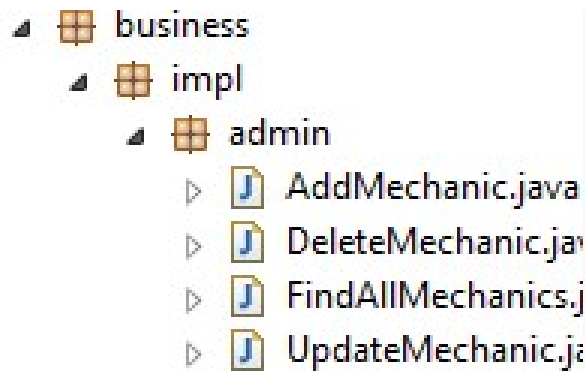


Desarrollo de aplicaciones

Repositorios de Información

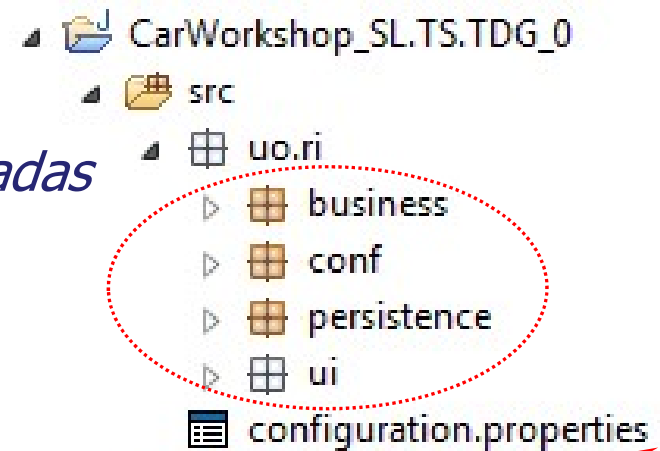
Introducción

- Vamos a adaptar el diseño de SL.TS.TDG_0
- Aprovecharemos la arquitectura:
 - Capas separadas
 - Factorías entre capas
 - SQL externalizado
 - Transaction Script para la lógica

