GIT Department of Computer Engineering
CSE 222/505 - Spring 2020
Homework 1 – Part 2 Report

Buğra Eren Yılmaz
1801042669

1. System Requirements

I did not change much of the project since HW1-Part 1. Only changes are listed as

Used ArrayList instead of a custom made array container for DbSets
UserAccess is now more object oriented.
DbSet-Controller is now more object oriented

Plus I seperated the Main class from my own package. I did forget this in the HW1-Part 1. Because of that, project compiles easier now. Just use:

javac Main.java

For javadoc generation you can use following command, even though I pre-generated the docs.

javadoc -d outputDirectory ./com/pionix/*.java

From the problem definition we see that we need a Log-in system for user authentication to permit user to manipulate parts of the “Database”.

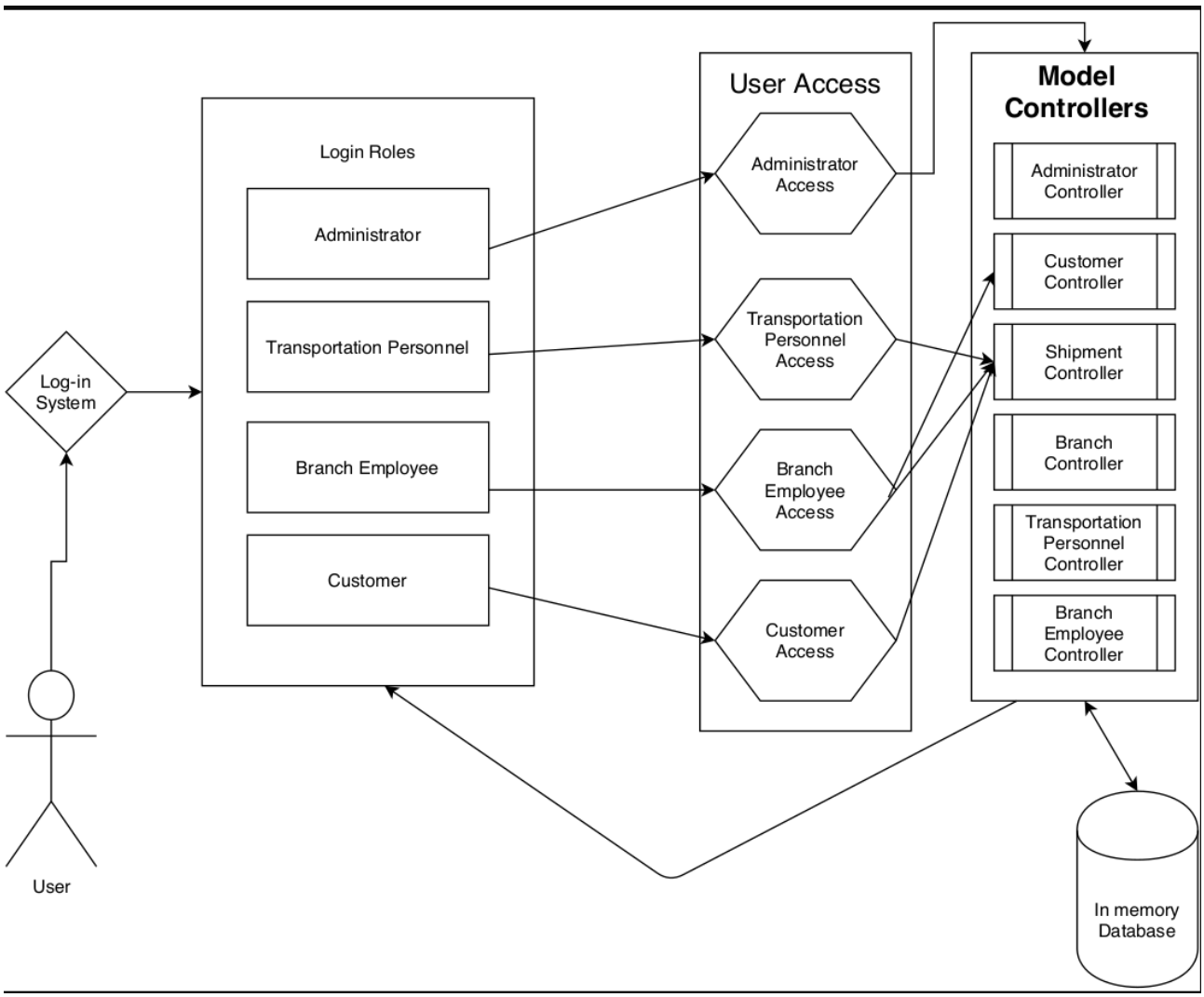
Here I refer the in-memory fake database system as Database. It utilizes the database object sets with the DbSet and DbController system which I designed.

The system is similar to MVC Architecture. I tried to implement it primarily using Dependency Injection techniques.

