Java

Интерфейсы

Лекция #14

Пустовалова О.Г.
доцент. каф. мат.мод.
ИММиКН ЮФУ

Содержание

ИНТЕРФЕЙСЫ

Интерфейсы

Механизм наследования очень удобен, но он имеет свои ограничения.

В частности мы можем наследовать только от одного класса, в отличие, например, от языка C++, где имеется множественное наследование.

В языке Java подобную проблему частично позволяют решить интерфейсы.

Интерфейсы определяют некоторый функционал, не имеющий конкретной реализации, который затем реализуют классы, применяющие эти интерфейсы.

И один класс может применить множество интерфейсов.

Создание интерфейса

Чтобы определить интерфейс, используется ключевое слово interface.

```
interface Printable{

    void print();
}
```

Все методы интерфейса не имеют модификаторов доступа, но фактически по умолчанию доступ **public**, так как цель интерфейса - определение функционала для реализации его классом.
Поэтому весь функционал должен быть открыт для реализации.

Применение интерфейса
Пример

Пример. Применение интерфейса. Часть 1

```
public class Main{

    public static void main(String[] args) {

Book b1 = new Book("Java. Complete Referense.", "H. Shildt");
b1.print();
    }
}
interface Printable{

    void print();
}
```

Пример. Применение интерфейса. Часть 2

```
class Book implements Printable{

    String name;
    String author;

    Book(String name, String author){

        this.name = name;
        this.author = author;
    }

    public void print() {

        System.out.printf("%s (%s) \n", name, author);
    }}
```

Пример. Применение интерфейса. Часть 3

В данном случае класс Book реализует интерфейс Printable.

При этом надо учитывать, что если класс применяет интерфейс, то он должен реализовать **все методы интерфейса**, как в случае выше реализован метод print.

Потом в методе main мы можем создать объект класса Book и вызвать его метод print.

Если класс не реализует какие-то методы интерфейса, то такой класс должен быть определен как абстрактный, а его неабстрактные классы-наследники затем должны будут реализовать эти методы.

Один интерфейс можно использовать в нескольких классах

Пример. Несколько классов. Один интерфейс. Часть 1

```
public class Main{

    public static void main(String[] args) {

   Printable printable = new Book("Java. Complete Reference", "H. Shildt");
   printable.print();      //  Java. Complete Reference (H. Shildt)

   printable = new Journal("Foreign Policy");
   printable.print();      // Foreign Policy
    }
}

interface Printable{
    void print();
}
```

Пример. Несколько классов. Один интерфейс. Часть 2

```
class Book implements Printable{

    String name;
    String author;

    Book(String name, String author){

        this.name = name;
        this.author = author;
    }

    public void print() {
        System.out.printf("%s (%s) \n", name, author);
    }
}
```

Пример. Несколько классов. Один интерфейс. Часть 3

```
class Journal implements Printable {

    private String name;

    String getName(){
        return name;
    }

    Journal(String name){
        this.name = name;
    }
    public void print() {
        System.out.println(name);
    }
}
```

Интерфейсы
и преобразование типов

Интерфейсы и преобразование типов

```
Printable p =new Journal("Foreign Affairs");

p.print();

// Интерфейс не имеет метода getName,
//  необходимо явное приведение

String name = ((Journal)p).getName();

System.out.println(name);
```

явное преобразование типов: **((Journal)p).getName();**

Методы по умолчанию

Методы по умолчанию

- Ранее до JDK 8 при реализации интерфейса мы должны были обязательно реализовать все его методы в классе.
- А сам интерфейс мог содержать только **определения методов без конкретной реализации**.
- В JDK 8 была добавлена такая функциональность как методы по умолчанию.
- И теперь интерфейсы кроме определения методов **могут иметь их реализацию по умолчанию**, которая используется, если класс, реализующий данный интерфейс, не реализует метод.

Методы по умолчанию. Пример

```
interface Printable {

    default void print(){

        System.out.println("Undefined printable");
    }
}
```

Метод по умолчанию - это обычный метод без модификаторов, который помечается ключевым словом **default**.

