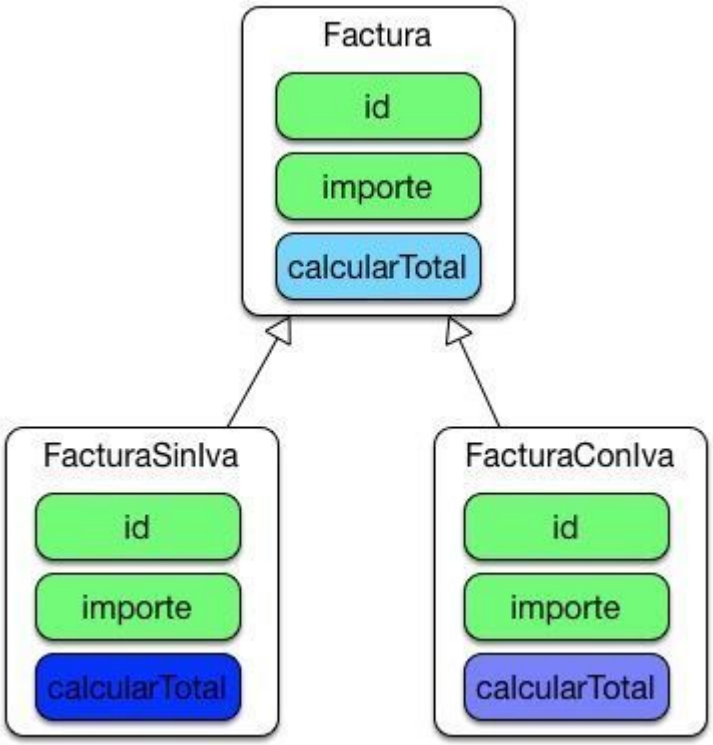
El concepto de Java Override o overriding es uno de los conceptos que cuesta más entender cuando una comienza a trabajar con programación orientada a objeto. ¿Para que sirve el overriding o polimorfismo dinámico? . En Java en muchas ocasiones nos encontramos con situaciones en las cuales tenemos una jerarquía de clases en la que existe diferencia en cuanto a la implementación de alguno de sus métodos comunes. Vamos a usar un ejemplo basado en el concepto de Facturas. Supongamos que disponemos de dos tipos de Facturas. Facturas con IVA y Facturas sin IVA. Ambas tienen como clase padre la clase Factura.



Vamos a ver estas clases implementadas en código:

```
package com.arquitecturajava;

public abstract class Factura {

    private String id;
    private double importe;
```

```
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public double getImporte() {
        return importe;
    }
    public void setImporte(double importe) {
        this.importe = importe;
    }
    public abstract double calcularTotal();
}
```

En primer lugar definimos la clase padre una vez hecho esto abordamos las clases hijas.

```
package com.arquitecturajava;

public class FacturaConIva extends Factura {

    @Override
    public double calcularTotal() {
        return this.getImporte() * 1.21;

    }

}
```

```
package com.arquitecturajava;

public class FacturaSinIva extends Factura {
```

```
    @Override
    public double calcularTotal() {
        return this.getImporte();

    }

}
```

Java Override y Facturas

En este caso la implementación es muy sencilla , el método calcularImporte nos devuelve el importe total. Si la factura es con IVA sera un 21% más cara que si no lo es, así de sencillo. Vamos a ver el programa principal:

```
package com.arquitecturajava;

public class Principal {

    public static void main(String[] args) {

        FacturaConIva f1 = new FacturaConIva();
        f1.setId("1");
        f1.setImporte(100);

        FacturaSinIva f2 = new FacturaSinIva();
        f2.setId("2");
        f2.setImporte(100);

        System.out.println(f1.calcularTotal());
        System.out.println(f2.calcularTotal());
    }
```

```
}
```

Si lo ejecutamos veremos como una Factura se imprime incluyendo el IVA y la otra Factura se imprime con el coste actual.

