



BiLOTeo

Dokumentacja Techniczna

Autorzy: Paweł Korzeniewski 6656 i Piotr Nowiński 6560

Spis Treści:

3.....	Wprowadzenie
3.....	Założenia programu
4.....	Funkcje programu
4.....	Instalacja
5.....	Składowe programu
6.....	Logowanie
8.....	Wybieranie rodzaju połączenia
9.....	Określenie detali lotu
10.....	Wyszukiwanie połączenia
12.....	Akceptacja i zapis do bazy danych
14.....	Drukowanie Biletu
15.....	Możliwe problemy w funkcjonowaniu aplikacji
16.....	Słownik pojęć
17.....	Źródła i użyte narzędzia

Wprowadzenie:

Program BiLOTeo to aplikacja mająca za zadanie usprawnić rezerwowanie biletów lotniczych. BiLOTeo umożliwia wyszukiwanie połączeń, sprawdzenie cen w zależności od klasy i rodzaju lotu, rezerwację i wydruk biletu dzięki czemu nie trzeba fatygować się specjalnie na lotnisko i stać w kolejce do okienka rezerwacji. Dzięki programowi BiLOTeo masz pewność, że zarezerwowane miejsce w samolocie będzie zawsze na Ciebie czekać.

Założenia programu:

BiLOTeo korzysta z architektury klient-serwer(rejestr). Klient platformy kontaktuje się z serwerem w celu tworzenia bazy danych użytkowników, prowadzenia rejestru pasażerów i używania aktualnego rozkładu lotów.

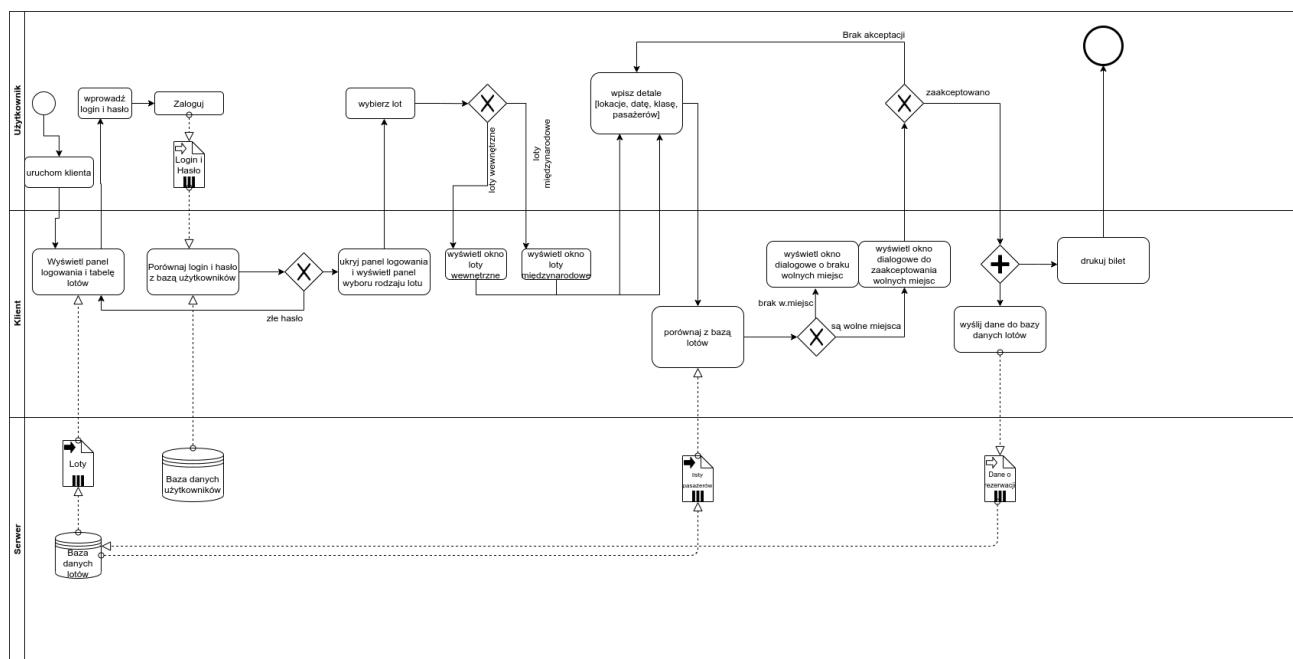


Diagram Procesów w standardzie BPMN

Funkcje Programu:

- Logowanie;
- Szukanie Połączeń;
- Rezerwacja Biletu;
- Drukowanie biletu;
- Obsługa Okienkowa;

Instalacja:

Aby poprawnie korzystać z BiLOTeo potrzebna jest instalacja aktualnego pakietu użytkownika Java (<https://java.com/pl/download/manual.jsp>). Po zainstalowaniu pakietu Java możemy korzystać z aplikacji.

Składowe Programu:

Program ten został stworzony w języku wysokiego poziomu, którym jest Java przy użyciu pakietu Swing, który zawiera framework – JFrame, co umożliwiło stworzenie GUI, które znacznie ułatwia poruszanie się po programie. Java jako język obiektowy pozwala na dużą przejrzystość kodu, przez co przy ewentualnych awariach istnieje łatwość w wyszukiwaniu błędów.

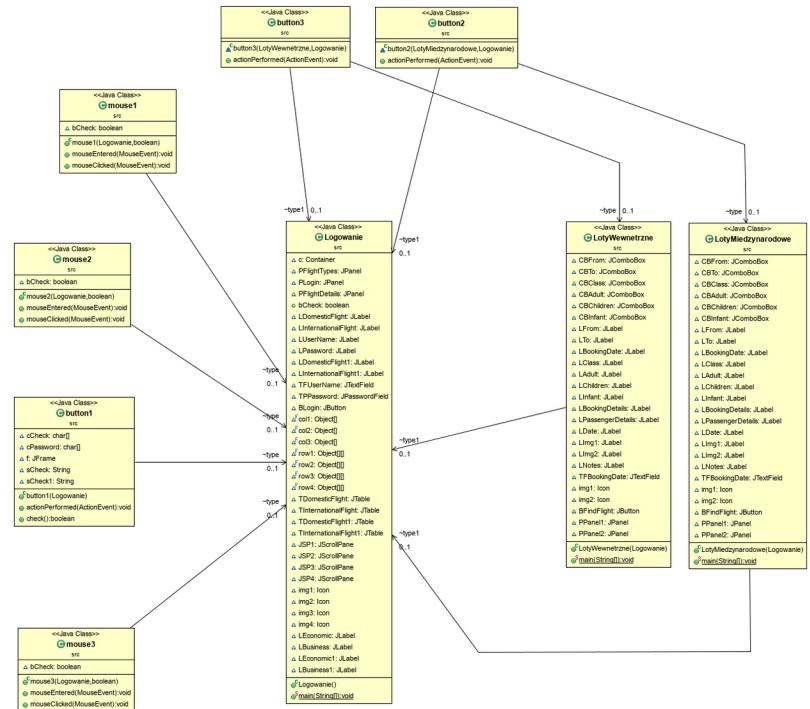


Diagram klas w standardzie UML

Logowanie:

Jest to klasa początkowa, odpowiedzialna za wprowadzenie użytkownika do programu poprzez wyświetlenie tabeli lotów w zależności od rodzaju połączenia jak i możliwości zalogowania się.

Okno logowania się do programu

Baza Tabelki jest zapisana w postaci obiektów.

```
final Object[] col1 = { "Z", "Do", "Cena[PLN]", "Godzina Odlotu" };

final Object[] col2 = { "Z", "Do", "Cena[PLN]", "Godzina Odlotu" };

final Object[] col3 = { "Z", "Do", "Cena[PLN]", "Godzina Odlotu" };

final Object[][] row1 = { { "Wroclaw", "Gdynia", "248", "16:30" },
{ "Wroclaw", "Warszawa-Modlin ", "225", "19:00" }...

final Object[][] row2 = { { "Wroclaw", "Bali", "4485", "06:20" },...;

...
```

