# 1.- What’s new on Java 8?

A bote pronto, la que más me llamó la atención es la **eliminación del PERM\_GEN,** que venía a ser donde vivían los objetos de que sobrevivían a su paso por el espacio old generation memory, y que ya no está sometido al recolector de basura, y dado que ahí se almacenaban metadatos relativos a clases y despliegues, bien podía darse la circunstancia de que se llenase este espacio sin que pasara nada.

Por el contrario en metaspace, se almacenan los metadatos de clases, pero permite eliminar metadatos de clases que no estén creadas, evitando que crezca indefinidamente, y además, puede crecer a demanda ( si no se especifica el tamaño)

Sin embargo, a nivel operativo, es la **introducción de características propias de la programación funcional**, que lo hacen posible las expresiones lambda. Las lambda nos permiten realizar operaciones sobre parámetros directamente en el código sin tener que realizar una clase que las encapsule, tan es así que es posible pasarlas como argumento a collections.sort.

Mi ejemplo para esto es,por ejemplo, para ordenar un array de nombres, en java “clásico” entiéndase, antes de java 8, hacíamos Collections.sort(vector), y al menos sabíamos que como tenía un método por defecto, que aplica el orden natural.

El problema aquí nos surge cuando lo que que se quiera ordenar con el orden lexicográfico, nos viene con tildes, eñes, etc. Caracteres no ascii…

En ese caso, vale, sort nos permite pasarle un comparador, que será una clase que implementará la interfaz Comparator, que devuelve un valor positivo si el primer operador es mayor que el segundo, negativos si es el segundo mayor que el primero y 0 si son iguales.

Con java 8 , en una línea lo plasmamos, bueno dos, pero no hay que hacer una clase.

Comparator<String> comparadorLongitud = (o1,02)-> o1.length()-o2.length();

Collections.sort(vector,compradorLongitud)

Otra cosa importante… Las interfaces pueden implementar **métodos “por defecto”.** Hasta la fecha, en una interfaz tiene que especificar el comportamiento exacto sin implementarlo.

Luego, aquellas clases que la implementen, deben dar una implementación para todos los métodos que se especifican en la interfaz, cada uno como le corresponda. Desde Java8 se puede indicar como default, y si es necesario, sobreescribirlo en la clase que implemente esa interfaz.

Otro aspecto útil para el uso con lambdas es que podemos declarar **métodos estáticos en las interfaces.**

Y también esto está ligado con algún concepto más, como el de interfaz funcional, que es aquella que tiene al menos un método abstracto.

Y por último, los streams, bueno, eso ya existía algo con este nombre ( para descargar binarios por ejemplo), pero ahora podemos usarlos como agrupaciones de elementos sobre los que se hacen operaciones, típicamente de convertir, filtrar y ordenar( map, filter y sort) .

Hay muchas más cosas, pero las que creo que son más relevantes y manejo con cierto detalle son éstas.

## 2.- Given the following list implement a solution in order to get even numbers using Java 8 Streams

List<Integer> list = Arrays.asList(1,2,3,4);

List<Integer> listSalida = list.stream().filter(num -> num % 2 ==0).collect(Collectors.toList());

## 3.- What do you notice when you do code review?

No tengo experiencia en esto, entiendo que va por el tema de pair programing.. Creo que se establece una serie de buenas prácticas que otro miembro del equipo se encarga de ver tu código para ver que se aplican. Además, puede que nos de alternativas, mejoras o se detecten errores que podrían haber pasado las pruebas unitarias, o cuestiones de eficiencia, enfoques diferentes a un problema, etc.

## Have you ever worked with Scrum? Tell us what it is, what events do you remember and what roles are involved?

Bueno, solamente pseudoscrum, ya que típicamente el alcance del sprint nos lo modifican sin ningún pudor. Los eventos que me vienen a la cabeza, la daily, donde se indica cada miembro del equipo qué tarea tiene con la intención de que pueda saber si hay problemas que impidan el avance, lo que ha hecho y se va a hacer en la jornada, si hay dependencias, problemas o bloqueos, etc.

