KOTLIN VS JAVA

Exploring the differences of Kotlin Vs Java

- Kotlin is a statically typed programming language for Java Virtual Machine (JVM) and JavaScript. Described as a general purpose language, Kotlin introduces functional features to support Java interoperability. The Kotlin project was born out of the aspiration for heightened productivity. The goal was to improve the coding experience in a way that was both practical and effective.

- A central focus for Kotlin is to enable mixed-language projects. Kotlin also introduces improved syntax, as well as concise expressions and abstractions. Using Kotlin with Java reduces excessive boilerplate code, which is a huge win for Android developers.

- In two years, Kotlin has become a more stable and congruous development option for Android Studio. Some developers seem to believe that Kotlin will oust Java for Android development in the coming years. Other experts see Kotlin and Java coexisting without one outweighing the other.

Kotlin features

- Null safety
- No checked exceptions
- Extension functions
- Higher-order functions
- Function types & lambdas
- Default & named arguments
- Properties
- Type inference
- Operator overloading
- Smart casts
- Data classes
- Immutable collections
- Enhanced switch-case
- String interpolation
- Ranges
- Inline functions
- Infix notation
- Tail recursion
- Coroutines (async/await)
- Great Standard Library
- Sealed classes
- Delegated & lazy properties
- Class delegation
- Singletons
- Nested functions
- Object decomposition
- Top-level functions
- Reified generics
- Raw Strings
- And more…

+ 100% Java interoperable + Compiles to Java 6 bytecode
+ Syntax similar to Java/C#/JavaScript + Great tooling
+ Great community + Rapid development

Basic syntax

Functions

```java
// Java
public int sum(int a, int b) {
    return a + b;
}
```

```kotlin
// Kotlin
public fun sum(a: Int, b: Int): Int {
    return a + b
}
```

Functions

```java
// Java
public int sum(int a, int b) {
    return a + b;
}
```

```kotlin
// Kotlin
public fun sum(a: Int, b: Int): Int {
    return a + b
}
```

Functions

```java
// Java
public int sum(int a, int b) {
    return a + b;
}
```

```kotlin
// Kotlin
public fun sum(a: Int, b: Int): Int {
    return a + b
}
```

Functions

```java
// Java
public int sum(int a, int b) {
    return a + b;
}
```

```kotlin
// Kotlin
public fun sum(a: Int, b: Int): Int {
    return a + b
}
```

Functions

```java
// Java
public int sum(int a, int b) {
    return a + b;
}
```

```kotlin
// Kotlin
public fun sum(a: Int, b: Int): Int {
    return a + b
}
```

Functions

```java
// Java
public int sum(int a, int b) {
    return a + b;
}
```

```kotlin
// Kotlin
public fun sum(a: Int, b: Int): Int {
    return a + b
}
```

Functions

```java
// Java
public int sum(int a, int b) {
    return a + b;
}
```

```kotlin
// Kotlin
public fun sum(a: Int, b: Int): Int = a + b
```

Functions

```java
// Java
public int sum(int a, int b) {
    return a + b;
}
```

```kotlin
// Kotlin
public fun sum(a: Int, b: Int) = a + b
```

Function Arguments

```
// Java
public void repeat(String str, int count, String separator) {
    ...
}

repeat("abc", 5, "");
```

Function Arguments

```java
// Java
public void repeat(String str, int count, String separator) {
    ...
}

repeat("abc", 5, "");
```

```kotlin
// Kotlin
fun repeat(str: String, count: Int, separator: String) {
    ...
}

repeat("abc", 5, "")
```

Function Arguments

```java
// Java
public void repeat(String str, int count, String separator) {
    ...
}

repeat("abc", 5, "");
```

```kotlin
// Kotlin
fun repeat(str: String, count: Int = 3, separator: String = "") {
    ...
}

repeat("abc", 5, "")
```

