



Uniwersytet Rzeszowski

Instytut Informatyki

Wydział Nauk Ścisłych i Technicznych

System automat z napojami

Praca Projektowa

Prowadzący: mgr. inż. Ewa Żesławska

Autor: **Bartosz Betlej**
Numer indeksu: **137132**
Data: **11.06.2025**

Rok akademicki: 2025/2026

Contents

1	Wprowadzenie	2
1.1	Cele pracy	2
1.2	Zakres funkcjonalny	2
1.3	Zastosowane technologie	3
1.4	Główne klasy projektu	3
2	Opis struktury projektu	4
2.1	Klasa Drink	4
2.2	Klasa VendingMachine	6
2.3	Klasa Transaction	8
2.4	Klasa TransactionManager	9
2.5	Klasa ConsoleUI	10
2.6	Klasa InsufficientFundsException	12
2.7	Klasa Main	13
2.8	Predefiniowana lista napojów	13
3	Interakcja z użytkownikiem	14
3.1	Struktura menu	14
3.2	Pętla główna programu	14
3.3	Obsługa danych wejściowych	15
3.4	Wygląd przykładowej sesji	16
4	Harmonogram realizacji projektu	16
5	Bibliografia	16
6	GitHub	16
7	Spis listningów	17
8	Spis rysunków	17
	Oświadczenie studenta o samodzielności pracy	18

1 Wprowadzenie

Celem projektu inżynierskiego było stworzenie aplikacji, którą będzie symulować działanie automatu z napojami. Program powstał w oparciu o zasady programowania obiektowego

Użytkownik może wpłacić pieniądze oraz zakupić produkt. Zaimplementowano również panel administracyjny, który umożliwia zarządzanie asortymentem automatu oraz możliwość dostępu do podglądu rejestru wszystkich dokonanych transakcji

1.1 Cele pracy

- Opracowanie i implementacja konsolowego symulatora automatu z napojami.
- Stworzenie funkcjonalności pozwalających użytkownikowi na deponowanie środków, wybór napoju oraz jego zakup.
- Zastosowanie programowania obiektowego – m.in. podziału odpowiedzialności pomiędzy klasami, hermetyzacji i obsługi wyjątków.
- Wykorzystanie odpowiednich struktur danych (np. listy, mapy) do przechowywania napojów i transakcji..
- Zapewnienie prostego i intuicyjnego interfejsu tekstowego dla użytkownika i administratora.

1.2 Zakres funkcjonalny

Aplikacja oferuje następujące funkcjonalności:

- Wybór trybu pracy: panel użytkownika lub panel administratora.
- Wyświetlanie listy dostępnych napojów wraz z ich cenami i ilością.
- Deponowanie środków, wybór i zakup napoju oraz wyświetlanie informacji o reszcie.
- Panel administratora, który dostępny jest po wpisaniu odpowiedniego hasła (admin123).
- Panel administratora dostępny po podaniu hasła (admin123), umożliwiający dodanie nowych napojów, usuwanie istniejących napojów oraz przegląd listy przeprowadzonych transakcji.

