Dawid Waligórski, 264015
Michał Dziedziak, 263901

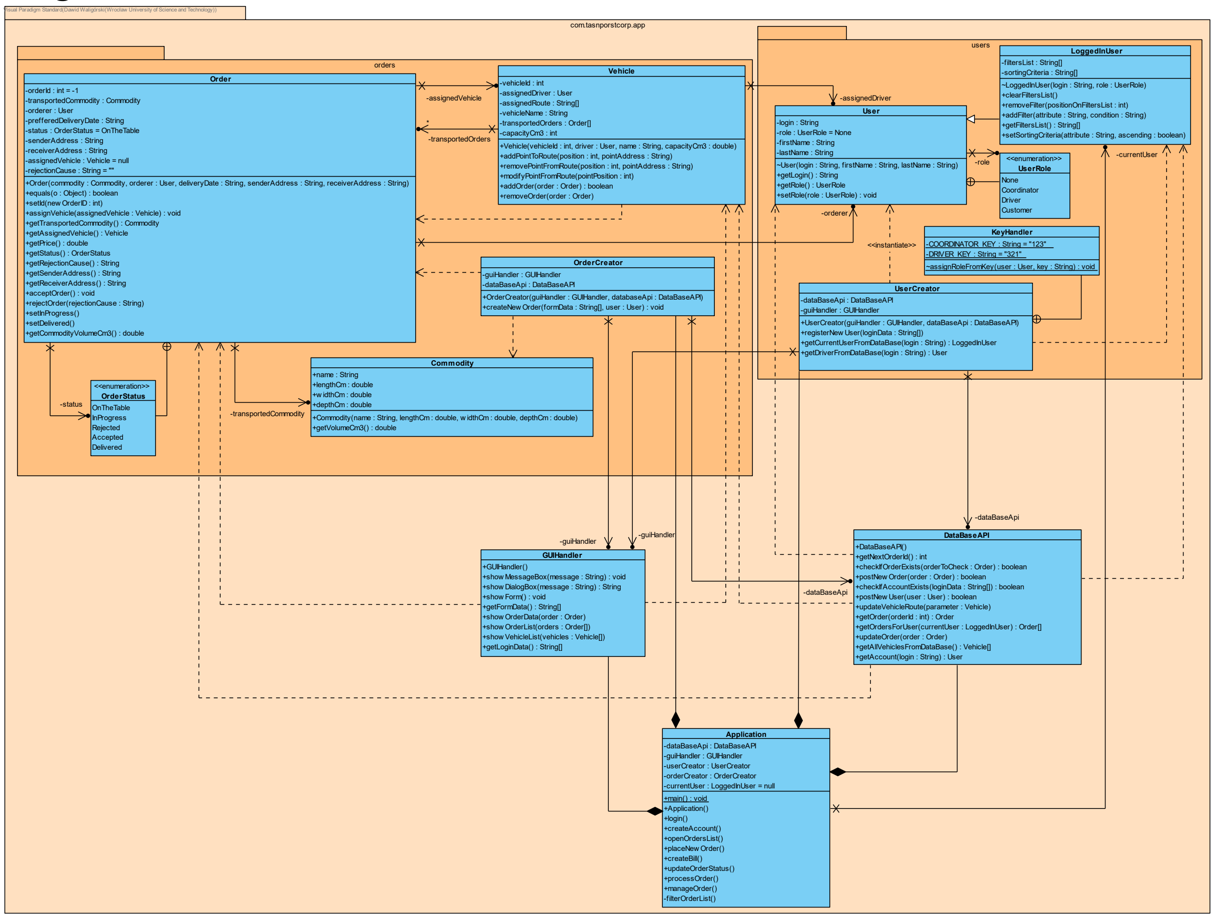
Laboratorium 7, 8, 9, 10

- Identyfikacja klas reprezentujących logikę biznesową projektowanego oprogramowania. Definicja atrybutów i operacji klas oraz związków między klasami na podstawie analizy scenariuszy przypadków użycia. Opracowanie diagramów klas i pakietów. Zastosowanie projektowych wzorców strukturalnych i wytwórczych.
- Opracowanie diagramów sekwencji dla wybranych przypadków użycia, reprezentujących usługi oprogramowania, wynikających również z wykonanych diagramów czynności. Definicja operacji klas na podstawie diagramów sekwencji w języku Java. Zastosowanie projektowych wzorców zachowania.
- Opracowanie diagramów sekwencji dla wybranych przypadków użycia reprezentujących usługi oprogramowania wynikających również z wykonanych diagramów czynności. Definicja operacji klas na podstawie diagramów sekwencji w języku Java. Zastosowanie projektowych wzorców zachowania.

Temat projektu

Program obsługujący zakład transportowy.

