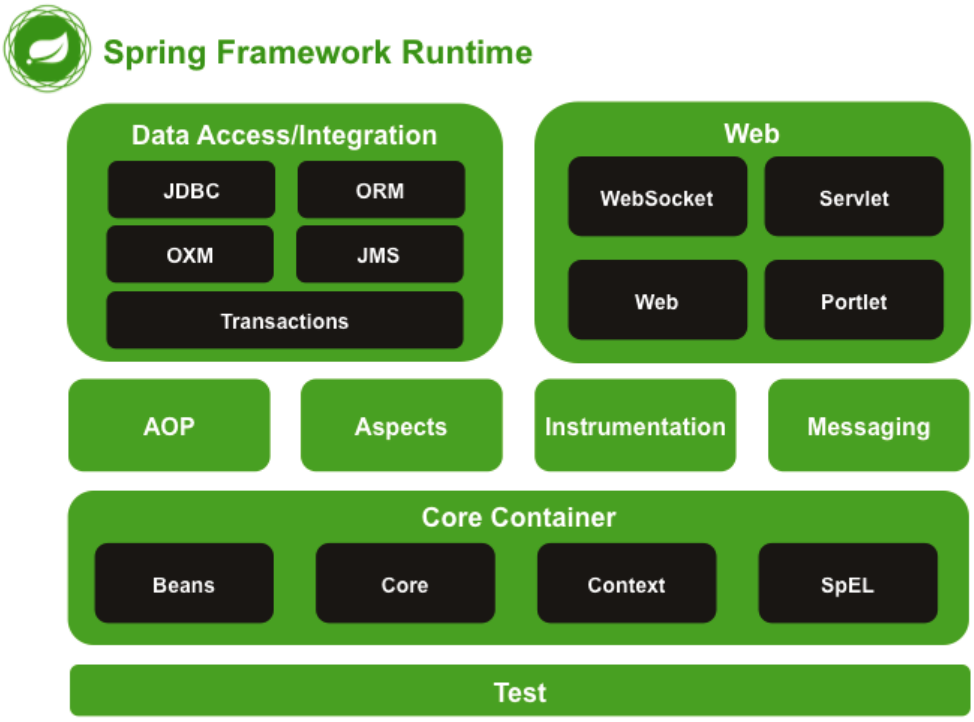
GUSTAVO CORONEL
DESARROLLA SOFTWARE

DESARROLLO WEB
CON SPRING BOOT

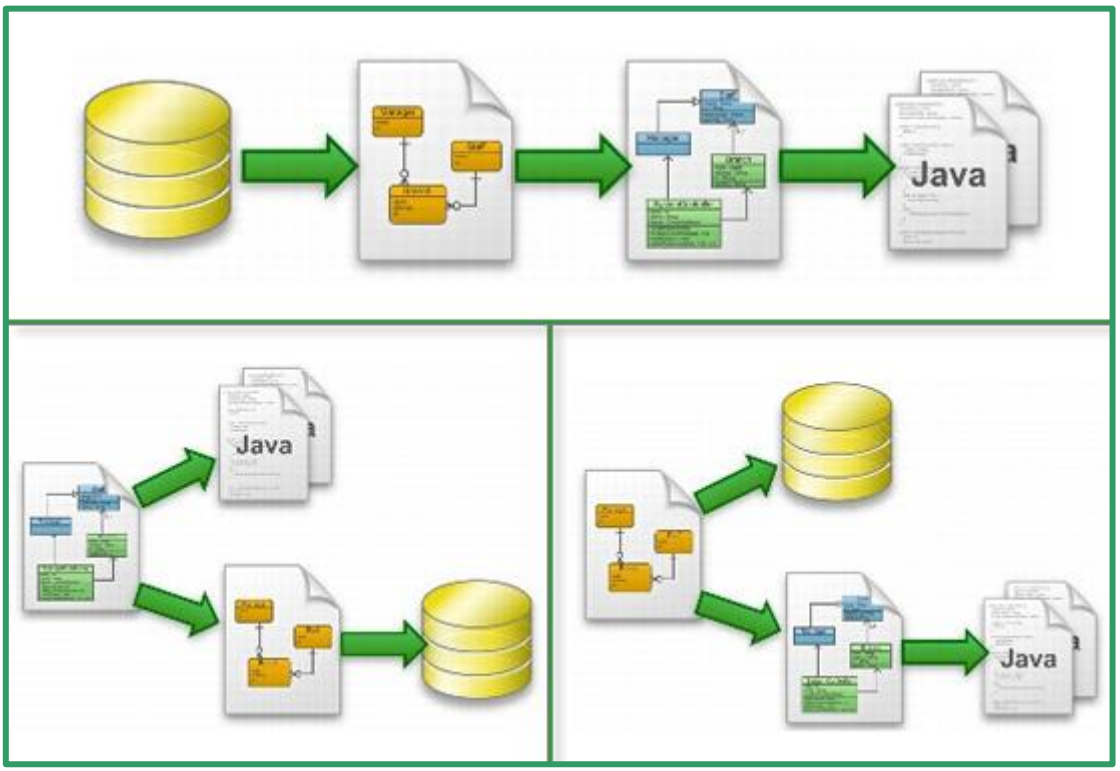


UNIDAD 08
SPRING DATA JPA

Eric Gustavo Coronel Castillo
www.desarrollasoftware.com
gcoronelc@gmail.com
I N S T R U C T O R

CONTENIDO

OBJECT RELATIONAL MAPPING - ORM

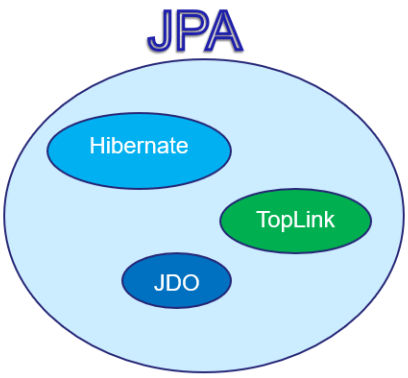


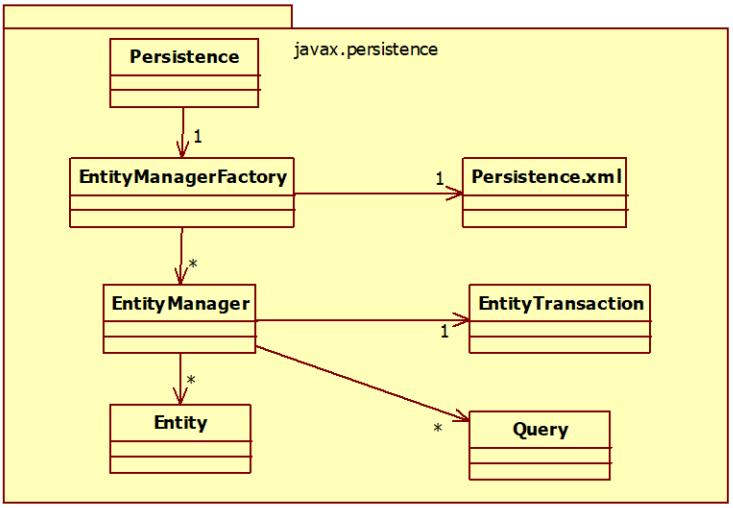
ORM es un acrónimo de mapeo relacional de objetos, lo que básicamente significa, es una forma de interactuar con RDBM de una manera orientada a objetos. Los ORM esencialmente brindan es una base de datos de objetos, que es mucho más fácil de manejar para un desarrollador. Los ORM brindan otra capa de abstracción para que no tengas que comprender la arquitectura subyacente. Los ORM, como herramienta, te permiten interactuar con las bases de datos utilizando tu lenguaje preferido en lugar de SQL.

