



Kotlin

函数表达式

- Statement & Expression
- Java 声明式语法
- 声明 no return value
- Kotlin 表达式语法
- 表达式 return value
- 如果函数是一个表达式就可以用=连接
- 如果是多行的语句块，就不能加“=”
- Scala “万物皆表达式” 语句块、表达式、函数

函数表达式

- 对于一个普通函数

```
fun add(a:Int,b:Int):Int{  
    return a+b  
}
```

- 函数体只有一行,可以简写

```
fun add(a:Int,b:Int):Int = a+b
```

- 还可以更加简洁

```
fun add(a:Int,b:Int) = a+b
```

函数表达式

- 定义函数变量保存函数引用
`val addFun1:(Int,Int)->Int = ::add`
- 执行这个函数
第一种: `addFun1(10,20)`
第二种: `addFun1.invoke(10,20)`
- 定义函数变量时定义函数(这个函数也是求a+b的和)
`var addFun:(Int,Int)->Int={a,b->a+b }`
- 执行这个函数
第一种: `val result = addFun(10, 20)`
第二种: `val result = addFun.invoke(10, 20)`

函数默认参数和具名参数

- 发送网络请求:path 请求方式GET
- 默认参数:定义函数变量时可以指定默认值

```
fun info(name:String = "zhangsan",age:Int){  
    println("name=$name age=$age")  
}
```
- 具名参数:调用函数时可以指定参数(没有顺序)

```
info(age=age,name = "李四")
```

可变参数

- 如果接收的参数个数不确定,可以用可变参数表示

```
fun add(vararg arr: Int): Int {  
    var result = 0  
    arr.forEach {  
        result += it  
    }  
    return result  
}
```

什么是异常

- 开车, 轮胎没气了
- 喝水, 水杯漏了
- 洗澡, 停水了

异常处理

- 处理异常
- 受检异常和运行时异常
- Kotlin没有受检异常

关于kotlin的checked exception

- 王垠:kotlin和Checked Exception

<https://news.cnblogs.com/n/570148>

- 知乎:如何评价王垠的《Kotlin和Checked Exception》？

<https://www.zhihu.com/question/60240474/answer/175002901>

递归

什么是递归

从前有座山，山里有座庙，庙里有个老和尚，
给小和尚讲故事。故事讲的是：

从前有座山，山里有座庙，庙里有个老和尚，
给小和尚讲故事。故事讲的是：

从前有座山，山里有座庙，庙里有个老和尚，
给小和尚讲故事。故事讲的是：

从前有座山，山里有座庙，庙里有个老和尚，
给小和尚讲故事...

包子馅的包子



递归

- 递归:把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解
- 只需少量的程序就可描述出解题过程所需要的多次重复计算

递归练习

- 求第n个斐波那契数列? 1 1 2 3 5 8 13 21...

