El concepto de Java 8 Factory Pattern es cada día más utilizado ya que nos permite simplificar la implementación del clásico patrón factoría utilizando Java 8 y reduciendo el número de clases. Vamos a ver como implementarlo, para ello partiremos de un ejemplo clásico de gestión de Facturas y calculo de IVAs y lo evolucionaremos a Java 8. Vamos a ver el código:

```
package com.arquitecturajava.ejemplo001;

public class Factura {

	private int numero;
	private double 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 Factura(int numero, double importe) {
		super();
		this.numero = numero;
		this.importe = importe;
	}
```

```
        public double getImporte  (TipoIva tipo) {

                return tipo.calcular(importe) ;
        }
}
```

```
package com.arquitecturajava.ejemplo001;

public interface TipoIva {

        public double calcular(double importe);
}
```

```
package com.arquitecturajava.ejemplo001;

public class IVANormal implements TipoIva {

        @Override
        public double calcular(double importe) {

                return importe *1.21;
        }

}
```

```
package com.arquitecturajava.ejemplo001;

public class IVAReducido implements TipoIva {

	@Override
	public double calcular(double importe) {

		return importe *1.07;
	}

}
```

Una vez tenemos las clases fundamentales construidas nos queda definir la clase Factoría y el programa principal:

```
package com.arquitecturajava.ejemplo001;

public class FactoriaIVA {

	public static  TipoIva getInstance(String tipo) {

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

			return new IVAReducido();
		}else {
			return new IVANormal();
		}
```

```
	}

}
```

```
package com.arquitecturajava.ejemplo001;

public class Principal {

	public static void main(String[] args) {

		Factura f1= new Factura(1,200);
		Factura f2= new Factura(1,300);
		GestorFacturas gf= new GestorFacturas();
		gf.add(f1);
		gf.add(f2);
		TipoIva tipo=FactoriaIVA.getInstance("IvaReducido");
		System.out.println(gf.importeTotal(tipo));

	}

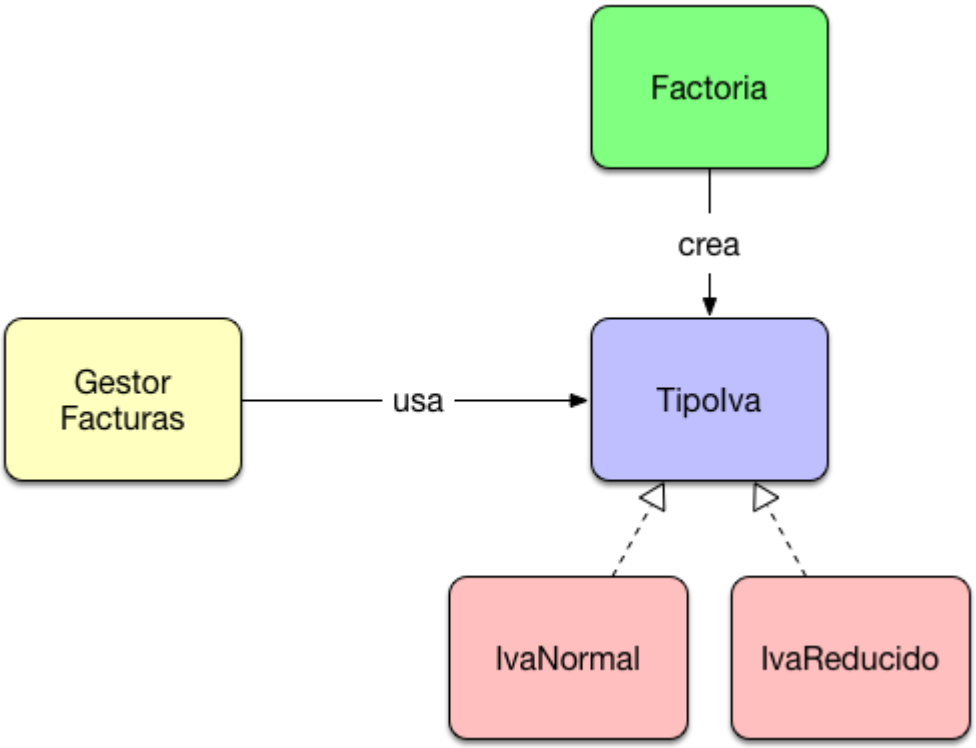
}
```

El resultado es :



Para configurar la factoría hemos necesitado bastantes clases , en concreto una por cada

algoritmo que implemente el calculo del Iva.



