# ЛАБОРАТОРНАЯ РАБОТА №6

# ПРОЕКТИРОВАНИЕ КЛАССОВ

**Цель работы:** получить основные понятия и навыки по проектированию и созданию классов с использованием UML.

**Необходимая теоретическая подготовка:**

• основы UML;

• объектно-ориентированное программирование.

Порядок выполнения работы

* изучить предлагаемый теоретический материал;
* разработать программу в соответствии с вашим индивидуальным заданием (вариантом);
* использовать возможности ООП;
* каждый класс должен иметь отражающее смысл название и информативный состав.

**Задания к лабораторной работе**

Общее для всех вариантов: реализовать механизм сериализации (десериализации) объекта согласно индивидуальному заданию.

**Индивидуальное задание:**

7. Город. Определить иерархию города. Создать несколько объектоврайонов. Собрать город;

**Ход выполнения работы:**

**1. Создание проекта и пакета**

1. В **IntelliJ IDEA** был создан новый проект **Java Application**.
2. В папке src создан пакет **com.example.city** для удобной организации классов.

**2. Разработка иерархии классов**

Была определена структура города, включающая:

* **Город (City)** – содержит название и список районов.
* **Район (District)** – содержит название и численность населения.
* **Сериализатор (CitySerializer)** – отвечает за сохранение и загрузку данных в файл.

**Класс City**

package com.example.city;  
  
import java.io.Serializable;  
import java.util.List;  
  
public class City implements Serializable {  
 private static final long *serialVersionUID* = 1L;  
  
 private String name;  
 private List<District> districts;  
  
 public City(String name, List<District> districts) {  
 this.name = name;  
 this.districts = districts;  
 }  
  
 @Override  
 public String toString() {  
 return "Город: " + name + "\nРайоны:\n" + districts;  
 }  
}

**Класс District**

package com.example.city;  
  
import java.io.Serializable;  
  
public class District implements Serializable {  
 private static final long *serialVersionUID* = 1L; // UID для сериализации  
  
 private String name;  
 private int population;  
  
 public District(String name, int population) {  
 this.name = name;  
 this.population = population;  
 }  
  
 @Override  
 public String toString() {  
 return "Район: " + name + ", Население: " + population;  
 }  
}

**3. Реализация сериализации и десериализации**

package com.example.city;  
  
import java.io.\*;  
import java.util.Arrays;  
  
public class CitySerializer {  
 public static void main(String[] args) {  
 // 1. Создаем районы  
 District d1 = new District("Центральный", 50000);  
 District d2 = new District("Северный", 30000);  
 District d3 = new District("Южный", 40000);  
  
 // 2. Создаем город с районами  
 City city = new City("Москва", Arrays.*asList*(d1, d2, d3));  
  
 // 3. Сериализуем объект в файл  
 *serialize*(city, "city.ser");  
  
 // 4. Десериализуем объект из файла  
 City deserializedCity = *deserialize*("city.ser");  
  
 // 5. Выводим результат  
 System.*out*.println("Восстановленный объект:\n" + deserializedCity);  
 }  
  
 // Метод для сериализации  
 public static void serialize(City city, String filename) {  
 try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(filename))) {  
 out.writeObject(city);  
 System.*out*.println("Объект успешно сериализован в " + filename);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 // Метод для десериализации  
 public static City deserialize(String filename) {  
 try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(filename))) {  
 return (City) in.readObject();  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
}

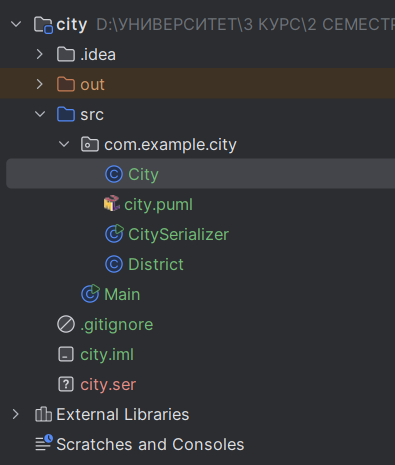


Рисунок 1 – Структура проекта

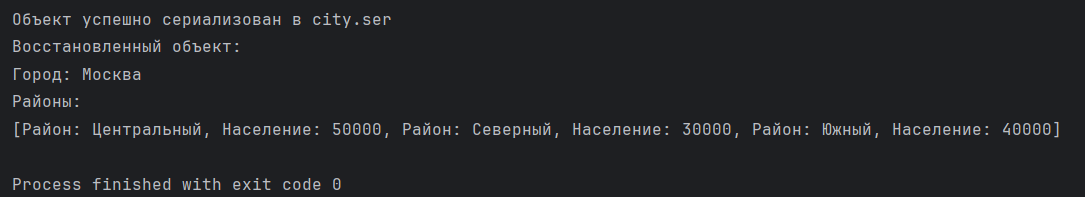


Рисунок 2 – Выполнение проекта

**4. UML-диаграмма классов**

@startuml  
class City {  
 - name: String  
 - districts: List<District>  
 + getName(): String  
 + getDistricts(): List<District>  
}  
  
class District {  
 - name: String  
 - population: int  
 + getName(): String  
 + getPopulation(): int  
}  
  
class CitySerializer {  
 + serialize(city: City): void  
 + deserialize(): City  
}  
  
City "1" -- "\*" District  
CitySerializer ..> City : uses  
  
@enduml

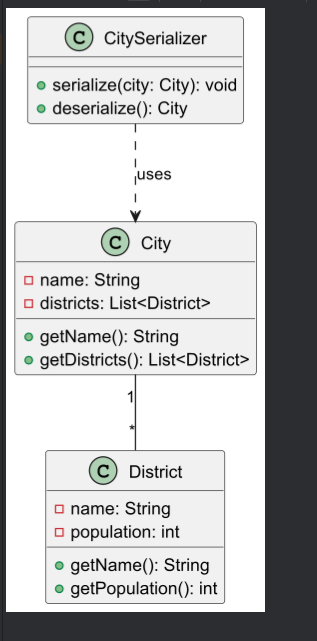


Рисунок 3 - **UML-диаграмма классов**

**Вывод**

В ходе выполнения работы были выполнены следующие задачи:

* Разработана иерархия классов для представления города.
* Реализована сериализация и десериализация объекта City с использованием ObjectOutputStream и ObjectInputStream.
* Проведено тестирование, подтверждающее корректную работу механизма.
* Создана UML-диаграмма, поясняющая структуру классов.