尾递归优化

- 递归和迭代的对比

尾递归优化

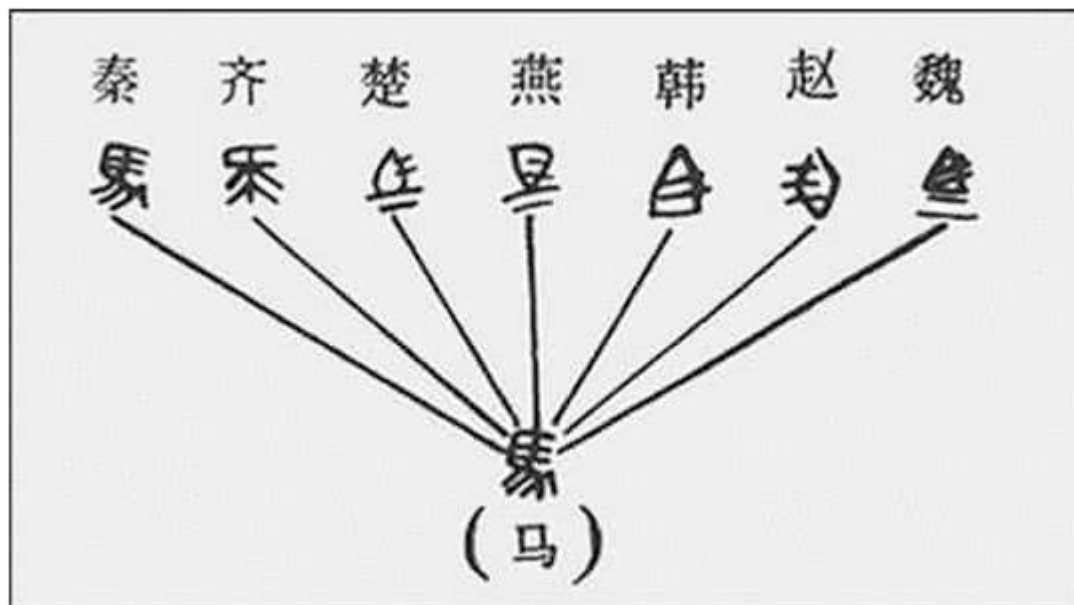
- 尾递归:函数在调用自己之后没有执行其他任何的操作就是尾递归
- 实现尾递归优化

```
tailrec fun gsTaAdd(n: Int,result:Int): Int {  
    if (n == 1) {  
        return result+1  
    } else {  
        return gsTaAdd(n - 1,result+n)  
    }  
}
```


面向对象入门



面向对象入门



面向对象入门



度量衡的统一大大提高了人民的生产劳动效率

常见容器的类型

Boolean true或false

Byte 128~127

Char 字符

Short 32768~32767

Int 2147483648~2147483647

Long 9223372036854775807~9223372036854775807

Float 小数,小数点可以精确到6位

Double 小数,小数点可以精确到15-16位

String 字符串,用""双引号引起来的字符串都可以存

描述的类型不是基本类型怎么办？

- 任何复杂的数据类型都是由基本的数据类型组成的



为什么没有90块的面额？

面向对象

- 用基本的数据类型描述复杂的事物
- 描述矩形
- 描述妹子

面向对象入门

```
class Person1{  
    var name:String = "zhangsan"  
    var age = 20  
    override fun toString(): String {  
        return "Person(name=$name age=$age)"  
    }  
}
```

运算符重载

Java运算符

- 算术运算符 + - * / %
- 自增自减 ++ --
- 关系运算符 == != > < >= <=
- 逻辑运算符 && || & |
- 赋值运算符 += -= *= /= %=
- 其他运算符 xx?xx:xx instanceof

运算符重载

- 基本运算符: + - * /

$$1+1=2$$

$$1+2=3$$

$$1*2=2$$

.....



+



=

?

运算符重载

- 运算符重载:对于+-* /等运算来说,其实就是一个函数

$a + b$ $a.\text{plus}(b)$

$a - b$ $a.\text{minus}(b)$

$a * b$ $a.\text{times}(b)$

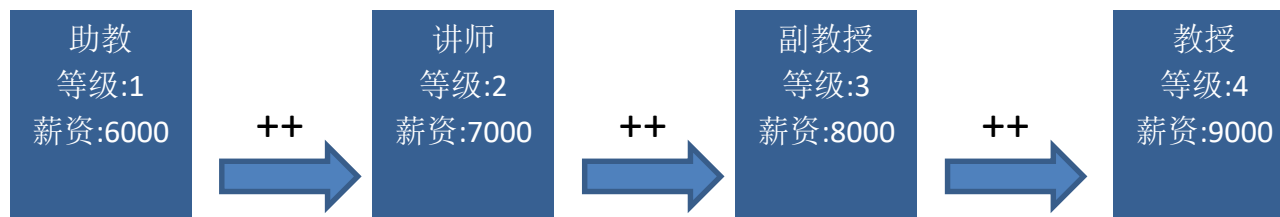
a / b $a.\text{div}(b)$

.....

