Kotlin Types for Java Developers

Chandra Sekar
@iChanSek

Type System in Java

Type System in Java

- Primitive Types

Type System in Java

- Primitive Types
- Class Types

Primitive Types

Primitive Types - Java

Primitive Types - Java

- long

Primitive Types - Java

- long
- int

Primitive Types - Java

- long
- int
- short

Primitive Types - Java

- long
- int
- short
- byte

Primitive Types - Java

- long
- int
- short
- byte
- char

Primitive Types - Java

- long
- int
- short
- byte
- char
- double

Primitive Types - Java

- long
- int
- short
- byte
- char
- double
- float

Primitive Types - Java

- long
- int
- short
- byte
- char
- double
- float
- boolean

Primitive Types - Kotlin

Primitive Types - Kotlin

- Long

Primitive Types - Kotlin

- Long
- Int

Primitive Types - Kotlin

- Long
- Int
- Short

Primitive Types - Kotlin

- Long
- Int
- Short
- Byte

Primitive Types - Kotlin

- Long
- Int
- Short
- Byte
- Char

Primitive Types - Kotlin

- Long
- Int
- Short
- Byte
- Char
- Double

Primitive Types - Kotlin

- Long
- Int
- Short
- Byte
- Char
- Double
- Float

Primitive Types - Kotlin

- Long
- Int
- Short
- Byte
- Char
- Double
- Float
- Boolean

Primitive Types - Comparison

Primitive Types - Comparison

Java	Kotlin

Primitive Types - Comparison

Java		Kotlin
long	=	Long

Primitive Types - Comparison

Java		Kotlin
long	=	Long
int	=	Int

Primitive Types - Comparison

Java		Kotlin
long	=	Long
int	=	Int
short	=	Short

Primitive Types - Comparison

Java		Kotlin
long	=	Long
int	=	Int
short	=	Short
byte	=	Byte

Primitive Types - Comparison

Java		Kotlin
long	=	Long
int	=	Int
short	=	Short
byte	=	Byte
char	=	Char

Primitive Types - Comparison

Java		Kotlin
long	=	Long
int	=	Int
short	=	Short
byte	=	Byte
char	=	Char
double	=	Double

Primitive Types - Comparison

Java		Kotlin
long	=	Long
int	=	Int
short	=	Short
byte	=	Byte
char	=	Char
double	=	Double
float	=	Float

Primitive Types - Comparison

Java		Kotlin
long	=	Long
int	=	Int
short	=	Short
byte	=	Byte
char	=	Char
double	=	Double
float	=	Float
boolean	=	Boolean

Wrapper Types - Java

Wrapper Types - Java

- Long

Wrapper Types - Java

- Long
- Integer

Wrapper Types - Java

- Long
- Integer
- Short

Wrapper Types - Java

- Long
- Integer
- Short
- Byte

Wrapper Types - Java

- Long
- Integer
- Short
- Byte
- Character

Wrapper Types - Java

- Long
- Integer
- Short
- Byte
- Character
- Double

Wrapper Types - Java

- Long
- Integer
- Short
- Byte
- Character
- Double
- Float

Wrapper Types - Java

- Long
- Integer
- Short
- Byte
- Character
- Double
- Float
- Boolean

Primitive Types - Comparison

Primitive Types - Comparison

Java	Kotlin	Java(Wrappers)

Primitive Types - Comparison

Java		Kotlin		Java(Wrappers)
long	=	Long	~	Long

Primitive Types - Comparison

Java		Kotlin		Java(Wrappers)
long	=	Long	~	Long
int	=	Int	~	Integer

Primitive Types - Comparison

Java		Kotlin		Java(Wrappers)
long	=	Long	~	Long
int	=	Int	~	Integer
short	=	Short	~	Short

Primitive Types - Comparison

Java		Kotlin		Java(Wrappers)
long	=	Long	~	Long
int	=	Int	~	Integer
short	=	Short	~	Short
byte	=	Byte	~	Byte

Primitive Types - Comparison

Java		Kotlin		Java(Wrappers)
long	=	Long	~	Long
int	=	Int	~	Integer
short	=	Short	~	Short
byte	=	Byte	~	Byte
char	=	Char	~	Character

Primitive Types - Comparison

Java		Kotlin		Java(Wrappers)
long	=	Long	~	Long
int	=	Int	~	Integer
short	=	Short	~	Short
byte	=	Byte	~	Byte
char	=	Char	~	Character
double	=	Double	~	Double

Primitive Types - Comparison

Java		Kotlin		Java(Wrappers)
long	=	Long	~	Long
int	=	Int	~	Integer
short	=	Short	~	Short
byte	=	Byte	~	Byte
char	=	Char	~	Character
double	=	Double	~	Double
float	=	Float	~	Float

Primitive Types - Comparison

Java		Kotlin		Java(Wrappers)
long	=	Long	~	Long
int	=	Int	~	Integer
short	=	Short	~	Short
byte	=	Byte	~	Byte
char	=	Char	~	Character
double	=	Double	~	Double
float	=	Float	~	Float
boolean	=	Boolean	~	Boolean

Example - Primitive Type

Example - Primitive Type

```
int x = 10;          // Java - Primitive int
```

Example - Primitive Type

```
int x = 10;                // Java - Primitive int

// is Same as
val x: Int = 10            // Kotlin - Primitive int
```

Example - Primitive Type

```
int x = 10;                // Java - Primitive int

// is Same as
val x: Int = 10            // Kotlin - Primitive int

// Even more simpler way
val x = 10                 // Kotlin - Type inferred as Int
```

Example - Wrapper Type

Example - Wrapper Type

```
Integer x = new Integer(10);        // Java - Wrapper Type
Integer y = 10;                     // Java - Wrapper Type (Auto Boxing)
```

Example - Wrapper Type

```
Integer x = new Integer(10);        // Java - Wrapper Type
Integer y = 10;                     // Java - Wrapper Type (Auto Boxing)

// Lets assume they are same as
val x: Int = 10
val y = 10
```

Example - Wrapper Type

```
Integer x = new Integer(10);        // Java - Wrapper Type
Integer y = 10;                     // Java - Wrapper Type (Auto Boxing)

// Lets assume they are same as
val x: Int = 10
val y = 10

// then, how to represent this in Kotlin?
Integer z = null;
```

Nullable Types

Nullable Types

Nullable Types

- Kotlin has separate type to handle null.

Nullable Types

- Kotlin has separate type to handle null.
- By default all types are non nullable.

Nullable Types

- Kotlin has separate type to handle null.
- By default all types are non nullable.
- If you want to assign null, then you have to declare them as nullable.

Nullable Types

- Kotlin has separate type to handle null.
- By default all types are non nullable.
- If you want to assign null, then you have to declare them as nullable.
- Syntax - postfix corresponding type with ‘?’

Nullable Types

- Kotlin has separate type to handle null.
- By default all types are non nullable.
- If you want to assign null, then you have to declare them as nullable.
- Syntax - postfix corresponding type with ‘?’
- Example - Int?, String?, User?

Back to Previous Example

Back to Previous Example

```
// then, how to represent this in Kotlin?
Integer z = null;
```

Back to Previous Example

```
// then, how to represent this in Kotlin?
Integer z = null;


// And, the answer is
val z: Int? = null
```

Primitive Types - Comparison

Primitive Types - Comparison

Java	Kotlin	Java(Wrappers)

Primitive Types - Comparison

Java		Kotlin				Java(Wrappers)
long	=	Long	~	Long?	=	Long

Primitive Types - Comparison

Java		Kotlin				Java(Wrappers)
long	=	Long	~	Long?	=	Long
int	=	Int	~	Int?	=	Integer

Primitive Types - Comparison

Java		Kotlin				Java(Wrappers)
long	=	Long	~	Long?	=	Long
int	=	Int	~	Int?	=	Integer
short	=	Short	~	Short?	=	Short

Primitive Types - Comparison

Java		Kotlin				Java(Wrappers)
long	=	Long	~	Long?	=	Long
int	=	Int	~	Int?	=	Integer
short	=	Short	~	Short?	=	Short
byte	=	Byte	~	Byte?	=	Byte

Primitive Types - Comparison

Java		Kotlin				Java(Wrappers)
long	=	Long	~	Long?	=	Long
int	=	Int	~	Int?	=	Integer
short	=	Short	~	Short?	=	Short
byte	=	Byte	~	Byte?	=	Byte
char	=	Char	~	Char?	=	Character

Primitive Types - Comparison

Java		Kotlin				Java(Wrappers)
long	=	Long	~	Long?	=	Long
int	=	Int	~	Int?	=	Integer
short	=	Short	~	Short?	=	Short
byte	=	Byte	~	Byte?	=	Byte
char	=	Char	~	Char?	=	Character
double	=	Double	~	Double?	=	Double

Primitive Types - Comparison

Java		Kotlin				Java(Wrappers)
long	=	Long	~	Long?	=	Long
int	=	Int	~	Int?	=	Integer
short	=	Short	~	Short?	=	Short
byte	=	Byte	~	Byte?	=	Byte
char	=	Char	~	Char?	=	Character
double	=	Double	~	Double?	=	Double
float	=	Float	~	Float?	=	Float

Primitive Types - Comparison

Java		Kotlin				Java(Wrappers)
long	=	Long	~	Long?	=	Long
int	=	Int	~	Int?	=	Integer
short	=	Short	~	Short?	=	Short
byte	=	Byte	~	Byte?	=	Byte
char	=	Char	~	Char?	=	Character
double	=	Double	~	Double?	=	Double
float	=	Float	~	Float?	=	Float
boolean	=	Boolean	~	Boolean?	=	Boolean

Take Away - Primitive Types

Take Away - Primitive Types

- Kotlin doesn’t have Primitive types explicitly.

Take Away - Primitive Types

- Kotlin doesn’t have Primitive types explicitly.
- Kotlin compiler chooses Primitive type when the type is mentioned as non nullable.

Take Away - Primitive Types

- Kotlin doesn’t have Primitive types explicitly.
- Kotlin compiler chooses Primitive type when the type is mentioned as non nullable.
- If the type is mentioned as nullable, Kotlin compiler treats it as a wrapper type.

Class Types

Class Types - Java

Class Types - Java

- class String

Class Types - Java

- class String
- interface CharSequence

Class Types - Java

- class String
- interface CharSequence
- enum Color

Class Types - Java

- class String
- interface CharSequence
- enum Color
- class User

Class Types - Java

- class String
- interface CharSequence
- enum Color
- class User

Class Types - Java

- class String
- interface CharSequence
- enum Color
- class User

Class Types - Java

- class String
- interface CharSequence
- enum Color
- class User

Object is common to all.