Weryfikacja użytkownika

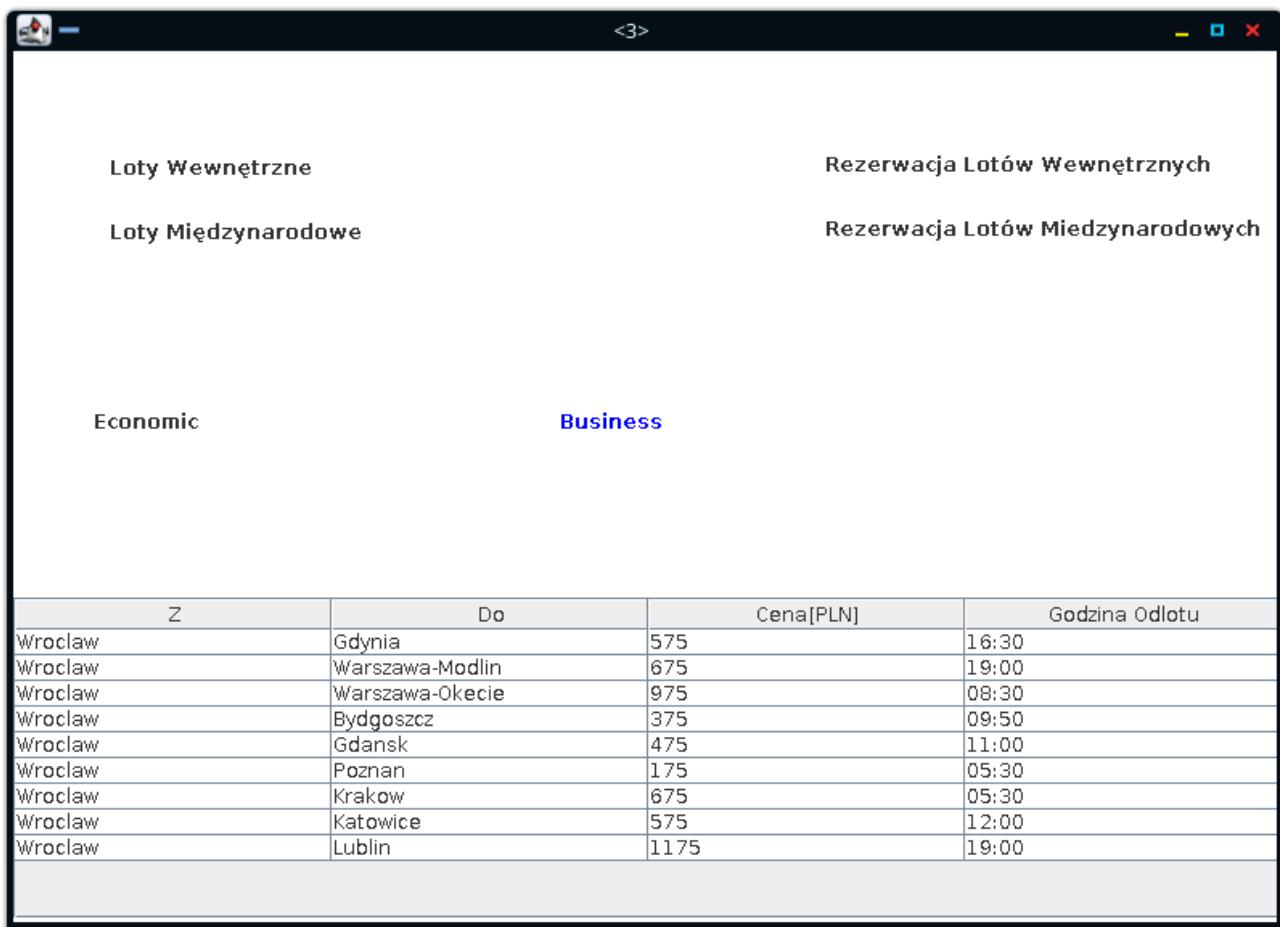
Użytkownik jest weryfikowany za pomocą algorytmu przyrównującego:

```
class button1 implements ActionListener
{
    Logowanie type;
    char[] cCheck, cPassword={'a','d','m','i','n','\0'};
    JFrame f;
    String sCheck,sCheck1="admin";

    public button1(Logowanie type)
    {
        this.type = type;
    }
    public void actionPerformed(ActionEvent e)
    {
        cCheck=type.TPPassword.getPassword();
        sCheck = type.TFUserName.getText();
        if ((sCheck1.equals(sCheck)) && check())
        {
            ...
        }
        else
        {
            JOptionPane.showMessageDialog(null, "Invalid username or
password. Try again");
        }
    }
    public boolean check()
    {
        if (cCheck.length >= 6 || cCheck.length < 4)
            return false;
        for(int i=0;i<=4;i++)
        {
            if(cCheck[i]!=cPassword[i])
                return false;
        }
        return true;
    }
}
```

