Java 9-11

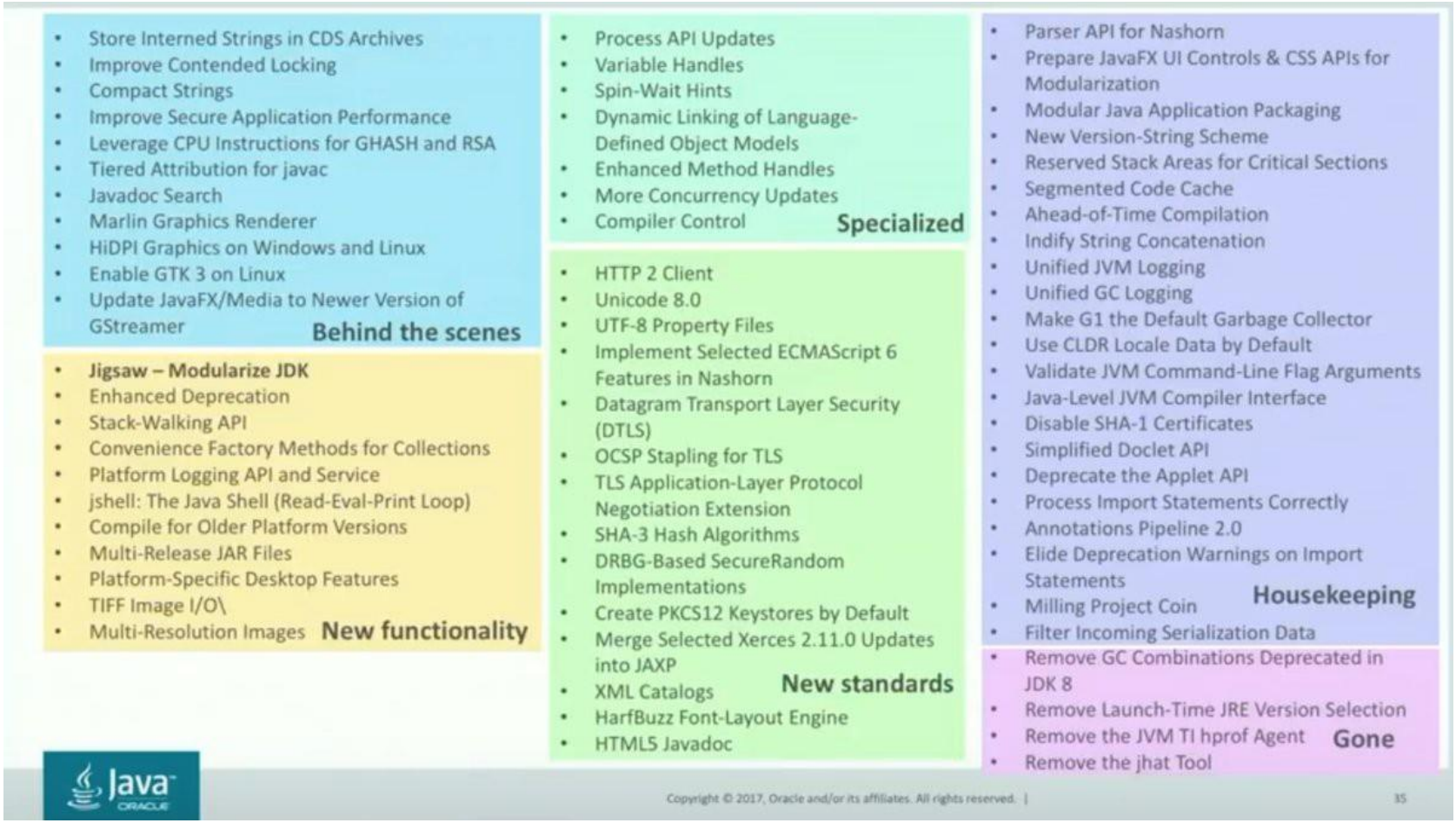
Profesor:
Máster Carlos Carranza Blanco
Programación III
Ingeniería del Software