Supermost Type

Supermost Type

Object (in Java)

Supermost Type

Object (in Java)

is Same as

Supermost Type

Object (in Java)

is Same as

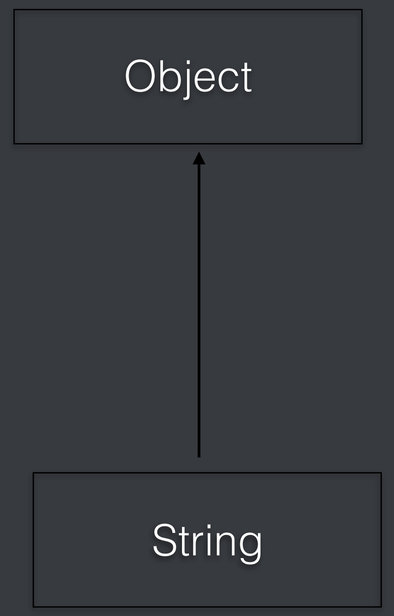
Any (in Kotlin)

Type Hierarchy - Java

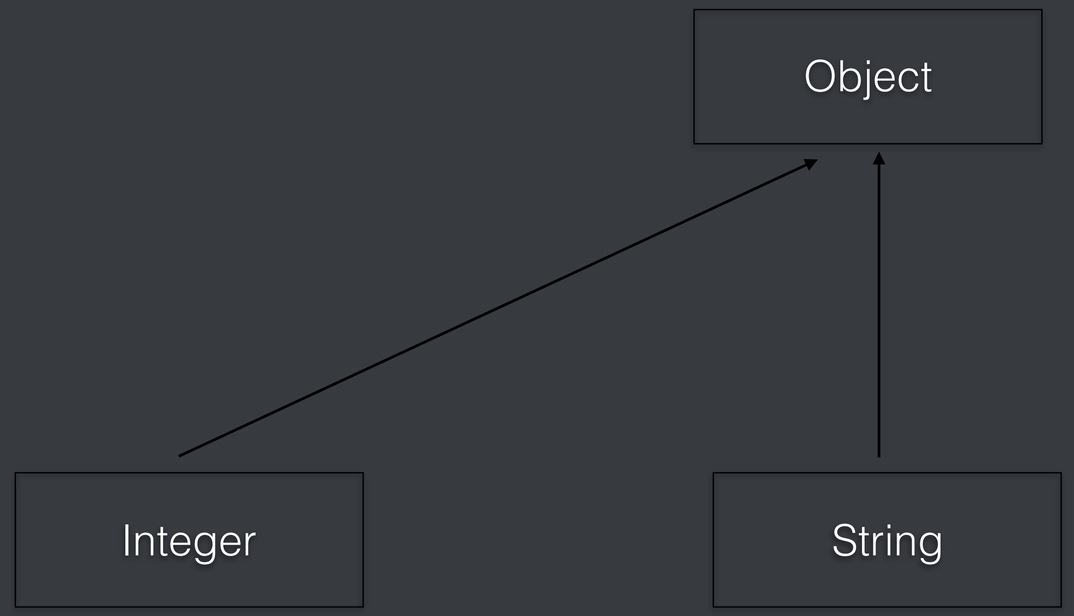
Type Hierarchy - Java

Object

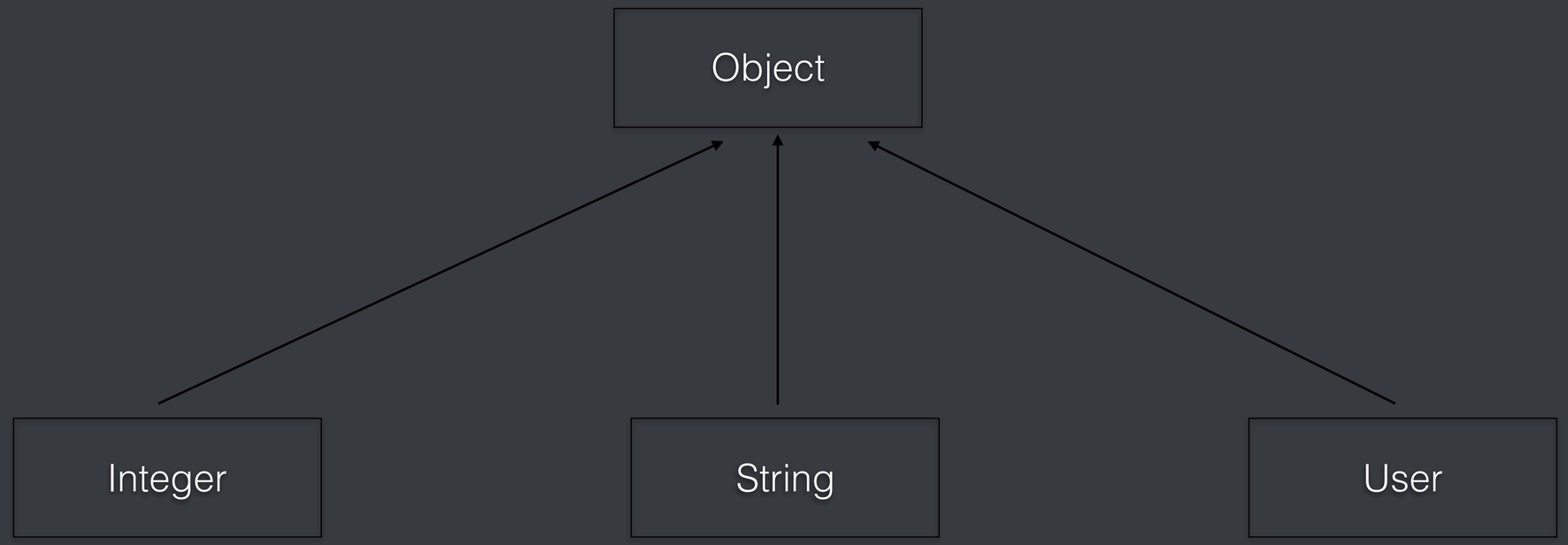
Type Hierarchy - Java



Type Hierarchy - Java



Type Hierarchy - Java

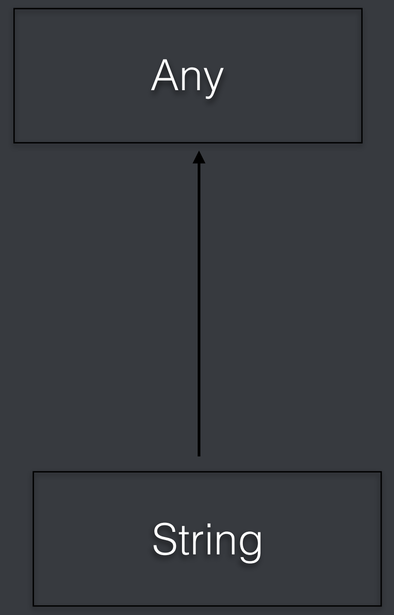


Type Hierarchy - Kotlin

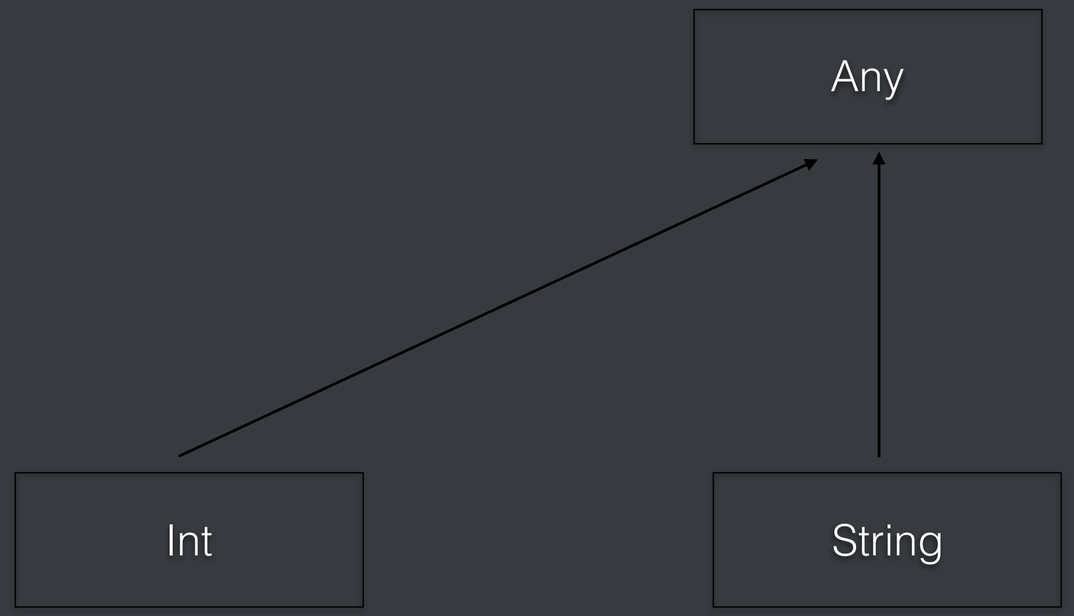
Type Hierarchy - Kotlin

Any

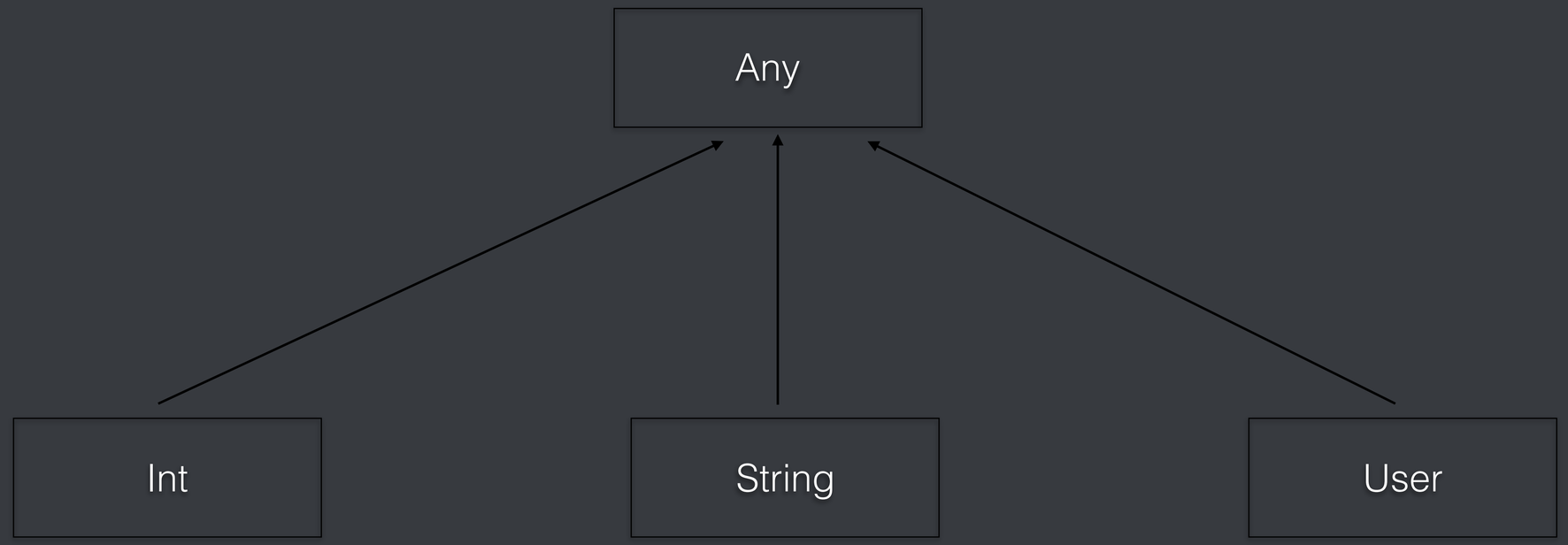
Type Hierarchy - Kotlin



Type Hierarchy - Kotlin



Type Hierarchy - Kotlin



Examples

Examples

```
Object obj = new Object();
Object user = new User();
Object str = "Test";
Object integer = 10;
```