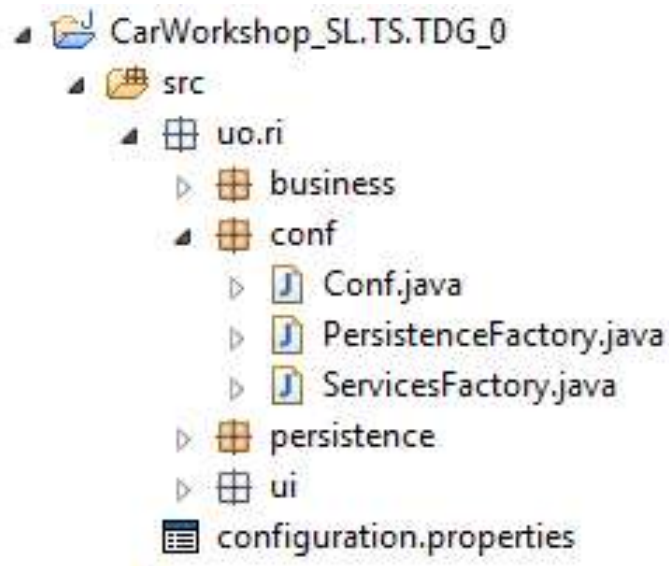


Transaction Script para la lógica

Capas separadas



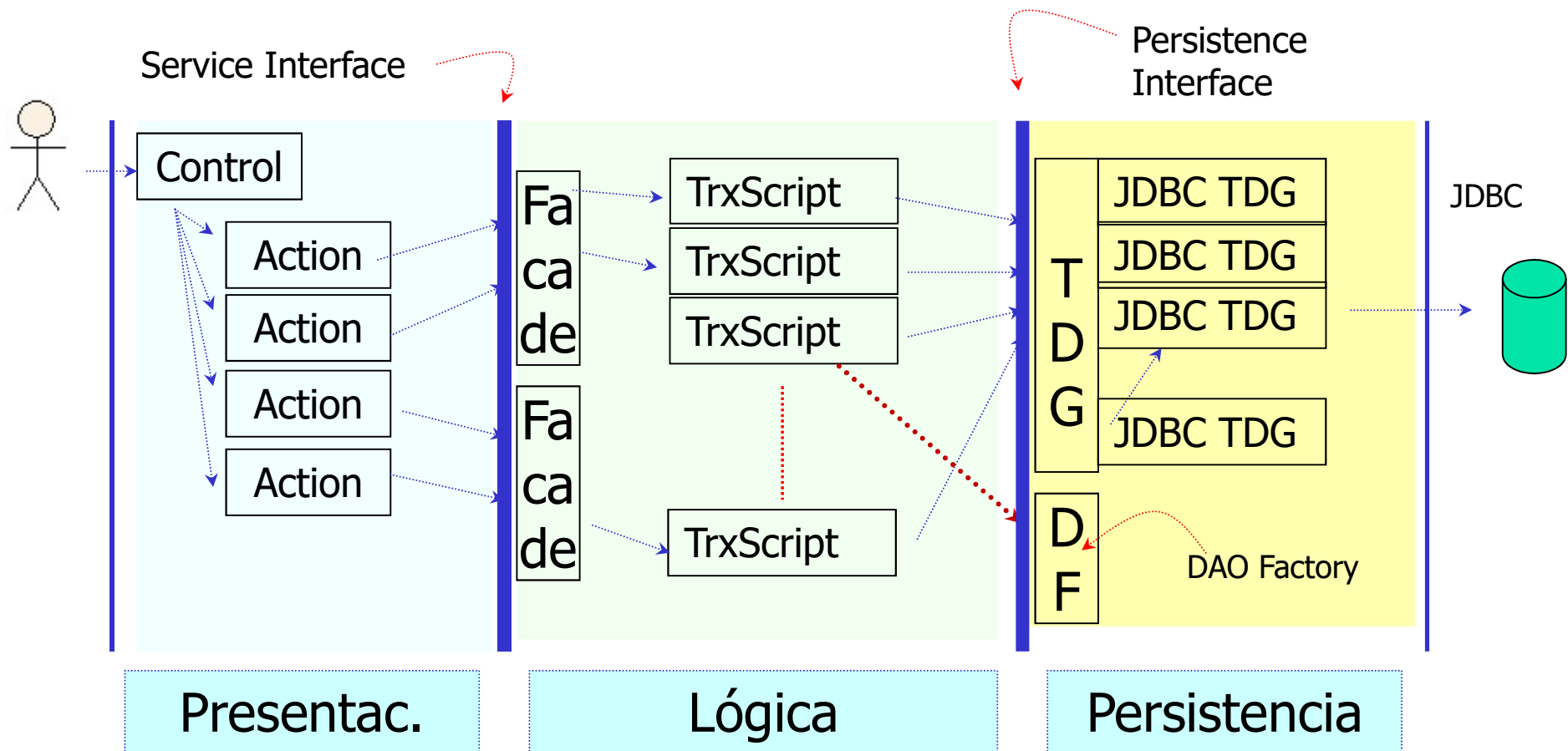
SQL externalizado



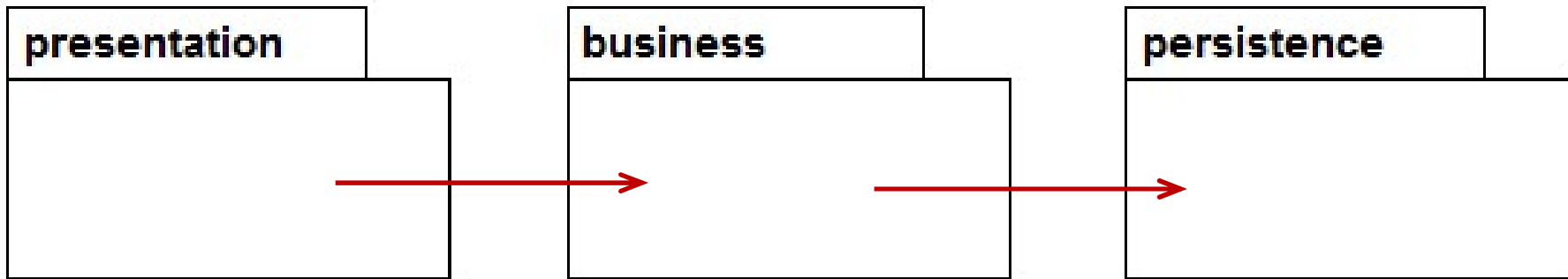
Factorías entre capas

SL.TS.TDG_0

Solución en capas



Dependencias



- Las dependencias deberían ir hacia los paquetes más importantes

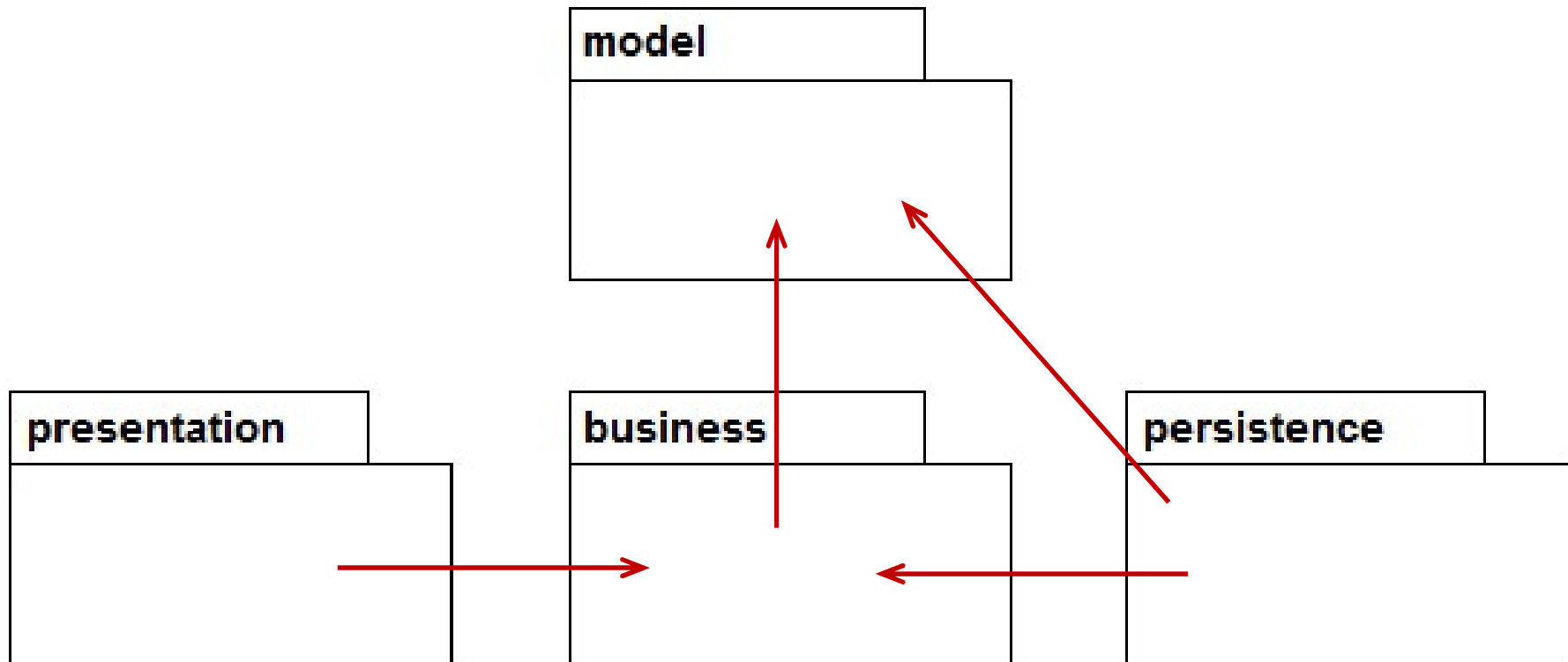
¿persistencia?

La persistencia es un detalle...