Otra, bueno, la he comentado de soslayo, la planificación del sprint, se revisa el backlog y se ve en función de la capacidad, qué tareas nos caben en el sprint, y luego el product owner dice lo que va en un momento dado.

Al acabar el sprint, se revisa para ver el grado cumplimiento ( el incremento entre los entregables), y en la retro, se hace reflexión de equipo y se buscan mejoras.

Los actores, el product owner, es quien prioriza, en última instancia quien decide que entra y que sale del sprint, el scrum master, que debe ser un facilitador, que ayude /resuelva o sea capaz de tener acceso a la tecla a tocar en cada situación, el equipo de desarrollo.

No sé si va englobado en la metodología, pero seguro que los stakeholders pueden tener cabida también.

## What access modifiers (or visibility) do you know in Java?

La verdad es que diría que sólo he usado public y private, si te digo otra cosa te engaño. Está protected también. La diferencia es que public permite el acceso al atributo o método desde todos lados otros paquetes, otras clases, etc. Private es lo contrario, sólo es para mi misma clase, y protected es visible sólo desde mi mismo paquete.

## Differences between an abstract class and an interface. When would you use one or the other?

Antes de Java 8, una clase abstracta podría implementar métodos y una interfaz no.

La interfaz define completamente el comportamiento de la clase que hereda, es decir, la clase que la implementa debe implementar todos los métodos (salvo el caso mencionado de los default de java 8).

También, una clase puede implementar varias interfaces, pero creo que sólo puede heredar de una única clase. Ahora mismo yo creo que este aspecto es el que me haría decantarme por uno o por otro.

## What is Maven and why is it used? What is Maven life cycle?

Yo sólo he usado Maven, no lo he configurado,pero para los que hemos tenido que pegarnos con los jar, incluirlas en el proyecto, subes, se pega con el jar que hay en el servidor, etc. es como “magia”.

Básicamente es un gestor de dependencias que usa el fichero pom.xml para indicarle qué librería y qué versión se usa, y él se encarga de incluirlas en el fichero y/o en el ejecutable, según se le indique.

Entiendo que se refiere a todos los “goals” que se definen en el proyecto, desde el clean, compile, test, deploy, package, deploy.

## What is Git and what is it used for? List all Git commands that you know.

Vaya, aquí me pillas, porque sólo he usado GIT en un proyecto ( somos leales a SVN), y se usa sourcetree.. GIT a fin de cuentas es un repositorio de código que permite generar ramas y así desarrollar diferentes funcionalidades por separado hasta que están listas para unirse ( merge) a la principal

Pero así te podría decir

* Config, para usuario y contraseña.
* Init para crear el repositorio
* Add y commit
* reset para volver al una versión.
* Checkout para descartar los cambios del local
* Clone para bajarnos un repositorio entero
* Pull hace lo mismo que un clone pero intenta hacer un merge automático

## What is a mock? What would you use it for?

Bueno yo entiendo un mock como algo de “cartón piedra” que nos permite simular una respuesta de un sistema que no etá listo aún. Algunas herramientas como mockito hacen que este comportamiento no sea tan estático.

## How would you explain to someone what Spring is? What can it bring to their projects?

Spring es un framework es decir, un “paquete” que integra diferentes piezas para facilitar/permitir ciertas tareas, como acceso a bases de datos, acceso web mediante diferentes especificaciones ( web, portlet/ servlets), que implementa patrones de diseño, como, como spring MVC, soporta transacciones, y se integra con facilidad con otros frameworks, que usa POJO’s, de modo que no se obliga ni a heredar de clase algún ni implementar nada de primeras.