Examples

```
Object obj = new Object();
Object user = new User();
Object str = "Test";
Object integer = 10;
```

```
val any = Any()
val user: Any = User()
val str: Any = "Test"
val int: Any = 10
```

Examples

```
Object obj = new Object();
Object user = new User();
Object str = "Test";
Object integer = 10;
```

```
val any = Any()
val user: Any = User()
val str: Any = "Test"
val int: Any = 10
```

```
User user = new User();
String str = "Test";
Integer i = 10;
```

Examples

```
Object obj = new Object();
Object user = new User();
Object str = "Test";
Object integer = 10;
```

```
val any = Any()
val user: Any = User()
val str: Any = "Test"
val int: Any = 10
```

```
User user = new User();
String str = "Test";
Integer i = 10;
```

```
val user: User = User()
val str: String = "Test"
val i: Int = 10
```

Examples

```
Object obj = new Object();
Object user = new User();
Object str = "Test";
Object integer = 10;
```

```
val any = Any()
val user: Any = User()
val str: Any = "Test"
val int: Any = 10
```

```
User user = new User();
String str = "Test";
Integer i = 10;
```

```
val user: User = User()
val str: String = "Test"
val i: Int = 10
```

```
val user = User()
val str = "Test"
val i = 10
```

Examples

```
Object obj = new Object();
Object user = new User();
Object str = "Test";
Object integer = 10;
```

```
val any = Any()
val user: Any = User()
val str: Any = "Test"
val int: Any = 10
```

```
User user = new User();
String str = "Test";
Integer i = 10;
```

```
val user: User = User()
val str: String = "Test"
val i: Int = 10
```

```
val user = User()
val str = "Test"
val i = 10
```

Type Inferrence

Examples (with null)

Examples (with null)

```
Object obj = null;
User user = null;
String str = null;
Integer i = null;
```

Examples (with null)

```
Object obj = null;
User user = null;
String str = null;
Integer i = null;
```

```
val any: Any = null
val user: User = null
val str: String = null
val i: Int = null
```

Examples (with null)

```
Object obj = null;
User user = null;
String str = null;
Integer i = null;
```

Examples (with null)

```
Object obj = null;
User user = null;
String str = null;
Integer i = null;
```

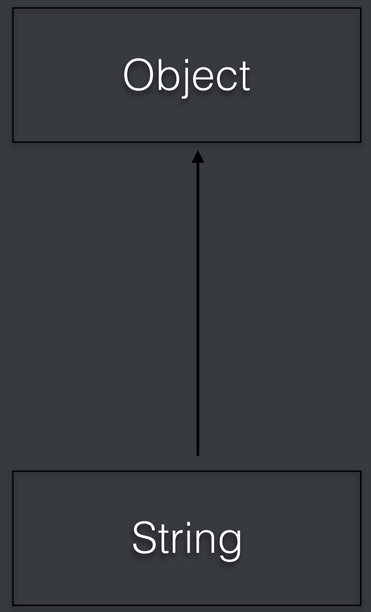
```
val any: Any? = null
val user: User? = null
val str: String? = null
val i: Int? = null
```