Diagram klas:


Visual Paradigm Standard(Dawid Waligórski(Wrocław University of Science and Technology))
com.tasnporstcorp.app
orders
Order
-orderId : int = -1
-transportedCommodity : Commodity
-orderer : User
-prefferedDeliveryDate : String
-status : OrderStatus = OnTheTable
-senderAddress : String
-receiverAddress : String
-assignedVehicle : Vehicle = null
-rejectionCause : String = ""
+Order(commodity : Commodity, orderer : User, deliveryDate : String, senderAddress : String, receiverAddress : String)
+equals(o : Object) : boolean
+setId(new OrderID : int)
+assignVehicle(assignedVehicle : Vehicle) : void
+getTransportedCommodity() : Commodity
+getAssignedVehicle() : Vehicle
+getPrice() : double
+getStatus() : OrderStatus
+getRejectionCause() : String
+getSenderAddress() : String
+getReceiverAddress() : String
+acceptOrder() : void
+rejectOrder(rejectionCause : String)
+setInProgress()
+setDelivered()
+getCommodityVolumeCm3() : double
-assignedVehicle
*
-transportedOrders
Vehicle
-vehicleId : int
-assignedDriver : User
-assignedRoute : String[]
-vehicleName : String
-transportedOrders : Order[]
-capacityCm3 : int
+Vehicle(vehicleId : int, driver : User, name : String, capacityCm3 : double)
+addPointToRoute(position : int, pointAddress : String)
+removePointFromRoute(position : int, pointAddress : String)
+modifyPointFromRoute(pointPosition : int)
+addOrder(order : Order) : boolean
+removeOrder(order : Order)
OrderCreator
-guiHandler : GUIHandler
-dataBaseApi : DataBaseAPI
+OrderCreator(guiHandler : GUIHandler, databaseApi : DataBaseAPI)
+createNew Order(formData : String[], user : User) : void
-status
<<enumeration>>
OrderStatus
OnTheTable
InProgress
Rejected
Accepted
Delivered
-transportedCommodity
Commodity
+name : String
+lengthCm : double
+w idthCm : double
+depthCm : double
+Commodity(name : String, lengthCm : double, w idthCm : double, depthCm : double)
+getVolumeCm3() : double
users
-assignedDriver
User
-login : String
-role : UserRole = None
-firstName : String
-lastName : String
~User(login : String, firstName : String, lastName : String)
+getLogin() : String
+getRole() : UserRole
+setRole(role : UserRole) : void
-orderer
-role
LoggedInUser
-filtersList : String[]
-sortingCriteria : String[]
~LoggedInUser(login : String, role : UserRole)
+clearFiltersList()
+removeFilter(positionOnFiltersList : int)
+addFilter(attribute : String, condition : String)
+getFiltersList() : String[]
+setSortingCriteria(attribute : String, ascending : boolean)
-currentUser
<<enumeration>>
UserRole
None
Coordinator
Driver
Customer
<<instantiate>>
KeyHandler
-COORDINATOR_KEY : String = "123"
-DRIVER_KEY : String = "321"
~assignRoleFromKey(user : User, key : String) : void
UserCreator
-dataBaseApi : DataBaseAPI
-guiHandler : GUIHandler
+UserCreator(guiHandler : GUIHandler, dataBaseApi : DataBaseAPI)
+registerNew User(loginData : String[])
+getCurrentUserFromDataBase(login : String) : LoggedInUser
+getDriverFromDataBase(login : String) : User
-guiHandler
-guiHandler
GUIHandler
+GUIHandler()
+show MessageBox(message : String) : void
+show DialogBox(message : String) : String
+show Form() : void
+getFormData() : String[]
+show OrderData(order : Order)
+show OrderList(orders : Order[])
+show VehicleList(vehicles : Vehicle[])
+getLoginData() : String[]
-dataBaseApi
-dataBaseApi
DataBaseAPI
+DataBaseAPI()
+getNextOrderId() : int
+checkIfOrderExists(orderToCheck : Order) : boolean
+postNew Order(order : Order) : boolean
+checkIfAccountExists(loginData : String[]) : boolean
+postNew User(user : User) : boolean
+updateVehicleRoute(parameter : Vehicle)
+getOrder(orderId : int) : Order
+getOrdersForUser(currentUser : LoggedInUser) : Order[]
+updateOrder(order : Order)
+getAllVehiclesFromDataBase() : Vehicle[]
+getAccount(login : String) : User
Application
-dataBaseApi : DataBaseAPI
-guiHandler : GUIHandler
-userCreator : UserCreator
-orderCreator : OrderCreator
-currentUser : LoggedInUser = null
+main() : void
+Application()
+login()
+createAccount()
+openOrdersList()
+placeNew Order()
+createBill()
+updateOrderStatus()
+processOrder()
+manageOrder()
-filterOrderList()

Kod klas

Application

```java
package com.tasnporstcorp.app;

import java.util.*;
import com.tasnporstcorp.app.users.*;
import com.tasnporstcorp.app.orders.*;

public class Application {

        private DataBaseAPI dataBaseApi;
        private GUIHandler guiHandler;
        private UserCreator userCreator;
        private OrderCreator orderCreator;
        private LoggedInUser currentUser = null;

        public static void main(String[] args) {

                Application app = new Application();
                app.createAccount();
                app.login();
                app.placeNewOrder();
```

```java
	}

	public Application() {
		guiHandler = new GUIHandler();
		dataBaseApi = new DataBaseAPI();
		orderCreator = new OrderCreator(guiHandler, dataBaseApi);
		userCreator = new UserCreator(guiHandler, dataBaseApi);
	}

	public void login() {
		currentUser = userCreator.getCurrentUserFromDataBase("login");
	}

	public void createAccount() {
		ArrayList<String> loginData = guiHandler.getLoginData();
		userCreator.registerNewUser(loginData);
	}

	public void openOrdersList() {
		throw new UnsupportedOperationException();
	}

	public void placeNewOrder() {
		ArrayList<String> formData = guiHandler.getFormData();
	    orderCreator.createNewOrder(formData, currentUser);
	}

	public void createBill() {
		throw new UnsupportedOperationException();
	}

	public void updateOrderStatus() {
		throw new UnsupportedOperationException();
	}

	public void processOrder() {
		throw new UnsupportedOperationException();
	}

	public void manageOrder() {
		throw new UnsupportedOperationException();
	}

	private void filterOrderList() {
		throw new UnsupportedOperationException();
	}

}
```