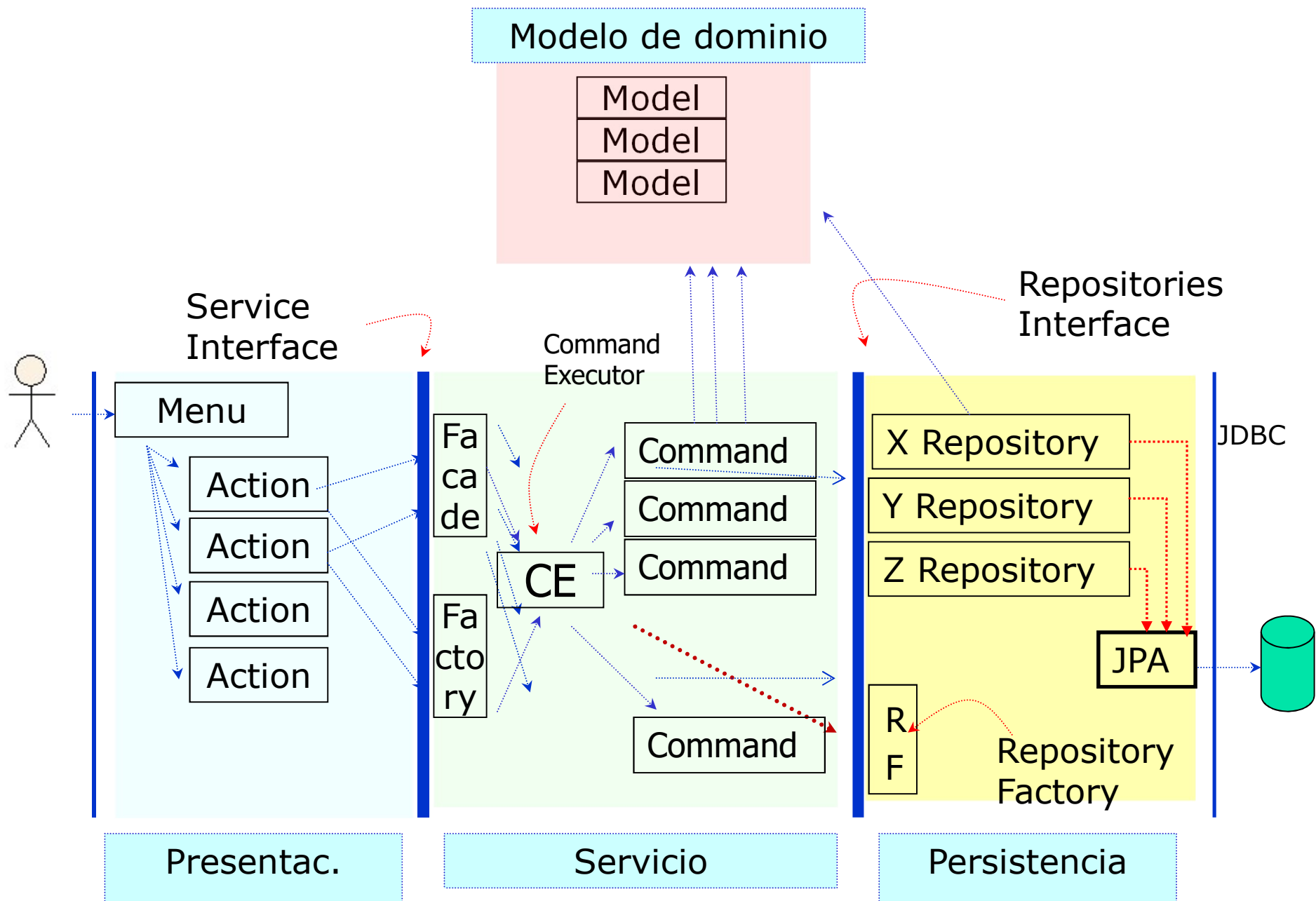
Ahora tendremos...

- Lógica en las clases de dominio
 - No toda, pero sí la mayor parte y la fundamental
 - Los transaction script aligeran, se transforman en comandos (patrón Command)
- El mapeador hace persistencia
 - La capa de persistencia casi desaparece
 - Se reduce a métodos de consulta
 - Consultas externalizadas en orm.xml

Dependencias



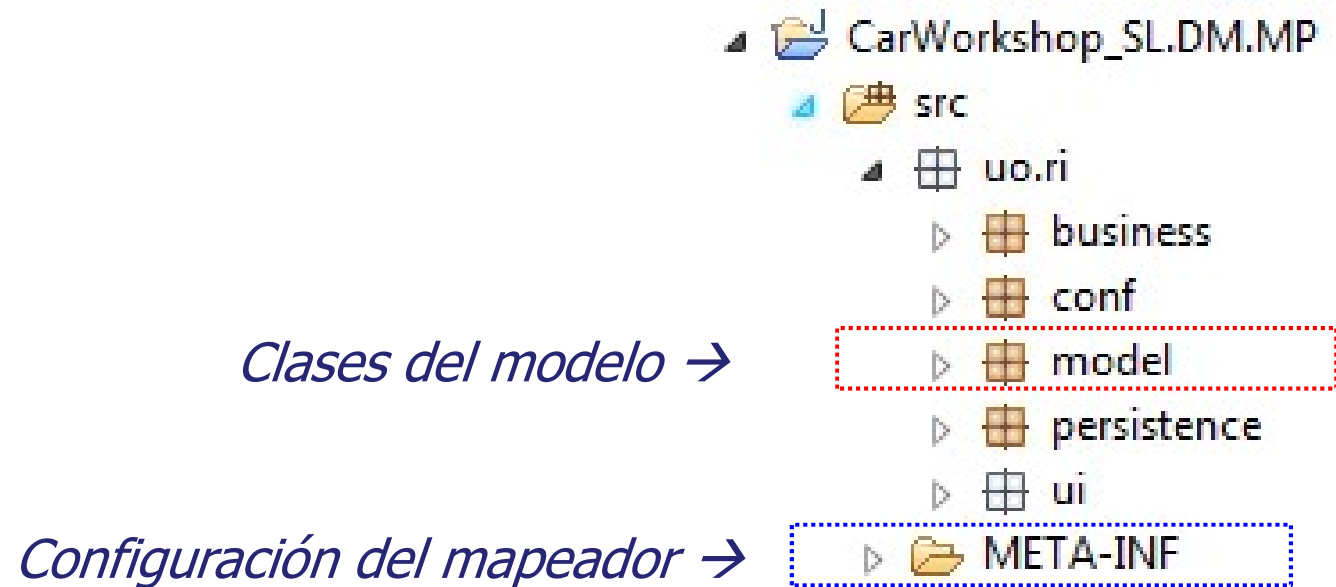
- El modelo de dominio es el más importante



Cambios en el diseño

- Capa del modelo
 - Capa suprema
 - Sin dependencias de ninguna otra capa
- Cambia la comunicación entre capas
 - Ya no se pasan `Maps<String, Object>`
 - Cambian interfaces de la capa de servicio
- Transaction Script refactorizados → comandos
 - Lógica en el modelo
- Persistencia con Repositorios
 - Interfaz tipo colección y consultas

Capa de modelo



Cambian interfaces de servicio

```
public interface AdminService {
```

En SL.TS.TDG_0

```
    void newMechanic(String nombre, String apellidos);
```

```
    void deleteMechanic(Long idMecanico) throws BusinessException;
```

```
    void updateMechanic(Long id, String nombre, String apellidos) throws BusinessException;
```

```
    List<Map<String, Object>> findAllMechanics();
```

```
public interface AdminService {
```

Se pasan objetos (patrón DTO)

```
    void addMechanic(MechanicDto mecanico) throws BusinessException;
```

```
    void deleteMechanic(Long idMechanicDto) throws BusinessException;
```

```
    void updateMechanic(MechanicDto mecanico) throws BusinessException;
```

```
    MechanicDto findMechanicById(Long id) throws BusinessException;
```

```
    List<MechanicDto> findAllMechanics() throws BusinessException;
```

