Kotlin分享(类，对象，接口)

接口

```
//声明
interface Clickable{
    fun click()
}

//实现接口
class Button : Clickable{
    //override关键词必须有
    override fun click() = println("Click me")
}
```

接口可以有带默认实现的方法，但是，一个类如果实现了两个接口，并且这两个接口都有默认实现，则这个类必须提供自己的实现

书上说会有编译错误，其实IDE就会提示你

如果你想调用接口中实现的默认方法，则按照如下调用

```
 override fun showOff() {
    super<Focusable>.showOff()
}
```

各种修饰符

控制继承的

java默认是允许创建类的子类，并重写任何方法

kotlin中，类默认是final的，不管是类，还是方法，如果子类要继承或重写，则需要加关键字open

需要注意的是： 如果你继承了父类，或者重写了父类的方法，那么当前这个类，或者方法默认是open的。如果不想让子类重写，需要加上final

这种方式带来的好处是，大部分类都可以不加思考的使用智能转换（智能转换，只能在类型检查后没有改变过的变量上起作用）

抽象类

```
abstract class Animated {
    //没有方法体的方法，就是抽象方法，abstract可以省略
    abstract fun animate()

     //默认不是open，需要可以加
    open fun stop(){

    }
    fun start(){

    }

}
```

控制可见性的

kotlin中默认是public

没有java中的包私有

增加了internal，控制只在模块内部可见，模块的概念（一个IDEA，一个eclipse，一个maven，一个gradle）

内部类，嵌套类

kotlin中内部类，默认是不持有外部类的引用的，如果需要持有的话，需要加上inner关键词 如下：

```
class Outer{
    inner class Inner{
        fun getOuter(): Outer = this@Outer
    }
}
```

密封类(sealed)

```
sealed class ExprBase{
    class Num(val value: Int):ExprBase()

    class Sum(val left:ExprBase,val right: ExprBase):ExprBase()
}


private fun eval(e :ExprBase):Int =

        when(e){
            is ExprBase.Num -> e.value
            is ExprBase.Sum -> eval(e.left)+ eval(e.right)
        }
```

kotlin1.0中必须直接嵌套在父类中，1.1取消了这个限制

构造方法

kotlin分为主构造方法和从构造方法

```
class User(val nickname :String) //这里就是主构造方法

//写法最完整的
class User constructor(_name :String){
    val nickname :String

    //init语句块
    init{
        nickname = _name
    }
}

//nickname是属性，直接设置属性
//主构造方法中没有注解或可见效修饰符(public,private...)，可以省略constructor关键字
class User (_name :String){
    val nickname  = _name;
}

//最后如果属性用的是构造方法中的参数初始化，可以最终简化为

class User(val nickname :String)


同样可以设置参数中的默认值
class User(val nickname :String,val ismale :Boolean = false)


最后是创建对象，不需要new关键词

val gaoteng = User("gaoteng")

//初始化父类

open class User (val _name :String,val ismale :Boolean)


class SohuUser(name:String):User(name,false)

//如果不提供任何构造方法，则必须调用父类的构造方法

class RadioButton :Button()


将构造方法设置为private如下
class User private constructor(){

}
```

从构造方法

如果没有主构造方法，从构造方法里面，至少的有一个初始化基类的构造方法

```
class MyView:View{
    constructor(ctx: Context):super(ctx){
        //
    }

    constructor(ctx:Context,arrt: Attributes):super(ctx,arrt){
        //
    }
}
```

属性

接口中可以包含抽象属性声明。

```
interface Cat{
    val nickname:String
}
```

接口中是不包含属性的存储的，实现这个接口的类，可以存储这个属性，或者定义getter方法获取他

如下例子

```

class RedCat(override val nickname: String):Cat


class BlackCat(val name: String):Cat{

    override val nickname:String
        get() = name+"black"
}

class WhiteCat(val name:String):Cat{
    override  val nickname = name+"white"
}
```

接口还可以包含具有getter和setter的属性

```
interface Cat{
    val nickname:String
    val firstname:String
        get() = nickname.getfirstname()
}
```

getter与setter访问支持字段

有个需求，我们要在每一次属性修改的时候，输出日志，该怎么做。我们需要用自定义的setter，然后在里面输出log

```
    class Book(){
    var author:String  = "gaoteng"
        set(value:String) {
            println("Book author change from$field to$value")
            field = value
        }
}

val book = Book()
```

同时可以将属性的get set方法设置为private，这样外部就无法修改类的属性

```
class Book(){
    var author:String  = "gaoteng"
       private set
}
```

编译器生成的方法：数据类和类委托

数据类

可以帮助我们在后台自动生成equals，hashcode，toString方法，不需要我们写在代码中

```
data class Client(val name:String,val num:Int)
```

注意没有在主构造方法中声明的属性，不会加入到想等下比较(equals)和hashcode的计算中

小点：kotlin中 == 和java中的equals一样，要判断引用是否相同，则用===

类委托

在实现装饰者模式时候，需要将一个类对象的行为，包装在另一个对象中。 如果手写的话，需要将所有方法

都重写一遍，kotlin中提供了委托支持

object关键词

这个object关键词定义一个类同时创建一个实例。 三种场景： 1.对象声明：单例的一种方式

```
    object UserInfoManger {

    var uid: Int = 0
    var name: String = ""
    var phone: String = ""

    fun getUserInfo():String{
        return "uid:"+uid+"   name:"+name+ "   phone:"+phone;
    }

    }
```

2.伴生对象： 因为kotlin中没有static关键词，所以要实现java的static访问成员的方法就得用到伴生对象

```
class A{
    companion object{
        fun bar(){
            //print some
        }
    }

    companion object Name{
        fun bar1(){
            //print some
        }
    }
}
这样调用
A.bar()
A.Name.bar1()
```

伴生对象也可以有名字

也可以实现接口

也可以扩展函数如下 ``` class Person(val firstname:String){ companion object{ } }

fun Person.Companion.create(json:String):Person{ // }

3.对象表达式：替代java匿名内部类

除了去掉对象的名字，语法和对象声明相同。

view.setOnclickListenner(object:OnClickListenner(){ override fun onClick(){

```
    }
}
```

) ```

也可以访问局部变量，不过不需要加final