DataBaseAPI

```java
package com.tasnporstcorp.app;
import java.util.*;
import com.tasnporstcorp.app.orders.*;
import com.tasnporstcorp.app.users.*;

public class DataBaseAPI {

	public DataBaseAPI() {

	}

	public int getNextOrderId() {
		return 1;
	}

	public boolean checkIfOrderExists(Order orderToCheck) {
		return false;
	}

	public boolean postNewOrder(Order order) {
		return true;
	}

	public boolean checkIfAccountExists(ArrayList<String> loginData) {
		return false;
	}

	public boolean postNewUser(User user) {
		return true;
	}

	public void updateVehicleRoute(Vehicle parameter) {
		throw new UnsupportedOperationException();
	}

	public Order getOrder(int orderId) {
		throw new UnsupportedOperationException();
	}

	public ArrayList<Order> getOrdersForUser(LoggedInUser currentUser) {
		throw new UnsupportedOperationException();
	}

	public void updateOrder(Order order) {
		throw new UnsupportedOperationException();
	}

	public ArrayList<Vehicle> getAllVehiclesFromDataBase() {
		throw new UnsupportedOperationException();
	}

	public User getAccount(String login) {
```

```
            throw new UnsupportedOperationException();
      }

}
```

GUIHandler

```
package com.tasnporstcorp.app;

import java.util.Scanner;
import java.util.*;

import com.tasnporstcorp.app.orders.*;

public class GUIHandler {

      public GUIHandler() {

      }

      public void showMessageBox(String message) {
            Scanner in = new Scanner(System.in);
            System.out.println(message);
            System.out.println("Wprowadź ENTER aby kontynuować...");
            in.nextLine();
      }

      public String showDialogBox(String message) {
            Scanner in = new Scanner(System.in);
            System.out.println(message);
            System.out.print("> ");
            return in.nextLine();
      }

      public void showForm() {
            throw new UnsupportedOperationException();
      }

      public ArrayList<String> getFormData() {
            var tmp = new String[]{"paczka", "1", "1", "1", "01-01-2023",
"10-01-2023", "Wrocław, ul. Rynek 4", "Wrocław, ul. Rynek 4"};
            return new ArrayList<String>(Arrays.asList(tmp));
      }

      public void showOrderData(Order order) {
            throw new UnsupportedOperationException();
      }

      public void showOrderList(ArrayList<Order> orders) {
            throw new UnsupportedOperationException();
      }

      public void showVehicleList(Vehicle vehicles) {
            throw new UnsupportedOperationException();
      }

      public ArrayList<String> getLoginData() {
            var tmp = new String[]{"login", "jan", "kowalski"};
            return new ArrayList<String>(Arrays.asList(tmp));
      }

}
```

UserCreator

```
package com.tasnporstcorp.app.users;

import com.tasnporstcorp.app.*;
import com.tasnporstcorp.app.users.User.UserRole;
import java.util.*;

public class UserCreator {

      private DataBaseAPI dataBaseApi;
      private GUIHandler guiHandler;

      public UserCreator(GUIHandler guiHandler, DataBaseAPI dataBaseApi) {
            this.dataBaseApi = dataBaseApi;
            this.guiHandler = guiHandler;
      }

      public void registerNewUser(ArrayList<String> loginData) {
            boolean isUserInDatabase =
dataBaseApi.checkIfAccountExists(loginData);
            if(isUserInDatabase) {
                  guiHandler.showDialogBox("Konto o danym loginie już
istnieje");
                  return;
            }

            User user = new User(loginData.get(0), loginData.get(1),
loginData.get(2));
            UserRole role = UserRole.None;

            while (role == UserRole.None) {
                  String userKey = guiHandler.showDialogBox("Wprowadź klucz
dostępu.");
                  KeyHandler.assignRoleFromKey(user, userKey);
                  role = user.getRole();
                  if(role == UserRole.None)
                        guiHandler.showMessageBox("Podany klucz jest
błędny.");
            }

            guiHandler.showMessageBox("Dodano nowe konto użytkownika!");
            dataBaseApi.postNewUser(user);
      }

      public LoggedInUser getCurrentUserFromDataBase(String login) {
            return new LoggedInUser(login, UserRole.Customer, "jan",
"kowalski");
      }

      public User getDriverFromDataBase(String login) {
            throw new UnsupportedOperationException();
      }


      public static class KeyHandler {

            private static final String COORDINATOR_KEY = "123";
            private static final String DRIVER_KEY = "321";

            static void assignRoleFromKey(User user, String key) {
                  if(key.equals("")){
                        user.setRole(UserRole.Customer);
                  }
                  else if(key.equals(COORDINATOR_KEY)){
                        user.setRole(UserRole.Coordinator);
                  }
                  else if(key.equals(DRIVER_KEY)){
                        user.setRole(UserRole.Driver);
                  }
                  return;
            }

      }

}
```

