

**UNIwersYTET RZESZOWSKI**  
**WYDZIAŁ NAUK ŚCISŁYCH I TECHNICZNYCH**  
**INSTYTUT INFORMATYKI**



*Radosław Kierepka*  
134921

*Informatyka*

*Dokumentacja do projektu "Kolejka FIFO"*

Praca projektowa

Praca wykonana pod kierunkiem  
Mgr inż Ewa Żesławska

Rzeszów 2025



## Spis treści

<b>1. Wprowadzenie</b>	6
1.1. Opis założeń projektu	6
1.2. Zawartość pracy	7
<b>2. Opis struktury projektu</b>	8
2.1. Polecenie SQL:	8
2.2. Zarządzanie danymi i baza danych:	8
2.3. Dane Techniczne:	9
2.4. Działanie	9
<b>3. Prezentacja elementów projektu:</b>	10
3.1. Prezentacja warstwy użytkowej	10
3.2. Bazy Danych MySQL	13
3.3. Prezentacja kodu	15
<b>4. Podsumowanie</b>	20
4.1. Podsumowanie ogólne	20
4.2. Harmonogram realizacji projektu	20
4.3. Wnioski	20
4.4. Oświadczenie studenta o samodzielności pracy	21
<b>Bibliografia</b>	22
<b>Spis rysunków</b>	23
<b>Spis tabel</b>	24
<b>Spis listingów</b>	25

# 1. Wprowadzenie

Kolejka FIFO (First In, First Out) to struktura danych, w której elementy są przetwarzane w takiej kolejności, w jakiej zostały dodane. Oznacza to, że pierwszy element dodany do kolejki zostanie również jako pierwszy z niej usunięty — dokładnie jak kolejka ludzi w sklepie. Celem projektu jest stworzenie graficznej aplikacji komputerowej do zarządzania zamówieniami klientów przy użyciu struktury kolejki FIFO. Aplikacja umożliwia dodawanie zamówień, ich realizację oraz przeglądanie zapisanych danych klientów. Dane zapisywane są w bazie danych MySQL. Projekt łączy technologię Swing (interfejs graficzny) z JDBC (połączenie z bazą danych). Kluczowe cechy FIFO:

- Dodawanie — nowy element trafia na koniec kolejki.
- Usuwanie — element usuwany jest z początku kolejki.
- Zastosowanie — systemy kolejkowe, buforowanie, drukarki, planowanie zadań (np. w systemach operacyjnych), sieci komputerowe itp.

## 1.1. Opis założeń projektu

Celem poniższej pracy jest symulacja kolejki FIFO w języku Java w kontekście zamówień w sklepie. Podstawowym założeniem jest, aby obsługiwać zamówienia w kolejności, w jakiej zostały złożone. Oznacza to, że zamówienie, które pojawiło się jako pierwsze, będzie realizowane jako pierwsze, a dopiero w następnej kolejności te złożone później. Problemem rozwiązywanym przez aplikację jest brak uporządkowanego systemu obsługi zamówień, co często prowadzi do chaosu, opóźnień oraz braku przejrzystości w kolejności realizacji zleceń. Korzyści kolejki:

- Zmniejszenie ryzyka pomyłek
- Zadowolenie klientów
- Sprawiedliwość
- Prostota i przejrzystość
- Zachowanie kolejności obsługi

Głównym źródłem problemu jest brak informatyzacji w małych podmiotach, które nie posiadają systemu ERP ani dedykowanego narzędzia do zarządzania kolejkami. Problem ten jest ważny, ponieważ wpływa na jakość obsługi klienta, czas realizacji zamówień i efektywność operacyjną.

Wymagania funkcjonalne:

- Dodawanie nowego zamówienia do kolejki.
- Pobieranie zamówienia (realizacja) z kolejki.
- Zapisywanie zamówienia do bazy danych.
- Wyświetlanie listy klientów i ich zamówień z bazy danych
- Obsługa wielu produktów w jednym zamówieniu.

- Interfejs graficzny pozwalający na interakcję z użytkownikiem.

Wymagania niefunkcjonalne:

- Intuicyjny interfejs graficzny
- Działanie aplikacji offline z możliwością zapisu do lokalnej bazy danych
- Niska awaryjność i obsługa błędów
- Szybkość działania nawet przy większej liczbie zamówień
- Minimalne wymagania sprzętowe.

## 1.2. Zawartość pracy

W projekcie zawarto 7 klas o następującej hierarchii:

- GUI – klasa odpowiedzialna za interfejs graficzny (JFrame). Obsługuje przyciski: dodaj, pobierz, pokaż klientów.
- KolejkaZamówień – logika kolejki FIFO, oparta na LinkedList. Metody: dodajZamowienie(), pobierzZamowienie(), isEmpty().
- Klient, Produkt, Zamówienie – klasy modelowe reprezentujące dane
- MenagerBazyDanych – obsługuje połączenie z MySQL oraz operacje: dodajZamowienie(), pobierzZamowienia().

