1. **CONSULTAR A NIVEL DE CONCEPTOS Y EJEMPLOS EL USO DE INTERFACES EN JAVA.**

**Interfaz (Java)**

Una interfaz en Java es una colección de métodos abstractos y propiedades constantes.

En las interfaces se especifica qué se debe hacer, pero no su implementación. Serán las clases que implementen estas interfaces las que describen la lógica del comportamiento de los métodos.

La principal diferencia entre interface y abstract es que una interface proporciona un mecanismo de encapsulación de los protocolos de los métodos sin forzar al usuario a utilizar la herencia.

**Ventajas**

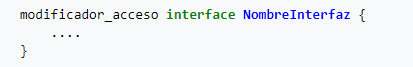
El uso de las interfaces Java proporciona las siguientes ventajas:

* Organizar la programación.
* Permiten declarar constantes que van a estar disponibles para todas las clases que queramos (implementando esa interfaz)
* Obligar a que ciertas clases utilicen los mismos métodos (nombres y parámetros).
* Establecer relaciones entre clases que no estén relacionadas.

**Uso**

Java proporciona dos palabras reservadas para trabajar con interfaces: **interface** e **implements.**

Para declarar una interfaz se utiliza:



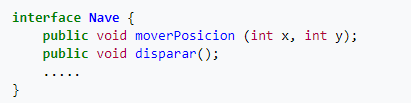
**modificador\_acceso** puede ser una clase de objetos que nos permite utilizar herencia en abstracción constante en las clases en las que se implemente.

Para implementarla en una clase, se utiliza la forma:

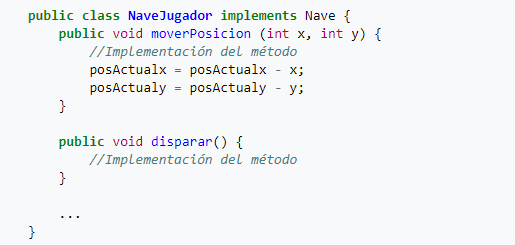


**Ejemplo**

**Definición de una interfaz:**



**Uso de la interfaz definida:**



**Concepto de interface:**

En el tema dedicado a la herencia en Java se explica el concepto de clase abstracta. Una clase que contenga un método abstracto se debe declarar como abstracta. También sabemos que una clase abstracta puede contener atributos, constructores y métodos no abstractos. De una clase abstracta no se pueden crear objtos y su finalidad es ser la base para construir la jerarquía de herencia entre clases y aplicar el polimorfismo.

Llevando estos conceptos un poco más allá aparece en concepto de Interface.

Podemos considerar una **interface** como una clase que sólo puede contener:

* **Métodos abstractos**
* **Atributos constantes**

A partir de **Java 8** el concepto de Interface se ha ampliado y a pertir de esa versión de Java las interfaces tambíen pueden contener:

* **Métodos por defecto**
* **Métodos estáticos**
* **Tipos anidados**

Y a partir de **Java 9** se les ha añadido una nueva funcionalidad a las interfaces y también pueden contener:

* Métodos privados

Una Interface tiene en común con una clase lo siguiente:

* Puede contener métodos.
* Se escribe en un archivo con extensión .java que debe llamarse exactamente igual que la interface.
* Al compilar el programa, el byte-code de la Interface se guarda en un archivo. class

Pero una inteface se diferencia de una clase en lo siguiente:

* No se pueden instanciar. No podemos crear objetos a partir de una Interface.
* No contiene constructores.
* Si contiene atributos todos deben ser public static final
* Una interface puede heredar de varias interfaces. Con las interfaces se permite la herencia múltiple.

Conceptos importantes a tener en cuenta cuando trabajamos con interfaces:

* Las clases no heredan las interfaces. Las clases **implementan** las interfaces.
* Una clase puede implementar varias interfaces.

Explicándolo de una manera simple, implementar una interface sería equivalente a copiar y pegar el código de la interface dentro de la clase. Cuando una clase implementa una interface está obligada a implementar todos los métodos abstractos que contiene ya que de otra forma debería declararse como clase abstracta.

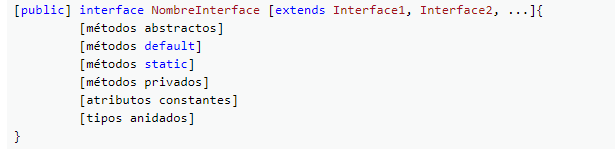
Usando Interfaces, si varias clases distintas implementan la misma interface nos aseguramos que todas tendrán implementados una serie de métodos en comunes.

Las interfaces juegan un papel fundamental en la creación de aplicaciones Java.

Las interfaces definen un protocolo de comportamiento y proporcionan un formato común para implementarlo en las clases.

