



Контрольная работа - 4 семестр

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.

Чтобы определить интерфейс, используется ключевое слово `interface`. Например:

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

Данный интерфейс называется `Printable`. Интерфейс может определять константы и методы, которые могут иметь, а могут и не иметь реализации. Методы без реализации похожи на абстрактные методы абстрактных классов. Так, в данном случае объявлен один метод, который не имеет реализации.

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.

Чтобы класс применил интерфейс, надо использовать ключевое слово `implements`:

```
public class Program{  
  
public static void main(String[] args) {  
  
Book b1 = new Book("Java. Complete Reference.", "H. Schildt");  
b1.print();  
}  
}  
interface Printable{  
  
void 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);  
}  
}
```

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.

В тоже время мы не можем напрямую создавать объекты интерфейсов, поэтому следующий код не будет работать:

```
Printable pr = new Printable();
pr.print();
```

Одним из преимуществ использования интерфейсов является то, что они позволяют добавить в приложение гибкости. Например, в дополнение к классу Book определим еще один класс, который будет реализовывать интерфейс Printable:

```
class Journal implements Printable {
```

```
    private String name;
```

```
    String getName(){
        return name;
    }
```

```
    Journal(String name){
```

```
        this.name = name;
    }
```

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

Класс Book и класс Journal связаны тем, что они реализуют интерфейс Printable. Поэтому мы динамически в программе можем создавать объекты Printable как экземпляры обоих классов:

```
public class Program{
```

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

```
        Printable printable = new Book("Java. Complete Reference", "H. Schildt");
        printable.print(); // Java. Complete Reference (H. Schildt)
        printable = new Journal("Foreign Policy");
        printable.print(); // Foreign Policy
    }
}
```

```
interface Printable{
```

```
    void 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);
    }
}
```

```
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 в преобразованиях типов

Все сказанное в отношении преобразования типов характерно и для интерфейсов. Например, так как класс `Journal` реализует интерфейс `Printable`, то переменная типа `Printable` может хранить ссылку на объект типа `Journal`:

```
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);
```

И если мы хотим обратиться к методам класса `Journal`, которые определены не в интерфейсе `Printable`, а в самом классе `Journal`, то нам надо явным образом выполнить преобразование типов: `((Journal)p).getName()`;

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. Например, создадим метод по умолчанию в интерфейсе `Printable`:

```
interface Printable {  
  
    default void print(){  
  
        System.out.println("Undefined printable");  
    }  
}
```

Метод по умолчанию - это обычный метод без модификаторов, который помечается ключевым словом `default`. Затем в классе `Journal` нам необязательно этот метод реализовать, хотя мы можем его и переопределить:

```
class Journal implements Printable {  
  
    private String name;  
  
    String getName(){  
        return name;  
    }  
    Journal(String name){  
  
        this.name = name;  
    }  
}
```

Статические методы

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

```
interface Printable {  
  
    void print();  
  
    static void read(){  
  
        System.out.println("Read printable");  
    }  
}
```

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

```
public static void main(String[] args) {  
  
    Printable.read();  
}
```

Приватные методы

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

Подобные методы могут использоваться только внутри самого интерфейса, в котором они определены. То есть к примеру нам надо выполнять в интерфейсе некоторые повторяющиеся действия, и в этом случае такие действия можно выделить в приватные методы:

```
public class Program{

    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{

}

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);
    }

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

Константы в интерфейсах

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.

Применение констант:

```
public class Program{

    public static void main(String[] args) {

        WaterPipe pipe = new WaterPipe();
        pipe.printState(1);
    }
}

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");
    }
}
```

```
interface Stateable{
```

```
int OPEN = 1;
```

```
int CLOSED = 0;
```

```
void printState(int n);
```

```
}
```

Множественная реализация интерфейсов

Если нам надо 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

Интерфейсы, как и классы, могут наследоваться:

```
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

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

```
class Printer{
```

```
interface Printable {
```

```
void print();
```

```
}
```

```
}
```

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);
```

```
}
```

```
}
```

Использование интерфейса будет аналогично предыдущим случаям:

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

```
p.print();
```

Интерфейсы как параметры и результаты методов

И также как и в случае с классами, интерфейсы могут использоваться в качестве типа параметров метода или в качестве возвращаемого типа:

```

public class Program{

public static void main(String[] args) {

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

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

static void read(Printable p){

p.print();
}

static Printable createPrintable(String name, boolean option){

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

interface Printable{

void 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);
}
}

class Journal implements Printable {

private String name;

String getName(){
return name;
}

Journal(String name){

this.name = name;
}

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

```

Метод read() в качестве параметра принимает объект интерфейса Printable, поэтому в этот метод мы можем передать как объект Book, так и объект Journal.

Метод createPrintable() возвращает объект Printable, поэтому также мы можем вернуть как объект Book, так и Journal.

Консольный вывод:

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

Модификаторы доступа

◀ [Объявления](#)

Перейти на...