Kotlin 1.0

Programming Language for JVM and Android

Costruiamo un treno in Kotlin <。。_。。\.[♛☺☺☺].[♿☺☺_]

Francesco Vasco

Light Streamer® Real-Time Messaging

Linguaggi JVM

Kotlin [edit]

```
fun main(args: Array<String>) {
    println("Hello world!")
}
```

Scala [edit]

```
println("Hello world!")
```

Groovy [edit]

```
println "Hello world!"
```

Gosu [edit]

```
print("Hello world!")
```

Java [edit]

```
public class HelloWorld
{
 public static void main(String[] args)
 {
  System.out.println("Hello world!");
 }
}
```

C# [edit]

Works with: Mono version 1.2
Works with: Visual C# version 2003

```
namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("Hello world!");
        }
    }
}
```

http://rosettacode.org/wiki/Hello_world/Text

Concise, simple and very easy to read (and write)

```
package hello

fun main(args: Array<String>) {
    println("Hello World!")
}
```

Optional package header

Package-level function, which returns Unit and takes an Array of strings as a parameter

Have you noticed? Semicolons are optional

Concise

Drastically reduce the amount of boilerplate code you need to write.

Safe

Avoid entire classes of errors such as null pointer exceptions.

Versatile

Build server-side applications, Android apps or frontend code running in the browser.

Interoperable

Leverage existing frameworks and libraries of the JVM with 100% Java Interoperability.

Tooling

Command Line Compiler or First Class IDE Support. Freedom to choose.

Motivazioni: luglio 2011

"We've looked at all of the existing JVM languages, and none of them meet our needs. Scala has the right features, but its most obvious deficiency is very slow compilation," said Dmitry Jemerov, JetBrains development lead, on Friday. "Other languages don't meet some of our requirements in terms of the feature set.

JetBrains wants the object-oriented Kotlin language to be safer than Java, statically checking for pitfalls such as null pointer dereference, and more concise than Java. Another goal is to make it simpler than its "most mature competitor," Scala.

http://www.infoworld.com/article/2622405/java/jetbrains-readies-jvm-based-language.html

The next thing is also fairly straightforward: we expect Kotlin to drive the sales of IntelliJ IDEA. We're working on a new language, but we do not plan to replace the entire ecosystem of libraries that have been built for the JVM. So you're likely to keep using Spring and Hibernate, or other similar frameworks, in your projects built with Kotlin.

https://blog.jetbrains.com/kotlin/2011/08/why-jetbrains-needs-kotlin/

Confronto con Java / 1

Some Java issues addressed in Kotlin

Kotlin fixes a series of issues that Java suffers from

- — Null references are controlled by the type system.
- — No raw types
- — Arrays in Kotlin are invariant
- — Kotlin has proper function types, as opposed to Java's SAM-conversions
- — Use-site variance without wildcards
- — Kotlin does not have checked exceptions

What Java has that Kotlin does not

- — Checked exceptions
- — Primitive types that are not classes
- — Static members
- — Non-private fields
- — Wildcard-types

Confronto con Java / 2

What Kotlin has that Java does not

- — Lambda expressions + Inline functions = performant custom control structures
- — Extension functions
- — Null-safety
- — Smart casts
- — String templates
- — Properties
- — Primary constructors
- — First-class delegation
- — Type inference for variable and property types
- — Singletons
- — Declaration-site variance & Type projections
- — Range expressions
- — Operator overloading
- — Companion objects
- — Data classes
- — Separate interfaces for read-only and mutable collections

http://kotlinlang.org/docs/reference/comparison-to-java.html

Strumenti

- IntelliJ IDEA / Android Studio / Eclipse / Netbeans
- Dokka (Javadoc con Markdown)
- Gradle / Maven / Ant / linea di comando
- Integrazione OSGi
- 100% interoperabile con l'ecosistema Java (Java EE, Spring, Hibernate, Guava, *Jackson*…)
- Supporto Android (JVM 1.6)
- Supporto di primo livello in Gradle 3
- Supporto REPL (Read Evaluate Print Loop)

