CS1812 Assessed Coursework 3

This assignment must be submitted by 10:00 on Thursday 23rd March.

Feedback will be provided by the date provided in the coursework grid.

Learning outcomes assessed

The learning outcomes assessed are use of streams, inheritance, sorting, and your general Java programming skills.

Instructions

Please submit your Java files on Moodle. Ensure that you submit **five** files, which should be named as follows: `Rank.java`, `Card.java`, `CardTest.java`, `HollomonClient.java`, and `CardInputStream.java`.

N.B. Moodle will only allow you to upload files with a `.java` extension.

Marking criteria

Full marks will be awarded for correct answers to the questions below. The assessment of the correctness of submissions is based on:

- functionality (i.e. whether the submitted code performs the requested tasks);
- proper use of object-oriented design as taught during the course lectures and exercise classes; and
- coding style.

You should aim to write well-formatted and well-commented code that compiles, and to make use of informative and meaningful variable names. You should also aim to make the code as readable as possible, i.e. do not clutter the code by providing too much information.

You should attempt all parts and submit what you have done, even if you are unable to complete some of the questions. Marks are awarded for each question individually, even when the overall solution is incomplete or incorrect.

A `jar` file for testing is provided. Please use this testing tool as it will give an indication of your progress. This tool will be used by the markers in assessing your work. Please note that passing all the provided tests does not guarantee full marks, since an automatic testing process cannot verify proper use of object-oriented design, coding style, or readability.

Please contact reuben.rowe@rhul.ac.uk or matteo.sammartino@rhul, or post in the Moodle or Piazza forums if you have any specific questions.

Academic misconduct

Coursework submissions are routinely checked for academic misconduct (working together, copying from sources, etc.). Penalties can range from a 10% deduction of the assignment mark, zero for the assignment, or (for repeat offences) the case being referred to a Senior Vice-Principal to make a decision.

Further details can be found here.

Extensions

This assignment **is subject to Department policy on extensions**. If you believe you require an extension please read the documentation carefully here.

Extenuating circumstances

If you submit an assessment and believe that the standard of your work was substantially affected by your current circumstance then you can apply for Extenuating Circumstances. Details on how to apply for this can be found here .

It is essential you read the accompanying documentation on the above link carefully. Please note decisions on Extenuating Circumstances are made at the end of the academic year.

Late submission

In the absence of acceptable extenuating cause, late submission of work will be penalised as follows:

1. for work submitted up to 24 hours late, the mark will be reduced by ten percentage marks;
2. for work submitted more than 24 hours late, the maximum mark will be zero.

Hollomon: Gotta cache 'em all!

In this assignment, you will implement a client for a (simplified) network-based trading card game running on a centralised server. Clients authenticate to the server with a username and password, after which the server acknowledges and sends the current list of cards owned by the player. A player receives a free randomly chosen card upon login.

Once the cards have been sent, the server accepts the commands `CREDITS`, `OFFERS`, `BUY`, `SELL`, and `CARDS` that allows the amount of credits available to be seen, get a list of cards for sale, to buy or sell cards, or to retrieve the player's cards again.

Cards can have one of four ranks, *common*, *uncommon*, *rare*, and *unique*. The odds of receiving less common cards for free are significantly reduced, so one might receive the same common card multiple times before obtaining a particular rare card. The goal of the game is to collect as many cards of different type as possible.

It pays off to be quick! The first players to successfully log in, to sell, or to buy cards, will receive unique cards as a reward. After the early rewards have been distributed, free random cards are given out upon reconnection every $2\sqrt{n}$ minutes, where n is the number of free cards already received.

The Hollomon Game was first devised and implemented by Prof. Johannes Kinder.

Login Details The hollomon game server is hosted at the URL `netsrv.cim.rhul.ac.uk`. You will write a program that connects to this URL on port 1812. The game leaderboard is viewable by visiting the following website.

https://www.cs.rhul.ac.uk/home/uhac003/hollomon/index.php

Trading accounts have been created for all students. Your individual login details will be distributed to you by email.

IMPORTANT: You will have to be connected to the University network in order to be able to connect. Since it is unlikely that you are on campus at the moment, you should either run your code on NoMachine, or connect via the VPN. You can find instructions for connecting to the VPN on the following web page.

https://intranet.royalholloway.ac.uk/students/help-support/it-services/access-off-campus/vpn/vpn.aspx

The testing tool can be used to verify that it is possible to connect with your given details. To make sure you can access the server, run the testing tool and supply your username and password on the command line. Supposing your login name is `candle` and your password is `password1234`, run

```
$ java -jar assignment3Tester.jar candle password1234
```

and you should see the output

```
Testing server availability by trying to connect to netsrv.cim.rhul.ac.uk:1812
as candle
Connection successful, the server is available.
```

If there is a problem, you will see a warning.

```
Testing server availability by trying to connect to netsrv.cim.rhul.ac.uk:1812
as candle
WARNING: could not connect to server, your code will not be able to connect.
Please check your username and password as well as your internet connection.
```

If it does not succeed, verify that you have the correct details and that you have a working internet connection.