2. Use Case Diagrams

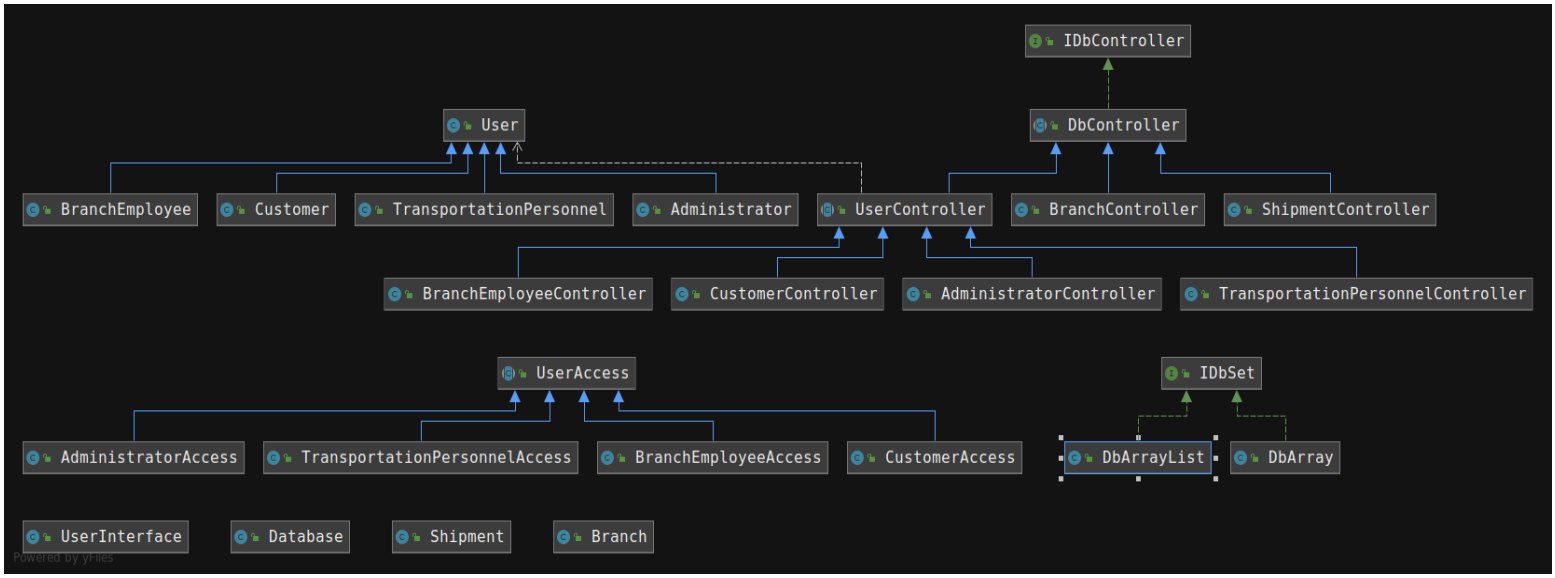
The user can login as 4 roles. Every role has different access levels to Model Controllers. Model Controllers control and manipulates the Models in Database.

For example a Branch Employee has UserAccess of a BranchEmployeeAccess. Which this access level has the access to ShipmentController and CustomerController. This way a BranchEmployee can manipulate shipments and customers in database.

