README.md

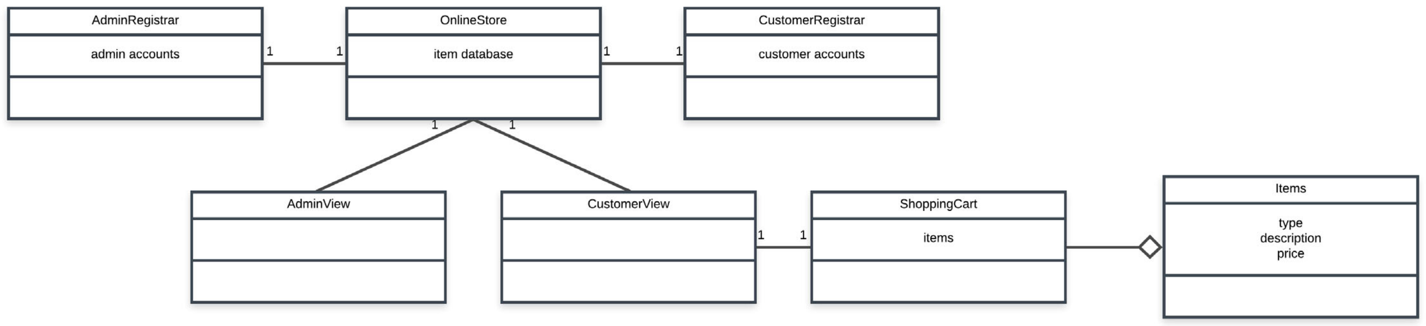
Assignment 3

Brady Dean
CSCI 450

StoreServer

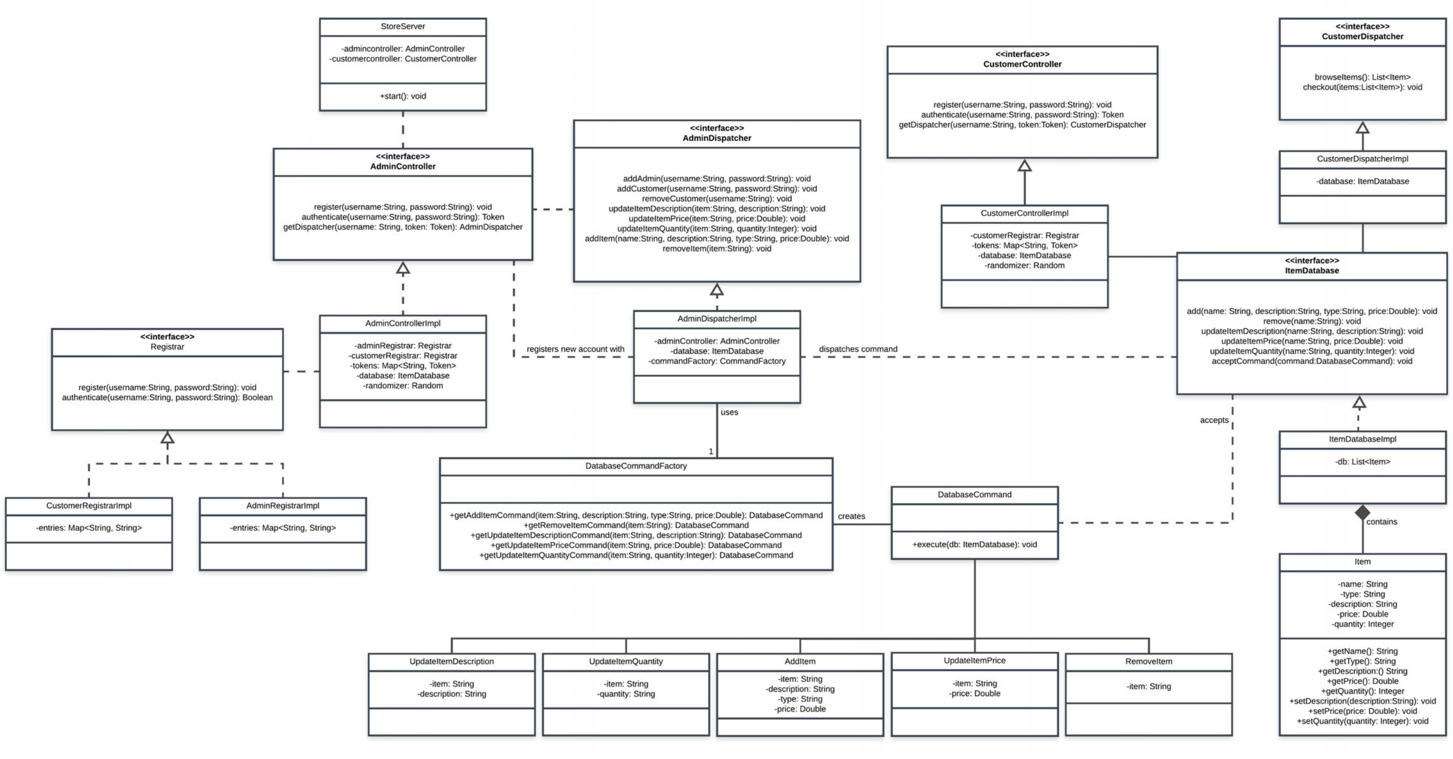
StoreServer is the name of my online store service. The system consists of one server backend, and many client programs. Client programs and the backend communicate through Remote Method Invocation.

