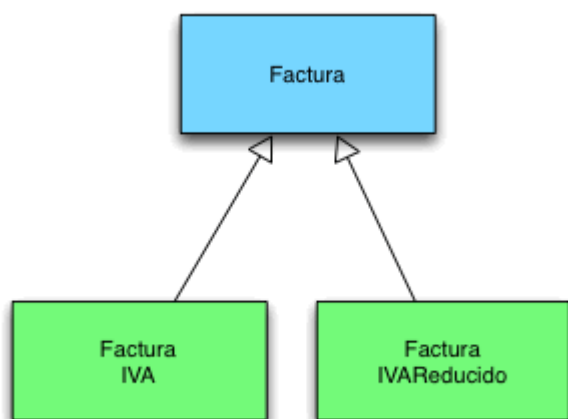


Tabla de Contenidos

- ◆
- [Patron Factory y manejo de Facturas](#)
- [El patron Factory y la Encapsulación](#)
- [Otros artículos relacionados:](#)

El patrón Factory es uno de los patrones fundamentales a nivel de diseño orientado a objeto. Este patrón pertenece al grupo de patrones creacionales y nos simplifica la construcción de una jerarquía de clases y las encapsula. Sin embargo a veces a la gente le cuesta ver como usar este patrón en su código. Vamos a utilizar un ejemplo sencillo en el que tendremos una jerarquía de clases Factura como se muestra a continuación.



Vamos a ver cada clase en código:

```
package com.arquitecturajava;
```

```
public abstract class Factura {
```

```
    private int id;  
    private double importe;  
    public int getId() {  
        return id;  
    }
```

```

    }
    public void setId(int id) {
        this.id = id;
    }
    public double getImporte() {
        return importe;
    }
    public void setImporte(double importe) {
        this.importe = importe;
    }

    public abstract double getImporteIva();
}

package com.arquitecturajava;

public class FacturaIvaReducido extends Factura{

    @Override
    public double getImporteIva() {
        // TODO Auto-generated method stub
        return getImporte()*1.07;
    }

}

package com.arquitecturajava;

public class FacturaIva extends Factura {

    @Override
    public double getImporteIva() {

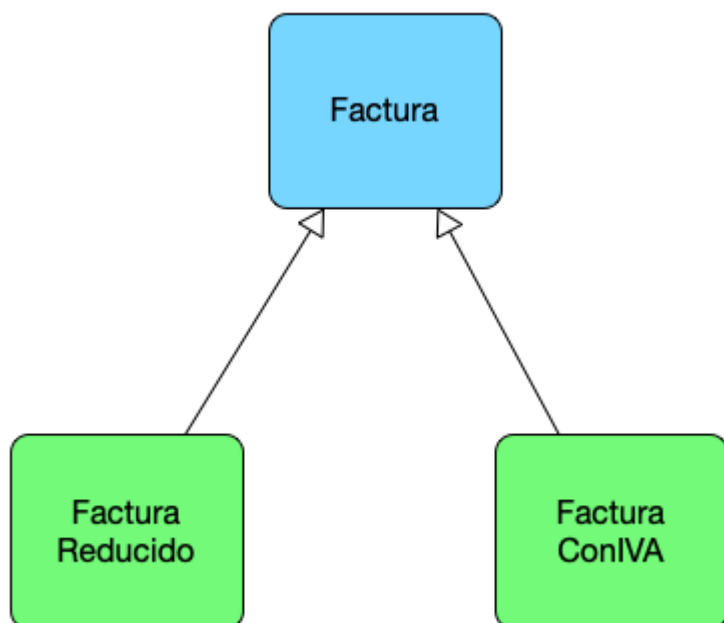
```

```
        // TODO Auto-generated method stub
        return getImporte() * 1.21;
    }

}
```

Patron Factory y manejo de Facturas

Ya disponemos de las tres clases la clase abstracta y sus clases hijas. Como vemos la clase Factura es una clase abstracta de la cual heredan nuestras dos clases concretas que implementan el cálculo del IVA.



Vamos a construir una Factoría para que se encargue de construir ambos objetos de la jerarquía.

```
package com.arquitecturajava;

public class FactoriaFacturas {

    public static Factura getFactura(String tipo) {
```

```

        if (tipo.equals("iva")) {

            return new FacturaIva();
        } else {
            return new FacturaIvaReducido();
        }
    }
}

```

Si nos fijamos la clase lo único que hace es instanciar un objeto u otro dependiendo del tipo que le solicitemos. Eso en un principio parece poco práctico. Pero vamos a ver como queda el programa main:

```

package com.arquitecturajava;

public class Principal {

    public static void main(String[] args) {

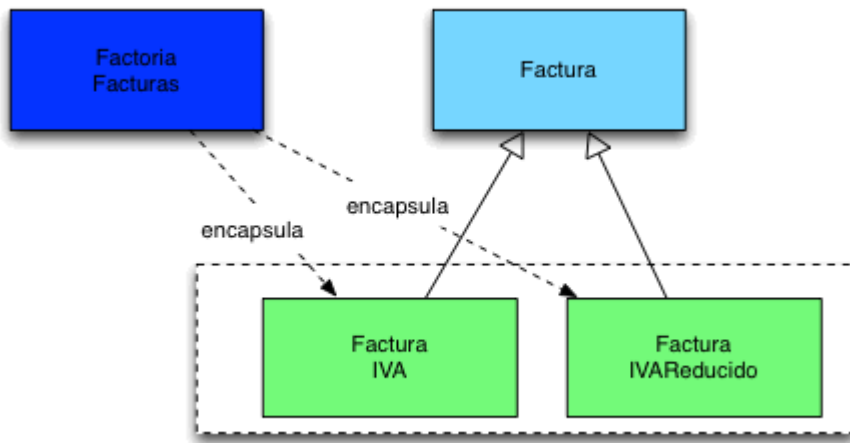
        Factura f = FactoriaFacturas.getFactura("iva");
        f.setId(1);
        f.setImporte(100);
        System.out.println(f.getImporteIva());
    }

}

```

El patron Factory y la Encapsulación

Nos podemos dar cuenta que el programador ya solo tiene que tratar con el concepto de Factura para el la clase FacturaIva y FacturaReducido no existen.



Esto permite una simplificación a la hora de trabajar clara. Es cierto que las Factorias se encargan de generar una jerarquía de clases pero su función fundamental es encapsular una jerarquía de objetos y reducir el conjunto de conceptos con los que trabajamos. Un ejemplo por ejemplo muy clásico es `Calendar.getInstance()` que nos devuelve una de las posibles implementaciones del calendario. No tenemos porque conocer que existe `GregorianCalendar`.

Otros artículos relacionados:

- [Java 8 Factory Pattern y su implementación](#)
- [Java 9 Factory Methods \(List,Set,Map\)](#)
- [Java Polimorfismo, Herencia y simplicidad](#)
- [JPA Polymorphic Query](#)
- [Java Encapsulamiento y reutilización](#)