Type Hierarchy (Final) - Java

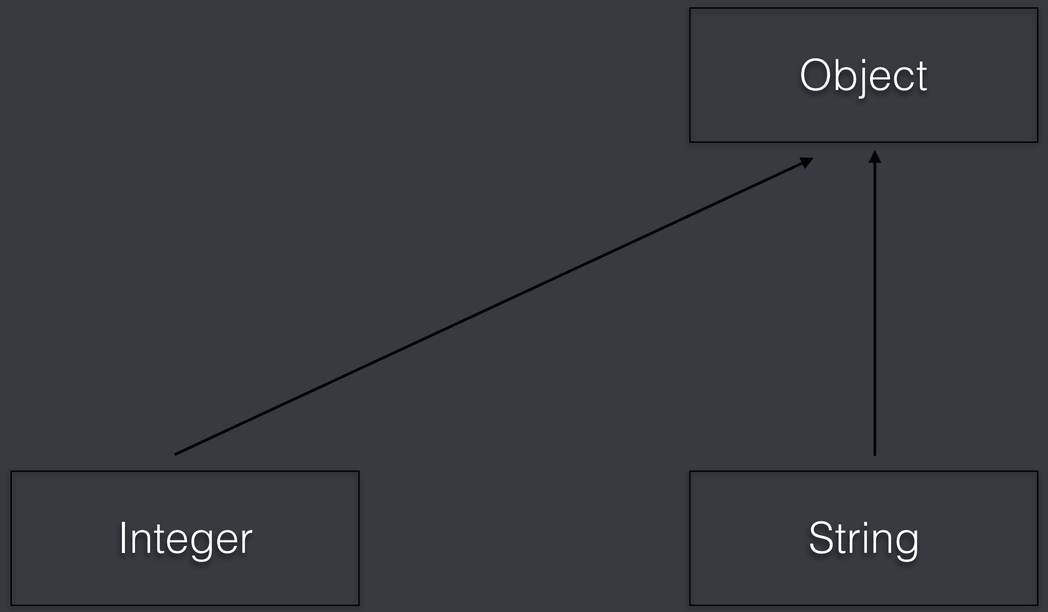
Type Hierarchy (Final) - Java

Object

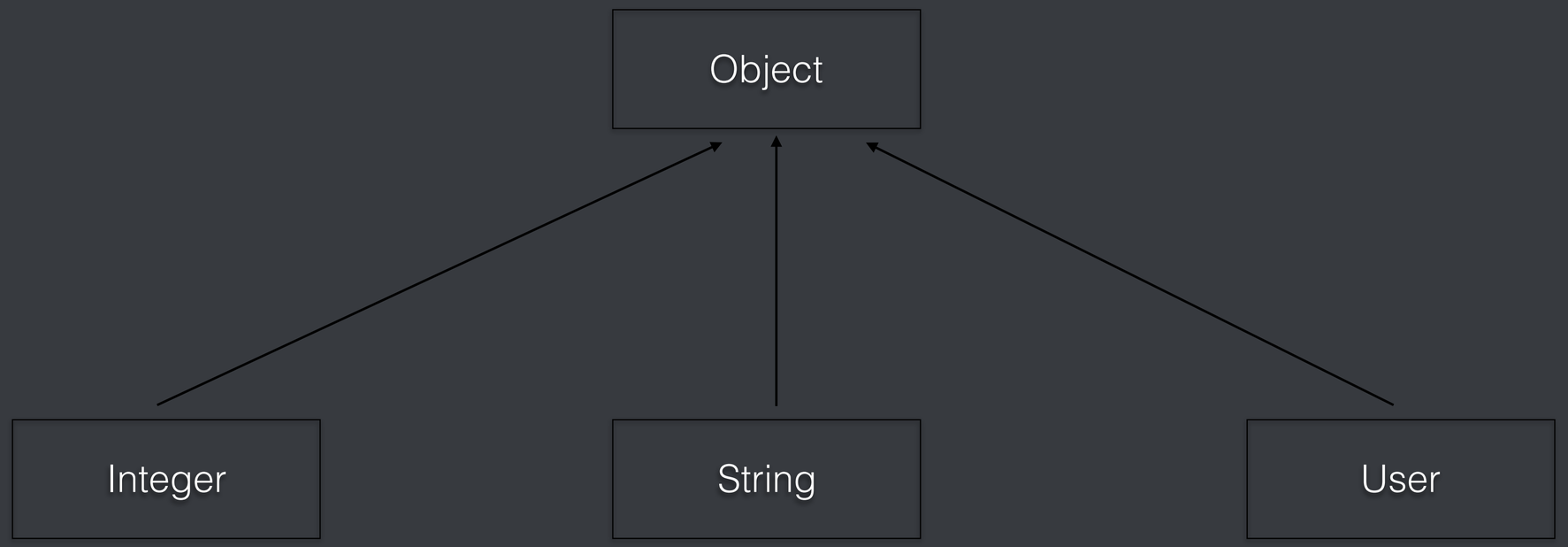
Type Hierarchy (Final) - Java



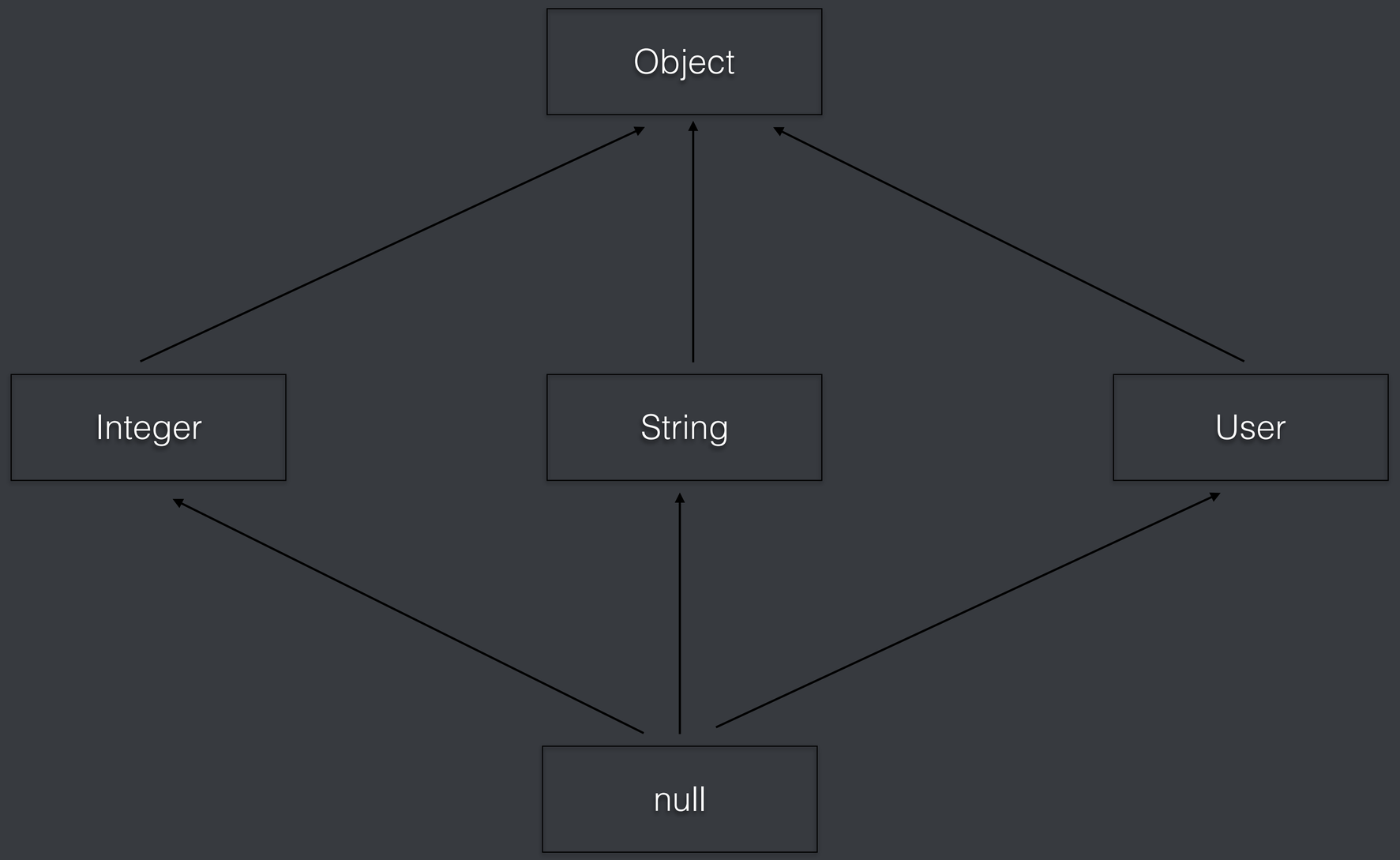
Type Hierarchy (Final) - Java



Type Hierarchy (Final) - Java



Type Hierarchy (Final) - Java

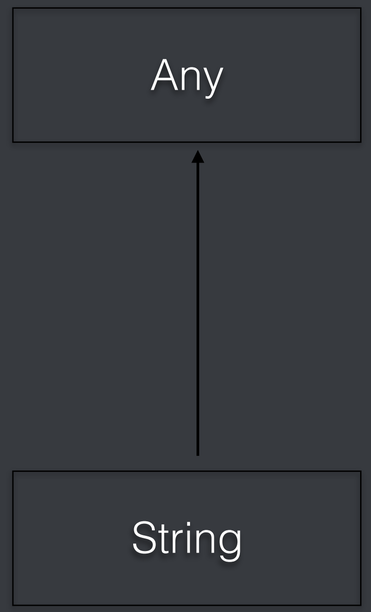


Type Hierarchy - Kotlin

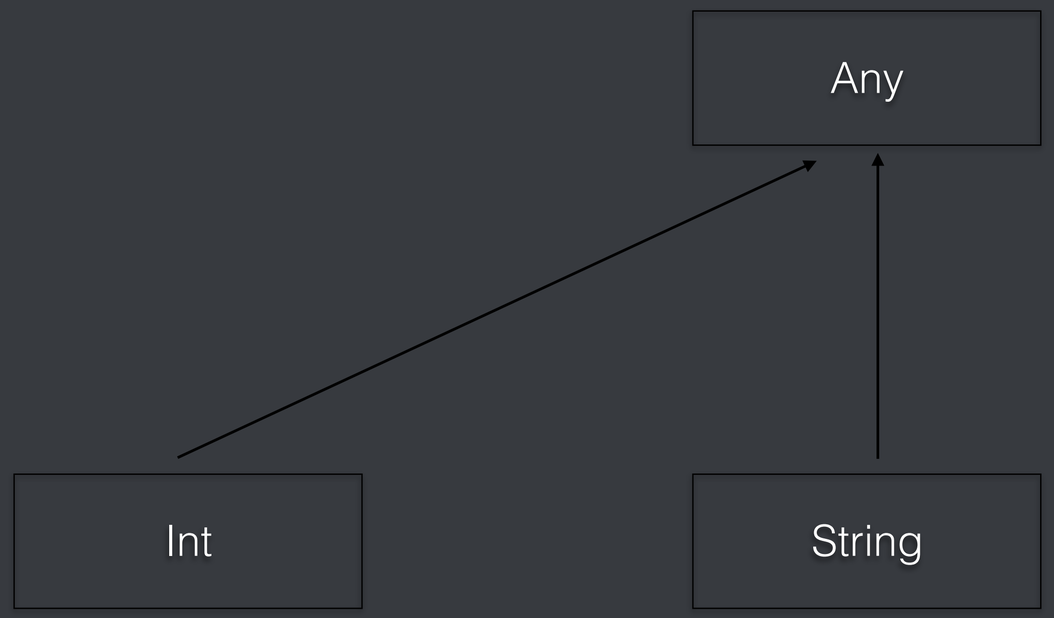
Type Hierarchy - Kotlin

Any

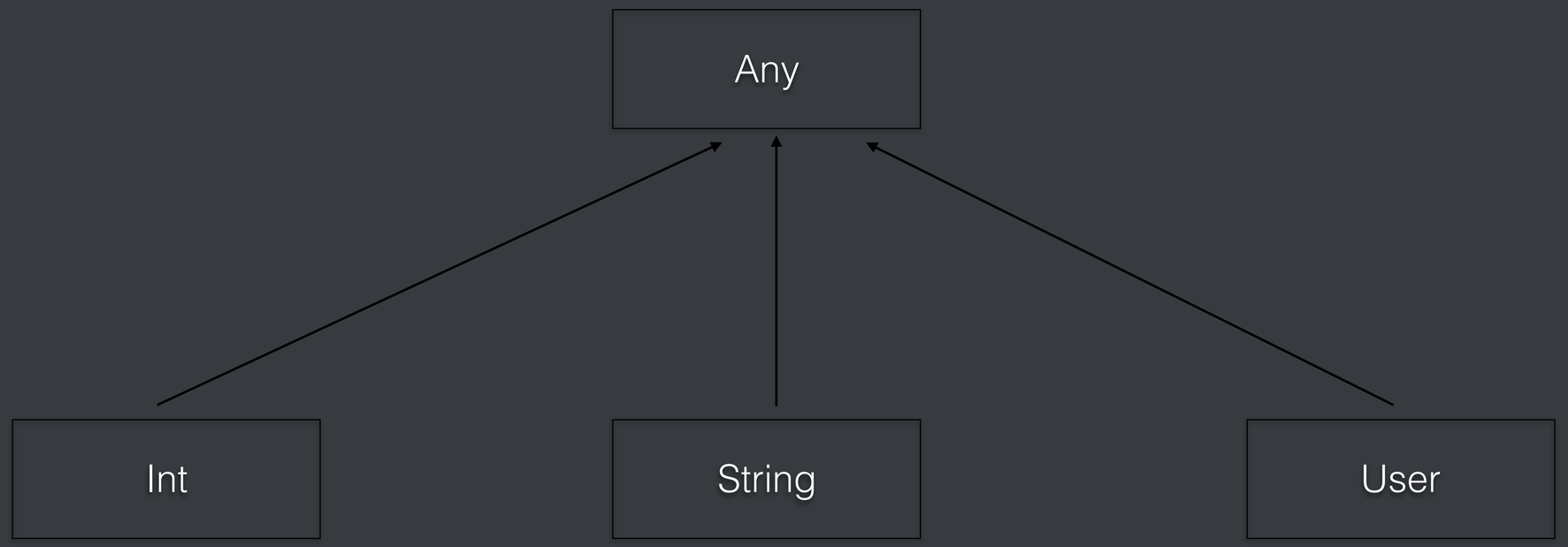
Type Hierarchy - Kotlin



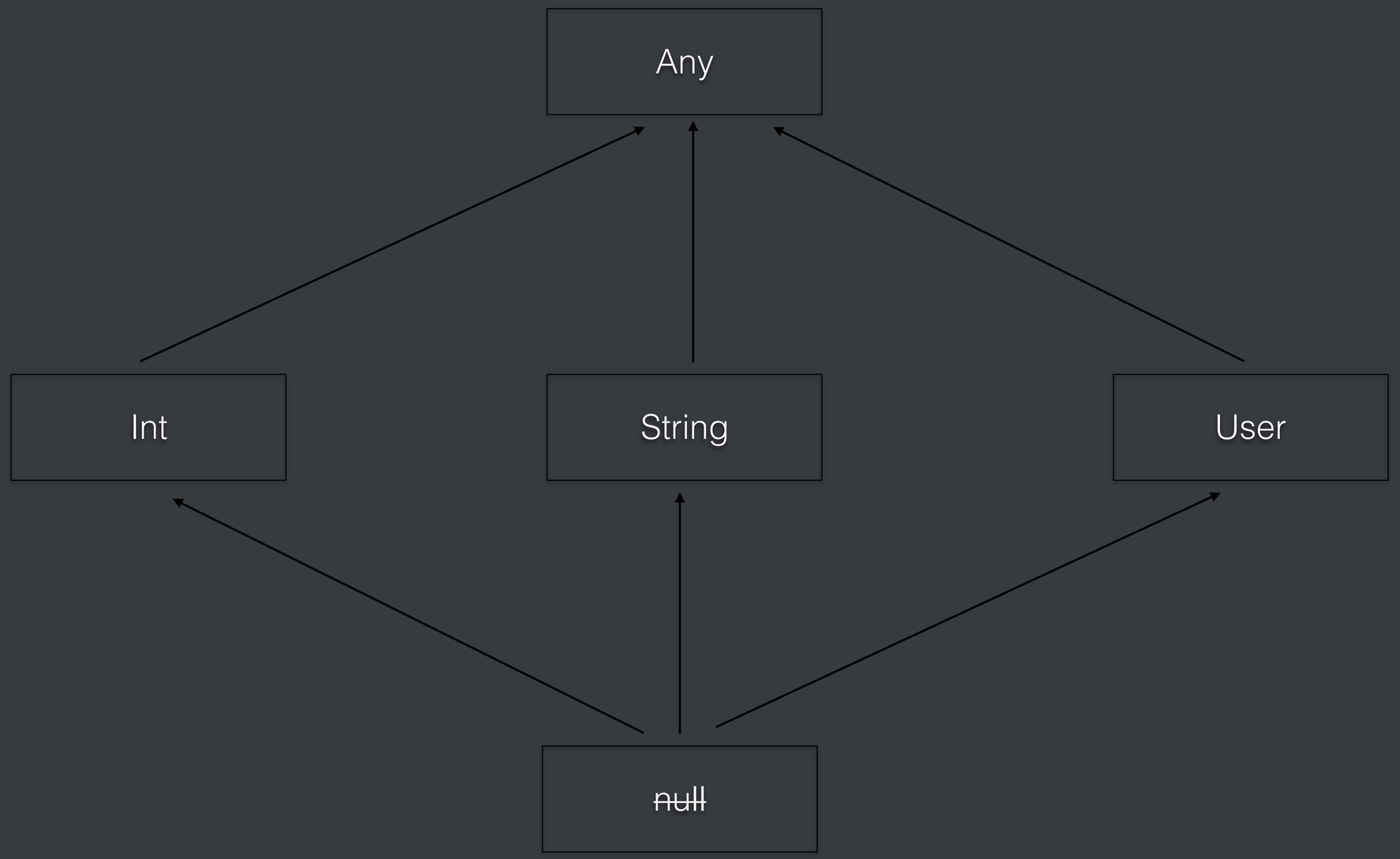
Type Hierarchy - Kotlin



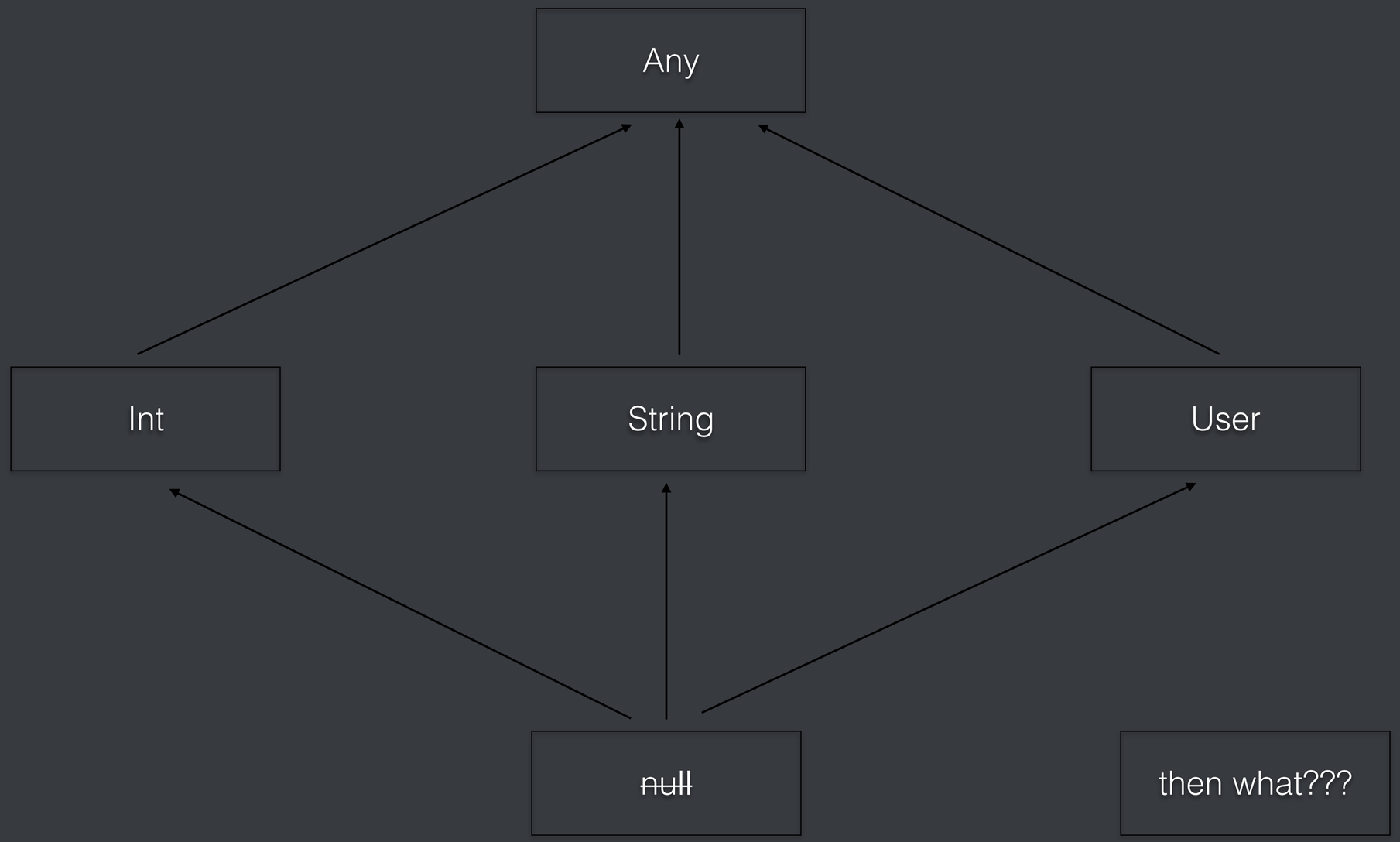
Type Hierarchy - Kotlin



Type Hierarchy - Kotlin



Type Hierarchy - Kotlin



Nothing Type

Nothing Type

Nothing Type

- A special type that exists but doesn’t represent any value.

Nothing Type