Patrón DTO Data Transfer Object

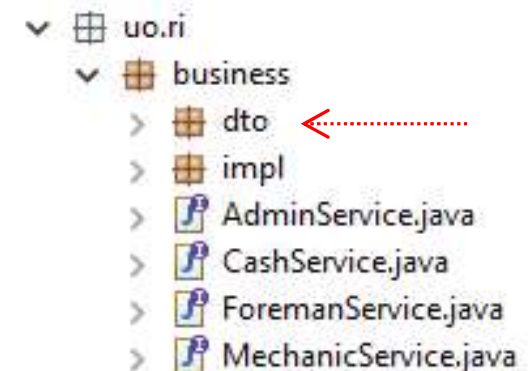
- Objetos simples → contenedor de datos
- Usados entre presentación y servicio
- Sólo datos, sin lógica

```
public class ClientDto {  
  
    public Long id;  
    public String dni;  
    public String name;  
    public String surname;  
    public Address address;  
    public String phone;  
    public String email;  
  
}
```

dic.-17

```
public class MechanicDto {  
  
    public Long id;  
    public String dni;  
    public String name;  
    public String surname;  
  
}
```

Alberto MFA alb@uniovi.es



TS se convierten en comandos

```
public AddMechanic(String nombre, String apellidos) {  
    this.nombre = nombre;  
    this.apellidos = apellidos;  
}
```

En SL.TS.TDG_0

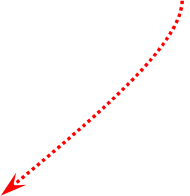
```
public void execute() {  
    try {  
        c = Jdbc.getConnection();  
  
        MecanicosGateway db = PersistenceFactory.getMecanicoGateway();  
  
        db.setConnection(c);  
        db.save(nombre, apellidos);  
  
    } catch (SQLException e) {  
        throw new RuntimeException(e);  
    }  
    finally {  
        Jdbc.close(c);  
    }  
}
```

TS se convierten en comandos

```
public AddMechanic(Mecanico mecanico) {  
    this.mecanico = mecanico;  
}  
  
public Object execute() {  
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("caveatemptor");  
    EntityManager em = emf.createEntityManager();  
    EntityTransaction trx = em.getTransaction();  
  
    em.persist( mecanico );  
  
    trx.commit();  
    em.close();  
  
    return null;  
}
```

Con modelo de dominio

```
public AddMechanic(MechanicDto mecanico) {  
    this.mecanico = mecanico;  
}  
  
@Override  
public Void execute() {  
    Mecanico m = DtoAssembler.toEntity( mecanico );  
    repository.add( m );  
    return null;  
}
```





`public class CreateInvoiceFor {`

← *En SL.TS.TDG_0*

Ejemplo crear factura

Con modelo de dominio

```
public class CreateInvoiceFor implements Command<InvoiceDto> {

    private List<Long> idsAveria;
    private AveriaRepository avrRepo = Factory.repository.forAveria();
    private FacturaRepository fctrRepo = Factory.repository.forFactura();

    public CreateInvoiceFor(List<Long> idsAveria) {
        this.idsAveria = idsAveria;
    }

    @Override
    public InvoiceDto execute() throws BusinessException {

        List<Averia> averias = avrRepo.findByIds( idsAveria );
        Long invoiceNumber = fctrRepo.getNextInvoiceNumber();

        Factura factura = new Factura( invoiceNumber, averias );
        fctrRepo.add( factura );

        return DtoAssembler.toDto( factura );
    }
}
```


Repositorios

Almacén de objetos

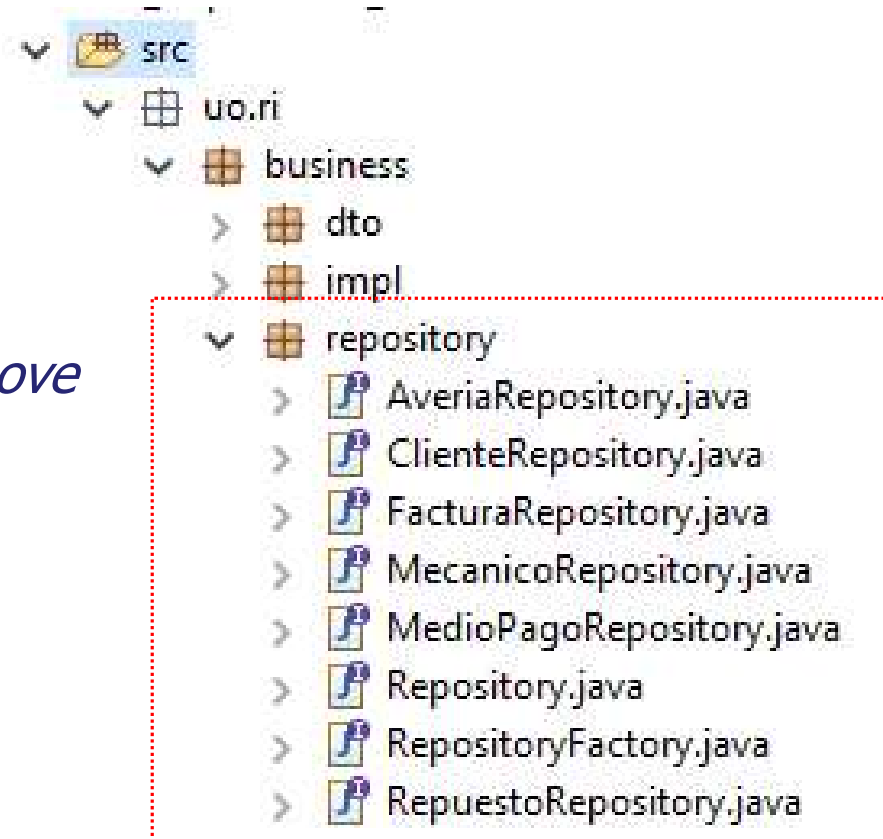
Interfaz tipo colección: add, remove

!!!No hay update!!!

```
public interface Repository<T> {  
    void add(T t);  
    void remove(T t);  
    T findById(Long id);  
    List<T> findAll();  
}
```

