Kotlin for FRC Robot Programming

Rhys Kenwell
FRC Team 4069

Solomon Greenberg
FRC Team 2898

August 3, 2018

Contents

Abstract

FIRST offers three officially supported languages for robot code development in the *FIRST* Robotics Competition: Java, C++, and LabVIEW. Unlike the other two, Java is compiled to an intermediary bytecode, allowing teams to use a wide range of languages that compile to the same intermediary. Some of these alternate JVM languages can prove to be highly useful, one of which is Kotlin, the subject of this paper.
Kotlin can provide major benefits to productivity and code safety without sacrificing any performance, benefits such as null safety, and boilerplate reduction. Through Kotlin, robot coding can be faster and less error prone.

1 A History

C++ and Java are both relatively older languages, having first appeared in 1985 and 1995 respectively. Over their lifespan, these languages have established themselves in many fields. For instance, Java is core to Android, and is used frequently for the creation of backend systems. C++ is used frequently in low level programming, such as in device drivers and operating systems. Due to their age, these languages have amassed problems that can make them more difficult to program in. Java's rampant getters and setters, and C++'s inconsistent operator overloading are two signs that these languages' age are starting to become a productivity risk to the developer

1.1 Kotlin

Kotlin is a language that first appeared in 2011. Created from the ground up by Jetbrains, Kotlin was designed to fix problems in Java caused by its age. Since its inception, Kotlin has expanded to be able to compile to JavaScript, and native machine code.
The design goals were simple

- Be Java-compatible
- Compile at least as fast as Java
- Be safer than Java
- Be more concise than Java
- Be expressive

[1]

2 Benefits of Kotlin

In using Kotlin, teams will be able to make use of the numerous benefits that it provides. Some benefits are outlined below[1]

2.1 Null Safety

One of the most common exceptions in Java code is the `NullPointerException`. An integral part of Kotlin's design is its intrinsic null safety, preventing NPEs at compile-time, protecting a robot from being potentially immobilized on the field

Kotlin's type system holds an intrinsic distinction for null types. Objects which can potentially be null — so-called "nullables" or "optionals" — are marked with a `?`. Kotlin provides methods of handling nullalble values without aborting program execution if they are null.
[2]

- Safe-call operator

[1] Not all of the benefits of Kotlin are outlined here. See Section 5 — Resources

The Kotlin safe call operator (`?.`) allows for safe access of a nullable, returning another nullable and avoiding situations where attempted access might lead to a crash

```
1
2  fun main(args: Array<String>) {
3    var s: String = "Hello" // s can never be null
4    s = null // Compile time error
5
6    val i: Int = null // Compile time error
7
8    val j: List<String>? = null // Ok, marked as nullable
9    val s: String? = j?.get(0) // Safe access on j
10 }
```

Listing 1: Usage of the safe call operator

- Elvis operator

Kotlin also provides an operator similar to the ternary operator in Java, but tailored to null types. The elvis operator (`?:`) takes a nullable on the left, and does one of two things

1. If the item is non-null, it returns the inner value of the optional
2. If the item is null, it runs the code on the right (This code may be used to provide a default value, or to bail out of a method call early).

```
1
2  fun main(args: Array<String>) {
3    val i: Int? = 4
4    println(takesANullable(i)) // Will print 5
5    val j: Int? = null
6    println(takesANullable(j)) // Will print -1
7
8    val s = j ?: 5 // Will contain 5
9  }
10
11 fun takesANullable(i: Int?): Int {
12   val iUnwrapped = i ?: return -1 // If i is not null, the inner value will be
         bound in iUnwrapped. If it's null the function will return -1
13
14   return iUnwrapped + 1
15 }
```

Listing 2: Usage of the Elvis operator

In Java, large amounts of code are devoted to ensuring null safety, code which can easily be avoided or shortened in Kotlin.

2.2 Java Interoperability

One of the key goals of Kotlin was to provide seamless Java compatibility. Kotlin and Java should be able to exist in the same codebase, and able to interact with each other with as little overhead as possible. Any existing Java code or libraries will function when being called through Kotlin and vice versaa, allowing robot code written in Kotlin to interact correctly with not just WPILib and Phoenix, but any existing Java library you may be using in your robot.[3]

