

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №16**

**з дисципліни «Об’єктно орієнтованне програмування»**

## на тему:

**«Серіалізація в Java.»**

Виконав:

студент гр. ПЗ2011

Кулик С. В.

Прийняв:

Андрющенко В. О.

Дніпро, 2022

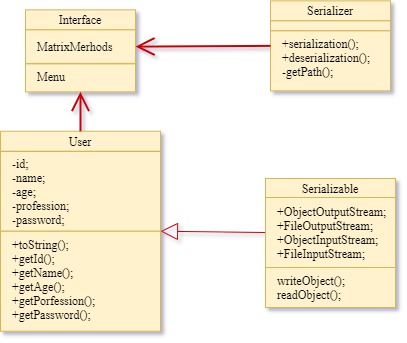
**Тема:** Серіалізація в Java.

**Мета:** Отримати навички серіалізування та десеріалузування, контролю версій.

**Постановка завдання**

Написати об'єктно-орієнтовану програму на мові Java, яка використовує серіалізацію та десеріалізацію для збереження та зчитування пов’язаних через посилання об’єктів в файл. Підготувати та виконати експерименти для перевірки збереження даних в повному обсязі та перевірки контролю версій класів, що десеріалізуються.

**Діаграма класів**



**Опис потоків**

**ObjectInputStream** і **ObjectOutputStream** є основними класами потоку об’єктів, які надаються API вводу-виводу Java File.

Клас **ObjectOutputStream** реалізує інтерфейс **ObjectOutput**, який визначає метод для запису об'єкта в вихідний потік:

* writeObject(Object) : записує об'єкт у основне сховище або потік. Цей методстворює IOException, якщо виникає помилка введення-виводу.

Процес запису об'єкта у вихідний потік називається **серіалізацією**.

Інтерфейс **ObjectOutput** поширюється на інтерфейс **DataOutput**, що означає, що **ObjectOutputStream** успадковує всю поведінку запису примітивних типів і рядків, як **DataOutputStream**.

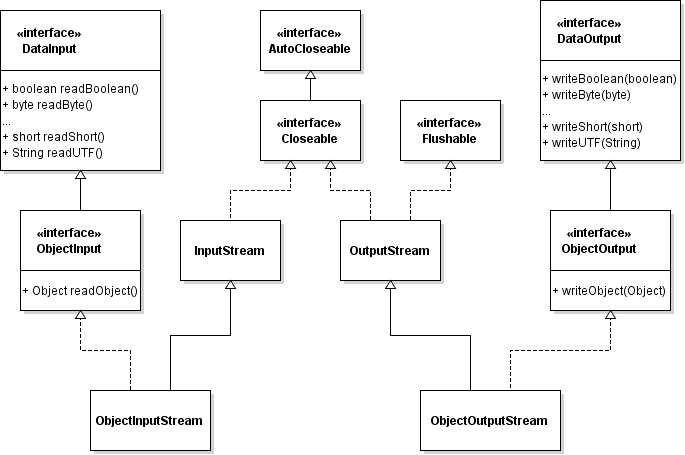
Так само, клас **ObjectInputStream** реалізує інтерфейс **ObjectInput**, який визначає метод для читання об’єкта з вхідного потоку:

* readObject() : читає і повертає об'єкт. Цей методстворює ClassNotFoundException, якщо не вдається знайти клас серіалізованого об’єкта, ігенерує IOException, якщо виникає помилка введення-виводу.

Процес реконструкції об'єкта з вхідного потоку називається **десеріалізацією**.

Інтерфейс **ObjectInput** поширюється на інтерфейс **DataInput** , що означає, що **ObjectInputStream** також має поведінку читання примітивних типів і рядків, як **DataInputStream** .

Наступна діаграма класів зображує ієрархію API класів та інтерфейсів потоку об’єктів:



Інтерфейс, що серіалізується, можна записувати та читати з потоку виведення/введення. **Serializable** — це інтерфейс маркера, який не визначає жодних методів. З **ObjectOutputStream** та **ObjectInputStream** можна використовувати лише об’єкти, позначені як «серіалізовані» .

