 1
Елемент з індексом 2: 3
Елементи масиву:
10
2
30
4
5
Елемент з індексами [1][1]: 5
Елементи двовимірного масиву:
1 2 30
4 5 6
70 8 9
```

Рис.1.10 — Код з реалізацією прикладу масиву та виводом з нього інформації

У цій програмі ми створили масив "numbers" та двовимірний масив "matrix". Ми звернулися до елементів масивів за допомогою індексів, змінили значення деяких елементів та вивели всі елементи масивів за допомогою циклів for.

1.4. Об'єктно-орієнтоване програмування мовою Java

Об'єктно-орієнтоване програмування (ООП) є підхідом до програмування, в якому програма створюється у вигляді набору взаємодіючих об'єктів, кожен з яких має свої властивості і методи. Java - це мова програмування, що підтримує ООП і заснована на класах та об'єктах.

У Java об'єкти створюються за допомогою класів. Клас - це шаблон, що визначає властивості та методи, які має об'єкт. Об'єкти можуть взаємодіяти один з одним, викликати методи і передавати інформацію один одному.

Один з основних принципів ООП - це спадкування. В Java клас може успадковувати властивості та методи з іншого класу. Це дозволяє розширювати функціональність і зменшувати дублювання коду.

```
public class Dog extends Animal {
 private String breed;

 public Dog(String name, int age, String breed) {
     super(name, age);
     this.breed = breed;
 }

 public void bark() {
     System.out.println("The dog is barking.");
 }
```

Рис.1.11 — Код з реалізацією. Клас "Dog", що наслідує клас "Animal"

```
public class Animal {
    protected String name;
    protected int age;

    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void eat() {
        System.out.println("The animal is eating.");
    }
}
```

```
The animal is eating.
The dog is barking.
Name: Max
Age: 3
Breed: Labrador
```

Рис.1.12 — Клас " Animal " та вивід результату

У цьому прикладі клас Dog успадковує властивості та метод eat() класу Animal. До того ж, клас Dog має свою власну властивість breed та метод bark(). Клас Dog може використовувати метод eat(), який успадковується від класу Animal, а також свій власний метод bark().

Інший принцип ООП - це інкапсуляція, це означає, що властивості та методи класу повинні бути приховані від користувача. Це забезпечує безпеку та захист даних.

```
public class Main {
    public static void main(String[] args) {
BankAccount account1 = new BankAccount("123456789", "John Doe");
System.out.println("Account number: " + account1.getAccountNumber());
System.out.println("Owner name: " + account1.getOwnerName());
System.out.println("Balance: " + account1.getBalance());

account1.deposit(1000.0);
System.out.println("New balance after deposit: " + account1.getBalance());

account1.withdraw(500.0);
System.out.println("New balance after withdrawal: " + account1.getBalance());

account1.withdraw(700.0);
System.out.println("New balance after attempted withdrawal: " + account1.getBalance());
}
}
```

Рис.1.13 — Клас "Main ", що виводить результат

```java
public class BankAccount {
    private double balance;
    private String accountNumber;
    private String ownerName;

    public BankAccount(String accountNumber, String ownerName) {
        this.accountNumber = accountNumber;
        this.ownerName = ownerName;
        this.balance = 0.0;
    }

    public void deposit(double amount) {
        this.balance += amount;
    }

    public void withdraw(double amount) {
        if (amount > this.balance) {
            System.out.println("Insufficient funds.");
        } else {
            this.balance -= amount;
        }
    }

    public double getBalance() {
        return this.balance;
    }

    public String getAccountNumber() {
        return this.accountNumber;
    }

    public String getOwnerName() {
        return this.ownerName;
    }
}
```

```
Account number: 123456789
Owner name: John Doe
Balance: 0.0
New balance after deposit: 1000.0
New balance after withdrawal: 500.0
Insufficient funds.
New balance after attempted withdrawal: 500.0
```

Рис.1.14 — Клас "BankAccount " та виведений результат

У цьому прикладі властивості "balance", "accountNumber" та "ownerName" є приватними, вони недоступні ззовні класу. Замість цього, клас має публічні методи, такі як "deposit()", "withdraw()", "getBalance()", "getAccountNumber()" та "getOwnerName()", які дозволяють користувачеві взаємодіяти з об'єктом через інтерфейс класу. Таким чином, дані про баланс рахунку, номер рахунку, ім'я власника захищені від прямого доступу користувачів та цим забезпечуються безпека даних.

Ще один принцип ООП - це поліморфізм, він дозволяє об'єкту вести себе по-різному в залежності від контексту використання. Наприклад, у Java метод може бути перевизначений у підкласі, що дозволяє підкласу змінити поведінку методу.

```
public class Main {
public static void main(String[] args) {
Animal animal = new Animal();
Cat cat = new Cat();

animal.makeSound(); // виведе "The animal makes a sound."
cat.makeSound(); // виведе "The cat meows."
}
}
```

```
public class Animal {
    public void makeSound() {
        System.out.println("The animal makes a sound.");
    }
}
```

```
public class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("The cat meows.");
    }
}
```

Рис.1.15 — Код з реалізацією прикладу поліморфізму

У цьому прикладі клас "Cat" успадковує метод "makeSound()" від класу Animal і перевизначає його, щоб вивести рядок "The cat meows." замість "The animal makes a sound.". Цей приклад показує, що метод може мати різну реалізацію в різних класах, які успадковують його.

ООП в Java також підтримує абстракцію та інтерфейси. Абстракція дозволяє приховати деталі реалізації від користувача, щоб спростити роботу з кодом. Інтерфейси, у свою чергу, дозволяють визначити набір методів, які має реалізувати клас, не вказуючи деталі реалізації.

```java
class Circle extends Shape {
    private double radius;

    public Circle(String name, double radius) {
        super(name);
        this.radius = radius;
    }

    @Override // Перевизначення абстрактного методу з Shape
    public double getArea() {
        return Math.PI * Math.pow(radius, 2);
    }
}

class Square extends Shape {
    private double sideLength;

    public Square(String name, double sideLength) {
        super(name);
        this.sideLength = sideLength;
    }

    @Override // Перевизначення абстрактного методу з Shape
    public double getArea() {
        return Math.pow(sideLength, 2);
    }
}

abstract class Shape {
    protected String name;

    public Shape(String name) {
        this.name = name;
    }

    public abstract double getArea(); // Абстрактний метод, який має бути перевизначений в підкласах
}

public class Painter implements Paintable {
    @Override // Реалізація методу з інтерфейсу Paintable
    public void paint(String color) {
        System.out.println("Painting shape with color: " + color);
    }
}

 interface Paintable {
 public void paint(String color);
}

public class Main {
    public static void main(String[] args) {
        Shape circle = new Circle("Circle 1", 5.0);
        Shape square = new Square("Square 1", 4.0);

        System.out.println(circle.name + " has an area of " + circle.getArea());
        System.out.println(square.name + " has an area of " + square.getArea());