Kotlin

- Kotlin 1.0
- JVM 1.6-1.8

- Package k

Java

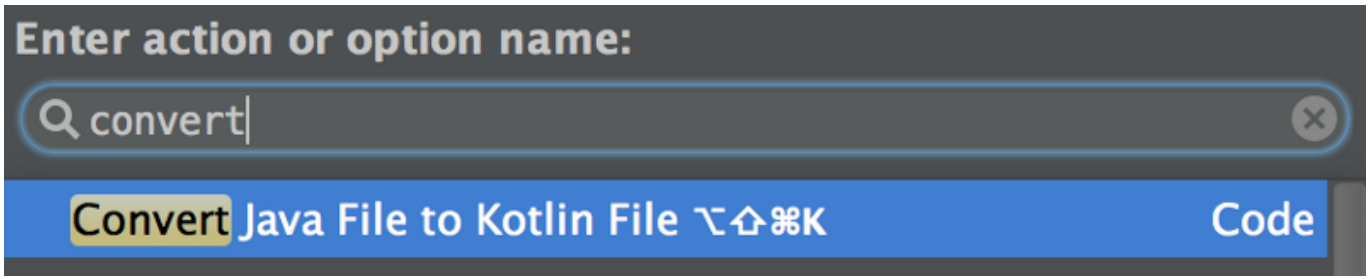
- Java 8.0
- JVM 1.8

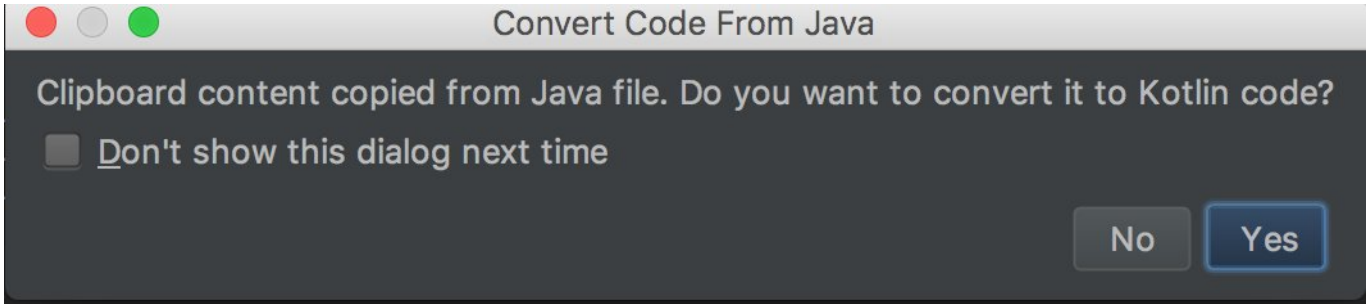
- Package j

Enum

```
public enum TicketType {
    REGULAR, REDUCED, FREE
}
```

```
enum class TicketType {
    REGULAR, REDUCED, FREE
}
```





Print ticket type

```java
import k.TicketType;

public final class PrintTicketTypeJava {
    public static void main(final String... args) {
        System.out.println("Ticket types:");
        for (final TicketType ticketType : TicketType.values()) {
            System.out.println(" - " + ticketType);
        }
    }
}
```

```
import j.TicketType

fun main(vararg args: String) {
    println("Ticket types:")
    for (ticketType in TicketType.values()) {
        println(" - $ticketType")
    }
}
```

Utility library

```java
public final class Util {

    private static final AtomicInteger idGenerator = new AtomicInteger();

    private Util() {
    }

    public static Object generateId() {
        return String.valueOf(idGenerator.incrementAndGet());
    }
}
```

```kotlin
private val idGenerator = AtomicInteger()

fun generateId(): Any = idGenerator.incrementAndGet().toString()
```

Java bean

```java
public class Passenger {

    private final Object ticketId;

    private final k.TicketType ticketType;

    public Passenger() {
        this(UtilKt.generateId(), TicketType.REGULAR);
    }

    public Passenger(Object ticketId, TicketType ticketType) {
        Objects.requireNonNull(ticketId);
        Objects.requireNonNull(ticketType);
        this.ticketId = ticketId;
        this.ticketType = ticketType;
    }
}
```

Java bean qualche click dopo

```java
public class Passenger {

    private final Object ticketId;

    private final k.TicketType ticketType;

    public Passenger() {
        this(UtilKt.generateId(), TicketType.REGULAR);
    }

    public Passenger(Object ticketId, TicketType ticketType) {
        Objects.requireNonNull(ticketId);
        Objects.requireNonNull(ticketType);
        this.ticketId = ticketId;
        this.ticketType = ticketType;
    }

    public Object getTicketId() {
        return ticketId;
    }

    public TicketType getTicketType() {
        return ticketType;
    }

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

        Passenger passenger = (Passenger) o;

        if (!ticketId.equals(passenger.ticketId)) return false;
        return ticketType == passenger.ticketType;

    }

    @Override
    public int hashCode() {
        int result = ticketId.hashCode();
        result = 31 * result + ticketType.hashCode();
        return result;
    }
}
```

