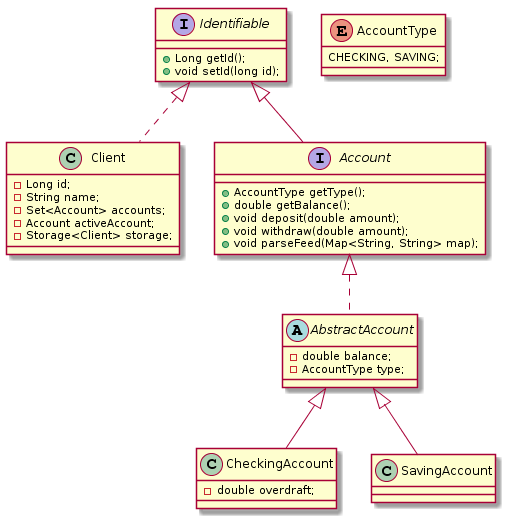
**Exercise 0 (20 minutes)**

**Study the Bank Application**

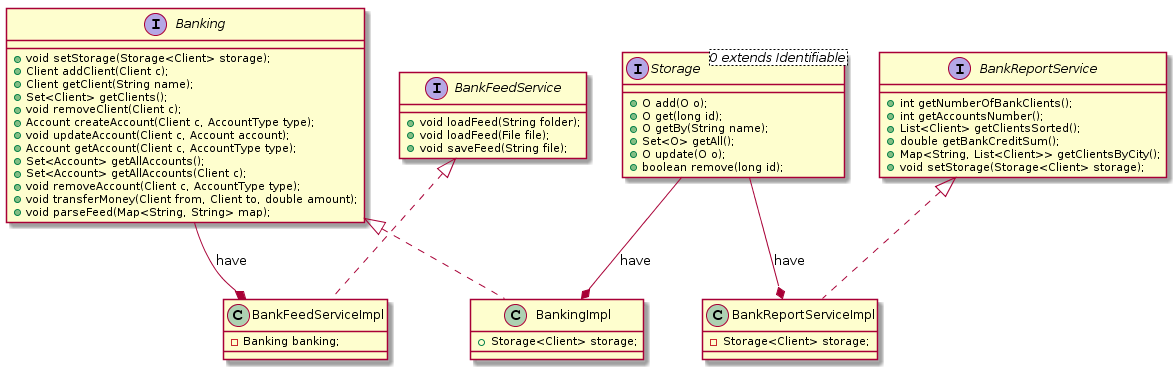
The idea is very simple. The bank has clients and every **Client** can open 2 accounts **SavingAccount** and **CheckingAccount** with overdraft.

Domain objects



Here there are two main interfaces: 1) **Account** that represents a bank account, and 2) **Identifiable** which contains a unique ID.

**Client** also contains logic to work with an active account, e.g. deposit, withdraw, and return current balance.

Services

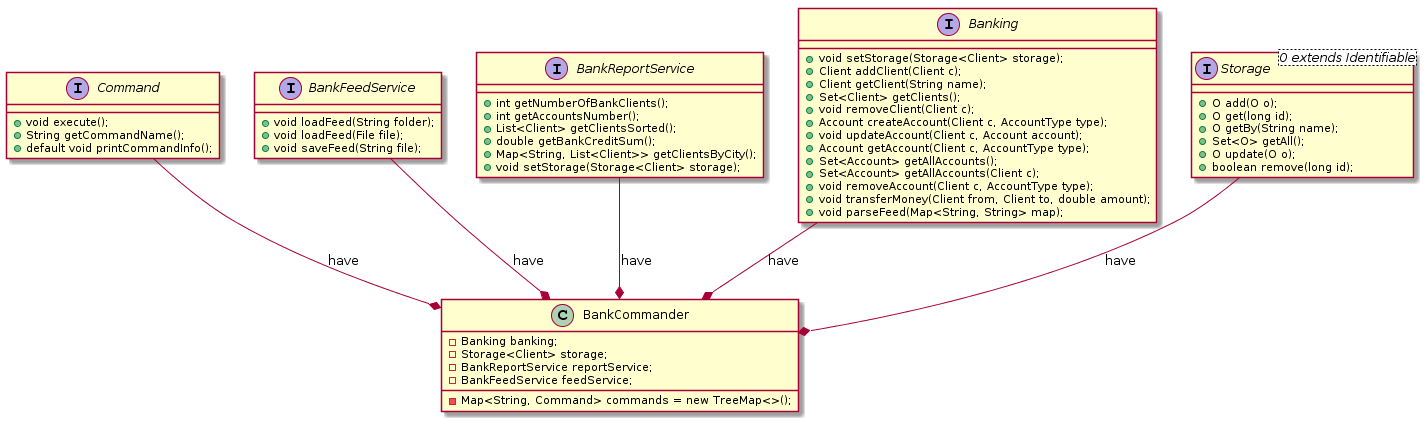
The **Banking** interface serves as a facade to implement all banking operations: working with clients, accounts, global money transfers, etc. As well, it has a link to **ClientStorage** (represented with **Storage** interface).