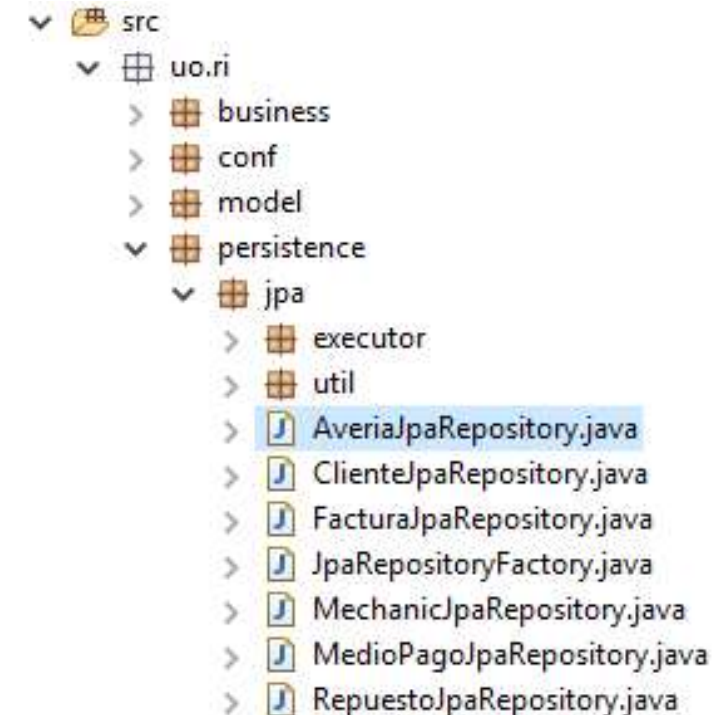
+ consultas

```
public interface AveriaRepository extends Repository<Averia>{  
  
    List<Averia> findByIds(List<Long> idsAveria);  
    List<Averia> findNoFacturadasByDni(String dni);  
}
```



Capa de persistencia: impl de repositorios

*Las implementaciones de repositorios
resuelven métodos de consulta usando el
mapeador*



```
public class AveriaJpaRepository implements AveriaRepository {  
  
    @Override  
    public List<Averia> findByIds(List<Long> idsAveria) {  
        return Jpa.getManager()  
            .createNamedQuery("Averia.findByIds", Averia.class)  
            .setParameter( 1, idsAveria )  
            .getResultList();  
    }  
}
```

Un paso más allá...

- Centralizando el control de transacciones
 - Y si se necesita, el de acceso, el de auditoría, etc.
- Eliminar código repetitivo de los TS (Transaction Script)

Eliminar código repetitivo

```
public class UpdateMechanic {  
  
    private Mecanico mecanico;  
  
    public UpdateMechanic(Mecanico mecanico) {}  
  
    public Object execute() throws BusinessException {  
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("car  
        EntityManager em = emf.createEntityManager();  
        EntityTransaction trx = em.getTransaction();  
  
        Mecanico m = em.merge( mecanico );  
  
        trx.commit();  
        em.close();  
  
        return m;  
    }  
}
```

Eliminar código repetitivo

```
public class AddMechanic {  
  
    private Mecanico mecanico;  
  
    public AddMechanic(Mecanico mecanico) {}  
  
    public Object execute() {  
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("car  
        EntityManager em = emf.createEntityManager();  
        EntityTransaction trx = em.getTransaction();  
  
        em.persist( mecanico );  
  
        trx.commit();  
        em.close();  
  
        return null;  
    }  
}
```

Compara con la anterior... ¿qué cambia?

Eliminar código repetitivo

```
public class AddMechanic {  
  
    private Mecanico mecanico;  
  
    public AddMechanic(Mecanico mecanico) {}  
  
    public Object execute() {  
        EntityManagerFactory emf = Persistence.createEntityManagerFa  
        EntityManager em = emf.createEntityManager();  
        EntityTransaction trx = em.getTransaction();  
  
        em.persist( mecanico );  
  
        trx.commit();  
        em.close();  
  
        return null;  
    }  
}
```

*Ese código se repite una y otra vez...
Vamos a factorizarlo*

Centralizar transacciones

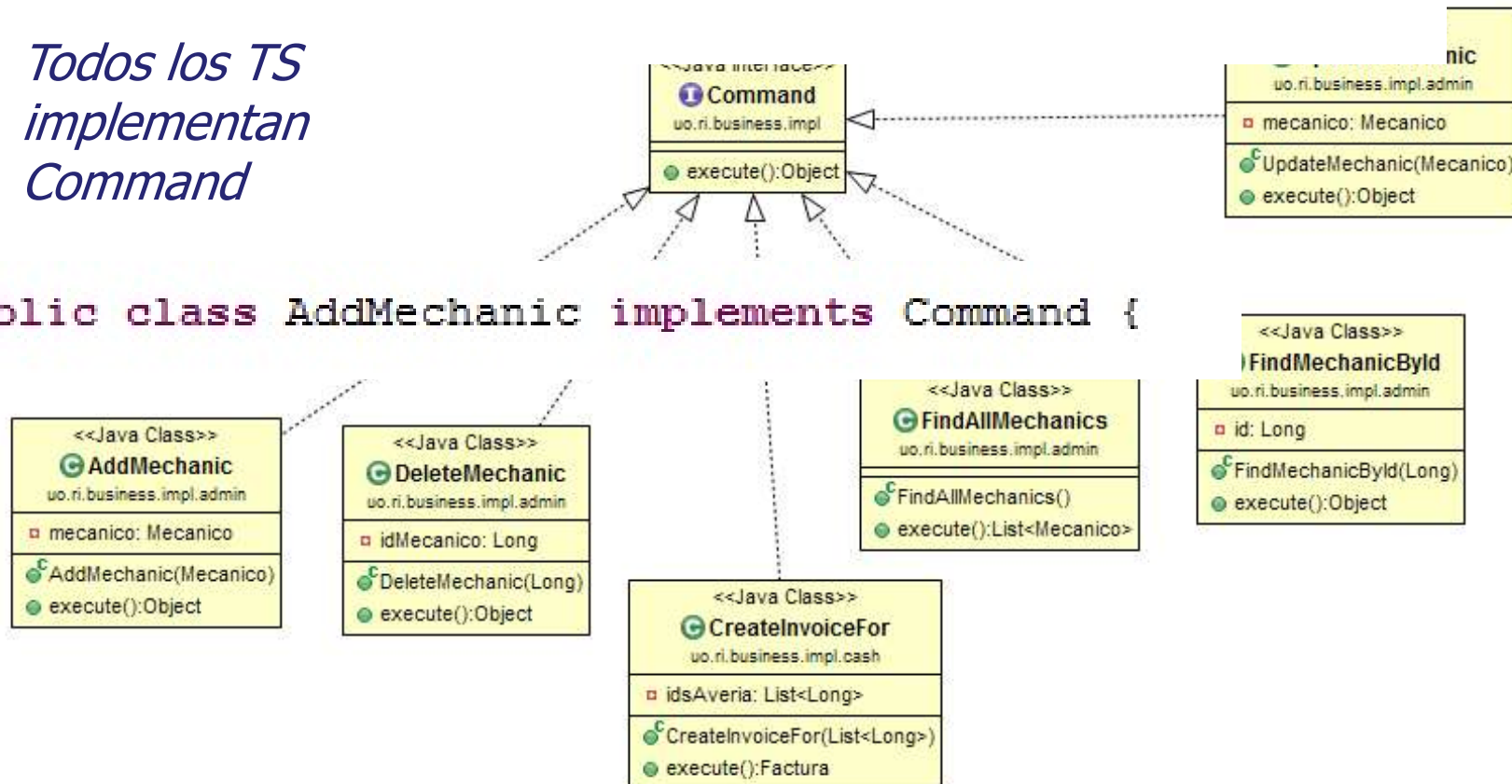
- Pasos:
 - Uniformizar los Transaction Script
 - Interfaz Command
 - Extraer el control de trx a una única clase
 - Command Executor
 - Modificar las implementaciones de las fachadas
 - AdminServiceImpl, etc.
 - Clase de Utilidad para gestionar el EntityManager