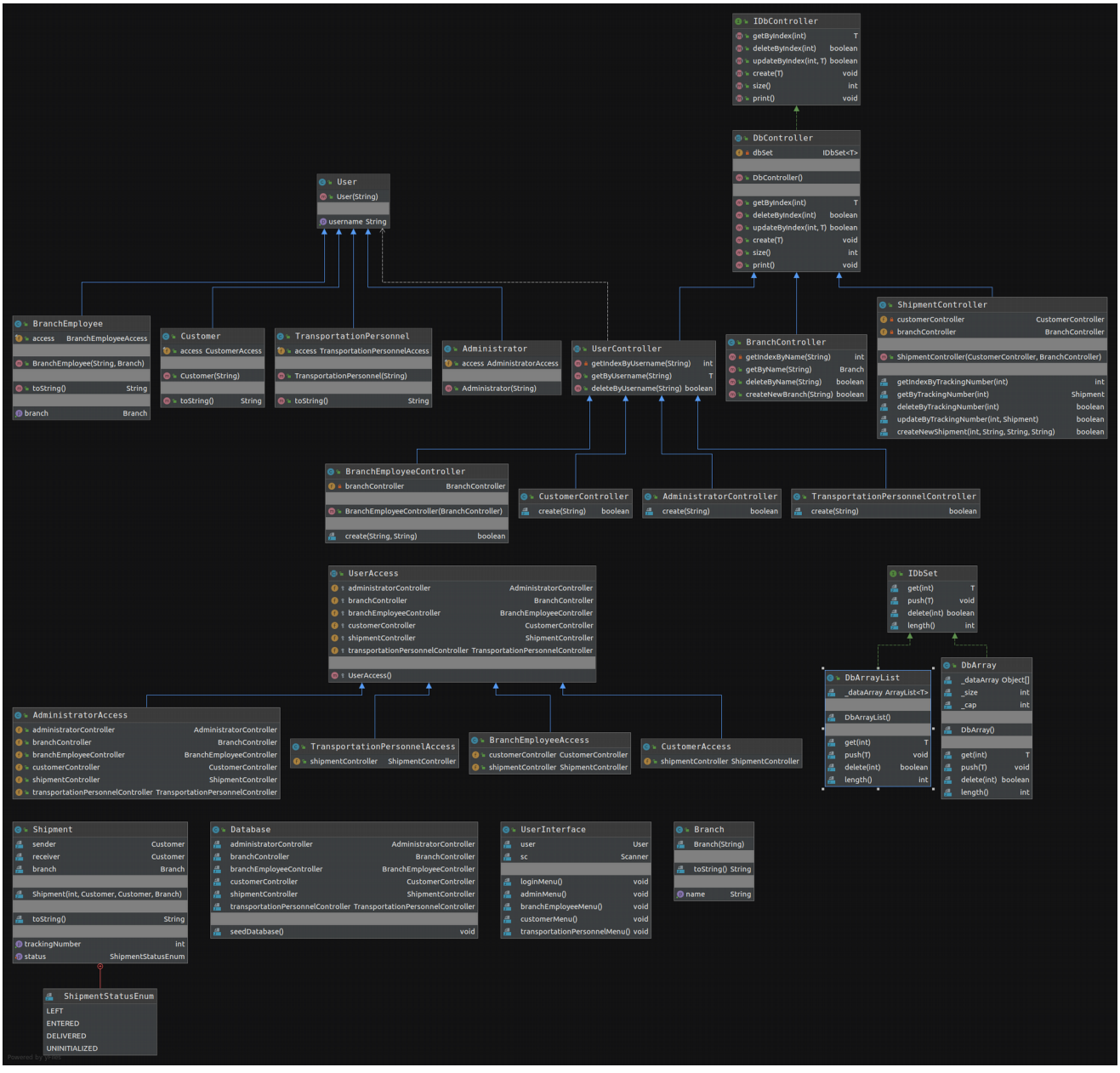


3. Class Diagrams

Minimal Class UML



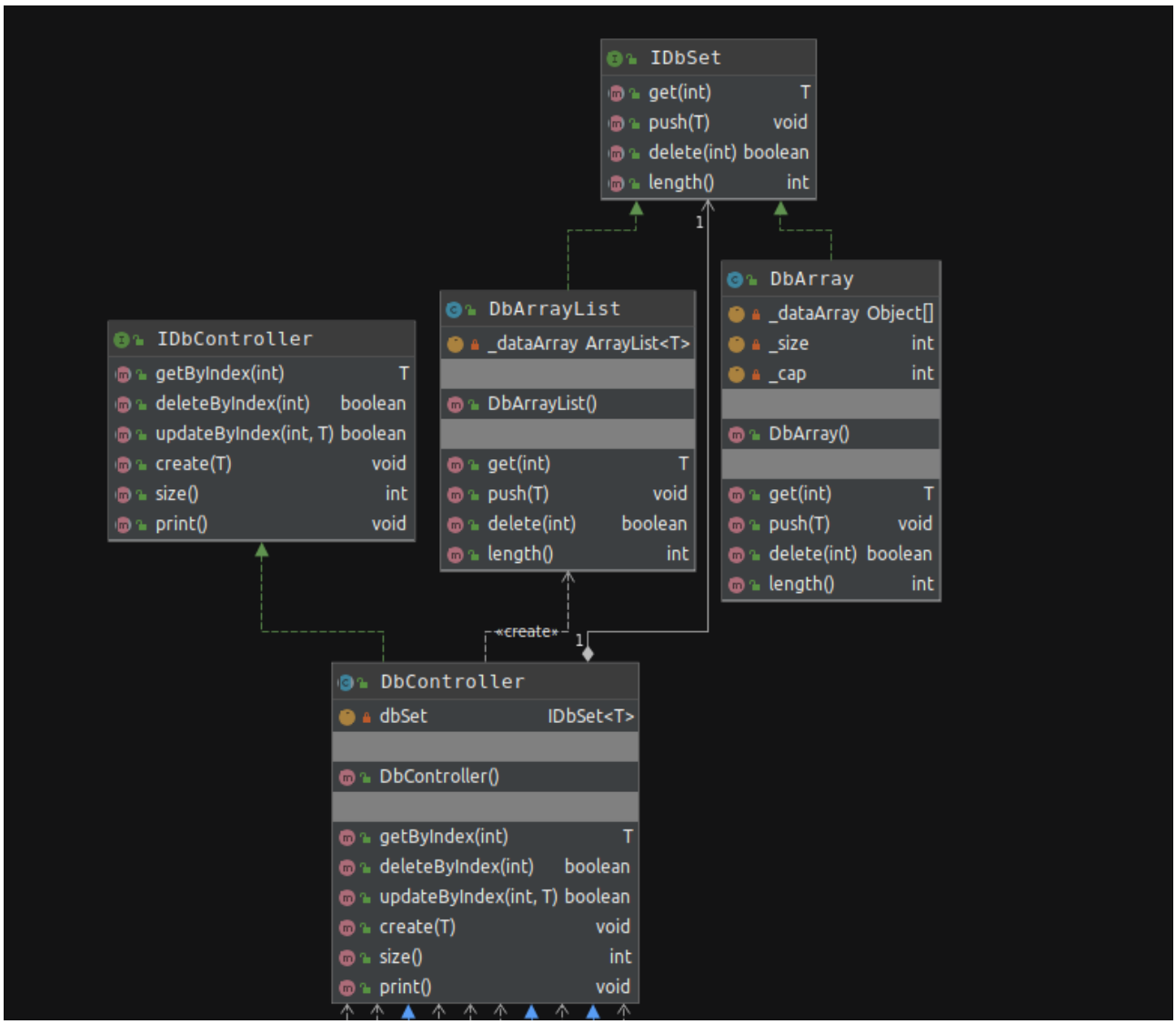
Full Class UML



3.1 Database MVC Design Diagrams

Database consists of 2 main things, DbSet and DbController. Every DbController has a DbSet. DbControllers controls and manipulates the data inside the DbSets. Currently there are two DbSet implementations available, one is DbArray which is a custom Array container. The new one is DbArrayList which is an ArrayList container.

DbSets are like a table in SQL. DbControllers are like pre-defined SQL scripts. This way every time I create a controller, I am creating a new table.



4.1 Mock-Database Design

To mimic a real SQL database, I made a class named Database and stored the DbControllers inside it as Singleton class members. Database is also a singleton class.

```java
public class Database {
    /**
     * AdministratorController holds the Table for Administrator objects.
     * Has the actions for it
     */
    public static AdministratorController administratorController = new AdministratorController();
    /**
     * BranchController holds the Table for Branch objects.
     * Has the actions for it
     */
    public static BranchController branchController = new BranchController();
    /**
     * BranchEmployeeController holds the Table for BranchEmployee objects.
     * Has the actions for it
     */
    public static BranchEmployeeController branchEmployeeController = new BranchEmployeeController(branchController);
    /**
     * CustomerController holds the Table for Customer objects.
     * Has the actions for it
     */
    public static CustomerController customerController = new CustomerController();
    /**
     * ShipmentController holds the Table for Shipment objects.
     * Has the actions for it
     */
    public static ShipmentController shipmentController = new ShipmentController(customerController, branchController);
    /**
     * TransportationController holds the Table for Transportation objects.
     * Has the actions for it
     */
    public static TransportationPersonnelController transportationPersonnelController = new TransportationPersonnelController();

    /**
     * Seeds the database with some initial data for every model
     */
    public static void seedDatabase() {
```