1.3 Zastosowane technologie

- **Język programowania:** Java 17
- **Środowisko programistyczne:** IntelliJ IDEA 2024.3.4.1
- **Typ aplikacji:** Konsolowa
- **Zastosowane biblioteki i mechanizmy:**
 - **Klasy:** Drink, VendingMachine, ConsoleUI, Transaction, InsufficientFundsException, TransactionManager.
 - **Hermetyzacja:** Prywatne pola oraz metody dostępne (getter, setter) – np. `Drink.getName()`, `setPrice()`.
 - **Dziedziczenie:** Własny wyjątek `InsufficientFundsException`, dziedziczący po `RuntimeException`.
 - **Polimorfizm:** Nadpisanie metody `toString()` – np. w klasie `Drink` i `Transaction`.
 - **Kompozycja:** Klasa `VendingMachine` zawiera listę napojów (`ArrayList<Drink>`).
 - **Obsługa błędów:** Bloki try-catch stosowane do zabezpieczenia przed błędami wejścia i niepoprawnymi danymi.
 - **CRUD:** Dodawanie, usuwanie i wyświetlanie napojów; zapisywanie i przeglądanie transakcji w czasie działania programu.
 - **Pętle i warunki:** `while`, `for`, `if`, `switch` – stosowane w logice menu i zakupu.
 - **Przetwarzanie tekstu:** `String.trim()`, `equalsIgnoreCase()` – porównywanie nazw napojów.
 - **Statyczne metody i pola:** Wykorzystane np. w klasie `TransactionManager`.
 - **Formatowanie danych:** `System.out.printf()` oraz `toString()` – do czytelnego wyświetlania informacji w konsoli.
 - **Biblioteki standardowe Javy:** `java.util.ArrayList`, `java.util.Scanner`, `java.util.HashMap`, `java.lang.RuntimeException`. `replace()`

1.4 Główne klasy projektu

- **Main** – klasa uruchamiająca aplikację.
- **ConsoleUI** – obsługuje interakcję użytkownika z programem poprzez menu tekstowe.
- **VendingMachine** – odpowiada za logikę działania automatu: dodawanie napojów, obsługę sprzedaży, kontrolę stanów magazynowych.
- **Drink** – reprezentuje pojedynczy napój dostępny w automacie.
- **Transaction** - reprezentuje zapis pojedynczej transakcji zakupu.
- **TransactionManager** - zarządza historią transakcji (przechowywaną w pamięci).
- **InsufficientFundsException** - własny wyjątek zgłaszany w przypadku braku wystarczających środków do zakupu.

2 Opis struktury projektu

Projekt został opracowany w oparciu o paradygmat obiektowy. Wszystkie klasy umieszczono w jednej przestrzeni nazw, z zachowaniem logicznego podziału według pełnionych funkcji. Taka organizacja kodu ułatwia jego utrzymanie, rozwój oraz wprowadzanie zmian w przyszłości.

2.1 Klasa Drink

Reprezentuje pojedynczy napój dostępny w automacie. Przechowuje informacje o nazwie, cenie oraz liczbie dostępnych sztuk.

- Wszystkie pola są prywatne (hermetyzacja danych).
- Klasa posiada konstruktor z parametrami umożliwiający utworzenie nowego napoju.
- Dostęp do pól odbywa się przez metody getter i setter (`getName()`, `setPrice()` itd.).
- W metodach setter zastosowano prostą walidację (np. `cena > 0`).
- Metoda `toString()` została przesłonięta, aby umożliwić czytelne wyświetlanie informacji o napoju.

Listing 1 – Klasa Drink:

```
1
2 package model;
3
4 // Klasa reprezentujaca napoj
5 public class Drink {
6     // Nazwa napoju
7     private String name;
8
9     // Cena napoju
10    private double price;
11
12    // Ilosc dostepnych sztuk napoju
13    private int quantity;
14
15    // Konstruktor przyjmujacy wszystkie pola jako argumenty
16    public Drink(String name, double price, int quantity) {
17        this.name = name;
18        this.price = price;
19        this.quantity = quantity;
20    }
21
22    // Konstruktor przyjmujacy tylko nazwe; cena i ilosc domyslnie ustawione na 0
23    public Drink(String name) {
24        this(name, (double)0.0F, 0);
25    }
26
27    // Zwraca nazwe napoju
28    public String getName() {
29        return this.name;
30    }
31
32    // Zwraca cene napoju
33    public double getPrice() {
34        return this.price;
35    }
36
37    // Zwraca ilosc napoju
38    public int getQuantity() {
39        return this.quantity;
40    }
41
42    // Ustawia nowa cene napoju
43    public void setPrice(double price) {
44        this.price = price;
45    }
46
47    // Ustawia nowa ilosc napoju
48    public void setQuantity(int quantity) {
49        this.quantity = quantity;
50    }
51
52    // Zmniejsza ilosc napoju o 1, jesli ilosc jest wieksza od zera
53    public void decreaseQuantity() {
54        if (this.quantity > 0) {
55            --this.quantity;
56        }
57    }
58
59    // Zwraca reprezentacje napoju jako tekst
60    public String toString() {
61        return this.name + " | Cena: " + this.price + " PLN | Ilo : " + this.
        quantity;
62    }
63 }
```

