ЛАБОРАТОРНАЯ РАБОТА 7.
ИНТЕРФЕЙСЫ

7.1. Цель и содержание

Цель лабораторной работы: изучить принципы проектирования приложений с использованием интерфейсных типов.

Задачи лабораторной работы:

– научиться объявлять и использовать интерфейсы;

– научиться проектировать приложения с использованием интерфейсов;

– научиться реализовывать полиморфное поведение в программах с использованием интерфейсов.

7.3. Теоретическая часть

Объектно-ориентированное программирование как теория, а вместе с ней и языки программирования, непрерывно развиваются и позволяют строить все более сложные программные архитектуры. Усложнение программных комплексов, необходимость взаимодействия со сторонними системами, повсеместное распространение коллективной разработки – все эти факторы способствуют появлению новых технологий в области разработки. Одним из

нововведений в объектно-ориентированных языках стали типы-интерфейсы (или просто интерфейсы).

Интерфейсы – это способ указывать, что должны делать классы, не описывая, как они это должны делать. Классы могут реализовывать один или несколько интерфейсов. Если возникает необходимость использовать интерфейс, применяются объекты этих классов.

Интерфейс в языке Java не является классом. Он представляет собой множество требований, предъявляемых к классу, который должен соответствовать интерфейсу. Интерфейс можно понимать как контракт (описание того, что необходимо сделать). Контракт, сам по себе, ничего не делает, не реализует никаких методов, не хранит в себе поля объектов. Реализацией контракта занимается класс. Если класс реализует интерфейс, то он должен реализовать все методы интерфейса. Один интерфейс могут реализовывать несколько различных классов, при этом каждый из них делает это по-своему.

Рассмотрим пример. На рис. 7.1 представлена иерархия классов. Допустим, что проект ГИС развивался и возникла необходимость рисовать все эти объекты на карте. Это общий функционал для всех объектов предметной области, то есть метод Draw(), который рисует объекты на карте, должен быть членом класса GISObject, а каждый подкласс в иерархии должен переопределить этот метод – реализовать полиморфное поведение. В данной цепочке рассуждений есть недостатки:

1. При рисовании объектов понадобятся новые данные: цвет, толщина линий, геометрические примитивы и т.п. Все эти данные нужно будет включать в GISObject.

2. Не все объекты требуют рисования. Например, объекты классов DynamicObject и StaticObject не представляют интереса для пользователя ГИС, они только инкапсулируют общий функционал для своих подклассов. Значит, эти объекты не должны рисоваться на карте. Что делать с методом Draw()?

Оставить пустым? Возникает альтернативная классификация типов: не «подвижные – статические», а «рисуемые – невидимые».

3. Рисование на карте – это только одна область, которая может возникнуть в процессе разработки. В системе ГИС может потребоваться: добавить к выборочным типам контактную информацию; реализовать механизм сохранения; реализовать сетевое взаимодействие; добавить механизм интерактивного взаимодействия с пользователем и т.п. Новый функционал повлечет перестроение иерархии классов и(или) создаст перегруженность суперкласса иерархии.

Для разрешения проблемы предназначены интерфейсы. Объявление интерфейсного типа показано на рис. 7.1

```
3  public interface IDrawable {
4      public void Draw();
5      public String Note();
6  }
```

Рисунок 7.1 – Пример объявления интерфейса.

Все методы интерфейса автоматически считаются общедоступными, поэтому, объявляя метод в интерфейсе, указывать модификатор public не обязательно. Этот конкретный интерфейс имеет два метода (естественно, программист может определить любое количество). Важен тот факт, что интерфейсы не могут реализовываться в виде объектов. В интерфейсах нет ни полей, ни тела методов; тела методов содержатся в классах, реализующих соответствующие интерфейсы. Таким образом, интерфейс можно воспринимать как абстрактный класс, лишенный каких-либо полей экземпляра. Однако между понятиями «абстрактный класс» и «интерфейс» есть существенные различия.

Теперь предположим, что объекты класса Human должны рисоваться на карте, то есть данный класс должен реализовывать интерфейс IDrawable.

Для того чтобы класс реализовал интерфейс, нужно выполнить два действия:

1. Объявить, что класс реализует интерфейс.

2. Определить в классе все методы, указанные в интерфейсе.

Реализация классом интерфейса показана на рис. 7.2 с использованием диаграммы классов.