4.2 User Access Hierarchy Design

I created a base abstract class, UserAccess, which holds every available controller in project and initializes these controllers from Database class singletons.

```
public abstract class UserAccess {
    /**
     * access to <code>Administrator Model<code/>
     */
    protected AdministratorController administratorController;
    /**
     * access to <code>Branch Model<code/>
     */
    protected BranchController branchController;
    /**
     * access to <code>BranchEmployee Model<code/>
     */
    protected BranchEmployeeController branchEmployeeController;
    /**
     * access to <code>Customer Model<code/>
     */
    protected CustomerController customerController;
    /**
     * access to <code>Shipment Model<code/>
     */
    protected ShipmentController shipmentController;
    /**
     * access to <code>TransportationPersonnel Model<code/>
     */
    protected TransportationPersonnelController transportationPersonnelController;

    /**
     * Dependency Inject required <code>DbControllers</code>
     */
    protected UserAccess() {
        administratorController = Database.administratorController;
        branchController = Database.branchController;
        branchEmployeeController = Database.branchEmployeeController;
        customerController = Database.customerController;
        shipmentController = Database.shipmentController;
        transportationPersonnelController = Database.transportationPersonnelController;
    }
}
```

In the derived Access Classes, I stretched the access modifiers for these controllers according to wanted access level. These way I can change every modifier in derived class and have access levels.

```
public class BranchEmployeeAccess extends UserAccess{
    /**
     * BranchEmployee has access to <code>Customer Model<code/>
     */
    public CustomerController customerController = super.customerController;
    /**
     * BranchEmployee has access to <code>Shipment Model<code/>
     */
    public ShipmentController shipmentController = super.shipmentController;
}
```

4.3 Problem Solution Approach

Problem was requiring me to somehow authorize users and restrict their access to the Database according to their roles. So I tried to make a system like MVC database architecture.

I started with building access units to database for every model. First I write every available model. Then made a Database Set class **DbSet** which holds the data as an ArrayList, can delete, add, get the data.

After that I made a Database controller class **DbController** which utilizes the given **DbSet** to manipulate the data inside it more.

So these classes are all abstract and we need the actual controllers so I started writing down every controller for every available model. Generally every model needs a specific way of get, set, delete and update.

For example **Shipment** model needs to be created with a tracking number, sender, receiver and an operating branch. So when creating this model I need to check if the sender is actually available in **Customers** database or the branch is an actual branch. These type of data checks are made in the controllers.

After writing controllers, I write access classes for roles. For instance Branch Employee person role needs to access Shipment and Customer controllers so I give that access to these access classes, BranchEmployeeAccess.

To Mimic a real life database I made the Database class which has the seedData function and Controllers as **Singleton classes.**

After that it was all about dependency injecting these access classes into Role classes. For example injectin BranchEmployeeAccess class into BranchEmployee class.

5. Test Cases

5.1 UserAccess Test Cases

It is impossible for a user to do illegal operation because every user has access-member-class which restricts the accessed DbControllers for the user, whatever the user is a Customer or an Admin, the restrictions are met with-in the code level.

```
/**
 * Test cases for access of user roles to database controllers
 */
public static void accessTests() {
    User user;

    // Get the BranchEmployee named Mehmet from database
    user = Database.branchEmployeeController.getByUsername("Mehmet");

    // Down-case user object to role
    BranchEmployee employee = (BranchEmployee) user;

    // Access the employees abilities
    // Just examples
    // Wont create anything because there is no such customer named Foo and Goo
    // No such branch named FooBranch
    employee.access.customerController.getByUsername("Foo");

    // No such tracking number
    employee.access.shipmentController.getByTrackingNumber(123);

    // Get the TPEmployee named Ali from database
    user = Database.transportationPersonnelController.getByUsername("Ali");

    // Down-case user object to role
    TransportationPersonnel tPersonnel = (TransportationPersonnel) user;

// No such tracking number
employee.access.shipmentController.getByTrackingNumber(123);

employee.access.
// Get the TP
user = Databa
// Down-case
Transportatio
// Access the
tPersonnel.ac
user = Databa
Administrator
```

```
shipmentController                          ShipmentController
customerController                          CustomerController
equals(Object obj)                                    boolean
hashCode()                                                int
toString()                                           String
getClass()          Class<? extends BranchEmployeeAccess>
notify()                                              void
notifyAll()                                           void
wait()                                                void
wait(long l)                                          void
wait(long timeoutMillis, int nanos)                   void
Ctrl+Down and Ctrl+Up will move caret down and up in the editor  Next Tip
```

5.2 Model Creation Test Cases

First of, start with getting users from database.

```
public static void creationTests() {
    Administrator admin = Database.administratorController.getByUsername("Pionix");
    BranchEmployee employee = Database.branchEmployeeController.getByUsername("Mehmet");
    TransportationPersonnel personnel = Database.transportationPersonnelController.getByUsername("Ali");
    Customer customer = Database.customerController.getByUsername("Hakan");
```

Try to create a shipment with non-existent customers as sender and receiver, it will fail.

```
// Should fail because
// No customer found named Foo and Goo
// Operating branch is found though
success = employee.access.shipmentController.createNewShipment(100, "Foo", "Goo", "Ankara");
if(success) System.out.println("creationTest 1 Failed!");
```

Now create those customers and try to create same shipment. It will succeed.

```
// Create foo and goo customers, employee can do that
success = employee.access.customerController.create("Foo");
success = employee.access.customerController.create("Goo");

// Now it should succeed
success = employee.access.shipmentController.createNewShipment(100, "Foo", "Goo", "Ankara");
if(!success) System.out.println("creationTest 2 Failed!");
```

Customer can see status of created shipment.

```
// Customer can see status of cargo
success = null != customer.access.shipmentController.getByTrackingNumber(100);
if(!success) System.out.println("creationTest 3 Failed!");
```

Transportation Personnel can deliver the shipment.

```
// Transportation personnel can deliver cargo
Shipment shipment = personnel.access.shipmentController.getByTrackingNumber(100);
shipment.setStatus(Shipment.ShipmentStatusEnum.DELIVERED);
success = personnel.access.shipmentController.updateByTrackingNumber(100, shipment);
if(!success) System.out.println("creationTest 4 Failed!");
```

Try to create a new employee in non-existent branch. It will fail.

```
// Admin can create new employee
// This should fail since no branch named foo is found
success = admin.access.branchEmployeeController.create("New Employee", "Foo");
if(success) System.out.println("creationTest 5 Failed!");
```

Now create that branch and try to create the employee again. It will succeed.

```
// Now create foo branch
// Notice that it can have the name 'Foo'
// Even tho we have a customer named 'Foo'
// This can be achieved because every model
// is stored in separate data
success = admin.access.branchController.createNewBranch("Foo");
if(!success) System.out.println("creationTest 6 Failed!");

// Now create that employee
success = admin.access.branchEmployeeController.create("New Employee", "Foo");
if(!success) System.out.println("creationTest 7 Failed!");
```

These were basic demonstration of manipulating data with authorized personnel. As you can see there is no way to a user to do unauthorized operation because it is handled at the code-level.