Utilizando interfaces es posible que clases no relacionadas, situadas en distintas jerarquías de clases sin relaciones de herencia, tengan comportamientos comunes.

Una interface se crea utilizando la palabra clave interface en lugar de class.

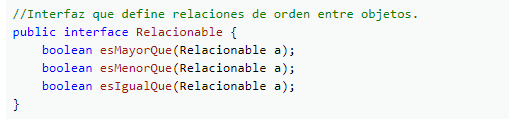


**Características**

1. **La interface puede definirse public o sin modificador de acceso,** y tiene el mismo significado que para las clases. Si tiene el modificador public el archivo .java que la contiene debe tener el mismo nombre que la interfaz.
2. **En los métodos abstractos** no es necesario escribir abstract.
3. **Los métodos por defecto** se especifican mediante el modificador default.
4. **Los métodos estáticos** se especifican mediante la palabra reservada static.
5. **Los métodos privados** se especifican mediante el modificador de acceso private.
6. **Todos los atributos son constates públicos y estáticos**. Por lo tanto, se pueden omitir los modificadores public static final cuando se declara el atributo. Se deben inicializar en la misma instrucción de declaración.
7. **Los nombres de las interface suelen acabar en able** aunque no es necesario: Configurable, Arrancable, Dibujable, Comparable, Clonable, etc.

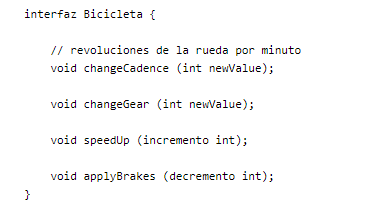
**Ejemplo:**

Creamos una interface llamada Relacionable con tres métodos abstractos.

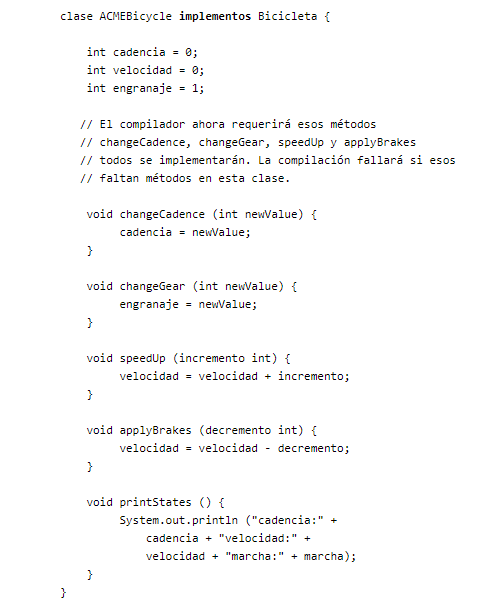


**Ejemplo:**

El comportamiento de una bicicleta, si se especifica como una interfaz, puede aparecer de la siguiente manera:

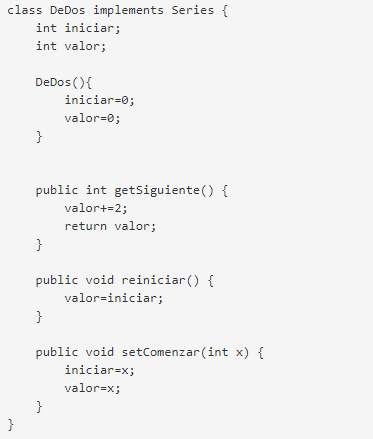


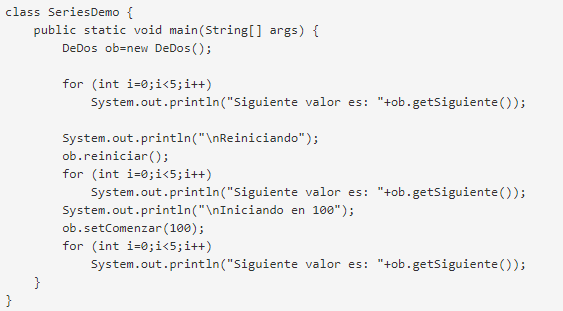
Para implementar esta interfaz, el nombre de su clase cambiaría (a una marca particular de bicicleta, por ejemplo, como ACMEBicycle), y usaría la implementspalabra clave en la declaración de la clase:



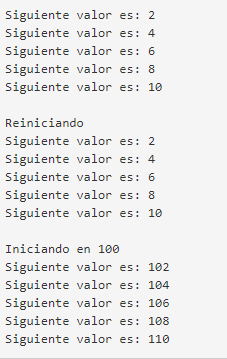
La implementación de una interfaz permite que una clase se vuelva más formal sobre el comportamiento que promete proporcionar. Las interfaces forman un contrato entre la clase y el mundo exterior, y el compilador hace cumplir este contrato en el momento de la compilación. Si su clase afirma implementar una interfaz, todos los métodos definidos por esa interfaz deben aparecer en su código fuente antes de que la clase se compile correctamente.

