



## Урок 2

# Продвинутое ООП

Принципы ООП, классы, объекты, интерфейсы, перечисления, внутренние/вложенные/анонимные/локальные классы.

### [Интерфейсы](#)

[Объявление интерфейса](#)

[Реализация интерфейсов](#)

[Доступ к реализациям через ссылки на интерфейсы](#)

### [Перечисления](#)

[Конструкторы, методы, переменные экземпляра и перечисления](#)

### [Внутренние и вложенные классы](#)

### [Практическое задание](#)

### [Дополнительные материалы](#)

### [Используемая литература](#)

# Интерфейсы

С помощью ключевого слова `interface` можно полностью абстрагировать интерфейс класса от его реализации, то есть указать, **что** именно должен выполнять класс, но не **как** это делать. Синтаксически интерфейсы аналогичны классам, но не содержат переменные экземпляра, а объявления их методов, как правило, не содержат тело метода. Каждый интерфейс может быть реализован любым количеством классов. Кроме того, один класс может реализовать любое количество интерфейсов. Чтобы реализовать интерфейс, в классе должен быть переопределён весь набор методов интерфейса.

## Объявление интерфейса

Определение интерфейса подобно определению класса.

```
Модификатор_доступа interface имя_интерфейса {  
    возвращаемый_тип имя_метода1(список_аргументов);  
    возвращаемый_тип имя_метода2(список_аргументов);  
    тип имя_переменной1 = значение;  
    тип имя_переменной2 = значение;  
}
```

Методы интерфейса имеют модификаторы `public` и `abstract` (даже если вы это явно не указали). Каждый класс, реализующий интерфейс, должен переопределить (реализовать) все его методы. В интерфейсах могут быть объявлены поля, они неявно будут иметь модификаторы `public static final`, и обязательно должны быть инициализированы. Ниже приведён пример объявления интерфейса.

```
public interface Callback {  
    void callback(int param);  
}
```

## Реализация интерфейсов

Интерфейс может быть реализован в одном или нескольких классах. Для этого в объявлении класса необходимо добавить ключевое слово `implements` (как показано ниже), а затем переопределить методы интерфейса. Поскольку реализация интерфейса происходит с помощью ключевого слова `implements`, этот процесс получил название «имплементировать».

```
Модификатор_доступа    class    имя_класса    [extend    суперкласс]    [implements  
имя_интерфейса, ...] { }
```

Если в классе имплементируется больше одного интерфейса, их имена разделяются запятыми. Если один и тот же метод объявлен в двух интерфейсах, реализуемых в классе, он прописывается в самом классе только один раз. Рассмотрим небольшой пример класса, где реализуется приведенный ранее интерфейс `Callback`.

```
public class Client implements Callback {  
    // имплементация метода callback() интерфейса Callback  
    public void callback(int param) {  
        System.out.println("param: " + param);  
    }  
}
```

```

    }

    // метод самого класса
    public void info() {
        System.out.println("Client Info");
    }
}

```

## Доступ к реализациям через ссылки на интерфейсы

По аналогии с тем, что ссылку на объект подкласса можно записать в ссылку на суперкласс (`Animal a = new Cat(...)`), можно сделать и ссылку на объект любого класса, который реализует указанный интерфейс (`Flyable f = new Bird(...)`; где `class Bird implements Flyable`). При вызове метода по одной из таких ссылок нужный вариант будет выбираться в зависимости от конкретного экземпляра интерфейса, на который делается ссылка.

```

public interface Callback {
    void callback(int param);
}

public class ClientOne implements Callback {
    public void callback(int param) {
        System.out.println("ClientOne param: " + param);
    }
}

public class ClientTwo implements Callback {
    public void callback(int param) {
        System.out.println("ClientTwo param: " + param);
    }
}

public class TestClass {
    public static void main(String[] args) {
        Callback c1 = new ClientOne();
        Callback c2 = new ClientTwo();
        c1.callback(1);
        c2.callback(2);
    }
}

```

### Результат:

```

ClientOne param: 1
ClientTwo param: 2

```

Вызываемый вариант метода `callback()` выбирается в зависимости от класса объекта, на который переменные `c1` и `c2` ссылаются во время выполнения.