W Projekcie użyto także Bazy danych MYSQL

## 2. Opis struktury projektu

Projekt został zaimplementowany w języku Java z wykorzystaniem bibliotek Swing i JDBC. Dane są przechowywane w relacyjnej bazie danych MySQL. W projekcie zastosowano 7 klas oraz plik mysql-connector-j-9.3.0.jar jest to sterownik, który umożliwia połączenie programu z bazą danych MYSQL.

### 2.1. Polecenie SQL:

```
CREATE DATABASE IF NOT EXISTS zamowienia_db;
USE zamowienia_db;

CREATE TABLE IF NOT EXISTS zamowienia (
  id INT AUTO_INCREMENT PRIMARY KEY,
  imie VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL
);

CREATE TABLE IF NOT EXISTS produkty (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nazwa VARCHAR(255) NOT NULL,
  cena DOUBLE NOT NULL,
  zamowienie_id INT,
  FOREIGN KEY (zamowienie_id) REFERENCES zamowienia(id) ON DELETE CASCADE
);
```

### 2.2. Zarządzanie danymi i baza danych:

- Tabela zamowienia: ID, imie, email

**Tabela 2.1.** Tabela Zamówienia (Baza danych)

Kolumna	Typ	Uwagi
id	INT	AUTO INCREMENT, PK
imie	VARCHAR(255)	Imię i nazwisko
e-mail	VARCHAR(255)	Adres e-mail

- Tabela produkty: ID, nazwa, cena, zamowienieid.

**Tabela 2.2.** Tabela produkty (Baza danych)

Kolumna	Typ	Uwagi
id	INT	AUTOINCREMENT, PK
nazwa	VARCHAR(255)	Nazwa produktu
cena	DOUBLE	Cena jednostkowa
zamowienie id	INT	FK -> zamowienia(id)

## 2.3. Dane Techniczne:

Minimalne wymagania sprzętowe:

- Procesor: 1.5 GHz,
- RAM: 2 GB
- Dysk: 100 MB wolnego miejsca
- Zainstalowana Java 8+
- System operacyjny: Windows 10+ lub Linux

Narzędzia:

- IntelliJ IDEA.
- MySQL Server
- JDBC
- Java SE
- Swing GUI Designer

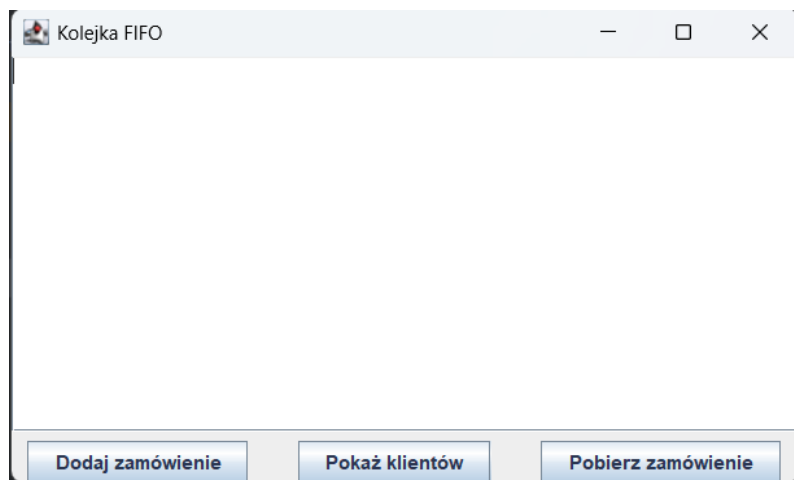
## 2.4. Działanie

1. Użytkownik uruchamia aplikację (Main -> GUI).
2. GUI pokazuje okno z przyciskami: "Dodaj", "Pobierz", "Pokaż klientów".
3. Użytkownik dodaje zamówienie:
  - Podaje dane klienta i listę produktów przez JOptionPane.
  - Zamówienie trafia do kolejki FIFO (KolejkaZamówień).
  - Dane są zapisywane do bazy danych (MenagerBazyDanych).
4. Użytkownik może zrealizować zamówienie z kolejki.
5. Może również podejrzeć wszystkie zapisane zamówienia z bazy.