Existen varios framework ORM para trabajar con Java, por ejemplo:

- JDO
- Hibernate
- TopLink
- EclipseLink

JAVA PERSISTENCE API - JPA

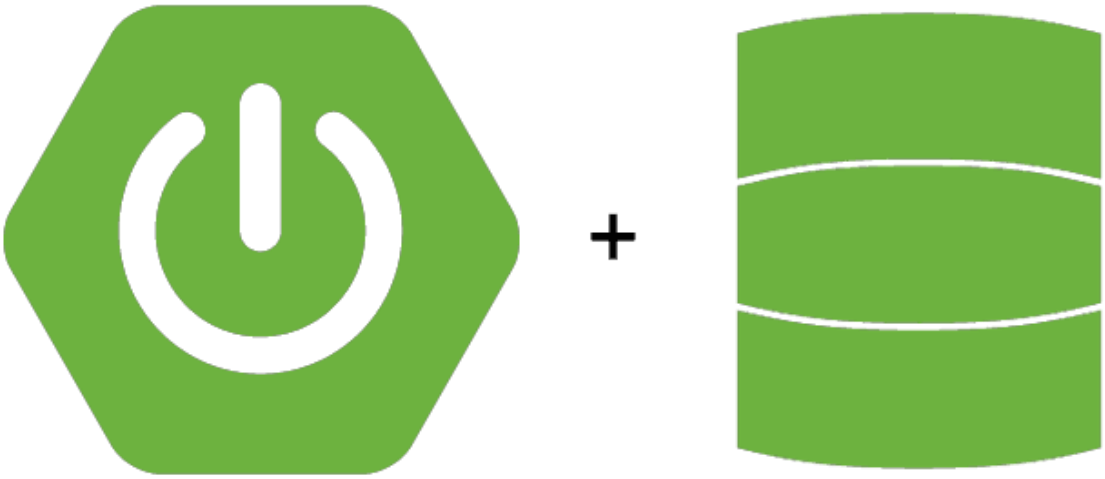




JPA es la propuesta estándar que ofrece Java para implementar un Framework Object Relational Mapping (ORM), que permite interactuar con la base de datos por medio de objetos, de esta forma, JPA es el encargado de convertir los objetos Java en instrucciones para el Manejador de Base de Datos.

Cuando empezamos a trabajamos con bases de datos en Java lo hacemos utilizando el API JDBC, el cual nos permite realizar consultas directas a la base de datos a través de consultas SQL nativas. JDBC por mucho tiempo fue la única forma de interactuar con las bases de datos, pero representaba un gran problema y es que Java es un lenguaje orientado a objetos y se tenía que convertir los atributos de las clases en una consulta SQL como SELECT, INSERT, UPDATE, DELTE, etc. lo que ocasionaba un gran esfuerzo de trabajo y provocaba muchos errores en tiempo de ejecución, debido principalmente a que las consultas SQL se tenían que generar frecuentemente al vuelo.

FUNDAMENTOS



¿Qué es Spring Data JPA?

Spring Data JPA es una librería que forma parte de la familia Spring Data. Spring Data JPA facilita la implementación de repositorios basados en JPA. Este marco tiene soporte mejorado para capas de acceso a datos basadas en JPA.

Spring Data JPA le ayuda a centrarse en la lógica empresarial en lugar de la complejidad técnica y el código estándar.

¿JPA y Spring Data JPA son iguales?

La respuesta es no.

JPA es una especificación que estandariza la forma en que los objetos Java se asignan a un sistema de base de datos relacional. Al ser solo una especificación, JPA consta de un conjunto de interfaces, como EntityManagerFactory, EntityManager y anotaciones que lo ayudan a asignar un objeto de entidad Java a una tabla de base de datos.

Hay varios proveedores de JPA, como HIbernate, EclipseLink u Open JPA, que puede utilizar.

Spring Data JPA es una abstracción de acceso a datos JPA. Al igual que JPA, Spring Data JPA no puede funcionar sin un proveedor de JPA. Genera consultas JPA en su nombre a través de convenciones de nombres de métodos.

JPA maneja la mayor parte de la complejidad del acceso a la base de datos basada en JDBC y los ORM (asignaciones relacionales de objetos). Además de eso, Spring Data JPA reduce la cantidad de código repetitivo requerido por JPA.