User

```
package com.tasnporstcorp.app.users;

public class User {

      private String login;
      private UserRole role =
com.tasnporstcorp.app.users.User.UserRole.None;
      private String firstName;
      private String lastName;

      User(String login, String firstName, String lastName) {
            this.login = login;
            this.firstName = firstName;
            this.lastName = lastName;
      }

      public String getLogin() {
```

```
		return this.login;
	}

	public UserRole getRole() {
		return this.role;
	}

	public void setRole(UserRole role) {
		this.role = role;
	}


	public enum UserRole {
		Coordinator,
		Driver,
		Customer,
		None
	}

}
```

LoggedInUser

```
package com.tasnporstcorp.app.users;
import com.tasnporstcorp.app.users.User.*;
import java.util.*;
public class LoggedInUser extends User {

	private ArrayList<String> filtersList;
	private ArrayList<String> sortingCriteria;

	LoggedInUser(String login, UserRole role, String firstName, String
lastName) {
		// TODO - implement LoggedInUser.LoggedInUser
		super(login, firstName, lastName);
		filtersList = new ArrayList<>();
		sortingCriteria = new ArrayList<>();
	}

	public void clearFiltersList() {
		throw new UnsupportedOperationException();
	}

	public void removeFilter(int positionOnFiltersList) {
		throw new UnsupportedOperationException();
	}

	public void addFilter(String attribute, String condition) {
		throw new UnsupportedOperationException();
	}

	public ArrayList<String> getFiltersList() {
		return this.filtersList;
	}

	public void setSortingCriteria(String attribute, boolean ascending) {
		throw new UnsupportedOperationException();
	}

}
```

Vehicle

```
package com.tasnporstcorp.app.orders;
import java.util.*;
import com.tasnporstcorp.app.users.*;

public class Vehicle {

	private int vehicleId;
	private User assignedDriver;
	private ArrayList<String> assignedRoute;
	private String vehicleName;
	private ArrayList<Order> transportedOrders;
	private int capacityCm3;

	public Vehicle(int vehicleId, User driver, String name, double
capacityCm3) {
		throw new UnsupportedOperationException();
	}

	public void addPointToRoute(int position, String pointAddress) {
		throw new UnsupportedOperationException();
	}

	public void removePointFromRoute(int position, String pointAddress) {
		throw new UnsupportedOperationException();
	}

	public void modifyPointFromRoute(int pointPosition) {
		throw new UnsupportedOperationException();
	}

	public boolean addOrder(Order order) {
		throw new UnsupportedOperationException();
	}

	public void removeOrder(Order order) {
		throw new UnsupportedOperationException();
	}

}
```

OrderCreator

```
package com.tasnporstcorp.app.orders;
import java.util.*;
import com.tasnporstcorp.app.*;
import com.tasnporstcorp.app.users.User;

public class OrderCreator {

      private GUIHandler guiHandler;
      private DataBaseAPI dataBaseApi;

      public OrderCreator(GUIHandler guiHandler, DataBaseAPI databaseApi) {
            this.guiHandler = guiHandler;
            this.dataBaseApi = databaseApi;
      }

      public void createNewOrder(ArrayList<String> formData, User user) {
            var commodity = new Commodity(formData.get(0),
Double.parseDouble(formData.get(1)), Double.parseDouble(formData.get(2)),
Double.parseDouble(formData.get(3)));
        var order = new Order(commodity, user, formData.get(4),
formData.get(5), formData.get(6));
        boolean doesOrderExists = dataBaseApi.checkIfOrderExists(order);
        if(doesOrderExists)
        {
            String userInteraction = guiHandler.showDialogBox("Zamówienie o
takich parametrach już istnieje. Czy chcesz kontynuować składanie
zamówienia?");
            if(userInteraction.contains("NO"))
            {
                return;
            }
        }

        int nextOrderId = dataBaseApi.getNextOrderId();
        order.setId(nextOrderId);

        boolean isSuccess = dataBaseApi.postNewOrder(order);
        if(isSuccess)
        {
            guiHandler.showMessageBox("Dodano nowe zamówienie");
        }
        else
        {
            guiHandler.showMessageBox("Nie udało się dodać zamówienia");
        }
      }

}
```

