Интерфейсы

[модификатор] **interface** ИмяНовогоИнтерфейса

[**extends** список Интерфейсов]

{Тело интерфейса, состоящее из описаний абстрактных методов и констант}

Интерфейс позволяет иметь различные реализации методов в разных классах и обращаться через него к объекту.

Интерфейсы имеют следующие ограничения:

- Модификатор доступа — могут быть только public или отсутствовать (тогда, поумолчанию, интерфейс доступен только членам пакета, в котором он объявлен).
- Методы — могут быть только абстрактными методами;
- Поля — final, static (константы, не меняющие значений, такие спецификации для них назначаются автоматически, должны быть инициализированы постоянными значениями);
- Сами интерфейсы — не могут иметь конструкторы и реализации методов.
- Нельзя создать объект типа интерфейса (но можно использовать в качестве типа — интерфейсные ссылки).

Интерфейсные константы

Интерфейсы можно использовать для импорта в различные классы совместно используемых констант.

```
public interface MyConstants
        {
   public static final double price = 1450.00;
   public static final int counter = 5;
        }
interface MyColors {
   int RED = 1, YELLOW = 2, BLUE = 4;
}
```

Описание и реализация методов интерфейса

```
public interface MyInterface
        {
    abstract public void add(int x, int y);
             void volume(int x,int y, int z);
        }
class Demo1 implements MyInterface                  class Demo2 implements MyInterface
    {                                  {
    public void add(int x, int y)                  public void add(int x, int y)
        {                                  {
    System.out.println(     +(x+y));               System.out.println(     +(x*y));
        }                                  }

    public void volume(int x, int y, int z)            public void volume(int x, int y, int z)
        {                                  {
    System.out.println(     +(x*y*z));             System.out.println(     +(x-y-z));
        }                                  }

public static void main(String args[])
        {
     MyInterface  d1= new Demo1();                  MyInterface d2= new Demo2();
    d1.add(10,20);                          d2.add(10,20);
    d1.volume(10,10,10);                        d2.volume(10,10,10);
        } }
```

- Интерфейс можно использовать как ссылочный тип при объявлении переменных.
- Переменная или выражение типа интерфейса могут ссылаться на любой объект, который является экземпляром класса, реализующего данный интерфейс.
- Переменную типа интерфейса можно использовать только после присвоения ей ссылки на объект ссылочного типа, для которого был реализован данный интерфейс.

Преобразование типов в интерфейсах

```
public interface Printable{
    void print();
}
public class Journal implements Printable {
    private String name;
     String getName(){ return name;     }
     Journal(String name){this.name = name;    }
   public void print() {
      System.out.printf("Журнал '%s'\n", name);
   }
}
Printable p =new Journal("Хакер");
p.print();

// Интерфейс не имеет метода getName, необходимо явное приведение
String name = ((Journal)p).getName();
System.out.println(name);
```

Методы по умолчанию в JDK 8

```
interface Printable {
    default void print(){
    System.out.println("Неизвестное печатное издание");
   }
}
public class Journal implements Printable {
// метод print() можно в классе не реализовывать
   private String name;

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

      this.name = name;
   }
}
```

Статические методы в интерфейсах JDK 8

```
interface Printable {

    void print();

    static void read(){

        System.out.println("Чтение печатного издания");
    }
}
public static void main(String[] args) {

    Printable.read();
}
```

Вложенные интерфейсы

Можно вкладывать описание интерфейса внутрь описания класса или другого интерфейса.

```
/описание класса
class SomeClass {
void MethodSomeClass(){}
 //описание вложенного интерфейса
interface SomeClassItf{
void SomeMethod();
} }

 //описание внешнего интерфейса
interface OuterInterface {
void OuterInterfaceMethod();
 //описание вложенного интерфейса
interface InnerInterface {
void InnerInterfaceMethod();
} }
class A implements OuterInterface.InnerInterface, SomeClass.SomeClassItf {
… // реализация InnerInterfaceMethod и SomeMethod
}
```

Использование вложенного интерфейса идет через имя внешнего класса или интерфейса:

```
SomeClass.SomeClassItf si = new A();
si.SomeMethod();

OuterInterface.InnerInterface ii = new A();
ii.InnerInterfaceMethod();
```