Listing 1: Klasa Drink

2.2 Klasa VendingMachine

Odpowiada za logikę działania automatu z napojami – m.in. sprzedaż, aktualizację stanu magazynowego, zarządzanie listą napojów.

- Przechowuje listę obiektów typu Drink (ArrayList<Drink>).
- Umożliwia dodawanie i usuwanie napojów.
- Obsługuje zakup napoju, sprawdzając dostępność i środki użytkownika.
- Metody zwracają informacje potrzebne do interakcji z interfejsem użytkownika.
- Hermetyzacja danych – lista napojów modyfikowana jest wyłącznie poprzez metody klasy.

Listing 2 – Klasa VendingMachine:

```
1 package model;
2
3 import exception.InsufficientFundsException;
4 import service.TransactionManager;
5
6 import java.util.List;
7
8 // Klasa reprezentująca automat sprzedający
9 public class VendingMachine {
10     // Lista dostępnych napojów w automacie
11     private List<Drink> drinks;
12
13     // Obiekt odpowiedzialny za zarządzanie transakcjami
14     private TransactionManager transactionManager;
15
16     // Konstruktor ustawiający listę napojów oraz menedżera transakcji
17     public VendingMachine(List<Drink> drinks, TransactionManager transactionManager)
18     {
19         this.drinks = drinks;
20         this.transactionManager = transactionManager;
21     }
22
23     // Zwraca listę wszystkich napojów w automacie
24     public List<Drink> getDrinks() {
25         return drinks;
26     }
27
28     // Dodaje nowy napój do automatu
29     public void addDrink(Drink drink) {
30         drinks.add(drink);
31     }
32
33     // Usuwa napój z automatu na podstawie jego nazwy (ignorując wielkość liter)
34     public void removeDrink(String name) {
35         drinks.removeIf(d -> d.getName().equalsIgnoreCase(name));
36     }
37
38     // Realizuje zakup napoju
39     public void buyDrink(String name, double insertedAmount) throws
40     InsufficientFundsException {
41         for (Drink d : drinks) {
42             // Sprawdza czy nazwa napoju zgadza się (ignorując wielkość liter i
43             // spacje)
44             if (d.getName().trim().equalsIgnoreCase(name.trim())) {
45                 // Sprawdza czy napój jest dostępny
46                 if (d.getQuantity() <= 0) {
```

```

44         throw new InsufficientFundsException("Brak produktu na stanie.")
45         ;
46     }
47     // Sprawdza czy uzytkownik wplacil wystarczajaca ilosc pieniedzy
48     if (insertedAmount < d.getPrice()) {
49         throw new InsufficientFundsException("Za malo pieniedzy! Napoj
50         kosztuje: " + d.getPrice() + " PLN");
51     }
52     // Zmniejsza ilosc napoju po zakupie
53     d.decreaseQuantity();
54     // Dodaje transakcje do historii
55     transactionManager.addTransaction(new model.Transaction(name, d.
56     getPrice()));
57     // Wyświetla reszta do wydania
58     System.out.println("Wydano reszta: " + (insertedAmount - d.getPrice
59     ()) + " PLN");
60     return;
61 }
62 // Rzuca wyjątek, jeśli napój o podanej nazwie nie został znaleziony
63 throw new InsufficientFundsException("Nie znaleziono napoju o podanej nazwie
64 .");
65 }
66 }

```

Listing 2: Klasa VendingMachine

2.3 Klasa Transaction

Modeluje pojedynczą transakcję zakupu napoju. Zawiera dane o nazwie napoju, dacie transakcji i zapłaconej kwocie.

