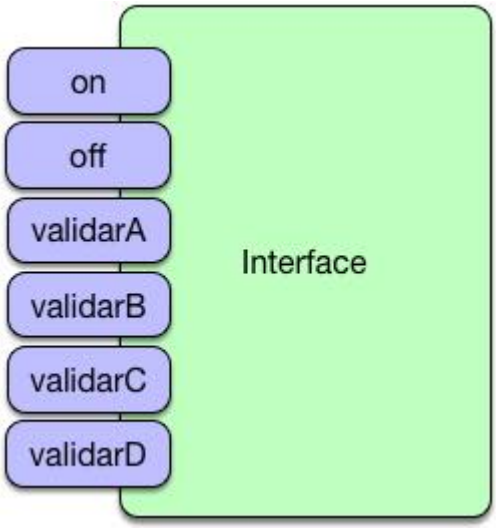
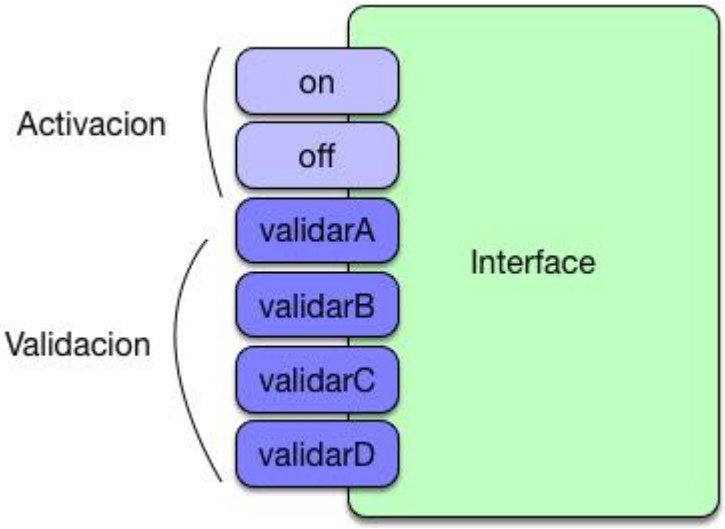
El concepto de Java Interfaces es de sobra conocido por todos. ¿ Qué son los Java interfaces?. Esta es una muy buena pregunta .La mayor parte de las veces cuando uno mira la definición de interface suele encontrarse con una respuesta del siguiente estilo. Un interface es la definición de un conjunto de métodos que una clase implementa. A mi esta definición siempre me ha parecido poco útil ya que definimos meramente lo que es no para que sirve. Otras definiciones suelen ser muy diferentes . Un interface es un contrato entre dos clases que define un comportamiento determinado. Nos hemos quedado un poco igual. ¿Así pues que es un interface?. Es difícil definirlo , esa es la realidad . Para mi un interface es una funcionalidad abierta a la reutilizacion y extensibilidad. ¿Como se entiendo esto? . Vamos a construir un ejemplo sencillo basado en la realidad. Para ello vamos a usar el concepto de Teléfono. Hoy en día todos tenemos un smartphone y para acceder a el nos validamos. Esta validación suele ser por código, huella o… o si a alguien le sobra mucho mucho el dinero con FaceId. Vamos a definir un interface que se encargue de validarnos contra el telefono.

```
package com.arquitecturajava.ejemplo1;

public interface Validar {

	public void on();
	public void off();
	public boolean validarCodigo(int numero);
	public boolean validarPatron(String patron);
	public boolean validarHuella(String patronHuella);
	public boolean validarRostro(String patronRostro);

}
```

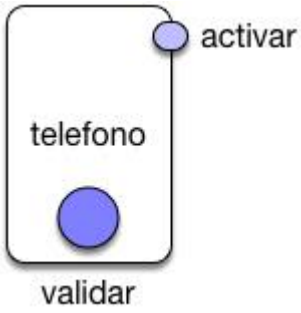
Aquí tenemos nuestro interface que nos permite validarnos contra cualquier tipo de teléfono. Hemos sido exhaustivos y cubierto todas las opciones. Podemos validarnos con código (un Iphone) , con patrón(Android) , con huella Android y Iphone y con la cara Iphone X. ¿ Es esto un interface?. No hay que mirar mucho para confirmar que lo es. Sin embargo esta pregunta tiene dos vertientes. Hemos respondido a la más sencilla . Si que es un interface a nivel puro de programación ya que usa la palabra reservada. Ahora bien ¿Es un interface a nivel conceptual?. La respuesta es que NO . Unicamente el Iphone X cumple con la funcionalidad completa del interface por lo tanto estamos ante algo NADA extensible. ¿Cual es el problema que tenemos? . El interface en estos momentos alberga demasiadas responsabilidades.



Por lo tanto lo que estamos viendo no es un interface por mucho que nos duela. Un interface es una funcionalidad orientada a la extensibilidad y a la reutilización . ¿Cuales son las funcionalidades que tenemos en este grupo de métodos?. Realmente existen dos funcionalidades. Una es la funcionalidad de activación (on/off) y la otra funcionalidad es la de validacion. Ese es el primer problema que tenemos.