Function Arguments

```java
// Java
public void repeat(String str, int count, String separator) {
    ...
}

repeat("abc", 5, "");
```

```kotlin
// Kotlin
fun repeat(str: String, count: Int = 3, separator: String = "") {
    ...
}

repeat("abc", 5)
```

Function Arguments

```java
// Java
public void repeat(String str, int count, String separator) {
    ...
}

repeat("abc", 5, "");
```

```kotlin
// Kotlin
fun repeat(str: String, count: Int = 3, separator: String = "") {
    ...
}

repeat("abc", count = 5)
```

Function Arguments

```java
// Java
public void repeat(String str, int count, String separator) {
    ...
}

repeat("abc", 5, "");
```

```kotlin
// Kotlin
fun repeat(str: String, count: Int = 3, separator: String = "") {
    ...
}

repeat("abc", count = 5, separator = ",")
```

Function Arguments

```java
// Java
public void repeat(String str, int count, String separator) {
    ...
}

repeat("abc", 5, "");
```

```kotlin
// Kotlin
fun repeat(str: String, count: Int = 3, separator: String = "") {
    ...
}

repeat("abc", separator = ",")
```

Variables

```java
// Java
int a = 1;
String b = "xyz";
```

```kotlin
// Kotlin
val a: Int = 1
val b: String = "xyz"
```

Variables

```java
// Java
int a = 1;
String b = "xyz";
```

```kotlin
// Kotlin
val a: Int = 1
val b: String = "xyz"
```

Variables

```java
// Java
int a = 1;
String b = "xyz";
```

```kotlin
// Kotlin
val a: Int = 1
val b: String = "xyz"
```

Variables

```java
// Java
int a = 1;
String b = "xyz";
```

```kotlin
// Kotlin
val a: Int = 1
val b: String = "xyz"
```

Variables

```java
// Java
int a = 1;
String b = "xyz";
```

```kotlin
// Kotlin
val a = 1        // Inferred type is Int
val b = "xyz"    // Inferred type is String
```

(Im)mutability

```
// Assign-once (read-only) variable
val x = 1
x = 2       // Compile-time error


// Mutable variable
var y = 5
y = 10      // OK
```

(Im)mutability

```
// Assign-once (read-only) variable
val x = 1
x = 2       // Compile-time error


// Mutable variable
var y = 5
y = 10      // OK


// Always start with immutable "val"
// Change to "var" only when necessary
```

Null safety

Null safety

```
var str: String = "xyz"

str = null  // ???
```

Null safety

```
var str: String = "xyz"

str = null  // Compile-time error
```

Null safety

```
var str: String? = "xyz"

str = null  // OK
```

Null safety

```
fun getLength(str: String): Int? {
    return str.length   // ???
}
```

Null safety

```
fun getLength(str: String): Int? {
    return str.length   // OK
}
```

Null safety

```
fun getLength(str: String?): Int? {
    return str.length   // ???
}
```

Null safety

```
fun getLength(str: String?): Int? {
    return str.length   // Compile-time error
}
```

Null safety

```
fun getLength(str: String?): Int? {
    if (str != null) {
        return str.length
    }
    return 0
}
```

Null safety

```
fun getLength(str: String?): Int? {
    if (str != null) {
        return str.length    // <-- Smart cast
    }
    return 0
}
```

Null safety

```
fun getLength(str: String?): Int? {
    return str?.length
}
```

Null safety

```
fun getLength(str: String?): Int {
    return str?.length ?: 0
}
```

Null safety

```java
// Java
public ZipCode getZipCode(User user) {
    if (user != null) {
        if (user.getAddress() != null) {
            return user.getAddress().getZipCode();
        }
    }
    return null;
}
```

```kotlin
// Kotlin
fun getZipCode(user: User?): ZipCode? {
    return user?.address?.zipCode
}
```

Null safety

```java
// Java
public ZipCode getZipCode(User user) {
    if (user != null) {
        if (user.getAddress() != null) {
            return user.getAddress().getZipCode();
        }
    }
    return null;
}
```

```kotlin
// Kotlin
fun getZipCode(user: User?) = user?.address?.zipCode
```

