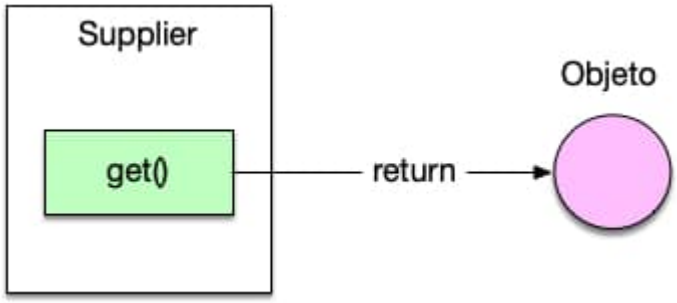
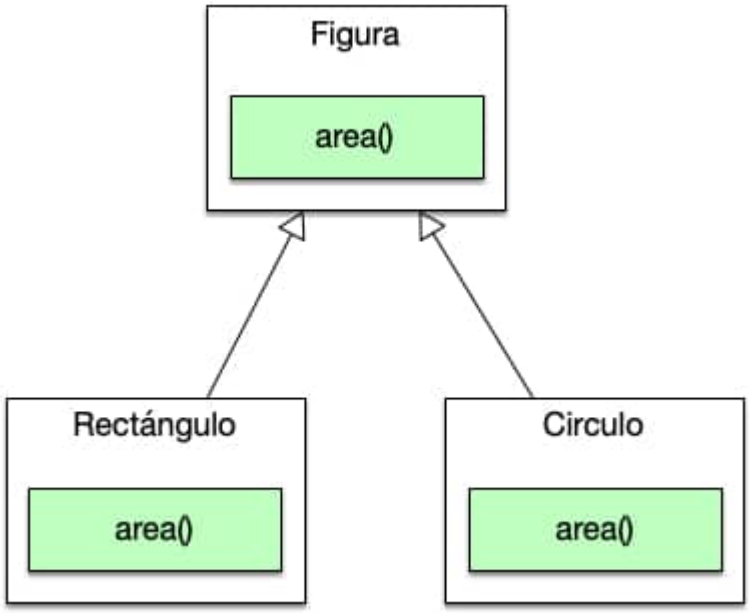
El concepto de Java Supplier Interface . Es uno de los conceptos de programación funcional que a veces más cuesta entender . No tanto por su complejidad sino por su posible utilidad. ¿Para que sirve un Java Supplier Interface?. Este interface es uno de los más sencillos ya que solo tiene un método que nos devuelve un valor . Un Supplier es un proveedor y por lo tanto nos devuelve un valor o un objeto sin que nosotros le pasemos ningún parámetro.



Este interface por lo tanto no esta tan orientado al trabajo con Java Streams ya que estos siempre pasan un parámetro en cada iteración sobre el stream. El caso del interface Supplier es diferente. Vamos a ver un ejemplo con el que podamos entender un poco mejor este concepto. Para ello vamos a usar un ejemplo clásico de Factorias. Supongamos que tenemos las clases Figura , Rectángulo y Circulo. Estas clases se pueden relacionar por herencia ya que Figura sería la clase abstracta y las otras sus clases hijas.



El código Java podría construirse de la siguiente forma:

```
package com.arquitecturajava.ejemplo2;

public  abstract class Figura {

  public abstract double area();
}
```

```
package com.arquitecturajava.ejemplo2;

public class Circulo extends Figura {

  private int radio;

  public Circulo() {
    this(2);
```

```
  }

  public Circulo(int radio) {
    super();
    this.radio = radio;
  }

  public int getRadio() {
    return radio;
  }

  public void setRadio(int radio) {
    this.radio = radio;
  }

  @Override
  public double area() {
    // TODO Auto-generated method stub
    return 2 * Math.PI * radio;
  }
}
```

```
package com.arquitecturajava.ejemplo2;

public class Rectangulo  extends Figura{
  private int lado1;
  private int lado2;

  public Rectangulo() {
    this(2,2);
  }
  public int getLado1() {
```

```
    return lado1;
  }
  public void setLado1(int lado1) {
    this.lado1 = lado1;
  }
  public int getLado2() {
    return lado2;
  }
  public Rectangulo(int lado1, int lado2) {
    super();
    this.lado1 = lado1;
    this.lado2 = lado2;
  }
  public void setLado2(int lado2) {
    this.lado2 = lado2;
  }
  @Override
  public double area() {
    // TODO Auto-generated method stub
    return this.lado1*this.lado2;
  }
}
```

Una vez creada cada una de las clases , el siguiente paso es crear una Factoría que nos devuelve uno u otro objeto dependiendo del tipo.

```
package com.arquitecturajava.ejemplo2;

public class FactoriaFiguras1 {

  public static Figura crearFigura(String tipo) {
    if (tipo.equals("Rectangulo")) {
```

```
      return new Rectangulo();
    }else {
      return new Circulo();
    }
  }

}
```

Nos queda ver el programa principal que usa la Factoria:

```
package com.arquitecturajava.ejemplo2;

public class Principal2 {

  public static void main(String[] args) {

    Figura f=FactoriaFiguras1.crearFigura("Rectangulo");
    System.out.println(f.area());
  }

}
```

Todo funciona bastante correcto y nos imprimirá 4.0 por la consola.