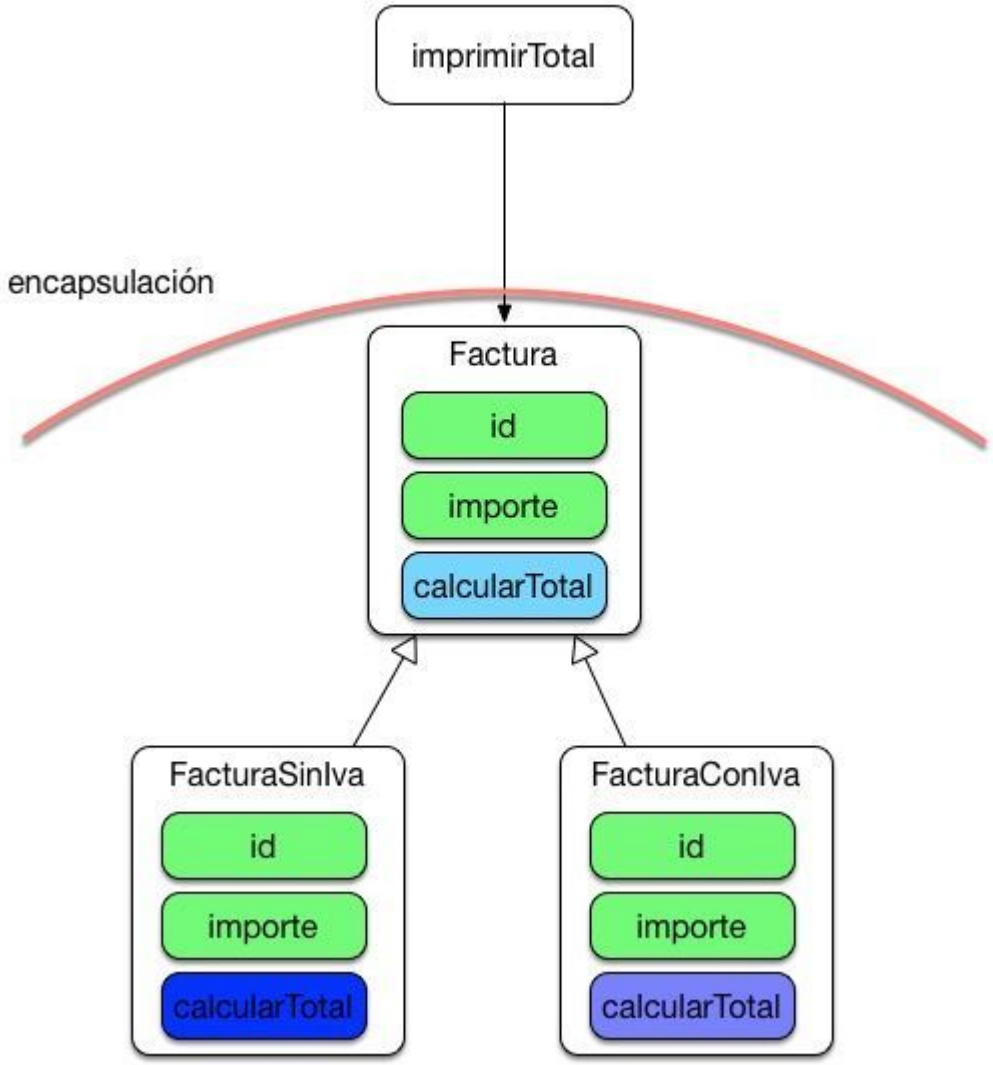
Facturas y métodos

Acabamos de usar un Java Override de la forma correcta sobrecargando los métodos de calcular importe. Sin embargo a mucha gente le cuesta entender que esto sea útil. Vamos a modificar el programa para realizar la misma operación de una forma un poco diferente.

```
package com.arquitecturajava;

public class Principal {

    public static void main(String[] args) {

        FacturaConIva f1 = new FacturaConIva();
        f1.setId("1");
        f1.setImporte(100);
        FacturaSinIva f2 = new FacturaSinIva();
        f2.setId("2");
        f2.setImporte(100);

        imprimirImporte(f1);
        imprimirImporte(f2);
    }
```

```
    public static void imprimirImporte(Factura f) {

        System.out.println(f.calcularTotal());
    }

}
```

El código parece muy similar de hecho el resultado por consola es idéntico. Ahora bien gracias al polimorfísmo la persona que programa el método imprimirImporte no tiene necesidad de conocer la jerarquía de clases Factura. Le es suficiente con conocer el concepto de Factura. Estamos encapsulando conceptos ya que puede haber muchas clases en una jerarquía con similitudes y nosotros nos abstraemos de ella usamos el polimorfismo y nos encargamos de realizar los calculos que serán diferentes dependiendo de la clase en la jerarquía y su implementación.



El uso de Java Override nos ayuda a encapsular conceptos y reducir los que tenemos que conocer haciendo mas sencilla la forma de trabajar con los programas. Mejora por otro lado la gestión de responsabilidades.

Otros artículos relacionados:

- Java override HashCode y curiosidades
- Kotlin Delegación buenas prácticas.
- Java Encapsulamiento y reutilización
- Mis Cursos de Java para desarrolladores