## Перечисления

В простейшей форме *перечисление* — это список именованных однотипных констант, определяющих новый тип данных, в объектах которого могут храниться только значения из этого списка. В качестве примера можно привести названия дней недели или месяцев в году — все они являются перечислениями.

Для создания перечислений используется ключевое слово `enum`.

```
public enum Fruit {  
    ORANGE, APPLE, BANANA, CHERRY  
}
```

Идентификаторы `ORANGE`, `APPLE` и т.д. — *константы перечисления*, каждая из которых неявно объявлена как `public` и `static` член перечисления `Fruit`. Тип этих констант — тип перечисления (в данном случае `Fruit`).

Определив перечисление, можно создавать переменные этого типа, как и переменные примитивных типов — без использования оператора `new`.

```
public static void main(String[] args) {  
    Fruit fruit = Fruit.APPLE;  
    System.out.println(fruit);  
    if (fruit == Fruit.APPLE) {  
        System.out.println("fruit действительно является яблоком");  
    }  
    switch (fruit) {  
        case APPLE:  
            System.out.println("fruit - яблоко");  
            break;  
        case ORANGE:  
            System.out.println("fruit - апельсин");  
            break;  
        case CHERRY:  
            System.out.println("fruit - вишня");  
            break;  
    }  
}
```

// Результат:  
// APPLE  
// fruit действительно является яблоком  
// fruit - яблоко

Поскольку переменная `fruit` относится к типу `Fruit`, ей можно присваивать только те значения, которые определены для данного типа.

Для проверки равенства констант перечислимого типа используется операция сравнения `==`. Перечисления можно использовать в качестве селектора в блоке `switch`, при этом используются простые имена констант (`APPLE`), а не уточненные (`Fruit.APPLE`).

При отображении константы перечислимого типа, например, с помощью метода `System.out.println()`, выводится её имя. Как правило, имена констант в перечислении `Fruit` указываются прописными (заглавными) буквами, поскольку они обычно играют ту же роль, что и `final`-переменные, которые традиционно обозначаются прописными буквами.

В Java перечисления реализованы как типы классов. Они отличаются от обычных классов тем, что не нужно использовать оператор `new`, и тем, что `enum` не могут выступать в роли супер- и подклассов.

Во всех перечислениях присутствуют методы: `values()` — возвращает массив, содержащий список констант, и `valueOf(String str)` — константу перечисления, значение которой соответствует строке аргументу `str`. Пример использования этих методов:

```
public static void main(String[] args) {
    System.out.println("Все элементы перечисления:");
    for(Fruit fruit : Fruit.values()) {
        System.out.println(fruit);
    }
    System.out.println("Поиск по названию: " + Fruit.valueOf("BANANA"));
}

// Результат:
// Все элементы перечисления:
// ORANGE
// APPLE
// BANANA
// CHERRY
// Поиск по названию: BANANA
```

## Конструкторы, методы, переменные экземпляра и перечисления

В перечислении каждая константа — объект класса. Таким образом, перечисление может иметь конструкторы, методы и переменные экземпляра. Если определить для объекта перечислимого типа конструктор, он будет вызываться всякий раз при создании константы перечисления. Для каждой константы перечисляемого типа можно вызвать любой метод, определённый в перечислении. Кроме того, у каждой константы перечисляемого типа есть копия любой переменной экземпляра, определённой в перечислении. Ниже приведён пример перечисления `Fruit`, к которому было добавлено название фрукта на русском языке и вес в условных единицах.

```

public enum Fruit {
    ORANGE("Апельсин", 3), APPLE("Яблоко", 3), BANANA("Банан", 2),
    CHERRY("Вишня", 1);
    private String russianTitle;
    private int weight;

    public String getRussianTitle() {
        return russianTitle;
    }

    public int getWeight() {
        return weight;
    }

    Fruit(String russianTitle, int weight) {
        this.russianTitle = russianTitle;
        this.weight = weight;
    }
}

public class Main {
    public static void main(String[] args) {
        for(Fruit fruit : Fruit.values()) {
            System.out.printf("Средний вес фрукта %s составляет: %d ед.\n",
            fruit.getRussianTitle(), fruit.getWeight());
        }
    }
}

// Результат:
// Средний вес фрукта Апельсин составляет: 3 ед.
// Средний вес фрукта Яблоко составляет: 3 ед.
// Средний вес фрукта Банан составляет: 2 ед.
// Средний вес фрукта Вишня составляет: 1 ед.