### 3. Prezentacja elementów projektu:

#### 3.1. Prezentacja warstwy użytkowej

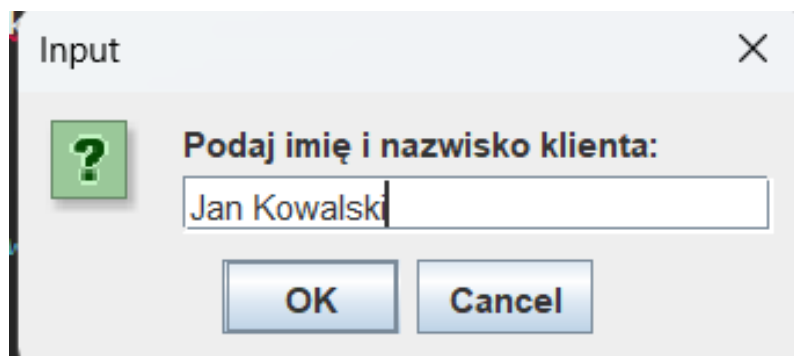
Główne okno aplikacji.



Rys. 3.1. Okno główne

Dodawanie zamówień:

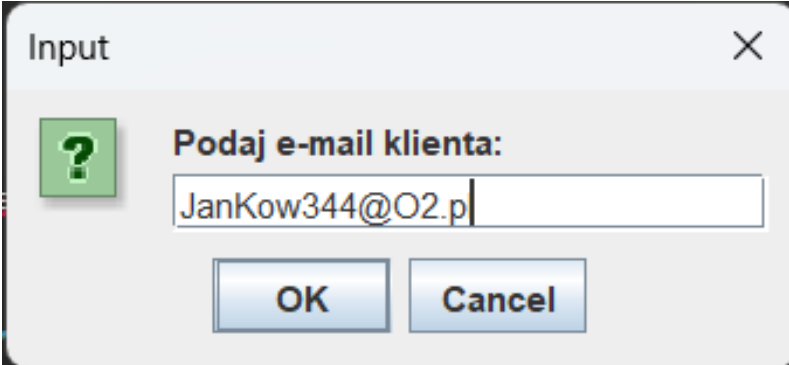
Aby dodać zamówienie należy, nacisnąć przycisk "Dodaj zamówienie". W następnym kroku podajemy imię i nazwisko



Rys. 3.2. Podawanie Imiona i nazwiska klienta

Gdy Klikniemy Ok, program poprosi o wpisanie adresu e-mail.

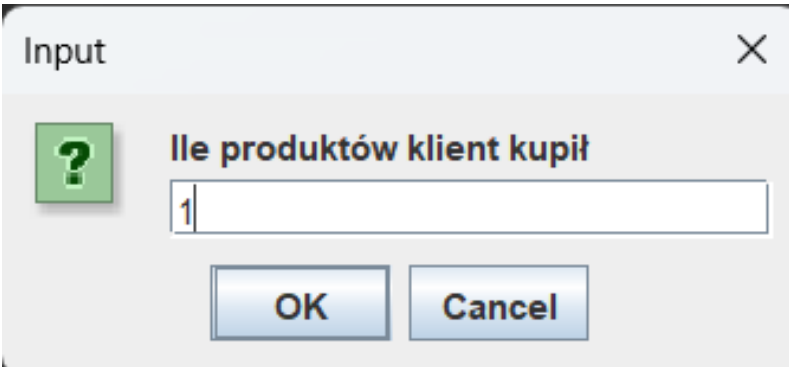




The screenshot shows a standard Windows-style dialog box with the title 'Input' and a close button (X) in the top right corner. On the left side, there is a green square icon containing a white question mark. To the right of the icon, the text 'Podaj e-mail klienta:' is displayed in a bold font. Below this text is a single-line text input field containing the email address 'JanKow344@O2.p'. At the bottom of the dialog, there are two buttons: 'OK' and 'Cancel'.

Rys. 3.3. Wpisywanie Adresu Email

Po wpisaniu e-maila i wciśnięciu ok, program nas poprosi o wpisanie ilości przedmiotów jaki dany klient kupił.



The screenshot shows a standard Windows-style dialog box with the title 'Input' and a close button (X) in the top right corner. On the left side, there is a green square icon containing a white question mark. To the right of the icon, the text 'Ile produktów klient kupił' is displayed in a bold font. Below this text is a single-line text input field containing the number '1'. At the bottom of the dialog, there are two buttons: 'OK' and 'Cancel'.

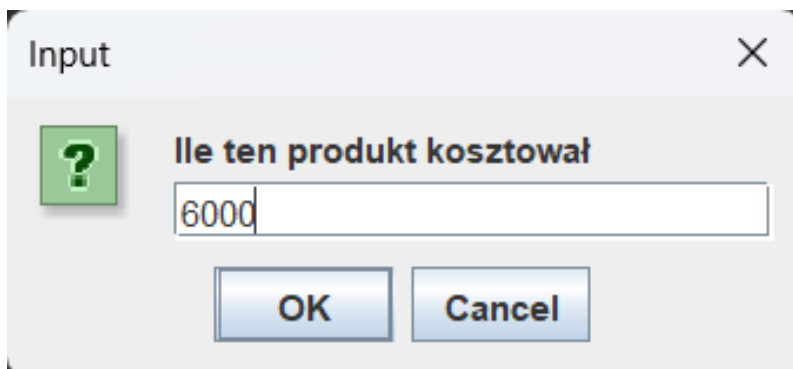
Rys. 3.4. Podawanie ilości przedmiotów

Następnie program poprosi o Nazwę produktu, oraz o jego cenę.