Domain Model



- `OnlineStore` - Online Store Service
- `AdminRegistrar` - Admin account database
- `CustomerRegistrar` - Customer account database
- `AdminView` - Admin view of system
- `CustomerView` - Customer view of system
- `ShoppingCart` - Shopping cart associated with customer
- `Items` - Items available for purchase

Class Diagram


StoreServer
-admincontroller: AdminController
-customercontroller: CustomerController
+start(): void
<<interface>>
AdminController
register(username:String, password:String): void
authenticate(username:String, password:String): Token
getDispatcher(username: String, token: Token): AdminDispatcher
<<interface>>
AdminDispatcher
addAdmin(username:String, password:String): void
addCustomer(username:String, password:String): void
removeCustomer(username:String): void
updateItemDescription(item:String, description:String): void
updateItemPrice(item:String, price:Double): void
updateItemQuantity(item:String, quantity:Integer): void
addItem(name:String, description:String, type:String, price:Double): void
removeItem(item:String): void
<<interface>>
CustomerController
register(username:String, password:String): void
authenticate(username:String, password:String): Token
getDispatcher(username:String, token:Token): CustomerDispatcher
<<interface>>
CustomerDispatcher
browseItems(): List<Item>
checkout(items:List<Item>): void
CustomerDispatcherImpl
-database: ItemDatabase
CustomerControllerImpl
-customerRegistrar: Registrar
-tokens: Map<String, Token>
-database: ItemDatabase
-randomizer: Random
<<interface>>
ItemDatabase
add(name: String, description:String, type:String, price:Double): void
remove(name:String): void
updateItemDescription(name:String, description:String): void
updateItemPrice(name:String, price:Double): void
updateItemQuantity(name:String, quantity:Integer): void
acceptCommand(command:DatabaseCommand): void
<<interface>>
Registrar
register(username:String, password:String): void
authenticate(username:String, password:String): Boolean
AdminControllerImpl
-adminRegistrar: Registrar
-customerRegistrar: Registrar
-tokens: Map<String, Token>
-database: ItemDatabase
-randomizer: Random
registers new account with
AdminDispatcherImpl
-adminController: AdminController
-database: ItemDatabase
-commandFactory: CommandFactory
dispatches command
uses
1
accepts
ItemDatabaseImpl
-db: List<Item>
CustomerRegistrarImpl
-entries: Map<String, String>
AdminRegistrarImpl
-entries: Map<String, String>
DatabaseCommandFactory
+getAddItemCommand(item:String, description:String, type:String, price:Double): DatabaseCommand
+getRemoveItemCommand(item:String): DatabaseCommand
+getUpdateItemDescriptionCommand(item:String, description:String): DatabaseCommand
+getUpdateItemPriceCommand(item:String, price:Double): DatabaseCommand
+getUpdateItemQuantityCommand(item:String, quantity:Integer): DatabaseCommand
creates
DatabaseCommand
+execute(db: ItemDatabase): void
contains
Item
-name: String
-type: String
-description: String
-price: Double
-quantity: Integer
+getName(): String
+getType(): String
+getDescription:() String
+getPrice(): Double
+getQuantity(): Integer
+setDescription(description:String): void
+setPrice(price: Double): void
+setQuantity(quantity: Integer): void
UpdateItemDescription
-item: String
-description: String
UpdateItemQuantity
-item: String
-quantity: String
AddItem
-item: String
-description: String
-type: String
-price: Double
UpdateItemPrice
-item: String
-price: Double
RemoveItem
-item: String

