

## Практическая работа №6. Интерфейсы в Java

**Цель:** цель данной практической работы – научиться разрабатывать практике пользовательские интерфейсы, и применять их в программах на языке Джава.

### Теоретические сведения

Интерфейс в Java это разновидность класса. В качестве компонентов интерфейс имеет поля данных только статические константы и в качестве методов только абстрактные методы.

Интерфейс в Java — это механизм для достижения определенного рода абстракции. Интерфейсе Java не может включать никаких других методов кроме абстрактных. В составе интерфейса методы только объявлены, у них нет реализации или тела метода. Это похоже на механизм виртуальных функций в языке C++. Интерфейс содержит только объявления методов, то есть тело метода отсутствует. В Джава может быть объявлен пустой интерфейс, который не содержит ни одного объявления метода. Такие интерфейсы называются этикетками. Все методы входящие в интерфейс объявляются как `abstract public`, но вы можете не писать это перед самым методом, так как все методы, входящие в интерфейс и так по умолчанию абстрактные и с открытым - `public` доступом (начиная с версии Java 8). В Java 9 методы могут быть объявлены с модификатором `private`. Интерфейсы используются для достижения абстракции и множественного наследования в языке Джава. Потому что один и тот же интерфейс может использоваться для реализации разными классами.

Интерфейс Java также представляет отношение классами “IS- a”.

### *Преимущества интерфейсов*

Существуют по крайней мере три веские причины использовать интерфейсы:

- они используются для достижения абстракции.
- Благодаря интерфейсам мы можем поддерживать механизм множественного наследования.
- они используются для достижения слабой связанности кода (`low coupling code`)<sup>1</sup>

---

<sup>1</sup> Термин `low coupling code` означает слабую связанность, то есть необходимо распределить ответственности между классами так, чтобы обеспечить минимальную связанность в коде. <https://enterprisecraftsmanship.com/posts/cohesion-coupling-difference/>

## ***Объявление интерфейсов***

Интерфейс объявляется с помощью ключевого слова `interface`. Он обеспечивает полную абстракцию; это означает, что все методы в интерфейсе объявлены с пустым телом, а все поля по умолчанию являются общедоступными, статическими и окончательными (объявлены с модификатором `final`). Класс, реализующий интерфейс, должен реализовывать все методы, объявленные в интерфейсе.

Внимание! Если вы забудете реализовать какой-нибудь из методов, то вы получите абстрактный класс.

Синтаксис объявления интерфейса:

```
interface <interface_name> {  
  
    // объявляем поля константы  
    // объявляем абстрактные методы  
    // (они по умолчанию абстрактные)  
}
```

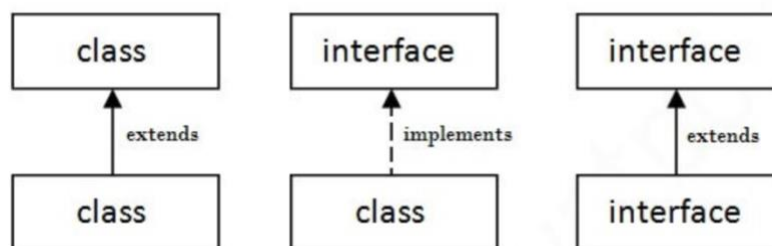
Замечание: Компилятор Java добавляет ключевые слова `public` и `abstract` перед методом интерфейса. Более того, он добавляет ключевые слова `public`, `static` и `final` перед полями данных класса.

## ***Улучшение интерфейсов, начиная с Java 8***

Начиная с Java 8, интерфейс может иметь методы по умолчанию и статические методы, которые обсуждаются позже.

## ***Отношения между классами и интерфейсами при наследовании***

Как показано на приведенном ниже рис. 6.1, класс может расширять другой класс при наследовании, интерфейс может расширять другой интерфейс при наследовании, но только класс реализует интерфейс. То есть в нем должны быть реализованы все методы интерфейса, который он при объявлении реализует.



*Рисунок 6.1. Схема отношений между классами*

### ***Пример интерфейса Java: реализация возможности рисования***

В этом примере интерфейс Drawable имеет только один метод. Его реализация обеспечивается в классах Rectangle и Circle, которые реализуют этот интерфейс. В реальном сценарии в жизни интерфейс определяется кем-то другим, но его реализация обеспечивается разными поставщиками реализации. Более того, им всегда пользуется кто-то другой. Часть реализации скрыта пользователем интерфейса, то есть классом, который и реализует интерфейс.

#### **Листинг 6.1 – Пример работы с интерфейсом**

```
// Объявление интерфейса: первым пользователем
interface Drawable {
    void draw ();
}
// Реализация: вторым пользователем
class Rectangle implements Drawable {
    public void draw () {
        System.out.println("Рисование прямоугольника\n" );
    }
}
class Circle implements Drawable {
    public void draw(){System.out.println("Рисование
круга\n" );
}
}
// Использование интерфейса: третьим пользователем
class Main {
    public static void main (String args []) {
        Drawable d = new Circle ();
        /* В реальном сценарии объект предоставляется методом,
        например, getDrawable () */
        d.draw ();
    }
}
```

На рис. 6.2 мы видим UML диаграмму трех классов и одного интерфейса. Как видно из отношений на схеме, все три класса реализуют общий интерфейс.