- A special type that exists but doesn’t represent any value.
- Should be used to mark any code that can never be reached.

Nothing Type

- A special type that exists but doesn’t represent any value.
- Should be used to mark any code that can never be reached.
- We can also use it while type inference.

Examples - Nothing Type

Examples - Nothing Type

```
val s = person.name ?: throw IllegalArgumentException("Name required")
println(s)     // 's' is known to be initialized at this point
```

Examples - Nothing Type

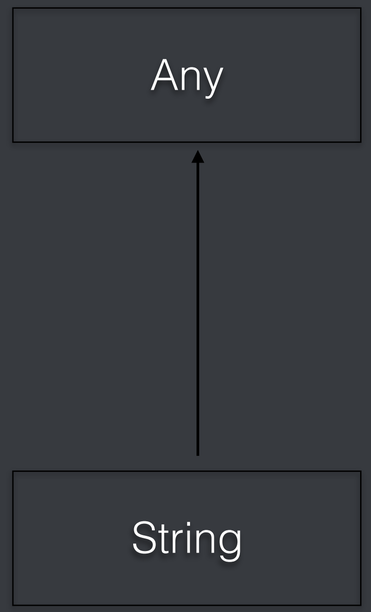
```
val s = person.name ?: throw IllegalArgumentException("Name required")
println(s)     // 's' is known to be initialized at this point


val x = null            // 'x' has type `Nothing?`
val l = listOf(null)   // 'l' has type `List<Nothing?>

var user: User? = x
```