Interfaces:

- `AdminController` - Administrator front-controller entry point, also RMI interface. Handles authentication
- `CustomerController` - Customer front-controller entry point, also RMI interface. Handles authentication. **New in project 3**
- `Registrar` - Account database, also RMI interface. Handles authentication
- `AdminDispatcher` - Administrator front-controller that accepts commands once authenticated, also RMI interface
- `CustomerDispatcher` - Customer front-controller that accepts commands once authenticated, also RMI interface. **New in project 3**
- `ItemDatabase` - Defined commands on the item database, also RMI interface. Accepts `DatabaseCommand`s

Database Command System:

- `DatabaseCommandFactory` - Factory class creates the various `DatabaseCommand`s
- `DatabaseCommand` - Abstract definition of a database command
- `UpdateItemDescription` - Update description of database item
- `UpdateItemQuantity` - Update quantity of database item
- `UpdateItemPrice` - Update price of database item
- `AddItem` - Add database item
- `RemoveItem` - Remove database item

Implementations:

- `StoreServer` - Main server backend object, contains exported `AdminController` and `ItemDatabase`
- `AdminControllerImpl` - Handles first-contact authentication and tokens
- `CustomerControllerImpl` - Handles first-contact authentication and tokens. **New in project 3**
- `AdminRegistrarImpl` - Handles authentication of administrator accounts
- `CustomerRegistrarImpl` - Handles authentication of customer accounts. **New in project 3**
- `AdminDispatcherImpl` - Dispatches administrator commands to database and registrar
- `CustomerDispatcherImpl` - Dispatches customer commands to database and registrar. **New in project 3**

- `ItemDatabaseImpl` - Maintains `Item`s
- `Item` - Contains database item attributes

Directory Structure Reference

```
images               - images used in the report
out                  - contains class files
src/client/admin     - administrator client programs
src/client/customer  - customer client programs
src/common           - interfaces and exceptions shared between client programs and server
src/database         - classes related to backend database
src/server           - classes related to server backend
src/META-INF         - contains StoreServer.jar manifest
src/Makefile         - builds StoreServer.jar
```

These files contain initial values:

```
src/server/adminaccounts.csv     - administrator accounts created when server starts
src/server/customeraccounts.csv  - customer accounts created when server starts
src/database/itemdatabase.csv    - database items created when server starts
```

Building

The server backend and client programs are hard-coded to send/receive on the same RMI port. The client programs are also hard-coded to look up the RMI registry from the same host.

The `changeport` target changes the hard-coded port across all files if needed. The default port is `54321`.

The `changeserverhost` target changes the hard-coded server host across all client program files. The default server host is `in-csci-rrpc01`.

The `StoreServer.jar` included in the submission uses the default port (`54321`) and default server host (`in-csci-rrpc01`).

```
cd src
make changeport PORT=55555  # needed if port 54321 is in use
make changeserverhost HOST=in-csci-rrpc02  # needed if server is not running on in-csci-rrpc01
make
```

Running

`StoreServer.jar` contains the server backend and client programs.

The server backend starts when `StoreServer.jar` is ran with no arguments. The server creates an RMI registry on `localhost:54321`, or on the port chosen during the build step. Client programs look for an RMI registry on `in-csci-rrpc01`, or on the server host chosen during the build step.

