**Чат (1)**

Сегодня мы напишем чат. Набор программ с помощью которого можно будет  
обмениваться текстовыми сообщения. Набор будет состоять из одного сервера и  
нескольких клиентов, по одному для каждого участника чата.

Начнем с сервера. Нам понадобятся такие классы:  
1) Server – основной класс сервера.  
2) MessageType – enum, который отвечает за тип сообщений пересылаемых между  
клиентом и сервером.  
3) Message – класс, отвечающий за пересылаемые сообщения.  
4) Connection – класс соединения между клиентом и сервером.  
5) ConsoleHelper – вспомогательный класс, для чтения или записи в консоль.

Создай эти классы и добавь метод main в класс Server.

Требования:  
1. Класс Server должен быть публичным и создан в отдельном файле.  
2. Перечисление MessageType должно быть публичным и создано в отдельном файле.  
3. Класс Message должен быть публичным и создан в отдельном файле.  
4. Класс Connection должен быть публичным и создан в отдельном файле.  
5. Класс ConsoleHelper должен быть публичным и создан в отдельном файле.  
6. В классе Server должен быть создан метод public static void main(String[] args).

### Чат (2)

Первым делом, для удобства работы с консолью реализуем класс **ConsoleHelper**. В  
дальнейшем, вся работа с консолью должна происходить через этот класс.

**Добавь в него:**  
1. Статическое поле типа BufferedReader, инициализированное с помощью System.in.  
2. Статический метод writeMessage(String **message**), который должен выводить  
сообщение message в консоль.  
3. Статический метод String readString(), который должен считывать строку с  
консоли. Если во время чтения произошло исключение, вывести пользователю  
сообщение «***Произошла ошибка при попытке ввода текста. Попробуйте еще раз.***» И  
повторить ввод. Метод не должен пробрасывать исключения IOException наружу.  
Другие исключения не должны обрабатываться.  
4. Статический метод int readInt(). Он должен возвращать введенное число и  
использовать метод readString(). Внутри метода обработать исключение  
NumberFormatException. Если оно произошло вывести сообщение «***Произошла ошибка***  
***при попытке ввода числа. Попробуйте еще раз.»*** И повторить ввод числа.

В этой задаче и далее, если не указано дополнительно другого, то все поля класса должны  
быть приватными, а методы публичными.

Требования:  
1. В классе ConsoleHelper должно быть создано и инициализировано приватное статическое поле типа BufferedReader.  
2. В классе ConsoleHelper должен быть реализован статический метод writeMessage(String message), выводящий сообщение на консоль.  
3. В классе ConsoleHelper должен быть реализован статический метод readString, возвращающий строку считанную с консоли.  
4. В классе ConsoleHelper должен быть реализован статический метод readInt, возвращающий число считанное с консоли.  
5. Метод readInt должен использовать метод readString для чтения с консоли.  
6. Метод readString должен перехватывать IOException, выводить сообщение о некорректном вводе и повторять считывание с консоли.  
7. Метод readInt должен перехватывать NumberFormatException, выводить сообщение о некорректном вводе и повторять считывание с консоли.

### Чат (3)

Прежде, чем двигаться дальше, нужно разработать протокол общения клиента и сервера.

**Сформулируем основные моменты протокола:**  
— Когда новый клиент хочет подсоединиться к серверу, сервер должен запросить имя  
клиента.  
— Когда клиент получает запрос имени от сервера он должен отправить свое имя серверу.  
— Когда сервер получает имя клиента он должен принять это имя или запросить новое.  
— Когда новый клиент добавился к чату, сервер должен сообщить остальным участникам о  
новом клиенте.  
— Когда клиент покидает чат, сервер должен сообщить остальным участникам об этом.  
— Когда сервер получает текстовое сообщение от клиента, он должен переслать его всем  
остальным участникам чата.

Добавь для каждого пункта вышеописанного протокола соответствующее значение в **enum**  
**MessageType:**  
1) NAME\_REQUEST – запрос имени.  
2) USER\_NAME – имя пользователя.  
3) NAME\_ACCEPTED – имя принято.  
4) TEXT – текстовое сообщение.  
5) USER\_ADDED – пользователь добавлен.  
6) USER\_REMOVED – пользователь удален.

Требования:  
1. В перечислении MessageType должен присутствовать элемент NAME\_REQUEST.  
2. В перечислении MessageType должен присутствовать элемент USER\_NAME.  
3. В перечислении MessageType должен присутствовать элемент NAME\_ACCEPTED.  
4. В перечислении MessageType должен присутствовать элемент TEXT.  
5. В перечислении MessageType должен присутствовать элемент USER\_ADDED.  
6. В перечислении MessageType должен присутствовать элемент USER\_REMOVED.