String Interpolation

```
override fun toString(): String {
    return "Song{id=" +
            id +
            ", title='" +
            title +
            "', author='" +
            author +
            "'}"
}
```

String Interpolation

```
override fun toString(): String {
    return "Song{id=$id, title='$title', author='$author'}"
}
```

String Interpolation

```
override fun toString() = "Song{id=$id, title='$title', author='$author'}"
```

Classes

```
// Java
public class User {

}
```

Classes

```java
// Java
public class User {

    String firstName;
    String lastName;
    int age;

}
```

Classes

```java
// Java
public class User {

    String firstName;
    String lastName;
    int age;

    public User(String firstName, String lastName, int age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

}
```

Classes

```java
// Java
public class User {

    String firstName;
    String lastName;
    int age;

    public User(String firstName, String lastName, int age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        User user = (User) o;

        if (age != user.age) return false;
        if (firstName != null ? !firstName.equals(user.firstName) : user.firstName != null)
            return false;
        return lastName != null ? lastName.equals(user.lastName) : user.lastName == null;

    }

    @Override
    public int hashCode() {
        int result = firstName != null ? firstName.hashCode() : 0;
        result = 31 * result + (lastName != null ? lastName.hashCode() : 0);
        result = 31 * result + age;
        return result;
    }

    @Override
    public String toString() {
        return "User{" +
                "firstName='" + firstName + '\'' +
                ", lastName='" + lastName + '\'' +
                ", age=" + age +
                '}';
    }

}
```

53 lines of code!

Classes

```kotlin
// Kotlin
class User {

    var firstName: String?
    var lastName: String?
    var age: Int

    constructor(firstName: String?, lastName: String?, age: Int) {
        this.firstName = firstName
        this.lastName = lastName
        this.age = age
    }

    override fun equals(other: Any?): Boolean {
        if (this === other) return true
        if (other?.javaClass != javaClass) return false

        other as User

        if (firstName != other.firstName) return false
        if (lastName != other.lastName) return false
        if (age != other.age) return false

        return true
    }

    override fun hashCode(): Int {
        var result = firstName?.hashCode() ?: 0
        result = 31 * result + (lastName?.hashCode() ?: 0)
        result = 31 * result + age
        return result
    }

    override fun toString(): String {
        return "User{" +
                "firstName='" + firstName + '\'' +
                ", lastName='" + lastName + '\'' +
                ", age=" + age +
                '}'
    }

}
```

Classes

```kotlin
// Kotlin
class User {

    var firstName: String?
    var lastName: String?
    var age: Int

    constructor(firstName: String?, lastName: String?, age: Int) {
        this.firstName = firstName
        this.lastName = lastName
        this.age = age
    }

    override fun equals(other: Any?): Boolean {
        if (this === other) return true
        if (other?.javaClass != javaClass) return false

        other as User

        if (firstName != other.firstName) return false
        if (lastName != other.lastName) return false
        if (age != other.age) return false

        return true
    }

    override fun hashCode(): Int {
        var result = firstName?.hashCode() ?: 0
        result = 31 * result + (lastName?.hashCode() ?: 0)
        result = 31 * result + age
        return result
    }

    override fun toString(): String {
        return "User{" +
                "firstName='" + firstName + '\'' +
                ", lastName='" + lastName + '\'' +
                ", age=" + age +
                '}'
    }
}
```

Classes

```kotlin
// Kotlin
class User {

    var firstName: String?
    var lastName: String?
    var age: Int

    constructor(firstName: String?, lastName: String?, age: Int) {
        this.firstName = firstName
        this.lastName = lastName
        this.age = age
    }

    override fun equals(other: Any?): Boolean {
        if (this === other) return true
        if (other?.javaClass != javaClass) return false

        other as User

        if (firstName != other.firstName) return false
        if (lastName != other.lastName) return false
        if (age != other.age) return false

        return true
    }

    override fun hashCode(): Int {
        var result = firstName?.hashCode() ?: 0
        result = 31 * result + (lastName?.hashCode() ?: 0)
        result = 31 * result + age
        return result
    }

    override fun toString() = "User{firstName='$firstName', lastName='$lastName', age=$age}"

}
```