**Ejemplo:**





Salida



1. **CONSULTAR A NIVEL DE CONCEPTOS Y EJEMPLOS LA IMPLEMENTACIÓN DEL PRINCIPIO DE INTERFACE SEGREGATION**

El principio de segregación de interfaces viene a decir que ninguna clase debería depender de métodos que no usa. Por tanto, cuando creemos interfaces que definan comportamientos, es importante estar seguros de que todas las clases que implementen esas interfaces vayan a necesitar y ser capaces de agregar comportamientos a todos los métodos. En caso contrario, es mejor tener varias interfaces más pequeñas.

Las interfaces nos ayudan a desacoplar módulos entre sí. Esto es así porque si tenemos una interfaz que explica el comportamiento que el módulo espera para comunicarse con otros módulos, nosotros siempre podremos crear una clase que lo implemente de modo que cumpla las condiciones.

El módulo que describe la interfaz no tiene que saber nada sobre nuestro código y, sin embargo, nosotros podemos trabajar con él sin problemas.

**El problema**

La problemática surge cuando esas interfaces intentan definir más cosas de las debidas, lo que se denominan fat interfaces.

Probablemente ocurrirá que las clases hijas acabarán por no usar muchos de esos métodos, y habrá que darles una implementación.

Muy habitual es lanzar una excepción, o simplemente no hacer nada.

Pero, al igual que vimos en algún ejemplo en el principio de sustitución de Liskov, esto es peligroso. Si lanzamos una excepción, es más que probable que el módulo que define esa interfaz use el método en algún momento, y esto hará fallar nuestro programa.

El resto de implementaciones “por defecto” que podamos dar, pueden generar efectos secundarios que no esperemos, y a los que sólo podemos responder conociendo el código fuente del módulo en cuestión, cosa que no nos interesa.

**¿Cómo detectar que estamos violando el Principio de segregación de interfaces?**

Como comentaba en los párrafos anteriores, si al implementar una interfaz ves que uno o varios de los métodos no tienen sentido y te hace falta dejarlos vacíos o lanzar excepciones, es muy probable que estés violando este principio.

Si la interfaz forma parte de tu código, divídela en varias interfaces que definan comportamientos más específicos.

Recuerda que no pasa nada porque una clase ahora necesite implementar varias interfaces. El punto importante es que use todos los métodos definidos por esas interfaces.

**¿Qué hacer con código antiguo?**

Si ya tienes código que utiliza fat interfaces, la solución puede ser utilizar el patrón de diseño “Adapter”. El patrón Adapter nos permite convertir unas interfaces en otras, por lo que puedes usar adaptadores que conviertan la interfaz antigua en las nuevas.

**Conclusión**

El principio de segregación de interfaces nos ayuda a no obligar a ninguna clase a implementar métodos que no utiliza. Esto nos evitará problemas que nos pueden llevar a errores inesperados y a dependencias no deseadas. Además, nos ayuda a reutilizar código de forma más inteligente.

**Ejemplo:**

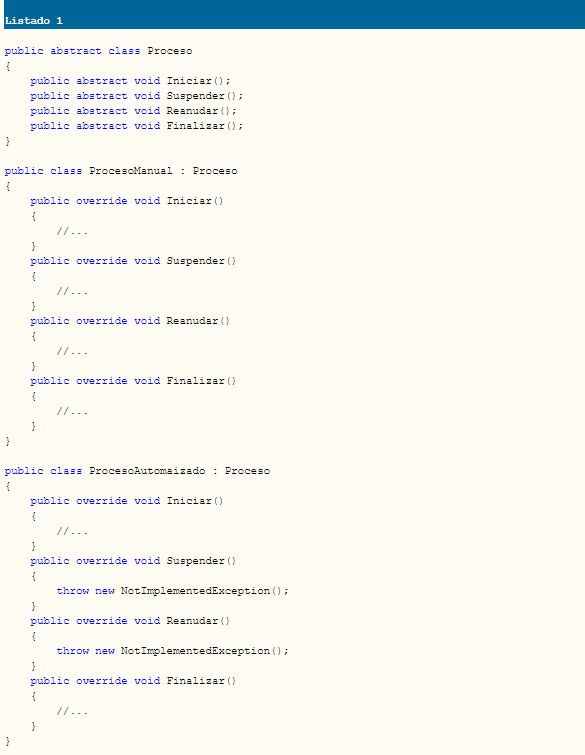
Veamos un ejemplo que esté violando este principio (aclaro que el ejemplo no tiene demasiado sentido, pero es útil para comprender el tema).

