CODE SPITZ 82

KOTLIN ELEMENTARY

1 2 3 4 [✓ 5] 6

infix function

infix method

```kotlin
class ListA{
    val list = mutableListOf<String>()
    operator fun get(i:Int) = list[i]
    operator fun plus(b:String) = run {
        list += b
        this
    }

}
```

```kotlin
val list = ListA() + "abc"
println( list[0] )
```

infix method

```kotlin
class ListA{
    val list = mutableListOf<String>()
    operator fun get(i:Int) = list[i]
    operator fun plus(b:String) = run {
        list += b
        this
    }
    infix fun add(b:String) = plus(b)
}
```

```kotlin
val list = ListA() + "abc"
println( list[0] )
```

infix method

```kotlin
class ListA{
    val list = mutableListOf<String>()
    operator fun get(i:Int) = list[i]
    operator fun plus(b:String) = run {
        list += b
        this
    }
    infix fun add(b:String) = plus(b)
}
```

```kotlin
val list = ListA() + "abc"
println( list[0] )
         (list add "def")[1]
```

infix method

```
class ListA{
    val list = mutableListOf<String>()
    operator fun get(i:Int) = list[i]
    operator fun plus(b:String) = run {
        list += b
        this
    }
    infix fun add(b:String) = plus(b)
}
```

```
val list = ListA() + "abc"
println( list[0] )
println( (list add "def")[1] )
```

infix function

```
infix fun <T> T.combine(v:T) = mutableListOf(this, v)
```

infix function

```
infix fun <T> T.combine(v:T) = mutableListOf(this, v)
```

```
val list = 10 combine 20
println("${JSON.stringify(list)}")
```

infix function

```
infix fun <T> T.combine(v:T) = mutableListOf(this, v)
infix fun <T> MutableList<T>.combine(v:T) = run{
    this.add(v)
    this
}
```

```
val list = 10 combine 20
println("${JSON.stringify(list)}")
```

infix function

```
infix fun <T> T.combine(v:T) = mutableListOf(this, v)
infix fun <T> MutableList<T>.combine(v:T) = run{
    this.add(v)
    this
}
```

```
val list = 10 combine 20 combine 30 combine 40
println("${JSON.stringify(list)}")
```

Pair

```
public infix fun <A, B> A.to(that: B): Pair<A, B> = Pair(this, that)
```

```
val map = mapOf("a" to 1, "b" to 2)
```

mutable delegation

ReadWriteProperty

```
public interface ReadWriteProperty<in R, T> {
    public operator fun getValue(thisRef: R, property: KProperty<*>): T
    public operator fun setValue(thisRef: R, property: KProperty<*>, value: T)
}
```

NotNull

```
private class NotNullVar<T : Any>() : ReadWriteProperty<Any?, T> {
    private var value: T? = null
    public override fun getValue(thisRef: Any?, property: KProperty<*>): T {
        return value ?: throw IllegalStateException("Property should be initialized before get.")
    }
    public override fun setValue(thisRef: Any?, property: KProperty<*>, value: T) {
        this.value = value
    }
}
```

NotNull

```kotlin
private class NotNullVar<T : Any>() : ReadWriteProperty<Any?, T> {
    private var value: T? = null
    public override fun getValue(thisRef: Any?, property: KProperty<*>): T {
        return value ?: throw IllegalStateException("Property should be initialized before get.")
    }
    public override fun setValue(thisRef: Any?, property: KProperty<*>, value: T) {
        this.value = value
    }
}
```

```kotlin
class Notnull{
    var a:String by Delegates.notNull()
    fun action(v:String){
        a = v
        println(a)
    }
}
```

lateinit var

```kotlin
private class NotNullVar<T : Any>() : ReadWriteProperty<Any?, T> {
    private var value: T? = null
    public override fun getValue(thisRef: Any?, property: KProperty<*>): T {
        return value ?: throw IllegalStateException("Property should be initialized before get.")
    }
    public override fun setValue(thisRef: Any?, property: KProperty<*>, value: T) {
        this.value = value
    }
}
```

```kotlin
class Notnull{
    lateinit var a:String
    fun action(v:String){
        a = v
        println(a)
    }
}
```