Esta división de responsabilidades queda clara cuando vemos el terminal ya que están separadas en el propio terminal



Así pues un mejor diseño de los interfaces sería :

```
package com.arquitecturajava.ejemplo2;

public interface Activar {

        public void on();
```

```
        public void off();
}
```

```
package com.arquitecturajava.ejemplo1;

public interface Validar {


        public boolean validarCodigo(int numero);
        public boolean validarPatron(String patron);
        public boolean validarHuella(String patronHuella);
        public boolean validarRostro(String patronRostro);

}
```

Java Interfaces y simplicidad

Acabamos de construir una solución mejor , hemos dividido las responsabilidades de una forma adecuada. Sin embargo los interfaces también cumplen con otra premisa. Son sencillos para facilitar la extensibilidad. El interface de Activar lo es . Sin embargo en el interface de Validar tenemos muchos métodos y obligamos a muchas cosas. ¿Podemos simplificar este interface?. Si nos podemos a pensar un poco más a detalle , un teléfono no valida un código , o una huella o un patrón . Lo que realmente valida es a la persona que introduce esa información. El diseño sería mucho mas sencillo si fuera algo así:

```
package com.arquitecturajava.ejemplo2;

public interface Validar {

        public boolean validar(Persona persona);

}
```

De esta forma podremos validar la huella , el código, o lo que queramos que una Persona tenga. El interface que muy simplificado y ayuda a la extensibilidad.

Java Interfaces y código

Vamos a ver como quedan las clases que tenemos asociadas.

```
package com.arquitecturajava.ejemplo2;

public class Persona {

        private String nombre;
        private int codigo;
        private String patron;
        private String patronRostro;
        private String huella;

        public String getHuella() {
```

```
        return huella;
}

public void setHuella(String huella) {
        this.huella = huella;
}

public String getPatronRostro() {
        return patronRostro;
}

public void setPatronRostro(String patronRostro) {
        this.patronRostro = patronRostro;
}

public String getPatron() {
        return patron;
}

public void setPatron(String patron) {
        this.patron = patron;
}

public int getCodigo() {
        return codigo;
}

public void setCodigo(int codigo) {
        this.codigo = codigo;
}
```

```
	public String getNombre() {
		return nombre;
	}

	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
}
```

```
package com.arquitecturajava.ejemplo2;

public  abstract class Telefono implements Activar{

	private String marca;
	private Persona persona;
	public String getMarca() {
		return marca;
	}
	public void setMarca(String marca) {
		this.marca = marca;
	}
	public Persona getPersona() {
		return persona;
	}
	public void setPersona(Persona persona) {
		this.persona = persona;
	}
	public Telefono(String marca, Persona persona) {
```

```
                super();
                this.marca = marca;
                this.persona = persona;
        }
        @Override
        public void on() {
                System.out.println("telefono encendido");
        }
        @Override
        public void off() {
                System.out.println("telefono apagado");
        }
}



package com.arquitecturajava.ejemplo2;



public class Android extends Telefono  implements Validar{


        public Android(String marca, Persona persona) {
                super(marca, persona);
                // TODO Auto-generated constructor stub
        }

        private String patron;

        public String getPatron() {
```

```
                return patron;
        }

        public void setPatron(String patron) {
                this.patron = patron;
        }

        @Override
        public boolean validar(Persona persona) {
                // TODO Auto-generated method stub
                return persona.getPatron().equals(patron);
        }

}
```

```
package com.arquitecturajava.ejemplo2;

public class IPhone8 extends  Telefono implements Validar {

         private int codigo;
         private String huella;
        public String getHuella() {
                return huella;
        }

        public void setHuella(String huella) {
                this.huella = huella;
        }
```

```
	public IPhone8(String marca, Persona persona) {
		super(marca, persona);
		// TODO Auto-generated constructor stub
	}

	public int getCodigo() {
		return codigo;
	}

	public void setCodigo(int codigo) {
		this.codigo = codigo;
	}

	@Override
	public boolean validar(Persona persona) {
		if (persona.getHuella()!=null) {
			return persona.getHuella().equals(huella);
		}
		if (persona.getCodigo()!=0) {
			return persona.getCodigo()==codigo;
		}
		return false;
	}
}



package com.arquitecturajava.ejemplo2;

public class IPhone10 extends IPhone8{

	 private String patronRostro;
```

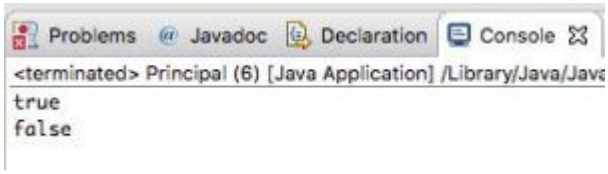
```
	public IPhone10(String marca, Persona persona) {
		super(marca, persona);
		// TODO Auto-generated constructor stub
	}

	@Override
	public boolean validar(Persona persona) {
		if(persona.getPatronRostro()!=null) {
			return
persona.getPatronRostro().equals(patronRostro);
		}
		return super.validar(persona);
	}

	public String getPatronRostro() {
		return patronRostro;
	}

	public void setPatronRostro(String patronRostro) {
		this.patronRostro = patronRostro;
	}

}
```

Ahora si que tenemos correctamente definidos los Java interfaces. Es momento de probar nuestro código en un main.

```
package com.arquitecturajava.ejemplo2;

public class Principal {
```

```
        public static void main(String[] args) {
                Persona p= new Persona();
                p.setNombre("juan");
                p.setHuella("huella1");
                IPhone8 telefono= new IPhone8("apple",p);
                telefono.setHuella("huella1");
                System.out.println(telefono.validar(p));
                Persona p1= new Persona();
                p1.setNombre("gema");
                p1.setHuella("huella2");
                System.out.println(telefono.validar(p1));

        }

}
```

Ejecutamos e imprimimos por la consola:



La primera huella es valida y la segunda no. Acabamos de ver un poco más a detalle el concepto de Java Interfaces y como la extensibilidad y la sencillez son claves para un diseño correcto.

1. Java Herencia vs Interfaces
2. Java Predicate Interface y sus métodos
3. Java Fluent Interface y Properties