Создайте сериализируемый класс **Subject**:

- String name;

- int credits;

Геттеры и сеттеры

Переопределите метод **String toString()** где мы выводим все поля объекта

Создайте сериализируемый класс **Student**:

- String name;

- String surname;

- List<Subject> subjects = new ArrayList<Students>();

Геттеры и сеттеры

В главном классе создайте меню:

PRESS [1] TO ADD STUDENT

Name:

Surname:

PRESS [1] TO ADD SUBJECT

Name:

Credits:

PRESS [0] TO GO TO MAIN MENU

PRESS [2] TO LIST STUDENTS

PRESS [0] TO EXIT

Состояние общего массива ArrayList<Students> должен сохранятся в сериализируемый файл memory.dat и загружаться оттуда при обращении к списку.

**Task 2**

Создайте сериализируемый класс **Players**:

- String nickName;

- double rating;

Геттеры и сеттеры

Создайте сериализируемый класс **Game**:

- String gameName;

- String ipAddress;

- int port;

- List <Players> players = new ArrayList<Players>();

Геттеры и сеттеры

В главном классе создайте метод **void createGame()**, который создает новую игру.

Метод void **addPlayers(Player player)** должен добавить игроков в игру.

После запуска каждого метода, мы должны сохранять состояние класса в определенный файл **settings.data**

В главном классе создайте меню:

PRESS 1 TO CREATE GAME

Game name:

IP address:

PORT:

PRESS 2 TO ADD PLAYER TO GAME

Nickname:

Rating:

PRESS 3 TO PLAY GAME

PRESS 0 TO EXIT

При нажатии на 1, мы создаем игру.

При нажатии на 2, мы добавляем игрока

При нажатии на 3 мы запускаем игру, где просто выводим список игроков и данные об игре. Если мы не добавляли игроков, то мы выводим "No players".

**Сериализация** - объекта это технология сохранения полной копии его объекта на которую он ссылается, при использовании потока ввода и вывода. Например, у нас есть какой-то список динамического массива **ArrayList** или какой-то объект, и мне нужно сохранить его состояние, и в будущем обратно вернуть и использовать его обратно.

Сериализация в **Java** предоставляет возможность для преобразования групп или отдельных объектов, в поток битов или массив байтов, для хранения или передаче по сети. Данный поток битов или массив байтов, можно преобразовать обратно в объекты **Java**. И все это происходит автоматически благодаря классам **ObjectInputStream** и **ObjectOutputStream**. Разработчик может решить применить эту функцию, путем реализации интерфейса **Serializable** при создании шаблона класса. Попробуем с простого примера. Создадим простой шаблон класса **Users**:

***Users.java***

**import** java.io.Serializable;

**public class** User **implements** Serializable{

**private String** name;

**private** int number;

**public** User(){

**this**.name = "No Number";

**this**.number = 0;

}

**public** User(**String** name, int number){

**this**.name = name;

**this**.number = number;

}

**public** void setName(**String** name){

**this**.name = name;

}

**public** void setNumber(int number){

**this**.number = number;

}

**public String** getName(){

**return this**.name;

}

**public** int getNumber(){

**return this**.number;

}

}

***Main.java***

**import** java.io.**\***;

**import** java.util.**\***;

**public class** Main{

**public static** void main(**String**[] args) {

ArrayList<User> usersList = **new** ArrayList<User>();

usersList.add(**new** User("Ilyas", 123));

usersList.add(**new** User("Timur", 321));

usersList.add(**new** User("Aibek", 555));

usersList.add(**new** User("Dauren", 777));

**try**{

**ObjectOutputStream** outStream =

**new ObjectOutputStream**(**new FileOutputStream**("text.dat"));

outStream.writeObject(usersList);

outStream.close();

}**catch** (**Exception** e) {

e.printStackTrace();

}

}

}

Данный пример показывает возможность записи объектов в поток, используя класс **ObjectOutputStream**.

С начала создается класс, который является сериализуемым, так как реализует интерфейс **Serializable**.

В программе мы создаем динамический массив и добавляем туда несколько объектов класса **User**. Мы можем создавать файл с любым расширением, например, **"text.dat"**. Далее, мы записываем наш объект с помощью метода **writeObject(Object**). После этого, мы закрываем поток с помощью метода **close()** и у нас в файловой системе создаться файл **bitlab.dat** в котором и храниться наш сериализируемый объект.

Далее, нашей задачей является десериализировать наш объект динамического массива обратно.

***MainRead.java***

**import** java.io.**\***;

**import** java.util.**\***;

**public class** MainRead{

**public static** void main(**String**[] args) {

ArrayList<User> usersList = **new** ArrayList<User>();

**try**{

**ObjectInputStream** inStream =

**new ObjectInputStream**(**new FileInputStream**("text.dat"));

usersList = (ArrayList<User>)inStream.readObject();

inStream.close();

}**catch** (**Exception** e) {

e.printStackTrace();

}

**for**(User u : usersList){

**System**.out.println(u.getName() + " - " + u.getNumber());

}

}

}

В данной программе, мы точно таким же образом ссылаемся на наш **text.dat** файл, и десериализируем его с помощью класса **ObjectInputStream**. В этом классе есть метод **readObject()** который возвращает объектное значение. Для этого на придется применить метод "кастинга", то есть преобразования из одного типа в другой, где мы превращаем объект в **ArrayList**.