### Чат (4)

Сообщение **Message** – это данные, которые одна сторона отправляет, а вторая принимает.  
Каждое сообщение должно иметь тип MessageType, а некоторые и дополнительные  
данные, например, текстовое сообщение должно содержать текст. Т.к. сообщения будут  
создаваться в одной программе, а читаться в другой, удобно воспользоваться механизмом  
сериализации для перевода класса в последовательность битов и наоборот.

**Добавь в класс** Message**:**  
1) Поддержку интерфейса Serializable.  
2) final поле типа **MessageType type**, которое будет содержать тип сообщения.  
3) final поле типа String data, которое будет содержать данные сообщения.  
4) **Геттеры** для этих полей.  
5) Конструктор, принимающий только MessageType, он должен проинициализировать  
поле type переданным параметром, а поле data оставить равным null.  
6) Конструктор, принимающий MessageType **type** и **String data**. Он должен также  
инициализировать все поля класса.

Требования:  
1. Класс Message должен поддерживать интерфейс Serializable.  
2. Приватное поле type в классе Message должно быть объявлено с модификатором final и быть типа MessageType.  
3. Приватное поле data в классе Message должно быть объявлено с модификатором final и быть типа String.  
4. В классе Message должны быть созданы корректные геттеры для полей data и type.  
5. Конструктор класса Message с одним параметром типа MessageType должен инициализировать поле type и оставлять поле data равным null.  
6. Конструктор класса Message с двумя параметрами (MessageType и String) должен инициализировать поля type и data.

### Чат (5)

Клиент и сервер будут общаться через сокетное соединение. Одна сторона будет  
записывать данные в сокет, а другая читать. Их общение представляет собой обмен  
сообщениями **Message**. Класс Connection будет выполнять роль обертки над классом  
**java.net.Socket**, которая должна будет уметь сериализовать и десериализовать объекты  
типа Message в сокет. Методы этого класса должны быть готовы к вызову из разных  
потоков.

**Добавь в класс Connection:**  
1) Final поля:  
а) Socket socket  
б) ObjectOutputStream out  
в) ObjectInputStream in  
2) Конструктор, который должен принимать Socket в качестве параметра и  
инициализировать поля класса. Для инициализации полей in и out используй  
соответствующие потоки сокета. Конструктор может бросать исключение IOException.  
Создать объект класса ObjectOutputStream нужно до того, как будет создаваться объект  
класса ObjectInputStream, иначе может возникнуть взаимная блокировка потоков,  
которые хотят установить соединение через класс Connection. Более подробно об этом  
ты можешь прочитать в спецификации класса ObjectInputStream.  
3) Метод void send(Message **message**) throws IOException. Он должен записывать  
(сериализовать) сообщение message в ObjectOutputStream. Этот метод будет  
вызываться из нескольких потоков. Позаботься, чтобы запись в объект  
ObjectOutputStream была возможна только одним потоком в определенный момент  
времени, остальные желающие ждали завершения записи. При этом другие методы  
класса Connection не должны быть заблокированы.  
4) Метод Message receive() throws IOException, ClassNotFoundException. Он должен читать  
(десериализовать) данные из ObjectInputStream. Сделай так, чтобы операция чтения  
не могла быть одновременно вызвана несколькими потоками, при этом вызов других  
методы класса Connection не блокировать.  
5) Метод SocketAddress getRemoteSocketAddress(), возвращающий удаленный адрес  
сокетного соединения.  
6) Метод void close() throws IOException, который должен закрывать все ресурсы класса.

Класс Connection должен поддерживать интерфейс Closeable.

Требования:  
1. Класс Connection должен поддерживать интерфейс Closeable.  
2. В классе Connection должно быть создано private final поле socket типа Socket.  
3. В классе Connection должно быть создано private final поле out типа ObjectOutputStream.  
4. В классе Connection должно быть создано private final поле in типа ObjectInputStream.  
5. В классе Connection должен быть создан конструктор с одним параметром типа Socket инициализирующий поля класса в соответствии с условием задачи.  
6. В классе Connection должен быть корректно реализован метод send c одним параметром типа Message.  
7. В классе Connection должен быть корректно реализован метод receive без параметров.  
8. Метод getRemoteSocketAddress класса Connection должен возвращать удаленный адрес сокетного соединения.  
9. Метод close класса Connection должен закрывать потоки чтения, записи и сокетное соединение.