8/8/2023

Nuevas Características Java 9 (2017)

- *Modularidad (proyecto Jigsaw).*
- *Métodos factoría para colecciones.*
- *Mejoras en la clase Optional.*
- *Mejoras en la API de streams.*
- *jlink para generar runtimes mínimos*
- *Strings compactos.*
- *Recolector de basura G1 por defecto.*
- *Métodos privados en interfaces.*
- *Archivos Jar multiversión.*
- *Nuevo modelo de publicación* ***(Cada 6 meses)***.

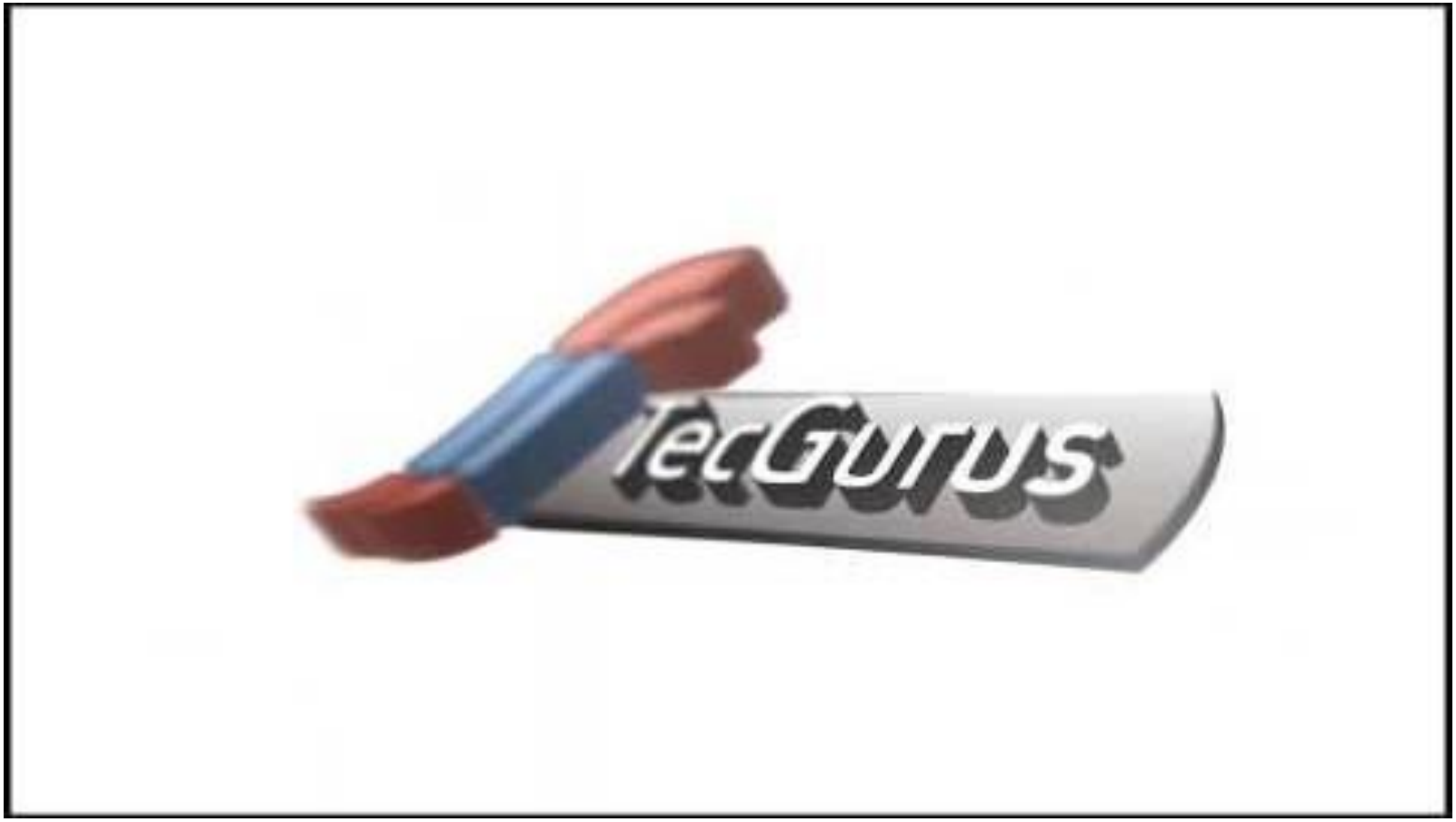
Java 9 ...