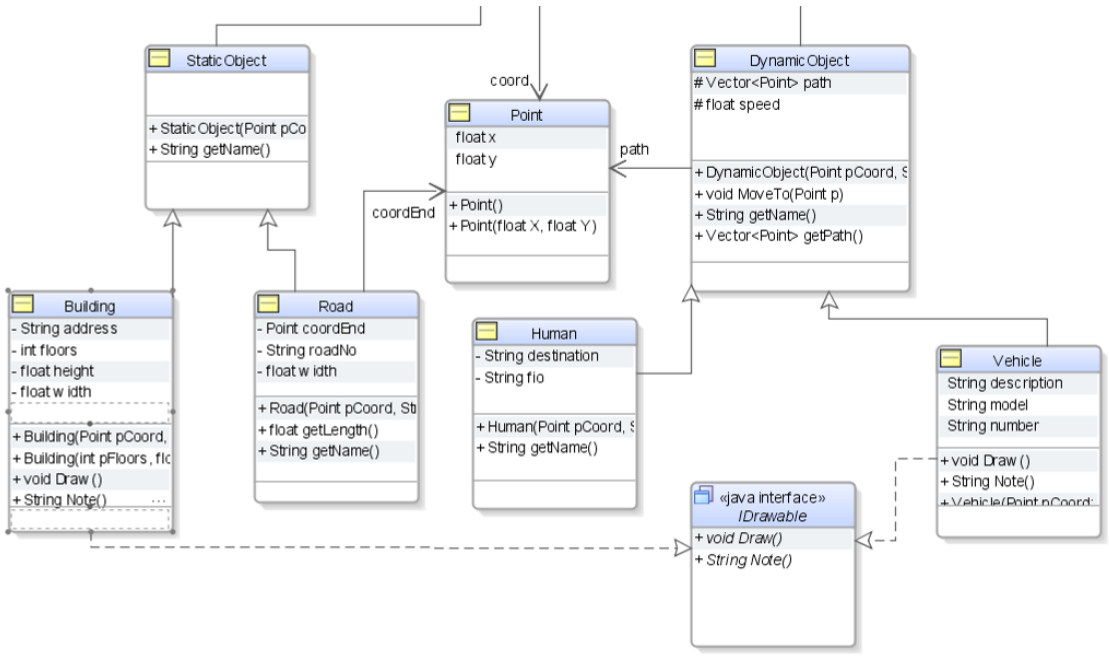

Рисунок 7.2 – Реализация интерфейса IDrawable на диаграмме.

В данном фрагменте интерфейс IDrawable реализуют классы Vehicle и Building. Это классы находятся в различных иерархических цепочках, но именно для них добавляется функционал рисования.

Синтаксически, реализация интерфейса оформляется так: указывается зарезервированное слово implements за которым следуют имена реализуемых интерфейсов. Необходимо также добавить методы, определенные в интерфейсе, в класс Vehicle (рис. 7.3).

```
public class Vehicle extends DynamicObject implements IDrawable {
    String model;
    String number;
    String description;

    public void Draw(){
        String str =
            "    -----------\n" +
            "___/             \\_______\n" +
            "|                        |\n" +
            " ---(@)------------(@)--\n";
        System.out.println(str);
    }

    public String Note(){
        return "Объект: транспорт" +
            model + " " + number;
    }
```

Рисунок 7.3 – Реализация интерфейса IDrawable классом Vehicle.

На рис. 7.4 представлено определение класса Building.

```
public class Building extends StaticObject implements IDrawable {
    private float width;
    private float height;
    private int floors;
    private String address;

    public void Draw(){
        String str =
            " /-/-/-/-/-/-/\\\n" +
            " | ___    __ |\n" +
            " ||_|_| |  ||\n" +
            " ||_|_| |  ||\n";
        System.out.println(str);
    }

    public String Note(){
        return "Объект: дом" +
            "Этажей: " + floors + "Площадь: " + getSquare();
    }

    public Building(Point pCoord, String pName) {
        super(pCoord, pName);

    }

    public Building(int pFloors, float pW, float pH) {
        this(new Point(), "Строение");
        floors = pFloors;
        width = pW; height = pH;
    }

    public float getSquare(){
        return width*height;
    }
```

```
36
    public String getName() {
38      return "Строение" +
39          "\n Этажность: " + String.valueOf(floors)
40          +"\n Адрес: " + address;
41  }
42 }
```

Рисунок 7.4 – Определение класса Building.