### Чат (6)

Приступим к самому важному – написанию класса Server. Сервер должен поддерживать

множество соединений с разными клиентами одновременно. Это можно реализовать с

помощью следующего алгоритма:

- Сервер создает серверное сокетное соединение.

- В цикле ожидает, когда какой-то клиент подключится к сокету.

- Создает новый поток обработчик Handler, в котором будет происходить обмен

сообщениями с клиентом.

- Ожидает следующее соединение.

Добавь:

1) В класс Server приватный статический вложенный класс Handler, унаследованный от

Thread.

2) В класс Handler поле socket типа Socket.

3) В класс Handler конструктор, принимающий в качестве параметра Socket и

инициализирующий им соответствующее поле класса.

4) Метод main класса Server, должен:

а) Запрашивать порт сервера, используя ConsoleHelper.

б) Создавать серверный сокет java.net.ServerSocket, используя порт из предыдущего пункта.

в) Выводить сообщение, что сервер запущен.

г) В бесконечном цикле слушать и принимать входящие сокетные соединения только что созданного

серверного сокета.

д) Создавать и запускать новый поток Handler, передавая в конструктор сокет из предыдущего пункта.

е) После создания потока обработчика Handler переходить на новый шаг цикла.

ж) Предусмотреть закрытие серверного сокета в случае возникновения исключения.

з) Если исключение Exception все же произошло, поймать его и вывести сообщение

об ошибке.

Требования:  
1. В классе Server должен быть создан приватный статический класс Handler, унаследованный от класса Thread.  
2. В классе Handler должно быть создано поле socket типа Socket.  
3. Конструктор класса Handler должен принимать один параметр типа Socket и инициализировать поле socket.  
4. Метод main должен считывать с клавиатуры порт сервера используя метод readInt класса ConsoleHelper.  
5. Метод main должен корректно реализовывать бесконечный цикл описанный в условии задачи.

### Чат (7)

Т.к. сервер может одновременно работать с несколькими клиентами, нам понадобится  
метод для отправки сообщения сразу всем.

**Добавь в класс Server:**  
1. Статическое поле **Map**<String, Connection> connectionMap, где ключом будет имя  
клиента, а значением — соединение с ним.  
2. Инициализацию поля из **п.7.1** с помощью подходящего **Map** из библиотеки  
**java.util.concurrent**, т.к. работа с этим полем будет происходить из разных потоков и  
нужно обеспечить потокобезопасность.  
3. Статический метод void sendBroadcastMessage(Message **message**), который должен  
отправлять сообщение message всем соединениям из **connectionMap**. Если при  
отправке сообщение произойдет исключение **IOException**, нужно отловить его и  
сообщить пользователю, что не смогли отправить сообщение.

### Чат (8)

Класс Handler должен реализовывать протокол общения с клиентом.  
Выделим из протокола отдельные этапы и реализуем их с помощью отдельных методов:

**Этап первый** – это этап рукопожатия (знакомства сервера с клиентом). Реализуем его с  
помощью приватного метода String serverHandshake(Connection connection) throws IOException, ClassNotFoundException. Метод в качестве параметра принимает соединение connection, а возвращает имя нового клиента.

**Реализация метода должна:**  
1) **Сформировать** и **отправить** команду запроса имени пользователя  
2) **Получить** **ответ** клиента  
3) **Проверить**, что получена команда с именем пользователя  
4) **Достать** из ответа **имя**, **проверить**, что оно не пустое и пользователь с таким именем еще не подключен (используй connectionMap)  
5) **Добавить** **нового** **пользователя** и **соединение** с ним в connectionMap  
6) **Отправить** **клиенту** команду информирующую, что его имя принято  
7) Если какая-то **проверка** не прошла, **заново** запросить имя клиента  
8) **Вернуть** принятое **имя** в качестве возвращаемого значения

Требования:  
1. В классе Handler должен присутствовать метод private String serverHandshake(Connection connection).  
2. Метод serverHandshake должен отправлять запрос имени используя метод send класса Connection.  
3. До тех пор, пока тип сообщения полученного в ответ не будет равен MessageType.USER\_NAME, запрос имени должен быть выполнен снова.  
4. В случае, если в ответ пришло пустое имя, запрос имени должен быть выполнен снова.  
5. В случае, если имя уже содержится в connectionMap, запрос имени должен быть выполнен снова.  
6. После успешного проведения всех проверок, метод serverHandshake должен добавлять новую пару (имя, соединение) в connectionMap и отправлять сообщение о том, что имя было принято.  
7. Метод serverHandshake должен возвращать имя нового клиента с которым было установлено соединение.