运算符重载的实现

```
class BeautifulGirl{  
    var name:String = ""  
    var age:Int = 0  
    //+运算符  
    operator fun plus(a:Int):BeautifulGirl{  
        age = age+a  
        return this  
    }  
  
    override fun toString(): String {  
        return "BeautifulGirl(name=' $name', age=$age)"  
    }  
}
```


运算符重载练习



类成员的get和set方法

- 对于kotlin成员变量来说,已经默认实现了get和set方法

修改访问器的可见性

- 修改访问器的可见性

var name: String = "张三"

private set

自定义访问器

- 自定义访问器

```
var name: String = ""  
    get() {  
        return "李四"  
    }  
    set(value) {  
        field = value //这里field代表name字段  
    }
```

主构造函数

- 定义Person对象:name age

- 主构造函数

```
class Person(var name:String,var age:Int){  
}
```

init

```
class Person(){//这里如果没有()就代表没有无参构造函数  
    init {  
        println("执行了init方法")  
    }  
    constructor(name:String):this()  
}
```

- 调用主构造函数

```
val person = Person()
```

会执行init

- 调用次构造函数

```
val person = Person("张三")
```

也会执行init

保存主构中的参数

- 在init方法中保存

```
class Person(name:String,age:Int){  
    var name = ""  
    var age = 0  
    init {  
        this.name = name  
        this.age = age  
    }  
}
```

主构造函数参数的var和val

- 参数没有var和val修饰,参数在其他地方不能使用

```
class Person8(name:String,age:Int){  
}
```

- 参数有var修饰,可以使用,可以修改

```
class Person(var name:String,var age:Int){  
    override fun toString(): String {  
        return "Person(name=$name age=$age)"  
    }  
}
```

```
val person = Person8("张三",20)  
person.name = "李四"
```

- 参数有val修饰,可以使用,不能修改

```
class Person(val name:String, val age:Int){  
    override fun toString(): String {  
        return "Person(name=$name age=$age)"  
    }  
}
```


次构造函数

- 次构造函数(次构造函数必须要调用主构)

```
class Person(name:String,age:Int){  
    constructor(name: String,age: Int,phone:String):this(name, age)  
}
```

次构造函数间的调用

- 次构造函数间的调用

```
class Person12(name:String,age:Int){  
    constructor(name: String,age: Int,phone:String):this(name, age)  
    constructor(name: String,age: Int,phone:  
String,email:String):this(name, age, phone)  
}
```

次构造函数参数使用

- 次构中参数不能加**var**或者**val**修饰
- 次构参数使用

```
class Person11(var name: String, var age: Int) {  
    var phone: String = ""  
    constructor(name:String,age:Int,phone:String):this(name,age){  
        this.phone = phone  
    }  
}
```

主构 次构 init调用顺序



```
class Person(name:String,age:Int){  
    init {  
        println("执行了init方法")  
    }  
    constructor(name: String,age: Int,phone:String):this(name, age){  
        println("执行了次构造函数")  
    }  
}
```

- 无论调用主构还是次构都会执行init
- 调用次构先执行init再执行次构中的操作

面向对象三大特征

- 封装
- 继承
- 多态



 **注意**
内有高压 非专业人士
请勿打开 

音频输入

右 左

右 左

音频输出



220V-

开

关


JDR128691


A001526

microlo

封装

- 隐藏内部实现的细节,只保留功能接口
- 隐藏内部实现的细节,只保留功能接口
- 隐藏内部实现的细节,只保留功能接口

面向对象实战_洗衣服





洗衣机的出现,大大解放了生产力



社会分工

- 洗衣机生产商(程序员A)

- 电机的工作原理

- 排水的工作原理

- 马达的工作原理

- 电压转换方式

- 保险

- ...

- 消费者(程序员B)

- 买个洗衣机

- 设置模式,开始洗

- ...

