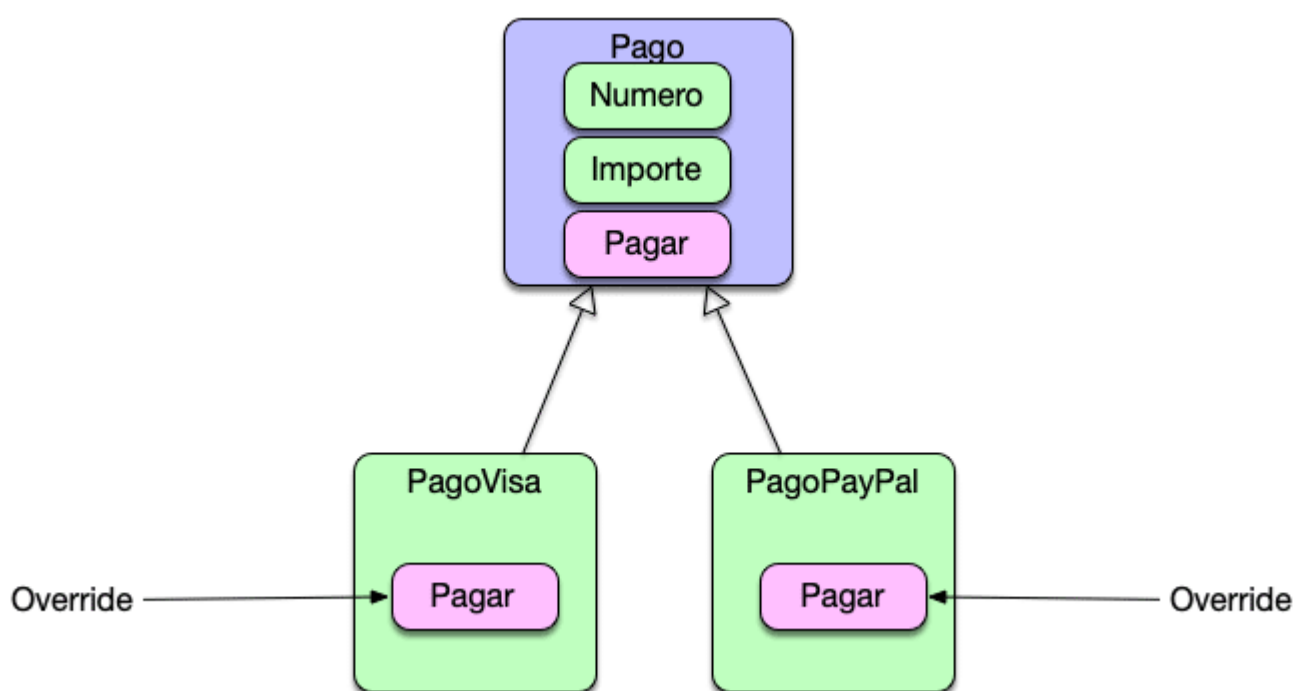


El concepto de JPA polymorphic query , es uno de los conceptos más clásicos cuando nos encontramos trabajando con JPA y herencia . En estas casuísticas es habitual tener que realizar consultas polimórficas que afecten a varias entidades y ejecuten una funcionalidad concreta en un árbol de herencia. Vamos a ver un ejemplo sencillo usando las clases Pago , PagoPayPal y PagoVisa que son clases que se encuentran relacionadas en una estructura de herencia siendo la clase Pago la padre de las otras dos .



JPA y Herencia

Para poder construir consultas polimórficas primero tendremos que diseñar una estructura jerárquica con JPA y las anotaciones orientadas a **Single Table Inheritance** . Vamos a verlo en código:

```
package com.arquitecturajava;
```

```
import javax.persistence.DiscriminatorColumn;  
import javax.persistence.Entity;
```

```
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.Table;

@Entity
@Table(name="pagos")
@DiscriminatorColumn(name="tipo")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public abstract class Pago {
    @Id
    private int numero;
    private double importe;
    public Pago(int numero, double importe) {
        super();
        this.numero = numero;
        this.importe = importe;
    }
    public int getNumero() {
        return numero;
    }
    public void setNumero(int numero) {
        this.numero = numero;
    }
    public double getImporte() {
        return importe;
    }
    public void setImporte(double importe) {
        this.importe = importe;
    }
    public abstract void pagar();
}
```

```
}
```