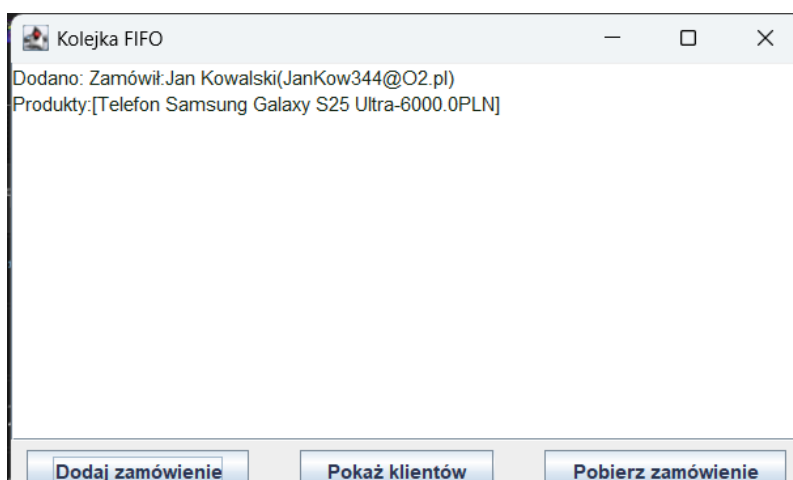
The screenshot shows a standard Windows-style dialog box with the title 'Input' and a close button (X) in the top right corner. On the left side, there is a green square icon containing a white question mark. To the right of the icon, the text 'Jaki produkt klient kupił' is displayed in a bold font. Below this text is a single-line text input field containing the text 'Telefon Samsung Galaxy S25 Ultra'. At the bottom of the dialog, there are two buttons: 'OK' and 'Cancel'.

Rys. 3.5. Podawanie nazwy produktu



Rys. 3.6. Podawanie ceny produktu

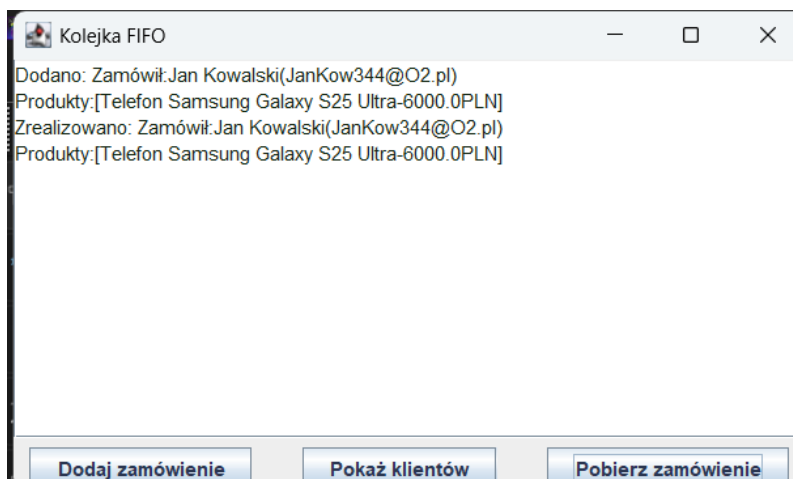
Po zakończeniu wszystkich powyższych kroków, program wypisze, dane zamówienia.



Rys. 3.7. Wyświetlanie danych dodanego sprzedawcy

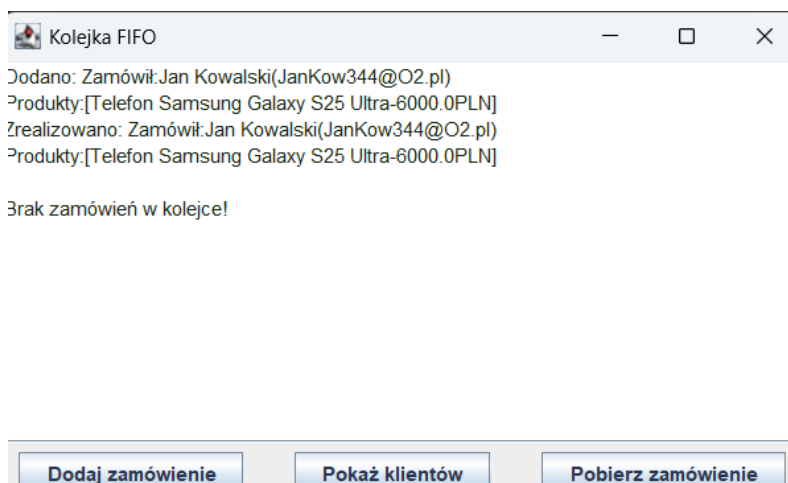
Przetwarzanie zamówień:

Aby przetworzyć zamówienie należy nacisnąć przycisk "Pobierz zamówienie". Po przetworzeniu zamówienia wyświetla się potwierdzenie jego realizacji.