Mario Fusco @mariofusco · 23 ott

Any piece of code that you're forced to duplicate or that can be automatically generated by a IDE is a missing language abstraction feature

Java bean toString

```java
public String toString() {
    final String string;
    switch (ticketType) {
        case REGULAR:
            string = "☺";
            break;
        case REDUCED:
            string = "♿";
            break;
        case FREE:
            string = "♛";
            break;
        default:
            throw new IllegalStateException("Unsupported ticket type " + ticketType);
    }
    return string;
}
```

Caso impossibile

Java bean in Kotlin

```
data class Passenger(val ticketId: Any = generateId(),
                     val ticketType: TicketType = TicketType.REGULAR) {

    override fun toString(): String {
        val string: String
        when (ticketType) {
            TicketType.REGULAR -> string = "☺"
            TicketType.REDUCED -> string = "♿"
            TicketType.FREE -> string = "♛"
        // else -> throw IllegalStateException("Unsupported ticket type " + ticketType)
        }
        return string
    }

}
```

```
val defaultPassenger = Passenger()
val freePassenger = defaultPassenger.copy(ticketType = TicketType.FREE)
```

copy(ticketId: Any = ..., ticketType: TicketType = ...) Passenger

Dot, space and some other keys will also close this lookup and be inserted into editor >>

Passenger in Kotlin

```
data class Passenger(val ticketId: Any = generateId(),
                     val ticketType: TicketType = TicketType.REGULAR) {

    override fun toString(): String =
        when (ticketType) {
            TicketType.REGULAR -> "☺"
            TicketType.REDUCED -> "♿"
            TicketType.FREE -> "♛"
        }

}
```

Seat

```java
public final class Seat {
    private Optional<k.Passenger> optionalPassenger = Optional.empty();

    @Override
    public String toString() {
        return optionalPassenger.map((p) -> p.toString()).orElse( other: "_");
    }

    public Optional<k.Passenger> getOptionalPassenger() {
        return optionalPassenger;
    }

    public void setOptionalPassenger(Optional<Passenger> optionalPassenger) {
        Objects.requireNonNull(optionalPassenger);
        this.optionalPassenger = optionalPassenger;
    }
}
```

```kotlin
class Seat(var passenger: Passenger? = null) {
    override fun toString() = passenger?.toString() ?: "_"
}
```

Railroad car

```java
public interface RailroadCar {
    int getWeight();
}
```

```kotlin
interface RailroadCar {
    val weight: Int
}
```

Locomotive Java

```java
public final class Locomotive implements k.RailroadCar {

    @Override
    public int getWeight() { return 8; }

    public int getTowableMass() { return 20; }

    @Override
    public String toString() { return "<.._..\\"; }
}
```

Locomotive Kotlin

```
class Locomotive : k.RailroadCar {

    override val weight: Int
        get() = 8

    val towableMass: Int
        get() = 20

    override fun toString() = """<o_o\"""
}
```

Coach Java

```java
public final class Coach implements k.RailroadCar {

    private final List<Seat> seats;

    public Coach(final int availableSeats) {
        if (availableSeats < 1) {
            throw new IllegalArgumentException("availableSeats " + availableSeats);
        }
        seats = IntStream.range(0, availableSeats)
                .mapToObj((i) -> new Seat())
                .collect(Collectors.toList());
    }

    @Override
    public int getWeight() { return 5; }

    @Override
    public String toString() {
        return "[" +
                seats.stream()
                        .map(Object::toString)
                        .collect(Collectors.joining(""))
                + "]";
    }
}
```

Coach Kotlin

```
class Coach(availableSeats: Int) : k.RailroadCar {

    init {
        require(availableSeats > 0) { "availableSeats $availableSeats" }
    }

    val seats: List<Seat> = (1..availableSeats).map { Seat() }

    override val weight: Int
        get() = 5

    override fun toString(): String =
            seats.joinToString(separator = "", prefix = "[", postfix = "]")
}
```

Train Java

```java
public class Train {

    private List<RailroadCar> railroadCars;
    private boolean running = false;

    public Train(final List<RailroadCar> composition) {
        Objects.requireNonNull(composition);
        railroadCars = composition;
    }

    public int getTotalWeight() { return railroadCars.stream().mapToInt(RailroadCar::getWeight).sum(); }

    public int getTotalTowableMass() {
        return railroadCars.stream()
                .filter((c) -> c instanceof k.Locomotive)
                .map((c) -> (Locomotive) c)
                .mapToInt(Locomotive::getTowableMass)
                .sum();
    }

    public void start() {
        if (getTotalTowableMass() < getTotalWeight()) {
            throw new IllegalStateException("Too many coaches");
        }
        running = true;
    }
```