Uniformizar los TS

```
public interface Command {  
  
    Object execute() throws BusinessException;  
  
}
```

Todos los TS implementan Command

```
public class AddMechanic implements Command {
```



Extraer control a una única clase

```
public class JpaCommandExecutorImpl implements Executor {
```

```
    @Override
```

```
    public Object execute(Command command) throws BusinessException {
```

```
        Object res = null;
```

```
        EntityManager em = Jpa.createEntityManager();
```

```
        EntityTransaction tx = em.getTransaction();
```

```
        tx.begin();
```

```
        try {
```

```
            res = command.execute();
```

```
            tx.commit();
```

```
        } catch (BusinessException bex) {
```

```
            rollback(tx);
```

```
            throw bex;
```

```
        } catch (RuntimeException rex) {
```

```
            rollback(tx);
```

```
            throw rex;
```

```
        } finally {
```

```
            close(em);
```

```
        }
```

```
        return res;
```

```
    }
```

*Gestión de excepciones y
transacción centralizado*

```
public interface Executor {
```

```
    Object execute(Command command) throws BusinessException;
```

```
}
```


Modificar implement. de fachada

*Gestión de excepciones y
transacción centralizado*

```
public class AdminServiceImpl implements AdminService {  
  
    private Executor executor = CommandExecutorFactory.getExecutor();  
  
    @Override  
    public void addMechanic(MechanicDto mecanico) throws BusinessException {  
        executor.execute( new AddMechanic( mecanico ) );  
    }  
  
    @Override  
    public void updateMechanic(MechanicDto mecanico) throws BusinessException {  
        executor.execute( new UpdateMechanic( mecanico ) );  
    }  
  
    @Override  
    public void deleteMechanic(Long idMecanico) throws BusinessException {  
        executor.execute( new DeleteMechanic(idMecanico) );  
    }  
}
```

Clase de utilidad para EM

```
public class Jpa {  
    public static EntityManager getManager() {  
  
    public static EntityManager createEntityManager() {
```

- **createEntityManager()**
 - Crea uno nuevo
 - Solo invocado por el commandExecutor
- **getManager()**
 - Invocado por el resto de clases
 - Devuelve el contexto de persistencia actual

```

public class AddMechanic implements Command<Void> {

    private MechanicDto mecanico;
    private MecanicoRepository repository = Factory.repository.forMechanic();

    public AddMechanic(MechanicDto mecanico) {
        this.mecanico = mecanico;
    }

    @Override
    public Void execute() {
        Mecanico m = DtoAssembler.toEntity( mecanico )
        repository.add( m );
        return null;
    }
}

