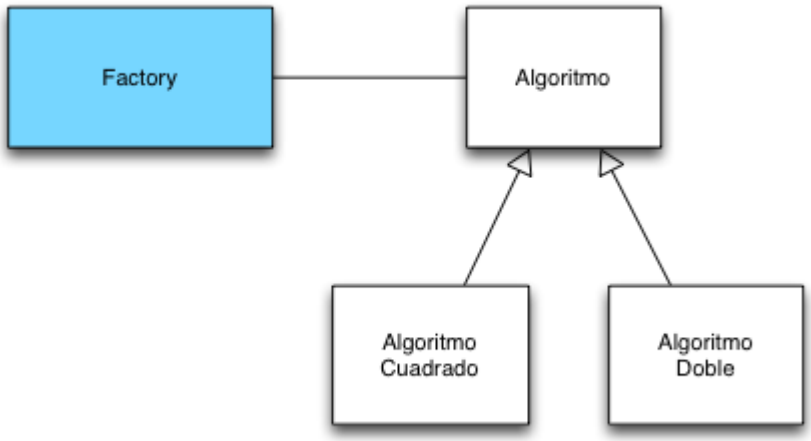
Las factorías o Java Factories es uno de los patrones de diseño más utilizados por los desarrolladores y se encargan de facilitar la creación de una jerarquía de clases. Ademas de construir dicha jerarquía nos permiten encapsular toda una jerarquía de clases apoyándonos en la clase padre o interface que definamos.



Una de las preguntas que me realizan más habitualmente sobre las factoría es si podemos evitar de alguna manera el paso de un parámetro de tipo String ya que facilita que los desarrolladores cometan errores. Vamos a ver el código de inicio:

```
package com.arquitecturajava.factory;

public interface Algoritmo {

public int calcular(int numero);

}
```

```
package com.arquitecturajava.factory;

public class AlgoritmoCuadrado implements Algoritmo {

@Override
public int calcular(int numero) {

return numero*numero;
}

}
```

```
package com.arquitecturajava.factory;

public class AlgoritmoDoble implements Algoritmo {

 @Override
 public int calcular(int numero) {

 return numero*2;

 }

}
```

En este caso se usa una interface para definir el concepto de Algoritmo e implementar dos operaciones diferentes una es la de duplicar el valor y la otra calcula el cuadrado del valor que se pase por parámetro. La forma clásica de implementar una Java Factory es a través de una sentencia if/else :

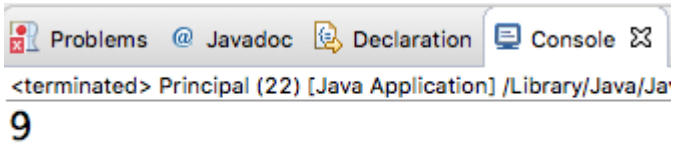
```
package com.arquitecturajava.factory;

public class Factory {

public static Algoritmo crearAlgoritmo(String tipo) {

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

return new AlgoritmoDoble();
}else {
return new AlgoritmoCuadrado();
}

}

}
```

El programa principal invocará el método crearAlgoritmo e instanciara uno de los Algoritmos solicitados:

```
package com.arquitecturajava.factory;

public class Principal {

 public static void main (String[] args) {

 Algoritmo a= Factory.crearAlgoritmo("doble");
```

```
 System.out.println(a.calcular(3));

 }
}
```

El problema de este código es que hemos pasado “doble” en minúscula y la Factoría lo esperaba en mayúscula. Por lo tanto el resultado que se muestra por pantalla será:



Java Factories y Enums

Para solventar este problema nos podemos apoyar en los Enums de Java y diseñar el sistema de factorías de otra forma:

```
package com.arquitecturajava.factory2;

public enum TipoAlgoritmo {

DOBLE {

public Algoritmo crear() {
return new AlgoritmoDoble();
}
},
CUADRADO {

public Algoritmo crear() {
```

```
return new AlgoritmoCuadrado();
}
};

public abstract Algoritmo crear();
}
```

En este caso hemos diseñado una enumeración que devuelve un tipo u otro de Factoría dependiendo de nuestras necesidades, nos queda integrarlo dentro de una Factoría.

```
package com.arquitecturajava.factory2;

public class Factory {

public static Algoritmo crearAlgoritmo(TipoAlgoritmo tipo) {

return tipo.crear();

}

}
```

Esto evitará que el desarrollador cometa errores cuando instancia los diferentes algoritmos a través de la factoría:

```
package com.arquitecturajava.factory2;

public class Principal {

public static void main (String[] args) {

Algoritmo a= Factory.crearAlgoritmo(TipoAlgoritmo.DOBLE);
System.out.println(a.calcular(2));

}
}
```

Las enumeraciones son unas desconocidas pero permiten soluciones originales.

Java Proxy

Java Singleton

Oracle Enums