封装

```
private String name;  
public String getName() {  
    return name;  
}  
public void setName(String name) {  
    this.name = name;  
}
```

封装

```
fun sendRequest(path:String, method:String) {  
    val url = URL(path)  
    val conn: HttpURLConnection = url.openConnection() as HttpURLConnection  
    val input: InputStream! = conn.InputStream  
}
```


继承

- 指按照法律或遵照遗嘱接受死者的财产、职务、头衔、地位等。
- 虎父无犬子

继承

虎父无犬子

- A father like a tiger will not have a son like a dog
- **A wise goose never lays a tame egg**

继承

- 继承是指一个对象直接使用另一对象的属性和方法。

类的继承

- 普通继承open关键字

```
open class Father{  
    var name:String = ""  
    var age:Int = 0  
    open fun sayHello(){  
        println("父类hello")  
    }  
}  
class Son:Father(){  
  
}
```

方法继承和属性继承

- 方法继承和属性继承也都是通过open关键字

```
open class Father2{  
    open var name = "小头爸爸"  
    var age = 30  
    open fun sayHello(){  
        println("hello")  
    }  
}  
class Son2: Father2(){  
    override var name = "大头儿子"  
    override fun sayHello() {  
        super.sayHello()  
    }  
}
```

继承父类主构

- 子类主构继承父类主构

```
open class Human(val name:String,var age:Int)  
class Man(name:String,age:Int):Human(name,age)
```

- 子类次构继承父类主构

```
open class Human(var name:String, var age:Int)
```

```
class Woman: Human {  
    constructor(name:String,age:Int):super(name,age)  
    constructor(name:String,age:Int,phone:String):super(name,age)  
  
    constructor(name:String,age:Int,phone:String,email:String):this(name,  
    age,phone)  
}
```

