Understand Kotlin's Improved Generic Support

Kevin Jones

@kevinrjones www.rocksolidknowledge.com

Overview

How to declare generics in Kotlin

Reified types and how they help

Variance

Inferring Types

```
val meetings = listOf(Meeting("Board"), Meeting("Finance"))

val meetings = mutableListOf<Meeting>()

val meetings:List<Meeting> = mutableListOf()
```

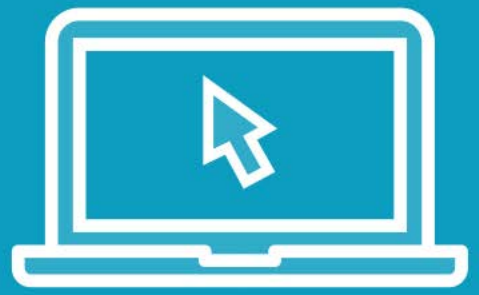
Declaring Generic Functions

```
fun <T> List<T>.itemAt(ndx: Int) : T {}
```

Declaring Generic Classes

```
class Node<T> (private val item:T) {
    fun value() :T {
        return item
    }
}
```

Demo



Generic classes and functions

Type Parameter Constraints

```
class Node<T : Number> (private val item:T) {
    fun value() :T {
        return item
    }
}
```

Generics at Runtime

Java erases all generic type information

Kotlin can reify some generic information

Reify

Make into a thing

Make real

Types are erased

Cannot check generic type at run time

Using 'is' Operator

```
fun printType(items: List<Any>) {
    if(items is List<String>) {
        println("We have strings")
    }
}
```

```
Error:(10, 17) Kotlin: Cannot check for instance of erased
type: List<String>
```

Unchecked Cast Warning

```
fun printType(items: List<Any>) {
    val names = items as List<String>
}
```

```
Warning:(10, 17) Kotlin: Unchecked cast: List<Any> to
List<String>
```

Reifying Types

Generic functions have types erased

However can mark type as reified

Only on inline functions

Inline Reified Types

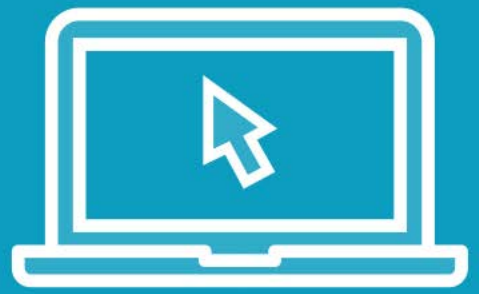
```
fun <T> foo(value: Any) = value is T
```

```
Error:(8, 36) Kotlin: Cannot check for instance of erased
type: T
```

Inline Reified Types

```
inline fun <reified T> foo(value: Any) = value is T
```

Demo

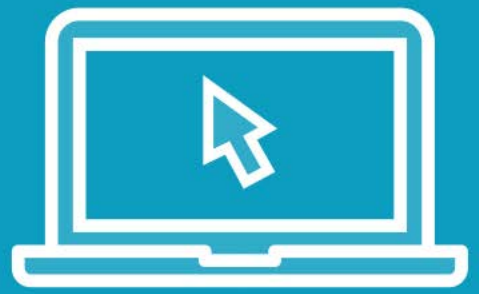


Reified Types

Non-inline Type Parameters

Sometimes need to make a parameter not-inline when the function is inline

Demo



Using noinline

Non-inline Type Parameters

Sometimes need to make a parameter not-inline when the function is inline

Types and SubTypes

Types have a relationship

- e.g. Student is a sub type of Person

Generic types have a more complex relationship

- Is List<Student> a sub type of List<Person>?

Variance determines whether a subtype can be used in place of a type

```
var meetings: MutableList<FinanceMeeting> = mutableListOf()

meetings.add(Meeting())
```

Passing Around Generic Types

Assume FinanceMeeting derives from Meeting

Is the above safe?

Java Variance

Java users 'Super' and 'Extends'

Josh Bloch PECS

- Producer Extends, Consumer Super

Use site variance

Kotlin Variance

'in' and 'out' Keywords

Declaration site variance

- Generally more elegant

Kotlin also supports 'use site' variance

Covariant

If types are Covariant

- Derived type can be used where base type is more specific

In Kotlin mark the type parameter as 'out'

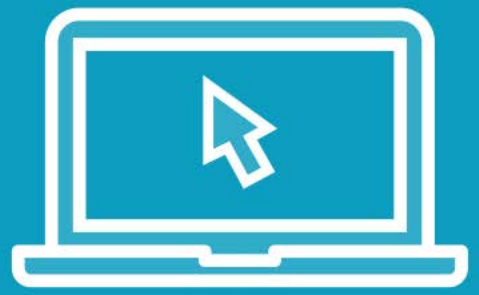
Contravariance

If types are Contravariant

- Base type can be used where derived type is more specific

If Kotlin mark a parameter as 'in'

Demo



Co and Contra Variance

Typical Contravariance Usage

Comparator interfaces are typically contravariant

```
fun <T> copyData(source: MutableList<out T>,
                 destination: MutableList<T>) {
for (item in source) {
        destination.add(item)
    }
}
```

Call Site Variance

Sometimes need more flexibility

Mutable list is now 'projected'

Can only call methods that return data

Summary

Kotlin provides generic functions

and generic classes

Unlike Java generics may be reified

- In certain circumstances

Kotlin supports co and contra variance

Uses declaration site variance

- with in and out keywords