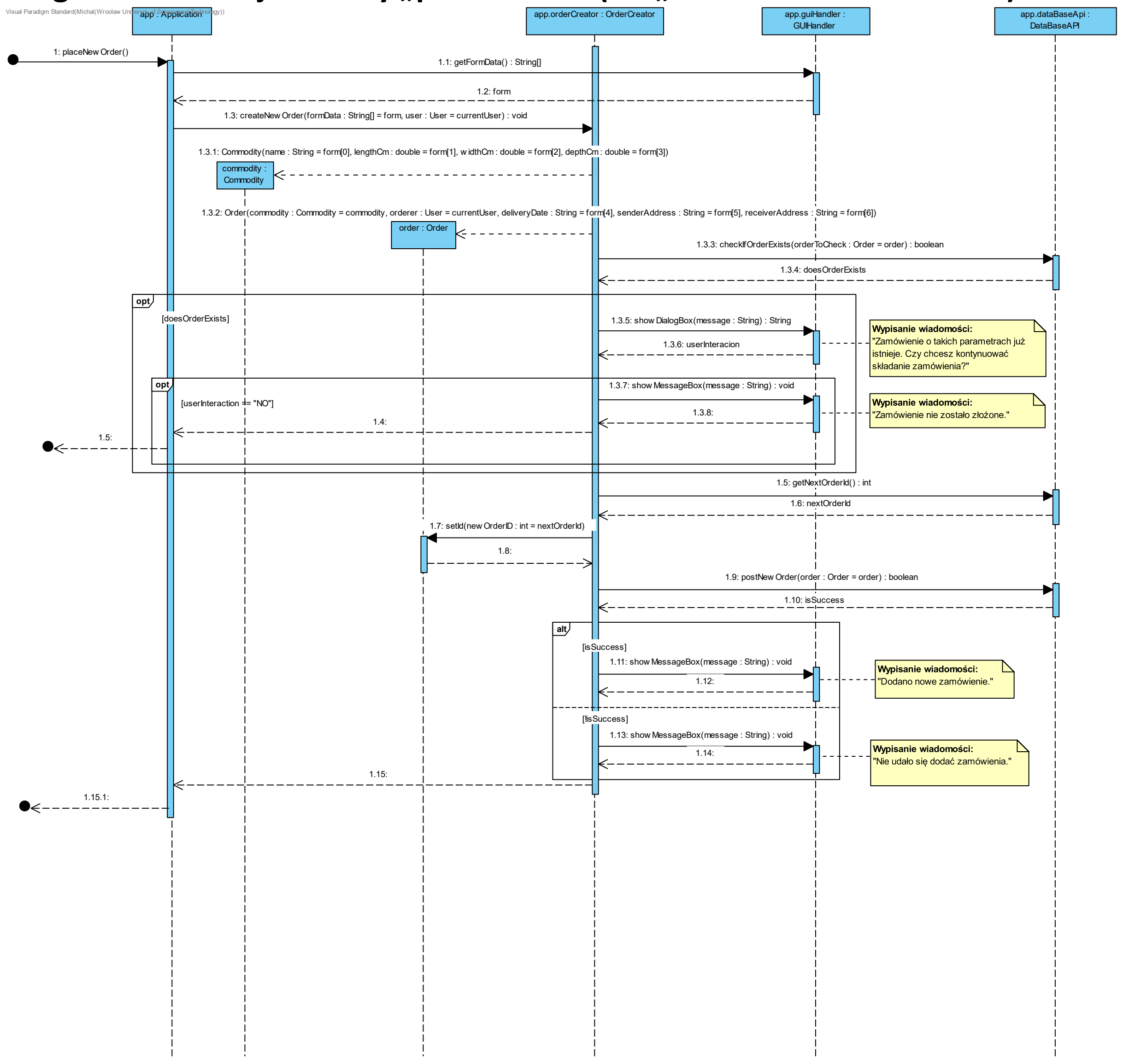
Order

```java
package com.tasnporstcorp.app.orders;

import com.tasnporstcorp.app.users.*;

public class Order {

	private int orderId = -1;
	private Commodity transportedCommodity;
	private User orderer;
	private String prefferedDeliveryDate;
	private OrderStatus status = 
com.tasnporstcorp.app.orders.Order.OrderStatus.OnTheTable;
	private String senderAddress;
	private String receiverAddress;
	private Vehicle assignedVehicle = null;
	private String rejectionCause = "";

	public Order(Commodity commodity, User orderer, String deliveryDate, 
String senderAddress, String receiverAddress) {
		transportedCommodity = commodity;
		this.orderer = orderer;
		this.prefferedDeliveryDate = deliveryDate;
		this.senderAddress = senderAddress;
		this.receiverAddress = receiverAddress;
	}

	public boolean equals(Object o) {
		throw new UnsupportedOperationException();
	}

	public void setId(int newOrderID) {
		orderId = newOrderID;
	}

	public void assignVehicle(Vehicle assignedVehicle) {
		throw new UnsupportedOperationException();
	}

	public Commodity getTransportedCommodity() {
		return this.transportedCommodity;
	}

	public Vehicle getAssignedVehicle() {
		return this.assignedVehicle;
	}

	public double getPrice() {
		throw new UnsupportedOperationException();
	}

	public OrderStatus getStatus() {
		return this.status;
	}

	public String getRejectionCause() {
		return this.rejectionCause;
	}

	public String getSenderAddress() {
		return this.senderAddress;
	}

	public String getReceiverAddress() {
		return this.receiverAddress;
	}

	public void acceptOrder() {
		throw new UnsupportedOperationException();
	}

	public void rejectOrder(String rejectionCause) {
		throw new UnsupportedOperationException();
	}

	public void setInProgress() {
		throw new UnsupportedOperationException();
	}

	public void setDelivered() {
		throw new UnsupportedOperationException();
	}


	public enum OrderStatus {
		OnTheTable,
		InProgress,
		Rejected,
		Accepted,
		Delivered
	}

}
```