- Konstruktor z parametrami tworzy nowy zapis transakcji.
- Przechowuje dane jako prywatne pola.
- Udostępnia metody typu getter do odczytu danych.
- Metoda `toString()` została przesłonięta w celu formatowania informacji o transakcji.

Listing 3 – Klasa Transaction :

```
1 package model;
2
3 import java.time.LocalDateTime;
4
5 public class Transaction {
6     private String drinkName;
7     private double amount;
8     private LocalDateTime dateTime;
9
10    public Transaction(String drinkName, double amount) {
11        this.drinkName = drinkName;
12        this.amount = amount;
13        this.dateTime = LocalDateTime.now();
14    }
15
16    @Override
17    public String toString() {
18        return dateTime + " - " + drinkName + ": " + amount + " PLN";
19    }
20 }
```

Listing 3: Klasa Transaction

2.4 Klasa TransactionManager

Zarządza historią transakcji zapisanych podczas działania programu.

- Przechowuje listę transakcji (ArrayList<Transaction>).
- Umożliwia dodawanie nowej transakcji oraz wyświetlanie historii.
- Metody pozwalają na filtrowanie i przeszukiwanie transakcji według kryteriów.
- Klasa nie zapisuje danych do pliku – historia istnieje tylko w czasie działania programu.

Listing 4 – Klasa TransactionManager:

```
1
2 package service;
3
4 import model.Transaction;
5
6 import java.util.ArrayList;
7 import java.util.List;
8
9 // Klasa odpowiedzialna za zarządzanie lista transakcji
10 public class TransactionManager {
11     // Lista przechowująca wszystkie transakcje
12     private List<Transaction> transactions = new ArrayList<>();
13
14     // Dodaje nowa transakcje do listy
15     public void addTransaction(Transaction t) {
16         transactions.add(t);
17     }
18
19     // Zwraca liste wszystkich transakcji
20     public List<Transaction> getTransactions() {
21         return transactions;
22     }
23 }
```

Listing 4: Klasa Transaction manager

2.5 Klasa ConsoleUI

Obsługuje interfejs tekstowy użytkownika oraz administratora. Odpowiada za interakcję z użytkownikiem – wprowadzanie danych i wyświetlanie wyników.

- Zawiera metody wyświetlające menu główne, menu użytkownika i menu administratora.
- Wykorzystuje klasę Scanner do pobierania danych z wejścia.
- Obsługuje logikę sterującą – np. przekierowanie do odpowiedniego modułu, odczyt danych.
- Korzysta z metod klas VendingMachine i TransactionManager do wykonywania operacji..