        Paintable painter = new Painter();
        painter.paint("blue");
    }
}
```

```
Circle 1 has an area of 78.53981633974483
Square 1 has an area of 16.0
Painting shape with color: blue
```

Рис.1.16 — Код з реалізацією прикладу реалізації абстракції та інтерфейсів

Приклад містить абстрактний клас "Animal" та конкретний підклас "Dog", який наслідується від "Animal". Також є інтерфейс "BankAccount", який містить методи для роботи з банківським рахунком та клас SavingsAccount, який реалізує цей інтерфейс. У головному класі "Main" створюються об'єкти типу "Dog" та "SavingsAccount" та викликаються їх методи.

Загалом, ООП в Java є досить потужним інструментом для розробки програмного забезпечення. Він дозволяє створювати складні системи, які легко розширюються та підтримуються. При цьому, добре структурований код, що використовує ООП підхід, є більш зрозумілим та зручним для розуміння та підтримки розробником.

РОЗДІЛ 2

СЕРЕДОВИЩЕ ПРОГРАМУВАННЯ APACHE NETBEANS IDE

Apache NetBeans IDE - це інтегроване середовище програмування (IDE) для розробки програм мовами програмування, такі як Java, C, C++, PHP, HTML та інші. Він був створений на базі NetBeans IDE, який розроблявся компанією Oracle, але був переданий Apache Software Foundation і став відкритим програмним забезпеченням під ліцензією Apache License 2.0.

2.1. Основні можливості Apache NetBeans IDE

Редагування коду. В середовищі програмування є потужний редактор з можливістю автодоповнення коду, підсвічування синтаксису, відступів та інших корисних функцій. Крім того, NetBeans IDE підтримує розширення для редагування коду різними мовами програмування.

Відлагодження коду. NetBeans IDE дозволяє відлагоджувати код різними мовами програмування, зокрема на Java, C і C++. Інструменти відлагодження дозволяють крок за кроком виконувати код, перевіряти значення змінних та знаходити помилки.

Управління проектами. NetBeans IDE дозволяє створювати, відкривати та управляти проектами. Він підтримує проекти на різних мовах програмування та дозволяє використовувати різні системи контролю версій, такі як Git, SVN, Mercurial та інші.

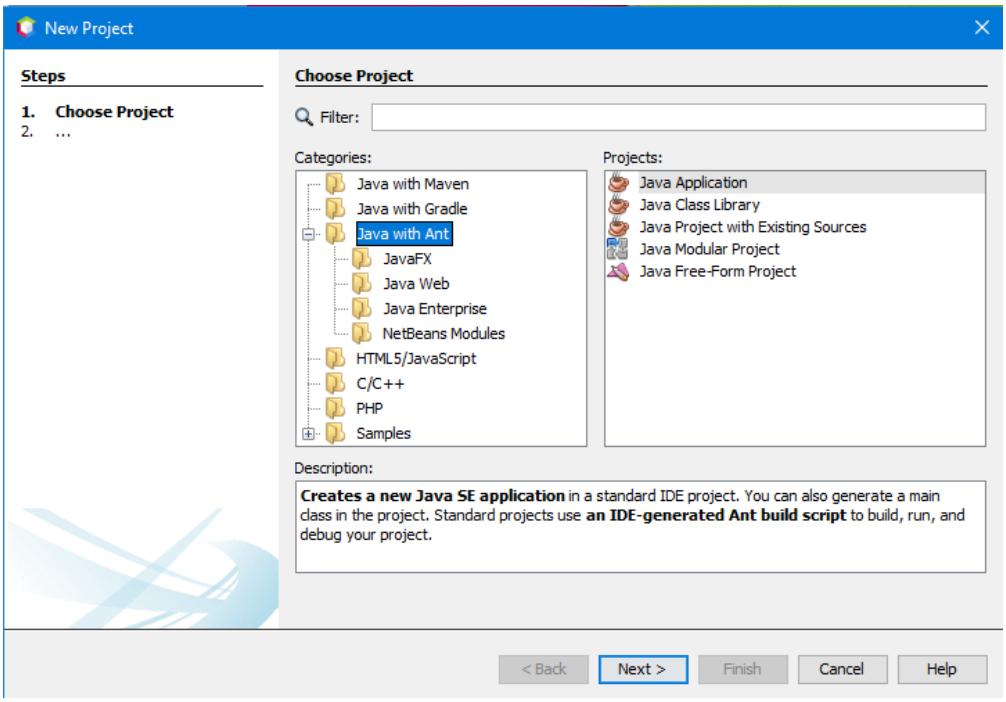

Рис.2.1 — Меню управління проектами

Графічна розробка інтерфейсів. NetBeans IDE має вбудовані інструменти для графічної розробки інтерфейсів користувача. Ці інструменти дозволяють створювати візуальні інтерфейси за допомогою перетягування елементів та налаштування їх властивостей.

Підтримка різних фреймворків та бібліотек. NetBeans IDE підтримує різні фреймворки та бібліотеки, що значно спрощує розробку програм. Наприклад, до них належать: JavaFX, Spring, Hibernate, Struts та інші. NetBeans IDE дозволяє швидко створювати проекти, які використовують ці фреймворки та надає потужні інструменти для роботи з ним.

Підтримка Maven. Maven - це інструмент для управління проектами в Java, який дозволяє автоматизувати процес збірки, тестування та розгортання програмного забезпечення. NetBeans IDE має вбудовану підтримку Maven, що дозволяє створювати та управляти проектами за допомогою цього інструменту.

Підтримка Git. Git - це система контролю версій, яка дозволяє зберігати та керувати кодом програми в репозиторії. NetBeans IDE має вбудовану підтримку Git, що дозволяє легко створювати, клонувати та синхронізувати репозиторії

розробки.

Підтримка віртуальних машин Java. NetBeans IDE підтримує віртуальні машини Java, такі як JDK, JRE та JVM. Він дозволяє швидко налаштовувати та переключатись між різними віртуальними машинами, що дозволяє розробляти та тестувати програми на різних версіях Java.

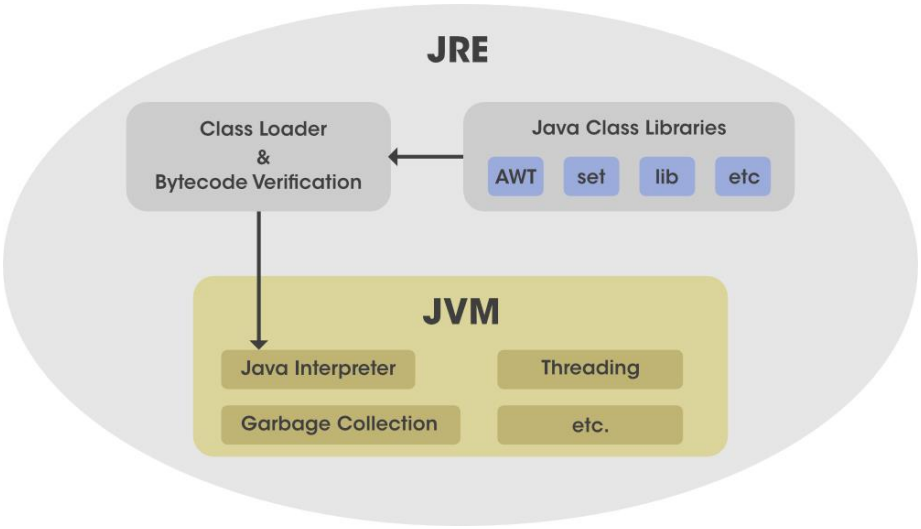

Рис.2.2 — Компоненти віртуальної машини[1]

Підтримка різних платформ. NetBeans IDE підтримує розробку програм для різних платформ: Windows, Linux та Mac OS. Він дозволяє налаштовувати та відлагоджувати код на різних платформах, що дозволяє створювати крос-платформні програми.

В цілому, Apache NetBeans IDE є зручним інструментом для розробки додатків на різних мовах програмування. Він має багато корисних функцій та можливостей, що дозволяють ефективно працювати з проектами будь-якої складності та розміру. Його інтерфейс є зрозумілим, що дозволяє швидко зорієнтуватись у програмі та почати роботу з нею. NetBeans IDE є безкоштовним та має відкритий вихідний код, що дозволяє користувачам вносити свої власні зміни та вдосконалення в програму.

Крім того, Apache NetBeans IDE підтримує відладку програм з використанням різних інструментів, таких як GDB, LLDB та JPDA. Він також має

вбудовану підтримку юніт-тестування, що дозволяє автоматизувати процес тестування програм.

Apache NetBeans IDE має розвинуту систему плагінів, яка дозволяє користувачам розширювати функціональність програми, додавати власні інструменти та функції. Це дозволяє створювати персоналізовані робочі середовища для конкретних задач та потреб користувачів.

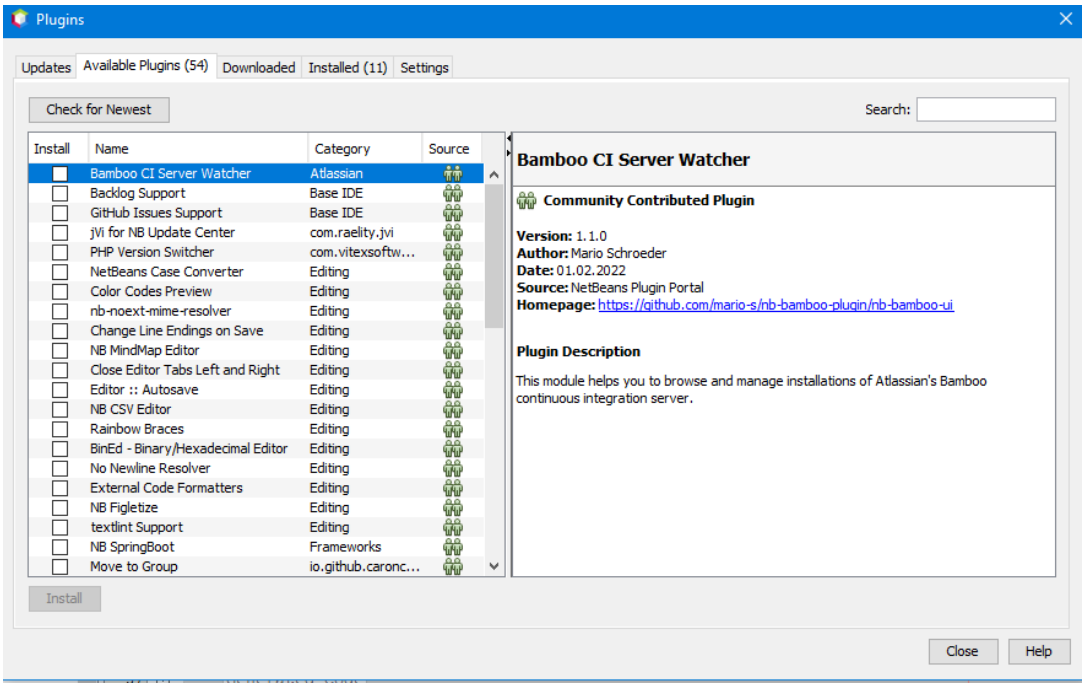

Рис.2.3— Система плагінів

РОЗДІЛ 3

РОЗРОБКА ІНФОРМАЦІЙНО - ДОВІДКОВОЇ СИСТЕМИ

Роздивимося декілька технологій, які знадобляться при створенні інформаційно - довідкової системи.

3.1. Java Swing

Java Swing - це зручний інструмент з графічним інтерфейсом, який має широкий спектр віджетів для створення оптимізованих віконних додатків. Це частина JFC (Java Foundation Classes), який базується на API AWT та повністю написаний на Java.

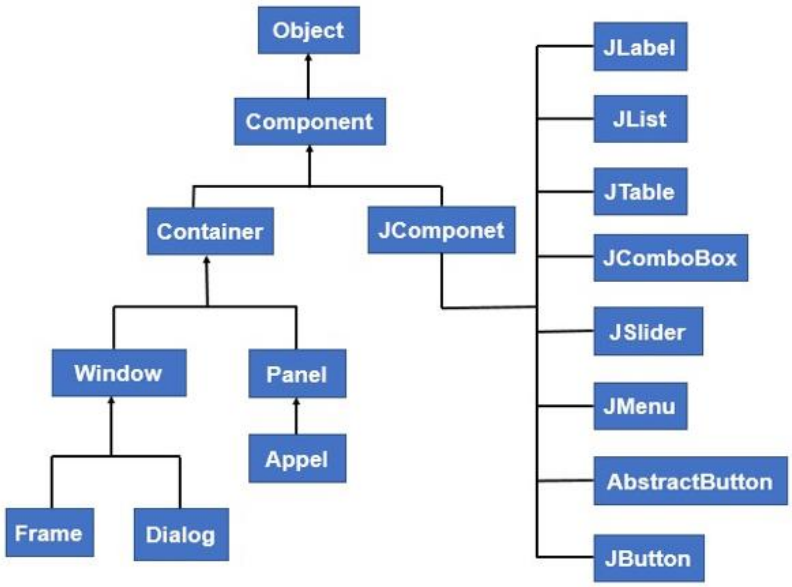

Рис.3.1 — Ієрархія технології Java Swing[2]

Всі компоненти у Swing, такі як JButton, JComboBox, JList, JLabel, успадковані від класу JComponent, який можна додати до класів контейнера.

Контейнери - це вікна, такі як рамка та діалогові вікна. Основні компоненти є будівельними блоками будь-якого графічного додатку. Такі методи, як setLayout, дозволяють перевизначити макет за замовчуванням у кожному контейнері. Контейнери, такі як JFrame та JDialog, можуть додавати компоненти тільки до себе.

Нижче наведено декілька компонентів з прикладами, щоб зрозуміти, як ми можемо їх використовувати.

JButton Class

Цей елемент інтерфейсу використовується для створення кнопки з маркером. При натисканні кнопки спрацьовує ActionListener і виконує певні дії. Він успадковує клас AbstractButton і не залежить від платформи.

```
import javax.swing.*;
import java.awt.*;

public class jButton {

    public static void main(String[] args) {
        JFrame frame = new JFrame("Example");
        JButton button = new JButton("Click me");
        button.setBackground(Color.GRAY);
        button.setForeground(Color.WHITE);
        button.setPreferredSize(new Dimension(120, 40));
        JPanel panel = new JPanel(new FlowLayout(FlowLayout.CENTER, 150, 150));
        panel.add(button);
        frame.setContentPane(panel);
        frame.pack();
        frame.setVisible(true);
    }
}
```

Рис.3.2 — Приклад реалізації елементу JButton

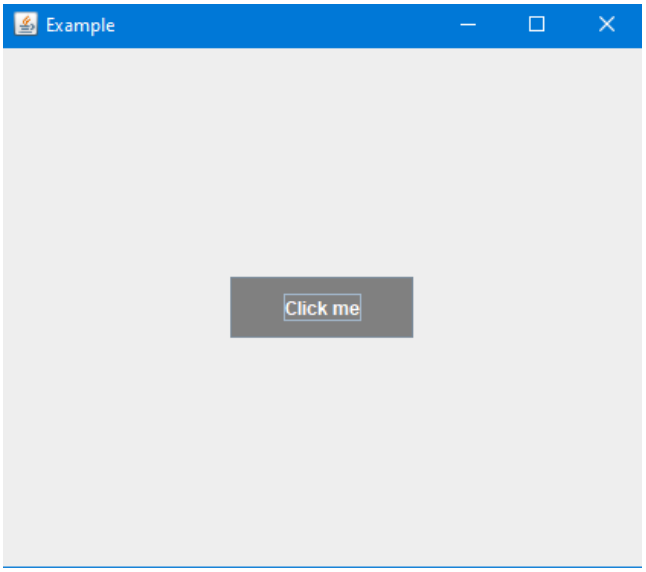

Рис.3.3 — Створений елемент JButton

JTextField Class

Цей клас успадковує властивості класу JTextComponent і використовується для редагування однорядкового тексту.