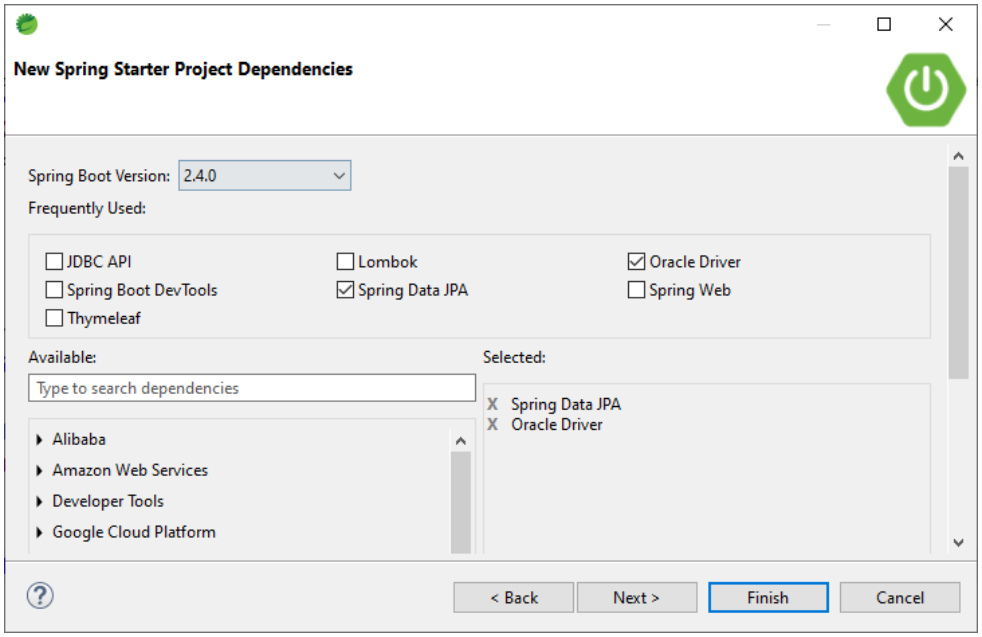
¿Por qué se desarrolló Spring Data JPA?

Implementar una capa de acceso a datos de una aplicación usando JPA o JDBC es engorroso. Demasiado código repetitivo para ejecutar consultas simples, realizar paginación y auditoría.

Para evitar todo esto, se desarrolló Spring Data JPA. Spring Data JPA mejora la implementación de capas de acceso a datos al reducir el esfuerzo que realmente se necesita.

INICIAR PROYECTO

Creación del proyecto



Dependencias

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
   <groupId>com.oracle.database.jdbc</groupId>
   <artifactId>ojdbc8</artifactId>
   <scope>runtime</scope>
</dependency>
```

Parámetros de Conexión

Estos parámetros se deben incluir en el archivo de propiedades (application.properties), también puedes establecer el puerto del servidor.

```
# Oracle settings
spring.datasource.url=jdbc:oracle:thin:@localhost:1521/XE
spring.datasource.username=ventas
spring.datasource.password=admin
spring.jpa.database-platform=org.hibernate.dialect.Oracle12cDialect
spring.jpa.hibernate.ddl-auto=none
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

EJEMPLO ILUSTRATIVO

Clase Entidad

```
package com.desarrollasoftware.app.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;

@Entity
@Table(name = "EMPLEADO")
@SequenceGenerator(name = "sq_empleado", sequenceName = "sq_empleado",
allocationSize = 1)
public class Empleado {

   @Id
   @Column(name = "idemp")
   @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "sq_empleado")
   private Long id;

   @Column(name = "nombre")
   private String nombre;

   @Column(name = "apellido")
   private String apellido;

   @Column(name = "email")
   private String email;

   @Column(name = "telefono")
   private String telefono;

   public Empleado() {
   }

   public Empleado(String nombre, String apellido,
   String email, String telefono) {
      super();
```

```
        this.nombre = nombre;
        this.apellido = apellido;
        this.email = email;
        this.telefono = telefono;
    }

    public Empleado(Long id, String nombre, String apellido,
    String email, String telefono) {
        super();
        this.id = id;
        this.nombre = nombre;
        this.apellido = apellido;
        this.email = email;
        this.telefono = telefono;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApellido() {
        return apellido;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
```

```
    this.email = email;
  }

  public String getTelefono() {
    return telefono;
  }

  public void setTelefono(String telefono) {
    this.telefono = telefono;
  }

  @Override
  public String toString() {
    return "Empleado [id=" + id + ", nombre=" + nombre + ", apellido="
        + apellido + ", email=" + email
        + ", telefono=" + telefono + "]";
  }

}
```