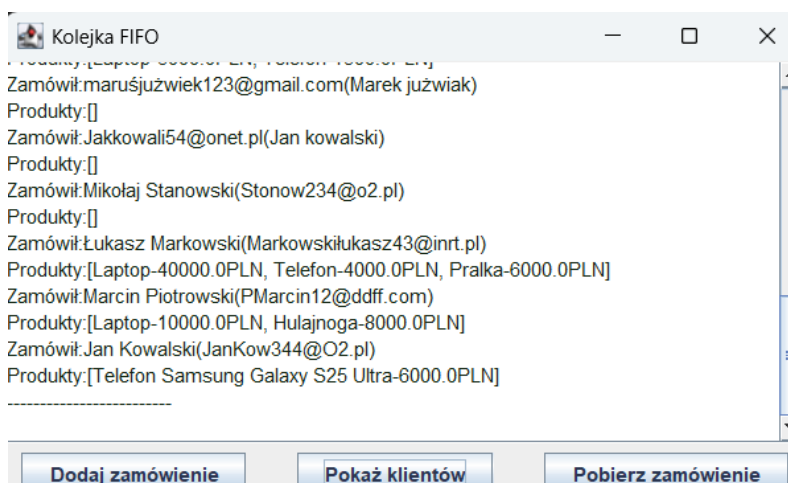
Rys. 3.8. Realizacja Zamówienia

W przypadku, próby przetworzenia zamówienia przy pustej kolejce, zostanie wyświetlany napis "Brak zamówień w kolejce!"



Rys. 3.9. Próba realizacji zamówienia, gdy kolejka jest pusta

Naciśnięcie przycisku "pokaż klientów" powoduje wyświetlenie, historii zamówień w bazie danych.



Rys. 3.10. Wyświetlanie zamówień z bazy danych

## 3.2. Bazy Danych MySQL

MySQL to popularny, relacyjny system zarządzania bazami danych, oparty na języku SQL. Jest dostępny jako oprogramowanie open source i rozwijany przez firmę Oracle Corporation. Poniżej znajduje się klasa w języku Java, która odpowiada za połączenie z tą bazą Danych. Klasa MenagerBazyDanych:

Listing 3.1. Klasa MenagerBazyDanych

```
1 import java.sql.Connection;
2 import java.sql.*;
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class MenagerBazyDanych {
7     private Connection połączenie;
```

```
8
9     public void DaneDoPołączenia() {
10         String url = "jdbc:mysql://localhost:3306/zamowienia_db?serverTimezone=UTC";
11         String użytkownik = "root";
12         String hasło = "Test111@";
13
14         try {
15             połączenie = DriverManager.getConnection(url, użytkownik, hasło);
16         } catch (SQLException e) {
17             e.printStackTrace();
18         }
19     }
20
21     public void dodajZamówienie(Zamówienie zamówienie) {
22         String sqlZamówienie = "INSERT INTO zamowienia (imie, email) VALUES (?, ?)";
23         String sqlProdukt = "INSERT INTO produkty (nazwa, cena, zamowienie_id) VALUES
24             (?, ?, ?)";
25
26         try {
27             połączenie.setAutoCommit(false);
28             int zamówienieId;
29             try (PreparedStatement pstmt = połączenie.prepareStatement(sqlZamówienie,
30                 Statement.RETURN_GENERATED_KEYS)) {
31                 pstmt.setString(1, zamówienie.getKlient().getImieinazwisko());
32                 pstmt.setString(2, zamówienie.getKlient().getEmail());
33                 pstmt.executeUpdate();
34
35                 ResultSet rs = pstmt.getGeneratedKeys();
36                 if (rs.next()) {
37                     zamówienieId = rs.getInt(1);
38                 } else {
39                     połączenie.rollback();
40                     return;
41                 }
42             }
43
44             try (PreparedStatement pstmt2 = połączenie.prepareStatement(sqlProdukt)) {
45                 for (Produkt p : zamówienie.getProdukty()) {
46                     pstmt2.setString(1, p.getNazwa());
47                     pstmt2.setDouble(2, p.getCena());
48                     pstmt2.setInt(3, zamówienieId);
49                     pstmt2.addBatch();
50                 }
51                 pstmt2.executeBatch();
52
53                 połączenie.commit();
54             } catch (SQLException e) {
55                 try {
56                     połączenie.rollback();
57                 } catch (SQLException ex) {
58                     ex.printStackTrace();
59                 }
60                 e.printStackTrace();
61             } finally {
62                 try {
63                     połączenie.setAutoCommit(true);
64                 }
65             }
66         }
67     }
68 }
```

```

63         } catch (SQLException e) {
64             e.printStackTrace();
65         }
66     }
67 }
68 public List<Zamówienie> pobierzZamówienia() {
69     List<Zamówienie> zamowienia = new ArrayList<>();
70     String sqlZamowienia = "SELECT * FROM zamowienia";
71     String sqlProdukty = "SELECT * FROM produkty WHERE zamowienie_id = ?";
72
73     try (Statement stmt = połączenie.createStatement();
74         ResultSet rs = stmt.executeQuery(sqlZamowienia)) {
75
76         while (rs.next()) {
77             int id = rs.getInt("id");
78             String imie = rs.getString("imie");
79             String email = rs.getString("email");
80             Klient klient = new Klient(email, imie);
81
82             List<Produkt> produkty = new ArrayList<>();
83             try (PreparedStatement pstmt = połączenie.prepareStatement(sqlProdukty))
84             {
85                 pstmt.setInt(1, id);
86                 ResultSet rsProdukty = pstmt.executeQuery();
87                 while (rsProdukty.next()) {
88                     String nazwa = rsProdukty.getString("nazwa");
89                     double cena = rsProdukty.getDouble("cena");
90                     produkty.add(new Produkt(nazwa, cena));
91                 }
92             }
93             zamowienia.add(new Zamówienie(klient, produkty));
94         }
95     } catch (SQLException e) {
96         e.printStackTrace();
97     }
98
99     return zamowienia;
100 }
101
102 }