```
import javax.swing.*;

public class jTextField {
    public static void main(String args[]) {
        JFrame a = new JFrame("example");
        JTextField b = new JTextField("Enter Text");
        b.setBounds(50,100,200,30);
        a.add(b);
        a.setSize(300,300);
        a.setLayout(null);
        a.setVisible(true);

    }

}
```

Рис.3.4 — Приклад реалізації елементу JTextField

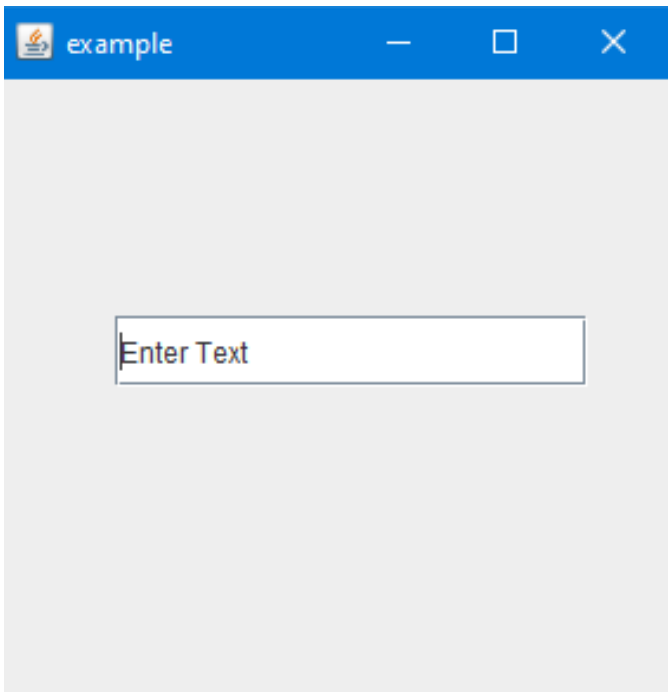

Рис.3.5 — Створений елемент JTextField

JPanel Class

Цей клас унаслідований від JComponent і забезпечує місце для додавання будь-якого іншого компонента у додатку.

```
import java.awt.*;
import javax.swing.*;

public class jPanel {
    jPanel(){
       JFrame a = new JFrame("example");
       JPanel p = new JPanel();
       p.setBounds(40,70,200,200);
       JButton b = new JButton("click me");
       b.setBounds(60,50,80,40);
       p.add(b);
       a.add(p);
       a.setSize(400,400);
       a.setLayout(null);
       a.setVisible(true);
    }

    public static void main(String args[]){
        new jPanel();
    }
}
```

Рис.3.6 — Приклад реалізації елементу JPanel

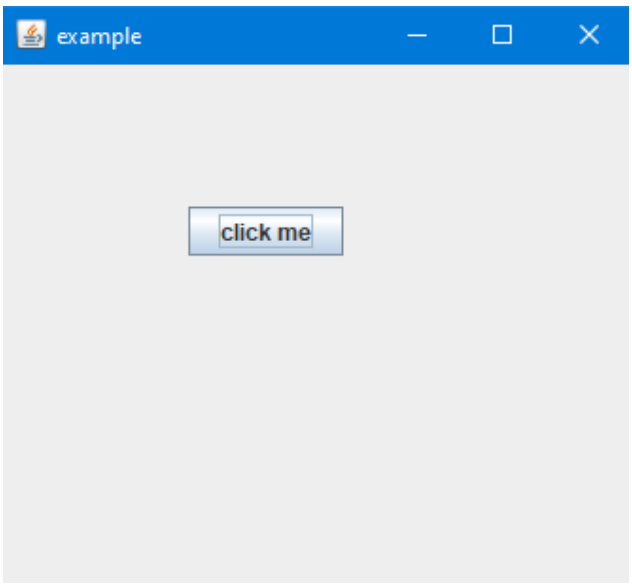

Рис.3.7 — Створений елемент JPanel

JMenuClass

Цей компонент унаслідувався від класу JMenuItem і є елементом спадного меню, який відображається з рядка меню.

```
import javax.swing.*;

public class jMenu {
    JMenu menu;
    JMenuItem a1,a2;
        jMenu(){
            JFrame a = new JFrame("Example");
            menu = new JMenu("options");
            JMenuBar ml = new JMenuBar();
            a1 = new JMenuItem("example");
            a2 = new JMenuItem("example1");
            menu.add(a1);
            menu.add(a2);
            ml.add(menu);
            a.setJMenuBar(ml);
            a.setSize(400,400);
            a.setLayout(null);
            a.setVisible(true);
        }
        public static void main(String args[]){
            new jMenu();
        }
}
```

Рис.3.8 — Приклад реалізації елементу JMenu

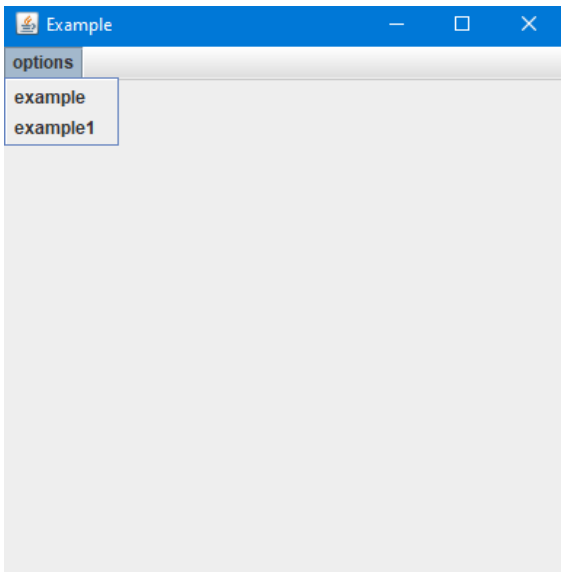

Рис.3.9 — Створений елемент JMenu

JList Class

Об'єкт класу JList наслідує клас JComponent та представляє список текстових елементів.

```
import javax.swing.*;

public class jList {
    jList(){
       JFrame a  = new JFrame("example");
       DefaultListModel<String> l = new DefaultListModel< >();
       l.addElement("first item");
       l.addElement("second item");
       JList<String> b = new JList< >(l);
       b.setBounds(100,100,75,75);
       a.add(b);
       a.setSize(400,400);
       a.setVisible(true);
       a.setLayout(null);
    }
    public static void main(String args[]){
        new jList();
    }
}
```

Рис.3.10 — Приклад реалізації елементу JList

Рис.3.11 — Створений елемент JList

JLabel Class

Цей компонент використовується для відображення тексту в контейнері та також успадковує клас JComponent.

```
import javax.swing.*;

public class jLabel {

    public static void main(String args[]){
        JFrame a = new JFrame("example");
        JLabel bl;
        bl = new JLabel("example");
        bl.setBounds(40,40,90,20);
        a.add(bl);
        a.setSize(400,400);
        a.setLayout(null);
        a.setVisible(true);
    }
}
```

Рис.3.12 — Приклад реалізації елементу JLabel

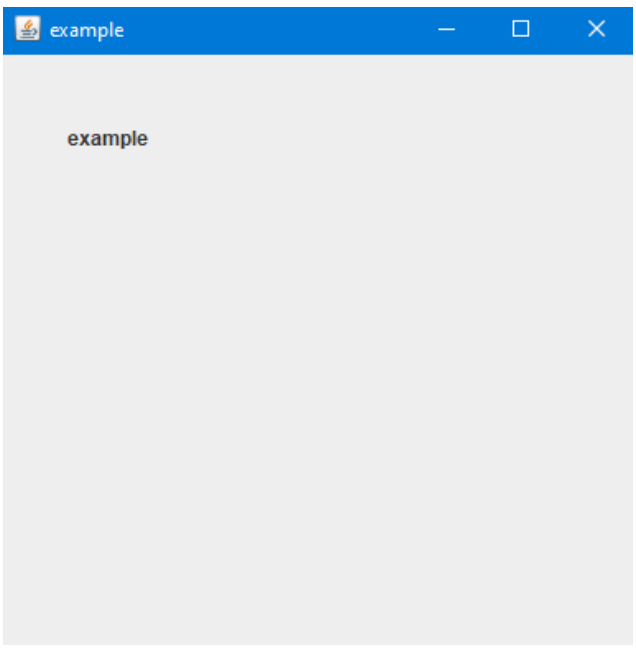

Рис.3.13 — Створений елемент JLabel

JComboBox Class

Цей компонент наслідує клас JComponent і призначений для відображення спливаючого вибіркового меню.

```
import javax.swing.*;