lateinit var

```kotlin
private class NotNullVar<T : Any>() : ReadWriteProperty<Any?, T> {
    private var value: T? = null
    public override fun getValue(thisRef: Any?, property: KProperty<*>): T {
        return value ?: throw IllegalStateException("Property should be initialized before get.")
    }
    public override fun setValue(thisRef: Any?, property: KProperty<*>, value: T) {
        this.value = value
    }
}
```

```kotlin
class Notnull{
    lateinit var a:String
    fun action(v:String){
        a = v
        println(a)
    }
}
```

```kotlin
class Notnull{
    var a:String by Delegates.notNull()
    fun action(v:String){
        a = v
        println(a)
    }
}
```

decorator

```kotlin
class Dele(val deco:String):ReadWriteProperty<Any?, String> {
    private var value: String? = null
    override fun getValue(thisRef: Any?, property: KProperty<*>):String {
        return "$deco$value"
    }
    override fun setValue(thisRef: Any?, property: KProperty<*>, value:String) {
        this.value = value
    }
}
```

decorator

```kotlin
class Dele(val deco:String):ReadWriteProperty<Any?, String> {
    private var value: String? = null
    override fun getValue(thisRef: Any?, property: KProperty<*>):String {
        return "$deco$value"
    }
    override fun setValue(thisRef: Any?, property: KProperty<*>, value:String) {
        this.value = value
    }
}
class CustomDele(deco:String){
    var a by Dele(deco)
    fun action(v:String){
        a = v
        println(a)
    }
}
```

decorator

```kotlin
class Dele(val deco:String):ReadWriteProperty<Any?, String> {
    private var value: String? = null
    override fun getValue(thisRef: Any?, property: KProperty<*>):String {
        return "$deco$value"
    }
    override fun setValue(thisRef: Any?, property: KProperty<*>, value:String) {
        this.value = value
    }
}
class CustomDele(deco:String){
    var a by Dele(deco)
    fun action(v:String){
        a = v
        println(a)
    }
}
```

```kotlin
val cd = CustomDele("^^;; ")
cd.action("abc")
```

immutable delegation

lazy

```
public interface Lazy<out T> {
    public val value: T
    public fun isInitialized(): Boolean
}
```

lazy

```
public interface Lazy<out T> {
    public val value: T
    public fun isInitialized(): Boolean
}
```

```
class Immun(override val value:String):Lazy<String> {
    override fun isInitialized() = true
}
```

lazy

```
public interface Lazy<out T> {
    public val value: T
    public fun isInitialized(): Boolean
}
```

```
class Immun(override val value:String):Lazy<String> {
    override fun isInitialized() = true
}
```

```
class CustomImmun{
    val a by Immun("abc")
    fun action(v:String){
        println(a)
    }
}
```

lazy

```
class Keys<T>(map:Map<T, Any>):Lazy<Set<T>> {
    override val value = map.keys
    override fun isInitialized() = true
}
```

lazy

```
class Keys<T>(map:Map<T, Any>):Lazy<Set<T>> {
    override val value = map.keys
    override fun isInitialized() = true
}

class CustomKeys(val map:Map<String, Any>){
    val keys by Keys(map)
    fun action(){
        println("${JSON.stringify(keys)}")
    }
}
```

map delegation

map delegation

```
class MapDele(var dele:MutableMap<String, Any?>){
    val a:String by dele
    val b:Int by dele
}
```

map delegation

```kotlin
class MapDele(var dele:MutableMap<String, Any?>){
    val a:String by dele
    val b:Int by dele
}
```

```kotlin
val md = MapDele(mutableMapOf("a" to "abc", "b" to 3))
println("md - ${md.a}")
println("md - ${md.b}")
```

map delegation

```kotlin
class MapDele(var dele:MutableMap<String, Any?>){
    val a:String by dele
    val b:Int by dele
}
```

```kotlin
val md = MapDele(mutableMapOf("a" to "abc", "b" to 3))
println("md - ${md.a}")
println("md - ${md.b}")
```

```kotlin
md.dele = mutableMapOf("a" to "def", "b" to 5)
println("md - ${md.a}")
println("md - ${md.b}")
```

class delegation

has a

```
interface Mobile{
    fun move():String
    fun stop():String
}
```

has a

```
interface Mobile{
    fun move():String
    fun stop():String
}
```

```
class Car(val name:String):Mobile{
    override fun move()= "$name 이동"
    override fun stop() = "$name 정지"
}
```