Wybór rodzaju połączenia:

Gdy weryfikacja przebiegnie pomyślnie otwierane zostaje pole wyboru rodzaju połączenia



Okno wyboru rodzaju lotu

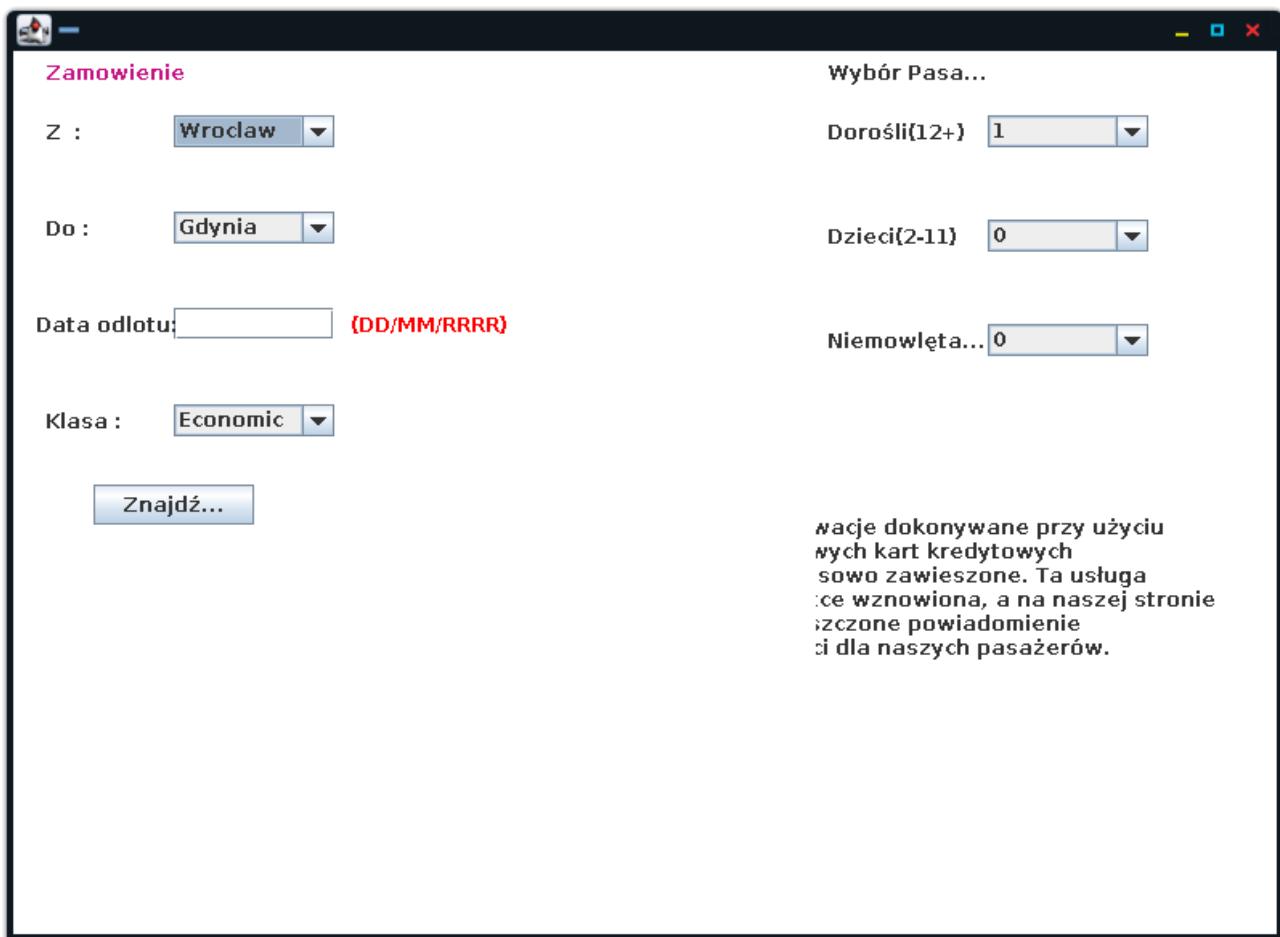
Użytkownik ma do wyboru dwie możliwości, *Rezerwacja Lotów Wewnętrznych* i *Rezerwacja Lotów Międzynarodowych*. Wybranie odpowiedniej opcji przeniesie użytkownika do nowego okna,

```
public void mouseEntered(MouseEvent e)
{
    type.LDomesticFlight1.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
    type.LInternationalFlight1.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
}
public void mouseClicked(MouseEvent e)
{
    if (bCheck)
        new LotyWewnetrzne(type);
    else
        new LotyMiedzynarodowe(type);
}
```