Статические методы

Статические методы

Иногда нужно вызвать метод, еще до того, как будет возможность создавать какие-то объекты (main).

Статические методы не могут обращаться к нестатическим методам или нестатическим переменным.

Для того, чтобы обратиться к статическим методам и переменным не надо передавать никакую ссылку на объект.

Переменная или метод являются статическими, если перед ними стоит ключевое слово static.

```
int bookCount = Book.getAllBookdsCount();
```

Статические методы

```
interface Printable {

    void print();

    static void read(){

        System.out.println("Read printable");
    }
}
```

Начиная с JDK 8 в интерфейсах доступны статические методы - они аналогичны методам класса.

Как обратится к статическому методу

```
public static void main(String[] args) {

Printable printable = new Book("Java. Complete Reference", "H. Shildt");

printable.print();
printable.read();
    }
}
```

Чтобы обратиться к статическому методу интерфейса также, как и в случае с классами, пишут название интерфейса и метод.

Приватные методы

Приватные методы

- По умолчанию все методы в интерфейсе фактически имеют модификатор **public**.
- Однако начиная с Java 9 можно определять в интерфейсе методы с модификатором **private**.
- Они могут быть статическими и нестатическими, но они не могут иметь реализации по умолчанию.

Приватные методы

- Подобные методы могут использоваться только внутри самого интерфейса, в котором они определены.
- Если необходимо выполнять в интерфейсе некоторые повторяющиеся действия, и в этом случае такие действия можно выделить в приватные методы.

Приватные методы. Пример. Часть 1

```
public class Main{

    public static void main(String[] args) {

        Calculatable c = new Calculation();
        System.out.println(c.sum(1, 2));
        System.out.println(c.sum(1, 2, 4));
    }
}

class Calculation implements Calculatable{


}
```

Приватные методы. Пример. Часть 2

```
interface Calculatable{


    default int sum(int a, int b){
        return sumAll(a, b);
    }
    default int sum(int a, int b, int c){
        return sumAll(a, b, c);
    }
```

Приватные методы. Пример. Часть 3

```
    private int sumAll(int... values){
        int result = 0;
        for(int n : values){
            result += n;
        }
        return result;
    }
}
```

Константы в интерфейсах

Статические константы

Кроме методов в интерфейсах могут быть определены статические константы.

```
interface Stateable{

    int OPEN = 1;

    int CLOSED = 0;


    void printState(int n);

}
```

Хотя такие константы также не имеют модификаторов, но по умолчанию они имеют модификатор доступа **public static final**, и поэтому их значение доступно из любого места программы.

Статические константы. Пример. Часть 1

```
public class Main{


    public static void main(String[] args) {


        WaterPipe pipe = new WaterPipe();

        pipe.printState(1);

    }

}
```

Water is opened

Статические константы. Пример. Часть 2

```
class WaterPipe implements Stateable{


    public void printState(int n){
        if(n==OPEN)
            System.out.println("Water is opened");
        else if(n==CLOSED)
            System.out.println("Water is closed");
        else
            System.out.println("State is invalid");
    }
}
```

Статические константы. Пример. Часть 3

```
interface Stateable{


    int OPEN = 1;

    int CLOSED = 0;


    void printState(int n);

}
```

Множественная реализация интерфейсов

Множественная реализация интерфейсов

Если нужно применить в классе несколько интерфейсов, то они все перечисляются через запятую после слова implements.

```
interface Printable {

    // методы интерфейса
}

interface Searchable {

    // методы интерфейса
}

class Book implements Printable, Searchable{

    // реализация класса
}
```

Наследование интерфейсов

Наследование интерфейсов

Интерфейсы, как и классы, могут наследоваться.

```
class Book implements BookPrintable {

    // реализация класса
}

interface BookPrintable extends Printable{

    void paint();
}
```

При применении этого интерфейса класс Book должен будет реализовать как методы интерфейса BookPrintable, так и методы базового интерфейса Printable.