```

Código sin duplicidades y sin dependencias

```

public class CreateInvoiceFor implements Command<InvoiceDto>{

    private List<Long> idsAveria;
    private AveriaRepository avrsRepo = Factory.repository.forAveria();
    private FacturaRepository fctrsRepo = Factory.repository.forFactura();

    public CreateInvoiceFor(List<Long> idsAveria) {
        this.idsAveria = idsAveria;
    }

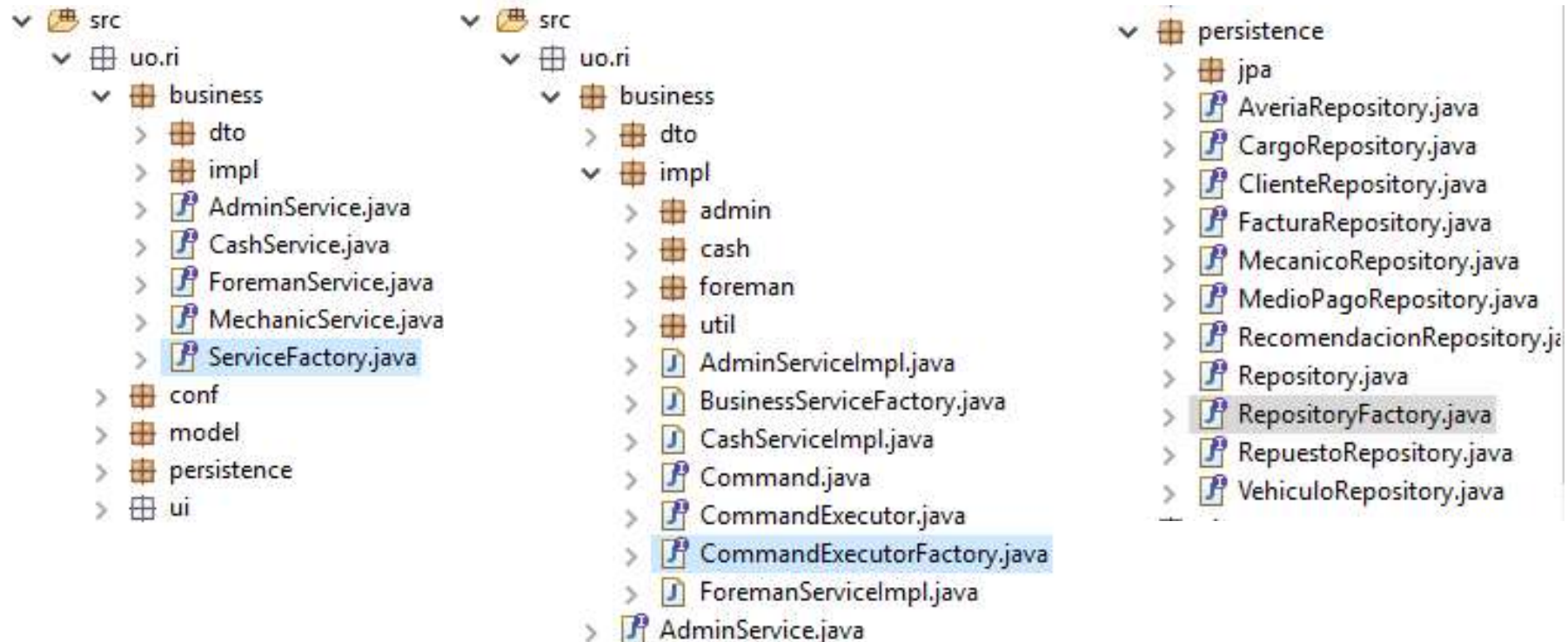
    @Override
    public InvoiceDto execute() throws BusinessException {
        List<Averia> avs = avrsRepo.findByIds( idsAveria );
        Long numero = fctrsRepo.getNextInvoiceNumber();

        Factura f = new Factura(numero, avs);
        fctrsRepo.add( f );

        return DtoAssembler.toDto( f );
    }
}

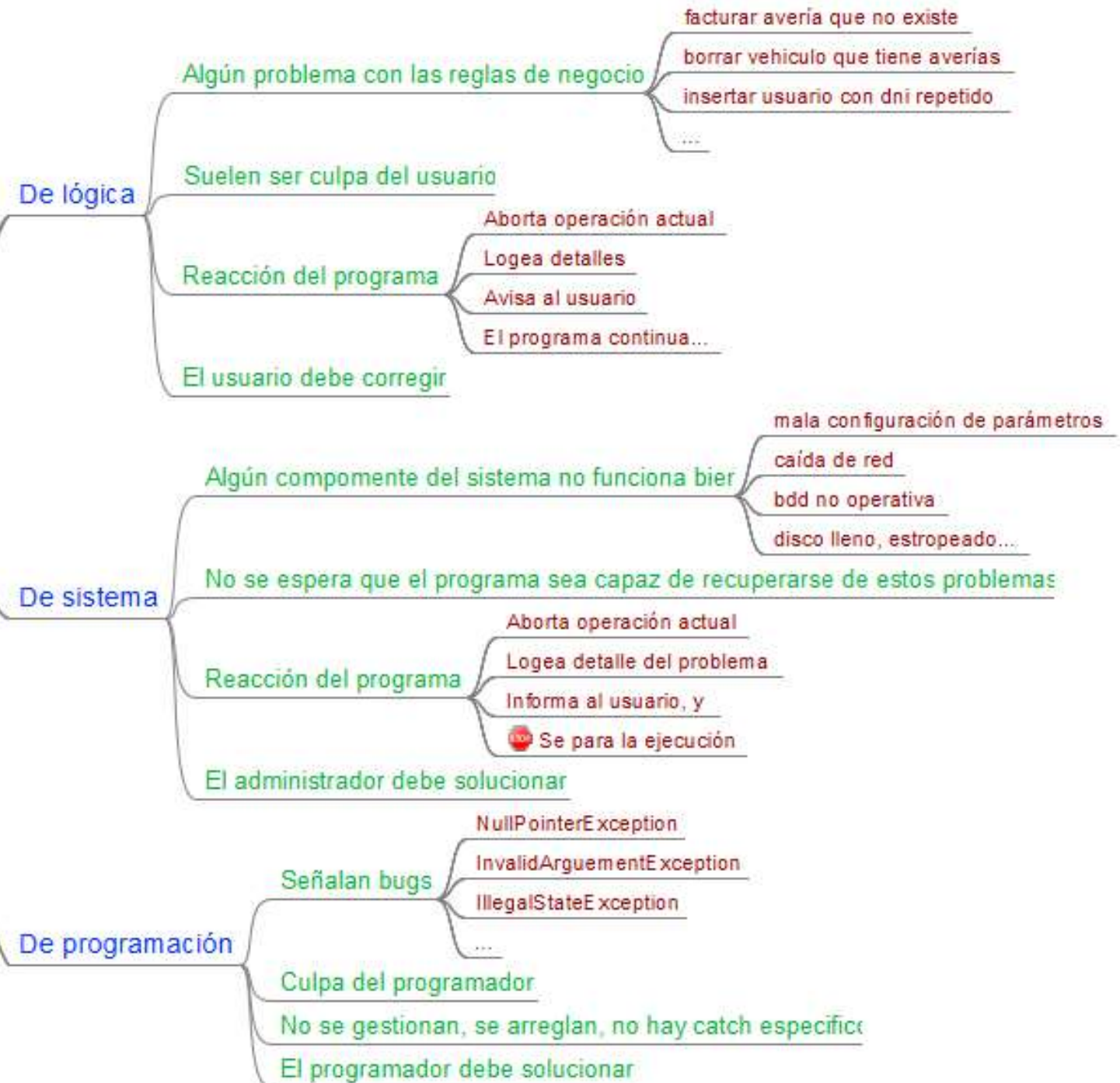
```

