En Java el uso de interfaces es algo común en la programación del día a día . Sin embargo también es bastante común tener muchas dudas de para que sirve un interface . Normalmente nos podemos encontrar con definiciones como la siguiente de Interface.

Interface : Conjunto de métodos Java que una clase debe implementar .Una interface define un contrato que una clase tiene que cumplir al implementar los métodos .

Definiciones similares a estas las podemos encontrar por decenas. Una vez leida la definición los programadores suelen ir corriendo a echar un vistazo al código fuente a ver si consiguen entenderlo mejor y se encuentran con algo como lo siguiente.

```
package com.arquitecturajava;

public interface Copiable {

public void copiar();
 public void copiarColor();
}
```

```
package com.arquitecturajava;

public class CopiadoraProfesional implements Copiable {

@Override
 public void copiar() {
 System.out.println("la copiadora copia");
```

```
}

@Override
 public void copiarColor() {
 System.out.println("la copiadora copia a color");

}

}
```

Después de revisar el código varias veces mucha gente entiende como se implementa un interface . Lo dificil es entender para que sirve y como usarlo. Es como si nos dijeran que tenemos un aparato que tiene 4 cilindros alineados recubiertos de una goma plastica y soldada con un metal a los cuales se les transfiere energia …. para decir que hablamos de un “Coche”. Normalmente las personas no hablamos así sino que decimos un Coche es un aparato que sirve para desplazarse mas rápido (eso se entiende mejor).

Java Interfaces (Uso)

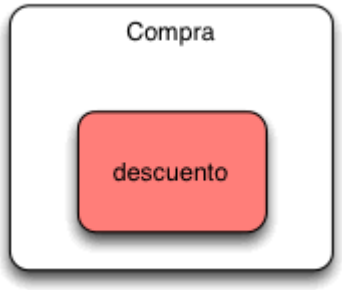
¿Para que se usa un interface en Java? .Un interface en Java se usa para añadir flexibilidad al programa que estamos construyendo. Supongamos que tenemos el siguiente bloque de código.

```
package com.arquitecturajava;

public class Compra {

private double importe;
 private String tipo;
```

```
public String getTipo() {
 return tipo;
 }

public void setTipo(String tipo) {
 this.tipo = tipo;
 }

public double getImporte() {
 return importe;
 }

public void setImporte(double importe) {
 this.importe = importe;
 }

public Compra(double importe, String tipo) {
 super();
 this.importe = importe;
 this.tipo = tipo;
 }

public double descuento() {

double descuento;

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

if (importe &amp;gt; 0 &amp;amp;&amp;amp; importe &amp;lt; 100) {

descuento = 10;
```

```
 } else {
 descuento = 20;

}

} else {

if (importe &amp;gt; 0 &amp;amp;&amp;amp; importe &amp;lt; 100) {

descuento = 20;
 } else {
 descuento = 30;

}
 }

 return importe -importe *descuento/100;
 }

}
```

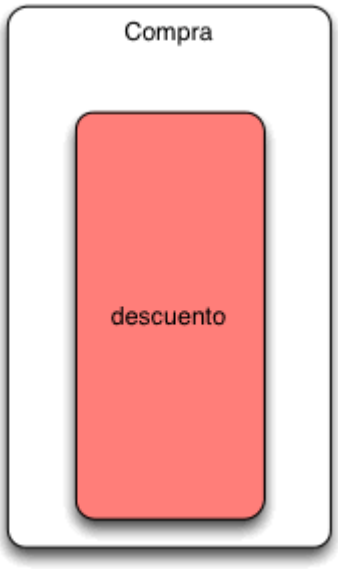
Se trata de la clase Compra que hace un calculo del descuento que nos aplican dependiendo del tipo de Compra que realicemos (Tienda u Online)



Aunque en principio el método es completamente válido y podemos ver como se usa en un programa de consola.