Вложенные интерфейсы

Вложенные интерфейсы

Как и классы, интерфейсы могут быть вложенными, то есть могут быть определены в классах или других интерфейсах.

```
class Printer{

    interface Printable {

        void print();
    }
}
```

Вложенные интерфейсы

При применении такого интерфейса нам надо указывать его полное имя вместе с именем класса:

```
public class Journal implements Printer.Printable {

    String name;

    Journal(String name){

        this.name = name;
    }
    public void print() {
        System.out.println(name);
    }}
```

Вложенные интерфейсы

Использование интерфейса будет аналогично предыдущим случаям:

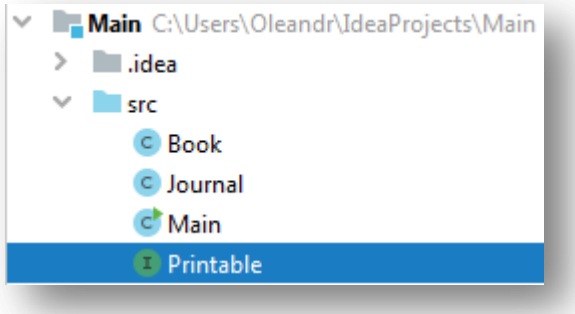
```
Printer.Printable p =new Journal("Foreign Affairs");

p.print();
```

Интерфейсы как параметры и результаты методов

Интерфейсы как параметры методов. Пример. Часть 1

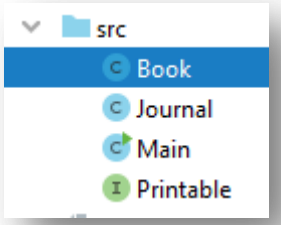
Это интерфейс Printable с единственным методом print():

```
interface Printable{

    void print();

}
```



Интерфейсы как параметры методов. Пример. Часть 2

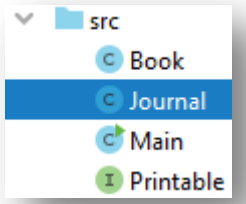
Это класс **Book**, реализующий метод print()




```
class Book implements Printable{

    String name;
    String author;

    Book(String name, String author){

        this.name = name;
        this.author = author;
    }

     public void print() {

         System.out.printf("%s (%s) \n", name, author);
    }
}
```

Интерфейсы как параметры методов. Пример. Часть 2

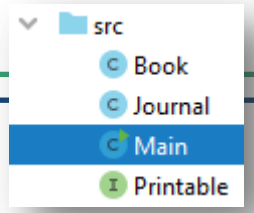
Это класс **Journal**, реализующий метод print()




```
class Journal implements Printable {

    private String name;

    String getName(){
        return name;
    }

    Journal(String name){

        this.name = name;
    }

    public void print() {
        System.out.println(name);
     }
}
```

Интерфейсы как параметры методов. Пример. Часть 4

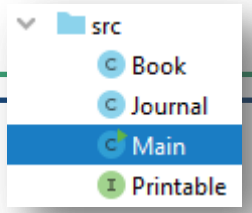
Это главная программа. В ней метод read с входным параметром - интерфейсом




```
public class Main{

    static void read(Printable p){

        p.print();
    }
```

Интерфейсы как параметры методов. Пример. Часть 5

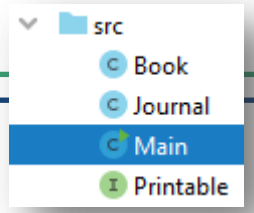
Это главная программа. В ней метод createPrintable с выходным параметром - интерфейсом




```
static Printable createPrintable(String name, boolean option)
{

    if(option)
        return new Book(name, "Undefined");
    else
        return new Journal(name);
}
```

Интерфейсы как параметры методов. Пример. Часть 6

Это главная программа.
В ней применяются методы print и createPrintable:




```
public static void main(String[] args) {

    Printable printable = createPrintable("Foreign Affairs",false);
    printable.print();

    read(new Book("Java for impatients", "Cay Horstmann"));
    read(new Journal("Java Dayly News"));
}}
```

Foreign Affairs
Java for impatients (Cay Horstmann)
Java Dayly News

Спасибо за внимание!