2.2.1 Null Safety With Java

Depending on the code, Kotlin may not be able to provide null safety guarantees when interoperating with Java. However, Java code with nullability annotation (e.g. JSR-305, Eclipse, SpotBugs) will show the correct type in Kotlin

```
1
2  import javax.annotation.Nonnull;
3  import javax.annotation.Nullable;
4
5  public class NullAnnotations {
6
7      /**
8       * Returns the literal String "Hello, World!"
9       * Return type in Kotlin is `String` (Marked with JSR-305 annotation).
10     */
11     @Nonnull
12     public String nonnullString() {
13         return "Hello, World!";
14     }
15
16
17     /**
18      * Returns null
19      * Return type in Kotlin is `String?`
20     */
21     @Nullable
22     public String nullableString() {
23         return null;
24     }
25
26     /**
27      * Returns the literal String "Hello"
28      * Return type in Kotlin is `String!` (May or may not be nullable)
29     */
30     public String platformType() {
31         return "Hello";
32     }
33 }
```

When the Kotlin compiler is unable to infer the nullability of a type given nullability annotations, the type will be marked as a platform type[2]. Platform types notify the programmer that a given type may be null, however null safety will not be strictly enforced, leading to the one and only scenario in which Kotlin may encounter an unexpected `NullPointerException`.

2.3 Coroutines

Kotlin 1.1 introduced a new method of creating non-blocking code: coroutines.[4] Kotlin's coroutines allows users to write code in an imperative, synchronous-looking fashion, while simultaneously leveraging asynchronous processing, and non-blocking IO.

[2]Platform types are denoted as `T!`

```kotlin
import kotlinx.coroutines.experimental.*

fun main(args: Array<String>) {
    launch {
      // suspend while asynchronously reading
      val bytesRead = inChannel.aRead(buf)
      // we only get to this line when reading completes
          ...
          ...
      process(buf, bytesRead)
      // suspend while asynchronously writing
      outChannel.aWrite(buf)
      // we only get to this line when writing completes
          ...
          ...
      outFile.close()
    }
}
```

Listing 3: Example of Kotlin Coroutines

Kotlin coroutines are designed to be much more lightweight than threads, while providing many of the same benefits.

2.3.1 How Coroutines Work

Kotlin coroutines provide functionality for dealing with computations whose result may not be immediately available. If the value of a computation in a coroutine is not ready, the coroutine can concurrently "wait" for the operation to complete, without blocking the thread.

In practice, coroutines may be applied anywhere that threads are used in a codebase. The added functionality for managing asynchronous computations can make asynchronous and concurrent code dramatically easier to understand and to maintain.

2.4 Properties and Delegation

A very common pattern in both Java and C++ code bases is that of encapsulating fields using getters and setters. While encapsulation in general is good, the way that it is handled in both of these languages leads to unnecessarily verbose boilerplate.

```java
/**
 * A class encapsulating two fields, with implementations of toString(), hashCode(), and equals()
*/
public class JavaPojo {
    private final String name;
    private int age;

    public JavaPojo(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return this.name;
    }

    public int getAge() {
        return this.age;
    }

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

    @Override
    public String toString() {
        return "JavaPojo(name=" + this.name + ",age=" + this.age + ")";
    }

    @Override
    public int hashCode() {
        return 31 * this.age + this.name.hashCode();
    }

    @Override
    public boolean equals(Object other) {
        if(this == other) {
            return true;
        }
        if(other instanceof JavaPojo) {
            JavaPojo obj = (JavaPojo) other;
            return obj.name.equals(this.name) && obj.age == this.age;
        }else {
            return false;
        }
    }
}
```

Listing 4: A Plain Old Java Object (POJO) with encapsulation

2.4.1 Data Classes

Kotlin provides a method of implementing not only getters and setters, but also implementations of toString(), hashCode(), and equals() on classes. These classes, denoted by the `data` modifier, act identically to equivalent POJOs, but at a fraction of the code.

```
1
2 /**
3  * A Kotlin data class encapsulating two fields
4  * [name] is an immutable property of type String (Cannot be reassigned)
5  * [age] is a mutable property of type Int (Can be reassigned)
6 */
7 data class Person(val name: String, var age: Int)
```

Listing 5: A Kotlin data class

2.4.2 Properties

In the example of the data class, there are no explicitly defined getters or setters for either one of the member fields the class encapsulates. Kotlin utilizes a special syntax known as property reference syntax to represent getters and setters.

```
1
2 /**
3 * Wrapper class around a Talon SRX
4 */
5 class CustomSRX(id: Int) : TalonSRX(id) {
6   /**
7   * Property around a velocity control loop on a talon
8   */
9   var speed: Double
10    get() = this.getSelectedSensorVelocity(0)
11    set(value) {
12      this.set(ControlMode.Velocity, value)
13    }
14 }
15
16 fun main(args: Array<String>) {
17   val customSRX = CustomSRX(1)
18   customSRX.speed = 150 // Sets the closed loop set point to 150 sensor units/100ms
19   ...
20
21   val velocity = customSRX.speed // Calls the custom getter (Gets the velocity from
         the sensor)
22 }
```

Listing 6: Properties in Action

Though it looks as though there is no encapsulation of the accesses, Kotlin implicitly calls encapsulating functions for each property.[3]

2.4.3 Property Delegation

Kotlin also provides a method of abstracting common initialization patterns behind Property Delegates. Delegates classes that contain abstracted logic for getting and setting the value of a property, denoted by the `by` keyword.