Listing 5 – ConsoleUI:

```
1 package ui;
2
3 import exception.InsufficientFundsException;
4 import model.Drink;
5 import model.VendingMachine;
6 import service.DrinkManager;
7 import service.FileManager;
8 import service.TransactionManager;
9
10 import java.util.List;
11 import java.util.Scanner;
12
13 // Klasa odpowiedzialna za interfejs konsolowy aplikacji
14 public class ConsoleUI {
15     // Obiekt do odczytu danych od uzytkownika
16     private Scanner scanner = new Scanner(System.in);
17
18     // Zarzadza zapisem i odczytem danych z/do pliku
19     private FileManager fileManager = new FileManager();
20
21     // Zarzadza lista transakcji
22     private TransactionManager transactionManager = new TransactionManager();
23
24     // Lista napojow wczytana z pliku
25     private List<Drink> drinks = fileManager.readDrinks();
26
27     // Automat sprzedajacy z lista napojow i menedzerem transakcji
28     private VendingMachine vendingMachine = new VendingMachine(drinks,
29         transactionManager);
30
31     // Obiekt pomocniczy do zarzadzania napojami (obecnie nieuzywany w tej klasie)
32     private DrinkManager drinkManager = new DrinkManager();
33
34     // Glowna metoda uruchamiajaca interfejs
35     public void start() {
36         System.out.println("Witaj w automacie z napojami!");
37         while (true) {
38             System.out.println("\n1. Uzytkownik\n2. Administrator\n3. Wyjście");
39             switch (scanner.nextLine()) {
40                 case "1" -> customerMenu(); // Przejdź do menu klienta
41                 case "2" -> adminMenu();     // Przejdź do menu administratora
42                 case "3" -> {
43                     fileManager.saveDrinks(drinks); // Zapisz dane przed wyjściem
44                     return;
45                 }
46                 default -> System.out.println("Nieznana opcja.");
47             }
48         }
49     }
50 }
```

```

49
50 // Menu klienta - zakup napojow
51 private void customerMenu() {
52     while (true) {
53         System.out.println("\n--- Menu klienta ---");
54
55         List<Drink> availableDrinks = vendingMachine.getDrinks();
56
57         // Wyświetl liste dostępnych napojow
58         for (int i = 0; i < availableDrinks.size(); i++) {
59             System.out.println((i + 1) + ". " + availableDrinks.get(i));
60         }
61
62         System.out.println("Wybierz numer napoju lub wpisz X, aby wrocic:");
63         String choice = scanner.nextLine();
64
65         if (choice.equalsIgnoreCase("X")) break;
66
67         try {
68             int index = Integer.parseInt(choice) - 1;
69
70             // Sprawdź czy numer napoju jest poprawny
71             if (index < 0 || index >= availableDrinks.size()) {
72                 System.out.println("Nieprawidłowy numer.");
73                 continue;
74             }
75
76             Drink selectedDrink = availableDrinks.get(index);
77
78             System.out.print("Wrzuc monety (PLN): ");
79             double amount = Double.parseDouble(scanner.nextLine());
80
81             // Proba zakupu napoju
82             vendingMachine.buyDrink(selectedDrink.getName(), amount);
83             System.out.println("Dziękujemy za zakup!");
84
85             } catch (NumberFormatException e) {
86                 System.out.println("Błąd: Wpisz numer lub X.");
87             } catch (InsufficientFundsException e) {
88                 System.out.println("Błąd: " + e.getMessage());
89             }
90         }
91     }
92
93 // Menu administratora - zarządzanie zawartością automatu
94 private void adminMenu() {
95     System.out.print("Podaj hasło admina: ");
96     if (!scanner.nextLine().equals("admin123")) {
97         System.out.println("Niepoprawne hasło.");
98         return;
99     }
100
101     while (true) {
102         System.out.println("\n--- Menu administratora ---");
103         System.out.println("1. Dodaj napoj\2. Usun napoj\3. Lista transakcji\
104             n4. Powrot");
105         switch (scanner.nextLine()) {
106             case "1" -> {
107                 // Dodanie nowego napoju
108                 System.out.print("Nazwa: ");
109                 String name = scanner.nextLine();
110                 System.out.print("Cena: ");
111                 double price = Double.parseDouble(scanner.nextLine());

```

```

111         System.out.print("Ilość: ");
112         int qty = Integer.parseInt(scanner.nextLine());
113         vendingMachine.addDrink(new Drink(name, price, qty));
114         System.out.println("Napój dodany.");
115     }
116     case "2" -> {
117         // Usuniecie napoju
118         System.out.print("Podaj nazwę napoju: ");
119         vendingMachine.removeDrink(scanner.nextLine());
120         System.out.println("Napój usunięty.");
121     }
122     case "3" -> {
123         // Wyświetlenie listy transakcji
124         transactionManager.getTransactions().forEach(System.out::println);
125     }
126     case "4" -> {
127         // Powrót do menu głównego
128         return;
129     }
130     default -> System.out.println("Nieznana opcja.");
131 }
132 }
133 }
134 }

```

Listing 5: Klasa ConsoleUI

2.6 Klasa InsufficientFundsException

Własny wyjątek zgłaszany w przypadku próby zakupu napoju bez wystarczającej ilości środków.