has a

```
interface Mobile{
    fun move():String
    fun stop():String
}
```

```
class Car(val name:String):Mobile{
    override fun move()= "$name 이동"
    override fun stop() = "$name 정지"
}
```

```
class SportsCar(val car:Mobile):Mobile{
    override fun stop() = car.stop()
    override fun move()  = car.move()
    fun highSpeed() = "고속이동"
}
```

has a

```
interface Mobile{
    fun move():String
    fun stop():String
}
```

```
class Car(val name:String):Mobile{
    override fun move()= "$name 이동"
    override fun stop() = "$name 정지"
}
```

```
class SportsCar(val car:Mobile):Mobile{
    override fun stop() = car.stop()
    override fun move()  = car.move()
    fun highSpeed() = "고속이동"
}
```

```
val scar = SportsCar(Car("페라리"))
println("scar - ${scar.move()}")
println("scar - ${scar.stop()}")
println("scar - ${scar.highSpeed()}")
```

by

```
interface Mobile{
    fun move():String
    fun stop():String
}
```

```
class Car(val name:String):Mobile{
    override fun move()= "$name 이동"
    override fun stop() = "$name 정지"
}
```

```
class SportsCar(val car:Mobile):Mobile{
    override fun stop() = car.stop()
    override fun move()  = car.move()
    fun highSpeed() = "고속이동"
}
```

```
class FastCar(car:Mobile):Mobile by car{
    fun fastSpeed() = "빠른 이동"
}
```

```
val scar = SportsCar(Car("페라리"))
println("scar - ${scar.move()}")
println("scar - ${scar.stop()}")
println("scar - ${scar.highSpeed()}")
```

by

```kotlin
interface Mobile{
    fun move():String
    fun stop():String
}
```

```kotlin
class Car(val name:String):Mobile{
    override fun move()= "$name 이동"
    override fun stop() = "$name 정지"
}
```

```kotlin
class SportsCar(val car:Mobile):Mobile{
    override fun stop() = car.stop()
    override fun move()  = car.move()
    fun highSpeed() = "고속이동"
}
```

```kotlin
class FastCar(car:Mobile):Mobile by car{
    fun fastSpeed() = "빠른 이동"
}
```

```kotlin
val scar = SportsCar(Car("페라리"))
println("scar - ${scar.move()}")
println("scar - ${scar.stop()}")
println("scar - ${scar.highSpeed()}")
```

```kotlin
val fcar = FastCar(Car("BMW"))
println("fcar - ${fcar.move()}")
println("fcar - ${fcar.stop()}")
println("fcar - ${fcar.fastSpeed()}")
```

by strategy

```
interface Mobile{
    fun move():String
    fun stop():String
}
```

```
class Car(val name:String):Mobile{
    override fun move()= "$name 이동"
    override fun stop() = "$name 정지"
}
```

```
class FastCar(car:Mobile):Mobile by car{
    fun fastSpeed() = "빠른 이동"
}
```

```
val fcar = FastCar(Car("BMW"))
println("fcar - ${fcar.move()}")
println("fcar - ${fcar.stop()}")
println("fcar - ${fcar.fastSpeed()}")
```

by strategy

```kotlin
interface Mobile{
    fun move():String
    fun stop():String
}
class UltraCar(var car:Mobile):Mobile by
car{
    fun UltraSpeed() = "초빠른 이동"
}
```

```kotlin
class Car(val name:String):Mobile{
    override fun move()= "$name 이동"
    override fun stop() = "$name 정지"
}
```

```kotlin
class FastCar(car:Mobile):Mobile by car{
    fun fastSpeed() = "빠른 이동"
}
```

```kotlin
val fcar = FastCar(Car("BMW"))
println("fcar - ${fcar.move()}")
println("fcar - ${fcar.stop()}")
println("fcar - ${fcar.fastSpeed()}")
```

by strategy

```kotlin
interface Mobile{
    fun move():String
    fun stop():String
}
class UltraCar(var car:Mobile):Mobile by
car{
    fun UltraSpeed() = "초빠른 이동"
}
val ucar = UltraCar(Car("택시"))
println("ucar - ${ucar.move()}")
println("ucar - ${ucar.stop()}")
println("ucar - ${ucar.UltraSpeed()}")
```

```kotlin
class Car(val name:String):Mobile{
    override fun move()= "$name 이동"
    override fun stop() = "$name 정지"
}
```

```kotlin
class FastCar(car:Mobile):Mobile by car{
    fun fastSpeed() = "빠른 이동"
}
```

```kotlin
val fcar = FastCar(Car("BMW"))
println("fcar - ${fcar.move()}")
println("fcar - ${fcar.stop()}")
println("fcar - ${fcar.fastSpeed()}")
```

