UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

vs

GRUPO 22

1991	1995	1996	2014	2019
Sun Microsystems lanza “OAK”	Versión Alpha de JAVA	JDK 1.0	JAVA versión 8	JAVA versión 12

2011

JetBrains lanza la primer versión

2012

Comienza a ser de software libre

2016

Versión 1.0

2017

Soporte de Google

BNF

Clases
declaration:
classDeclaration
| objectDeclaration
| functionDeclaration
| propertyDeclaration
| typeAlias

classDeclaration:
[modifiers]
('class' | (['fun' {NL}] 'interface'))
{NL}
simpleIdentifier
[{NL} typeParameters]
[{NL} primaryConstructor]
[{NL} ':' {NL}
delegationSpecifiers]
[{NL} typeConstraints]
[({NL} classBody) | ({NL}
enumClassBody)]

Tokens
KotlinToken
ShebangLine
| DelimitedComment
| LineComment
| WS
| NL
| RESERVED
| DOT
| COMMA
| LPAREN
| RPAREN
| LSQUARE

Clases
<class declaration> ::= <class modifiers>? class
<identifier> <super>? <interfaces>? <class body>
<class modifiers> ::= <class modifier> | <class modifiers>
<class modifier>
<class modifier> ::= public | abstract | final

Tokens
<package name> ::= <identifier> | <package name> .
<identifier>
<type name> ::= <identifier> | <package name> .
<identifier>
<simple type name> ::= <identifier>
<expression name> ::= <identifier> | <ambiguous name> .
<identifier>

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Java Codes > hworld.java > hworld

```java
final class hworld {
    Run | Debug
    public static void main(String args[]) {
        String n = args.length > 0 ? args[0] : "";
        System.out.println(String.format(format:"Hello world %s!", n));
    }
}
```

Java

otlin Codes > hworld.kt

```kotlin
fun main(args: Array<String>) {
    var name = if(args.size > 0) args[0] else ""
    println("Hello world ${name}!")
}
```

Kotlin

Java POJO Kotlin

```java
class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    // toString…

    // hashCode…

    // equals…

    // copy…
}
```

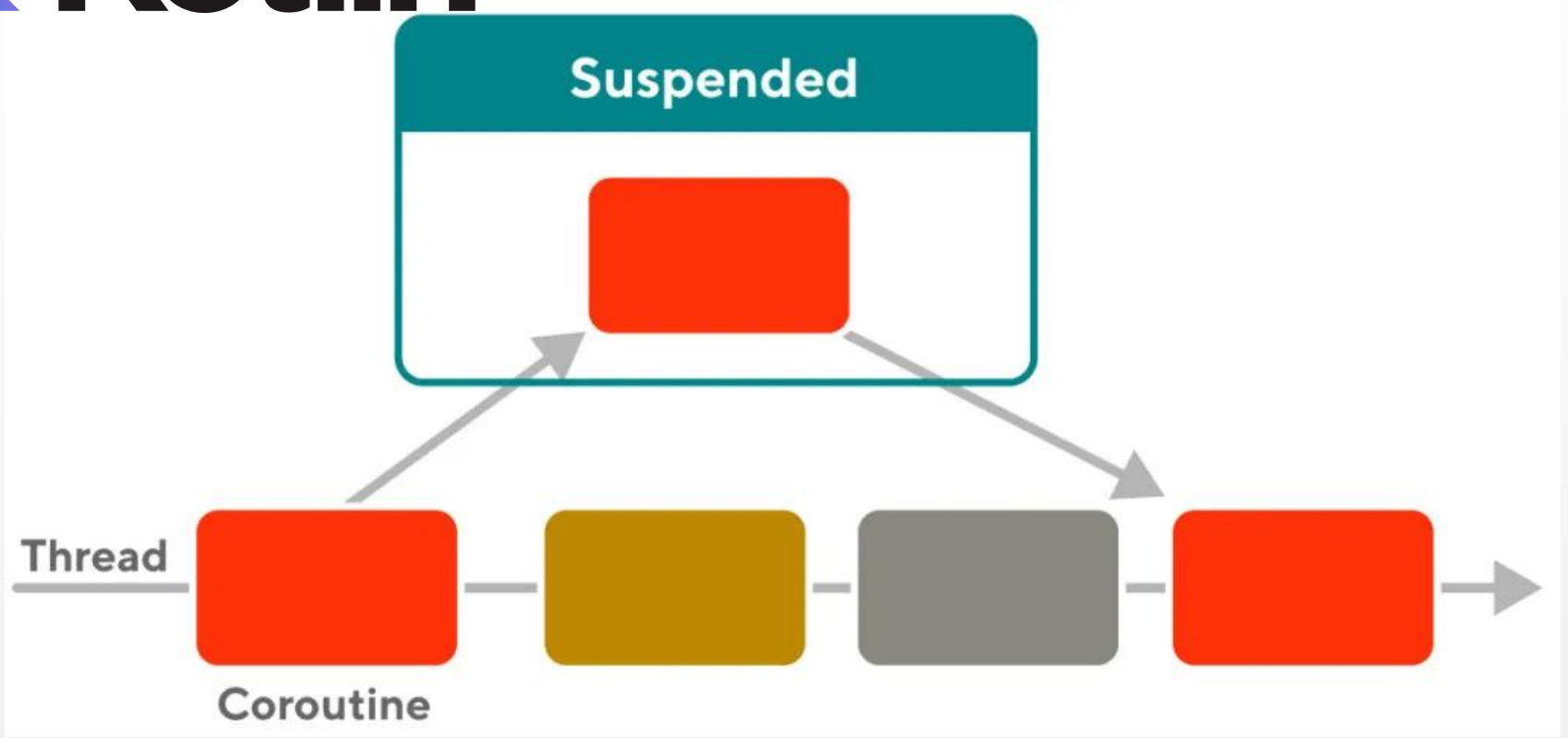
```kotlin
data class Person(val name: String)
```