Наследование интерфейсов

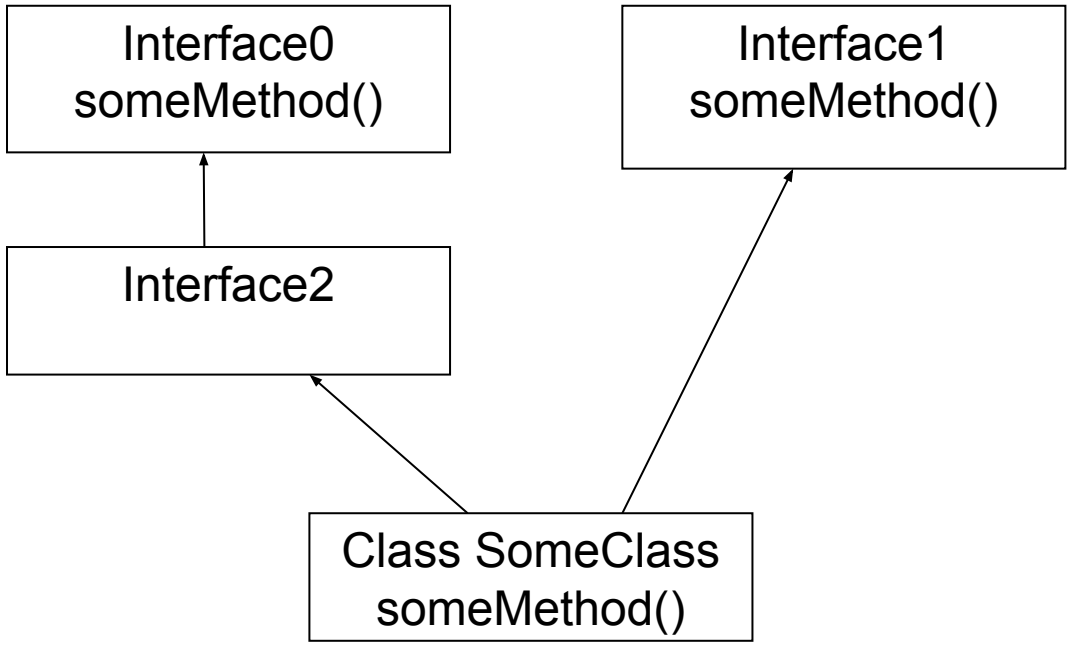
```
//суперинтерфейс A
interface A {
 int a_value = 1;
 void A();
}
 //интерфейс B расширяет интерфейс A
interface B extends A{
 int b_value = 2;
 void B();
}
 //интерфейс C расширяет интерфейс B
interface C extends B{
 int c_value = 3;
 void C();
}

class Test implements C{ . . . }
Test t = new Test();
 t.A();
 t.B();
 t.C();
```

Использование констант при множественном наследовании интерфейсов

```
public interface I1 {
   Double PI=3.14;
}
public interface I2 {
   Double PI=3.1415;
}
class C1 implements I1,I2 {
  void m1(){
    System.out.println("I1.PI="+ I1.PI);
    System.out.println("I2.PI="+ I2.PI);
  };
}
```

Для использования констант с одинаковыми именами из разных интерфейсов решением является квалификация имени константы именем соответствующего интерфейса

Наследование интерфейсов и реализация интерфейсов

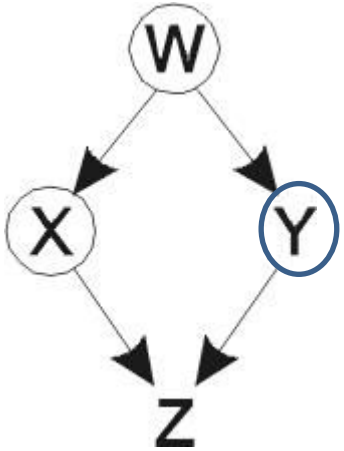


- Класс должен полностью реализовать все методы интерфейса, либо часть методов, но в этом случае должен быть объявлен как абстрактный.
- Если класс реализует несколько интерфейсов, в которых есть одноимённые методы, то в нём может задаваться лишь одна реализация общая для всех этих методов

Использование переменных типа интерфейс

```
interface Interface0 {    int someField = 10;     String someMethod(); }
interface Interface1 {     int someField = 100;     String someMethod(); }
interface Interface2 extends Interface0{     int someField = 200;     String someMethod(); }
class SomeClass implements Interface1, Interface2 {
   public String someMethod() {    return "Метод";     }
}
public class Main {
      public static void main(String[] args) {
        SomeClass a = new SomeClass();
       Interface1 I1=a;
         System.out.println( a.someMethod() );                              // Метод
    //   System.out.println( a.someField );                                 // ошибка
      System.out.println( ( (Interface1) a).someField );                  // 100
       System.out.println( Interface1.someField );                         // 100
      Interface2 I2=a;
      System.out.println( I2.someField );                  // 200
       System.out.println( I2.someMethod() );                    // Метод
       Interface0 I0=a;
       System.out.println( I0.someField );                  // 10
       System.out.println( Interface0.someField );               // 10
      }
}
```

Конфликты имен




```
interface W { }
interface X extends W { }
interface Y extends W { }
class Z implements X, Y { }
```

Если интерфейсы X и Y содержат одноименные методы с разным количеством или типом параметров, то Z будет содержать два перегруженных метода с одинаковыми именами, но разными сигнатурами.
Если же сигнатуры в точности совпадают, то Z может содержать лишь один метод с данной сигнатурой.
Если методы отличаются лишь типом возвращаемого значения, вы не можете реализовать оба интерфейса.

Если два метода отличаются только типом возбуждаемых исключений, метод класса обязан соответствовать обоим объявлениям с одинаковыми сигнатурами , но может иметь не больший список возможных исключений.

Пример конфликтов имен

```
interface I1 { void f(); }
interface I2 { int f(int i); }
interface I3 { int f(); }
class C { public int f() { return 1; } }
class C2 implements I1, I2 {
public void f() {}
public int f(int i) { return 1; } // перегружен }
class C3 extends C implements I2 {
public int f(int i) { return 1; } // перегружен }
class C4 extends C implements I3 {
// Одинаковы f () в C и I3, нет проблем:
public int f() { return 1; } }
 // Методы различаются только возвращаемым типом:
class C5 extends C implements I1 {} //Ошибка
interface I4 extends I1, I3 {} //Ошибка
```