Określenie Detali Lotu

Po wyborze rodzaju lotu przechodzimy do okna dotyczącego naszego lotu (*LotyWewnętrzne/LotyMiędzynarodowe*). W tym oknie wybieramy detale takie jak:

- miejsce początkowe;
- miejsce docelowe;
- data lotu;
- klasa;
- ilość i rodzaj pasażerów;

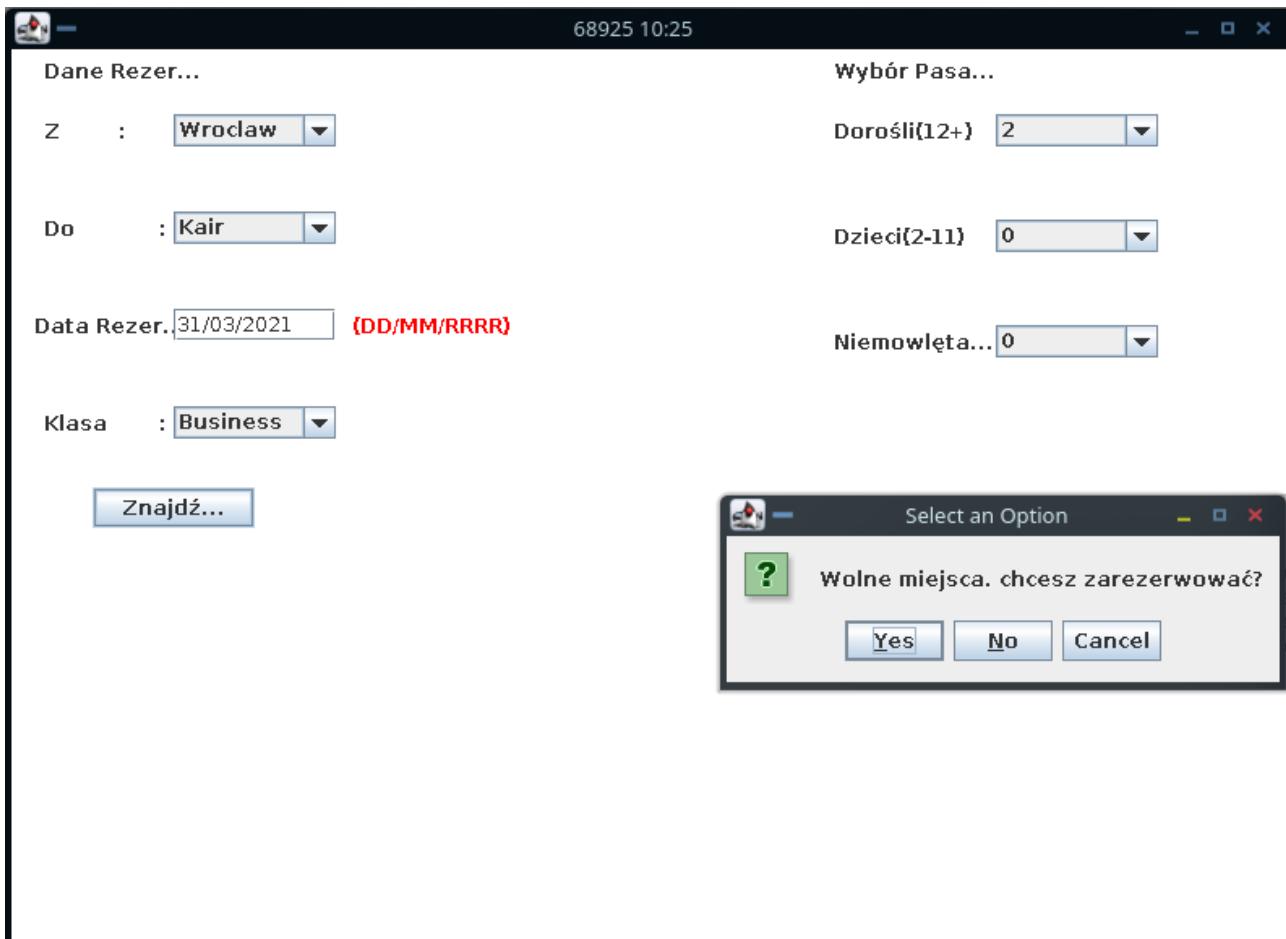


Baza miejsc jest określona w klasie przypisanej do tego okna.

```
public LotyWewnętrzne(Logowanie type1)
{
    Container c = getContentPane();
    c.setLayout(new BorderLayout());
    String[] sItem1={"Wroclaw"};
    String[] sItem2 {"Gdynia","Warszawa-Modlin","Warszawa-Okocie","Bydgoszcz","Gdansk","Poznan","Krakow","Katowice","Lublin","Rzeszow","Lodz","Olsztyn","Radom"};
    String[] sItem3={"Economic","Business"};
```

Wyszukiwanie Połączenia

Wyszukiwanie połączenia działa w sposób porównania obiektów, obiektów z tabeli lotów i obiektu stworzonego w oknie wyboru detali. Gdy algorytm odnajdzie równość między dwoma obiektyami to program da nam komunikat o wolnych miejscach.