Java 8 Factory Pattern

Podemos implementar el concepto de Factoría de otra forma para eliminar parte de las clases que acabamos de construir. En primer lugar definimos el interface y una serie de métodos estáticos asociados:

```
package com.arquitecturajava.ejemplo002;

public interface TipoIva {

        public double calcular(double importe);

        public static double IVANormal(double importe) {
```

```
                return importe * 1.21;

        }

        public static double IVAReducido(double importe) {

                return importe * 1.07;

        }
}
```

La factoría podrá referenciar a estos métodos como expresiones Lambda:

```
package com.arquitecturajava.ejemplo002;

public class FactoriaIVA {

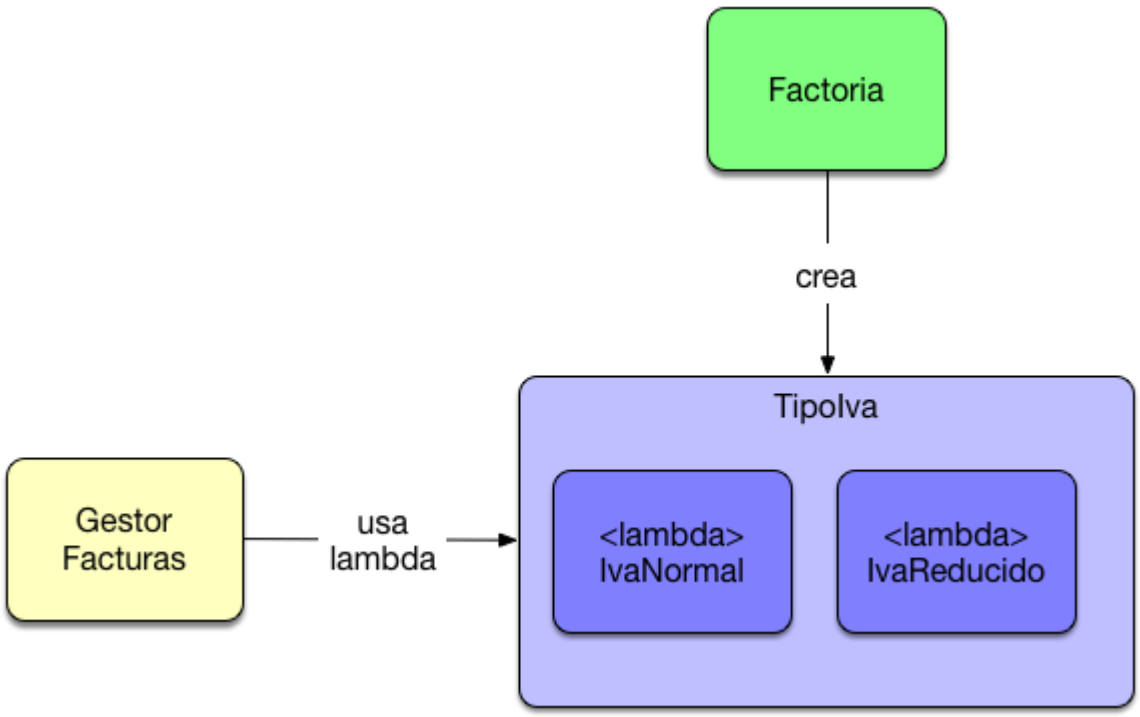
        public static  TipoIva getInstance(String tipo) {

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

                        return TipoIva::IVAReducido;
                }else {
                        return TipoIva::IVANormal;
                }
        }

}
```

Configurándolo así podemos eliminar parte de nuestras clases y compactar el código:



El resultado será identico:



Acabamos de eliminar la jerarquía de clases utilizando Java 8 Factory Pattern.

Otros artículos relacionados:

1. Java Stream forEach y colecciones
2. Java Lambda reduce y wrappers
3. Java 8 Lambda y forEach (II)
4. Factory Pattern