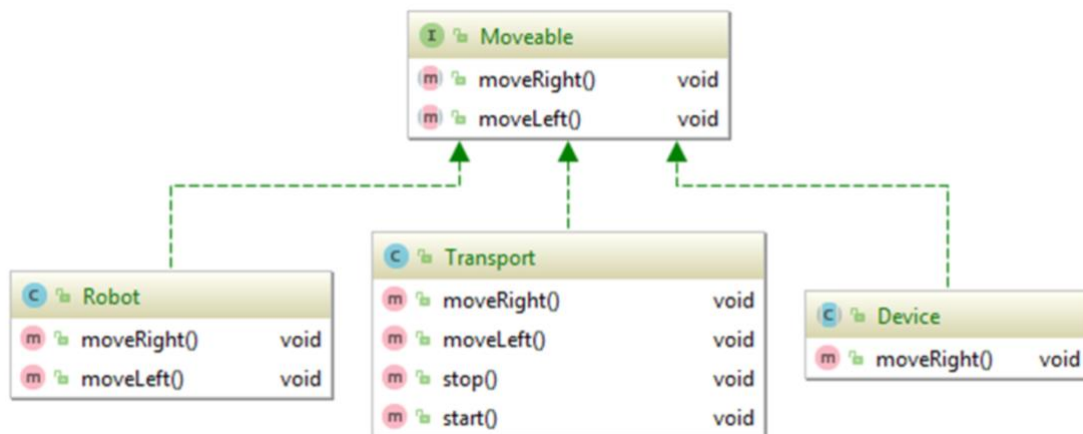


Рисунок 6.2. UML диаграмма классов

Из примера, приведенного в листинге 6.1 мы видим использование интерфейсной ссылки для инициализации объектом созданного класса. Один интерфейс в Джава может быть реализован множеством классов см. рисунок

### Задания на практическую работу №6

1. Напишите два класса MovablePoint и MovableCircle - которые реализуют интерфейс Movable (см рис. 6.3)

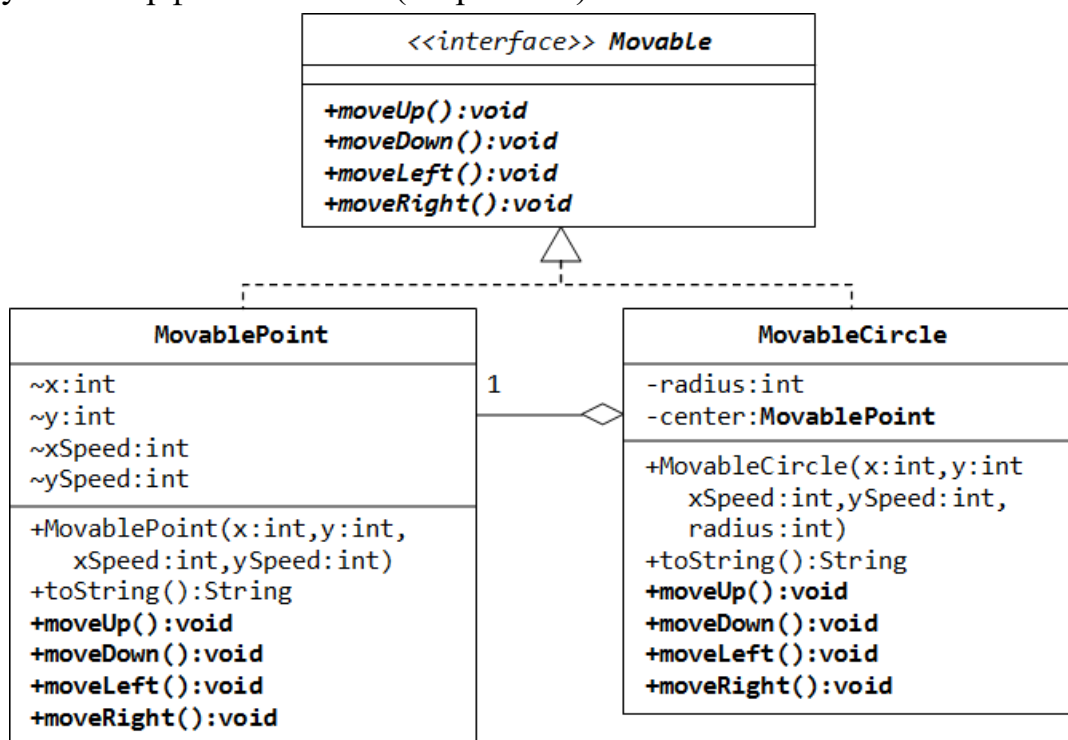


Рисунок 6.3. Диаграмма реализации интерфейса Movable.

```

public interface Movable {
    // сохраните как "Movable.java"
    public void moveUp(); //метод интерфейса
  }
  
```

```

.....
}

```

2. Напишите новый класс `MovableRectangle` (движущийся прямоугольник). Его можно представить как две движущиеся точки `MovablePoint` (представляющих верхняя левая и нижняя правая точки) и реализующие интерфейс `Movable`, см рис. 6.4. Убедитесь, что две точки имеет одну и ту же скорость (вам понадобится метод проверяющий это условие).