```

Итак, перечисление `Fruit` претерпело ряд изменений. Во-первых, появились две переменные экземпляра: `russianTitle` — название фрукта на русском и `weight` — средний вес фрукта в условных единицах. Во-вторых, добавлен конструктор, заполняющий поля. В-третьих, добавлены геттеры. В-четвертых, список констант перечислимого типа стал завершаться точкой с запятой, которая требуется, если класс перечисления содержит наряду с константами и другие члены.

## Внутренние и вложенные классы

Есть возможность определять один класс в другом. Он будет называться вложенным и его область действия будет ограничена областью действия внешнего класса. Так, если класс `B` определён в классе `A`, `B` не может существовать независимо от `A`. Вложенный класс имеет доступ к членам (в том числе закрытым) того класса, в который он вложен. Но внешний класс не имеет доступа к членам

вложенного класса. Вложенный класс, объявленный в области действия своего внешнего класса, считается его членом. Классы, объявленные внутри кодовых блоков, называются локальными.

Существуют два типа вложенных классов: статический и нестатический.

Статическим называется такой вложенный класс, который объявлен с модификатором `static`, поэтому он должен обращаться к нестатическим членам своего внешнего класса посредством объекта. Это означает, что вложенный статический класс не может ссылаться непосредственно на нестатические члены своего внешнего класса.

Внутренний класс — это нестатический вложенный класс. Он имеет доступ ко всем переменным и методам своего внешнего класса и может ссылаться на них так же, как это делают остальные нестатические члены внешнего класса. Ниже приведён пример работы с внутренним классом.

```
public class Outer {
    class Inner {
        private int innerVar;

        public Inner(int innerVar) {
            this.innerVar = innerVar;
        }

        void innerTest() {
            System.out.println("innerVar: " + innerVar);
            System.out.println("outerVar: " + outerVar);
        }
    }

    private int outerVar;

    public Outer(int outerVar) {
        this.outerVar = outerVar;
    }

    public void outerTest() {
        System.out.println("outerVar: " + outerVar);
        // System.out.println("innerVar: " + innerVar); тут ошибка
        Inner io = new Inner(20);
        System.out.println("io.innerVar = " + io.innerVar);
    }
}
```

## Практическое задание

1. Продолжаем работать с участниками и выполнением действий. Создайте три класса Человек, Кот, Робот, которые не наследуются от одного класса. Эти классы должны уметь бегать и прыгать, все также с выводом информации о действии в консоль.
2. Создайте два класса: беговая дорожка и стена, при прохождении через которые, участники должны выполнять соответствующие действия (бежать или прыгать), результат выполнения

печатаем в консоль (успешно пробежал, не смог пробежать и т.д.). У препятствий есть длина (для дорожки) или высота (для стены), а участников ограничения на бег и прыжки.

3. Создайте два массив: с участниками и препятствиями, и заставьте всех участников пройти этот набор препятствий. Если участник не смог пройти одно из препятствий, то дальше по списку он препятствий не идет.

## Дополнительные материалы

1. Кей С. Хорстманн, Гари Корнелл Java. Библиотека профессионала. Том 1. Основы // Пер. с англ. — М.: Вильямс, 2014. — 864 с.
2. Брюс Эккель. Философия Java // 4-е изд.: Пер. с англ. — СПб.: Питер, 2016. — 1 168 с.
3. Г. Шилдт. Java 8. Полное руководство // 9-е изд.: Пер. с англ. — М.: Вильямс, 2015. — 1 376 с.
4. Г. Шилдт. Java 8: Руководство для начинающих. // 6-е изд.: Пер. с англ. — М.: Вильямс, 2015. — 720 с.

## Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Г. Шилдт. Java 8. Полное руководство // 9-е изд.: Пер. с англ. — М.: Вильямс, 2015. — 1 376 с.