1 Rank [5 marks]

Write an enum `Rank` to represent the four ranks. It should contain the values `UNIQUE`, `RARE`, `UNCOMMON`, and `COMMON`.

Now run the testing tool.

```
$ java -jar assignment3Tester.jar
```

and test `testQ1_RankEnum` should now pass. Note, you do not need to pass your username and password for this part.

2 Cards [25 marks]

Write a class `Card` to represent a card in the game.

- A card should have:
  1. a numeric ID (a long integer);
  2. a name;
  3. a rank, which should be of an enumerated type; and
  4. a price (a `long` integer), which is the price the card was last on sale for.
- Add a constructor `public Card(long id, String name, Rank rank)` that creates a new card for given values of ID, name, and rank, and sets the price to 0.
- Override the `toString` method in `Card` to return a representation of the card as a string.
- Override `hashCode` and `equals` in `Card` so that two cards are considered equal if and only if they have the same numeric ID, name, and rank. (**N.B.** you *don't* want to compare the prices.)
- Make `Card` comparable: implement the `Comparable` interface and add a `compareTo` method that compares two cards first by rank, with unique cards coming first, then by name, and finally by their numeric ID. If two cards have the same rank, name, and ID, then `compareTo` should consider them to be equal (just like `equals`).

Your code should now pass `testQ2_01_Card`.

Do your own testing of your `Card` class by writing a class `CardTest` with a main method that creates some cards with combinations of values that test all the different cases.

- Test whether adding the cards to a `HashSet` leaves out multiple objects representing the same card if they only differ in price.
- Add the cards to a `TreeSet` and check that if you print out all elements of that set, they are ordered by rank and name.

Your code should now pass `testQ2_02_CardTest`.

3 Connecting [20 marks]

In this question you will create a `HollomonClient` class that will handle interaction with the Hollomon server. You will complete the login process for this question, and handle further interaction in later questions.

The Hollomon server is available at the address `netsrv.cim.rhul.ac.uk` on the port `1812`.

IMPORTANT: You will have to be connected to the University network in order to be able to connect. Since it is unlikely that you are on campus at the moment, you should either run your code on NoMachine, or connect via the VPN. You can find instructions for connecting to the VPN on the following web page.

https://intranet.royalholloway.ac.uk/students/help-support/it-services/access-off-campus/vpn/vpn.aspx

Connect Early! Aim to be able to get a connection as soon as possible: if you cannot get this to work, you will struggle with the rest of the assignment.

A Note About Security The password handling in this exercise is probably not secure enough for use in the real-world. If you are tasked, during your future career, with implementing a login system, brush-up on industry best-practice first!

The `HollomonClient` Class Create a class `HollomonClient` that opens a connection to the server. The class should contain the items below. Advice on the required components is given after the list.

- A constructor `public HollomonClient(String server, int port)` that instantiates the `HollomonClient` class with the given server and port.
- A method `public List<Card> login(String username, String password)` that
  - logs the user into the Hollomon server;

  - returns `null` if the login is unsuccessful, and an empty list if the login is successful (you will populate this list in Section 4).

- A method `public void close()` that closes all streams used by your `HollomonClient`.

Creating the Socket You will need to create a socket to connect to the server. You can use the `Socket` class, in the `java.net` package, to create a connection by passing the server name and port number as a string `host` and integer `port`, respectively, to the constructor, as follows.

```
Socket socket = new Socket(host, port);
```

(Note that the constructor may throw an `UnknownHostException`, also from the `java.net` package.) You can then read from and write data to the server using an `InputStream` and `OutputStream`, respectivey, which can be obtained from the socket object as follows.

```
InputStream is = socket.getInputStream();
OutputStream os = socket.getOutputStream();
```

(Note that an `IOException` may be thrown by the `getInputStream()` and `getOutputStream()` methods.)

Logging In After creating the socket, the server is waiting for your client to log in. To log in, make your client:

1. send your username in lower case followed by a single newline character; and
2. send the password you set for the game followed by a single newline character.

If your credentials are correct, the server should respond with

```
User <yourusername> logged in successfully.
```

In this case your `login` method should return an empty list, `new ArrayList<Card>()`. In Section 4 you will see how to populate this list. If the login is not successful, then `null` should be returned.

If the server does not seem to acknowledge the login attempt, verify that you are flushing the buffer of the output stream if necessary, and that you are not sending too many or too few newline characters.

Closing the Connections The method `public void close()` should close all streams you may have opened, as well as the socket. When testing your code, if you forget call `close()`, the connection will stay open. This will mean that the next time you try to log in, you may get a server error that multiple connections are not allowed.

Test your login code both by creating a `HollomonClientTest` class and by running the testing tool. You should see that `testQ3_Login` is successful.

4 Reading Cards [25 marks]

Directly after a successful login, the server will send a list of cards to the client. This list of should be returned by the `login` method. You should write a `CardInputStream` class, described below, to handle card reading.