```

### 3.3. Prezentacja kodu

Klasa Main (Główna trasa służąca do uruchomienia programu)

**Listing 3.2.** Klasa Main (główna)

```

1 //TIP To <b>Run</b> code, press <shortcut actionId="Run"/> or
2 // click the <icon src="AllIcons.Actions.Execute"/> icon in the gutter.
3 public class Main {
4     public static void main(String[] args) {
5         GUI SwingExample = new GUI();
6         SwingExample.setVisible(true);
7
8
9     }

```

```
10     }
```

Klasa GUI (Odpowiedzialna za graficzny interfejs użytkownika)

**Listing 3.3.** Klasa GUI (Odpowiedzialna za graficzny interfejs użytkownika)

```
1  import javax.swing.*;
2  import java.awt.event.ActionEvent;
3  import java.awt.event.ActionListener;
4  import java.util.ArrayList;
5  import java.util.List;
6  import java.util.Scanner;
7
8  public class GUI extends JFrame {
9      private JPanel panel1;
10     private JButton pobierzZamówienieButton;
11     private JButton dodajZamówienieButton;
12     private JTextArea textArea1;
13     private JButton pokazKlientówButton;
14     protected KolejkaZamówień kolejkaZamówień;
15     private MenagerBazyDanych db;
16
17
18     public GUI() {
19         super("Kolejka FIFO ");
20         this.setContentPane(this.panel1);
21         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22         setSize(500, 300);
23         this.kolejkaZamówień = new KolejkaZamówień();
24         this.db = new MenagerBazyDanych();
25         db.DaneDoPołączenia();
26
27         pobierzZamówienieButton.addActionListener(new ActionListener() {
28             @Override
29             public void actionPerformed(ActionEvent e) {
30                 pobierzZamówienie();
31             }
32         });
33
34         dodajZamówienieButton.addActionListener(new ActionListener() {
35             @Override
36             public void actionPerformed(ActionEvent e) {
37                 dodajZamówienie();
38             }
39         });
40
41         pokazKlientówButton.addActionListener(new ActionListener() {
42             @Override
43             public void actionPerformed(ActionEvent e) {
44                 pokazKlientów();
45             }
46         });
47     }
48
49     ;
50
51     public void dodajZamówienie() {
```

```

52     String imieinazwisko = JOptionPane.showInputDialog("Podaj imię i nazwisko
klienta:");
53     String email = JOptionPane.showInputDialog("Podaj e-mail klienta:");
54     Klient klient = new Klient(email, imieinazwisko);
55     int ilość = Integer.parseInt(JOptionPane.showInputDialog("Ile produktów klient
kupił"));
56     List<Produkt> produkty = new ArrayList<>();
57     for(int i = 0; i < ilość; i++) {
58
59         produkty.add(new Produkt(JOptionPane.showInputDialog("Jaki produkt klient
kupił"), Double.parseDouble(JOptionPane.showInputDialog("Ile ten produkt kosztował")
)));
60     }
61
62     Zamówienie zamówienie = new Zamówienie(klient, produkty);
63     kolejkaZamówień.dodajZamówienie(zamówienie);
64     textAreal.append("Dodano: " + zamówienie + "\n");
65     db.dodajZamówienie(zamówienie);
66 }
67
68 public void pobierzZamówienie() {
69     Zamówienie zamówienie = kolejkaZamówień.pobierzZamówienie();
70     if (zamówienie != null) {
71         textAreal.append("Zrealizowano: " + zamówienie + "\n");
72     } else {
73         textAreal.append("Brak zamówień w kolejce!\n");
74     }
75 }
76
77 public void pokażklientów() {
78     List<Zamówienie> zamowienia = db.pobierzZamówienia();
79     if (zamowienia.isEmpty()) {
80         textAreal.append("Brak zapisanych klientów w bazie.\n");
81     } else {
82         textAreal.append("Lista klientów z bazy:\n");
83         for (Zamówienie z : zamowienia) {
84             textAreal.append(z.toString() + "\n");
85         }
86         textAreal.append("-----\n");
87     }
88 }
89 }

