

Kotlin Workshop: Classes, Inheritance and more...

Andrey Breslav
Dmitry Jemerov



Outline

- Classes
 - Having multiple supertypes (Mixins)
 - First-class delegation
- Generics
 - Variance annotations
 - Type projections
- Class objects
 - No static members in classes
 - Generic bounds
- (Pattern matching)

Top and Bottom

- **class** Any {}
 - Has **no** members
 - Every other class inherits from Any
 - Every type is a subtype of Any?
- special type Nothing
 - Has no values
 - Every type is a supertype of Nothing
 - Nothing? is the most precise type of **null**

Intrinsics

- Built-in types
 - Any, Nothing
- Intrinsic types
 - Unit, String, Array<T>
 - Int, Int?, Double, Double?, ...
 - Functions, Tuples
- Intrinsic operations
 - Arithmetic
 - Array access, length, iteration
 - String operations
 - ... (anything you like)

Correspondence to Java

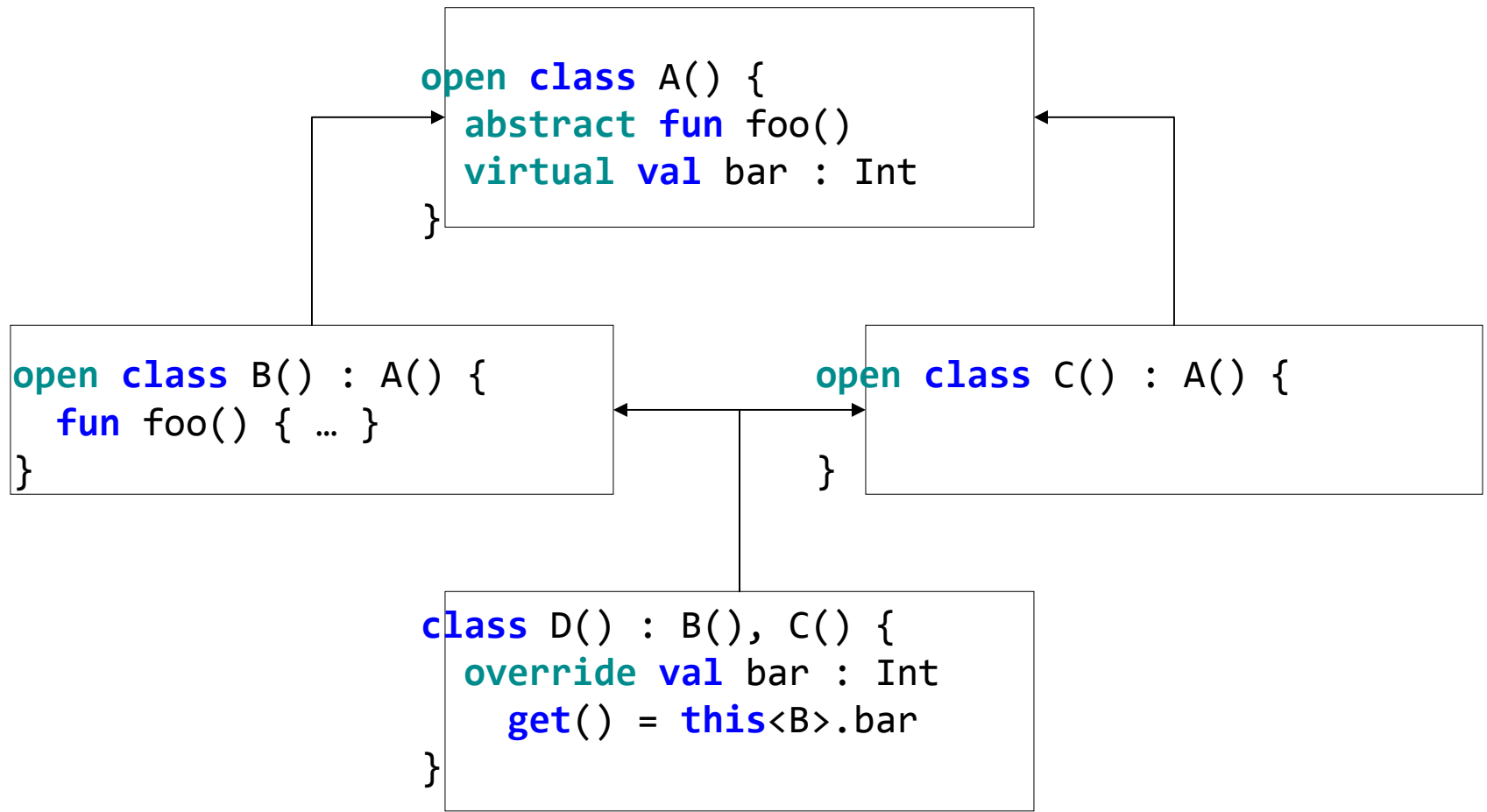
Kotlin	GEN	Java	LOAD	Kotlin
Any		Object		Any?
Unit		void		Unit
Int		int		Int
Int?		Integer		Int?
String		String		String?
Array<Foo>		Foo[]		Array<Foo?>?
Array<Int>		int[]		Array<Int>?
Nothing		-		-
Foo		Foo		Foo?