public class jComboBox {
    JFrame a;
        jComboBox(){
            a = new JFrame("example");
            String courses[] = { "first","second", "third"};
            JComboBox c = new JComboBox(courses);
            c.setBounds(40,40,90,20);
            a.add(c);
            a.setSize(400,400);
            a.setLayout(null);
            a.setVisible(true);
        }
        public static void main(String args[]){
            new jComboBox();
        }
}
```

Рис.3.14 — Приклад реалізації елементу JComboBox

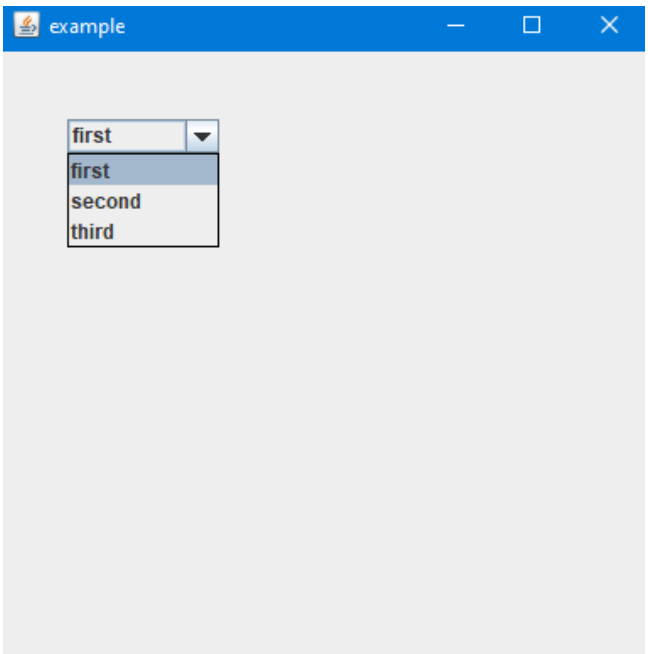

Рис.3.15 — Створений елемент JComboBox

3.2. JDBC: Apache Derby

Apache Derby є однією з баз даних, яка може бути використана в Java-проектах. Технологія JDBC Derby дозволяє Java-розробникам підключатись до бази даних Derby. Apache NetBeans є інтегрованою розробкою середовищем для Java, яка включає в себе підтримку JDBC Derby. Це означає, що ви можете створювати, налаштовувати та взаємодіяти з базами даних Derby безпосередньо з свого IDE.

Щоб підключитись до бази даних Derby в NetBeans, вам потрібно спочатку створити підключення до бази даних. Для цього ви можете використовувати вбудовану утиліту NetBeans для створення підключення до бази даних, вказавши параметри підключення, такі як URL бази даних та ім'я користувача.

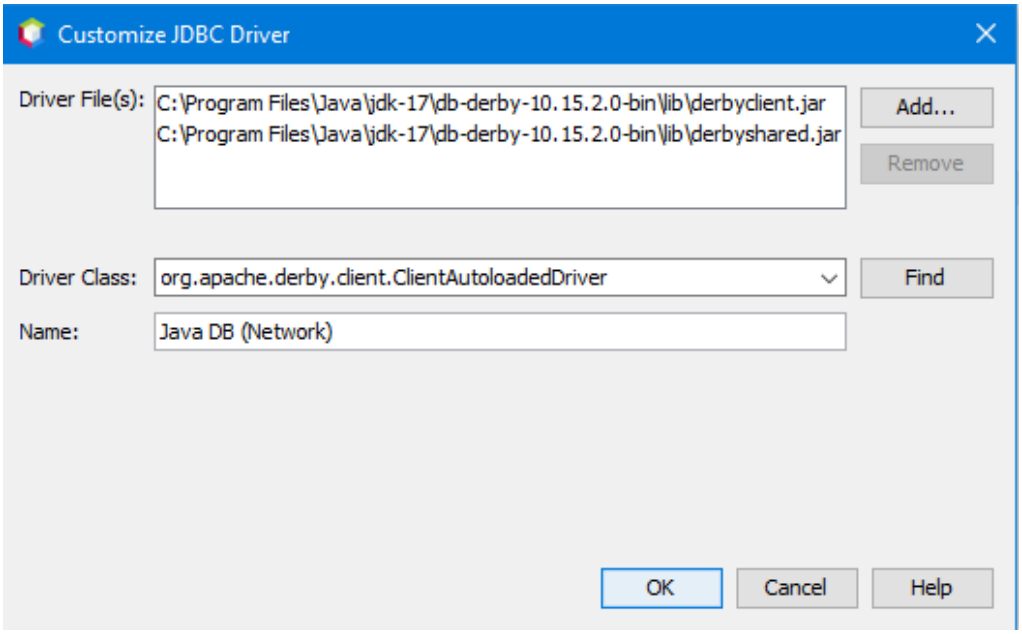

Рис.3.16— Підключення драйвера бази даних Derby

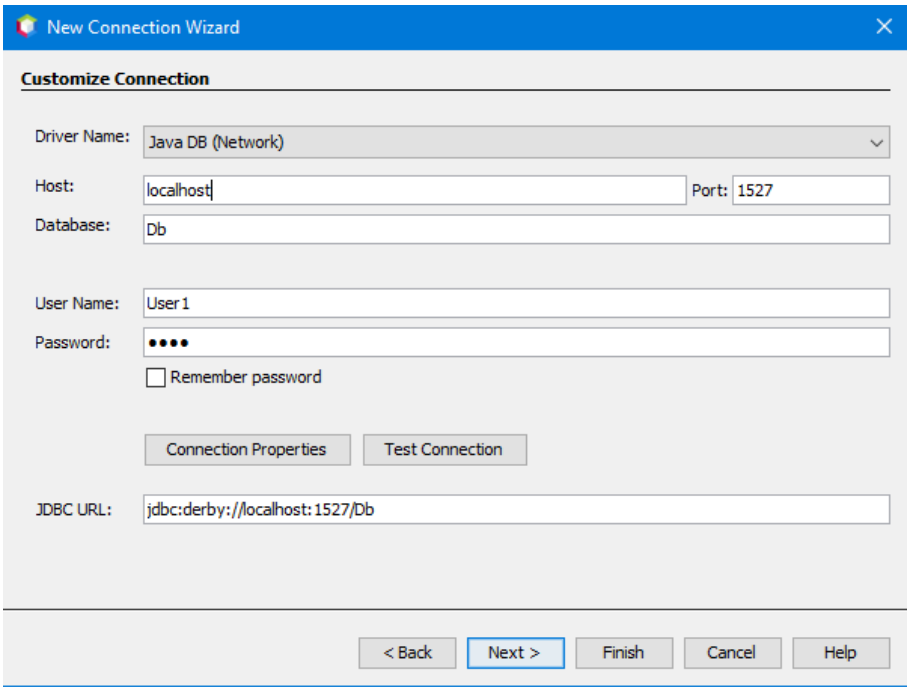

Рис.3.17— Створення бази даних Derby

Після створення підключення до бази даних в NetBeans, ви можете використовувати JDBC Derby для створення, збереження, вибірки, оновлення та видалення даних з бази даних.

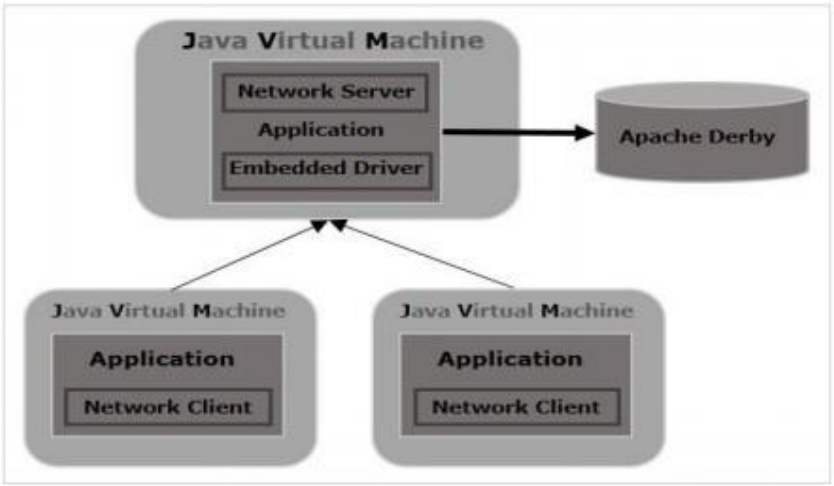

Рис.3.18— Взаємодія бази даних Derby

Окрім того, Apache Derby має підтримку транзакцій, дозволяючи вам створювати безпечні транзакції баз даних, які можна відмінити у випадку помилки. Це забезпечує цілісність даних та запобігає їх втраті у випадку непередбачуваних ситуацій.

Таким чином, технологія JDBC Derby в Apache NetBeans дозволяє та управляти базами даних Derby у своїх Java-проектах, забезпечуючи безпечну та надійну роботу з даними.

3.3. Створення плану інформаційно - довідкової системи

На основі обраних технологій та баз даних, був розроблений план взаємодії програмної частини з створеними базами даних:

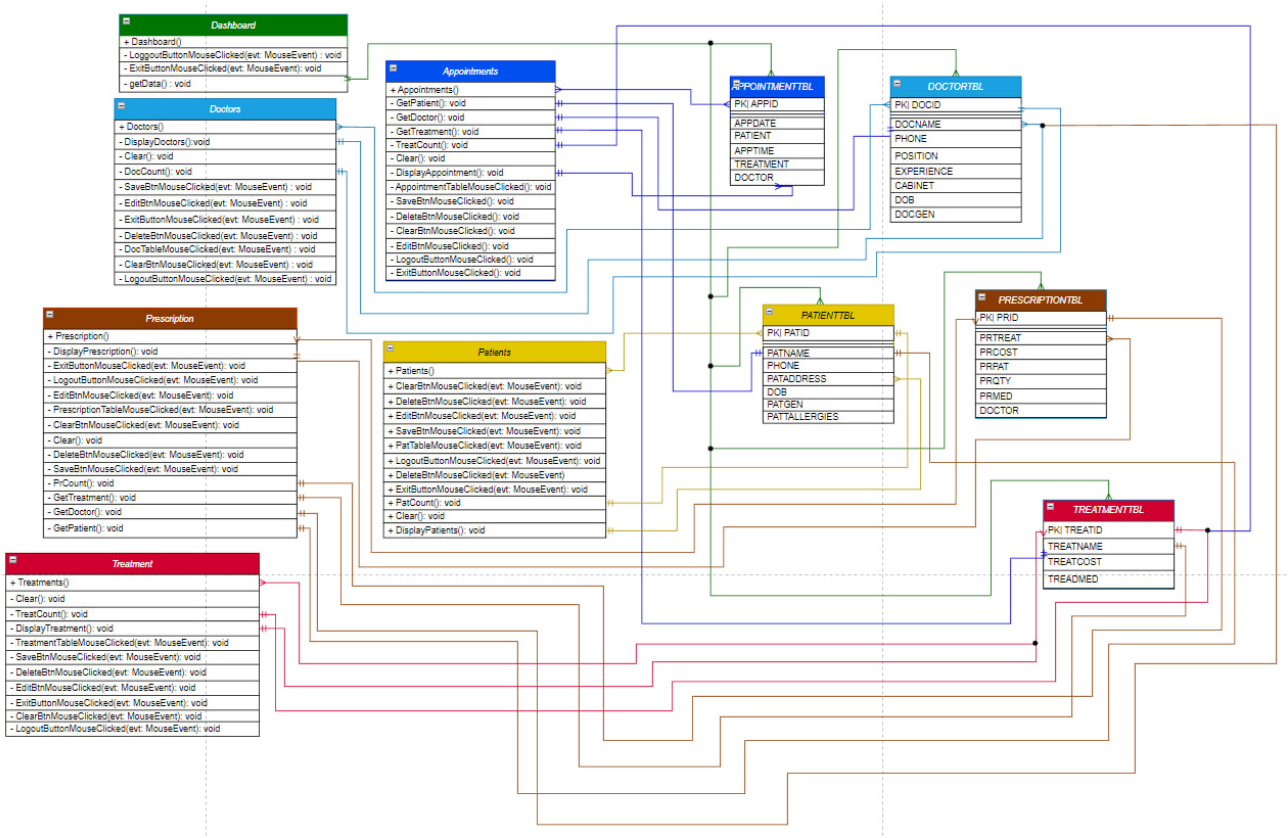

Рис.3.19— План додатку

3.4 Розробка функціоналу

Клас Splash:

При запуску додатку, програма створює вікно з завантаженням, яке відображається на екрані. Вікно має на меті показати інформацію про процес завантаження додатку. Вікно завантаження має прогрес-бар, щоб візуально показати, що додаток завантажується.

Під час завантаження додатку система може виконувати різні операції, такі як перевірка цілісності, завантаження або ініціалізація необхідних компонентів.

Коли завантаження завершується, система відкриває вікно логіну, де користувач може пройти аутентифікацію.

Рис.3.20— Вікно завантаження додатку

Клас Login:

Система використовує механізм перевірки логіну та паролю, за допомогою методів аутентифікації. Коли користувач вводить свої дані, вона їх перевіряє на наявність співпадінь в базі даних, у відомостях, що знаходяться в базах даних.

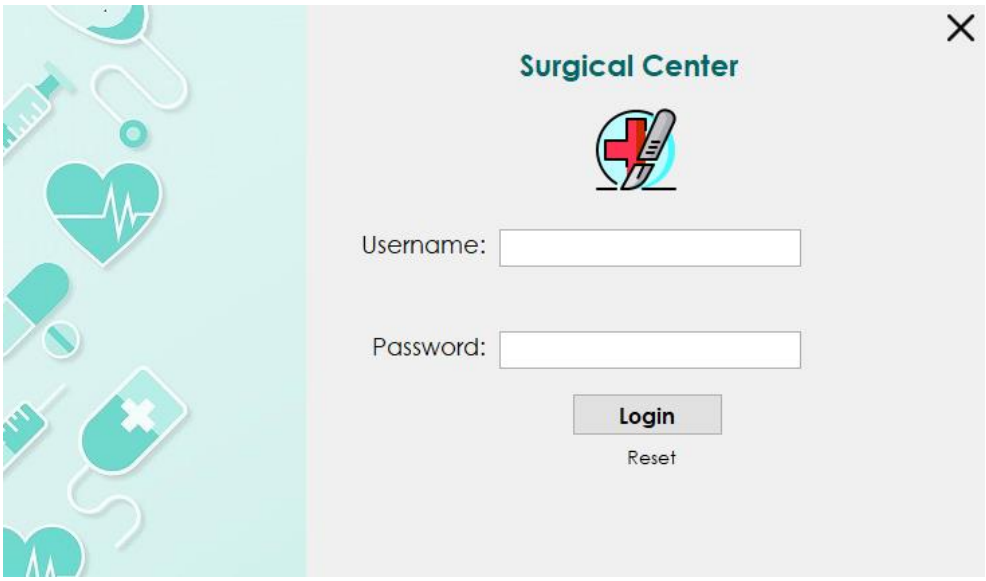

Рис.3.21— Вікно логіну додатку

У разі успішної операції, система надає доступ до своїх функцій та можливостей. Але, якщо введені дані не співпадають з тими, що збережені в системі, користувачу відображається вікно з помилкою. Це вікно повідомляє користувача про невдалу спробу аутентифікації та надає можливість повторного введення логіну та пароля. Такий підхід до перевірки допомагає забезпечити безпеку та зручність для користувача.

Клас Patient:

Клас Patient виконує операції з пацієнтами. При завантаженні програми клас Patient виконує запит на показ всіх пацієнтів у системі. Цей запит передається до бази даних, щоб отримати повну інформацію про всіх пацієнтів, що зареєстровані в системі.

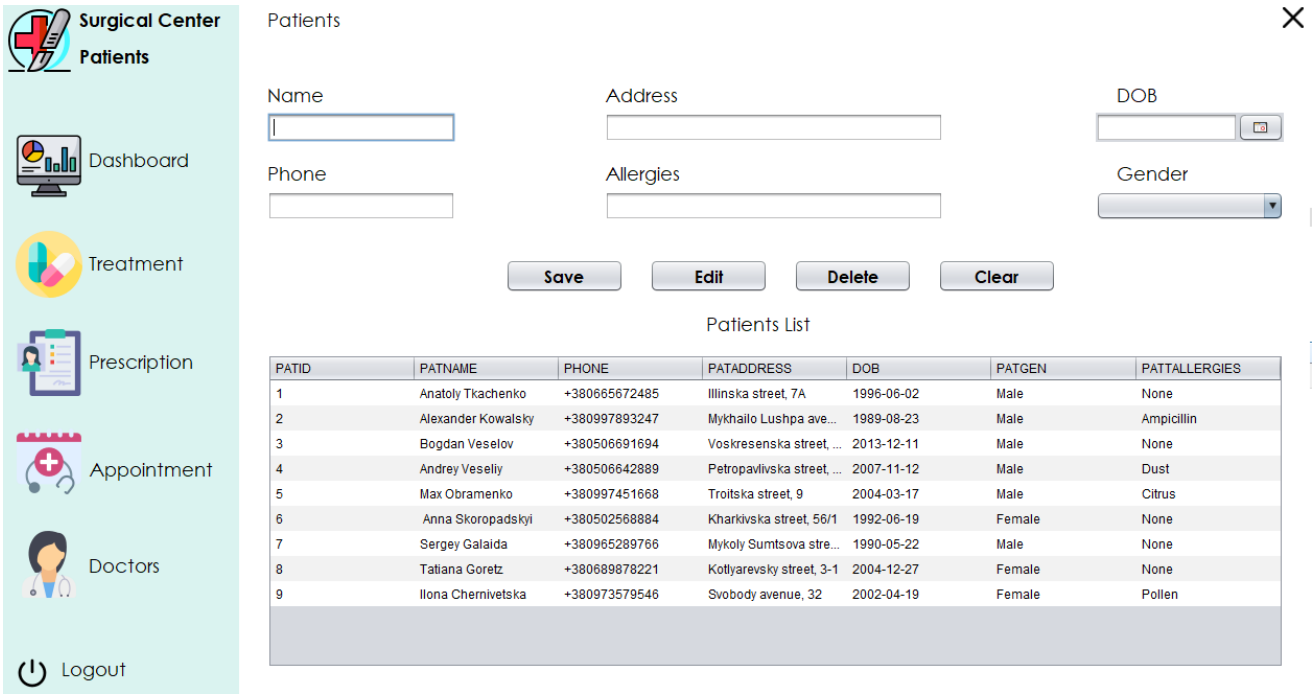

Рис.3.22— Вкладка Patient

При створенні нового пацієнта, користувач вводить відповідні дані. Ці дані разом із запитом на додавання нового пацієнта надсилаються до таблиці з пацієнтами PATIENTTBL. Так створюється новий пацієнт в базі даних.

При редагуванні існуючих даних пацієнта, оновлені дані надсилаються до бази даних. Це забезпечує та точність інформації при вводі даних про пацієнтів у системі.

При видаленні даних про пацієнта з системи, відповідний запит надсилається до бази даних. Цей запит вказує набір конкретний даних потрібно видалити з таблиці. Таким способом, база даних використовується для збереження, оновлення та видалення даних пацієнтів. Це спрощує управління медичними записами та забезпечує актуальність даних.

Нижче привдений код з реалізацією цього класу:

```java
package dyplomasurgeoncentre;

import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import net.proteanit.sql.DbUtils;

/**

Клас Patients відповідає за управління пацієнтами
*/
public class Patients extends javax.swing.JFrame {

/**
Створює нову форму Patients
*/
    public Patients() {
        initComponents();
        DisplayPatients();
        PatCount();
        PatGen.setSelectedItem(null);

    }
```

Рис.3.23— Створення нової форми пацієнта

```java
// Обробник події для кнопки "ClearBtn"
    private void ClearBtnMouseClicked(java.awt.event.MouseEvent evt) {
        Clear();
    }

// Обробник події видалення для кнопки "DeleteBtn"
    private void DeleteBtnMouseClicked(java.awt.event.MouseEvent evt) {
        if (Key == 0) {
            JOptionPane.showMessageDialog(this, "Select The Patient");
        } else {
            try {

                Con = DriverManager.getConnection("jdbc:derby://localhost:1527/SurgeonCentreDB", "User1", "Pass");
                String Query = "Delete from User1.PatientTbl where PATID=" + Key;
                Statement Add = Con.createStatement();
                Add.executeUpdate(Query);
                JOptionPane.showMessageDialog(this, "Patient Deleted Successfully");
                DisplayPatients();
                Clear();
            } catch (Exception Ex) {
                Ex.printStackTrace();
            }
        }
    }
```

Рис.3.24— Ініціалізація кнопки Clear та DeleteButton

```
// Обробник події зміни для кнопки "EditBtn"
    private void EditBtnMouseClicked(java.awt.event.MouseEvent evt) {
        if (Key == 0) {
            JOptionPane.showMessageDialog(this, "Select The Patient");
        } else {
            try {
                Con = DriverManager.getConnection("jdbc:derby://localhost:1527/SurgeonCentreDB", "User1", "Pass");
                String Query = "Update User1.PatientTbl set PatName='" + PatNameTb.getText() + "'" + ",PHONE='" +
                        PhoneTb.getText() + "'" + ",PATADDRESS='" + AddressTb.getText() + "'" +
                        ",PATGEN='" + PatGen.getSelectedItem().toString() + "'" +
                        ",PATTALLERGIES='" + AllergieTb.getText() + "'";


                if (DOB.getDate() != null) {
                    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
                    String dobString = dateFormat.format(DOB.getDate());
                    Query += ",DOB='" + dobString + "'";
                }

                Query += " where PATID=" + Key;
                Statement Add = Con.createStatement();
                Add.executeUpdate(Query);
                JOptionPane.showMessageDialog(this, "Patient Updated Successfully");
                DisplayPatients();
                Clear();
            } catch (Exception Ex) {
                Ex.printStackTrace();
            }
        }
    }
```

Рис.3.25— Створення кнопки Edit

```
// Обробник події збереження для кнопки "SaveBtn"
    private void SaveBtnMouseClicked(java.awt.event.MouseEvent evt) {
        if (PatNameTb.getText().isEmpty() || AllergieTb.getText().isEmpty() || AddressTb.getText().isEmpty() ||
                PhoneTb.getText().isEmpty()|| PatGen.getSelectedItem() == null || DOB.getDate() == null) {

        JOptionPane.showMessageDialog(this, "Missing Information");
        } else {
            try {

                PatCount();
                Con = DriverManager.getConnection("jdbc:derby://localhost:1527/SurgeonCentreDB", "User1", "Pass");
                SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
                String dobString = dateFormat.format(DOB.getDate());
                PreparedStatement add = Con.prepareStatement("INSERT INTO PatientTbl (PATID, PATNAME,"
                        + " PHONE, " + "PATADDRESS," + " DOB, PATGEN, PATTALLERGIES) VALUES (?,?,?,?,?,?,?)");
                add.setString(1, String.valueOf(PatId));
                    add.setString(2, PatNameTb.getText());
                    add.setString(3, PhoneTb.getText());
                    add.setString(4, AddressTb.getText());
                    add.setString(5, dobString);
                    add.setString(6, PatGen.getSelectedItem().toString());
                    add.setString(7, AllergieTb.getText());

                    int row = add.executeUpdate();
                    JOptionPane.showMessageDialog(this, "Patient Added Successfully");
                    Con.close();
                    DisplayPatients();
                    Clear();
            } catch (Exception Ex) {
                Ex.printStackTrace();

            }

        }
    }
```

Рис.3.26— Створення кнопки Save

Кнопки Save, Edit, Delete та Clear використовуються для збереження, зміни, видалення та очистки полів взаємодії, тому вони будуть повторюватися в інших класах.

