Inheritance & Polymorphism & Interface & Collections

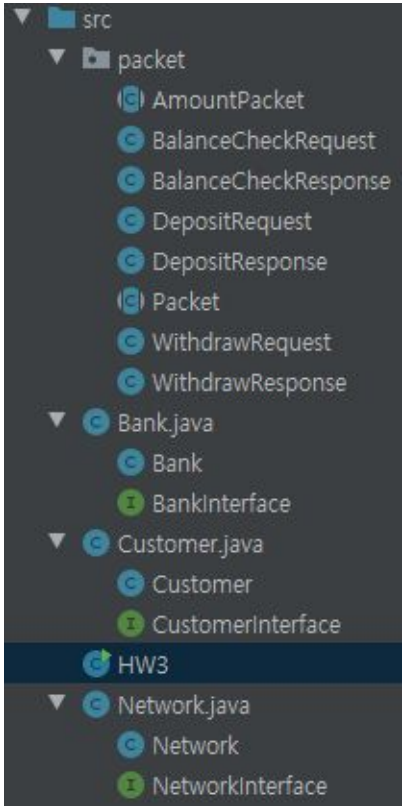
Homework 3

Homework 3: Async Bank

- Let’s implement the asynchronous bank system
- The term *asynchronous* means that your action doesn’t necessarily cause the consequence immediately.
  - You can do some other things until you’re notified by the delayed consequence.
  - E.g. In a cafe, you don’t have to wait in a line until your coffee is ready
- In the programming perspective, asynchronous tasks are mediated by several *requests* and *responses*
  - A function call makes some requests, which are to be queued in a event queue somewhere.
  - When queued events are processed, responses are created, and queued again in some other queue to wait for your attention.
- In the bank situation, customers are going to request withdrawal, deposit, and balance check to banks.
  - Banks in turn send some responses to customers.
  - When customers process those response, they will print the result to stdout

Class structure

- There 3 major classes
  - **Customer class** has its name and money, and can request withdrawal, deposit, and balance check to a bank
    - Customer class has its own queue to store bank's response, and can process it later.
  - **Bank class** maintains customers' balance.
    - Bank class also has its own queue to store customer's response to be processed later
  - **Network class** is responsible for delivering request and response messages between customers and banks
    - **Packet class** and its variants represents a request or response



Main code and its output

```java
public static void main(String args[]) {
    Network network = new Network();
    Customer minsu = new Customer( id: "Minsu",  money: 10000);
    Bank hanabank = new Bank( id: "Hanabank");
    Bank kakaobank = new Bank( id: "Kakaobank");
    network.addCustomer(minsu);
    network.addBank(hanabank);
    network.addBank(kakaobank);

    minsu.requestDeposit(hanabank,  amount: 3000);
    minsu.requestDeposit(kakaobank,  amount: 5000);
    minsu.requestBalanceCheck(hanabank);
    minsu.requestBalanceCheck(kakaobank);

    while(hanabank.hasNextRequest()) hanabank.processNextRequest();
    while(kakaobank.hasNextRequest()) kakaobank.processNextRequest();
    while(minsu.hasNextResponse()) minsu.processNextResponse();

    minsu.requestWithdraw(hanabank,  amount: 2000);
    minsu.requestWithdraw(kakaobank,  amount: 3000);
    minsu.requestBalanceCheck(hanabank);
    minsu.requestBalanceCheck(kakaobank);

    while(hanabank.hasNextRequest()) hanabank.processNextRequest();
    while(kakaobank.hasNextRequest()) kakaobank.processNextRequest();
    while(minsu.hasNextResponse()) minsu.processNextResponse();

    System.out.println("Minsu's cash: " + minsu.getMoney());
}
```

```
Minsu made deposit request to Hanabank with amount 3000
Minsu made deposit request to Kakaobank with amount 5000
Minsu made balance check request to Hanabank
Minsu made balance check request to Kakaobank
Response: Successfully deposit 3000 to Hanabank
Response: Minsu now has 3000 in Hanabank
Response: Successfully deposit 5000 to Kakaobank
Response: Minsu now has 5000 in Kakaobank
Minsu made withdrawal request to Hanabank with amount 2000
Minsu made withdrawal request to Kakaobank with amount 3000
Minsu made balance check request to Hanabank
Minsu made balance check request to Kakaobank
Response: Successfully withdrew 2000 to Hanabank
Response: Minsu now has 1000 in Hanabank
Response: Successfully withdrew 3000 to Kakaobank
Response: Minsu now has 2000 in Kakaobank
Minsu's cash: 7000
```