Оба класса реализуют интерфейс IDrawable, но каждый вкладывает свой смысл в реализацию.

На рис. 7.5 продемонстрировано манипулирование объектами разработанной иерархии классов при использовании интерфейса.

```
6  public class Starter {
7      public static void main(String[] args) {
8
9          Vector<IDrawable> allDynamics =
10             new Vector<IDrawable>();
11         allDynamics.add(new Vehicle(
12             new Point(), "Hyundai Sol", "У445КЕ", "?"));
13         allDynamics.add(new Building(5, 20.0f, 10.0f));
14         allDynamics.add(new Vehicle(
15             new Point(), "Kia Rio", "У111КЕ", "новый"));
16         allDynamics.add(new Building(2, 5.0f, 7.0f));
17         allDynamics.add(new Vehicle(
18             new Point(), "Лада", "К433ТЕ", "белый"));
19
20         // Единообразная работа со всеми элементами
21         for(int i = 0; i < allDynamics.size(); i++) {
22             System.out.println(allDynamics.get(i).Note());
23             allDynamics.get(i).Draw();
24         }
25     }
26 }
```

Рисунок 7.5 – Работа с коллекцией объектов через реализуемый интерфейс.

В данном примере добавление в коллекцию осуществляется в соответствии с правилом: добавить можно только объект того класса, который реализует интерфейс IDrawable.

На рис. 7.6 показан вывод программы.

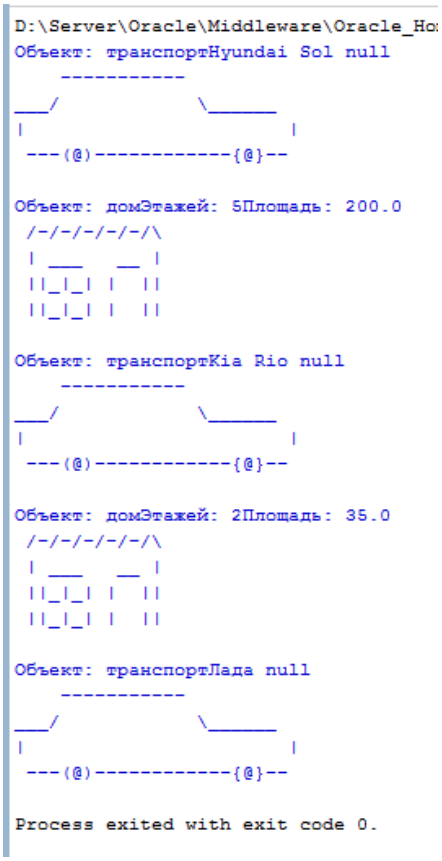

Рисунок 7.6 – Вывод программы.

Синтаксически наследование реализации (используется extends) и наследование интерфейсов (implements) отличается, но отличие существует также в идеи механизма: каждый класс может иметь только один суперкласс, а интерфейсов может реализовывать – множество. На рис. 5.5 продемонстрировано полиморфное поведение объектов коллекции, достигнутое использованием интерфейсов.

Следует понимать, что интерфейсные типы также могут образовывать иерархию.

7.6. Методика и порядок выполнения работы

Необходимо модифицировать приложение, разработанное в рамках лабораторной работы №6 следующим образом:

1. Проанализировать, какой функционал требуется дополнительно реализовать в приложении.

2. Спроектировать новые функциональные возможности с использованием инструментария диаграмм. Новый функционал добавить на базе интерфейсных типов с минимальными изменениями уже разработанной иерархии классов.

3. Модифицировать код приложения в соответствии с разработанным проектом.

4. Продемонстрируйте в программе полиморфное поведение объектов на основе использования интерфейсных типов.

7.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.

3. Ответы на контрольные вопросы.

4. Диаграмма классов, экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы, подписанный студентом, сдается преподавателю.

7.8. Контрольные вопросы

1. Что такое интерфейс?

2. Опишите каким образом представляется наследование интерфейса на диаграмме классов и в коде приложения.

3. Опишите механизм реализации полиморфизма на основе использования интерфейсов.

4. Чем интерфейс отличается от абстрактного класса?