```
// Обробник події натискання на таблицю PatTable
    private void PatTableMouseClicked(java.awt.event.MouseEvent evt) {
        DefaultTableModel model = (DefaultTableModel) PatTable.getModel();
        int MyIndex = PatTable.getSelectedRow();
        Key = Integer.valueOf(model.getValueAt(MyIndex, 0).toString());
        PatNameTb.setText(model.getValueAt(MyIndex, 1).toString());
        PhoneTb.setText(model.getValueAt(MyIndex, 2).toString());
        AddressTb.setText(model.getValueAt(MyIndex, 3).toString());

        String dobString = model.getValueAt(MyIndex, 4).toString();

        if (dobString != null && !dobString.isEmpty() && !dobString.equals("null")) {
            try {
                SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
                Date dobDate = dateFormat.parse(dobString);
                DOB.setDate(dobDate);
            } catch (ParseException ex) {
                ex.printStackTrace();
            }
        }

        PatGen.setSelectedItem(model.getValueAt(MyIndex, 5).toString());
        AllergieTb.setText(model.getValueAt(MyIndex, 6).toString());
    }
```

Рис.3.27— Створення таблиці пацієнтів

Обробники подій для кнопок використовуються у якості переходів між розділами додатку, тому вони теж будуть повторюватися в інших класах.

```
// Обробник події для кнопки "LogoutButton"
    private void LogoutButtonMouseClicked(java.awt.event.MouseEvent evt) {
        new Login().setVisible(true);
        this.dispose();
    }

// Обробник події для елемента "Dashboard"
    private void DashboardMouseClicked(java.awt.event.MouseEvent evt) {
        new Dashboard().setVisible(true);
        this.dispose();
    }

// Обробник події для елемента "Treatment"
    private void TreatmentMouseClicked(java.awt.event.MouseEvent evt) {
        new Treatment().setVisible(true);
        this.dispose();
    }

// Обробник події для елемента "Prescription"
    private void PrescriptionMouseClicked(java.awt.event.MouseEvent evt) {
        new Prescription().setVisible(true);
        this.dispose();
    }

// Обробник події для елемента "Appointment"
    private void AppointmentMouseClicked(java.awt.event.MouseEvent evt) {
        new Appointments().setVisible(true);
        this.dispose();
    }
```

Рис.3.28— Обробники подій 1

```
// Обробник події для елемента "IconDashboard"
    private void IconDashboardMouseClicked(java.awt.event.MouseEvent evt) {
        new Dashboard().setVisible(true);
        this.dispose();
    }

// Обробник події для елемента "IconTreatment"
    private void IconTreatmentMouseClicked(java.awt.event.MouseEvent evt) {
        new Treatment().setVisible(true);
        this.dispose();
    }

// Обробник події для елемента "IconPrescription"
    private void IconPrescriptionMouseClicked(java.awt.event.MouseEvent evt) {
        new Prescription().setVisible(true);
        this.dispose();
    }

// Обробник події для елемента "IconAppointment"
    private void IconAppointmentMouseClicked(java.awt.event.MouseEvent evt) {
        new Appointments().setVisible(true);
        this.dispose();
    }

// Обробник події для елемента "LogoutIcon"
    private void LogoutIconMouseClicked(java.awt.event.MouseEvent evt) {
        new Login().setVisible(true);
        this.dispose();
    }

// Обробник події для елемента "Doctors"
    private void DoctorsMouseClicked(java.awt.event.MouseEvent evt) {
        new Doctors().setVisible(true);
        this.dispose();
    }

// Обробник події для елемента "IconDoctors"
    private void IconDoctorsMouseClicked(java.awt.event.MouseEvent evt) {
        new Doctors().setVisible(true);
        this.dispose();
    }

// Обробник події для елемента "ExitButton"
    private void ExitButtonMouseClicked(java.awt.event.MouseEvent evt) {
        System.exit(0);
    }
```

Рис.3.29— Обробники подій 2

```
Connection Con = null;
Statement St = null, Stl = null;
ResultSet Rs = null, Rsl = null;
int Key = 0;
int PatId = 0;

// Функція для підрахунку пацієнтів у таблиці
private void PatCount() {
    try {
        Stl = Con.createStatement();
        Rsl = Stl.executeQuery("select Max(PATID)from User1.PatientTbl");
        Rsl.next();
        PatId = Rsl.getInt(1) + 1;
    } catch (Exception Ex) {
    }
}

// Функція для очищення полів введення
private void Clear() {
    PatNameTb.setText("");
    PhoneTb.setText("");
    AllergieTb.setText("");
    AddressTb.setText("");
    DOB.setDate(null);
    PatGen.setSelectedItem(null);
}