Type Hierarchy - Kotlin

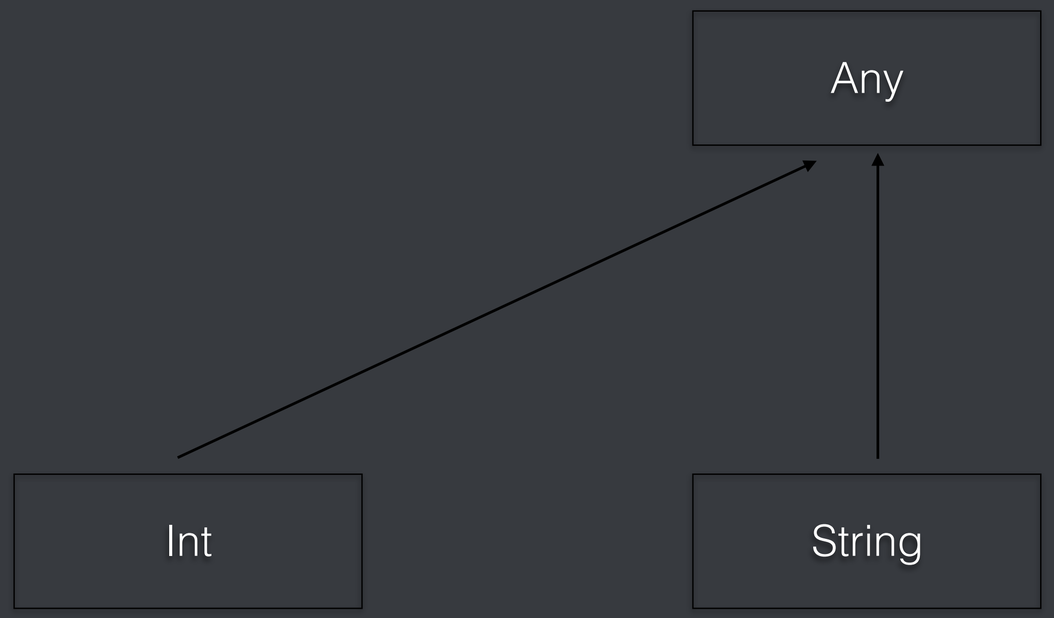
Type Hierarchy - Kotlin

Any

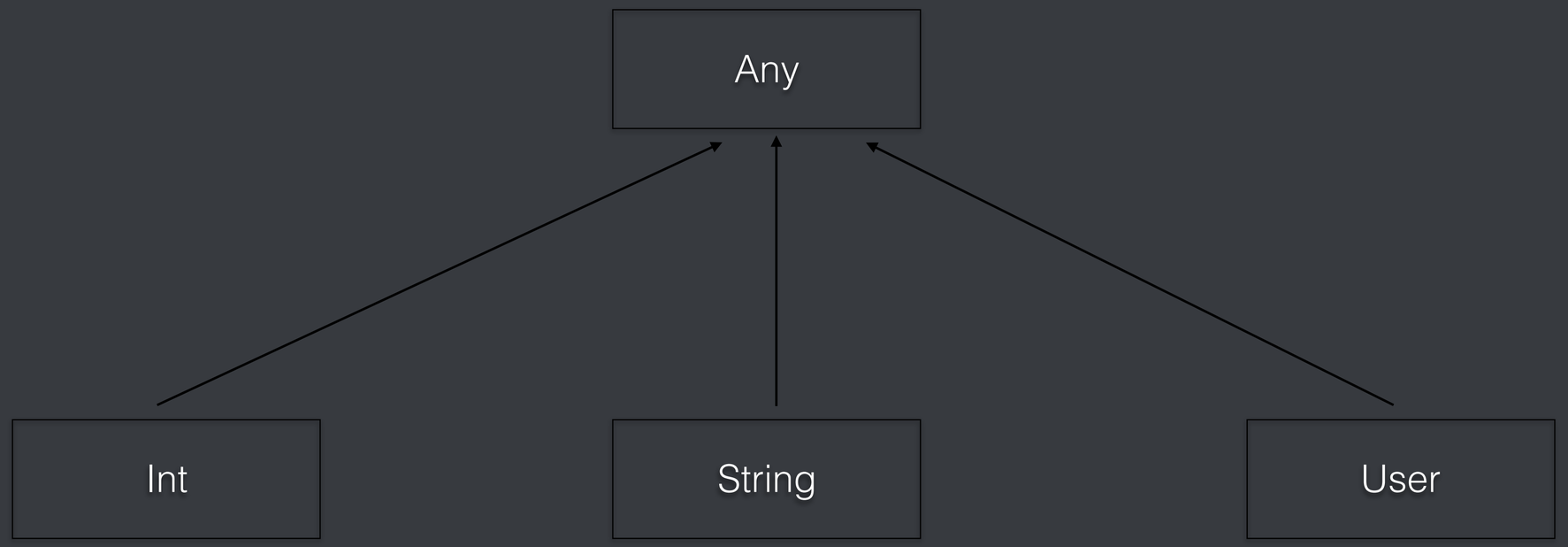
Type Hierarchy - Kotlin



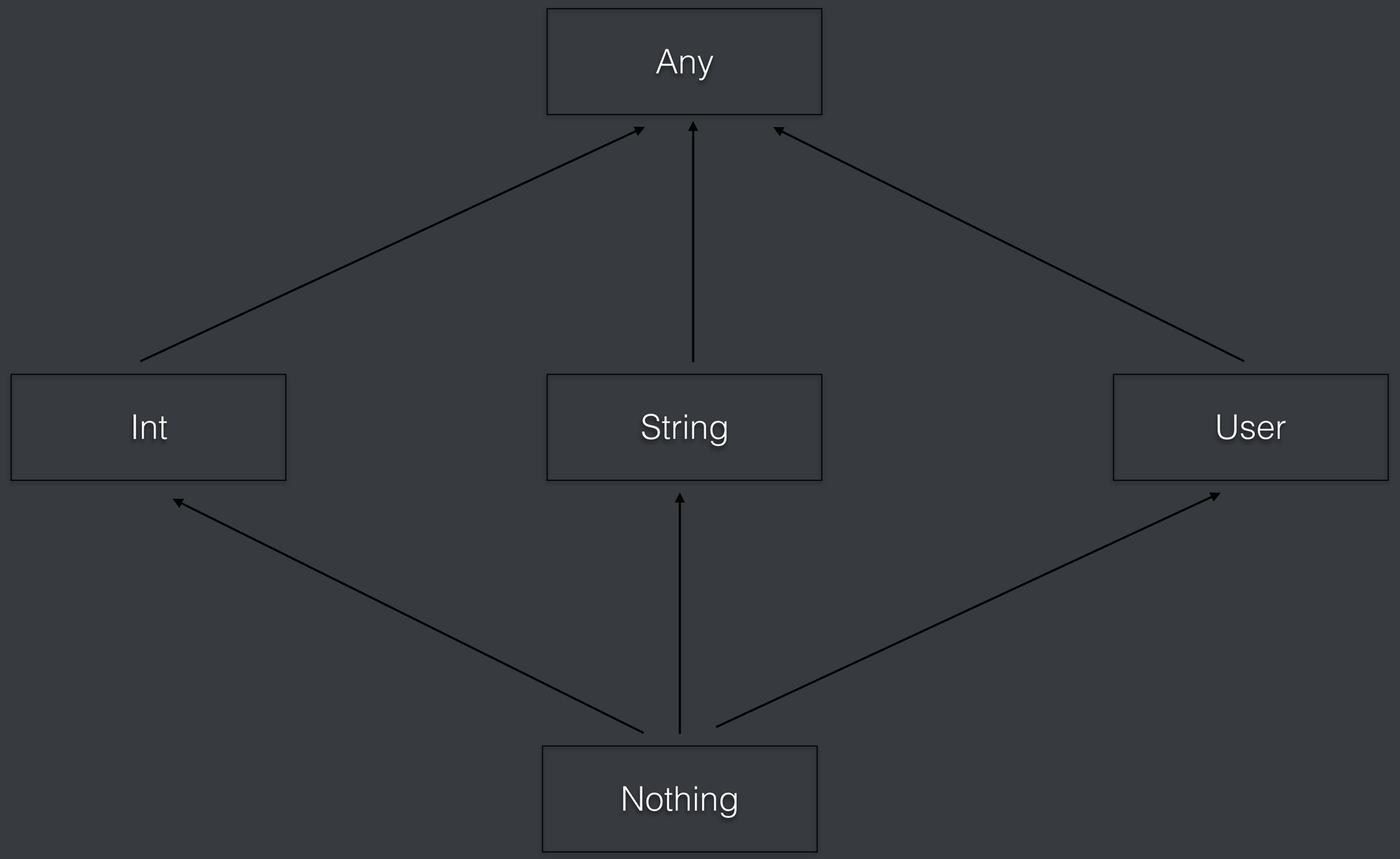
Type Hierarchy - Kotlin



Type Hierarchy - Kotlin

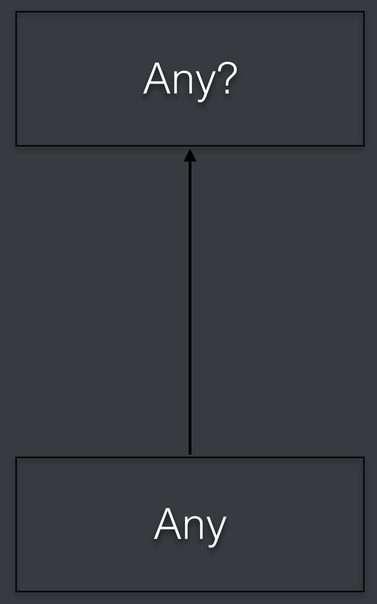


Type Hierarchy - Kotlin

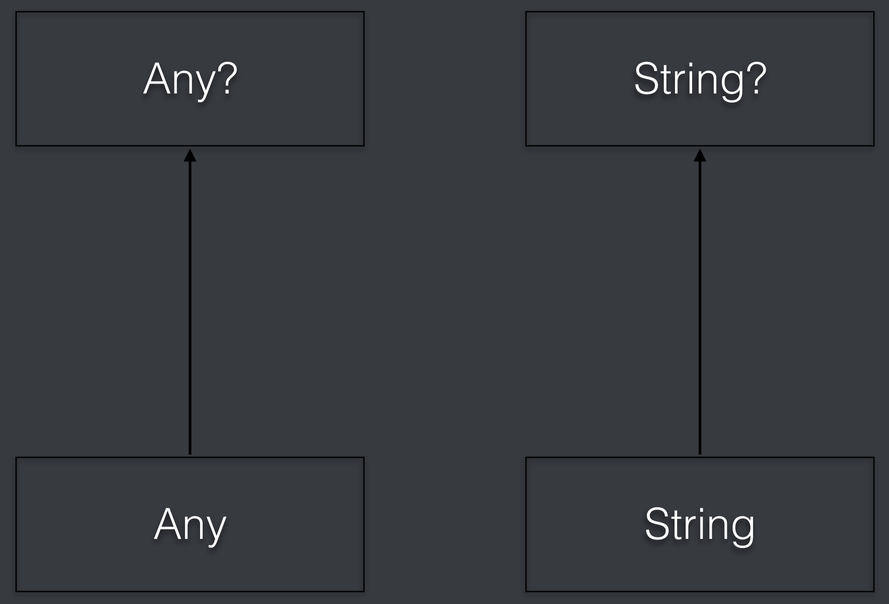


Type Hierarchy - with null - Kotlin

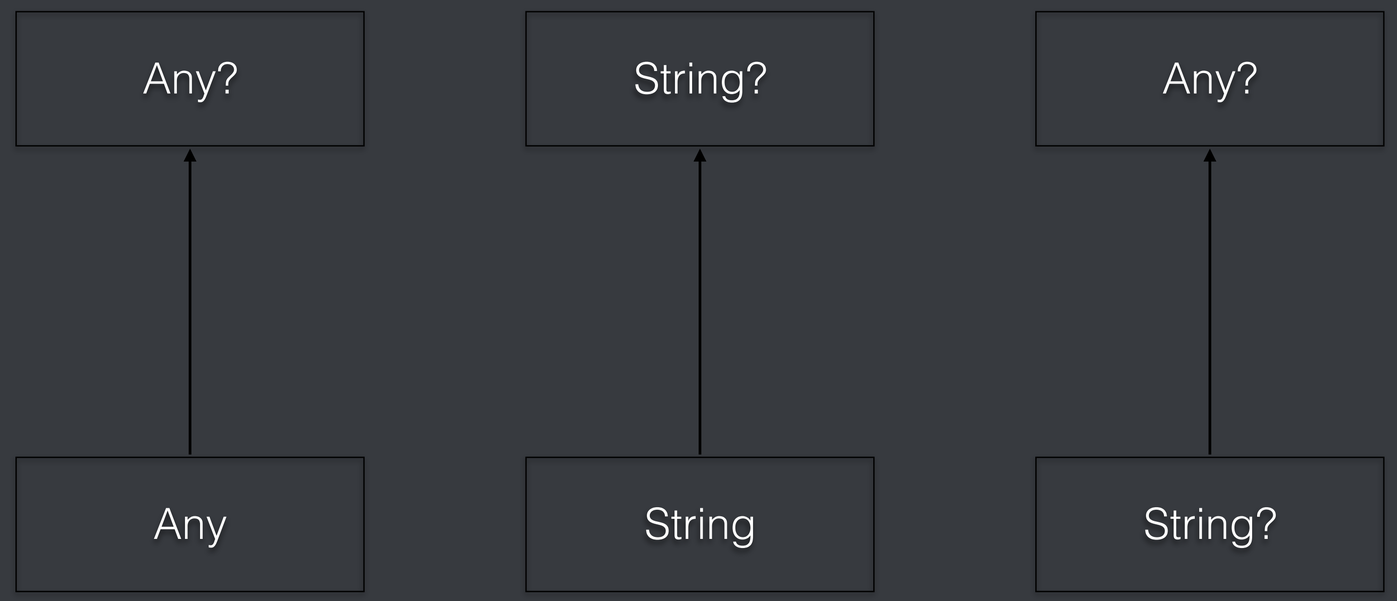
Type Hierarchy - with null - Kotlin



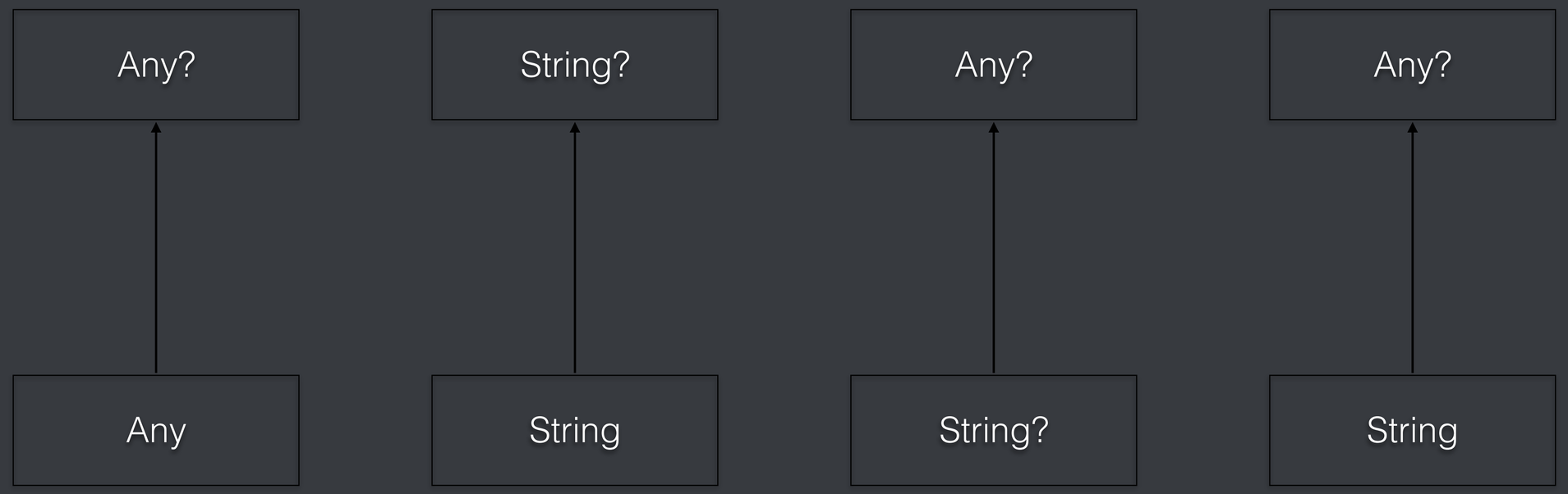
Type Hierarchy - with null - Kotlin



Type Hierarchy - with null - Kotlin



Type Hierarchy - with null - Kotlin

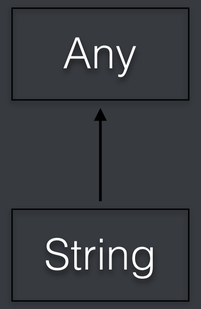


Type Hierarchy (Final) - Kotlin

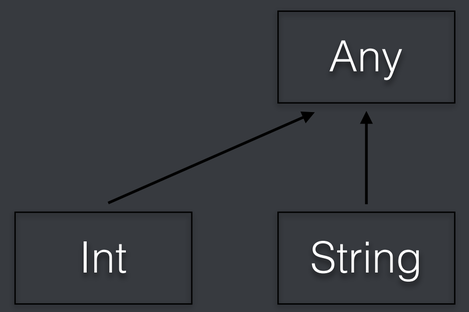
Type Hierarchy (Final) - Kotlin

Any

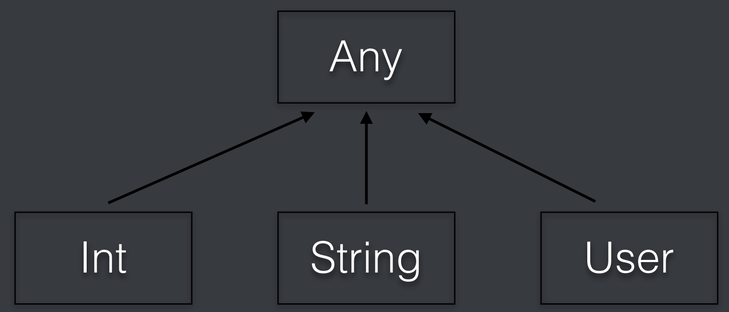
Type Hierarchy (Final) - Kotlin



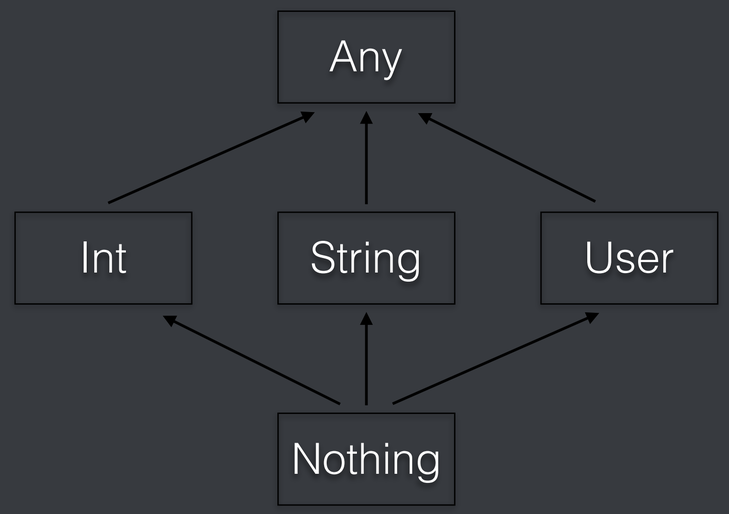
Type Hierarchy (Final) - Kotlin



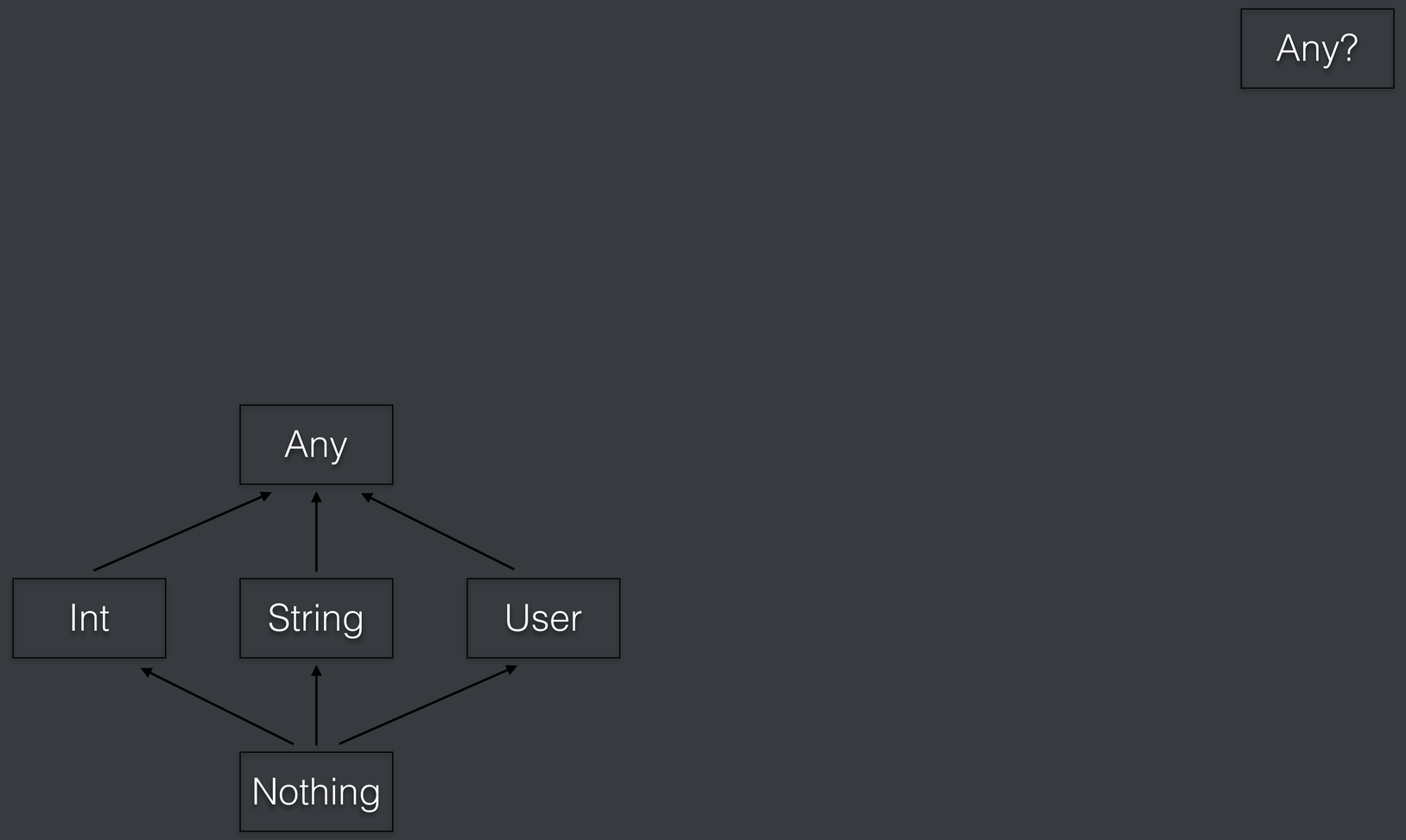
Type Hierarchy (Final) - Kotlin



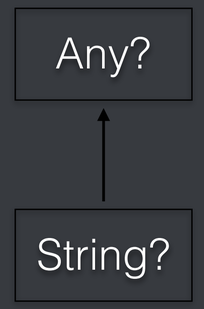
Type Hierarchy (Final) - Kotlin

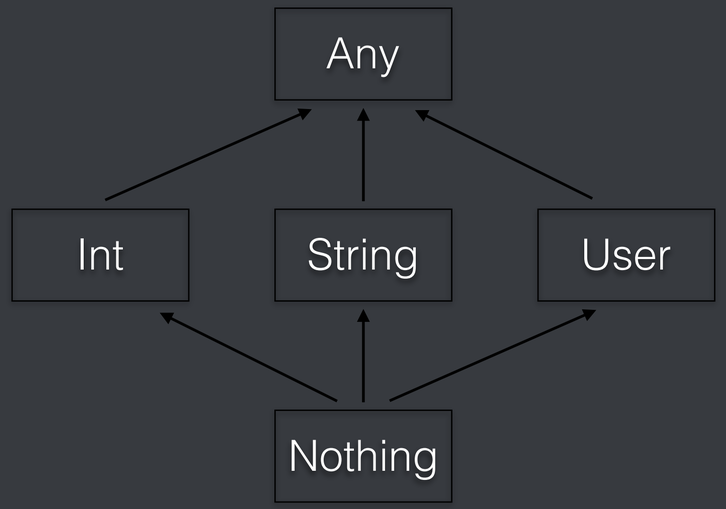


Type Hierarchy (Final) - Kotlin

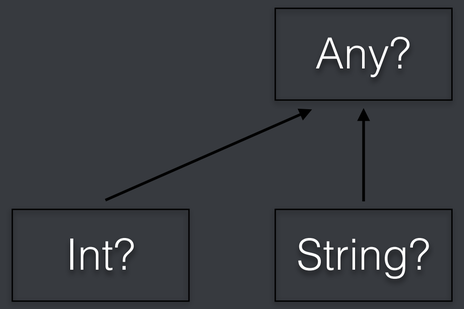


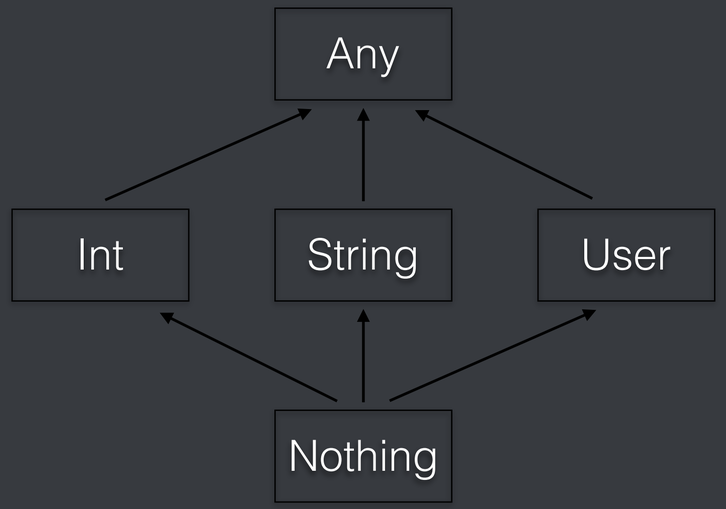
Type Hierarchy (Final) - Kotlin



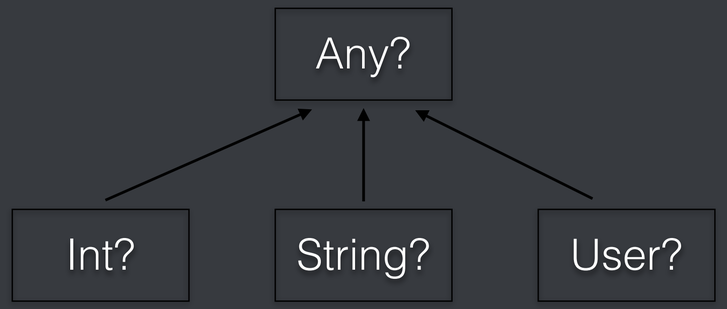


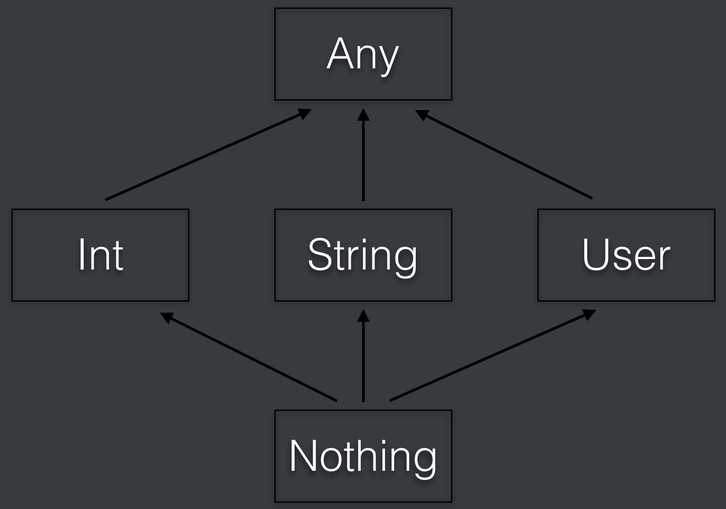
Type Hierarchy (Final) - Kotlin



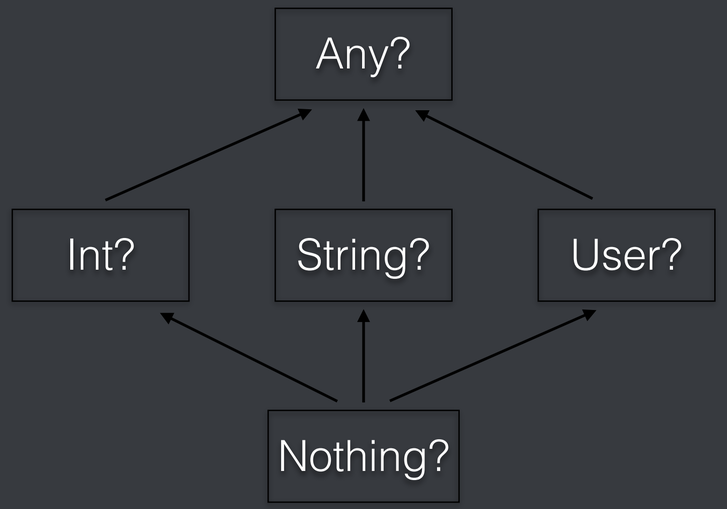


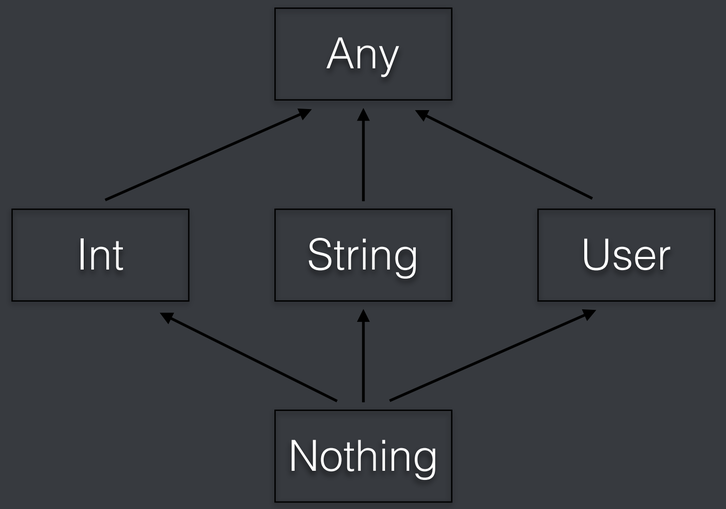
Type Hierarchy (Final) - Kotlin



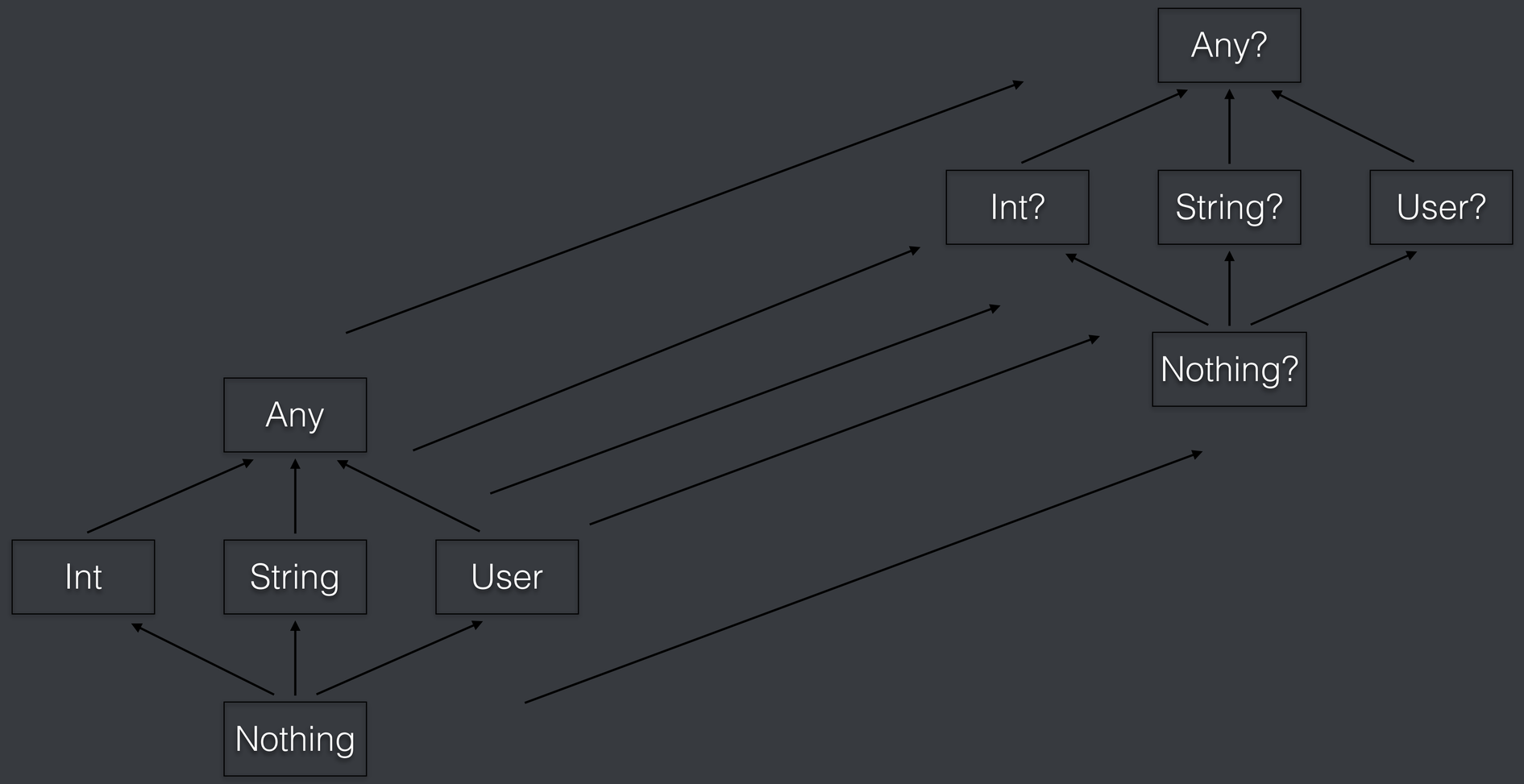


Type Hierarchy (Final) - Kotlin





Type Hierarchy (Final) - Kotlin



void == Unit

Examples - void and Unit

Examples - void and Unit

```
private void test() {}

private void test() {
    return;
}
```

Examples - void and Unit

```
private void test() {}

private void test() {
    return;
}
```

```
private fun test(): Unit {}

private fun test(): Unit {
    return Unit
}

private fun test() {}

private fun test(): Unit {}
```

Array Type

Array Type

Array Type

- Kotlin has no [] syntax for creating Arrays.

Array Type

- Kotlin has no [] syntax for creating Arrays.
- Instead it has a special classs called Array<T>

Examples - Array Type

Examples - Array Type

```
String[] names = new String[10];
String[] colors = new String[] {"Red", "Green", "Blue"};
```

Examples - Array Type