Las ventajas, pues las inherentes al uso de framework, que son que la base que utiliza ya está probada, que proporciona gestión de excepciones ligadas a los diferentes componentes, también seguridad, etc. Las dos ventajas que se me ocurren son la inyección de dependencias y la inversión de control. Básicamente lo que ocurre es que en lugar de ser el componente X el que se va a buscar en el entono esa dependencia, es el entorno el que le da la dependencia, de modo que la clase X no se hace necesario saber cómo se crea un objeto dependiente, sino solamente que necesita uno de esos.

## What's the difference between Spring and Spring Boot?

A mi modo de ver, una aplicación springboot es mucho más fácil de configurar, ya que automatiza al máximo el inicio.

Además, el despligue, ya que llevan un servidor apache embebido, el despligue es inmediato.

## Do you know what CQRS is? And Event Sourcing?

Me suenan a patrones de diseño, pero realmente no sé nada.

Differences between IaaS and PaaS. Do you know any of each type?

IaaS es proveer la infraestructura como un servicio, vamos lo que que es AWS o Azure… PaaS es platform as a service, y sería el Oracle cloud, que podría correr sobre un IaaS

## Explain what a Service Mesh is? Do you have an example?

Así de pasada leí que era montar una infraestructura pensada a la mejora de la comunicación de los microservicios, pero no me quedé con más.

## Explain what is TDD? What is triangulation?

Conozco el nombre, Test-drive development, pero no manejo el concepto.

## Apply the Factory pattern with lambda expressions

Ésta pregunta no sé responderla, desconozco dicho patrón.

## Reduce the 3 classes (*OldWayPaymentStrategy*, *CashPaymentStrategy* and *CreditCardStrategy*) into a single class (*PaymentStrategy*). You do not need to create any more classes or interfaces. Also, tell me how you would use *PaymentStrategy*, i.e. the different payment strategies in the Main class

La respuesta creo que la he comentado arriba, y es el uso de la interfaces de métodos estáticos.

La interfaz PaymentStrategy, tendría ambos métodos

package com.babel.pruebaIng1;

public interface PaymentStrategy {

public static double pagaEnEfectivo(double enEfectivo) {

double salida = 0;

double serviceCharge =5.00;

double creditCardFee = 10.00;

if ( enEfectivo >0) {

salida = enEfectivo+serviceCharge;

}

return salida;

}

public static double pagaConTarjeta( double conTarjeta) {

double salida = 0;

double serviceCharge =5.00;

double creditCardFee = 10.00;

if (conTarjeta >0 ){

salida = conTarjeta + serviceCharge + creditCardFee;

}

return salida;

}

Y en el main, que al ser estático, también permite ponerlo en la misma interfaz, he usado mis inicios, cuando preguntábamos por la consola, en función de la selección del usuario, el importe y el medio de pago..

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String txtSalida = null;

String medioPago = null;

double pago = 0;

double pagoTotal = 0;

do {

System.out.println("\*\*Por favor introduzca un valor para el pago. FIN para salir");

try {

txtSalida = br.readLine();

if (!"fin".equalsIgnoreCase(txtSalida)) {

pago = Double.parseDouble(txtSalida);

System.out.println("Pulse 1 para pagar en efectivo, 2 para pagar con tarjeta");

medioPago = br.readLine();

if ("1".equals(medioPago)) {

pagoTotal = PaymentStrategy.pagaEnEfectivo(pago);

System.out.println("Ha efectuado el pago de "+pago+" y en total "+pagoTotal);

}else if ("2".equals(medioPago)) {

pagoTotal = PaymentStrategy.pagaConTarjeta(pago);

System.out.println("Ha efectuado el pago de "+pago+" y en total "+pagoTotal);

}else System.out.println("Opcion no valida, pago cancelado.");

}

} catch (IOException ex) {

System.out.println(" La entrada no es valida "+ex.getMessage());

}

} while(!txtSalida.equalsIgnoreCase("FIN"));

System.out.println("\*\*\*El programa ha terminado\*\*\*\*");

La salida por pantalla