- Dziedziczy po klasie RuntimeException.
- Umożliwia czytelniejsze zarządzanie błędami związanymi z brakiem funduszy.
- Może zawierać spersonalizowaną wiadomość błędu przekazywaną do użytkownika.

Listing 6 – Klasa `InsufficientFundsException`:

```
1 package exception;
2
3 // Klasa reprezentujaca wyjatek rzucany, gdy brakuje srodkow lub napoju
4 public class InsufficientFundsException extends Exception {
5
6     // Konstruktor przekazujacy wiadomosc do klasy bazowej Exception
7     public InsufficientFundsException(String message) {
8         super(message);
9     }
10 }
```

Listing 6: Klasa `InsufficientFundsException`

2.7 Klasa `Main`

Punkt wejścia do aplikacji. Inicjalizuje obiekty `ConsoleUI`, `VendingMachine`, `TransactionManager` i rozpoczyna działanie programu.

- Zawiera metodę `main(String[] args)`.
- Odpowiada za uruchomienie logiki aplikacji oraz pierwsze wyświetlenie menu.

Listing 7 – Klasa `Main`:

```
1 // Główna klasa uruchamiająca aplikację
2 public class Main {
3     public static void main(String[] args) {
4         // Tworzy obiekt interfejsu konsolowego
5         ui.ConsoleUI ui = new ui.ConsoleUI();
6
7         // Uruchamia interfejs
8         ui.start();
9     }
10 }
```

Listing 7: Klasa `Main`

2.8 Predefiniowana lista napojów

Na początku działania aplikacji ładowana jest domyślna lista napojów, która symuluje początkowy asortyment automatu. Lista ta znajduje się w klasie `VendingMachine`.

Listing 7 – Inicjalizacja domyślnych napojów:

```
1 // Dodanie trzech napojow do listy z domyslna nazwa, cena i iloscia
2 drinks.add(new Drink("Cola", 5.00, 10)); // Nazwa: Cola, Cena: 5.00 zł, Ilość: 10
   sztuk
3 drinks.add(new Drink("Woda", 3.00, 15)); // Nazwa: Woda, Cena: 3.00 zł, Ilość: 15
   sztuk
4 drinks.add(new Drink("Sok", 4.50, 8));   // Nazwa: Sok, Cena: 4.50 zł, Ilość: 8
   sztuk
```

Listing 8: Inicjalizacja domyślnych napojów

3 Interakcja z użytkownikiem

plikacja działa w trybie konsolowym z wykorzystaniem prostego interfejsu tekstowego, który prowadzi użytkownika krok po kroku przez dostępne operacje.

Klasa ConsoleUI odpowiada za:

- prezentację menu użytkownika i administratora,
- odbieranie danych wejściowych od użytkownika,
- obsługę wyboru i przekierowanie do odpowiednich metod systemowych,
- walidację i wyświetlanie komunikatów informacyjnych lub błędów.

3.1 Struktura menu

Menu główne aplikacji umożliwia wybór jednej z dwóch ścieżek:

1. **Panel użytkownika** – zakup napojów
2. **Panel administratora** – zarządzanie automatami (hasło: admin123)

Opcje dostępne dla użytkownika:

1. Wyświetl dostępne napoje
2. Wpłać środki
3. Wybierz napój i dokonaj zakupu
4. Cofnij do menu głównego

Opcje dostępne dla administratora:

1. Wyświetl dostępne napoje
2. Dodaj nowy napój
3. Usuń napój
4. Wyświetl historię transakcji
5. Powrót do menu głównego

3.2 Pętla główna programu

Cała logika aplikacji znajduje się w pętli, która działa aż do wybrania opcji wyjścia. Dzięki temu użytkownik może wykonywać wiele operacji bez konieczności restartu programu.

```
1 while (true) {  
2     displayMainMenu();           // Wyświetlenie glownego menu  
3     int choice = scanner.nextInt(); // Odczyt wyboru uzytkownika  
4     handleChoice(choice);       // Przekazanie sterowania do odpowiedniej funkcji  
5 }
```

Listing 9: Pętla główna aplikacji:

3.3 Obsługa danych wejściowych

Dane wejściowe od użytkownika przetwarzane są przy użyciu klasy Scanner. Zaimplementowano podstawową walidację:

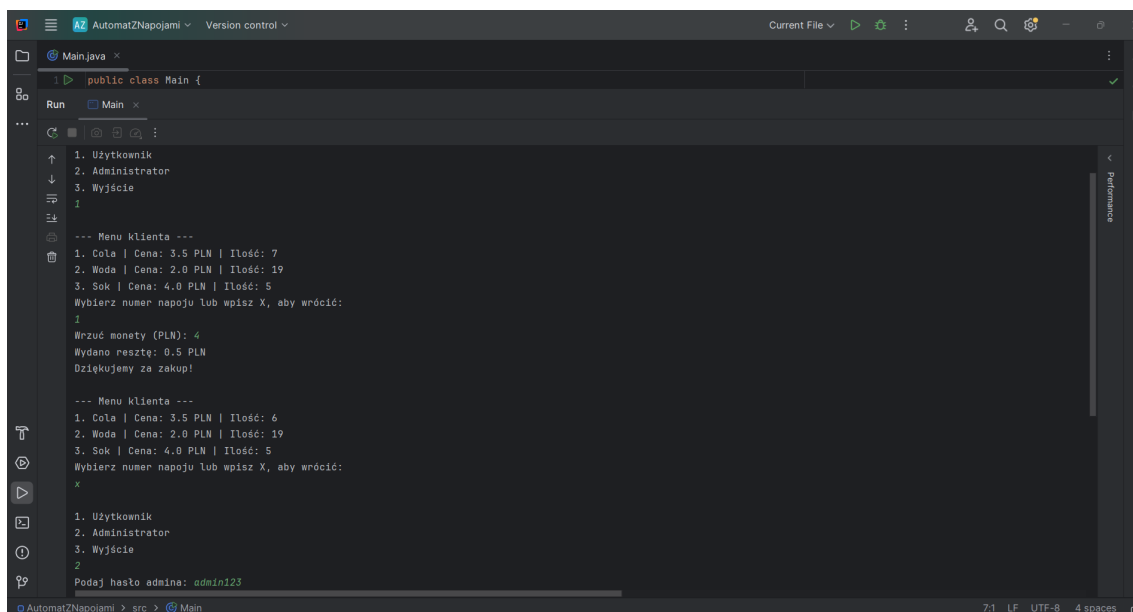
- Sprawdzanie, czy dane są niepuste,
- Obsługa wyjątków (np. InputMismatchException),
- Obsługa błędnych danych i ponowienie próby,
- Automatyczne oczyszczanie bufora wejściowego w przypadku błędu.

```
1 try {
2     System.out.print("Podaaj kwote: ");
3     double amount = scanner.nextDouble();    // Proba odczytu liczby
        zmiennoprzecinkowej
4     if (amount <= 0) {
5         System.out.println("Kwota musi byc wieksza od zera.");
6     } else {
7         vendingMachine.depositMoney(amount);    // Dodanie srodk w do salda
            u ytkownika
8     }
9 } catch (InputMismatchException e) {
10     System.out.println("Bledny format liczby. Spróbuj ponownie.");
11     scanner.nextLine();    // Oczyszczenie bufora
12 }
```