```
String[] names = new String[10];
String[] colors = new String[] {"Red", "Green", "Blue"};
```

```
val names: Array<String> = emptyArray()
val colors: Array<String> = arrayOf("Red", "Green", "Blue")
val nulls: Array<String?> = arrayOfNulls(10)
```

Examples - Array of Primitive Type

Examples - Array of Primitive Type

```
int[] numbers = new int[10];
int[] nums = new int[] {1, 2, 3, 4, 5};
```

Examples - Array of Primitive Type

```
int[] numbers = new int[10];
int[] nums = new int[] {1, 2, 3, 4, 5};

val numbers: Array<Int> = emptyArray()
val nums: Array<Int> = arrayOf(2, 3, 4)
val nulls: Array<Int?> = arrayOfNulls(10)
```

Examples - Array of Primitive Type

```
int[] numbers = new int[10];
int[] nums = new int[] {1, 2, 3, 4, 5};

val numbers: Array<Int> = emptyArray()
val nums: Array<Int> = arrayOf(2, 3, 4)
val nulls: Array<Int?> = arrayOfNulls(10)
```

} Integer[]

Examples - Array of Primitive Type

```
int[] numbers = new int[10];
int[] nums = new int[] {1, 2, 3, 4, 5};

val numbers: Array<Int> = emptyArray()
val nums: Array<Int> = arrayOf(2, 3, 4)
val nulls: Array<Int?> = arrayOfNulls(10)

val nums: IntArray = intArrayOf(1, 2, 3)
val longs: LongArray = longArrayOf(1L, 2L, 3L)
```

} Integer[]

Examples - Array of Primitive Type

```
int[] numbers = new int[10];
int[] nums = new int[] {1, 2, 3, 4, 5};

val numbers: Array<Int> = emptyArray()
val nums: Array<Int> = arrayOf(2, 3, 4)
val nulls: Array<Int?> = arrayOfNulls(10)

val nums: IntArray = intArrayOf(1, 2, 3)
val longs: LongArray = longArrayOf(1L, 2L, 3L)
```

} Integer[]

} int[]

Nullable Types in Java

Nullable Types - Interoperability

Nullable Types - Interoperability

Since Java doesn’t have nullable types, we need to take extra care of our code while taking adventage of interoperability.

Nullable Types - Interoperability

Nullable Types - Interoperability

```
// Test.java
public class Test {
    public String test() {
        return null;
    }
}
```

Nullable Types - Interoperability

```java
// Test.java
public class Test {
    public String test() {
        return null;
    }
}
```

```kotlin
// TestKotlin.kt
class Test {
    private test() {
        val test = Test()
        test.test().toString()
    }
}
```

Nullable Types - Interoperability

```java
// Test.java
public class Test {
    public String test() {
        return null;
    }