by strategy

```kotlin
interface Mobile{
    fun move():String
    fun stop():String
}
class UltraCar(var car:Mobile):Mobile by
car{
    fun UltraSpeed() = "초빠른 이동"
}
val ucar = UltraCar(Car("택시"))
println("ucar - ${ucar.move()}")
println("ucar - ${ucar.stop()}")
println("ucar - ${ucar.UltraSpeed()}")
ucar.car = Car("야간버스")
println("ucar - ${ucar.move()}")
```

```kotlin
class Car(val name:String):Mobile{
    override fun move()= "$name 이동"
    override fun stop() = "$name 정지"
}
```

```kotlin
class FastCar(car:Mobile):Mobile by car{
    fun fastSpeed() = "빠른 이동"
}
```

```kotlin
val fcar = FastCar(Car("BMW"))
println("fcar - ${fcar.move()}")
println("fcar - ${fcar.stop()}")
println("fcar - ${fcar.fastSpeed()}")
```

by object

```kotlin
interface Mobile{
    fun move():String
    fun stop():String
}
class UltraCar(var car:Mobile):Mobile by
car{
    fun UltraSpeed() = "초빠른 이동"
}
val ucar = UltraCar(Car("택시"))
println("ucar - ${ucar.move()}")
println("ucar - ${ucar.stop()}")
println("ucar - ${ucar.UltraSpeed()}")
ucar.car = Car("야간버스")
println("ucar - ${ucar.move()}")
```

```kotlin
class Car(val name:String):Mobile{
    override fun move()= "$name 이동"
    override fun stop() = "$name 정지"
}
```

by object

```
interface Mobile{
    fun move():String
    fun stop():String
}
class UltraCar(var car:Mobile):Mobile by
car{
    fun UltraSpeed() = "초빠른 이동"
}
val ucar = UltraCar(Car("택시"))
println("ucar - ${ucar.move()}")
println("ucar - ${ucar.stop()}")
println("ucar - ${ucar.UltraSpeed()}")
ucar.car = Car("야간버스")
println("ucar - ${ucar.move()}")
```

```
class Car(val name:String):Mobile{
    override fun move()= "$name 이동"
    override fun stop() = "$name 정지"
}
```

```
class DogCar:Mobile by object:Mobile{
    val name = "아인"
    override fun move() = "$name 달려"
    override fun stop() = "$name 멈춰"
}{
    fun fastSpeed() = "개빠름"
}
```

```
val dcar = DogCar()
println("dcar - ${dcar.move()}")
println("dcar - ${dcar.stop()}")
println("dcar - ${dcar.fastSpeed()}")
```

by by by

```
interface AA{fun a()}
interface BB{fun b()}
```

by by by

```
interface AA{fun a()}
interface BB{fun b()}
```

```
class AB0:AA by object:AA{
    override fun a() {
        //a
    }
}, BB by object:BB{
    override fun b() {
        //b
    }
}{
    //AB0
}
```

by by by

```
interface AA{fun a()}
interface BB{fun b()}
```

```
class AB0:AA by object:AA{
    override fun a() {
        //a
    }
}, BB by object:BB{
    override fun b() {
        //b
    }
}{
    //AB0
}
```

```
class AB1(v:AA = object:AA{
    override fun a() {
        //a
    }
}):AA by v{
    //AB1
}
```

by by by

```kotlin
interface AA{fun a()}
interface BB{fun b()}
```

```kotlin
class AB0:AA by object:AA{
    override fun a() {
        //a
    }
}, BB by object:BB{
    override fun b() {
        //b
    }
}{
    //AB0
}
```

```kotlin
class AB1(v:AA = object:AA{
    override fun a() {
        //a
    }
}):AA by v{
    //AB1
}
```

```kotlin
class AB2:AA by object:AA by object :AA{
    override fun a() {
        //a
    }
}{
    //obj
}{
    //AB2
}
```