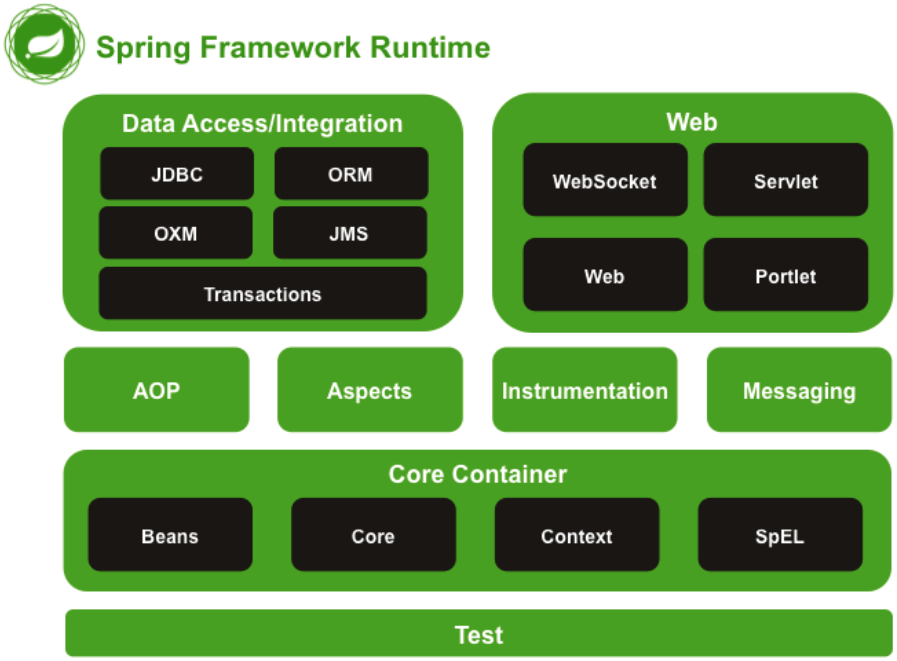
GUSTAVO CORONEL
DESARROLLA SOFTWARE

DESARROLLO WEB
CON SPRING BOOT



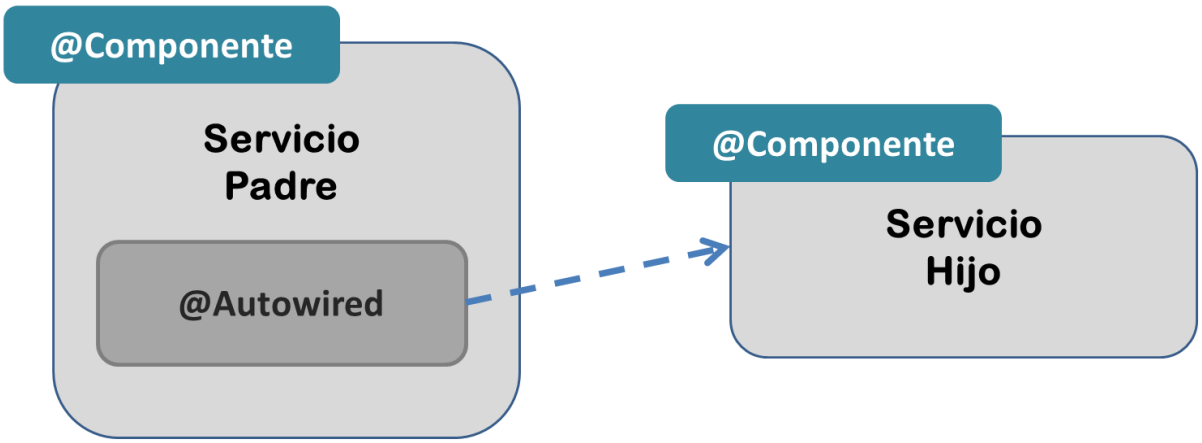
UNIDAD 03
SPRING CORE

Eric Gustavo Coronel Castillo
I N S T R U C T O R
youtube.com/DesarrollaSoftware
gcoronelc@gmail.com

CONTENIDO

INTRODUCCIÓN

Contexto



Usar anotaciones dentro de las clases permite que la configuración y la implementación estén en un único sitio.

A continuación, tienes algunas de las anotaciones de Spring Core.