CardInputStream To read the cards sent to the client after logging in, create your own input stream that is specialised for reading cards and wraps another input stream. Create a class `CardInputStream` that inherits from `InputStream` and add the following:

- A constructor `public CardInputStream(InputStream input)` that creates an instance of `BufferedReader` from `input`, which will be used in the remainder of the class.
- An overriding method `close()` to close the reader.
- A method `Card readCard()` that reads the details for one card from the reader and creates and returns a corresponding new `Card` object. The format used by the server is plain text, separated by new lines. For example, a card with ID `12345`, name `"Butler"`, rank `COMMON`, and a last sale price of `0` credits would be sent as:

  ```
  CARD
  12345
  Butler
  COMMON
  0
  ```

  The tag `CARD` lets the client know that the next four lines are card details.
- A method `String readResponse()` that returns a full line read from the socket. You will need this method to read the server response to any commands (non-card messages like `OK` or the number of credits)

Hint: Consider using the `valueOf` method of the enumerated type to parse the card rank.

After completing this part, your code should pass `testQ4_01_CardInputStream`.

Receiving and Sorting Cards You can now complete your implementation of

```
public List<Card> login(String username, String password)
```

by returning a populated list. After a successful login, the server will send card data. To read this data, create a `CardInputStream` from the `InputStream` coming from the socket. Using a loop, read all cards from the server and add them to the `List`. After all cards have been sent, the server sends `OK` on a single line. Your `readCard` method of `CardInputStream` should check for this and return `null` in this case. Thus the cards are read in a loop using the same pattern you would use to read text line by line.

Before returning the list from the `login` method, use the `Collections.sort` method to sort the list using the ordering you implemented in Question 2.

Your code should now pass `testQ4_02_GetCardList`. You should also write your own tests to verify and debug your code.

5 Trading [25 marks]

To support card trading, you will have to implement methods for issuing different commands to the server and processing the response. Commands consist of a single word, sometimes followed by some extra data. The end of a command is signalled using a new line character.

You should implement these methods in your `HollomonClient` class. Each method should deal gracefully with errors and exceptions (i.e., your client shouldn't crash and in the worst case report that the connection was lost).

Reading Credits Add a method `public long getCredits()` to your `HollomonClient` class that sends the `CREDITS` command and returns the result as a long integer. The server responds to the `CREDITS` command by sending the current amount as a text string, followed by `OK` on a new line. Every new player receives a starting budget of 100 credits.

You should now pass `testQ5_01_Credits`.

Retrieving Cards without Reconnecting You may want to read all cards from the server again, without reconnecting. Implement a method `public List<Card> getCards()` in your `HollomonClient` class that sends the command `CARDS` to the server, receives the cards, and returns them in a sorted list. Use your `CardInputStream` to read all cards into a list.

You should now pass `testQ5_02_Cards`.

Getting Cards for Sale The client can ask for a list of cards currently on sale by other players using the command `OFFERS`. Add to your `HollomonClient` class a method

```
public List<Card> getOffers()
```

that sends the command `OFFERS` to the server, receives the cards, and returns them in a sorted list.

Use your `CardInputStream` to read all cards into a list. The price field of each card will hold the price for which the card is on sale.

You should now pass `testQ5_03_Offers`.

Buying To buy a card, the client sends a `BUY <cardid>` command, e.g., `BUY 23311`. If buying succeeds, the server replies with `OK`. If it fails (because the player does not have enough funds or because another client concurrently bought the card), the server replies with `ERROR`.

Write a method `public boolean buyCard(Card card)` that implements buying a card. The method should first verify that you have sufficient funds to buy a card and then send the command. The method should return `true` in case the server returns `OK` and `false` otherwise.

Your code should now pass `testQ4_04_Buy`.

Selling The client can issue an order to sell by sending the following command.

```
SELL <cardId> <price>
```

For example, to offer the card `12345` for `7` credits, the client would send `SELL 12345 7`. If placing the order succeeds, the server responds with `OK`, and with `ERROR` otherwise.

Write a method `public boolean sellCard(Card card, long price)` in `HollomonClient` that implements selling a card. Like `buyCard` the method should return `true` or `false` depending on the server reply.

Once another player buys your card, the card will be removed from your collection and you will receive the amount asked for. You can change the price by sending a new `SELL` command. You can cancel an order to sell by buying your own card.

Your code should now pass `testQ4_05_Sell`.

Trading Optionally, you can implement a strategy for automatically buying and selling cards. For instance, you could buy all cards on offer that you do not yet own, as long as you have sufficient credits. Unique cards will be available to buy at increasing prices — trade successfully to be able to afford them.

Players are ranked according to the cards they own, with each card weighted according to its rarity. Unique cards count `10`, rare cards `5`, uncommon `2`, and common `1`. The current ranking is shown on https://www.cs.rhul.ac.uk/home/uhac003/hollomon/index.php.