Java Supplier Interface

De igual forma que podemos construir la Factoría de esta manera podríamos haber hecho lo mismo pero utilizando el interface Supplier

```
package com.arquitecturajava.ejemplo2;

import java.util.HashMap;
```

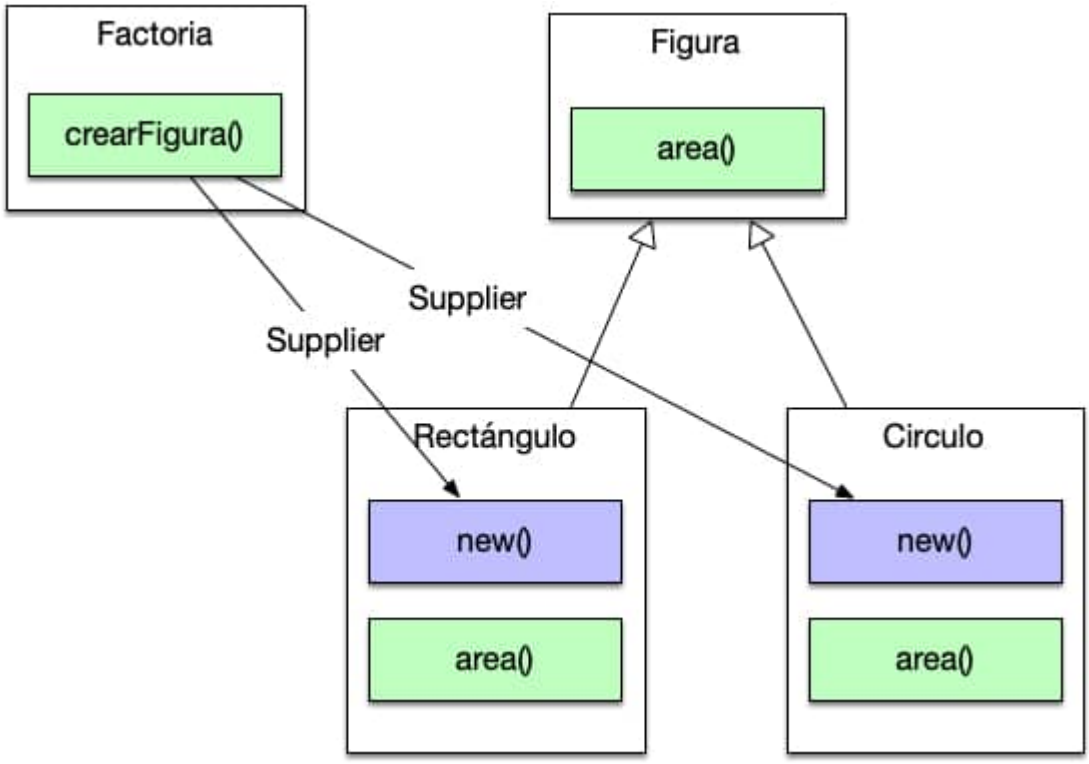
```
import java.util.function.Supplier;

public class Factoria8Figuras {

  private static HashMap<String,Supplier<Figura>> mapa= new
HashMap<>();
  static {
    mapa.put("rectangulo", Rectangulo::new);
    mapa.put("circulo", Circulo::new);
  }
  public static Figura crearFigura(String tipo) {
      if (mapa.get(tipo)!=null) {
        return mapa.get(tipo).get();
      }else {
        return null;
      }
  }

}
```

En este caso estamos usando como Supplier un reference method a un constructor (Rectangulo::new) o (Circulo::new) . De esta forma la factoría funcionará de una forma similar pero apoyandose en el concepto de Supplier y en un HashMap que nos aporta versatilidad a la hora de ir añadiendo nuevas implementaciones. De esta forma hemos usado una referencia de Supplier para apuntar con métodos de referencia a cada uno de los constructores. Recordemos que un constructor por defecto no recibe nada y devuelve una instancia de la clase . Ese es el concepto de Supplier.



Otros artículos relacionados

1. Java Predicate Interface y sus métodos
2. Java Functional Interface
3. Java 8 Functional Interfaces y sus tipos
4. Oracle interfaces funcionales