Capa Repository

```
package com.desarrollasoftware.app.repository;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import com.desarrollasoftware.app.entity.Empleado;

@Repository
public interface EmpleadoRepository extends CrudRepository<Empleado, Long>{

}
```

Capa Service

Interface

```
package com.desarrollasoftware.app.service;

import java.util.List;

import com.desarrollasoftware.app.entity.Empleado;

public interface EmpleadoService {

   List<Empleado> listarTodos();

   Empleado buscarPorId(Long id);

   Empleado grabar(Empleado empleado);

   void eliminar(Long id);

}
```

Implementación

```
package com.desarrollasoftware.app.service.impl;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.desarrollasoftware.app.entity.Empleado;
import com.desarrollasoftware.app.repository.EmpleadoRepository;
import com.desarrollasoftware.app.service.EmpleadoService;

@Service
public class EmpleadoServiceImpl implements EmpleadoService {

   @Autowired
   private EmpleadoRepository empleadoRepository;

   @Override
```

```
    public List<Empleado> listarTodos(){
        return (List<Empleado>) empleadoRepository.findAll();
    }

    @Override
    public Empleado buscarPorId(Long id) {
        return empleadoRepository.findById(id).orElse(null);
    }

    @Override
    public Empleado grabar(Empleado empleado) {
        return empleadoRepository.save(empleado);
    }

    @Override
    public void eliminar(Long id) {
        empleadoRepository.deleteById(id);
    }

}
```

Controlador

```
package com.desarrollasoftware.app.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import com.desarrollasoftware.app.entity.Empleado;
import com.desarrollasoftware.app.service.EmpleadoService;

@Controller
@RequestMapping("/empleados")
public class EmpleadoController {
```

```
    @Autowired
    private EmpleadoService empleadoService;

    @GetMapping({ "/", "/todos" })
    @ResponseBody
    public   List<Empleado> listar() {
       List<Empleado> lista = empleadoService.listarTodos();
       return lista;
    }

    @GetMapping("/leer/{id}")
    @ResponseBody
    public Empleado editar(@PathVariable("id") Long id) {
       Empleado empleado = empleadoService.buscarPorId(id);
       return empleado;
    }

    @PostMapping("/grabar")
    @ResponseBody
    public Empleado guardar(@ModelAttribute Empleado empleado) {
       System.err.println(empleado);
       Empleado bean = empleadoService.grabar(empleado);
       return bean;
    }

    @GetMapping("/eliminar/{id}")
    @ResponseBody
    public Empleado eliminar(@PathVariable("id") Long id) {
       Empleado empleado = empleadoService.buscarPorId(id);
       empleadoService.eliminar(id);
       return empleado;
    }

}
```

CREANDO CONSULTAS

Fundamentos

El módulo JPA permite definir una consulta manualmente utilizando ciertas palabras claves y sus respectivos argumentos.

A continuación, tienes algunos ejemplos:

```
public interface EmpleadoRepository extends CrudRepository<Empleado, Long>{

    List<Empleado> findByNombreAndApellido(String nombre, String apellido);

    List<Empleado> findByNombreLike(String nombre);

    List<Empleado> findByNombreContaining(String nombre);

    List<Empleado> findByNombreContainingIgnoreCase(String nombre);

    List<Empleado> findByNombreLikeIgnoreCase(String nombre);

}
```

Palabras Claves

La siguiente tabla describe las palabras clave admitidas para JPA y qué puedes utilizar para construir tus métodos, según tus necesidades.