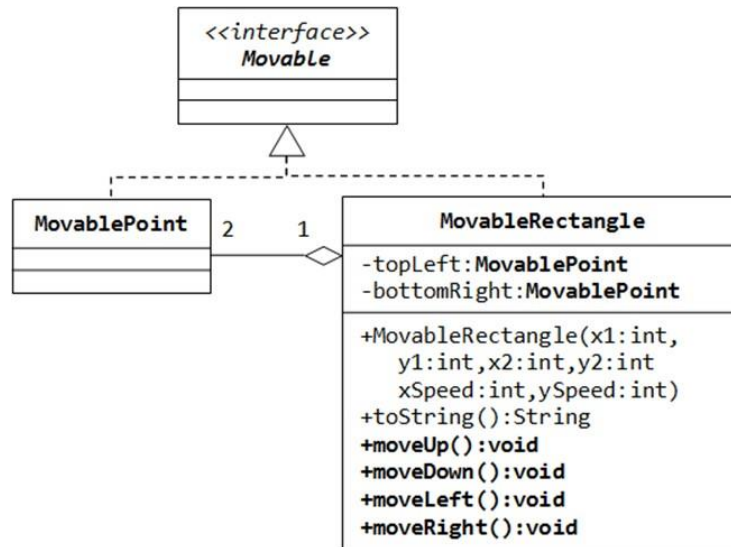


Рисунок 6.4 – Диаграмма класса `MovableRectangle`.

3. Создать интерфейс `Nameable`, с методом `getName()`, возвращающим имя объекта, реализующего интерфейс. Проверить работу для различных объектов (например, можно создать классы, описывающие разные сущности, которые могут иметь имя: планеты, машины, животные и т. д.).

4. Реализовать интерфейс `Priceable`, имеющий метод `getPrice()`, возвращающий некоторую цену для объекта. Проверить работу для различных классов, сущности которых могут иметь цену.

5. Вам нужно написать два класса `MovablePoint` и `MovableCircle` - которые реализуют интерфейс `Movable` на основе классов, разработанных в практической работе № 5. Изучите UML диаграмму и представьте реализацию класса `Замечание`. Создайте в `draw.io` UML диаграмму, а затем напишите по ней реализацию.

6. Определить интерфейс `Printable`, содержащий метод `void print()`.
7. Определить класс `Book`, реализующий интерфейс `Printable`.
8. Определить класс `Shop`, реализующий интерфейс `Printable`.

9. Создать массив типа `Printable`, который будет содержать книги и журналы. В цикле пройти по массиву объектов и вызвать метод `print()` для каждого объекта.

10. Мини приложение Интернет-магазин компьютерной техники. Создать класс, описывающий сущность компьютер (`Computer`). Для описания составных частей компьютера использовать отдельные классы (`Processor`, `Memory`, `Monitor`). Описать необходимые свойства и методы для каждого класса. Для названий марок компьютера используйте перечисления (`enum`). Разработайте класс `Shop` для, реализуйте методы добавления и удаления компьютеров в магазине, добавьте метод поиска в магазине компьютера, нужного пользователю. Протестируйте работу созданных классов. Данные для заполнения массива компьютеров вводятся с клавиатуры пользователем. Для этого реализуйте интерфейс.

11. Напишите программу для перевода температуры по Цельсию в температуру по Кельвину и Фаренгейту. Для этого добавьте интерфейс `Convertible` у которого есть метод `convert` для конвертации из одной системы измерения в другую.

12. Напишите свой класс `StringBuilder` с поддержкой операции `undo`. Для этого делегируйте все методы стандартному `StringBuilder`, а в собственном классе храните список всех операций для выполнения `undo()`. Данная программа пример реализации поведенческого шаблона «Команда»<sup>3</sup>. Пример можно посмотреть здесь <https://refactoring.guru/ru/design-patterns/command/java/example>.

13. Напишите свой класс `StringBuilder`, с возможностью оповещения других объектов об изменении своего состояния. Для этого делегируйте все методы стандартному `StringBuilder`, а в собственном классе реализуйте шаблон проектирования «Наблюдатель»<sup>4</sup>. Пример реализации можно посмотреть здесь <https://refactoring.guru/ru/design-patterns/observer/java/example>

14. Составьте отчет и представьте на проверку преподавателю

---

<sup>2</sup> <https://docs.oracle.com/javase/10/docs/api/java/lang/StringBuilder.html>

<sup>3</sup> [https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%B0\\_\(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD\\_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F\)](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%B0_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))

<sup>4</sup> [https://ru.wikipedia.org/wiki/Наблюдатель\\_\(шаблон\\_проектирования\)#Java](https://ru.wikipedia.org/wiki/Наблюдатель_(шаблон_проектирования)#Java)