Store Interned Strings in CDS Archives
Improve Contended Locking
Compact Strings
Improve Secure Application Performance
Leverage CPU Instructions for GHASH and RSA
Tiered Attribution for javac
Javadoc Search
Marlin Graphics Renderer
HiDPI Graphics on Windows and Linux
Enable GTK 3 on Linux
Update JavaFX/Media to Newer Version of GStreamer
Behind the scenes
Jigsaw – Modularize JDK
Enhanced Deprecation
Stack-Walking API
Convenience Factory Methods for Collections
Platform Logging API and Service
jshell: The Java Shell (Read-Eval-Print Loop)
Compile for Older Platform Versions
Multi-Release JAR Files
Platform-Specific Desktop Features
TIFF Image I/O\
Multi-Resolution Images
New functionality
Process API Updates
Variable Handles
Spin-Wait Hints
Dynamic Linking of Language-Defined Object Models
Enhanced Method Handles
More Concurrency Updates
Compiler Control
Specialized
HTTP 2 Client
Unicode 8.0
UTF-8 Property Files
Implement Selected ECMAScript 6 Features in Nashorn
Datagram Transport Layer Security (DTLS)
OCSP Stapling for TLS
TLS Application-Layer Protocol Negotiation Extension
SHA-3 Hash Algorithms
DRBG-Based SecureRandom Implementations
Create PKCS12 Keystores by Default
Merge Selected Xerces 2.11.0 Updates into JAXP
XML Catalogs
HarfBuzz Font-Layout Engine
HTMLS Javadoc
New standards
Parser API for Nashorn
Prepare JavaFX UI Controls & CSS APIs for Modularization
Modular Java Application Packaging
New Version-String Scheme
Reserved Stack Areas for Critical Sections
Segmented Code Cache
Ahead-of-Time Compilation
Indify String Concatenation
Unified JVM Logging
Unified GC Logging
Make G1 the Default Garbage Collector
Use CLDR Locale Data by Default
Validate JVM Command-Line Flag Arguments
Java-Level JVM Compiler Interface
Disable SHA-1 Certificates
Simplified Doclet API
Deprecate the Applet API
Process Import Statements Correctly
Annotations Pipeline 2.0
Elide Deprecation Warnings on Import Statements
Milling Project Coin
Filter Incoming Serialization Data
Housekeeping
Remove GC Combinations Deprecated in JDK 8
Remove Launch-Time JRE Version Selection
Remove the JVM TI hprof Agent
Remove the jhat Tool
Gone
Java
ORACLE
Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |
35

Modularidad

- *Supone grandes mejoras como una mejor encapsulación de los paquetes, interfaces entre módulos bien definidas y dependencias explícitas que proporcionan optimización al usarse sólo los módulos que se necesitan, mayor seguridad al ser menor la superficie de ataque y configuración confiable al comprobar las dependencias al compilar o iniciarse la máquina virtual.*