Commodity

```
package com.tasnporstcorp.app.orders;

public class Commodity {

	public String name;
	public double lengthCm;
	public double widthCm;
	public double depthCm;

	public Commodity(String name, double lengthCm, double widthCm, double 
depthCm) {
		this.name = name;
		this.lengthCm = lengthCm;
		this.widthCm = widthCm;
		this.depthCm = depthCm;
	}

	public double getVolumeCm3() {
		throw new UnsupportedOperationException();
	}

}
```

Diagram sekwencji metody „placeOrder” (PU „Składanie zamówienia”):



Visual Paradigm Standard(Michal(Wrocław University of Science and Technology))
app : Application
app.orderCreator : OrderCreator
app.guiHandler : GUIHandler
app.dataBaseApi : DataBaseAPI
1: placeNew Order()
1.1: getFormData() : String[]
1.2: form
1.3: createNew Order(formData : String[] = form, user : User = currentUser) : void
1.3.1: Commodity(name : String = form[0], lengthCm : double = form[1], w idthCm : double = form[2], depthCm : double = form[3])
commodity : Commodity
1.3.2: Order(commodity : Commodity = commodity, orderer : User = currentUser, deliveryDate : String = form[4], senderAddress : String = form[5], receiverAddress : String = form[6])
order : Order
1.3.3: checkIfOrderExists(orderToCheck : Order = order) : boolean
1.3.4: doesOrderExists
opt
[doesOrderExists]
1.3.5: show DialogBox(message : String) : String
1.3.6: userInteracion
Wypisanie wiadomości: "Zamówienie o takich parametrach już istnieje. Czy chcesz kontynuować składanie zamówienia?"
opt
[userInteraction == "NO"]
1.3.7: show MessageBox(message : String) : void
1.3.8:
Wypisanie wiadomości: "Zamówienie nie zostało złożone."
1.4:
1.5:
1.5: getNextOrderId() : int
1.6: nextOrderId
1.7: setId(new OrderID : int = nextOrderId)
1.8:
1.9: postNew Order(order : Order = order) : boolean
1.10: isSuccess
alt
[isSuccess]
1.11: show MessageBox(message : String) : void
1.12:
Wypisanie wiadomości: "Dodano nowe zamówienie."
[!isSuccess]
1.13: show MessageBox(message : String) : void
1.14:
Wypisanie wiadomości: "Nie udało się dodać zamówienia."
1.15:
1.15.1:

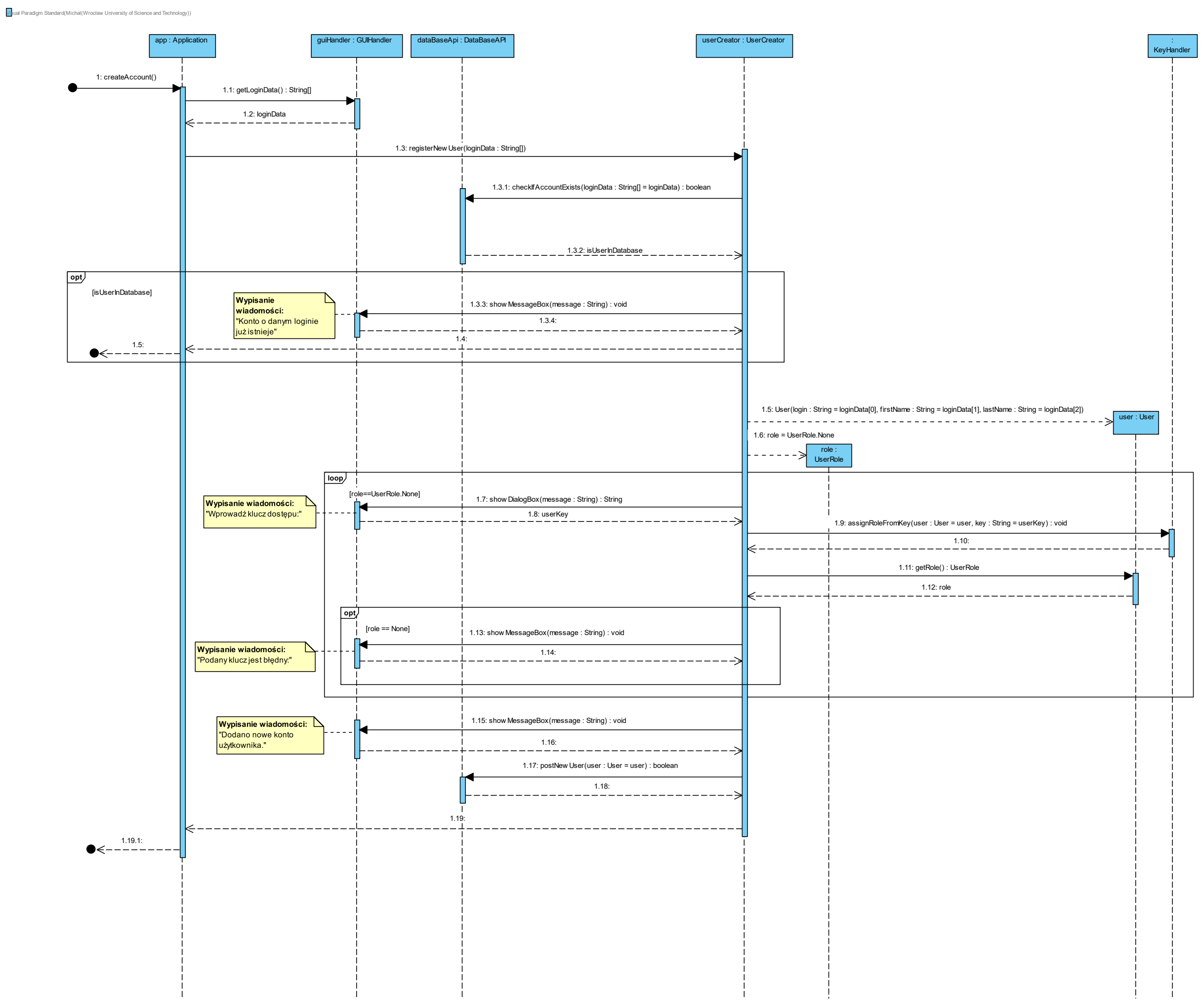
Kod metody `placeNewOrder` z klasy `Application` zamodelowanej przez diagram sekwencji:

```
public void placeNewOrder() {
      String[] formData = guiHandler.getFormData();
      orderCreator.createNewOrder(formData, currentUser);
}
```