Keyword	Sample / JPQL snippet	
Distinct	Sample	`findDistinctByLastnameAndFirstname`
	JPQL snippet	`select distinct ... where x.lastname = ?1 and x.firstname = ?2`
And	Sample	`findByLastnameAndFirstname`
	JPQL snippet	`... where x.lastname = ?1 and x.firstname = ?2`
Or	Sample	`findByLastnameOrFirstname`
	JPQL snippet	`... where x.lastname = ?1 or x.firstname = ?2`
Is, Equals	Sample	`findByFirstname, findByFirstnameIs, findByFirstnameEquals`
	JPQL snippet	`... where x.firstname = ?1`
Between	Sample	`findByStartDateBetween`
	JPQL snippet	`... where x.startDate between ?1 and ?2`
LessThan	Sample	`findByAgeLessThan`
	JPQL snippet	`... where x.age < ?1`
LessThanEqual	Sample	`findByAgeLessThanEqual`
	JPQL snippet	`... where x.age <= ?1`
GreaterThan	Sample	`findByAgeGreaterThan`
	JPQL snippet	`... where x.age > ?1`
GreaterThanEqual	Sample	`findByAgeGreaterThanEqual`
	JPQL snippet	`... where x.age >= ?1`
After	Sample	`findByStartDateAfter`
	JPQL snippet	`... where x.startDate > ?1`
Before	Sample	`findByStartDateBefore`
	JPQL snippet	`... where x.startDate < ?1`
IsNull, Null	Sample	`findByAge(Is)Null`
	JPQL snippet	`... where x.age is null`
IsNotNull, NotNull	Sample	`findByAge(Is)NotNull`

Keyword	Sample / JPQL snippet	
	JPQL snippet	... where x.age not null
Like	Sample	findByFirstnameLike
	JPQL snippet	... where x.firstname like ?1
NotLike	Sample	findByFirstnameNotLike
	JPQL snippet	... where x.firstname not like ?1
StartingWith	Sample	findByFirstnameStartingWith
	JPQL snippet	... where x.firstname like ?1 (parameter bound with appended %)
EndingWith	Sample	findByFirstnameEndingWith
	JPQL snippet	... where x.firstname like ?1 (parameter bound with prepended %)
Containing	Sample	findByFirstnameContaining
	JPQL snippet	... where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	Sample	findByAgeOrderByLastnameDesc
	JPQL snippet	... where x.age = ?1 order by x.lastname desc
Not	Sample	findByLastnameNot
	JPQL snippet	... where x.lastname <> ?1
In	Sample	findByAgeIn(Collection<Age> ages)
	JPQL snippet	... where x.age in ?1
NotIn	Sample	findByAgeNotIn(Collection<Age> ages)
	JPQL snippet	... where x.age not in ?1
True	Sample	findByActiveTrue()
	JPQL snippet	... where x.active = true
False	Sample	findByActiveFalse()
	JPQL snippet	... where x.active = false
IgnoreCase	Sample	findByFirstnameIgnoreCase
	JPQL snippet	... where UPPER(x.firstname) = UPPER(?1)

Consultas Nombradas

Puedes utilizar la anotación `@NamedQuery` para definir consultas en la misma entidad, tal como lo puedes observar en el siguiente ejemplo:

```
@Entity
@Table(name = "EMPLEADO")
@SequenceGenerator(name = "sq_empleado", sequenceName = "sq_empleado",
                    allocationSize = 1)
@NamedQuery(name= "Empleado.buscarPorEmail",
             query = "SELECT e FROM Empleado e WHERE e.email = ?1")
public class Empleado {

}
```

Para que puedas utilizar estas consultas nombradas, debes declararlas en la interface en el repositorio de la siguiente manera:

```
@Repository
public interface EmpleadoRepository extends CrudRepository<Empleado, Long>{

    Empleado buscarPorEmail(String email);

}
```

Usando la Anotación @Query

Puedes utilizar la anotación @Query para definir una consulta, tal como se ilustra en el siguiente ejemplo:

```
@Repository
public interface EmpleadoRepository extends CrudRepository<Empleado, Long> {

   @Query("select e from Empleado e where e.id = ?1")
   Empleado buscarPorId(Long id);

}
```