Modularidad…

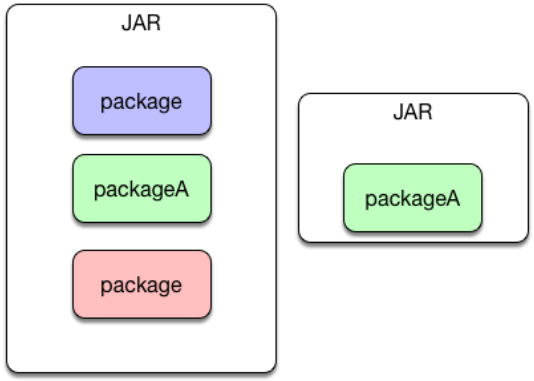


Ver hasta minuto 18

Link video

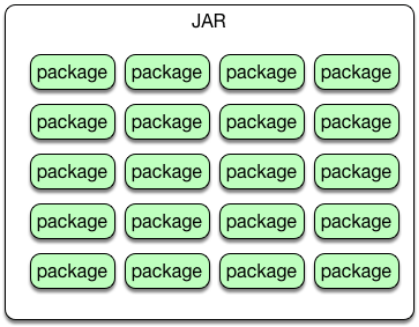
Por qué Modularidad?

- *Hasta hoy en día Java ha organizado sus clases a través del concepto de paquetes que es un concepto puramente lógico. Un conjunto de clases pertenecen a un paquete determinado. A nivel físico varios packages son ubicados en un JAR o Java Archive.*
- *En ocasiones es necesario tener más de organización y modularidad a la hora de trabajar con grupos de clases y sus dependencias. Por ejemplo clases de un mismo paquete podrían estar ubicadas en dos JARs diferentes.*

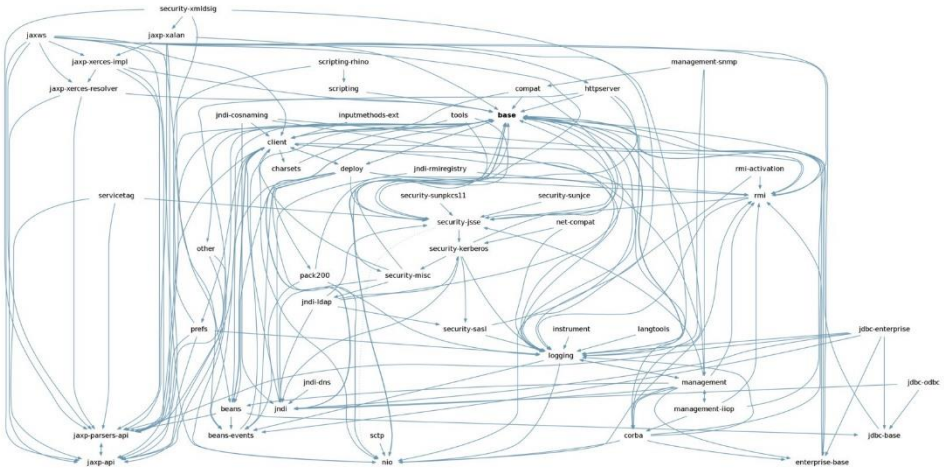


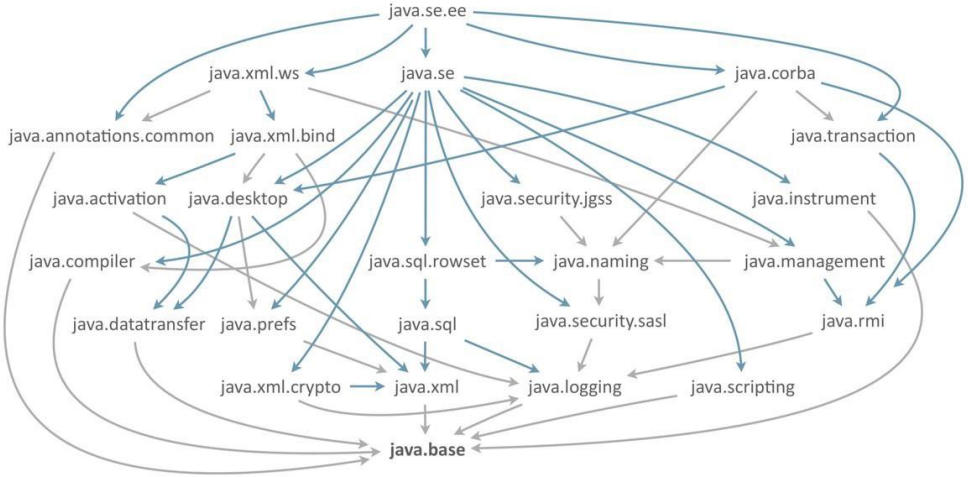
Por qué Modularidad?...