Classes and Inheritance

```
class Example(b : B) : Base(b) { ... }
```

- Any is the default supertype
- The (primary) constructor initializes supertypes
- Classes are **final** by default
- Members are **non-virtual** by default
- **Overrides** are marked explicitly
- There can be **many** supertypes


Diamonds before us



Mixin implementation (I)


```

open class A() {
    fun foo() = bar + 1
    virtual val bar : Int
}
    
```




```

interface A {
    int foo();
    int bar();
}
    
```




```

class AImpl implements A {
    private final int bar;
    public int foo() {
        return this.bar() + 1;
    }
    public int bar() {
        return this.bar;
    }
}
    
```



```


class ADImpl implements A {
    private final A $this;
    private final int bar;
    public int foo() {
        return $this.bar() + 1;
    }
    public int bar() {
        return this.bar;
    }
}
    
```



Mixin implementation (II)


```

open class A() {
    fun foo() = bar + 1
    virtual val bar : Int
}
    
```



```

class B() : X(), A() {
    override val bar : Int
        get() = 2
}
    
```



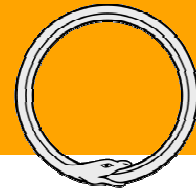
```

class B extends XImpl implements A {
    private final ADImpl $a;

    public B() {
        this.$a = new ADImpl(this);
    }

    public int foo() {
        return $a.foo();
    }

    public int bar() { return 2; }
}
    
```



Generic classes and types

```
class Producer<out T> {  
    fun produce() : T  
}  
class Consumer<in T> {  
    fun consume(t : T)  
}  
class Ouoroboros<T> {  
    fun consume(t : T)  
    fun produce() : T  
}
```

```
Producer<Int> <:  
    Producer<Any>
```

```
Consumer<Any> <:  
    Consumer<Int>
```

```
Ouoroboros<Int> <: <:  
    Ouoroboros<Any>
```

```
Ouoroboros<out Int> <: Ouoroboros<out Any>
```

```
Ouoroboros<in Any> <: Ouoroboros<in Int>
```

Reified generics

- Objects of `C<T>`
 - extra field of type `TypeInfo<C<T>>`
- Internal Java interface `KotlinObject`
 - `TypeInfo<?> getTypeInfo()`
- Generic functions
 - extra arguments of type `TypeInfo<T>`
- **is** and **as** perform a “deep” subtype check
 - **as** performs a `CHECKCAST` and then a further check
 - **is** performs an **instanceof** and then a further check
- `TypeInfo` objects are reused as much as possible

Type erasure and Kotlin

- Java's generic classes remain type-erased in Kotlin
 - **as** performs a CHECKCAST
 - **is** performs an **instanceof**
 - compiler prohibits deep checks like
 - `x is java.util.List<Int>`
- Java's `List` becomes `List<*>`
 - `Foo<*>` is a shorthand for `Foo<out B>`
 - **class** `Foo<T : B>`

Class objects

- Kotlin has no static class members
- What *is* there then?
 - Namespace-level functions/properties
 - Object declarations (singletons)
 - **Class objects**
 - Any class can have one
 - Each class can have only one
 - Class objects can access internals of their classes

Class object example (I)

```
class Example() {  
    class object {  
        fun create() = Example()  
    }  
}
```

```
fun demo() {  
    val e = Example.create()  
}
```

Class object example (II)

```
class Example() {  
    class object : Factory<Example> {  
        override fun create() = Example()  
    }  
}  
  
fun demo() {  
    val factory : Factory<Example> = Example  
    val e = factory.create()  
}  
  
abstract class Factory<T> {  
    fun create() : T  
}
```

Class object example (III)

```
class Lazy<T>()
  where class object T : Factory<T>
{
  private var store : T? = null
  public val value : T
    get() {
      if (store == null) {
        store = T.create()
      }
      return store
    }
}

fun demo() {
  val l = Lazy<Example>()
  val v = l.value
}
```


**And now for something
completely different...**

Pattern matching

When expressions

```
when (x) {  
    1, 2, 3 => ...  
    in 4..10 => ...  
    !in 0..10000 => ...  
    is Tree => ...  
    is Tree @ (val l, *) => ...  
    is Tree @ (null, Tree @ (*, *)) => ...  
    is Tree @ (val l is Tree @ (*, *), *) => ...  
    is TreeValue @ (val v in 1..100, *, *) => ...  
}
```

```
fun Any?.TreeValue() : (Int, Tree?, Tree?)? {  
    if (this !is Tree) return null  
    return (value, right, left)  
}
```

And now for something completely different...

Breaks in custom loops
(a design I would like to discuss)

Labels, Break and Continue

```
@outer for (x in list1) {  
    for (y in list2) {  
        if (...) {  
            // Breaks the inner loop  
            break  
        }  
        if (...) {  
            // Breaks the outer loop  
            break@outer  
        }  
    }  
}
```

Breaks in foreach()

```
@outer list1.foreach { x =>
  list2.foreach { y =>
    if (...) {
      // Breaks the inner loop
      break
    }
    if (...) {
      // Breaks the outer loop
      break@outer
    }
  }
}
```

Breakable foreach()

```
inline fun <T> Iterable<T>.foreach(  
    body : breakable fun(T) : Unit  
) {  
    @@ for (item in this) {  
        // A break from body() breaks the loop  
        body(item)  
    }  
}
```

Resources

- <http://jetbrains.com/kotlin>
- <http://blog.jetbrains.com/kotlin>
- @project_kotlin
- @intellijole
- @abreslav

Kotlin Workshop: Classes, Inheritance and more...

Andrey Breslav
Dmitry Jemerov