Kod metody `createNewOrder` z klasy `OrderCreator` zamodelowanej przez diagram sekwencji:

```
public void createNewOrder(String[] formData, User user) {
        var commodity = new Commodity(
            formData[0],
            Double.parseDouble(formData[1]),
            Double.parseDouble(formData[2]),
            Double.parseDouble(formData[3])
        );

        var order = new Order(
            commodity,
            user,
            formData[4],
            formData[5],
            formData[6]
        );

        boolean doesOrderExists = dataBaseApi.checkIfOrderExists(order);
        if(doesOrderExists)
        {
            String userInteraction = guiHandler.showDialogBox(
                  "Zamówienie o takich parametrach już istnieje. Czy chcesz
                  kontynuować składanie zamówienia?"
            );

            if(userInteraction.contains("NO"))
            {
                return;
            }
        }

        int nextOrderId = dataBaseApi.getNextOrderId();
        order.setId(nextOrderId);

        boolean isSuccess = dataBaseApi.postNewOrder(order);
        if(isSuccess)
        {
            guiHandler.showMessageBox("Dodano nowe zamówienie");
        }
        else
        {
            guiHandler.showMessageBox("Nie udało się dodać zamówienia");
        }
}
```

Diagram sekwencji metody „createAccount” (PU „Załozenie konta”):


Visual Paradigm Standard(Michal(Wrocław University of Science and Technology))
app : Application
guiHandler : GUIHandler
dataBaseApi : DataBaseAPI
userCreator : UserCreator
: KeyHandler
1: createAccount()
1.1: getLoginData() : String[]
1.2: loginData
1.3: registerNewUser(loginData : String[])
1.3.1: checkIfAccountExists(loginData : String[] = loginData) : boolean
1.3.2: isUserInDatabase
opt
[isUserInDatabase]
Wypisanie wiadomości: "Konto o danym loginie już istnieje"
1.3.3: showMessageBox(message : String) : void
1.3.4:
1.4:
1.5:
1.5: User(login : String = loginData[0], firstName : String = loginData[1], lastName : String = loginData[2])
user : User
1.6: role = UserRole.None
role : UserRole
loop
[role==UserRole.None]
Wypisanie wiadomości: "Wprowadź klucz dostępu:"
1.7: showDialogBox(message : String) : String
1.8: userKey
1.9: assignRoleFromKey(user : User = user, key : String = userKey) : void
1.10:
1.11: getRole() : UserRole
1.12: role
opt
[role == None]
Wypisanie wiadomości: "Podany klucz jest błędny:"
1.13: showMessageBox(message : String) : void
1.14:
Wypisanie wiadomości: "Dodano nowe konto użytkownika."
1.15: showMessageBox(message : String) : void
1.16:
1.17: postNewUser(user : User = user) : boolean
1.18:
1.19:
1.19.1:

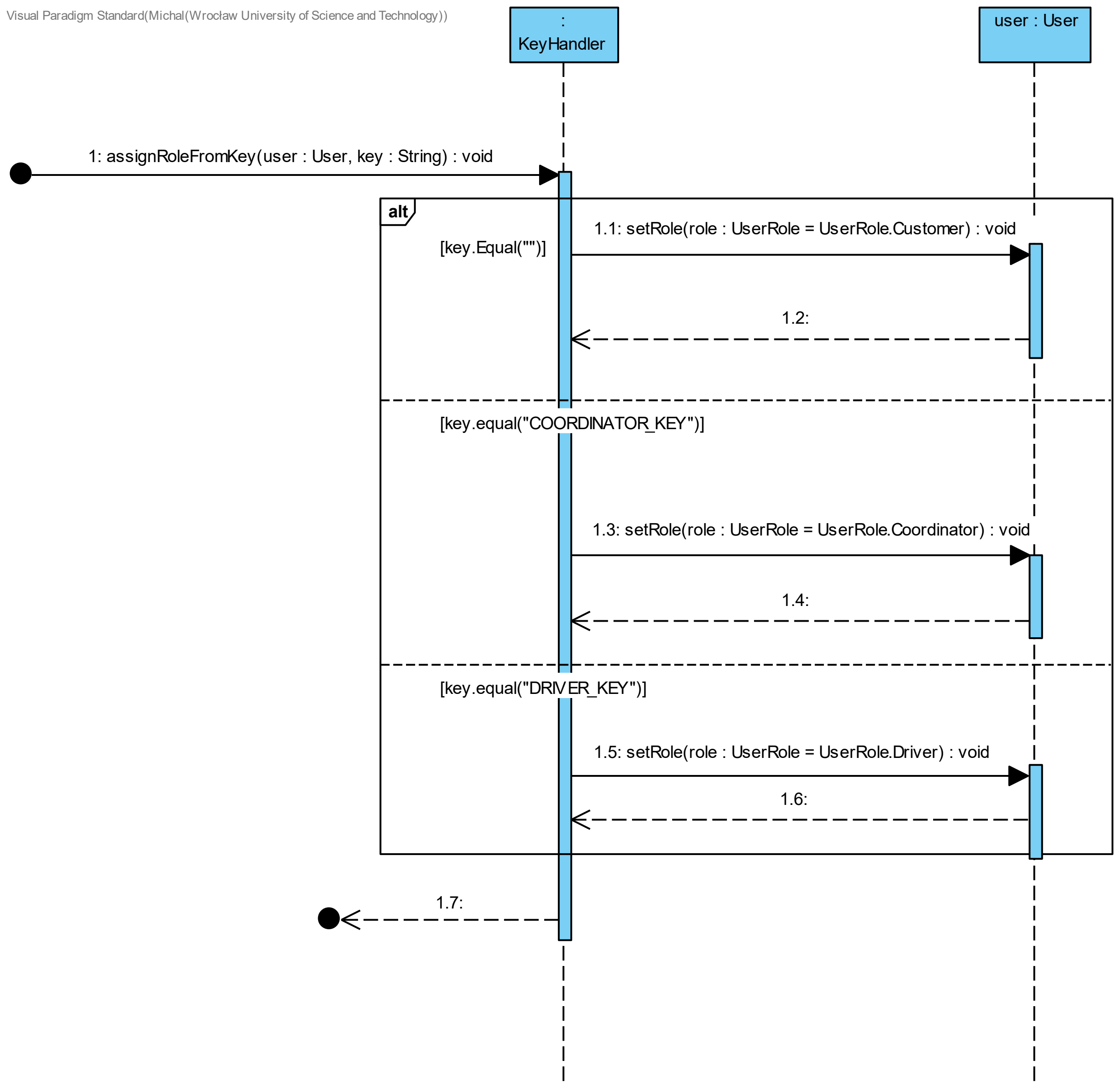
Kod metody `createAccount` z klasy `Application` zamodelowanej przez diagram sekwencji:

```java
public void createAccount() {
	String[] loginData = guiHandler.getLoginData();
	userCreator.registerNewUser(loginData);
}
```

Kod metody `registerNewUser` z klasy `UserCreator` zamodelowanej przez diagram sekwencji:

```
public void registerNewUser(String[] loginData) {
      boolean isUserInDatabase =
dataBaseApi.checkIfAccountExists(loginData);
      if(isUserInDatabase) {
            guiHandler.showDialogBox("Konto o danym loginie już istnieje");
            return;
      }

      User user = new User(loginData[0], loginData[1], loginData[2]);
      UserRole role = UserRole.None;

      while (role == UserRole.None) {
            String userKey = guiHandler.showDialogBox("Wprowadź klucz
dostępu.");
            KeyHandler.assignRoleFromKey(user, userKey);
            role = user.getRole();
            if(role == UserRole.None)
                  guiHandler.showMessageBox("Podany klucz jest błędny.");
      }

      guiHandler.showMessageBox("Dodano nowe konto użytkownika!");
      dataBaseApi.postNewUser(user);
}
```

Dla czytelności z diagramu modelującego metodę `createAccount` wydzielono do osobnego diagramu sekwencji metodę `assignRoleFromKey` z klasy `KeyHandler`.


Visual Paradigm Standard(Michal(Wrocław University of Science and Technology))
: KeyHandler
user : User
1: assignRoleFromKey(user : User, key : String) : void
alt
[key.Equal("")]
1.1: setRole(role : UserRole = UserRole.Customer) : void
1.2:
[key.equal("COORDINATOR_KEY")]
1.3: setRole(role : UserRole = UserRole.Coordinator) : void
1.4:
[key.equal("DRIVER_KEY")]
1.5: setRole(role : UserRole = UserRole.Driver) : void
1.6:
1.7:

Kod metody `assignRoleFromKey` z klasy `KeyHandler` zamodelowanej przez diagram sekwencji:

```
static void assignRoleFromKey(User user, String key) {
      if(key.equals("")){
            user.setRole(UserRole.Customer);
      }
      else if(key.equals(COORDINATOR_KEY)){
            user.setRole(UserRole.Coordinator);
      }
      else if(key.equals(DRIVER_KEY)){
            user.setRole(UserRole.Driver);
      }
      return;
}
```