Listing 10: Przykład walidacji wejścia:

3.4 Wygląd przykładowej sesji

Oto przykładowy sposób działania programu:



```
public class Main {  
    1. Użytkownik  
    2. Administrator  
    3. Wyjście  
    1  
  
    --- Menu klienta ---  
    1. Cola | Cena: 3.5 PLN | Ilość: 7  
    2. Woda | Cena: 2.0 PLN | Ilość: 19  
    3. Sok | Cena: 4.0 PLN | Ilość: 5  
    Wybierz numer napoju lub wpisz X, aby wrócić:  
    1  
    Wrzucić monety (PLN): 4  
    Wydano resztę: 0.5 PLN  
    Dziękujemy za zakup!  
  
    --- Menu klienta ---  
    1. Cola | Cena: 3.5 PLN | Ilość: 6  
    2. Woda | Cena: 2.0 PLN | Ilość: 19  
    3. Sok | Cena: 4.0 PLN | Ilość: 5  
    Wybierz numer napoju lub wpisz X, aby wrócić:  
    x  
  
    1. Użytkownik  
    2. Administrator  
    3. Wyjście  
    2  
    Podaj hasło admina: admin123
```

Figure 1: Rysunek 3.1: Przykładowy sposób działania programu

4 Harmonogram realizacji projektu

Realizacja projektu przebiegała według zaplanowanych etapów:

- **Tydzień 1:** Zebranie i analiza wymagań oraz określenie funkcji, które ma spełniać aplikacja. Ustalenie zakresu projektu oraz przygotowanie wstępnej specyfikacji. aplikacji.
- **Tydzień 2:** Projektowanie struktury klas i pakietów oraz ogólnej architektury systemu. Rozpoczęcie kodowania podstawowych elementów i logiki aplikacji.
- **Tydzień 3:** Implementacja interfejsu użytkownika (konsolowego), obsługa błędów i wyjątków. Testowanie aplikacji, wprowadzanie poprawek oraz opracowanie dokumentacji technicznej.

5 Bibliografia

1. Jacek Galowicz, *Java. Wprowadzenie. Kurs video* -, Helion 2020
2. Krzysztof Jung, *Java. Programowanie obiektowe. Praktyczne wprowadzenie* -, Helion 2019
3. Java Tutorials, Official Java Tutorials - <https://docs.oracle.com/javase/tutorial/>
4. Jakub Nabrdalik, Java – poradnik programisty - <https://javastart.pl/java/poradnik-programisty>

6 GitHub

Projekt zarządzany był z wykorzystaniem systemu kontroli wersji Git. Repozytorium zostało utworzone na platformie GitHub pod adresem: <https://github.com/Bentleyyy44/AutomatZNapojami>. Projekt zawiera historię zmian, obejmującą co najmniej pięć commitów.

7 Spis listningów

Listings

1	Klasa Drink	5
2	Klasa VendingMachine	6
3	Klasa Transaction	9
4	Klasa Transaction manager	9
5	Klasa ConsoleUI	10
6	Klasa InsufficientFundsException	13
7	Klasa Main	13
8	Inicjalizacja domyślnych napojów	13
9	Pętla główna aplikacji:	14
10	Przykład walidacji wejścia:	15

8 Spis rysunków

List of Figures

1	Rysunek 3.1: Przykładowy sposób działania programu	16
---	--	----

*

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

Bartosz Paweł Betlej

Informatyka i Ekonometria

137132

1. Oświadczam, że moja praca projektowa pt.: System automat z napojami

- została przygotowana przeze mnie samodzielnie,
- 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,
- nie zawiera danych i informacji, które uzyskałem/am w sposób niedozwolony,
- nie była podstawą otrzymania oceny z innego przedmiotu na uczelni wyższej ani mnie, ani innej osobie.

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

(Rzeszów 11.06.25)

(Bartosz Betlej)