// Функція для відображення пацієнтів у таблиці
private void DisplayPatients() {
    try {
        Con = DriverManager.getConnection("jdbc:derby://localhost:1527/SurgeonCentreDB", "User1", "Pass");
        St = Con.createStatement();
        Rs = St.executeQuery("Select * from User1.PatientTbl");
        PatTable.setModel(DbUtils.resultSetToTableModel(Rs));
    } catch (Exception Ex) {
    }
}
```

Рис.3.30 — Створення блоків коду з підрахунком пацієнтів, очищенням полів інформації та показом пацієнтів

Клас Doctors:

Клас Doctors виконує операції з лікарями. Робота починається з запиту на відображення всіх наявних спеціалістів. При створенні нового лікаря користувач вводить необхідні дані, які надсилаються до таблиці бази даних DOCTORTBL. Ці дані включають інформацію про лікаря, його спеціалізацію та контактні дані. Система зберігає ці дані у таблиці бази даних для їх використання.

При редагуванні даних про спеціаліста внесені зміни надсилаються до бази даних. Коли користувач вносить зміни в інформацію про лікаря, система забезпечує актуалізацію полів таблиці бази даних DOCTORTBL.

При видаленні лікаря, надсилається запит на видалення відповідних даних з бази даних. Система виконує цей запит і видаляє інформацію про лікаря з таблиці DOCTORTBL. Отже, клас Doctors взаємодіє з базою даних для забезпечення

доступу до інформації про лікарів, додавання, редагування та видалення даних, забезпечуючи ефективну роботу з медичним персоналом.

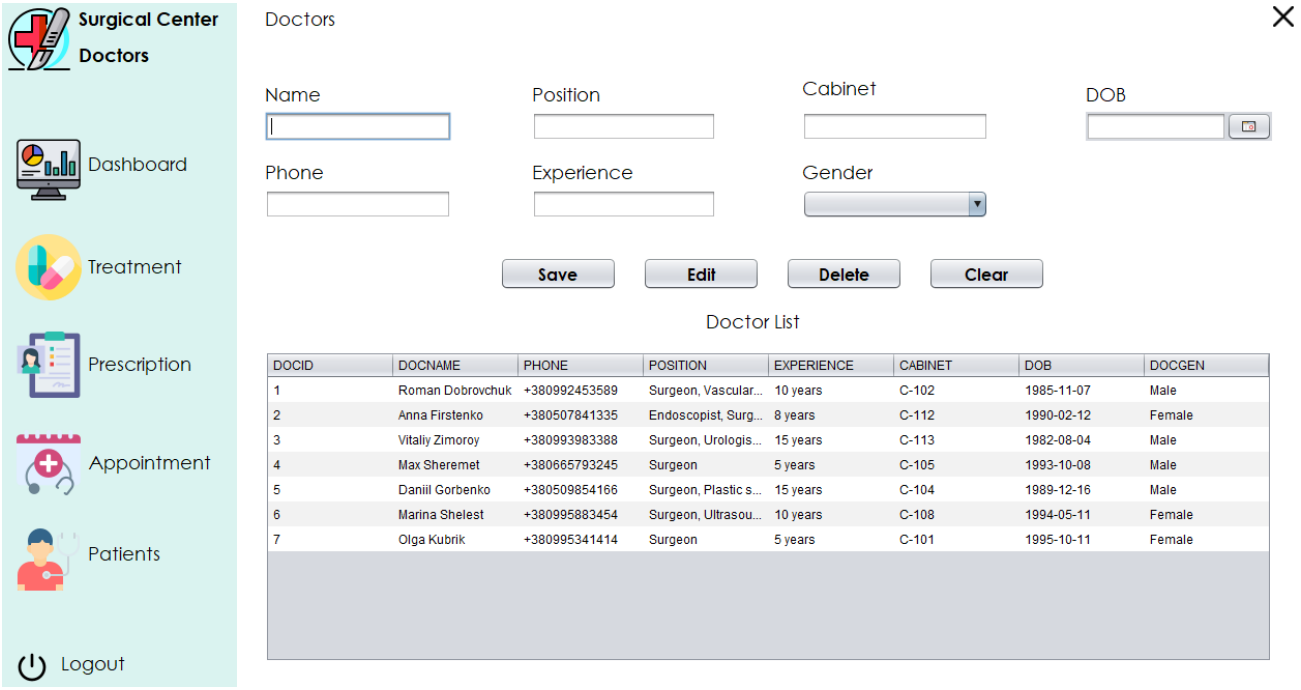

Рис.3.31— Вкладка Doctors

Клас Treatments:

Клас Treatments виконує операції з послугами. Система надсилає відповідний запит до бази даних, щоб отримати повний перелік послуг.

При створенні нової послуги користувач вводить дані, які надсилаються до таблиці бази даних TREATMENTTBL. Ці дані можуть включати такі характеристики як: назву послуги, ціну та інші.

При редагуванні даних всі внесені правки надсилаються до бази даних. Коли користувач вносить зміни, то система актуалізує інформацію у таблиці TREATMENTTBL.

При видаленні послуги з системи, надсилається запит на видалення даних з таблиці бази даних. Система виконує цей запит і видаляє інформацію про послугу з таблиці TREATMENTTBL.

Клас Treatments взаємодіє з базою даних для забезпечення доступу до інформації про послуги. Це спрощує як організацію роботи центру, так і надання якісних послуг своїм клієнтам.

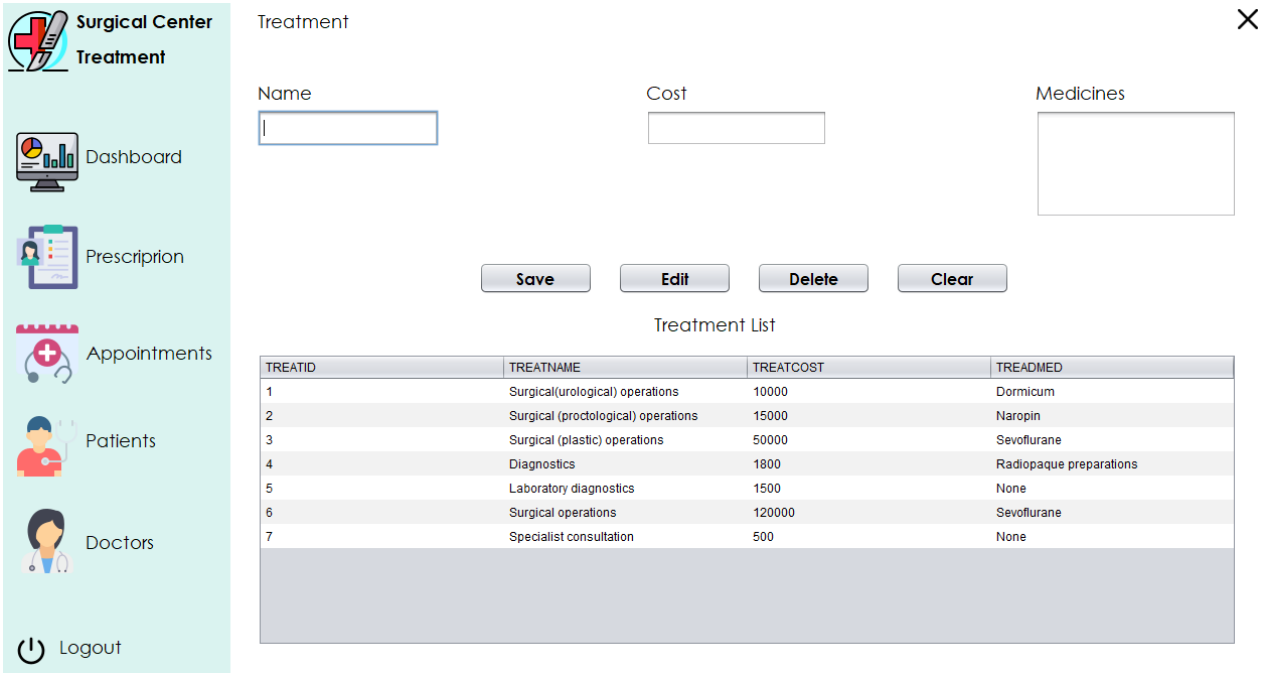

Рис.3.32— Вкладка Treatment

Клас Prescription:

Клас Prescription виконує різноманітні операції пов'язані з лікуванням. Система відправляє запит до бази даних, щоб отримати відповідні дані про лікування.

При створенні нового лікування користувач вводить дані, які надсилаються до таблиці PRESCRIPTIONTBL. Ці дані можуть включати інформацію про пацієнта, прописані ліки, кількість, ціну та інші.

При редагуванні даних про лікування всі внесені зміни надсилаються до бази даних. Коли користувач вносить зміни в інформацію про лікування, система забезпечує актуалізацію полів інформації у базі даних PRESCRIPTIONTBL.

При видаленні даних про лікування, надсилається запит на видалення даних з таблиці бази даних. Система виконує цей запит і видаляє інформацію про лікування з таблиці PRESCRIPTIONTBL.

Клас Prescription здійснює взаємодію з базою даних для забезпечення ефективного контролю над призначеними лікуваннями.

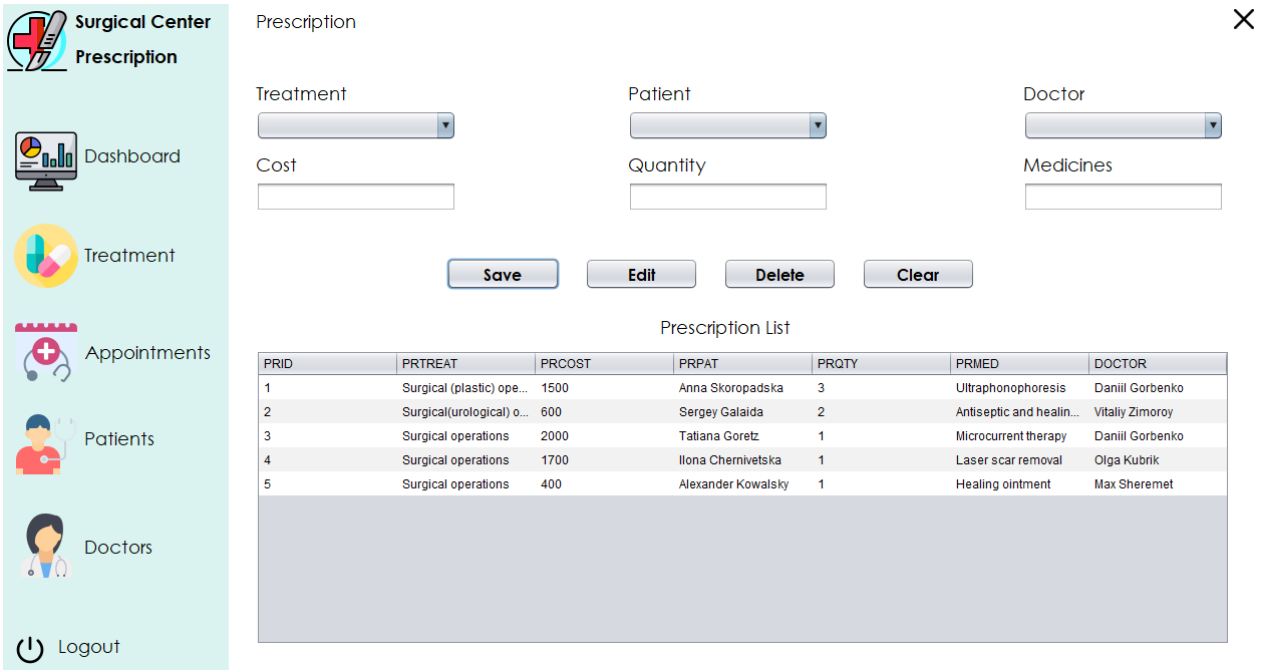

Рис.3.33— Вкладка Prescription

Клас Appointments:

Клас Appointments відповідає за операції, пов'язані зі зустрічами. Система ініціює запит на показ зустрічей, щоб отримати відповідні дані про заплановані зустрічі.

При створенні нової зустрічі користувач вводить дані, які разом надсилаються до таблиці APPOINTMENTSTBL. Ці дані можуть включати інформацію про пацієнта, дату та час зустрічі, лікаря.

При редагуванні даних про певну зустріч всі внесені зміни надсилаються до бази даних APPOINTMENTSTBL, щоб зберегти оновлену інформацію.

У разі видалення даних про зустріч з системи, надсилається запит на видалення відповідних даних з таблиці бази даних. Клас Appointments здійснює взаємодію з базою даних для забезпечення доступу до інформації про заплановані зустрічі, додавання, редагування та видалення зустрічей, що дозволяє ефективно керувати процесом запланованих зустрічей.

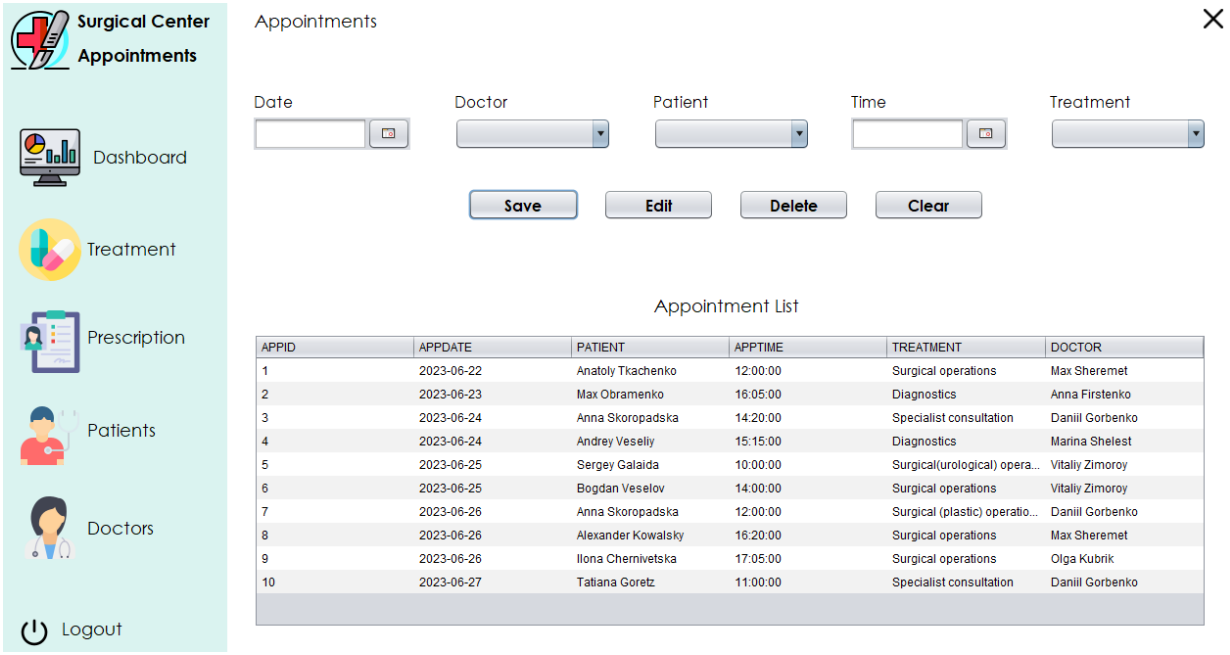

Рис.3.34— Вкладка Appointments

Нижче привдений код з реалізацією цього класу:

```
package dyplomasurgeoncentre;

import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import net.proteanit.sql.DbUtils;
import java.sql.SQLException;
import java.util.Calendar;

/**
 *
 * @author 1
 */
public class Appointments extends javax.swing.JFrame {

    /**
Створює нову форму Appointments
*/
    public Appointments() {
        initComponents();
        GetPatient();
        GetDoctor();
        GetTreatment();
        TreatCount();
        DisplayAppointment();
        Clear();
    }
```

Рис.3.35— Створення нової форми Appointments

```
Connection Con = null;
Statement St = null, St1 = null;
ResultSet Rs = null, Rs1 = null;

// Функція для отримання пацієнтів з таблиці Patients
private void GetPatient() {
    try {
        Con = DriverManager.getConnection("jdbc:derby://localhost:1527/SurgeonCentreDB", "User1", "Pass");
        St = Con.createStatement();
        String query = "Select * from User1.PatientTbl";
        Rs = St.executeQuery(query);
        while (Rs.next()) {
            String MyPat = Rs.getString("PatName");
            PatName.addItem(MyPat);
        }
    } catch (Exception Ex) {
        Ex.printStackTrace();
    } finally {
        try {
            if (Rs != null)
                Rs.close();
            if (St != null)
                St.close();
            if (Con != null)
                Con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Рис.3.36— Блок коду, що потрібен для отримання інформації про пацієнтів

```
// Функція для отримання спеціалістів з таблиці Doctors
private void GetDoctor() {
    try {
        Con = DriverManager.getConnection("jdbc:derby://localhost:1527/SurgeonCentreDB", "User1", "Pass");
        St = Con.createStatement();
        String query = "Select * from User1.DoctorTbl";
        Rs = St.executeQuery(query);
        while (Rs.next()) {
            String MyDoc = Rs.getString("DocName");
            DocName.addItem(MyDoc);
        }
    } catch (Exception Ex) {
        Ex.printStackTrace();
    } finally {
        try {
            if (Rs != null)
                Rs.close();
            if (St != null)
                St.close();
            if (Con != null)
                Con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Рис.3.37— Блок коду, що потрібен для отримання інформації про спеціалістів