抽象类和接口

- 抽象类和具体类

抽象类

Human



ZhHuman



UsHuman

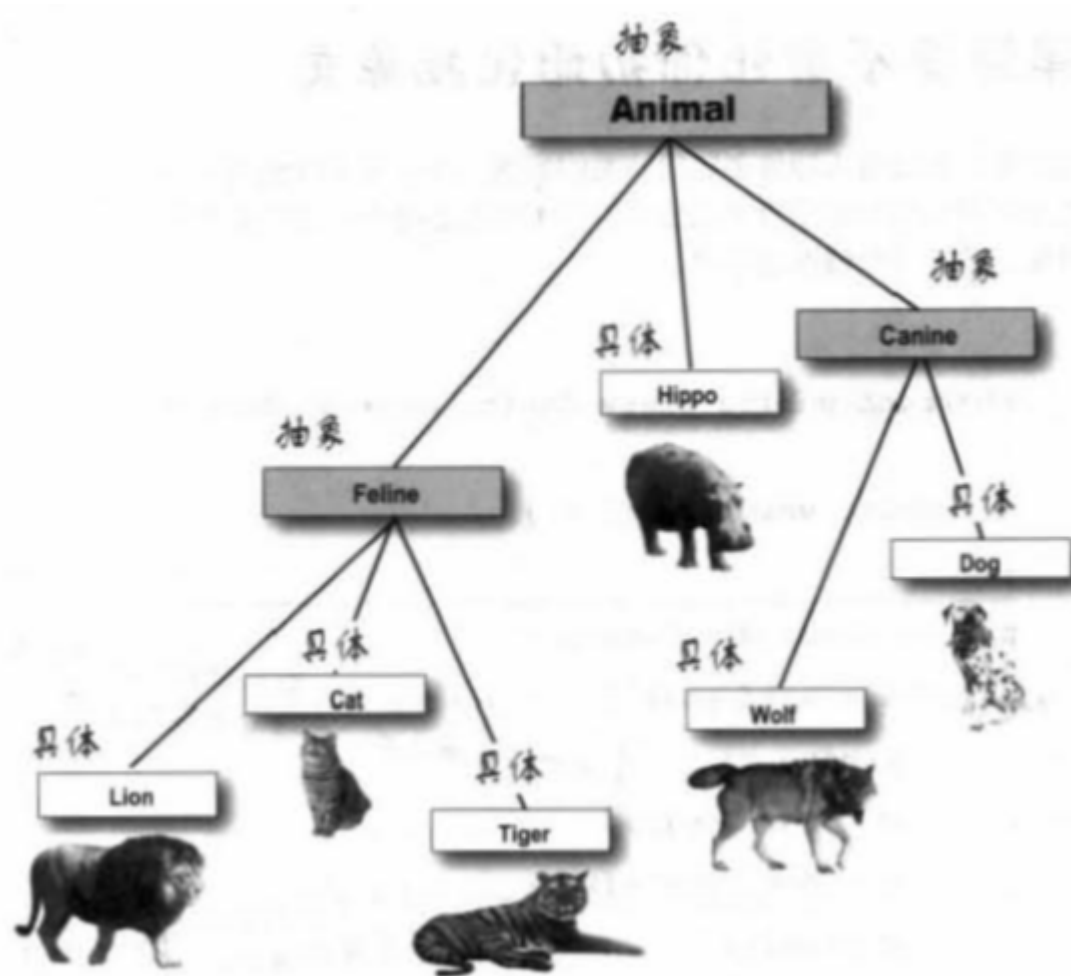


AfricaHuman

各种人之间的不同

- 静态属性不同:肤色 语言(中文 英文 Portuguese)
- 动态行为:吃饭

抽象类



- 抽象类反映的是事物的本质,只能单继承
- 抽象类也可以继承抽象类

接口

- 抽象类反映的是事物的本质,接口反映的是事物的能力



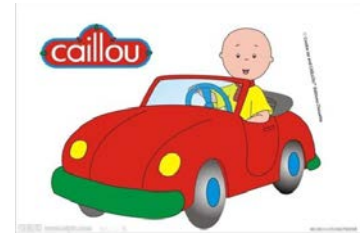
xiaoming

本质: ZhHuman



RideBike

能力



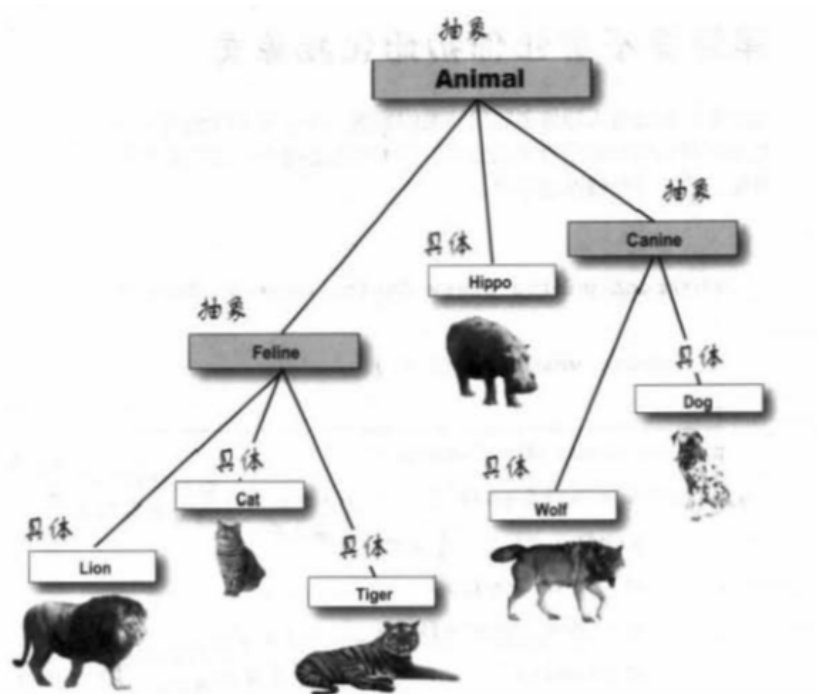
DriveCar

能力

抽象类和接口

- 抽象类反映事物的本质,接口代表能力
- 类只能单继承,接口可以多实现

抽象类练习



把这张图用抽象类表示出来

接口补充

```
//开车的能力
interface DriveCar1{
    //驾照
    var liscence:String
    //行为
    fun drive(){
        println("挂挡 踩油门 走")
    }
}
```