Factorías entre capas

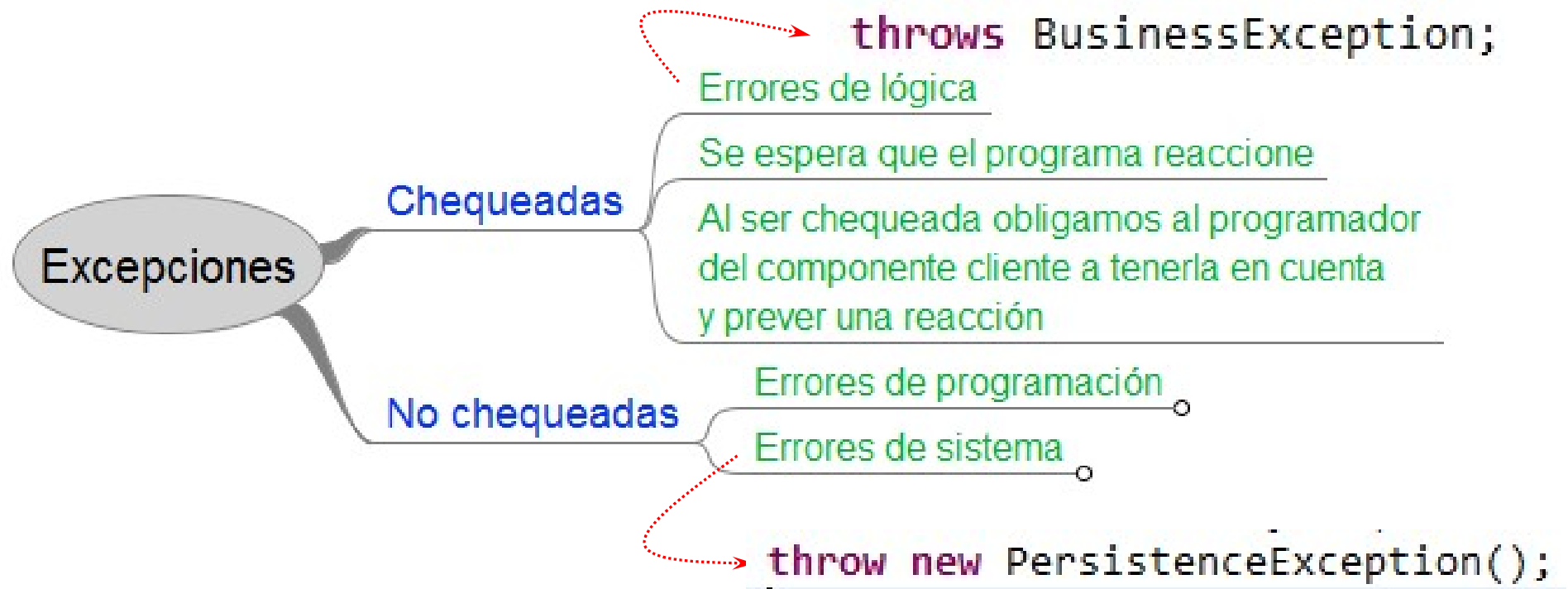


■ Patrón Abstract Factory

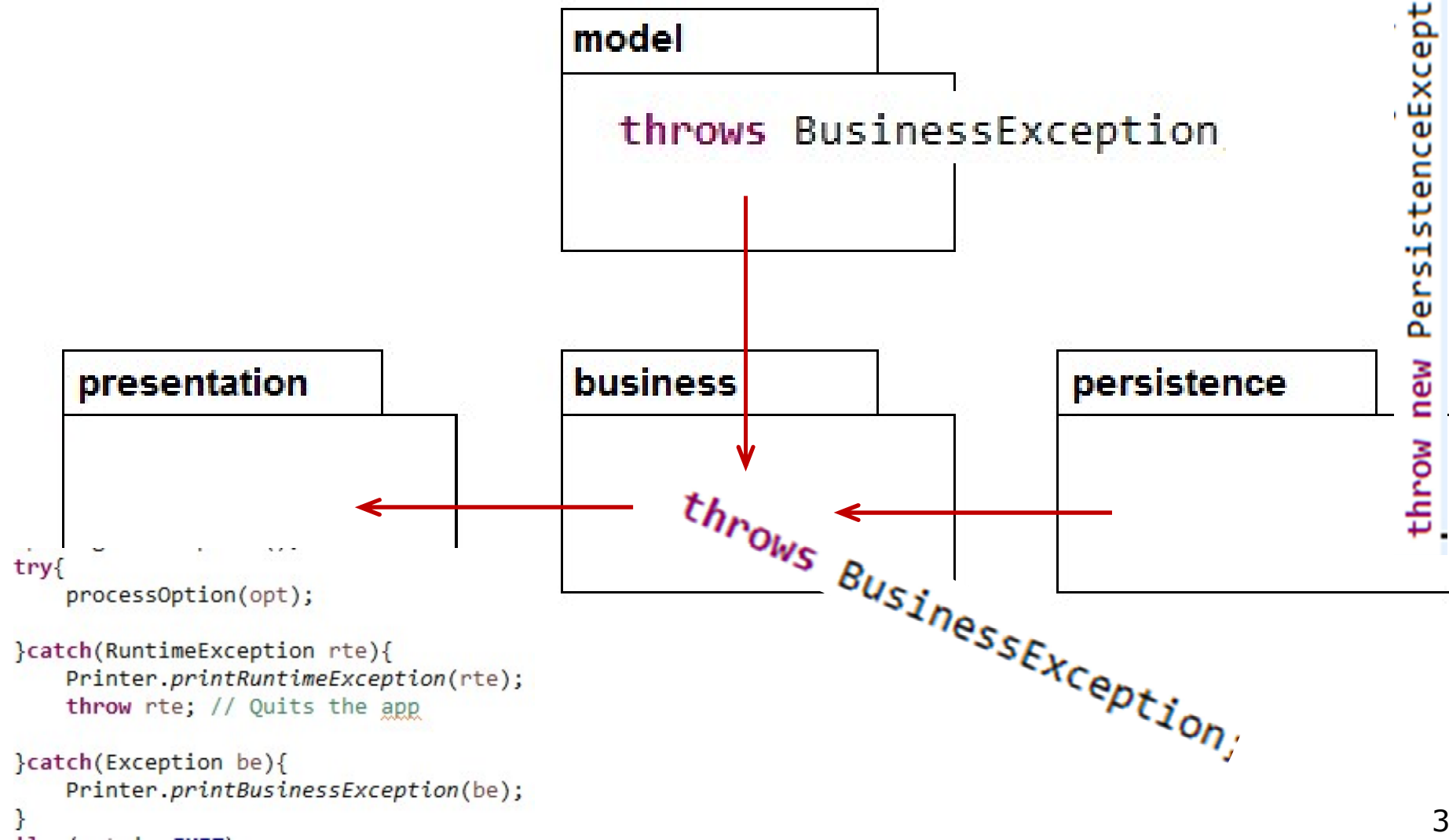
Gestión de errores



Gestión de errores



Gestión de errores



```

public class MainMenu extends BaseMenu {

    public MainMenu() {
        menuOptions = new Object[][] {
            { "Administrador", null },
            { "Gestión de mecánicos",
            { "Gestión de repuestos",
            { "Gestión de tipos de vehículo",
            { "Gestión de bonos",
        };
    }

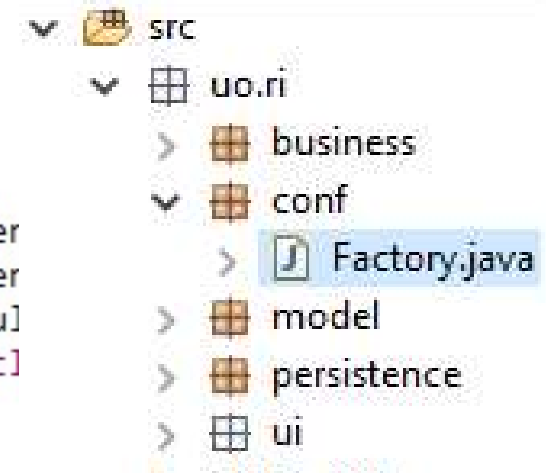
    public static void main(String[] args) {
        new MainMenu().configure().execute();
    }

```

```

MecanicosMer
RepuestosMer
TiposVehicul
BonosMenu.cl

```



Configuración

```

private MainMenu configure() {
    Factory.service = new BusinessServiceFactory();
    Factory.repository = new JpaRepositoryFactory();
    Factory.executor = new JpaExecutorFactory();

    return this;
}

```

```

public class Factory {

    public static RepositoryFactory repository;
    public static ServiceFactory service;
    public static CommandExecutorFactory executor;

}

```