# Tema 3 – Spring Services: Lógica de negocio e ID

## Grado en Ingeniería Informática en Tecnologías de la Información

Departamento de Ingeniería de Sistemas Informáticos y Telemáticos

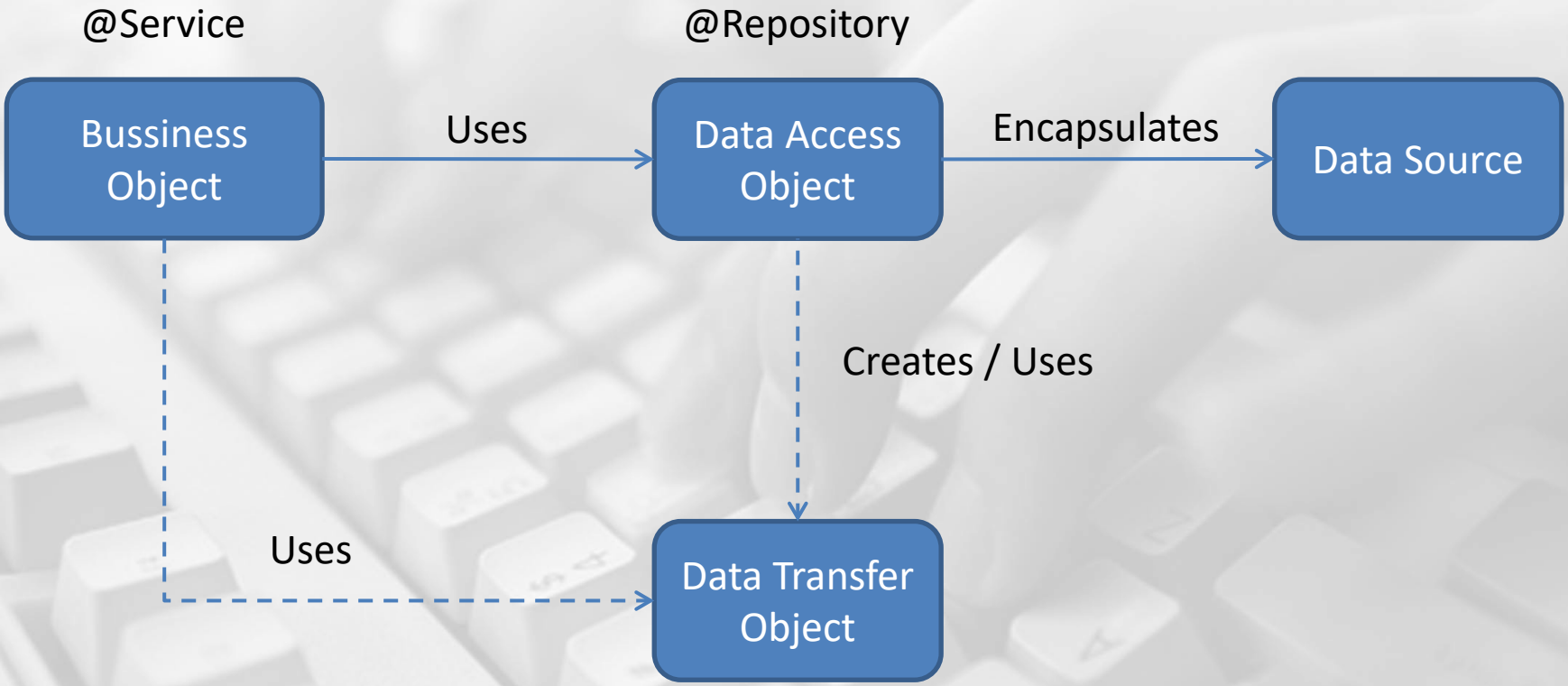
Área de Lenguajes y Sistemas Informáticos

Dr. Luis V. Calderita

# Antecedentes

- Hasta el momento hemos desarrollado el nivel de acceso a datos de nuestra aplicación web
- Ahora vamos a desarrollar un nivel superior que se encargue de manejar la lógica de negocio de nuestra aplicación siguiendo el patrón DAO
- Este nuevo nivel o capa se conoce como capa de servicios, capa de lógica de negocios, capa de servicio de negocios…

# DAO Pattern


@Service
@Repository
Bussiness Object
Uses
Data Access Object
Encapsulates
Data Source
Creates / Uses
Uses
Data Transfer Object

# Introducción

- Capa de Servicios
  - Incluye los objetos que se encargan de implementar la lógica del negocio
  - Son los objetos que desarrollan la funcionalidad de la aplicación
  - Se conectan con la capa de acceso a datos para realizar operaciones
  - Reciben peticiones o llamadas de la capa de presentación y transmiten de vuelta la información solicitada

# Capa de Servicios en Spring

- Los servicios son *Beans* y son un subtipo del estereotipo @Component
- La anotación utilizada es **@Service**
- Una clase anotada con @Service se convertirá en un servicio dentro del contenedor de dependencias de Spring
- Normalmente implementan una interfaz. La clase que implementa la interfaz es la que se anota con @Service

# Declarando servicios en Spring

- Interfaz

```
public interface UsuarioService {
    //
}
```

- Implementación

```
import org.springframework.stereotype.Service;
@Service
public class UsuarioServiceImpl implements UsuarioService {
```

# Implementación de un servicio

- Crear un sub-paquete para los servicios
- Crear una interfaz y su implementación para el servicio
  - Anotar la implementación con @Service
- Normalmente un Servicio incluye el acceso a uno o varios repositorios
  - Los repositorios suelen ser inicializados en el constructor

# Ejemplo básico de un servicio

```java
import org.springframework.stereotype.Service;
@Service
public class UsuarioServiceImpl implements UsuarioService {

    private final UsuarioRepository usuarioRepository;

    @Autowired //No es estrictamente necesaria
    public UsuarioServiceImpl(UsuarioRepository usuarioRepository) {
        // TODO Auto-generated constructor stub
        System.out.println("\t Constructor UsuarioServiceImpl ");
        this.usuarioRepository = usuarioRepository;
    }
```

- @Autowired para inyectar la dependencia del repositorio. Si la clase es un *Bean,* y hay un solo constructor Spring lo usa y lo "autowired" por defecto. No hace falta anotarlo
- **Aviso**: no tiene porqué existir un servicio por cada repositorio

# Practicando

- Extender el ejemplo del repositorio para el One-to-Many del CV y añadir un servicio para el Usuario
  - Incluye, de alguna forma, solamente los métodos o tests de **creación**. De forma que se “inicialice” el contenido de la BD con algunos datos
- Actualiza el fichero properties (ver siguiente diapositiva) y ejecuta la app-web
  - Mientras está en ejecución accede a la BD y comprueba que todo es correcto

# Acceso a la BD en tiempo de ejecución

- Incluir en el fichero *application.properties*

```
# Database URL en memoria
spring.datasource.url=jdbc:h2:mem:luiky
#Enable H2 console (http://localhost:8080/h2-console/)
spring.h2.console.enabled=true
# Database Username (sa usuario por defecto)
spring.datasource.username=sa
# Database Password (vacia por defecto)
spring.datasource.password=
```

- Ejecución normal de la app-web (sin tests):
  - Consola H2 http://localhost:8080/h2-console

# Recursos

- Patrón DAO
  - https://www.oracle.com/java/technologies/dataaccessobject.html
- H2 Tutorial
  - https://www.h2database.com/html/tutorial.html