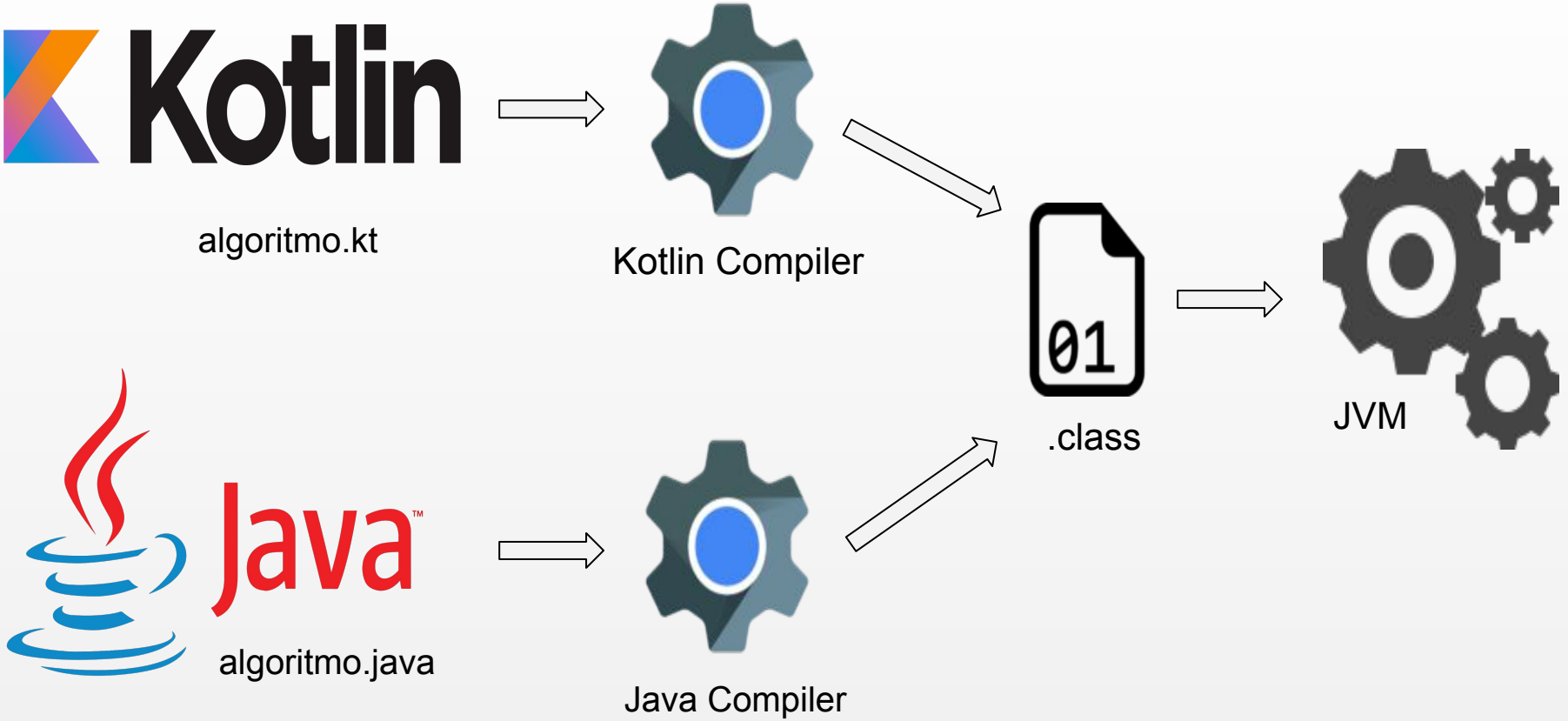
Java Code Kotlin

```java
public void createAndPrintPerson() {
    String name = "Pieter";
    Person person = new Person(name);

    printName(person.getName());
    // Prints: Pieter Otten
}
```

```kotlin
fun createAndPrintPerson() {
    val name = "Pieter"
    val person = Person(name)

    printName(person.name)
    // Prints: Pieter Otten
}
```