- *Algunos de los ficheros JAR a nivel de Java incluyen cientos de packages. Por lo tanto estamos ante una situación que se acerca bastante al concepto de monolito (una única pieza).*
- *Otro de los problemas que siempre han existido es como gestionar las dependencias entre un JAR y otro con los packages que están asociados. Maven siempre ha ayudado a ello , pero es cierto que es un herramienta aparte , no algo propio del lenguaje.*



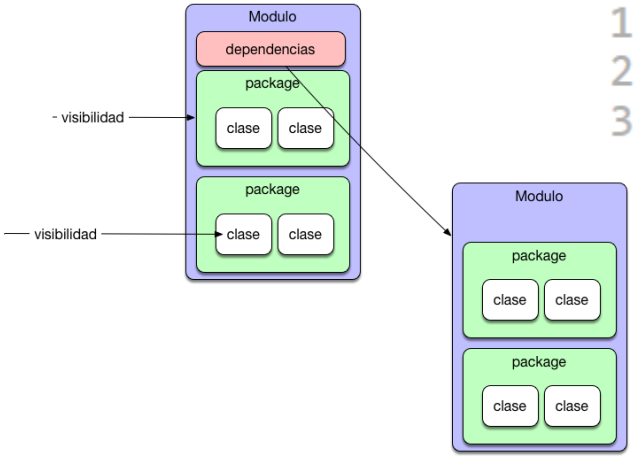
Por qué Modularidad?...





Modularidad...

- *Un módulo es un conjunto de clases que pueden contener uno o varios packages y que define las dependencias con el resto de módulos así como la visibilidad de las clases que contiene.*




```
1  module ModuloA {
2   exports com.arquitecturajava.core;
3  }
```

Ejemplo básico

Mejoras con la Modularidad

- ***Encapsulación fuerte:*** *La encapsulación se cumple durante compilación y ejecución, incluso frente a intentos de reflexión.*
- ***Configuración fiable:*** *El runtime comprueba la disponibilidad de las dependencias antes de lanzar la aplicación.*
- ***Creación de imágenes*** *que empaqueta la aplicación con una plataforma Java hecha a medida. Esto implica:*
  - *Menores requerimientos de memoria y disco (útil para microservicios y dispositivos pequeños)*
  - *Mayor seguridad, porque el código implicado es menor.*
  - *Optimización mejorada (dead-code elimination, constant folding, compression, etc.).*

Mejoras con la Modularidad…

- ***Servicios desacoplados*** *sin escaneo del classpath (las implementaciones de un interfaz se indican explícitamente).*
- ***Carga rápida de tipos:*** *El sistema sabe dónde está cada paquete sin tener que escanear el classpath.*
- ***Preserva las fronteras*** *establecidas por la arquitectura.*
- *La encapsulación fuerte implica otros beneficios, como la posibilidad de realizar pruebas aisladas de un módulo, evitar la decadencia del código al introducir dependencias accidentales, y la reducción de dependencias cuando varios equipos trabajan en paralelo.*

Descriptor de Modulo

- *Es la meta-información sobre el módulo. Contiene lo siguiente:*
  - *Nombre del módulo*
  - *Paquetes expuestos*
  - *Dependencias con otros módulos*
  - *Servicios consumidos e implementados*
- *Los descriptores se escriben en un fichero module-info.java en la raíz del fichero JAR o directorio. Este fichero se compila junto al resto de ficheros Java.*

```
1  module ejemplo {
2      requires java.util.logging;
3      exports com.ejemplo;
4  }
```