多态

- 同种功能, 不同表现形态



ZhHuman



UsHuman



AfricaHuman



多态

- 多态就是同种功能不同的表现形式
- 通过父类接收子类类型,调用的还是子类的方法

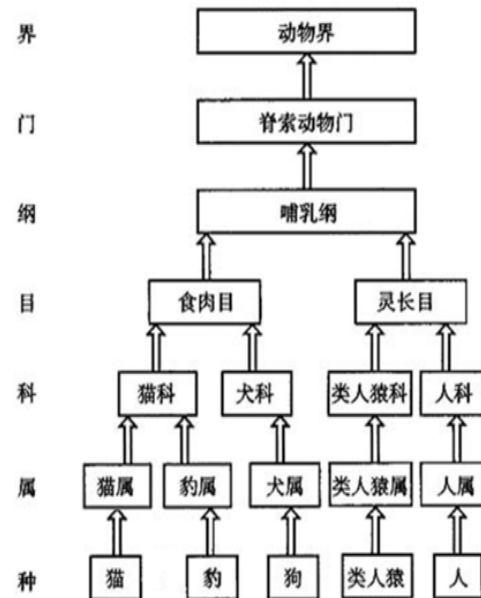
智能类型转换

生物多样性

遗传多样性



物种多样性



ShepHerdDog



放羊

Rural Dog



看家



智能类型转换

- 当判断出类型之后就自动将类型转换成该类型了

```
open class IAnimal
class IDog : IAnimal() {
    fun wangwang() {
        println("狗汪汪叫")
    }
}
```

```
val animal1: IAnimal = IDog()
if (animal1 is IDog)
    animal1.wangwang()
```


嵌套类和内部类

```
class OuterClass {  
    private String name = "张三";  
    class InClass{  
        public void sayHello() {  
            System.out.println("hello "+name);  
        }  
    }  
}
```

嵌套类和内部类

- 嵌套类

```
class OutClass{  
    var outName = "张三"  
    class InClass{  
        fun sayHello(){  
            //println(outName) 无法访问外部类中字段  
        }  
    }  
}
```

- 内部类inner

```
class OutClass1{  
    var outName = "张三"  
    inner class InClass{  
        fun sayHello(){  
            println(outName) //可以访问外部类中字段  
        }  
    }  
}
```