Kotlin


Suspended
Thread
Coroutine


Kotlin
algoritmo.kt
Kotlin Compiler
01
.class
JVM
Java
algoritmo.java
Java Compiler

VS

Objetivo

Comparar el rendimiento de un algoritmo.

Se decidió llevar a cabo un algoritmo que ordena una colección de 100000 elementos para así ver su tiempo de ejecución.


```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Sort {
    Run | Debug
    public static void main(String[] args) {
        // Crear una lista de ejemplo con 100000 elementos no ordenados
        List<Integer> lista = new ArrayList<>();
        for (int i = 0; i < 100000; i++) {
            lista.add((int) (Math.random() * 1000)); // Agregar números aleatorios
        }

        // Ordenar la lista
        Collections.sort(lista);

        // Imprimir la lista ordenada
        System.out.println(x:"Lista ordenada:");
        for (Integer elemento : lista) {
            System.out.print(elemento + " ");
        }
    }
```

mem: 20MB
Compile: 115 millisec
Exec: 1.14 seconds

Kotlin

```kotlin
fun main() {
    // Crear una lista de ejemplo con 100000 elementos no ordenados
    val lista = mutableListOf<Int>()
    repeat(100000) {
        lista.add((0..999).random()) // Agregar números aleatorios
    }

    // Ordenar la lista
    lista.sort()

    // Imprimir la lista ordenada
    println("Lista ordenada:")
    lista.forEach { elemento ->
        print("$elemento ")
    }
}
```

mem: 18MB
Compile: 387 millisec
Exec: 5.583 seconds

Kotlin

mem: 18MB
Compile: 387 millisec
Exec: 5.583 seconds

VS

Java

mem: 20MB
Compile: 115 millisec
Exec: 1.14 seconds

VS

Objetivo

Comparar el rendimiento de un algoritmo que da la cantidad de dígitos de Pi según lo indique la entrada. En este caso se pidió los 5000 primero dígitos.

```java
import java.math.BigInteger;

final class piDigits {
  static final int L = 10;

  Run | Debug
  public static void main (String args[]) {
    int n = 5000;
    int j = 0;

    PiDigitSpigot digits = new PiDigitSpigot();

    while (n > 0) {
      if (n >= L) {
        for (int i = 0; i < L; i++) System.out.print(digits.next());
        j += L;
      } else {
        for (int i = 0; i < n; i++) System.out.print(digits.next());
        for (int i = n; i < L; i++) System.out.print(s:" ");
        j += n;
      }
```

Mem: 338.9MB
Compile: 491millisec
Exec: 2.731 seconds

Kotlin

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

```kotlin
import java.math.BigInteger;

fun main(args: Array<String>) {
    val L = 10

    // var n = if(args.size > 0) args[0].toInt() else 27
    var n = 5000
    var j = 0

    val digits = PiDigitSpigot()

    while (n > 0) {
        if (n >= L) {
            for (i in 0..L - 1) print(digits.next())
            j += L
        } else {
            for (i in 0..n - 1) print(digits.next())
            for (i in n..L - 1) print(" ")
            j += n
        }
        print("\t:")
        println(j)
        n -= L
```

Mem: 235.7MB
Compile: 550 millisec
Exec: 8.189 seconds

Kotlin

Mem: 235.7MB
Compile: 550 millisec
Exec: 8.189 seconds

VS

Java

Mem: 338.9MB
Compile: 491millisec
Exec: 2.731 seconds

VS

Conclusión

Java resultó ser más rápido a la hora de su ejecución. Sin embargo Kotlin suele ser recomendado ya que es más conciso y seguro. Además puede consumir los recursos, librerías y hasta la máquina virtual de Java. Por lo que ambos lenguajes pueden convivir en un mismo proyecto.

Sin embargo, para otros tipos de desarrollo, la elección entre Java y Kotlin puede depender de varios factores.

Preguntas

¿Pueden sobrevivir ambos en un mismo proyecto?

¿Cuál utiliza JVM?

¿Cuál es más rápido?

¿Cuál consume más recursos?

¿Cuál es más seguro?