Administrator client programs:

- `client.admin.RegisterAccount` - Register new administrator account
- `client.admin.RegisterFromExistingAccount` - Register new administrator account through existing account
- `client.admin.AddItem` - Add new item to database
- `client.admin.RemoveItem` - Remove item from database
- `client.admin.UpdateItemDescription` - Update item description
- `client.admin.UpdateItemPrice` - Update item price

- `client.admin.UpdateItemQuantity` - Update item quantity
- `client.admin.RegisterCustomerAccount` - Register new customer account. **New in project 3**
- `client.admin.RemoveCustomerAccount` - Remove existing customer account. **New in project 3**

Customer client programs:

- `client.customer.RegisterAccount` - Register new customer account. **New in project 3**
- `client.customer.BrowseAndCheckout` - Add items to cart and checkout. **New in project 3**

Server backend program:

- `server.StoreServer` - Start server backend (default)

The `StoreServer` contains one built-in administrator account with username `admin` and password `password`.
To execute a client program where `PROGRAM` is one of the above:

```
java -cp StoreServer.jar PROGRAM
```

Sample Runs

First, start the `StoreServer` up on `in-csci-rrpc01`.

```
[bddean@in-csci-rrpc01 src]$ java -jar StoreServer.jar
```

Now from `in-csci-rrpc02`, interact with the server.

Create a new administrator `brady`:

```
[bddean@in-csci-rrpc02 src]$ java -cp StoreServer.jar client.admin.RegisterFromExistingAccount
Login to existing administrator account
Username: admin
Password: password
Creating new administrator account
New username: brady
New password: password
New administrator account brady created
```

Add `Cake` to the database:

```
[bddean@in-csci-rrpc02 src]$ java -cp StoreServer.jar client.admin.AddItem
Login to existing administrator account
Username: brady
Password: password
Adding new item
New item name: Cake
New item description: Classic birthday food
New item type: Bakery
New price: 9.99
New item Cake added
```

Update `Cake` price:

```
[bddean@in-csci-rrpc02 src]$ java -cp StoreServer.jar client.admin.UpdateItemPrice
Login to existing administrator account
Username: brady
Password: password
Updating item price
Item name: Cake
New price: 8.50
Item Cake price updated
```

Register new customer account

```
[bddean@in-csci-rrpc02 src]$ java -cp StoreServer.jar client.customer.RegisterAccount
Creating new customer account
Username: brady
Password: password
Account brady created
```

Browse and checkout items

```
[bddean@in-csci-rrpc02 src]$ java -cp StoreServer.jar client.customer.BrowseAndCheckout
Login to existing customer account
Username: brady
Password: password

Available Items
ID: 0
Name: Bread
Price: $2.50
Type: Dry Grocery
Description: Loaf of bread
Available quantity: 10

ID: 1
Name: Cake
Price: $8.50
Type: Bakery
Description: Classic birthday dessert
Available quantity: 10

ID: 2
Name: Hot Dog
Price: $1.00
Type: Chilled
Description: Mystery meat tube
Available quantity: 20

Add items to your cart by specifying the ID followed by the quantity requested
Eg: 0 2
Enter "cart" to see items in your cart
Enter "checkout" when you are ready to checkout
0 2
2 10
cart
2 Bread
10 Hot Dog
Total: $3.50
checkout
Cart was successfully checked out
```

Now inventory has been reduced

```
[bddean@in-csci-rrpc02 src]$ java -cp StoreServer.jar client.customer.BrowseAndCheckout
Login to existing customer account
Username: brady
Password: password

Available Items
ID: 0
Name: Bread
Price: $2.50
Type: Dry Grocery
Description: Loaf of bread
Available quantity: 8

ID: 1
Name: Cake
Price: $8.50
Type: Bakery
Description: Classic birthday dessert
Available quantity: 10

ID: 2
Name: Hot Dog
Price: $1.00
Type: Chilled
Description: Mystery meat tube
Available quantity: 10

Add items to your cart by specifying the ID followed by the quantity requested
Eg: 0 2
Enter "cart" to see items in your cart
Enter "checkout" when you are ready to checkout
checkout
Cart was successfully checked out
```

Remove customer account `brady`

```
[bddean@in-csci-rrpc02 src]$ java -cp StoreServer.jar client.admin.RemoveCustomerAccount
Login to existing administrator account
Username: brady
Password: password
Removing customer account
Customer username: brady
Customer account brady removed
```

Customer `brady` can no longer log in

```
[bddean@in-csci-rrpc02 src]$ java -cp StoreServer.jar client.customer.BrowseAndCheckout
Login to existing customer account
Username: brady
Password: password
Exception: The username brady does not exist
Could not authenticate
```