Como podemos ver esta es una clase Abstracta que almacena las propiedades y define un método pagar vacío para que otras clases hijas lo implementen.

```
package com.arquitecturajava;
```

```
import javax.persistence.DiscriminatorValue;
```

```
import javax.persistence.Entity;
```

```
@Entity
```

```
@DiscriminatorValue(value="paypal")
```

```
public class PagoPayPal extends Pago {
```

```
    private String cuenta;
```

```
    public PagoPayPal(int numero, double importe,String cuenta) {
```

```
        super(numero, importe);
```

```
        this.cuenta=cuenta;
```

```
        // TODO Auto-generated constructor stub
```

```
    }
```

```
    public String getCuenta() {
```

```
        return cuenta;
```

```
    }
```

```
    public void setCuenta(String cuenta) {
```

```
        this.cuenta = cuenta;
```

```
    }
```

```
@Override
```

```
public void pagar() {
```

```
    System.out.println("estoy pagando con paypal");
```

```
}
```

```

}

package com.arquitecturajava;

import javax.persistence.DiscriminatorColumn;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.Table;

@Entity
@Table(name="pagos")
@DiscriminatorColumn(name="tipo")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public abstract class Pago {
    @Id
    private int numero;
    private double importe;
    public Pago(int numero, double importe) {
        super();
        this.numero = numero;
        this.importe = importe;
    }
    public int getNumero() {
        return numero;
    }
    public void setNumero(int numero) {
        this.numero = numero;
    }
    public double getImporte() {
        return importe;
    }
}

```

```

    }
    public void setImporte(double importe) {
        this.importe = importe;
    }
    public abstract void pagar();
}

```

Acabamos de construir las clases que directamente heredan de la clase Pago y sobreescriben el método pagar(). Es momento de construir un programa main que se encargue de ejecutar una consulta contra la base de datos y obtenernos una entidad :

```

package com.arquitecturajava;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import javax.persistence.TypedQuery;

public class Principal19 {

    public static void main(String[] args) {
        EntityManagerFactory emf=
Persistence.createEntityManagerFactory("biblioteca");
        EntityManager em= emf.createEntityManager();
        PagoVisa pv= new PagoVisa(2, 200, 23456789);
        PagoPayPal pp= new PagoPayPal(3, 200, "cecilio");

        EntityTransaction t= em.getTransaction();
        t.begin();
    }
}

```

```

em.persist(pv);
em.persist(pp);
TypedQuery<PagoVisa> consulta=
em.createQuery("select p from PagoVisa p",PagoVisa.class);
List<PagoVisa> lista= consulta.getResultList();
for (PagoVisa p: lista) {
    System.out.println(p.getTarjeta());
    p.pagar();
}
t.commit();

}

}

```

En este caso he creado dos objetos y los he insertado en la base de datos para posteriormente realizar una consulta con el PagoVisa. Si ejecuto el código el resultado es el esperado:



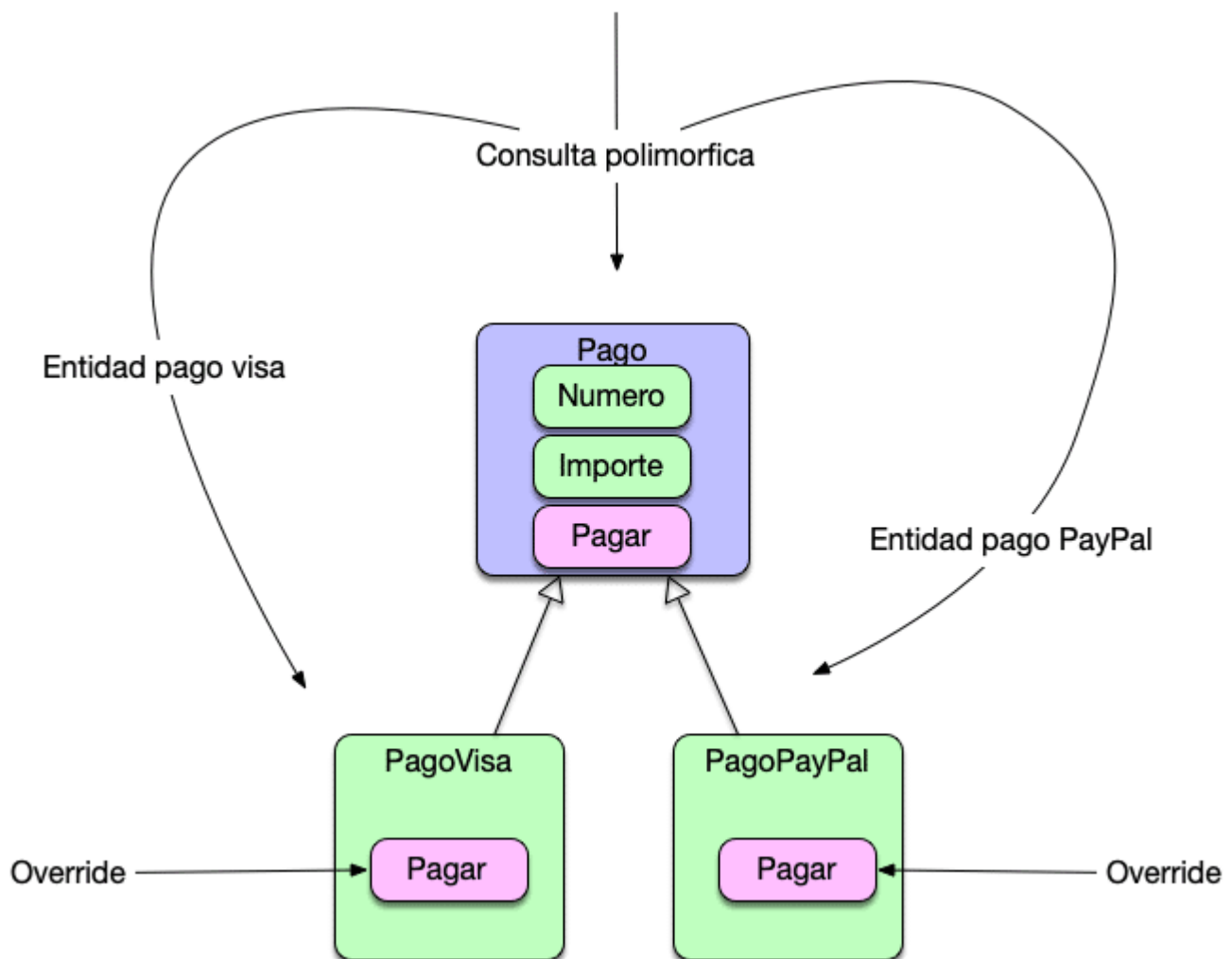
```

23456789
estoy pagando con visa

```

JPA Polymorphic Query

Ahora bien podemos construir un código más flexible que pueda pagar tanto con PagoVisa como con PagoPayPal . Para ello cambiaremos la consulta para que sea una consulta polimórfica y pueda gestionar las dos entidades al mismo tiempo:

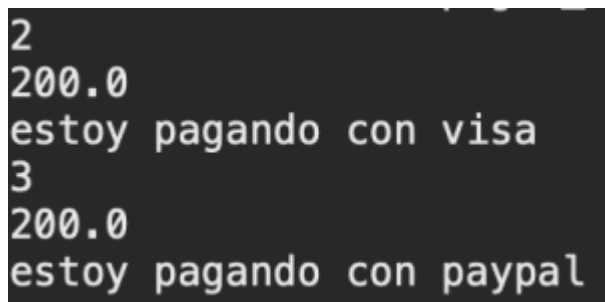


Veamos el código:

```
TypedQuery<Pago> consulta=
    em.createQuery("select p from Pago p where Type(p) IN
(PagoVisa,PagoPayPal)",Pago.class);
List<Pago> lista= consulta.getResultList();
for (Pago p: lista) {
    System.out.println(p.getNumero());
}
```

```
        System.out.println(p.getImporte());  
        p.pagar();  
    }
```

En este caso estamos recorriendo con el concepto de Pago ambas entidades e imprimiendo por la consola los datos de cada pago así como el tipo de pago que se realiza finalmente.



```
2  
200.0  
estoy pagando con visa  
3  
200.0  
estoy pagando con paypal
```

Otros artículos relacionados

1. [JPA Transaction, un concepto importante](#)
2. [JPA Orphan Removal y como usarlo](#)
3. [Eclipse JPA y clases de dominio](#)