By porównać należy przywołać obiekt z klasy okna *Logowanie*

```
LotyMiedzynarodowe type;
    Logowanie type1;
    button2(LotyMiedzynarodowe type, Logowanie type1)
    {
        this.type = type;
        this.type1 = type1;
    }
```

Potem wykorzystać algorytm porównawczy `equals()`

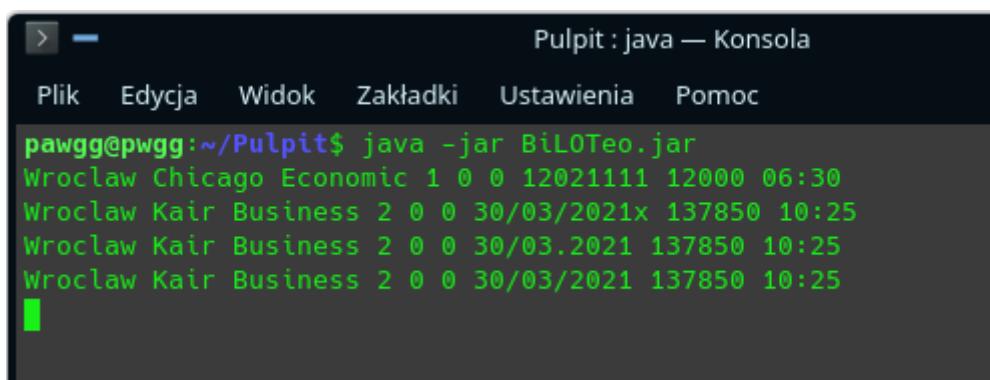
```
if(sClass.equals("Economic"))
{
    try{
        while(i<20)
        {
            if(type1.row2[i][1].equals(sTo))
            {
                iPrice =
Integer.parseInt((String)type1.row2[i][2]);
                sTime = (String)type1.row2[i][3];
                break;
            }
            i++;
        }
    }catch(Exception e1)
    {
        JOptionPane.showMessageDialog(null, "brak dostępu");
        System.exit(0);
    }
}
```

W przypadku nie znalezienia odpowiedniego typu wyświetli się komunikat o odmowie dostępu.

Gdy zostanie znaleziony odpowiedni typ, program stworzy obiekt wyjściowy uzupełniony o cenę biletu i czas odlotu.