@Configuration

Se utiliza para indicar que una clase declara uno o más `@Bean` métodos. Estas clases son procesadas por el contenedor de Spring para generar definiciones de beans y solicitudes de servicio para esos beans en tiempo de ejecución.

@Bean

Indica que un método produce un bean para ser administrado por el contenedor Spring. Esta es una de las anotaciones más utilizadas e importantes. La anotación @Bean también se puede usar con parámetros como `name`, `initMethod` y `destroyMethod`.

- **name:** Permite dar nombre a bean.
- **initMethod:** Permite elegir el método de inicialización.
- **destroyMethod:** Permite elegir el método de destrucción.

```
@Configuration
public class AppConfig {

   @Bean(name = "comp", initMethod = "turnOn", destroyMethod = "turnOff")
   Computer computer(){
      return new Computer();
   }

}

public class Computer {

   public void turnOn(){
      System.out.println("Load operating system");
   }
   public void turnOff(){
      System.out.println("Close all programs");
   }
}
```

@PreDestroy y @PostConstruct

Son una forma alternativa para bean `initMethod` y `destroyMethod`. Se puede usar cuando la clase de bean está definida por nosotros. Por ejemplo;

```
public class Computer {

   @PostConstruct
   public void turnOn(){
      System.out.println("Load operating system");
   }

   @PreDestroy
   public void turnOff(){
      System.out.println("Close all programs");
   }
}
```

@ComponentScan

Configura las directivas de exploración de componentes para su uso con las clases `@Configuration`. Aquí podemos especificar los paquetes base para buscar componentes Spring.

@Component

Indica que una clase anotada es un “componente”. Dichas clases se consideran candidatas para la detección automática cuando se utiliza la configuración basada en anotaciones y el escaneo de classpath.

@PropertySource

Proporciona un mecanismo declarativo simple para agregar una fuente de propiedad al entorno de Spring. Hay una anotación similar para agregar una matriz de archivos fuente de propiedades, es decir `@PropertySources`.

@Service

Indica que una clase anotada es un “Servicio”. Esta anotación es una especialización de `@Component`, lo que permite que las clases de implementación se detecten automáticamente a través del escaneo de classpath.

@Repository

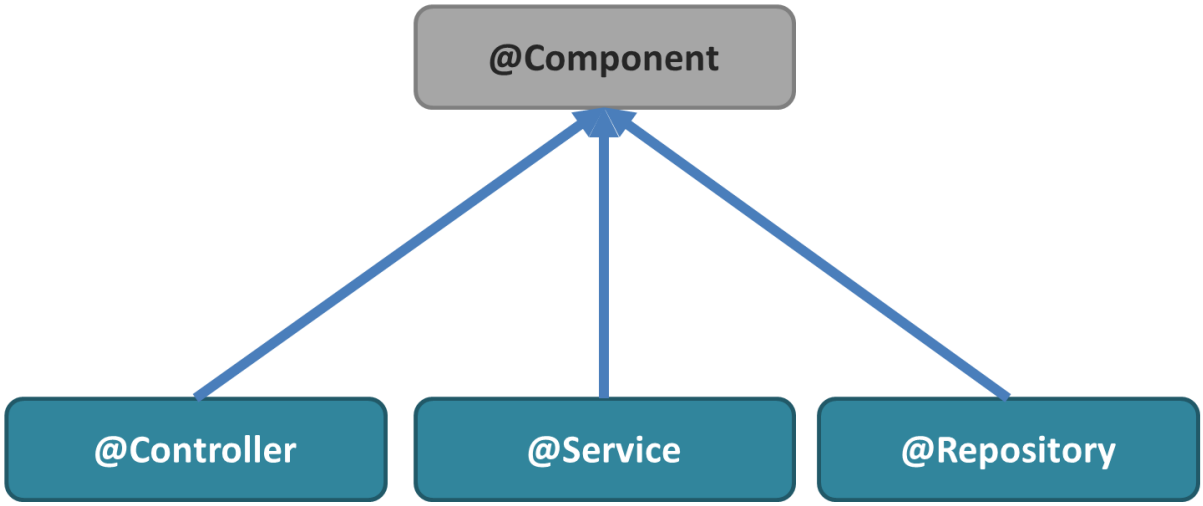
Indica que una clase anotada es un “Repositorio”. Esta anotación es una especialización de `@Component` y es recomendable usarla con clases DAO.