Більшість класів у Java (включаючи Date та примітивні обгортки Integer , Double , Long тощо) реалізують інтерфейс **Serializable**.

**Код програми**

**Main.java**

import java.io.InvalidObjectException;  
import java.util.Scanner;  
  
public class Main {  
  
 public static void main(String[] args) {  
 Scanner inInt = new Scanner(System.*in*);  
  
 Print get = new Print();  
 User userS = null, userD = null;  
 Serializer serializer = new Serializer();  
  
 while (true){  
 System.*out*.println("Choose operation:\n"+  
 "1 -> Create user\n" +  
 "2 -> Serialize user\n" +  
 "3 -> Deserialize user\n" +  
 "4 -> Exit\n");  
 int choice = inInt.nextInt();  
 switch (choice){  
 case 1:  
 userS = new User(get.id(), get.name(), get.age(), get.prof(), get.pas());  
 System.*out*.println("Data has been gotten");  
 break;  
 case 2:  
 if (userS != null){  
 serializer.serialization(userS);  
 System.*out*.println("Succeed");  
 } else {  
 System.*out*.println("User not created");  
 }  
 break;  
 case 3:  
 try {  
 userD = serializer.deserialization();  
 System.*out*.println(userD.toString());  
 } catch (InvalidObjectException e) {  
 e.printStackTrace();  
 }  
 break;  
 case 4:  
 System.*out*.println("You have exited the program");  
 System.*exit*(0);  
 break;  
 default:  
 System.*out*.println("Error!\n");  
 }  
 }  
 }  
 */\*\*Class for printing in console\*/* public static class Print {  
 Scanner inInt = new Scanner(System.*in*);  
 Scanner inString = new Scanner(System.*in*);  
  
 public String id() {  
 System.*out*.print("Enter user id: ");  
 return inString.nextLine();  
 }  
  
 public String name() {  
 System.*out*.print("Enter user name: ");  
 return inString.nextLine();  
 }  
  
 public int age() {  
 System.*out*.print("Enter user age: ");  
 return inInt.nextInt();  
 }  
  
 public String prof() {  
 System.*out*.print("Enter user profession: ");  
 return inString.nextLine();  
 }  
  
 public String pas() {  
 System.*out*.print("Enter user password: ");  
 return inString.nextLine();  
 }  
 }  
}