Train Kotlin

```
class Train(val railroadCars: List<RailroadCar>) {

    var running = false
        private set

    val totalWeight: Int
        get() = railroadCars.sumBy { it.weight }

    val totalTowableMass: Int
        get() {
            var total = 0
            for (car in railroadCars)
                if (car is Locomotive)
                    total += car.towableMass
            return total
        }
    //railroadCars
    // .filterIsInstance(Locomotive::class.java)
    // .sumBy { it.towableMass }

    fun start() {
        check(totalTowableMass >= totalWeight) { "Too many coaches" }
        running  = true
    }

    fun stop() { running  = false }

    override fun toString(): String = railroadCars.joinToString(".")
}
```

Read-only list

Train check

```java
public interface TrainCheck {

    Optional<String> searchIssue(Train train);
}
```

```kotlin
interface TrainCheck {

    fun searchIssue(train: Train): String?
}
```

Train size check Java

```java
public class TrainSizeCheck implements TrainCheck {

    private int minSize, maxSize;

    public TrainSizeCheck(int minSize, int maxSize) {
        this.minSize = minSize;
        this.maxSize = maxSize;
    }

    @Override
    public Optional<String> searchIssue(Train train) {
        final int trainSize = train.getRailroadCars().size();
        if (trainSize < minSize)
            return Optional.of("Train too small");
        else if (trainSize > maxSize)
            return Optional.of("Train too big");
        else
            return Optional.empty();
    }
}
```

Train size check Kotlin

```kotlin
class TrainSizeCheck(private val validTrainSize: IntRange) : TrainCheck {

    override fun searchIssue(train: Train): String? {
        val trainSize = train.railroadCars.size
        return when {
            trainSize in validTrainSize -> null
            trainSize < validTrainSize.first -> "Train too small"
            else -> "Train too big"
        }
    }

}
```

Standard check list Java

```java
public final class TrainCheckList {

    public static final TrainCheckList INSTANCE = new TrainCheckList();
    private List<TrainCheck> standardCheckList;

    public synchronized List<TrainCheck> getStandardCheckList() {
        if (standardCheckList == null) {
            standardCheckList = Util.loadStandardCheckList();
        }
        return standardCheckList;
    }

    public List<String> execute(final Train train) {
        return getStandardCheckList().stream()
                .map((check) -> check.searchIssue(train))
                .filter(Optional::isPresent)
                .map(Optional::get)
                .collect(Collectors.toList());
    }

    private TrainCheckList() {...}
}
```

Standard check list Kotlin

```kotlin
object TrainCheckList {

    val standardCheckList by lazy { loadStandardCheckList() }

    fun execute(train: Train) =
            standardCheckList
                    .map { it.searchIssue(train) }
                    .filterNotNull()

}
```

Train extension functions

```
fun checkAndStart(train: Train) {
    require(TrainCheckList.execute(train).isEmpty()) { "Check list failed" }
    train.start()
}

 ...
checkAndStart(train)
 ...
```

```
fun Train.checkAndStart() {
    require(TrainCheckList.execute(this).isEmpty()) { "Check list failed" }
    start()
}


 ...
train.checkAndStart()
 ...
```

Kotlin 1.1

- Coroutines
- Type aliases
- Bound callable references
- Local delegated properties & Inline properties
- Relaxed rules for sealed classes and data classes
- JSR-223 (javax.script API)
- Java 7/8/9 support
- JavaScript

https://blog.jetbrains.com/kotlin/2016/07/first-glimpse-of-kotlin-1-1-coroutines-type-aliases-and-more/

Riferimenti

- Kotlin Programmin Language: http://kotlinlang.org/
- Try Kotlin: http://try.kotlinlang.org/
- Links: http://kotlin.link/
- Kotlin Evolution and Enhancement Process: https://github.com/Kotlin/KEEP

Grazie

```
val locomotives: List<Locomotive> = listOf(Locomotive())

val choaches: MutableList<Coach> = ArrayList()
choaches.add(makeCoachWithTickets(FREE, REGULAR, REGULAR, REGULAR))
choaches += makeCoachWithTickets(REDUCED, REGULAR, REGULAR, null)

val railroadCars /* : List<RailroadCar> */ = locomotives + choaches
val train = Train(railroadCars)
println("Don't miss the train   $train")
train.checkAndStart()
```

```
/usr/lib/jvm/default-java/bin/java ...
Don't miss the train     <。。_。。\.[♛☺☺☺].[♿☺☺_]

Process finished with exit code 0
```