Kotlin的class

- 定义class默认都是final
- 类里面定义类都是嵌套类(static)
- 加上inner之后变成内部类

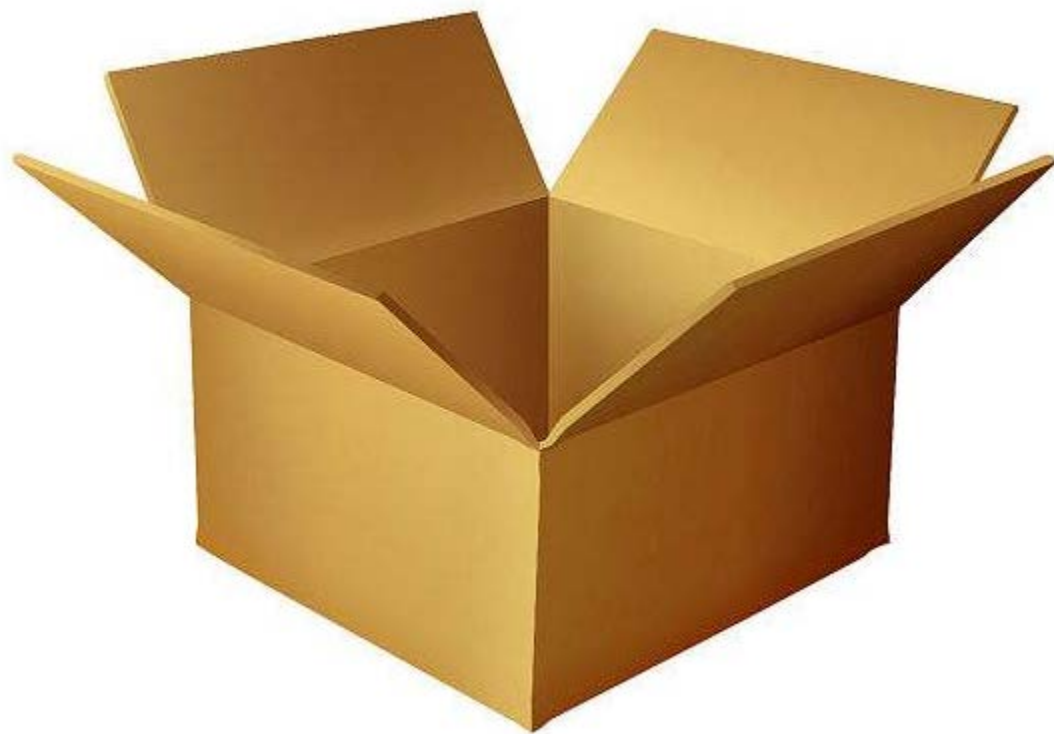
内部类中使用this

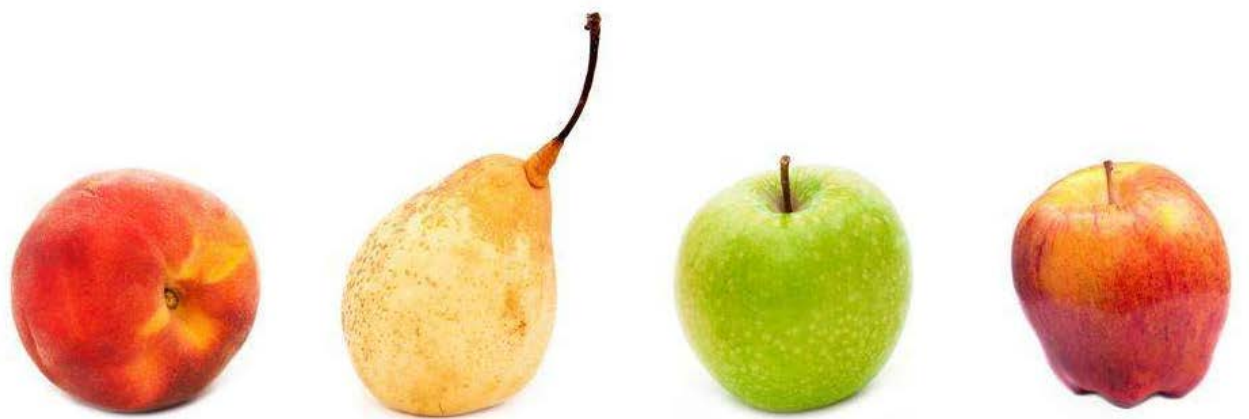
- 可以指定内部this和外部this

```
class OutClass2 {  
    var name = "张三"  
    inner class InClass {  
        var name = "李四"  
        fun sayHello() {  
            println(this@OutClass2.name)  
        }  
    }  
}
```

泛型

- 在强类型程序设计语言中编写代码时定义一些可变部分







泛型类

- 使用泛型类需要指定泛型具体类型
- 继承泛型类可以指定泛型类型
- 继承泛型类不知道具体类型可以使用泛型传递

泛型函数

- 泛型类
- 泛型函数

泛型

- 泛型类

```
class Box<T>(var value: T)//不知道具体传递的类型
```

- 泛型函数

```
fun <T> printInfo(content: T) {  
    when (content) {  
        is Int -> println("传入的$content,是一个Int类型")  
        is String -> println("传入的$content,是一个String类型")  
        else -> println("传入的$content,不是Int也不是String")  
    }  
}
```

泛型上限

```
fun main(args: Array<String>) {  
    val box1 = FruitBox<Fruit>()  
    val box2 = FruitBox<Pear>()  
}
```

//泛型上限

```
class FruitBox<T:Fruit>
```

```
abstract class Thing
```

```
abstract class Fruit:Thing()
```

```
class Pear:Fruit()
```

类型投射(in out)

- List<out Fruit>: 相当于java的List<? Extends Fruit>
- List<in Fruit>: 相当于java的List<? super Fruit>

泛型上限和类型投射(in out)

- 泛型上限是在定义泛型时指定的
- 类型投射是在使用泛型的时候指定的

星号投射

- 相当于java的?

```
/**  
 * 集合泛型里面可以写任意类型  
 */  
fun setList(list:List<*>) {  
  
}
```