Interfaces

En la programación orientada a objetos, a veces es útil definir qué debe hacer una clase pero no cómo lo hará. Ya has visto un ejemplo de esto: el método abstracto. Un método abstracto define la firma de un método pero no proporciona ninguna implementación. Una subclase debe proporcionar su propia implementación de cada método abstracto definido por su superclase. Por lo tanto, un método abstracto especifica la interfaz con el método pero no la implementación. Si bien las clases y los métodos abstractos son útiles, es posible llevar este concepto un paso más allá. En Java, puede separar completamente la interfaz de una clase de su implementación utilizando la interfaz .

Una interfaz es sintácticamente similar a una clase abstracta, ya que puede especificar uno o más métodos que no tienen cuerpo. Esos métodos deben ser implementados por una clase para que sus acciones sean definidas. Por lo tanto, una interfaz especifica lo que se debe hacer, pero no cómo hacerlo. Una vez que se define una interfaz, cualquier número de clases puede implementarla. Además, una clase puede implementar cualquier número de interfaces.

Para implementar una interfaz, una clase debe proporcionar cuerpos (implementaciones) para los métodos descritos por la interfaz.

Cada clase es libre de determinar los detalles de su propia implementación.

Dos clases podrían implementar la misma interfaz de diferentes maneras, pero cada clase aún admite el mismo conjunto de métodos.
Por lo tanto, el código que tiene conocimiento de la interfaz puede usar objetos de cualquiera de las clases ya que la interfaz con esos objetos es la misma. Al proporcionar la palabra clave de la interfaz, Java le permite utilizar completamente el aspecto de polimorfismo de “una interfaz, múltiples métodos”.

Antes de continuar hay que hacer un punto importante.
JDK 8 agregó una característica a la interfaz que hace un cambio significativo en sus capacidades.
Antes de JDK 8, una interfaz no podía definir ninguna implementación.
Por lo tanto, antes de JDK 8, una interfaz podría definir solo qué, pero no cómo, como se acaba de describir.
JDK 8 cambia esto.
Hoy en día, es posible agregar una implementación predeterminada a un método de interfaz.
Por lo tanto, ahora es posible que la interfaz especifique algún comportamiento.
Sin embargo, los métodos predeterminados constituyen lo que es, en esencia, una característica de uso especial, y la intención original detrás de la interfaz aún permanece.
Por lo tanto, como regla general, a menudo creará y usará interfaces en las que no existen métodos predeterminados. Por esta razón, comenzaremos discutiendo la interfaz en su forma tradicional.

```
access interface name {
   ret-type method-name1(param-list);
   ret-type method-name2(param-list);
   type var1 = value;
   type var2 = value;
```

```
    // ...
    ret-type method-nameN(param-list);
    type varN = value;
}
```

El acceso es público o no se utiliza. Cuando no se incluye un modificador de acceso, los resultados de acceso predeterminados y la interfaz están disponibles solo para otros miembros de su paquete.

Cuando se declara como pública, la interfaz puede ser utilizada por cualquier otro código. (Cuando una interfaz se declara pública, debe estar en un archivo del mismo nombre.) Nombre es el nombre de la interfaz y puede ser cualquier identificador válido.
En la forma tradicional de una interfaz, los métodos se declaran utilizando solo su tipo de retorno y su firma. Son, esencialmente, métodos abstractos. Por lo tanto, cada clase que incluya dicha interfaz debe implementar todos sus métodos. En una interfaz, los métodos son implícitamente públicos.
Las variables declaradas en una interfaz no son variables de instancia. En su lugar, son implícitamente públicos, finales y estáticos, y deben inicializarse.

```
public interface Series {

    int getNext(); // return next number in series

    void reset(); // restart

    void setStart(int x); // set starting value
}
```

Implementing Interfaces

Una vez que se ha definido una interfaz, una o más clases pueden implementar esa interfaz. Para implementar una interfaz, incluya la cláusula de implementos en una definición de clase y luego cree los métodos requeridos por la interfaz. La forma general de una clase que incluye la cláusula de implementos se ve así:
clase nombre de clase extiende la interfaz de implementos de superclase {
// clase-cuerpo
}

Para implementar más de una interfaz, las interfaces se separan con una coma. Por supuesto, la cláusula extendida es opcional.
Los métodos que implementan una interfaz deben ser declarados públicos. Además, la firma de tipo del método de implementación debe coincidir exactamente con la firma de tipo especificada en la definición de la interfaz.

```
class DeDos implements Series {
```

```
    int start;
    int val;

    DeDos() {
      start = 0;
      val = 0;
    }
    public int getNext() {
      val += 2;
      return val;
    }
    public void reset() {
      val = start;
    }
    public void setStart(int x) {
      start = x;
      val = x;
    }
  }
```