For example a customer can not create a branch employee because you can not write it in code. There is no such thing as:

> customer.access.branchEmployeeController.create(..)
> // wont compile because customers access do not have branchEmployee Controller

But an admin can create a branchEmployee as following:

> admin.access.branchEmployeeController.create(..)
> // will work because admins access has branchEmployeeController

5.3 Name Duplication Test Cases

Create a customer named CUSTOMER1

```
// create CUSTOMER1
success = admin.access.customerController.create("CUSTOMER1");
if(!success) System.out.println("nameDuplicationTest failed!");
```

Try to create it again, it should fail.

```
// try to create it again
// it should fail
success = admin.access.customerController.create("CUSTOMER1");
if(success) System.out.println("nameDuplicationTest failed!");
```

Try to create a transport personnel named CUSTOMER1, it should pass.

```
// try to create a transport personnel named CUSTOMER1
// it should pass
// this is achieved by storing every model in separate data sets
// even though a Customer and a TransportPersonnel are both derived
// from User class, their storage are not at the same place!
success = admin.access.transportationPersonnelController.create("CUSTOMER1");
if(!success) System.out.println("nameDuplicationTest failed!");
```

As you can see, even though a Customer and TransportationPersonnel are both derived from User class, their storage systems are separate.

That is because of the patter I designed.

The UserController is a generic class and it takes type of the user as template. Thus we have different controllers for different user types. This means different storage places.

Name duplication can be demonstrated with any other model. If a models name is already in that DbSet, you can not create it again.

By the name I meant an Unique Id for that model. It can be easily converted to any field.

All those tests are available in Main class

6. Running and Results

Running and results are same with the previous part of the homework. Nothing changed UI wise.

Admin menu example

```
└─ make run
java -cp ./build com.pionix.Main;
====== LOG-IN ======
Example users: Admin(Pionix), BranchEr
Check the seeder of database at Databa
1- Admin
2- Branch Employee
3- Transportation Employee
4- Customer
5- Exit the system
1
Username: Pionix
====== ADMIN ======
1- Add Branch
2- Remove Branch
3- List Branches
4- Add Branch Employee
5- Remove Branch Employee
6- List Branch Employees
7- Add Transportation Employee
8- Remove Transportation Personnel
9- List Transportation Personnel
10- List Shipments
11- List Customers
0- Logout
3
Branch name: Ankara
Branch name: Istanbul
Branch name: Izmir
Branch name: Adana
Operation was successful!
```

Trying to add a duplicate branch

```
====== ADMIN ======
1- Add Branch
2- Remove Branch
3- List Branches
4- Add Branch Employee
5- Remove Branch Employee
6- List Branch Employees
7- Add Transportation Employee
8- Remove Transportation Personnel
9- List Transportation Personnel
10- List Shipments
11- List Customers
0- Logout
1
Enter name of the new branch: Ankara
Branch name duplicate when creating new branch!
Operation was NOT successful!
```

Adding branch employee to existing branch

```
====== ADMIN ======
1- Add Branch
2- Remove Branch
3- List Branches
4- Add Branch Employee
5- Remove Branch Employee
6- List Branch Employees
7- Add Transportation Employee
8- Remove Transportation Personnel
9- List Transportation Personnel
10- List Shipments
11- List Customers
0- Logout
4
Enter username of the new employee: Batu
Enter name of the branch to station employee: Giresun
Operation was successful!
```

```
====== ADMIN ======
1- Add Branch
2- Remove Branch
3- List Branches
4- Add Branch Employee
5- Remove Branch Employee
6- List Branch Employees
7- Add Transportation Employee
8- Remove Transportation Personnel
9- List Transportation Personnel
10- List Shipments
11- List Customers
0- Logout
1
Enter name of the new branch: Giresun
Operation was successful!

====== ADMIN ======
1- Add Branch
2- Remove Branch
3- List Branches
4- Add Branch Employee
5- Remove Branch Employee
6- List Branch Employees
7- Add Transportation Employee
8- Remove Transportation Personnel
9- List Transportation Personnel
10- List Shipments
11- List Customers
0- Logout
3
Branch name: Ankara
Branch name: Istanbul
Branch name: Izmir
Branch name: Adana
Branch name: Giresun
Operation was successful!
```