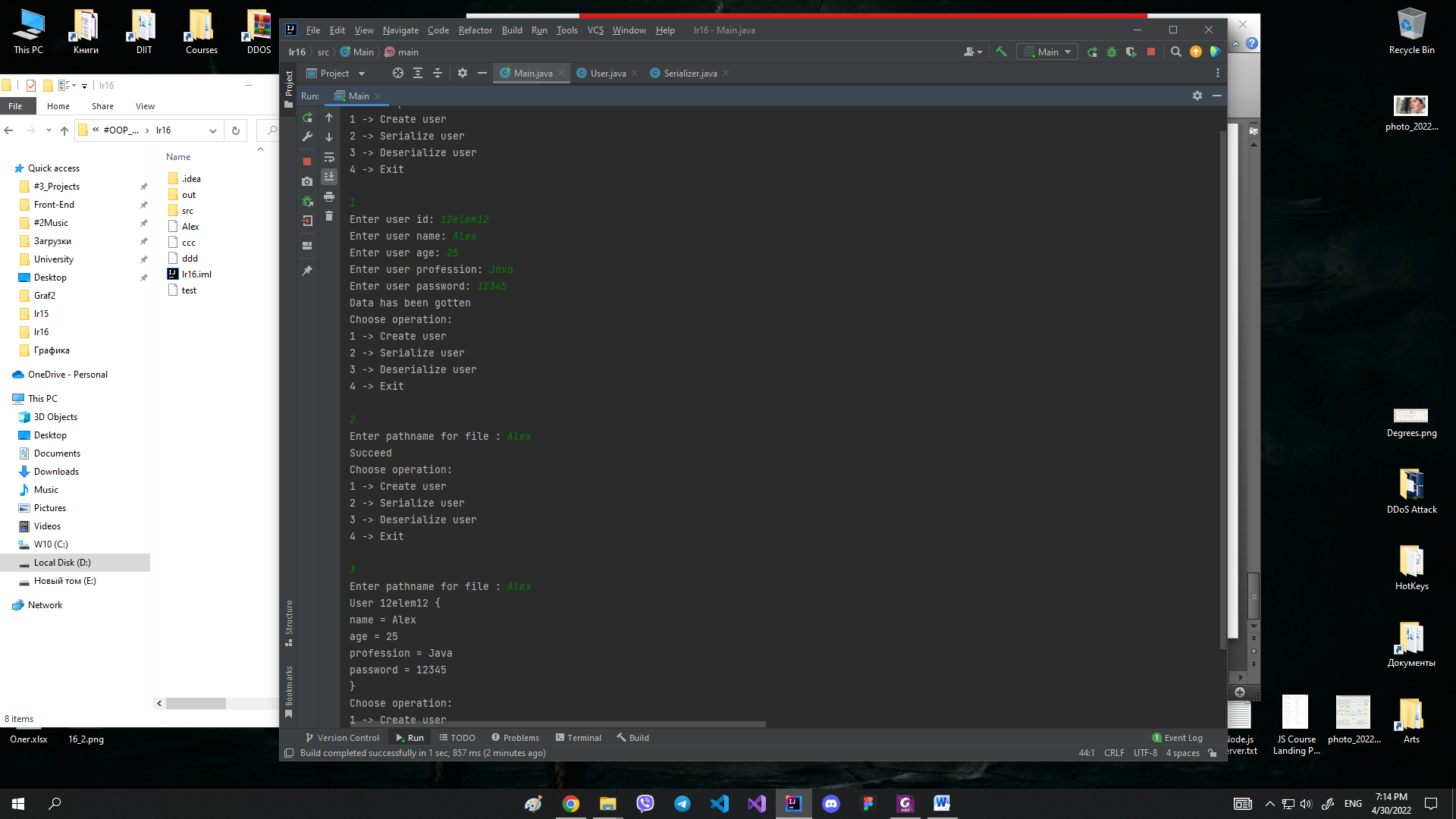
**User.java**

import java.io.Serializable;  
  
public class User implements Serializable {  
 //private static final long serialVersionUID = 1L;  
 private final String id;  
 private final String name;  
 private final int age;  
 private final String profession;  
 private final String password;  
  
 public User(String id, String name, int age, String profession, String password) {  
 this.id = id;  
 this.name = name;  
 this.age = age;  
 this.profession = profession;  
 this.password = password;  
 }  
  
 @Override  
 public String toString() {  
 return "User " + id + " {\n" +  
 "name = " + name + '\n' +  
 "age = " + age + "\n"+  
 "profession = " + profession + '\n' +  
 "password = " + password + '\n' +  
 '}';  
 }  
 public String getId() {  
 return id;  
 }  
 public String getName() {  
 return name;  
 }  
 public int getAge() {  
 return age;  
 }  
 public String getProfession() {  
 return profession;  
 }  
 public String getPassword() {  
 return password;  
 }  
  
}

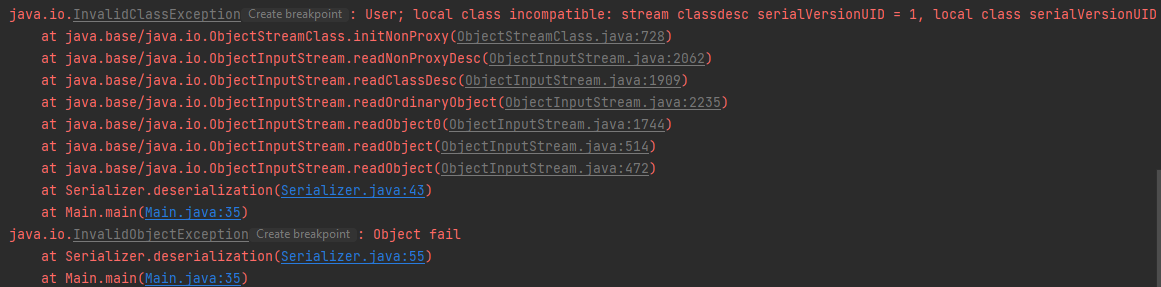
**Serializer.java**

1. import java.io.\*;  
   import java.util.Scanner;  
     