Akceptacja i zapis do bazy danych

Po zatwierdzeniu wyboru detali dane zostają przesłane do bazy, z której idzie potwierdzenie do użytkownika o zakupionych wcześniej biletach co odbywa się już po za okiem użytkownika.



```
pawgg@pwgg:~/Pulpit$ java -jar BiLOTeo.jar
Wroclaw Chicago Economic 1 0 0 12021111 12000 06:30
Wroclaw Kair Business 2 0 0 30/03/2021x 137850 10:25
Wroclaw Kair Business 2 0 0 30/03/2021 137850 10:25
Wroclaw Kair Business 2 0 0 30/03/2021 137850 10:25
```

Klasa odpowiedzialna za wczytanie biletów klienta i jej wyświetlenie w terminalu

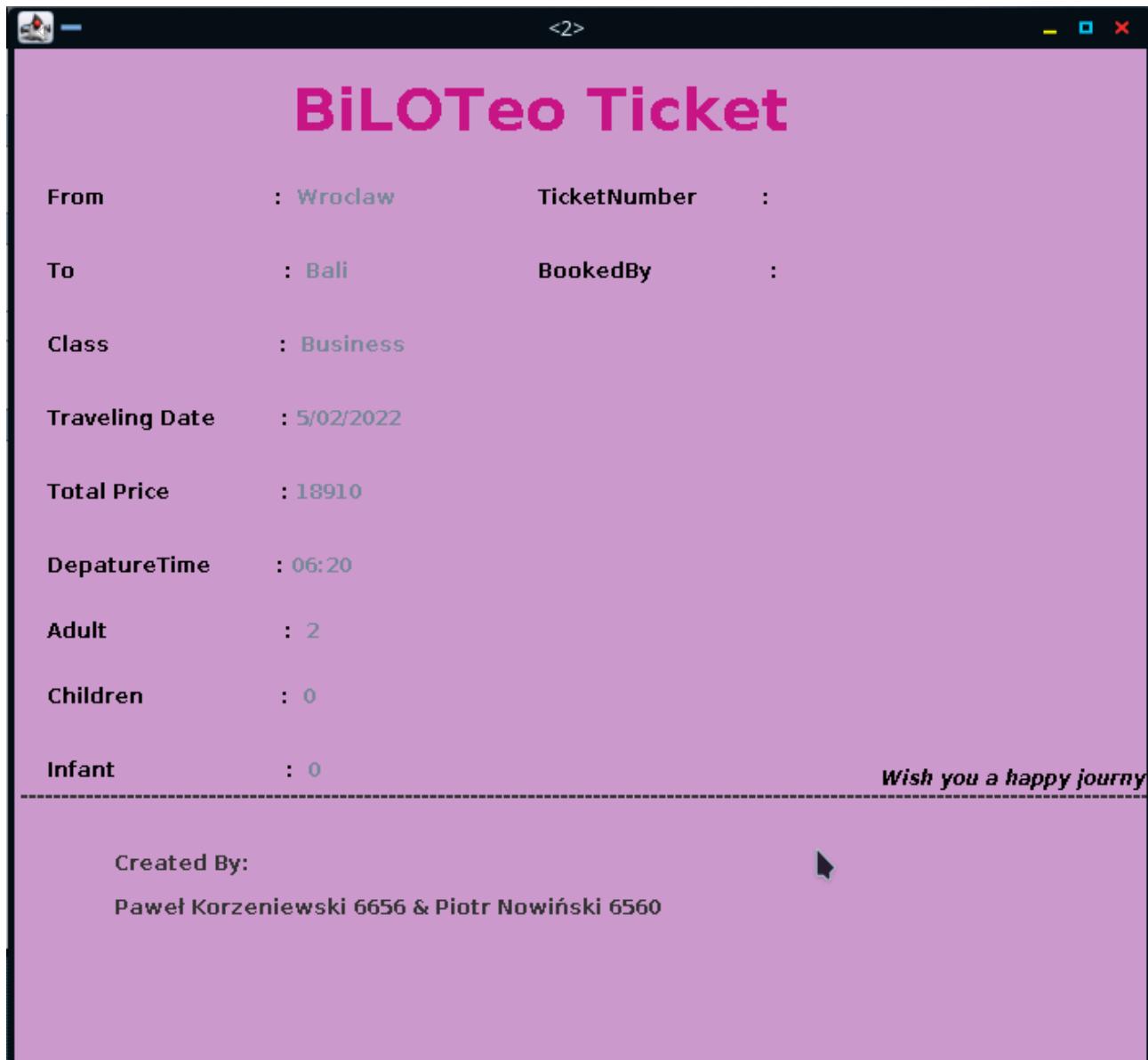
```
Save1 save2=new Save1(sFrom, sTo, sClass, iAdult, iChildren, iInfant,
sBookingDate, iPrice, sTime);
ObjectOutputStream 00S1 = new ObjectOutputStream(new
FileOutputStream("save1"));
for(i=0;i<iCount;i++)
{
    Save1 templ1=new Save1(sTempFrom[i], sTempTo[i],
sTempClass[i], iTempAdult[i], iTempChildren[i], iTempInfant[i],
sTempBookingDate[i], iTempPrice[i], sTempTime[i]);
    00S1.writeObject(templ1);
System.out.println(templ1);
}
00S1.writeObject(save2);
00S1.close();
}catch(Exception e1)
{
    System.out.println(e1);
}
}
else
{
}
}
}
}
}
```