}
```

```kotlin
// TestKotlin.kt
class Test {
    private test() {
        val test = Test()
        test.test().toString()
    }
}
```

This will lead to NPE

How to avoid such NPE?

How to avoid such NPE?

```
// Test.java
public class Test {
    @Nullable
    public String test() {
        return null;
    }
}
```

How to avoid such NPE?

```java
// Test.java
public class Test {
    @Nullable
    public String test() {
        return null;
    }
}
```

```kotlin
// TestKotlin.kt
class Test {
    private test() {
        val test = Test()
        test.test().toString()
    }
}
```

How to avoid such NPE?

```java
// Test.java
public class Test {
    @Nullable
    public String test() {
        return null;
    }
}
```

```kotlin
// TestKotlin.kt
class Test {
    private test() {
        val test = Test()
        test.test().toString()
    }
}
```

Now no NPE as it fails at Compile Time

Take Away

Take Away

- Kotlin has no primitive and wrapper type speartely.

Take Away

- Kotlin has no primitive and wrapper type speartely.
- Nullable and Non-Nullable values are two separate types.

Take Away

- Kotlin has no primitive and wrapper type speartely.
- Nullable and Non-Nullable values are two separate types.
- By default all types are Non-Nullable.

Take Away

- Kotlin has no primitive and wrapper type speartely.
- Nullable and Non-Nullable values are two separate types.
- By default all types are Non-Nullable.
- Type and Type? are not same.

Take Away - Continued…

Take Away - Continued…

- Object in Java is same as Any in Kotlin.

Take Away - Continued…

- Object in Java is same as Any in Kotlin.
- void in Java is same as Unit in Kotlin.

Take Away - Continued…

- Object in Java is same as Any in Kotlin.
- void in Java is same as Unit in Kotlin.
- Nothing is not same as Unit or void.

Take Away - Continued…

- Object in Java is same as Any in Kotlin.
- void in Java is same as Unit in Kotlin.
- Nothing is not same as Unit or void.
- Nothing is converted into void in byte code level as JVM doesn’t has it.

Take Away - Continued…

- Object in Java is same as Any in Kotlin.
- void in Java is same as Unit in Kotlin.
- Nothing is not same as Unit or void.
- Nothing is converted into void in byte code level as JVM doesn’t has it.
- Always annotate your Java types to avoid NPE in Kotlin world.

Thank You

Chandra Sekhar
@iChanSek