Classes

```kotlin
// Kotlin
class User(var firstName: String?, var lastName: String?, var age: Int) {

    override fun equals(other: Any?): Boolean {
        if (this === other) return true
        if (other?.javaClass != javaClass) return false

        other as User

        if (firstName != other.firstName) return false
        if (lastName != other.lastName) return false
        if (age != other.age) return false

        return true
    }

    override fun hashCode(): Int {
        var result = firstName?.hashCode() ?: 0
        result = 31 * result + (lastName?.hashCode() ?: 0)
        result = 31 * result + age
        return result
    }

    override fun toString() = "User{firstName='$firstName', lastName='$lastName', age=$age}"

}
```

Data Classes

```kotlin
// Kotlin
data class User(var firstName: String?, var lastName: String?, var age: Int)
```

Data Classes

```kotlin
// Kotlin
data class User(var firstName: String?, var lastName: String?, var age: Int)

// data = equals() + hashCode() + toString() + copy()
```

1 line in Kotlin
vs
53 lines in Java

Multiple items in file

```kotlin
// Models.kt
data class User(val firstName: String, val lastName: String, val age: Int)
data class Address(val street: String, val zipCode: ZipCode)
data class ZipCode(val prefix: String, val postfix: String)
```

Properties

```
class User {
    var firstName: String? = ...    // mutable (getter/setter)
    var lastName: String = ...      // mutable
    val age: Int = ...              // read-only (getter only)
}


// Use them as fields
fun test() {
    val user = User()
    user.firstName = “Grzegorz”             // setter is called
    print("firstName = ${user.firstName}")  // getter is called
}
```

Properties

```
class User {
    var firstName: String? = ""
    var lastName: String = ""
    val age: Int = 0
}
```

Properties

```
class User {
    var firstName: String? = ""
        get() {
            return "abc"
        }

    var lastName: String = ""
    val age: Int = 0
}
```

Properties

```
class User {
    var firstName: String? = ""
        get() = "abc"
        set(value) {
            field = value + "xyz"
        }

    var lastName: String = ""
    val age: Int = 0
}
```

Extensions

```
// Java
public class StringUtils {
    public static String toCamelCase(String str) {
        return str.replaceAll...
    }
}
```

Extensions

```
// Java
public class StringUtils {
    public static String toCamelCase(String str) {
        return str.replaceAll...
    }
}
```

```
String camelStr = StringUtils.toCamelCase("lorem ipsum");
```

Extensions

```
// Java
public class StringUtils {
    public static String toCamelCase(String str) {
        return str.replaceAll...
    }
}
```

```
import static com.demo.StringUtils.toCamelCase;

String camelStr = toCamelCase("lorem ipsum");
```

Extensions

```
// Java
public class StringUtils {
    public static String toCamelCase(String str) {
        return str.replaceAll...
    }
}
```

```
import static com.demo.StringUtils.toCamelCase;

BufferedReader br = new BufferedReader(new
StringReader(toCamelCase("lorem ipsum")));

...

br.readLine();
```

Extensions

```
// Kotlin
fun toCamelCase(str: String): String {
    return str.replace...
}
```

Extensions

```
// Kotlin
fun String.toCamelCase(str: String): String {
    return str.replace...
}
```

Extensions

```kotlin
// Kotlin
fun String.toCamelCase(): String {
    return this.replace...
}
```

Extensions

```
// Kotlin
fun String.toCamelCase(): String {
    return this.replace...
}
```

```
"lorem ipsum".toCamelCase()
```