Klasa odpowiedzialna za dopisanie biletów do bazy

```
class Save1 implements Serializable
{
    String sFrom, sTo, sClass, sBookingDate, sTime;
    Integer iPrice, iAdult, iChildren, iInfant;
    public Save1(String sFrom, String sTo, String sClass, Integer iAdult,
    Integer iChildren, Integer iInfant, String sBookingDate, Integer iPrice, String
    sTime)
    {
        this.sFrom=sFrom;
        this.sTo=sTo;
        this.sClass=sClass;
        this.iAdult=iAdult;
        this.iChildren=iChildren;
        this.iInfant=iInfant;
        this.sBookingDate=sBookingDate;
        this.iPrice=iPrice;
        this.sTime=sTime;
    }
    public String toString()
    {
        return sFrom+" "+sTo+" "+sClass+" "+iAdult+" "+iChildren+
        "+iInfant+" "+sBookingDate+" "+iPrice+" "+sTime;
    }
}
```

Drukowanie Biletu

Równocześnie gdy dane zostają wpisane do bazy, użytkownik otrzymuje swój bilet wypełniony danymi opartymi o zatwierdzone wcześniej detale.



Możliwe problemy w funkcjonowaniu aplikacji

Brak dostępu do bazy danych:

W przypadku braku dostępu do bazy danych, program tworzy nową bazę, gdyby to była aplikacja, która działa z zewnętrzną bazą danych użyte zostałoby API do synchronizowania danych z bazą wewnętrzną(save1 i save2).

Brak dostępu do aplikacji

W przypadku braku dostępu do aplikacji z powodu nadużycia(manipulowania w lotach) lub błędnego logowania otrzymamy komunikat i użytkownika przeniesieni do strony logowania.

Słownik Pojęć

- **GUI** - Graficzny interfejs użytkownika (**Graphical User Interface**) czyli okienkowy wygląd aplikacji.
- **Framework** – Narzędzie pozwalające tworzyć oprogramowanie za pomocą wcześniej ustalonych zasad(pot. Szkieletu) zawierające biblioteki i komponenty.
- **Architektura klient-serwer** – model działania systemu opierający się na podziale zadań między oprogramowaniem(**klientem**) a **serwerem**.
- **Baza danych** – zbiór wszystkich danych zapisanych w jednym pliku.
- **Obiekt** – Zbiór detali unormowanych przez klasę, która go stworzyła.
- **Klasa** – Zbiór definicji potrzebnych do tworzenia obiektu.

Źródła

- Grafika ze strony tytułowej <https://wallpapercave.com/wp/3y909ok.jpg>

Użyte narzędzia

- Debian <http://www.debian.org>
- Eclipse IDE <https://www.eclipse.org>
- ObjectAid UML Explorer <https://marketplace.eclipse.org/content/objectaid-uml-explorer>
- bpmn.io <https://bpmn.io>
- Java JDK 8 <https://www.oracle.com/pl/java/technologies/javase/javase-jdk8-downloads.html>
- Edytor tekstu Libre Office Writer(<https://pl.libreoffice.org>