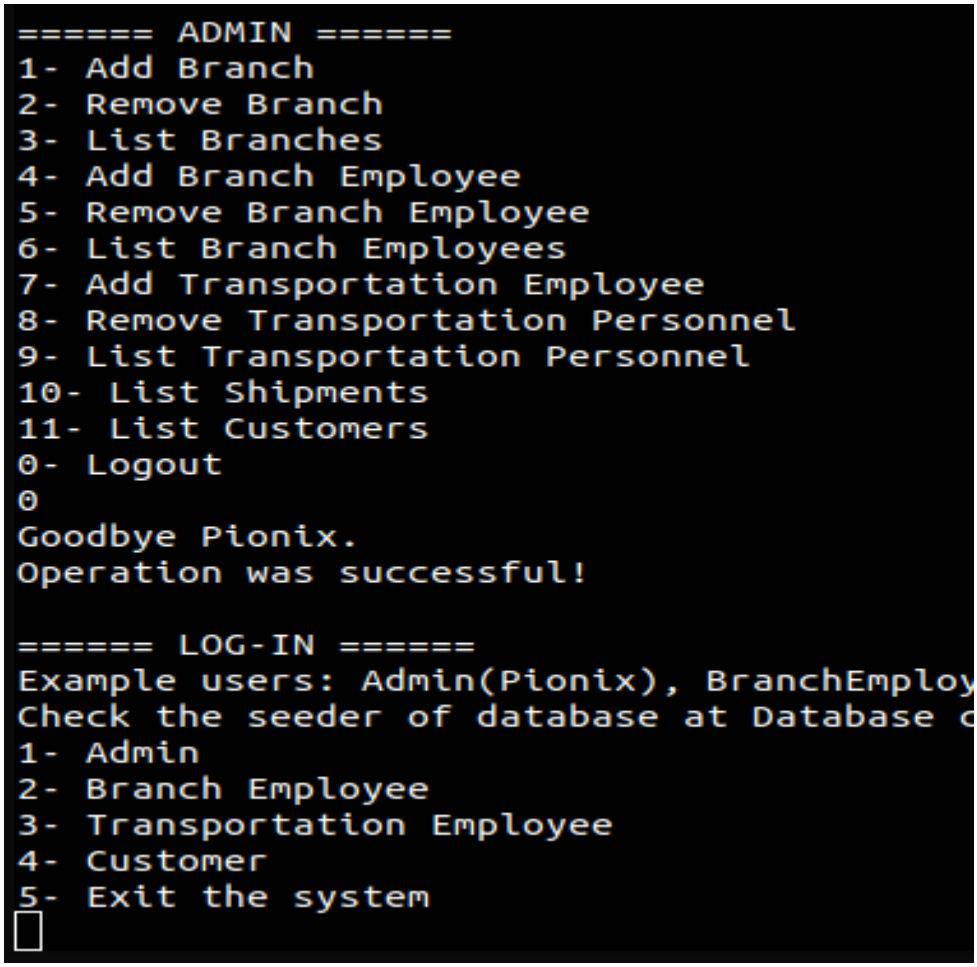
Trying to add a employee to non existing branch

```
====== ADMIN ======
1- Add Branch
2- Remove Branch
3- List Branches
4- Add Branch Employee
5- Remove Branch Employee
6- List Branch Employees
7- Add Transportation Employee
8- Remove Transportation Personnel
9- List Transportation Personnel
10- List Shipments
11- List Customers
0- Logout
4
Enter username of the new employee: JohnDoe
Enter name of the branch to station employee: NonExistin
Branch not found when creating new Branch Employee!
Operation was NOT successful!
```

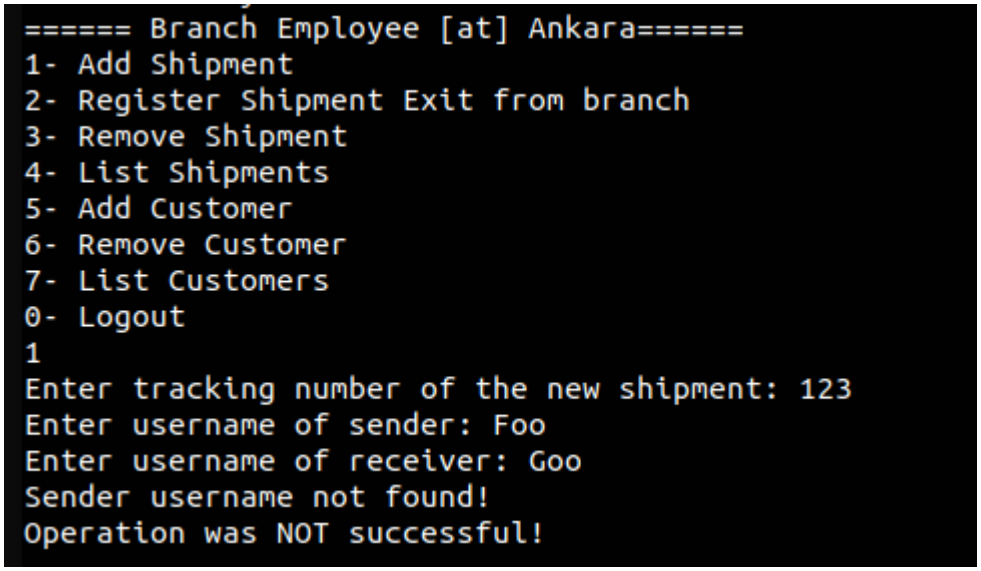
Listing shipments

```
====== ADMIN ======
1- Add Branch
2- Remove Branch
3- List Branches
4- Add Branch Employee
5- Remove Branch Employee
6- List Branch Employees
7- Add Transportation Employee
8- Remove Transportation Personnel
9- List Transportation Personnel
10- List Shipments
11- List Customers
0- Logout
10
100001: [Hakan] --> [Fatma]
        shipment entered the Ankara branch.
100002: [Asya] --> [Kerem]
        shipment entered the Izmir branch.
Operation was successful!
```