```
// Функція для отримання послуг з таблиці Treatment
private void GetTreatment() {
    try {
        Con = DriverManager.getConnection("jdbc:derby://localhost:1527/SurgeonCentreDB", "User1", "Pass");
        St = Con.createStatement();
        String query = "Select * from User1.TreatmentTbl";
        Rs = St.executeQuery(query);
        while (Rs.next()) {
            String MyTreat = Rs.getString("TREATNAME");
            TreatName.addItem(MyTreat);
        }
    } catch (Exception Ex) {
        Ex.printStackTrace();
    } finally {
        try {
            if (Rs != null)
                Rs.close();
            if (St != null)
                St.close();
            if (Con != null)
                Con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Рис.3.383— Блок коду, що потрібен для отримання інформації про послуги

```
// Обробник події натискання на таблицю AppointmentTable
int Key = 1000;
private void AppointmentTableMouseClicked(java.awt.event.MouseEvent evt) {
 DefaultTableModel model = (DefaultTableModel) AppointmentTable.getModel();
int MyIndex = AppointmentTable.getSelectedRow();
Key = Integer.valueOf(model.getValueAt(MyIndex, 0).toString());
String appDateString = model.getValueAt(MyIndex, 1).toString();
String appTimeString = model.getValueAt(MyIndex, 3).toString();

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
SimpleDateFormat timeFormat = new SimpleDateFormat("HH:mm:ss");

if (appDateString != null && !appDateString.isEmpty() && !appDateString.equals("null")) {
    try {
        Date appDate = dateFormat.parse(appDateString);
        AppDate.setDate(appDate);
    } catch (ParseException ex) {
        ex.printStackTrace();
    }
}

if (appTimeString != null && !appTimeString.isEmpty() && !appTimeString.equals("null")) {
 try {
     Date appTime = timeFormat.parse(appTimeString);
     AppTime.setDate(appTime);
 } catch (ParseException ex) {
     ex.printStackTrace();
 }
 }

 PatName.setSelectedItem(model.getValueAt(MyIndex, 2).toString());
 TreatName.setSelectedItem(model.getValueAt(MyIndex, 4).toString());
 DocName.setSelectedItem(model.getValueAt(MyIndex, 5).toString());
 }
```

Рис.3.39— Створення таблиці зустрічей

```
// Функція для підрахунку послуг у таблиці
int AppId;
private void TreatCount() {
    try {
        Con = DriverManager.getConnection("jdbc:derby://localhost:1527/SurgeonCentreDB", "User1", "Pass");
        St1 = Con.createStatement();
        Rs1 = St1.executeQuery("select Max(APPID) from User1.AppointmentTbl");
        Rs1.next();
        AppId = Rs1.getInt(1) + 1;
    } catch (Exception Ex) {
        Ex.printStackTrace();
    } finally {
        try {
            if (Rs1 != null)
                Rs1.close();
            if (St1 != null)
                St1.close();
            if (Con != null)
                Con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Рис.3.40— Блок коду, що потрібен для підрахунку послуг

```
// Функція для відображення зустрічей у таблиці
    private void DisplayAppointment() {
        try {
            Con = DriverManager.getConnection("jdbc:derby://localhost:1527/SurgeonCentreDB", "User1", "Pass");
            St = Con.createStatement();
            Rs = St.executeQuery("Select * from User1.AppointmentTbl");
            AppointmentTable.setModel(DbUtils.resultSetToTableModel(Rs));
        } catch (Exception Ex) {
            Ex.printStackTrace();
        } finally {
            try {
                if (Rs != null)
                    Rs.close();
                if (St != null)
                    St.close();
                if (Con != null)
                    Con.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
```

Рис.3.41— Блок коду, що відповідає за показ зустрічей

Клас Dashboard:

Клас Dashboard відповідає за надання статистики даних. Система відправляє відповідний запит до бази даних, щоб отримати необхідні дані з усіх таблиць.

Після надходження запиту до бази даних, система звертається до кожної

таблиці бази даних, отримує інформацію та обробляє її. Зібрані дані відображаються у вкладці Dashboard.

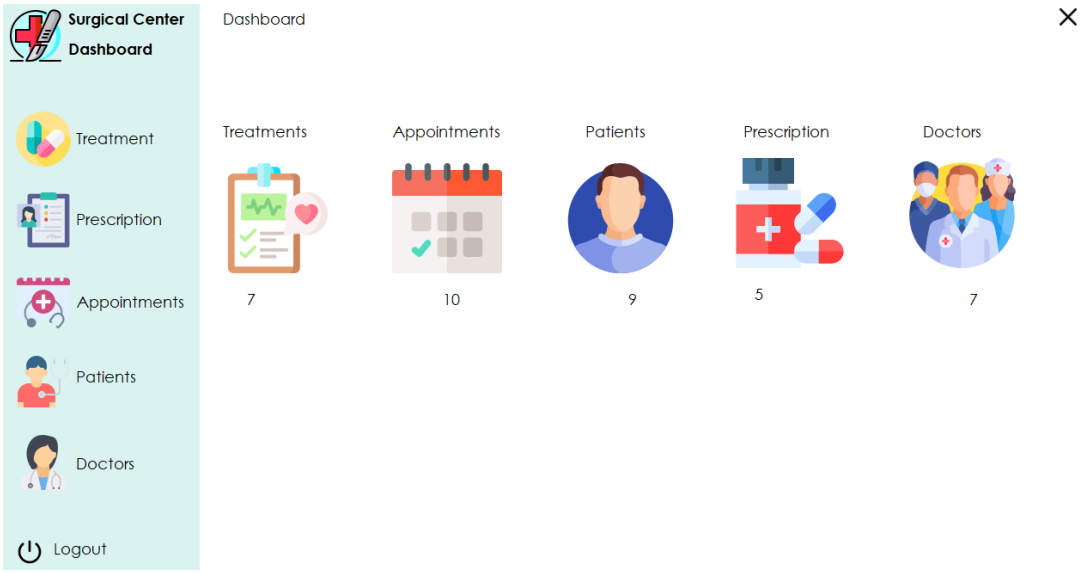

Рис.3.42— Вкладка Dashboard

Нижче привдений код з реалізацією цього класу:

```
package dyplomasurgeoncentre;

import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import net.proteanit.sql.DbUtils;


public class Dashboard extends javax.swing.JFrame {

    /**
Створює нову форму Dashboard
*/
    public Dashboard() {
        initComponents();
        getData();
    }
```

Рис.3.43— Створення нової форми статистики

```
Connection Con = null;
Statement St = null, St1 = null, St2 = null, St3 = null,St4 = null;
ResultSet Rs = null, Rs1 = null, Rs2 = null, Rs3 = null,Rs4 = null;

private void getData() {
    try {
        Con = DriverManager.getConnection("jdbc:derby://localhost:1527/SurgeonCentreDB", "User1", "Pass");
        St = Con.createStatement();
        St1 = Con.createStatement();
        St2 = Con.createStatement();
        St3 = Con.createStatement();
        St4 = Con.createStatement();
        Rs = St.executeQuery("select count(*) from User1.PatientTbl");
        Rs1 = St1.executeQuery("select count(*) from User1.TreatmentTbl");
        Rs2 = St2.executeQuery("select count(*) from User1.PrescriptionTbl");
        Rs3 = St3.executeQuery("select count(*) from User1.DoctorTbl");
        Rs4 = St4.executeQuery("select count(*) from User1.AppointmentTbl");
        while (Rs.next()) {
            PNumLbl.setText("" + Rs.getInt(1));
        }
        while (Rs1.next()) {
            TNumLbl.setText("" + Rs1.getInt(1));
        }
        while (Rs2.next()) {
            PrescriptionNumLbl.setText("" + Rs2.getInt(1));
        }
        while (Rs3.next()) {
            DoctorsNumLbl.setText("" + Rs3.getInt(1));
        }
        while (Rs4.next()) {
            AppLbl.setText("" + Rs4.getInt(1));
        }
    } catch (Exception Ex) {
    }

}
```

Рис.3.44— Отримання інформації з всіх таблиць баз даних для відображення їх в розділі Dashboard

ВИСНОВКИ

Розробка автоматизованих інформаційно-довідкових систем для медичних установ з використанням мови програмування Java є неабияк актуальною та перспективною темою. Java відома своєю зручністю та ефективністю у розробці програмного забезпечення, а також високим рівнем безпеки та надійності, тому використання Java для створення інформаційно-довідкових систем має численні переваги, включаючи кросплатформеність та широкий вибір бібліотек і фреймворків, що сприяють швидкій та зручній розробці програмного забезпечення.

В даній роботі було розглянуто процес розробки та автоматизації інформаційно-довідкових систем для медичних установ проаналізовано їх особливості та переваги, а також висвітлено принципи використання баз даних та створення ефективного програмного забезпечення для потреб медичного сектору.

Загалом, можна зробити висновок, що розробка таких систем для медичних установ є актуальною та перспективною темою, яка може значно покращити якість та ефективність медичного обслуговування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Components of JRE. URL:https://www.geeksforgeeks.org/jre-in-java/[1] (дата звернення: 07.05.2023).

2. Swing Program in JAVA. URL: https://www.educba.com/swing-program-in-java/ [2]

3. Conrod P., Tylee L. Learn Java GUI Applications - 11th Edition: A JFC Swing Tutorial. Biblebyte Books, 2019. 1122 p.

4. Schildt H. Java: The Complete Reference, Eleventh Edition. McGraw-Hill Education, 2018. 1248 p.

5. The Java language specification / ed. by G. J. 1955-, G. J. 1955-. 2nd ed. Boston : Addison-Wesley, 2000. 505 p.

6. Wielenga G. Beginning NetBeans IDE: For Java Developers. Apress, 2015. 284 p.

7. Harrer S., Lenhard J., Dietz L. Java By Comparison: Become a Java Craftsman in 70 Examples. Pragmatic Bookshelf, 2018. 208 p.

8. Sierra K. Head first Java. 2nd ed. Sebastopol, CA : O'Reilly, 2005. 688 p.

9. C M. R. Clean code: A Handbook of Agile Software Craftsmanship. Upper Saddle River, NJ : Prentice Hall, 2008. 431 p

10. Eckel B. Thinking in Java. 4th ed. Upper Saddle River, NJ : Prentice Hall, 2006. 1482 p.

11. Java Swing, Second Edition / B. Cole et al. O'Reilly Media, Inc., 2002. 1278 p.

12. Java Fundamentals: A Fast-Paced and Pragmatic Introduction to One of the World's Most Popular Programming Languages / G. Alankus et al. Packt Publishing, Limited, 2019. 408 p.

13. Derby Developer's Guide Version 10.14. URL: https://db.apache.org/derby/docs/10.15/getstart/getstartderby.pdf

14. Bloch J. Effective Java. Addison-Wesley Professional, 2018. 412 p.

15. Oaks S. Java Performance: In-Depth Advice for Tuning and Programming Java 8, 11, and Beyond. O'Reilly Media, Incorporated, 2020. 450 p.

16. Darwin I. F. Java Cookbook: Problems and Solutions for Java Developers. O'Reilly Media, Incorporated, 2020. 638 p.

17. Java Fundamentals: A Fast-Paced and Pragmatic Introduction to One of the World's Most Popular Programming Languages / G. Alankus et al. Packt Publishing, Limited, 2019. 408 p.