- El módulo se llama

  `ejemplo`
- Depende del paquete

  `java.util.logging`
- Y expone el paquete

  `com.ejemplo`

Descriptor de Modulo…

- *Un módulo se define con las siguientes palabras clave:*
  - ***exports… to:*** *expone un paquete, opcionalmente a un módulo concreto.*
  - ***import****: el típico import de Java.*
  - ***module:*** *Comienza la definición de un módulo.*
  - ***open:*** *Permite la reflexión en un módulo.*
  - ***opens:*** *Permite la reflexión en un paquete concreto, para alguno o todos los paquetes.*
  - ***provides…with:*** *Indica un servicio y su implementación.*
  - ***requires, static, transitive: requires*** *indica la dependencia con un módulo. Añade* ***static*** *para que sea requerido durante compilación y opcional durante la ejecución. Añade* ***transitive*** *para indicar dependencia con las dependencias del módulo requerido.*

Descriptor de Modulo...

Java

```java
// nombre del módulo. open permite la reflexión en todo el módulo
open module com.ejemplo
{
    // exporta un paquete para que otros módulos accedan a sus paquetes públicos
    exports com.apple;

    // indica una dependencia con el módulo com.orange
    requires com.orange;

    // indica una dependencia con com.banana. el 'static' hace que la dependencia
    // sea obligatoria durante compilación pero opcional durante ejecución
    requires static com.banana;

    // indica una dependencia al módulo com.berry y sus dependencias
    requires transitive com.berry;

    // permite reflexión en el módulo com.pear
    opens com.pear;

    // permite reflexión en el paquete com.lemon pero solo desde el módulo com.mango
    opens com.lemon to com.mango;

    // expone el tipo MyImplementation que implementa el servicio MyService
    provides com.service.MyService with com.consumer.MyImplementation

    // usa el servicio com.service.MyService
    uses com.service.MyService
}
```

Más allá de la modularidad

Ver hasta 1:17:15, luego del 1:30:50 al 1:36:30 y del 1:44:35 al 1:52:35

Link video

Métodos factoría para colecciones

- Aparte de definir este tipo de colecciones de una forma mucho más sencilla que hasta Java 8, las colecciones además son significativamente más eficientes.

```java
// Java 8
List<String> list = Collections.unmodifiableList(Arrays.asList("a", "b", "c"));

Set<String> set = Collections.unmodifiableSet(new HashSet<>(Arrays.asList("a", "b", "c")));

Map<String, Integer> map = new HashMap<>();
map.put("a", 1);
map.put("b", 2);
map.put("c", 3);
map = Collections.unmodifiableMap(map);

// Java 9
List<String> list = List.of("a", "b", "c");

Set<String> set = Set.of("a", "b", "c");

Map<String, Integer> map = Map.of("a", 1);
```

Collections.java

Mejoras en la clase Optional

- *Los métodos or() y ifPresentOrElse() así como stream() mejoran la experiencia de uso en esta clase que contiene o no un objeto.*
- *El método or() en caso de no contener el Optional un objeto permite proporcionar un Optional alternativo.*
- *Los métodos ifPresent() y ifPresentOrElse() permiten realizar una acción con el objeto del opcional si está presente u otra acción con un valor vacío si no está presente.*
- *El método stream() convierte el Optional en un stream de cero o un elemento.*

Mejoras en la API de streams

- *Los nuevos métodos de los streams dropWhile(), takeWhile() permiten descartar o tomar elementos del stream mientras se comprueba una condición.*
- *El método ofNullable() devuelve un stream de un elemento o vacío dependiendo de si el objeto es null o no.*
- *Los métodos iterate() permiten generar un secuencia de valores similar a un bucle for.*

jlink

- *Permite generar runtimes aprovechando la nueva modularidad del JDK con únicamente los módulos que necesite la aplicación.*
- *Esto es especialmente útil para los contenedores de Docker y los entornos cloud ya que permite generar imágenes de contenedores con un tamaño significativamente menor.*
- *Por ejemplo, una imagen de Docker basada en la distribución Alpine Linux con el JDK completo ocupa unos 360 MiB, con jlink si una aplicación solo necesita del módulo java.base se puede generar un runtime con únicamente ese módulo, con este runtime adaptado la imagen del contenedor tiene un tamaño mucho menor, en este caso de únicamente de unos 40 MiB.*