Supongamos que tenemos la clase abstracta Proceso: (refiriéndonos con proceso a un tipo de tarea a llevar a cabo). En este definimos cuatro métodos, el primero y el último para iniciar y terminar el proceso, y el segundo y tercero para suspenderlo y reiniciarlo (estos dos últimos suponiendo que quien lo lleva a cabo la tarea es una persona y que, evidentemente, debe parar cada cierto tiempo a descansar).

Supongamos que tenemos dos tipos de procesos, unos van a ser manuales (ejecutado por personas) y otros automatizados (ejecutados por máquinas):

Una primera aproximación al problema sería la siguiente

**Solución Errónea**



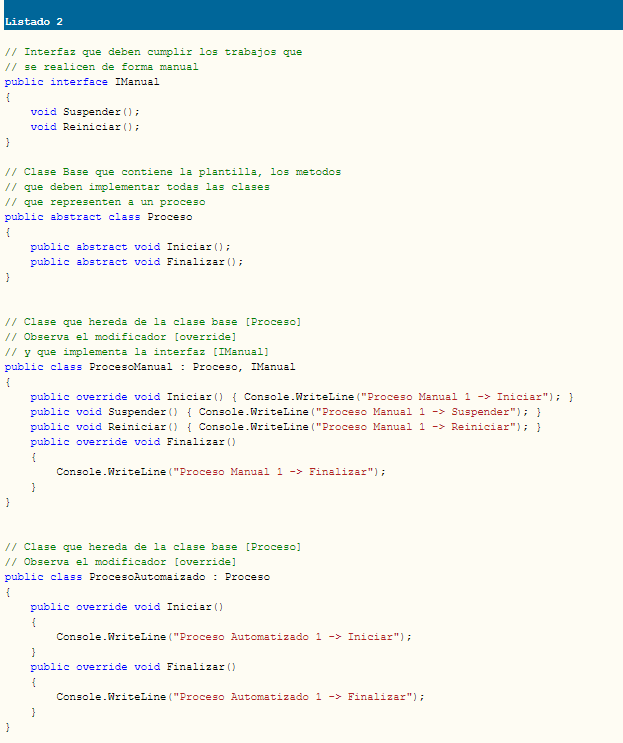
En la clase ProcesoManual se implementan todos los métodos, pero en la clase ProcesoAutomatizado no se implementan los métodos suspender y reiniciar, porque las máquinas no necesitan tomarse un descanso y por lo tanto no es necesario ni suspender el proceso ni volver a reiniciarlo.

Además, cualquier llamada es uno de esos métodos generaría una excepción del tipo NotImplementedException. Podríamos quitar esa excepción... por ejemplo dejando el método vacío; pero entonces el programador que utilice la clase se encontrará con un método que sencillamente no hace nada. Podríamos intentar solucionarlo, mediante documentación, indicando que el método no es funcional, mediante comentarios en el código (pero es posible que un programador que utilice la clase no tenga acceso al código), etc. En cualquier caso, el problema seguiría existiendo, y solo estaríamos ocultándolo.

Como podemos ver en este caso estamos violando el principio ISP, ya que estamos obligando a una clase a implementar métodos que no va a utilizar.

**Solución Correcta**

Una solución a este problema es dividir un poco las cosas y crear, por ejemplo, una interfaz que tenga definida las operaciones propias de los procesos manuales, Y en segundo lugar modificar la clase Proceso que ahora tiene únicamente, los métodos que son comunes a todos los subtipos de procesos. De esta forma evitaremos que los procesos automatizados implementen métodos que no les son útiles:



**Bibliografía**

Macaray, J.-F., & Nicolas, C. (1996). Programacion java. Gestion 2000.

Principio de segregación de interfaces (SOLID 4a parte). (2016, enero 21). Recuperado el 15 de julio de 2021, de Devexperto.com website: https://devexperto.com/principio-de-segregacion-de-interfaces/

Serrano, N. M., name/, W. M., email=mailto:, cia=jms, & :-P, C. S. F. (2012). [ISP] Principio de segregación de interfaces. Recuperado de http://joaquin.medina.name/web2008/documentos/informatica/documentacion/logica/OOP/Principios/Oop\_Solid\_ISP/2012\_09\_14\_ISP\_InterfaceSegregationPrincipe.html

Walton, A. (2020, mayo 7). Interfaces en Java con Ejemplos. Recuperado el 15 de julio de 2021, de Javadesdecero.es website: https://javadesdecero.es/intermedio/interfaces-ejemplos/

What Is an Interface? (s/f). Recuperado el 15 de julio de 2021, de Oracle.com website: https://docs.oracle.com/javase/tutorial/java/concepts/interface.html

Wikipedia contributors. (s/f). Interfaz (Java). Recuperado el 15 de julio de 2021, de Wikipedia, The Free Encyclopedia website: https://es.wikipedia.org/w/index.php?title=Interfaz\_(Java)&oldid=126025257