```

Klasa Klient(Klasa służąca do przechowywania danych klienta)

**Listing 3.4.** Klasa Klient(Klasa służąca do przechowywania danych klienta)

```

1 public class Klient {
2     private String imieinazwisko;
3     private String email;
4
5     public Klient(String email, String imieinazwisko) {
6         this.email = email;
7         this.imieinazwisko = imieinazwisko;
8     }
9
10    public String getImieinazwisko() {
11        return imieinazwisko;

```

```
12     }
13
14     public String getEmail() {
15         return email;
16     }
17
18     @Override
19     public String toString() {
20         return imieinazwisko + "(" + email + ")";
21     }
22 }
```

Klasa Produkt(Klasa służąca do przechowywania informacji o produkcie)

**Listing 3.5.** Klasa Produkt(Klasa służąca do przechowywania informacji o produkcie)

```
1 public class Produkt {
2     private String nazwa;
3     private double cena;
4
5     public Produkt(String nazwa, double cena) {
6         this.nazwa = nazwa;
7         this.cena = cena;
8     }
9
10    public String getNazwa() {
11        return nazwa;
12    }
13
14    public double getCena() {
15        return cena;
16    }
17
18    @Override
19    public String toString() {
20        return nazwa + "-" + cena + "PLN";
21    }
22 }
```

Klasa Zamówienie

**Listing 3.6.** Klasa Zamówienia

```
1 import java.util.List;
2
3 public class Zamówienie
4 {
5     private Klient klient;
6     private List<Produkt> produkty;
7
8     public Zamówienie(Klient klient, List<Produkt> produkty) {
9         this.klient = klient;
10        this.produkty = produkty;
11    }
12
13    public Klient getKlient() {
14        return klient;
15    }
16 }
```



```
17     public List<Produkt> getProdukty() {
18         return produkty;
19     }
20     @Override
21     public String toString(){
22         return "Zamówił:" + klient+ "\nProdukty:" + produkty;
23     }
24 }
```

### Klasa KolejkaZamówień

**Listing 3.7.** Klasa KolejkaZamówień

```
1 import java.util.LinkedList;
2
3 public class KolejkaZamówień {
4     private LinkedList<Zamówienie> kolejka = new LinkedList<>();
5     public void dodajZamówienie(Zamówienie zamówienie){
6         kolejka.addLast(zamówienie);
7     }
8     public Zamówienie pobierzZamówienie(){
9         return kolejka.pollFirst();
10    }
11    public boolean isEmpty(){
12
13        return kolejka.isEmpty();
14    }
15
16
17 }
```

## 4. Podsumowanie

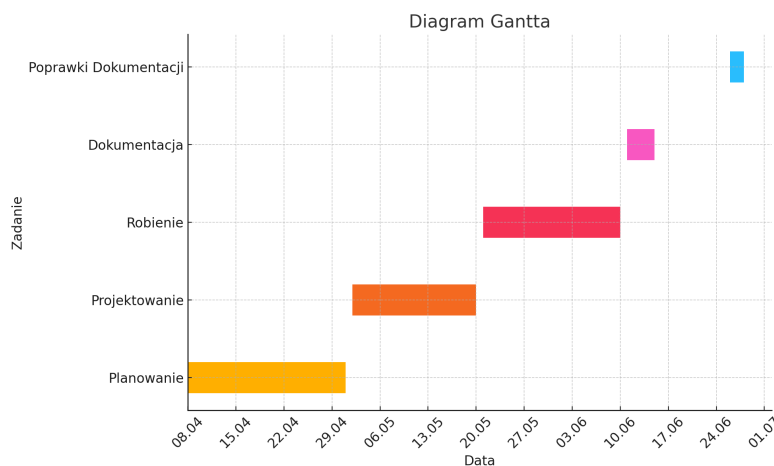
### 4.1. Podsumowanie ogólne

Projekt zakładał stworzenie aplikacji desktopowej wspomagającej zarządzanie zamówieniami w kolejce FIFO. Osiągnięto wszystkie założone cele, w tym poprawne działanie kolejki, zapis i odczyt danych z bazy oraz prosty graficzny interfejs.