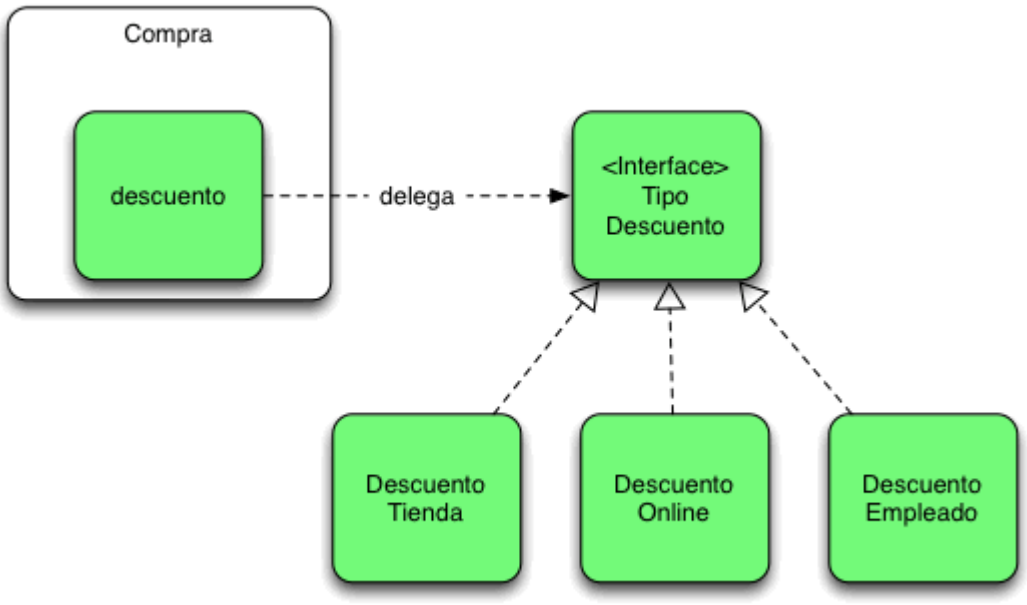
```
package com.arquitecturajava;

public class Principal {

public static void main(String[] args) {

 Compra c= new Compra(1000,"tienda");
 System.out.println(c.descuento());

}

}
```

Nos podemos encontrar con problemas si las especificaciones cambian y se añaden mas tipos de descuentos . Por ejemplo descuento por ser empleado y descuento por ser directivo de la empresa.Podemos añadir esta funcionalidad a la clase pero el método “descuento” cada vez será mas grande y mas dificil de entender.



Puntos de extensibilidad (Plugin-points)

Para solventar este problema podemos reubicar la lógica de negocio de la aplicación diseñando un interface que sirva como punto de extensibilidad para la funcionalidad del descuento.



El diagrama no es sencillo de entender de entrada así que vamos a desglosarlo con

código.Empezamos con la definición del interface.

```
package com.arquitecturajava.estrategia;

public interface TipoDescuento {

public double importeDescontar(Compra c);

}
```

Este interface recibe en su método importeDescontar el concepto de Compra. Para después calcular el descuento. Una vez visto el código del interface vamos a ver dos de sus implementaciones.

```
package com.arquitecturajava.estrategia;

public class DescuentoInternet implements TipoDescuento {

@Override
 public double importeDescontar(Compra c) {

 double descuento;
 if (c.getImporte() &amp;gt; 0 &amp;amp;&amp;amp; c.getImporte()
&amp;lt; 200) {

descuento = 20;
```

```
 } else {
 descuento = 30;

}

 return c.getImporte() * descuento / 100;
 }

}
```

```
package com.arquitecturajava.estrategia;

public class DescuentoTienda implements TipoDescuento {

public double importeDescontar(Compra c) {

double descuento;

 if (c.getImporte() &amp;gt; 0 &amp;amp;&amp;amp; c.getImporte()
&amp;lt; 100) {

descuento = 10;
 } else {
 descuento = 20;

}
 return c.getImporte() * descuento / 100;
 }
```

```
}
```

Creadas estas clases ,modificaremos la clase Compra para que se apoye en el nuevo interface (TipoDescuento).

```
package com.arquitecturajava.estrategia;

public class Compra {

private double importe;

 public double getImporte() {
 return importe;
 }

public void setImporte(double importe) {
 this.importe = importe;
 }

public Compra(double importe) {
 super();
 this.importe = importe;
 }

public double descuento(TipoDescuento tipoDescuento) {

return importe- tipoDescuento.importeDescontar(this);
 }
```

```
}
```

```
&amp;amp;nbsp;
```

Cuando nosotros queramos aplicar un descuento será tan sencillo como construir el siguiente código a nivel del programa principal.

```
package com.arquitecturajava.estrategia;

public class Principal {

public static void main(String[] args) {

 Compra c= new Compra(1000);
 System.out.println(c.descuento(new DescuentoTienda()));

}

}
```

Si queremos crear un nuevo tipo de descuento simplemente diseñaremos una nueva clase que implemente el interface .