[3] A Kotlin data class being utilized in Java uses idiomatic getters and setters

```kotlin

fun main(args: Array<String>) {
    // [str] is abstracted behind the Lazy delegate.
    val str by lazy {
        // This code will be run the first time the property is accessed,
            afterwards the returned value will be given immediately
        Thread.sleep(5000) // Simulate expensive computation
        "Hello, World!"
    }

    println(str) // This operation will be slowed by 5 seconds as the value is
        computed for the first time
    println(str) // This operation will be immediate. The value returned the first
        time is given again
}
```

Listing 7: The Kotlin Lazy delegate

2.5 Extension Functions

In many Java projects, the need to provide custom functionality manifests itself by way of utility classes. Such classes typically contain functions that take the type they wish to operate on as the first parameter, and any other required operands as extra parameters.

```java

public class StringUtils {
    /**
     * Capitalizes the first letter of a String
     * Usage: StringUtils.capitalize("hello")
    */
    public static String capitalize(String str) {
        if(str.isEmpty()) {
            return "";
        }

        char[] arr = str.toCharArray();
        if(Character.isLowerCase(arr[0])) {
            return str.substring(0, 1).toUpperCase() + str.substring(1);
        }else {
            return str;
        }
    }
}
```

Listing 8: Example utility class

Kotlin allows for the use of a special syntax to represent such operations: extension functions. Extension functions allow for custom functionality to be implemented for a class, and they are called as though they are a member function of the class.

```kotlin
/**
 * Capitalizes the first letter of a String
 * Usage: "hello".capitalize()
*/
fun String.capitalize(): String {
    return if (isNotEmpty() && this[0].isLowerCase()) {
        substring(0, 1).toUpperCase() + substring(1)
    } else {
        this
    }
}
```

Listing 9: Extension functions in Kotlin

3 Detriments of Using Kotlin

3.1 Official Support

As was stated, the only languages that are officially support for use in FRC are Java, C++, and LabVIEW. If official support is needed at competition by CSAs, using Kotlin decreases the likelihood that they will be able to help.

3.2 Resources

As Kotlin is less commonly used for FRC than other languages, there will be less code resources written in Kotlin specifically.

4 How to Switch

In order to use Kotlin, teams will need to include more information in the build.gradle file used by GradleRIO. GradleRIO provides an official quickstart for Kotlin located at https://github.com/wpilibsuite/GradleRIO/tree/backtrack2019/examples/kotlin

5 Resources

Below are listed some resources for learning Kotlin

- Kotlin Koans — Resource to learn the Kotlin syntax
  https://kotlinlang.org/docs/tutorials/koans.html
- Kotlin Documentation — Contains the documentation for the Kotlin standard library
  https://kotlinlang.org/docs/reference/
- Kotlin Coroutines guide — Contains a guide to using coroutines effectively
  https://github.com/Kotlin/kotlinx.coroutines/blob/master/coroutines-guide.md
- Comparison to Java — Contains an in depth comparison to Java
  https://kotlinlang.org/docs/reference/comparison-to-java.html

6 Conclusion

By making the switch to Kotlin, a team will be able to reduce boilerplate, write more understandable code, and utilize asynchronous and concurrent functionality. Despite not being officially supported, Kotlin is a strong choice worth considering for teams willing to make the switch.

References

[1] *Welcome - Kotlin*. English. Jetbrains. URL: https://web.archive.org/web/20110812111559/http://confluence.jetbrains.net/display/Kotlin/Welcome.

[2] *Null Safety - Kotlin Programming Language*. English. Jetbrains. URL: http://kotlinlang.org/docs/reference/null-safety.html.

[3] *Calling Java code from Kotlin*. English. Jetbrains. URL: https://kotlinlang.org/docs/reference/java-interop.html.

[4] Roman Belov. *Kotlin 1.1 Released with JavaScript Support, Coroutines and more*. English. Jetbrains. Mar. 2017. URL: https://blog.jetbrains.com/kotlin/2017/03/kotlin-1-1/.