# Интерфейсы на Java

# Оригинальные интерфейсы

Интерфейсы в Java 1.0 были достаточно простыми. Они могли содержать только два типа элементов: константы и публичные абстрактные методы.

# Поля-константы

Интерфейсы могут содержать поля, как и обычные классы, но с несколькими отличиями:

- Поля должны быть проинициализированы
- Поля считаются публичными статическими финальными
- Модификаторы public, static и final не нужно указывать явно (они «проставляются» по умолчанию)

# Абстрактные методы

Наиболее важными элементами интерфейса являю
его методы. Методы интерфейса также отличаются
методов обычного класса:

```
public interface MyInterface {
    int doSomething();
    String doSomethingCompletelyDifferent();
}
```

- У методов нет тела
- Реализация методов предоставляется классами, реализующими данный интерфейс
- Методы считаются публичными и абстрактными даже, если это не задано явно
- Методы не могут быть финальными, поскольку в Java комбинация модификаторов abstract и final запрещена

# Вложенность

Java 1.1 представила концепцию классов, которь можно размещать внутри других классов. Такие классы бывают двух видов: статические и нестатические. Интерфейсы могут содержать внутри себя другие интерфейсы и классы.

```
public interface MyInterface {
    class MyClass {
        //...
    }

    interface MyOtherInterface {
        //...
    }
}
```

Даже если это не задано явно, такие интерфейсы и классы считаются публичными и статическими.

# Перечисления и аннотации

В Java 5 были введены ещё два типа: Перечисления и Аннотации.  Они могут быть помещены внутрь интерфейсов.

```
public interface MyInterface {
    enum MyEnum {
        FOO, BAR;
    }

    @interface MyAnnotation {
        //...
    }
}
```

# Обобщенные типы

Java 5 ввела концепцию «дженериков» — обобщенных типов. Они позволяют использовать обобщенный тип вместо указания конкретного типа. Таким образом, можно написать код, который работает с различным количеством типов, не жертвуя при этом безопасностью и не предоставляя отдельную реализацию для каждого типа.

```
interface Box<T> {
    void insert(T item);
}

class ShoeBox implements Box<Shoe> {
    public void insert(Shoe item) {
        //...
    }
}
```

В интерфейсах, начиная с Java 5, можно определить обобщенный тип, а затем использовать его в качестве типа возвращаемого значения метода или в качестве типа аргумента метода.
Интерфейс Box работает независимо от того, используете ли вы его для хранения объектов типа String, Integer, List или каких-либо других.

# Статические методы

Начиная с Java 8, в интерфейсы можно включать статические методы. Данный подход изменил привычный способ работы интерфейсов. Теперь они работают по-другому. Первоначально все методы в интерфейсах были абстрактными. Это означало, что интерфейс предоставлял лишь сигнатуру. Реализация оставалась за классами.

```
public interface MyInterface {

    // This works
    static int foo() {
        return 0;
    }

    // This does not work,
    // static methods in interfaces need body
    static int bar();
}
```

При использовании статических методов в интерфейсах нужно предоставить реализацию тела метода. Чтобы задействовать в интерфейсе такой метод, используйте ключевое слово static. Статические методы считаются публичными по умолчанию.

# Наследование статических методов

В отличие от обычных статических методов, статические методы не наследуются в интерфейсах. Это означает, что такой метод нужно вызывать напрямую из интерфейса, а не из реализующего его класса. Это поведение очень полезно для избегания проблем при множественном наследовании.

```
MyInterface.staticMethod();
```

# Методы по умолчанию

Методы по умолчанию похожи на статические методы - для них также нужно предоставлять тело. Чтобы объявить метод по умолчанию, используйте ключевое слово default.

```
public interface MyInterface {
    default int doSomething() {
        return 0;
    }
}
```

В отличие от статических методов, методы по умолчанию наследуются классами, реализующими интерфейс. Такие классы могут при необходимости переопределять их поведение.

Но есть одно исключение. В интерфейсе не может быть методов по умолчанию с такой же сигнатурой, как у методов toString, equals и hashCode класса Object.

# Приватные методы

С появлением Java 8 и введением методов по умолчанию и статических методов, у интерфейсов появилась возможность содержать не только сигнатуры методов, но и их реализации. При написании таких реализаций рекомендуется разделять сложные методы на более простые. Такой код легче переиспользовать, поддерживать и понимать.

Для такой цели использовали приватные методы, поскольку они могут содержать все детали реализации, которые не должны быть видимы и использованы извне.

К сожалению в Java 8 интерфейс не может содержать приватные методы. Это означает, что вы можете использовать:

- Длинные, сложные и трудные в понимании тела методов.
- Вспомогательные методы, которые являются частью интерфейса. Это нарушает принцип инкапсуляции и загрязняет публичный API интерфейса и классов-реализаций.

# Приватные методы

К счастью, начиная с Java 9, вы можете использовать приватные методы в интерфейсах, но у них есть следующие особенности:

- у приватных методов есть тело, они не абстрактные
- они могут быть как статическими, так и нестатическими
- они не наследуются классами, реализующими интерфейс, и интерфейсами
- они могут вызывать другие методы интерфейса
- приватные методы могут вызывать другие приватные, абстрактные, статические методы или методы по умолчанию
- приватные статические методы могут вызывать только другие статические и приватные статические методы

```
public interface MyInterface {

    private static int staticMethod() {
        return 42;
    }

    private int nonStaticMethod() {
        return 0;
    }
}
```

# Спасибо за внимание!