@Autowired

La anotación `@Autowired` se utiliza para la inyección automática de beans. La anotación `@Qualifier` se usa junto con `@Autowired` para evitar confusiones cuando tenemos dos o más beans configurados para el mismo tipo.

COMPONENTES

Anotaciones de Beans



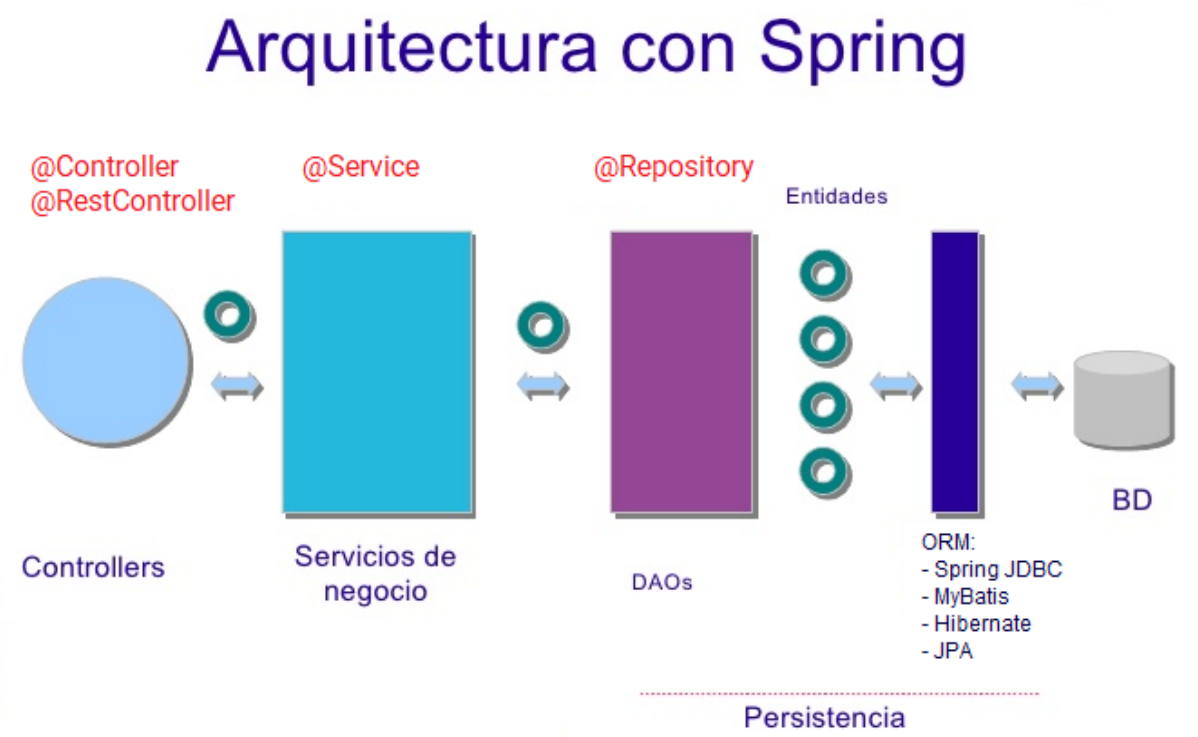
Las anotaciones se utilizan para que Spring detecte las clases importantes dentro del propio código fuente de Java.

Tienes cuatro anotaciones para definir tus componentes:

- **@Component:** Para que definas componentes genéricos.
- **@Controller:** Para que definas tus componentes controladores.
- **@RestController:** Para que definas controladores para servicios REST.
- **@Service:** Para que definas tus componentes de negocio.
- **@Repository:** Para que definas tus componentes de persistencia.

Las anotaciones Controller, RestController, Service y Repository heredan de Component.

Arquitectura



La figura anterior, te muestra la arquitectura que debes utilizar con Spring para el desarrollo de aplicaciones, como puedes observar, la capa de persistencia la puedes trabajar con Spring JDBC, Spring Data o utilizar algún Framework ORM.