**BankReportService** contains everything related to reports.

**BankFeedService** serves to back up and then load client information to the file. This service scan domain classes & stores every field with **Feed** annotation.

Commands

The application also contains a **BankCommander** class with a list of commands. This interface provides an ability to work with the bank via command line interface.



Every command implements a **Command** interface and represents one specific action that can be executed independently.

**Exercise 1 (40 minutes)**

**Task 1: Set up banking**

Set up **Spring** and add all services to application context. Services **Banking** and **Storage** should be created and configured via Spring. Banking operations should work as before.

1. Create and add **application-context.xml** to resources folder;

2. Create **ClassPathXmlApplicationContext** using **application-context.xml** as input in **BankApplication** class;

3. Set up **ClientStorage** and **BankingImpl** beans using autowire by type for banking to find storage dependency;

4. Modify **initialize** method to get **ApplicationContest** instead of **Storage** / Banking initialize (ApplicationContext context);

5. Make sure methods **workWithExistingClients(Banking)** and **bankingServiceDemo(Banking)** work properly;

Note: All tests from **BankApplicationTask1Tests** should pass.

**Task 2: Configure Reporting**

**BankReportService** should be created and configured with Spring. Logic should not be updated.

1. Add **BankReportServiceImpl** to **application-context.xml**;

2. Modify the **bankReportsDemo** method to get **ApplicationContest** instead of **Storage / void** bankReportsDemo(ApplicationContext context);

3. Make sure methods **bankReportsDemo(ApplicationContext)** work properly;

Note: All tests from **BankApplicationTask2Tests** should pass.

**Task 3: Configure Feed Backup**

Currently, we have **BankFeedService** that can back up client information to file and then restore banking from backup.

Now it’s time to create and configure **BankFeedService** with Spring. Logic should not be updated.

1. Add **BankFeedServiceImpl** to **application-context.xml**. Use constructor to provide **Banking;**

2. Modify the **bankFeedDemo** method to get **ApplicationContest** instead of **Banking / void** bankFeedDemo(ApplicationContext context)

3. Make sure methods **bankFeedDemo(ApplicationContext)** work properly;

Note: All tests from **BankApplicationTask3Tests** should pass.

**Task 4: Configure Clients**

Here we are configuring and adding two clients inside the **BankApplication** class.

**Banking initialize(ApplicationContext context)**

We need these clients and accounts to demo the application after start. You should move these clients to a separate Spring configuration file. Then get these clients from **Spring** context and add to **Banking**.

1. Create and add **test-clients.xml** to resources folder;

2. Add this file to **ClassPathXmlApplicationContext;**

3. Now we should import **application-context.xml** to **test-clients.xml** because we will need some beans from this file;

Configure two beans:

4. Bean **savingAccount1** with **initialBalance** value **1000;**

5. Bean **checkingAccount1** with **overdraft** value **1000;**

6. Configure bean **client1** with **name** *Jonny Bravo*, also set **gender**, **storage** and **accounts** properties;

* Note that we already have **storage** configured in **application-context.xml**
* Also you should add setter method for **accounts** to the **Client** class

7. Replace **client\_1** with bean **client1** inside Banking initialize(ApplicationContext context) method;

8. Check everything should work as before;

9. Repeat points 4-7 for **client\_2** and check the results.

* Note that **client\_2** has only one checking account with overdraft of 1500.
* Now we have only one issue. We should put all test data like client names, cities, etc. to .xml file.

10. Create **clients.properties** file and fill it with test data like this:

**client1**=**Jonny Bravo**

**client2**=**Adam Budzinski**

**client3**=**Anna Smith**

Now when we have this file we can set up Spring to read it and replace all values with needed names. Also you can put their cities, account balances, etc.

11. Configure a new bean with **PropertyPlaceholderConfigurer** class and set **clients.properties** to location property;

12. Replace all values with placeholders from **clients.properties.** Use this syntax **${key\_from\_file};**

13. Check if everything should work as before. All tests from **BankApplicationTask4Tests** should pass.

**Exercise 2 (20 minutes)**

**Task 1: Annotation Based Dependencies**

In this task we move all dependency configuration from **application-context.xml** to annotations.