Logging out



Branch Employee Menu and trying to add shipment with non existing customers



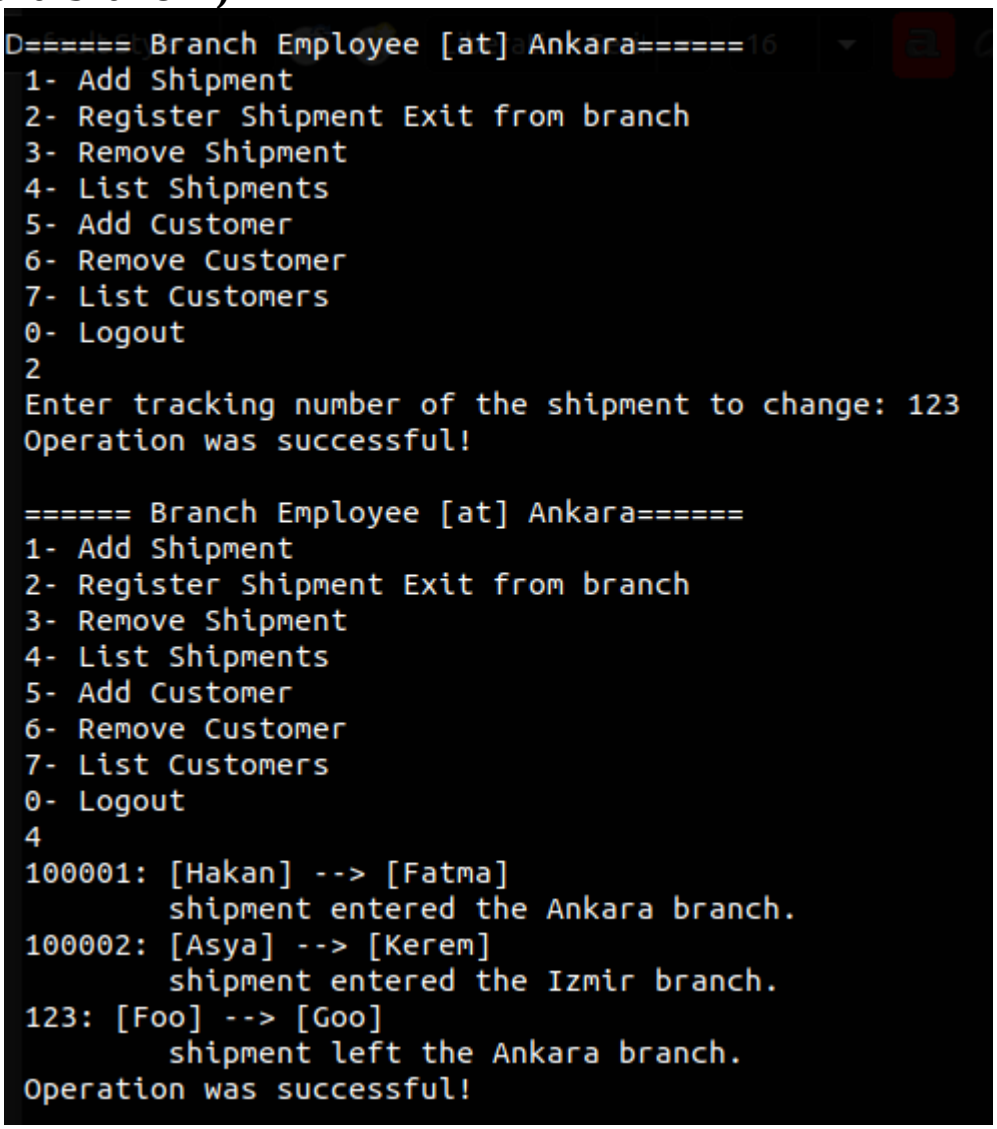
Adding the non existing customers

```
====== Branch Employee [at] Ankara======
1- Add Shipment
2- Register Shipment Exit from branch
3- Remove Shipment
4- List Shipments
5- Add Customer
6- Remove Customer
7- List Customers
0- Logout
5
Enter username of the new customer: Goo
Operation was successful!

====== Branch Employee [at] Ankara======
1- Add Shipment
2- Register Shipment Exit from branch
3- Remove Shipment
4- List Shipments
5- Add Customer
6- Remove Customer
7- List Customers
0- Logout
7
Customer name: Kerem
Customer name: Eren
Customer name: Asya
Customer name: Fatma
Customer name: Hakan
Customer name: Foo
Customer name: Goo
Operation was successful!
```

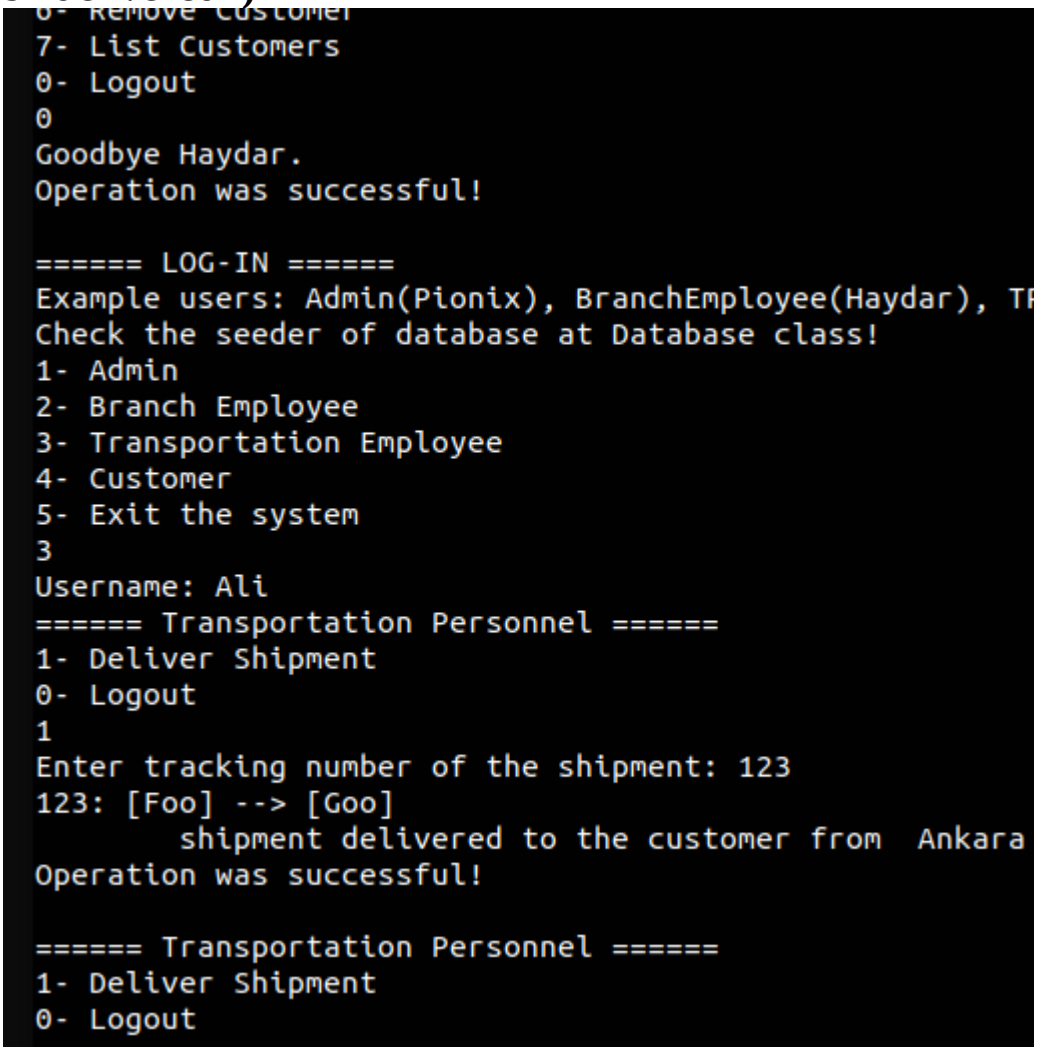
Creating a shipment (its status is “entered the ankara branch”)

```
====== Branch Employee [at] Ankara======
1- Add Shipment
2- Register Shipment Exit from branch
3- Remove Shipment
4- List Shipments
5- Add Customer
6- Remove Customer
7- List Customers
0- Logout
1
Enter tracking number of the new shipment: 123
Enter username of sender: Foo
Enter username of receiver: Goo
Operation was successful!

====== Branch Employee [at] Ankara======
1- Add Shipment
2- Register Shipment Exit from branch
3- Remove Shipment
4- List Shipments
5- Add Customer
6- Remove Customer
7- List Customers
0- Logout
4
100001: [Hakan] --> [Fatma]
        shipment entered the Ankara branch.
100002: [Asya] --> [Kerem]
        shipment entered the Izmir branch.
123: [Foo] --> [Goo]
        shipment entered the Ankara branch.
Operation was successful!
```