@Component

Esta anotación te sirve para añadir un estereotipo de forma genérica a una clase. Un "estereotipo" es una manera de clasificar las clases a un alto nivel. Esta información normalmente de tipo semántica, pero puede utilizarse para caracterizar una clase al punto de establecer como debe comportarse cuando se produzca una excepción, por ejemplo.

```
import org.springframework.stereotype.Component;

@Component
public class VentaService {
   . . .
   . . .
}
```

La anotación `@Component` es genérica y no se espera que realmente la utilices, lo esperado es que uses algunas de las otras anotaciones derivadas de ella como `@Repository`, `@Service`, `@Controller` y `@RestController`, para los componentes de persistencia, servicios y controladores respectivamente.

@Scope

Por defecto el alcance de un bean es `sigleton`, lo que indica que solo existirá una sola instancie de la clase, o que es lo mismo, solo se creará un solo objeto.

Esta anotación debes utilizarla cuando quieras modificar el alcance por defecto de un bean. A continuación, tienes un ejemplo ilustrativo.

```
import org.springframework.stereotype.Component;

@Component
@Scope("session")
public class VentaService {
   . . .
   . . .
}
```

Los valores asociados con esta anotación son: `singleton`, `prototype`, `request` y `session`.

DEPENDENCIAS

@Autowired

Esta anotación hace que el framework aplique el autodescubrimiento e inyección automática de dependencias.

Lo vas a usar frecuentemente sobre setters, de forma que Spring automáticamente busca el bean que mejor se adapta al tipo del parámetro:

```
@Autowired
public void setMotor(Motor motor) {
   this.motor = motor;
}
```

Pero también puedes utilizar esta anotación directamente sobre las propiedades, sobre el constructor, o sobre un método, con cualquier nombre y con cualquier cantidad y tipo de parámetros:

```
// Sobre una propiedad
@Autowired
private Motor motor;

// Sobre un constructor
@Autowired
public Coche(Motor motor) {
   this.motor = motor;
}

// Sobre un método
@Autowired
public void montaje(Motor motor, Volante volante)       {
   this.motor = motor;
   this.volante = volante;
}
```

Incluso lo puedes utilizar para obtener todos los beans de un mismo tipo declarados en la configuración y que se almacenen en algún tipo de colección:

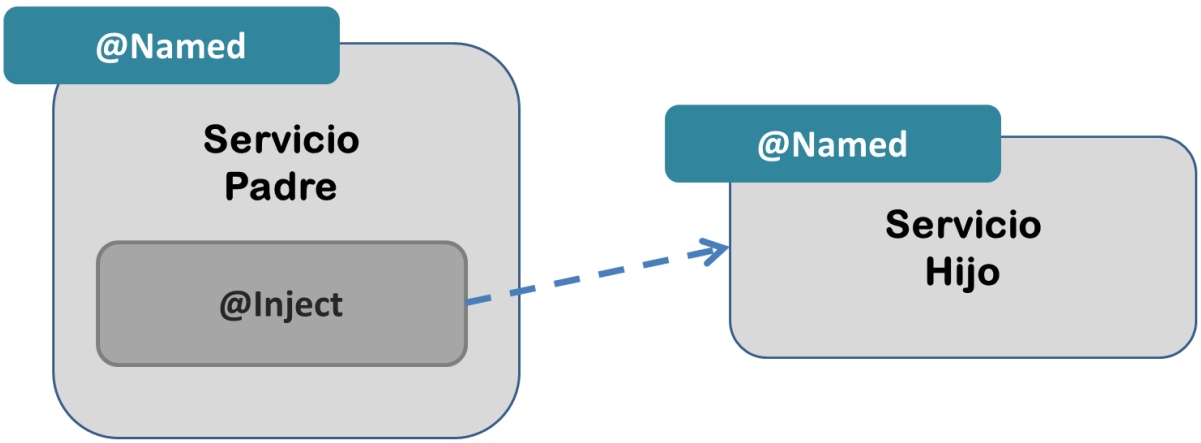
```
@Autowired
private Pieza[] piezas;

@Autowired
private List<Pieza> piezas;

@Autowired
private Map<String, Pieza> piezas;
```

Si la dependencia no se puede resolver se dispara una excepción. No obstante, este comportamiento lo puede modificar utilizando el parámetro `required`, tal como lo observas a continuación:

```
@Autowired(required=false)
private Copiloto copiloto;
```

ANOTACIONES DE JAVA EE

Contexto



La competencia entre los estándares de Java EE y el Spring Framework es cada vez más dura ya que las similitudes entre ambos son muchas. Elegir uno u otro depende de muchas cosas. Spring también soporte las anotaciones de Java EE.