Strings compactos

- *Internamente los la clase String contiene un array de char, cada char se representa en formato con la codificación UTF-8 ocupando 16 bits o 2 bytes por cada caracter. Para cadenas en aquellos lenguajes como inglés los caracteres pueden ser representados usando un único byte.*

- *Una buena parte de la memoria ocupada en la JVM por cualquier aplicación es debido a las cadenas de modo que tiene sentido compactar aquellas cadenas en las que sea posible representándolas con un único byte por caracter.*

- *Lo mejor de todo es que esta optimización será transparente para los programadores y para las aplicaciones proporcionando una reducción en el uso de la memoria y aumento del rendimiento, también en el recolector de basura.*

Recolector de basura G1 por defecto

- *Se cambia el recolector de basura por defecto al llamado G1 optimizado para una balance adecuado entre alto rendimiento y baja latencia.*
- *Al igual que los string compactos para la mayoría de los programadores será un cambio transparente que no tenga repercusión en la forma de programar las aplicaciones.*
- *El recolector de basura G1 usa ambas estrategias la paralela y la concurrente. Usa threads concurrentes mientras la aplicación está funcionando buscando los objetos vivos y usa la estrategia paralela para realizar la recolección y compactación rápidamente manteniendo las pausas bajas.*

Ampliar concepto

Métodos privados en interfaces

- Ahora se pueden crear métodos privados en interfaces como utilidad a las implementaciones de los métodos por defecto.

```
1  package com.arquitecturajava.ejemplo2;
2
3  public interface Imprimible {
4
5
6      public default void encender() {
7
8          formatear("encendiendo el dispositivo");
9      }
10
11     public default void apagar() {
12
13         formatear("apagando el dispositivo");
14     }
15
16     private static void formatear( String mensaje) {
17
18         System.out.println("**********");
19         System.out.println(mensaje);
20         System.out.println("**********");
21     }
22
23
24     public void imprimir();
25 }
```

Ampliar concepto

Archivos Jar multiversión

- *Los desarrolladores de librerías para dar soporte a varias versiones de Java debían optar entre generar un artefacto para cada versión o un único archivo jar limitándose a usar la mínima versión soportada y sin aprovechar las nuevas capacidades de siguientes versiones. Esto es un impedimento para el uso de nuevas versiones.*

- *Con Java 9 se puede generar un único archivo jar con algunas clases para una o varias versiones de Java. Por ejemplo, en un archivo jar con las clases A, B, C y D compatibles con Java 6 el desarrollador ahora puede decidir que para la versión 9 la clase A y B sean unas optimizadas para esta versión. Esto se consigue con una estructura específica de directorios en el archivo jar, ubicándose la clase optimizada para Java 9 A en META-INF/versions/9/A.class y para Java 10 en META-INF/versions/10/A.class.*

Ampliar concepto

Enlaces complementarios

- https://picodotdev.github.io/blog-bitix/2017/09/novedades-de-java-9-mas-alla-de-la-modularidad/
- https://picodotdev.github.io/blog-bitix/2017/09/novedades-y-nuevas-caracteristicas-de-java-9-los-modulos/
- https://www.arquitecturajava.com/java-9-modules/
- https://www.adictosaltrabajo.com/2017/10/30/modularidad-en-java-9-12/
- https://javadesdecero.es/avanzado/modulos-java-ejemplos/

Nuevas Características Java 10 (2018.3)