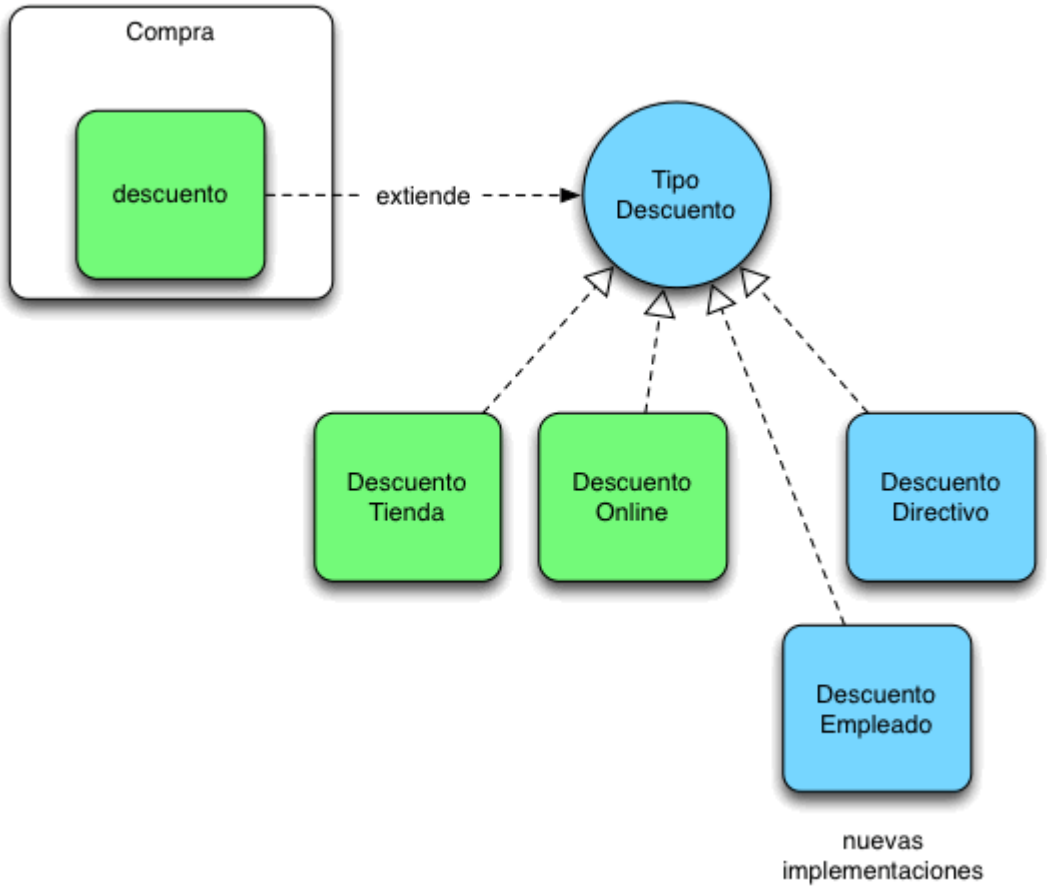
```
package com.arquitecturajava.estrategia;

public class DescuentoDirectivo implements TipoDescuento {

public double importeDescontar(Compra c) {
```

```
return c.getImporte() * 50 / 100;
 }

}
```

Creado el nuevo tipo de descuento simplemente nos queda aplicarle en el código .

```
package com.arquitecturajava.estrategia;

public class Principal {

public static void main(String[] args) {

 Compra c= new Compra(1000);
 System.out.println(c.descuento(new DescuentoDirectivo()));

}

}
```

Es por ello que a los interfaces también se les conoce a nivel de diseño como plugin-points o puntos de extensibilidad.


Compra
descuento
extiende
Tipo Descuento
Descuento Tienda
Descuento Online
Descuento Directivo
Descuento Empleado
nuevas implementaciones