Configuración

Para que puedas utilizar estas anotaciones es necesario añadir la librería respectiva.

Es necesario que añadas la dependencia en el fichero `pom.xml`:

```
<dependency>
   <groupId>javax.inject</groupId>
   <artifactId>javax.inject</artifactId>
   <version>1</version>
</dependency>
```

@Named

Esta anotación es similar a `@Component`.

```
import javax.inject.Named;

@Named
public class Motor {
   . . .
   . . .
}
```

@Inject

Esta anotación tiene el mismo comportamiento que la anterior `@Autowired`, aunque carece del parámetro `required`.

```
import javax.inject.Named;
import javax.inject.Inject;

@Named
public class Coche {

   @ Inject
   private Motor motor;

}
```

@Resource

Esta anotación la utilizaras para eliminar ambigüedades a la hora de inyectar dependencias automáticamente, sobre todo si se aplica la anotación `@Autowired` en propiedades que tienen como tipo una interface, ya que distintas clases pueden implementar una misma interface.

```
@Autowired
@Resource(name="volante")
private Pieza volante;

@Autowired
@Resource(name="retrovisor")
private Pieza retrovisor;
```

Si omites el valor del parámetro `name` intentará resolver la dependencia buscando un bean que tenga el mismo nombre de la propiedad a la que se ha aplicado la anotación. Y si no encuentra un bean intentará resolverlo por tipo.

@Qualifier

Esta anotación tiene un comportamiento similar a la anterior `@Resource`, pero utiliza los roles de los beans en vez de sus identificadores para resolver las dependencias.

Su uso más común es aplicarla usando el valor de alguna característica semántica de los beans definidos en la configuración. La "característica semántica" es simplemente el valor de la etiqueta `qualifier` que tiene un bean, y que se usa para indicar el "rol" que tiene un bean dentro de la aplicación.

Supongamos que tienes la siguiente definición de beans, donde los beans representan piezas de un coche que se clasifican en función de su posición:

```
public interface Pieza {
   . . .
}

@Component
@Qualifier("frontal")
public class Parachoque implements Pieza {
   . . .
}

@Component
@Qualifier("maletera")
public class Maletera implements Pieza {
   . . .
}
```

Utilizando la anotación `@Qualifier` se puede eliminar la ambigüedad a la hora de resolver las dependencias de forma automática, incluso cuando se aplican a colecciones:

```
  @Autowired
  @Qualifier("posterior")
  private Pieza maletera;

  @Autowired
  @Qualifier("frontal")
  private List<Pieza> morro;
```

@PostConstruct

Esta anotación la debes utilizar cuando necesitas ejecutar un método de un bean después de que ha sido instanciado por Spring y resuelto todas sus dependencias.

```
import javax.inject.Named;
import javax.inject.PostConstruct;

@Named
public class Coche {

   @PostConstruct
   public void initBean() {
      System.out.println("Bean iniciado.");
   }

}
```

Esta misma funcionalidad la consigues implementando la interface `InitializingBean`.

@PreDestroy

Esta anotación la debes utilizar para configurar un método que se ejecuta antes de que sea destruido del contexto de Spring.

```
import javax.inject.Named;
import javax.inject.PreDestroy;

@Named
public class Coche {

   @PreDestroy
   public void resetBean() {
      System.out.println("Bean listo para ser destruido.");
   }

}
```

Esta misma funcionalidad la consigues implementando la interfaz `DisposableBean`.

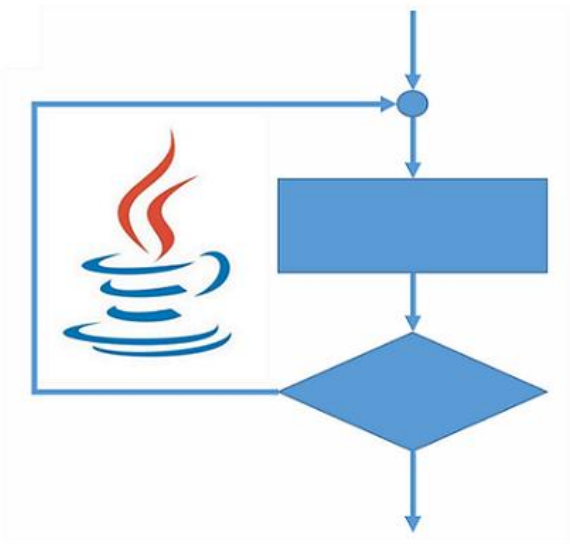
CURSOS VIRTUALES

Acceso a los Cursos Virtuales

En esta URL tienes los accesos a los cursos virtuales:

http://gcoronelc.github.io

Fundamentos de Programación con Java



Tener bases sólidas de programación muchas veces no es fácil, creo que es principalmente por que en algún momento de tu aprendizaje mezclas la entrada de datos con el proceso de los mismos, o mezclas el proceso con la salida o reporte, esto te lleva a utilizar malas prácticas de programación que luego te serán muy difíciles de superar.

En este curso aprenderás las mejores prácticas de programación para que te inicies con éxito en este competitivo mundo del desarrollo de software.

URL del Curso: https://n9.cl/gcoronelc-java-fund

Avance del curso: https://n9.cl/gcoronelc-fp-avance

Cupones de descuento: http://gcoronelc.github.io

Java Orientado a Objetos

CURSO PROFESIONAL DE JAVA ORIENTADO A OBJETOS

Eric Gustavo Coronel Castillo
www.desarrollasoftware.com
INSTRUCTOR

En este curso aprenderás a crear software aplicando la Orientación a Objetos, la programación en capas, el uso de patrones de software y Swing.

Cada tema está desarrollado con ejemplos que demuestran los conceptos teóricos y finalizan con un proyecto aplicativo.

URL del Curso: https://bit.ly/2B3ixUW

Avance del curso: https://bit.ly/2RYGXIt

Cupones de descuento: http://gcoronelc.github.io

Programación con Java JDBC

PROGRAMACIÓN DE BASE DE DATOS ORACLE CON JAVA JDBC

Eric Gustavo Coronel Castillo
www.desarrollasoftware.com
INSTRUCTOR

En este curso aprenderás a programas bases de datos Oracle con JDBC utilizando los objetos Statement, PreparedStatement, CallableStatement y a programar transacciones correctamente teniendo en cuenta su rendimiento y concurrencia.

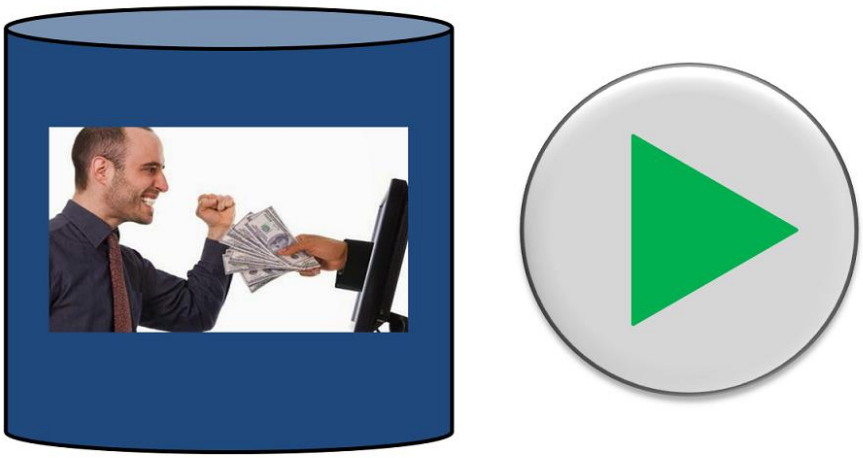
Al final del curso se integra todo lo desarrollado en una aplicación de escritorio.

URL del Curso: https://bit.ly/31apy0O

Avance del curso: https://bit.ly/2vatZOT

Cupones de descuento: http://gcoronelc.github.io

Programación con Oracle PL/SQL



En este curso aprenderás a programas las bases de datos ORACLE con PL/SQL, de esta manera estarás aprovechando las ventas que brinda este motor de base de datos y mejoraras el rendimiento de tus consultas, transacciones y la concurrencia.

Los procedimientos almacenados que desarrolles con PL/SQL se pueden ejecutarlo de Java, C#, PHP y otros lenguajes de programación.

URL del Curso: https://bit.ly/2YZjfxT

Avance del curso: https://bit.ly/3bciqYb

Cupones de descuento: http://gcoronelc.github.io