Możliwości rozwoju:

- logowanie użytkowników
- generowanie faktur PDF
- Rozdzielenie bazy danych, przechowującej klientów, produkty oraz Zamówienia.
- Pokazywanie ilości produktów w magazynie.
- Integrację z systemem wysyłek.
- W przypadku braku produktu, na półkach, uniemożliwienie jego zakupu.

### 4.2. Harmonogram realizacji projektu



Rys. 4.1. Harmonogram realizacji projektu

### 4.3. Wnioski

Projekt udowadnia, że przy użyciu podstawowych narzędzi Javy można stworzyć w pełni działający system do zarządzania zamówieniami, z czytelny GUI oraz trwałym przechowywaniem danych. Kod jest modularny, co ułatwia jego dalszy rozwój i integrację z innymi systemami. Istnieje jednak kilka opcji rozwoju programu w kierunku bardziej zaawansowanego narzędzia do zarządzania w bardziej profesjonalny sposób zakupami, stanem magazynu m.in: Przez dodanie stanów magazynów, możliwość logowania, zarówno pracowników jak i klientów.

## 4.4. Oświadczenie studenta o samodzielności pracy

Oświadczenie należy wydrukować, podpisać, zeskanować i umieścić jako załącznik do niniejszej dokumentacji.

Załącznik nr 2 do Zarządzenia nr 228/2021 Rektora Uniwersytetu Rzeszowskiego z dnia 1 grudnia 2021 roku w sprawie ustalenia procedury antyplagiatowej w Uniwersytecie Rzeszowskim

**OŚWIADCZENIE STUDENTA O SAMODZIELNOŚCI PRACY**

.....  
Radosław Kierepka  
Imię (imiona) i nazwisko studenta

.....  
Wydział Nauk Ścisłych i Technicznych

.....  
Informatyka  
Nazwa kierunku

.....  
134921  
Numer albumu

1. Oświadczam, że moja praca projektowa pt.: Dokumentacja do projektu "Kolejka FIFO"

1) została przygotowana przez mnie samodzielnie\*,

2) nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2021 r., poz. 1062) oraz dóbr osobistych chronionych prawem cywilnym,

3) nie zawiera danych i informacji, które uzyskałem/am w sposób niedozwolony,

4) nie była podstawą otrzymania oceny z innego przedmiotu na uczelni wyższej ani mnie, ani innej osobie.

2. Jednocześnie wyrażam zgodę/~~nie wyrażam zgody~~\*\* na udostępnienie mojej pracy projektowej do celów naukowo-badawczych z poszanowaniem przepisów ustawy o prawie autorskim i prawach pokrewnych.

15.06.2025r. \_\_\_\_\_  
(miejscowość, data)

\_\_\_\_\_ Radosław Kierepka  
(czytelny podpis studenta)

\* Uwzględniając merytoryczny wkład prowadzącego przedmiot  
\*\* – niepotrzebne skreślić

Rys. 4.2. Oświadczenie studenta o samodzielności pracy

## **Bibliografia**

# Spis rysunków

3.1	Okno główne . . . . .	10
3.2	Podawanie Imiona i nazwiska klienta . . . . .	10
3.3	Wpisywanie Adresu Email . . . . .	11
3.4	Podawanie ilości przedmiotów . . . . .	11
3.5	Podawanie nazwy produktu . . . . .	11
3.6	Podawanie ceny produktu . . . . .	12
3.7	Wyświetlanie danych dodanego sprzedawcy . . . . .	12
3.8	Realizacja Zamówienia . . . . .	12
3.9	Próba realizacji zamówienia, gdy kolejka jest pusta . . . . .	13
3.10	Wyświetlanie zamówień z bazy danych . . . . .	13
4.1	Harmonogram realizacji projektu . . . . .	20
4.2	Oświadczenie studenta o samodzielności pracy . . . . .	21

# Spis tabel

2.1	Tabela Zamówienia (Baza danych) . . . . .	8
2.2	Tabela produkty (Baza danych) . . . . .	8

## Spis listingów

3.1	Klasa MenagerBazyDanych . . . . .	13
3.2	Klasa Main (główna) . . . . .	15
3.3	Klasa GUI (Odpowiedzialna za graficzny interfejs użytkownika) . . . . .	16
3.4	Klasa Klient(Klasa służąca do przechowywania danych klienta) . . . . .	17
3.5	Klasa Produkt(Klasa służąca do przechowywania informacji o produkcie) . . . . .	18
3.6	Klasa Zamówienia . . . . .	18
3.7	Klasa KolejkaZamówień . . . . .	19