   public class Serializer {  
    private Scanner inString = new Scanner(System.*in*);  
    public boolean serialization(User user){  
    boolean flag = false;  
     
    File file = new File(getPath());  
    ObjectOutputStream oos = null;  
     
    try {  
    FileOutputStream fos = new FileOutputStream(file);  
    if (fos != null) {  
    oos = new ObjectOutputStream(fos);  
    oos.writeObject(user);  
    }  
    } catch (FileNotFoundException e) {  
    e.printStackTrace();  
    } catch (IOException e) {  
    e.printStackTrace();  
    } finally {  
    if (oos != null){  
    try {  
    oos.close();  
    } catch (IOException e) {  
    e.printStackTrace();  
    }  
    }  
    }  
     
    return flag;  
    }  
     
    public User deserialization() throws InvalidObjectException {  
    File file = new File(getPath());  
    ObjectInputStream ois= null;  
     
    try {  
    FileInputStream fis = new FileInputStream(file);  
    if (fis != null){  
    ois = new ObjectInputStream(fis);  
    User user = (User) ois.readObject();  
    return user;  
    }  
    } catch (IOException | ClassNotFoundException e) {  
    e.printStackTrace();  
    } finally {  
    try {  
    ois.close();  
    } catch (IOException e) {  
    e.printStackTrace();  
    }  
    }  
    throw new InvalidObjectException("Object fail");  
    }  
     
    private String getPath(){  
    System.*out*.print("Enter pathname for file : ");  
    return inString.nextLine();  
    }  
   }

**Приклад роботи**



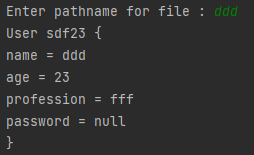
Якщо змінити представлення класу User і спробувати записати серіалізований об’єкт буде створена помилка, що свідчить про неможливість запису даного об’єкту у екземпляр даного класу. Так, у класі було поле password зі специфікатором **transient** і було створено та серіалізовано об’єкт даного класу. Після затирання данного специфікатора при спробі десеріалізувати об’єкт отримуємо помилку



Якщо до класу User додати дане поле

private static final long *serialVersionUID* = 1L;

дані будуть адаптовані під існуючий клас а додаткові поля будуть заповнені як null.



**Висновок**

**Серіалізація** - це процес збереження стану об'єкта в електронних даних.

**Десеріалізація** - це процес відновлення об'єкта з цих байт. **Java Serialization API** надає стандартний механізм для створення серіалізованих об'єктів.

Серіалізація об'єкта - це здатність об'єкта зберігати повну копію його і будь-яких інших об'єктів на які він посилається, використовуючи потік виводу (наприклад, у зовнішній файл). Таким чином, об'єкт може бути відтворений з серіалізованої (збереженої) копії трохи пізніше, коли це буде потрібно.

Серіалізація зручна, коли потрібно зберегти стан вашої проrрами в області постійного зберігання, такому як файл. Пізніше можна буде відновити ці об'єкти, використовуючи процес дeceріалізаціі.

Серіалізация також необхідна в реалізації віддаленого виклику методів (Remote Method Invocation RMI). RMI дозволяє об'єкту Java на одній машині звертатися до методу об'єкта Java на іншій машині. Об'єкт може бути застосований як аргумент цього віддаленого методу. Машина, яка посилає об'єкт, серіалізує і передає його. Машина, яка приймає об'єкт, десеріалізує його.