Usando Expresiones `LIKE` Avanzadas

La creación de consultas manuales con `@Query` permite la definición de expresiones LIKE avanzadas dentro de la definición de consulta, como se muestra en el siguiente ejemplo:

```
@Repository
public interface EmpleadoRepository extends CrudRepository<Empleado, Long> {

   @Query("select e from Empleado e where UPPER(e.nombre) like %?1%")
   List<Empleado> buscarPorNombre(String nombre);

}
```

En el ejemplo anterior, el operador LIKE reconoce el carácter % y la consulta se transforma en una consulta JPQL válido. Al ejecutar la consulta, el parámetro que le pasas en la llamada al método se aumenta con el patrón LIKE previamente reconocido.

Consultas Nativas

La anotación `@Query` te permite ejecutar consultas nativas, para lo cual debes utilizar el flag `nativeQuery`, como se muestra en el siguiente ejemplo:

```
@Repository
public interface EmpleadoRepository extends CrudRepository<Empleado, Long> {

   @Query(value = "SELECT * FROM EMPLEADO WHERE UPPER(nombre) LIKE ?1 "
      + " AND UPPER(apellido) LIKE ?2", nativeQuery = true)
   List<Empleado> buscarEmpleados(String nombre, String apellido);

}
```

En este caso debes tener en cuenta el orden de los parámetros.

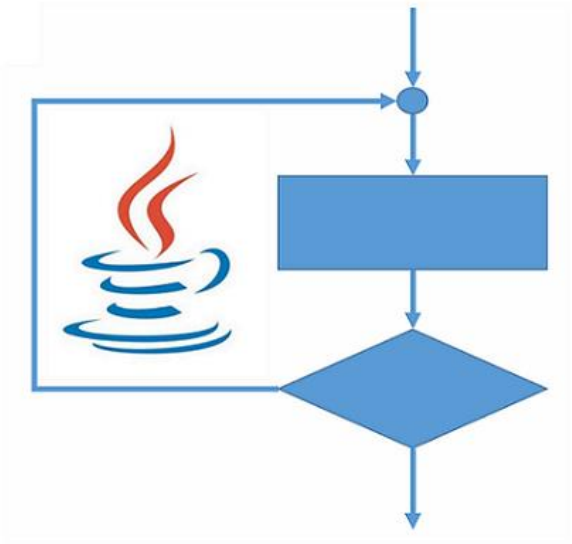
CURSOS VIRTUALES

Cupones

En esta URL se publican cupones de descuento:

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

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.

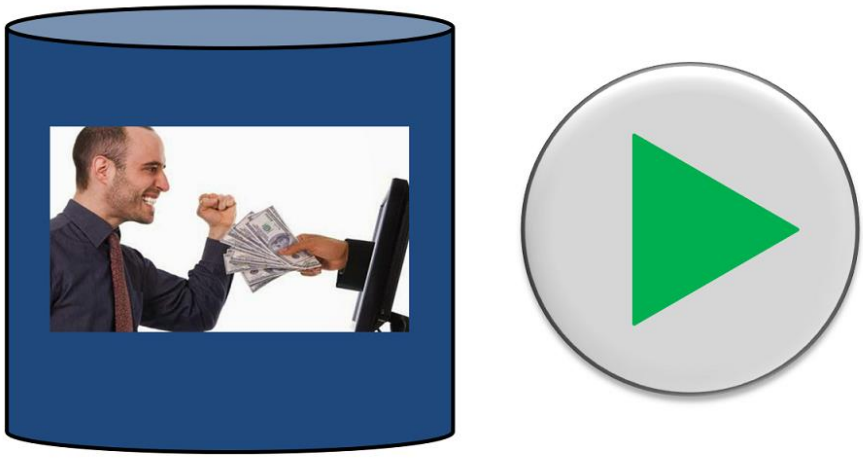
Al final del curso se integra todo lo desarrollado en una aplicación de escritorio.

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