1. Open project **bank-ex2-t1-start**. Basically you need just unit tests from this project, so you can copy it to your current application if you want;

2. To enable annotation processing, add to your **application-context.xml (**next line on top) **<context:annotation-config />**

3. Remove **autowire** attribute from the **banking** bean configuration;

4. Mark **storage** field of **BankingImpl** class with **@Autowired** annotation;

5. Check your work. All tests from **BankApplicationTask1Tests** should pass.

6. Remove **storage** property from the **bankReport** bean configuration;

7. Mark **storage** field of **BankReportServiceImpl** class with **@Autowired** annotation;

8. Check your work. All tests from **BankApplicationTask2Tests** should pass.

9. Remove **constructor-arg** from **feedService** bean configuration;

10. Mark **constructor** of **BankFeedServiceImpl** class with **@Autowired** annotation;

11. Check your work. All tests from **BankApplicationTask3Tests** should pass;

12. Check your work. All tests from **BankApplicationTask4Tests** should pass.

**Exercise 3 (30 minutes)**

**Task 1: Java Based Configuration**

In this task we move configuration from **application-context.xml** to annotations and remove this file from the project. After refactoring we will have Java Based Spring configuration with no xml.

1. Add **@Service** annotations to **ClientStorage**, **BankingImpl**, **BankReportServiceImpl**, and **BankFeedServiceImpl;**

2. Add **@Scope(value = ConfigurableBeanFactory.SCOPE\_SINGLETON)** annotation to **ClientStorage**, and **BankingImpl;**

3. Add **@Configuration** and **@ComponentScan("com.luxoft.bankapp")** annotations to **BankApplication** class;

4. Open **BankApplication** class and replace **ClassPathXmlApplicationContext** class with

**AnnotationConfigApplicationContext(BankApplication.class) ;**

5. Open **BankApplication#initialize** method and replace it with code below:

**public static** Banking initialize(ApplicationContext context)

{

Banking banking = context.getBean(BankingImpl.**class**);

Client client\_1 = **new** Client(***CLIENT\_NAMES***[0], Gender.***MALE***);

Account savingAccount = **new** SavingAccount(1000);

client\_1.addAccount(savingAccount);

Account checkingAccount = **new** CheckingAccount(1000);

client\_1.addAccount(checkingAccount);

Client client\_2 = **new** Client(***CLIENT\_NAMES***[1], Gender.***MALE***);

Account checking = **new** CheckingAccount(1500);

client\_2.addAccount(checking);

banking.addClient(client\_1);

banking.addClient(client\_2);

**return** banking;

}

6. Remove **application-context.xml** file;

7. Check your work. All tests from **BankApplicationTask1Tests**, **BankApplicationTask2Tests**, **BankApplicationTask3Tests** should pass.

**Task 2: Java Based Configuration (for test clients)**

In this task we move configuration from **test-clients.xml** to **BankApplication** class and remove this file from the project. After refactoring we will have Java Based Spring configuration with no xml at all.

1. Open **BankApplication** class;

2. Add **@PropertySource("classpath:clients.properties")** annotation;

3. Add property:

@Autowired

**private** ApplicationContext **applicationContext**;

4. Add property:

@Autowired

**private** Environment **environment**;

5. Configure the **ChekingAccount** bean with next code:

@Bean(name = **"checkingAccount2"**)

**public** CheckingAccount getDemoCheckingAccount2()

{

**return new** CheckingAccount(1500);

}

6. Configure **Client** bean with next code:

@Bean(name = **"client2"**)

**public** Client getDemoClient2()

{

String name = **environment**.getProperty(**"client2"**);

Client client = **new** Client(name, Gender.***MALE***);

client.setCity(**"Kiev"**);

Account checking = (CheckingAccount) **applicationContext**.getBean(**"checkingAccount2"**);

client.addAccount(checking);

**return** client;

}

6. Configure the **client1** bean in the same way as we did for **client2**;

7. Now you should have 5 configured beans **client1**, **client2**, **checkingAccount1**, **checkingAccount2**, and **savingAccount1**;

8. Open the **BankApplication#initialize** method and replace client with next code:

Client client\_1 = (Client) context.getBean(**"client1"**);

Client client\_2 = (Client) context.getBean(**"client2"**);

9. Remove **test-clients.xml** file;

10. Check your work. All tests from **BankApplicationTask4Tests** should pass;

11. Open the **BankApplication#bankReportsDemo** method and replace code to get **reportService** with next code:

BankReportService reportService = context.getBean(BankReportServiceImpl.**class**);

12. Open the **BankApplication#bankFeedDemo** method and replace code to get **feedService** with next code:

BankFeedService feedService = context.getBean(BankFeedServiceImpl.**class**);

13. Check your work. All tests from **BankApplication#main** method should run without exceptions and all 4 test cases should pass.