Extensions

```
// MyFunctions.kt
package com.kotlin.demo

fun String.toCamelCase(): String {
    return this.replace...
}
```

```
// Demo.kt
import com.kotlin.demo.toCamelCase

"lorem ipsum".toCamelCase()
```

Extensions

```
// MyFunctions.kt
package com.kotlin.demo

fun String.toCamelCase(): String {
    return this.replace...
}
```

```
// Demo.kt
import com.kotlin.demo.toCamelCase

"lorem ipsum".toCamelCase().reader().forEachLine { line ->
    print(line)
}
```

Extensions

```
listOf(1, 2, null, 4, 5)
        .filterNotNull()
        .filter({ it -> it > 2 })
        .forEach({ it ->
            print(it)
        })
```

Extensions

```
listOf(1, 2, null, 4, 5)                    // factory of immutable lists
        .filterNotNull()
        .filter({ it -> it > 2 })
        .forEach({ it ->
            print(it)
        })
```

Extensions

```
listOf(1, 2, null, 4, 5)
        .filterNotNull()                // extension of Iterable
        .filter({ it -> it > 2 })       // extension of Iterable
        .forEach({ it ->                // extension of Iterable
            print(it)
        })


                                        // Why extensions?!
```

Extensions

```
listOf(1, 2, null, 4, 5)                    // kotlin.collections.List<T>
        .filterNotNull()                    // covers java.util.List<T>
        .filter({ it -> it > 2 })
        .forEach({ it ->
            print(it)
        })
```

Lambdas

```
listOf(1, 2, null, 4, 5)
        .filterNotNull()
        .filter({ it -> it > 2 })
        .forEach({ it ->
            print(it)
        })
```

Lambdas

```
listOf(1, 2, null, 4, 5)
        .filterNotNull()
        .filter() { it -> it > 2 }
        .forEach() { it ->
            print(it)
        }
```

Lambdas

```
listOf(1, 2, null, 4, 5)
        .filterNotNull()
        .filter { it -> it > 2 }
        .forEach { it ->
            print(it)
        }
```

Lambdas

```
listOf(1, 2, null, 4, 5)
        .filterNotNull()
        .filter { it > 2 }
        .forEach {
            print(it)
        }
```

Example - RxJava

```java
// Java
Observable.just("1", "5", "10", "20")
        .map(new Func1<String, Integer>() {
            @Override
            public Integer call(String s) {
                return Integer.parseInt(s);
            }
        })
        .filter(new Func1<Integer, Boolean>() {
            @Override
            public Boolean call(Integer integer) {
                return integer > 5;
            }
        })
        .subscribe(new Action1<Integer>() {
            @Override
            public void call(Integer integer) {
                System.out.println(integer);
            }
        });
```

Example - RxJava

```java
// Java
Observable.just("1", "5", "10", "20")
        .map(new Func1<String, Integer>() {
            @Override
            public Integer call(String s) {
                return Integer.parseInt(s);
            }
        })
        .filter(new Func1<Integer, Boolean>() {
            @Override
            public Boolean call(Integer integer) {
                return integer > 5;
            }
        })
        .subscribe(new Action1<Integer>() {
            @Override
            public void call(Integer integer) {
                System.out.println(integer);
            }
        });
```

```kotlin
// Kotlin
Observable.just("1", "5", "10", "20")
        .map { it.toInt() }
        .filter { it > 10 }
        .subscribe {
            print(it)
        }
```

Conclusions

1. Kotlin Increases Productivity

- Increased readability (minimum boilerplate)
- Includes modern features
- Improved safety (no NPEs)
- Since most barriers are gone, devs can focus on more important things - e.g. system architecture & security

2. Kotlin is Ready for Production

- IntelliJ has parts written in Kotlin
- JetBrains Rider - C# IDE - is written entirely in Kotlin
- 150,000 developers all over the world use Kotlin
- Gradle introduced Kotlin support
- Spring 5 introduces Kotlin support
- Kotlin enters TIOBE top 100