Ejemplo de uso:

```
class SeriesDemo {
  public static void main(String args[]) {
    DeDos ob = new DeDos();
    for (int i = 0; i < 5; i++) {
      System.out.println("Next value is " + ob.getNext());
    }
    System.out.println("\nResetting");
    ob.reset();
    for (int i = 0; i < 5; i++) {
      System.out.println("Next value is " + ob.getNext());
    }
    System.out.println("\nStarting at 100");
    ob.setStart(100);
    for (int i = 0; i < 5; i++) {
      System.out.println("Next value is " + ob.getNext());
    }
  }
}
```

Es permisible y común para las clases que implementan interfaces definir miembros adicionales propios.

```
class DeDosBis implements Series {

   int start;
   int val;
   int prev;

   DeDosBis() {
      start = 0;
      val = 0;

      prev = -2;
   }
   public int getNext() {
      prev = val;
      val += 2;
      return val;
   }
   public void reset() {
      val = start;
      prev = start - 2;
   }
   public void setStart(int x) {
      start = x;
      val = x;
      prev = x - 2;
   }
   int getPrevious() {
      return prev;
   }
}
```

Variables in Interfaces

Como se mencionó, las variables se pueden declarar en una interfaz, pero son implícitamente públicas, estáticas y finales. A primera vista, podría pensar que habría un uso muy limitado para tales variables, pero lo contrario es cierto. Los programas grandes generalmente utilizan varios valores constantes que describen cosas tales como el tamaño de la matriz, varios límites, valores especiales y similares. Debido a que un programa grande generalmente se guarda en una cantidad de archivos fuente separados, debe haber una manera conveniente de hacer que estas constantes

estén disponibles para cada archivo. En Java, las variables de interfaz ofrecen una solución. Para definir un conjunto de constantes compartidas, cree una interfaz que contenga solo estas Constantes, sin ningún método. Cada archivo que necesita acceso a las constantes simplemente "implementa" la interfaz. Esto trae las constantes a la vista. Aquí hay un ejemplo:

```
interface IConstant {

   int MIN = 0;
   int MAX = 10;
   String ERRORMSG = "Limite Error";
}

class IConstD implements IConstant {

   public static void main(String args[]) {
      int nums[] = new int[MAX];
      for (int i = MIN; i < 11; i++) {
         if (i >= MAX) {
            System.out.println(ERRORMSG);
         } else {
            nums[i] = i;
            System.out.print(nums[i] + " ");
         }
      }
   }
}
```

Interfaces Can Be Extended

Una interfaz puede heredar otra mediante el uso de la palabra clave se extents. La sintaxis es la misma que para heredar clases. Cuando una clase implementa una interfaz que hereda otra interfaz, debe proporcionar implementaciones para todos los métodos requeridos por la cadena de herencia de la interfaz.

```
interface A {

   void meth1();

   void meth2();
}
// B now includes meth1() and meth2() – it adds meth3().

interface B extends A {

   void meth3();
}
// This class must implement all of A and B

class MyClass implements B {

   public void meth1() {
      System.out.println("Implement meth1().");

   }

   public void meth2() {
      System.out.println("Implement meth2().");
   }

   public void meth3() {
      System.out.println("Implement meth3().");
   }
}

class IFExtend {

   public static void main(String args[]) {
      MyClass ob = new MyClass();
      ob.meth1();
      ob.meth2();
      ob.meth3();
   }
}
```

Ejercicio 1:

Crea una **interfaz** llamada *Televisor*:

- Que defina los siguientes métodos:
  void encender ( ), void apagar ( ), void cambiarCanal (int)

- Crea una clase llamada *MiTele* que implemente la interfaz definida.
  Con los atributos *int numCanales, int canalActual* y *boolean encendida*.
  Desarrolla el constructor y los métodos definidos en la interfaz.

- Crea una clase aparte con el método *main* para comprobar que no se puede instanciar la interfaz *Televisor*. Instancia la clase *MiTele* y comprueba su funcionamiento.

Ejercicio2:

Crea la siguiente interfaz que define distintas operaciones con una lista de objetos:

```
public interface Lista {

   void putInicio(Object obj);//Añade un objeto al principio de la lista
   void putFinal(Object obj);//Añade un objeto al final de la lista
   Object popPrincipio();//Obtiene y elimina el primer objeto de la lista
   Object popFinal();//Obtiene y elimina el último objeto de la lista
   Object getAt(int i);//Obtiene y elimina el elemento i de la lista
   int count();//Devuelve el número de objetos de la lista
}
```

Realiza 2 implementaciones de la interfaz anterior:

a) La primera utilizará un ArrayList para almacenar los objetos.

b) La segunda utilizará un LinkedList para almacenar los objetos.

Completa el proyecto definiendo una clase Objeto similar a la siguiente:

```
public class Objeto {

   int dato1;
   String dato2;

   //Constructores
   //Métodos set y get
}
```

Finalmente, escribe una clase que contenga el método main para probar las implementaciones anteriores.