Packet class

```java
abstract public class Packet {
    private String customerId;
    private String bankId;

    public Packet(String customerId, String bankId) {
        this.customerId = customerId;
        this.bankId = bankId;
    }

    public String getCustomerId() {
        return customerId;
    }

    public String getBankId() {
        return bankId;
    }

}
```

```java
abstract public class AmountPacket extends Packet {
    private int amount;
    public AmountPacket(String customerId, String bankId, int amount) {
        super(customerId, bankId);
        this.amount = amount;
    }
    public int getAmount() { return amount; }
}
```

```java
public class DepositRequest extends AmountPacket {
    public DepositRequest(String customerId, String bankId, int amount) {
        super(customerId, bankId, amount);
    }
}
```

and so on...

Network class

- addCustomer, addBank
  - Set itself as the object's `network`
  - Store the object in its hash map
- sendRequest
  - Enqueue the packet to the bank described in the packet
- sendResponse
  - Enqueue the packet to the customer described in the packet

```
interface NetworkInterface {
    void addCustomer(Customer customer);
    void addBank(Bank bank);
    void sendRequest(Packet packet);
    void sendResponse(Packet packet);
}

public class Network implements NetworkInterface {
    private HashMap<String, Customer> idToCustomer;
    private HashMap<String, Bank> idToBank;
```

Customer class

- requestDeposit
  - Send a deposit request to the network
  - It should immediately reduce `money`
- requestWithdraw
  - Send a withdraw request to the network
  - It should not change `money` at this stage.
  - Instead, `money` should be increased when processing a corresponding response
- requestBalanceCheck
  - Send a balance check request to the network
- processNextResponse
  - Poll one response from the queue, process it, and print the result

```
interface CustomerInterface {
    String getId();
    int getMoney(); void setMoney(int money);
    Network getNetwork(); void setNetwork(Network network);

    void enqueueResponse(Packet packet);
    void processNextResponse();
    boolean hasNextResponse();

    void requestDeposit(Bank bank, int money);
    void requestWithdraw(Bank bank, int money);
    void requestBalanceCheck(Bank bank);

}

public class Customer implements CustomerInterface {
    private String id;
    private int money;
    private LinkedList<Packet> responseQueue;
    private Network network;
```

Bank class

- HashMap<String, Integer> accounts
  - Maintains balance of customers
  - getOrDefault method might help
- processNextRequest
  - Poll one request from the queue, and adjust customers' balance according to it.
  - Make a response that describes the changed state

```
interface BankInterface {
    String getId();
    Network getNetwork(); void setNetwork(Network network);

    void enqueueRequest(Packet request);
    boolean hasNextRequest();
    void processNextRequest();
}

public class Bank implements BankInterface {
    private String id;
    private LinkedList<Packet> requestQueue;
    private Network network;
    private HashMap<String, Integer> accounts;
```

Advanced Questions

Consider the following questions to further your practice.
(There is no exact answers, and we won't provide it either)

1. What if a customer wants to withdraw money more than they has? Handle this exceptions, represent it as a response, and describe it in processNextResponse
2. Banks must be persistent. Use file I/O to save and load banks' states.
3. Currently, request and response classes are all extended from Packet or AmountPacket. Make Request, AmountRequest, Response, AmountResponse class extending from it, and make every unit request and response class extend from the newly made classes.
4. Add register/unregister routine to the system (i.e. a customer should first register to a bank to use their system). In the Bank class, use Set to maintain the membership information