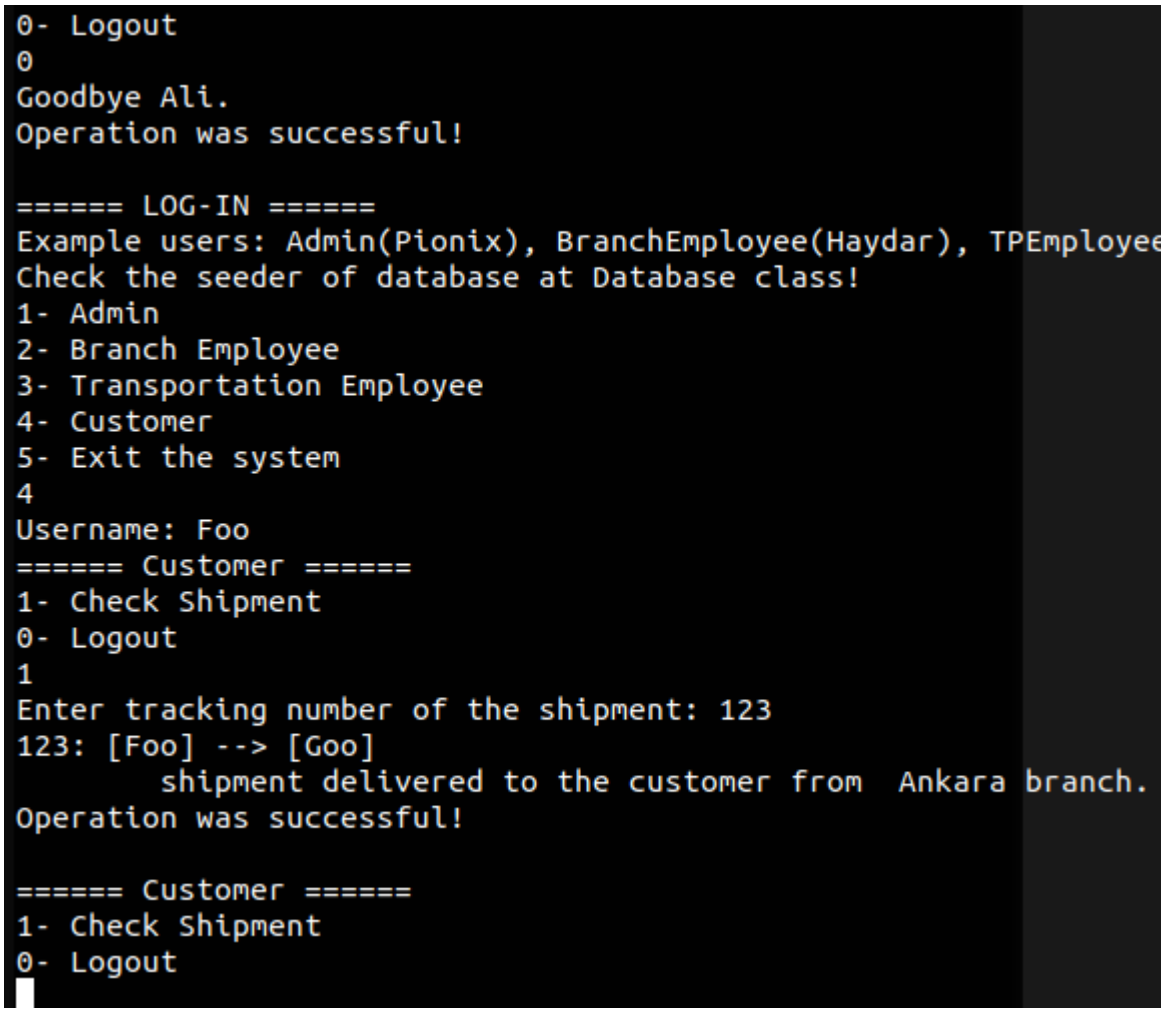
Sending shipment out to transportation personnel (now status is “left the ankara branch”)



Logout and login as transportation personnel and deliver cargo(now status is “delivered”)



Logout and login as customer and check the cargo status



Logout and exit system