- *Java 10 tiene una lista más reducida de cambios que Java 9 pero importantes y significativos, muy orientado en optimizaciones internas.*
- *Java es el último en unirse a la fiesta de la inferencia de tipos pero ha sido de forma intencionada ya que el coste de implementarla de forma incorrecta supone un alto coste que hay que mantener en adelante.*
- *En el siguiente enlace pueden ver algunas de las novedades más relevantes:*
- *https://picodotdev.github.io/blog-bitix/2018/03/novedades-de-java-10/*

Inferencia de tipos para variables locales

- *Adición de la nueva palabra reservada var, esto ayuda a no tener que repetir varias veces los tipos en la construcción de un objeto.*
- *En las lambdas los parámetros no es necesario declararlos infiriéndose de la interfaz que implementan.*
- *La inferencia de tipos es la idea que permite al compilador obtener el tipo estático sin que sea necesario escribirlo de forma explícita.*
- *En realidad la inferencia de tipos incluida en Java 10 con var es muy limitada y restringida de manera intencionada.*

Inferencia de tipos para variables locales...

```java
// Java 5
List<String> channels = Collections.<String>emptyList();

List<String> channels = Collections.emptyList();

// Java 7
Map<String, List<String>> userChannels = new HashMap<String, List<String>>();

Map<User, List<String>> userChannels = new HashMap<>();

// Java 8
Predicate<String> nameValidation = (String x) -> x.length() > 0;

Predicate<String> nameValidation = x -> x.length() > 0;

// Java 10
var userChannels = new HashMap<User, List<String>>();

var channels = lookupUserChannels("Tom");
channels.forEach(System.out::println);
```

JavaTypeInference.java

Inferencia de tipos para variables locales...

El tipo en las variables locales no es tan importante ya que normalmente los nombres de las variables son el del tipo. Con *var* se evita repetición entre el tipo y el nombre de la variable, la brevedad de *var* hace destacar el nombre de la variable y proporciona mayor claridad además de tener que escribir menos código repetitivo.

```java
// Java 9
Class.forName("org.postgresql.Driver");
Connection connection = DriverManager.getConnection("jdbc:postgresql://localhost/database", "user", "password");
PreparedStatement statement = connection.prepareStatement("select * from user");
ResultSet resultSet = statement.executeQuery();

// Java 10
Class.forName("org.postgresql.Driver");
var connection = DriverManager.getConnection("jdbc:postgresql://localhost/database", "user", "password");
var statement = connection.prepareStatement("select * from user");
var resultSet = statement.executeQuery();
```

NamesAlign.java

La existencia de *var* no significa que haya de usarse de forma indiscriminada para todas las variables locales sino juiciosamente. En este caso quizá es preferible declarar el tipo por no ser obvio lo que retorna el método *getCities()*.

```java
Map<String, List<City>> countryToCities = getCities();
var countryToCities = getCities();
```

TypeVsVar.java

Nuevas Características Java 11 (2018.9)

- *Java 11 es la primera versión de soporte extendido publicada o LTS bajo el nuevo ciclo de publicaciones que adoptó Java en la versión 9.*
- *Añade varias novedades importantes en cuanto a seguridad y elimina otras que en versiones anteriores ya fueron marcadas como desaconsejadas.*
- *El soporte de Java 11 está planificado que dure hasta 2023 y hasta 2026 de forma extendida lo que son 8 años de soporte.*
- *En esta nueva versión de Java 11 las novedades no son tan relevantes como lo fuero Java 8 con las lambadas y Java 9 con los módulos pero continúa con las mejoras incrementales y* ***proporciona una versión LTS en la que empresas grandes confiarán como base para sus desarrollos****.*
- *En el siguiente enlace pueden ver algunas de las novedades más relevantes:*
- *https://picodotdev.github.io/blog-bitix/2018/09/novedades-y-nuevas-caracteristicas-de-java-11/*

Otros Conceptos Importantes

- *Reflection:*
  - *https://www.arquitecturajava.com/el-concepto-java-reflection/*
  - *https://jarroba.com/reflection-en-java/*
  - *https://www.logicbig.com/tutorials/core-java-tutorial/modules/reflective-access.html*