Tema 7.1

UTILIZACIÓN AVANZADA DE CLASES: INTERFACES

1. INTERFACES

- En POO, algunas veces nos interesa definir qué debe hacer una clase, pero no cómo lo hace.
- Ya hemos visto un ejemplo de esto: un método abstracto, el cual define la signatura del método pero no provee su implementación.
- Una subclase debe proveer su propia implementación de cada método abstracto definido por su superclase.
- Entonces, un método abstracto especifica la **interface** del método pero no su **implementación**.

1. INTERFACES

- Mientras las clases y los métodos abstractos son útiles en Java, es posible dar un paso más.
- En Java, se puede separar por completo la interfaz de una clase de su implementación usando la palabra reservada **interface**.
- Una interfaz es sintácticamente similar a una clase abstracta, en la que se puede especificar uno o más métodos que no tienen cuerpo.
- Estos métodos deben ser implementados por una clase para definir sus acciones.
- Por lo tanto, una interface especifica lo que se debe hacer, pero no cómo hacerlo.

1. INTERFACES

- Una vez que una interface es definida, cualquier número de clases puede implementarla.
- Además, una clase puede implementar de cualquier cantidad de interfaces.
- Para implementar una interface, una clase debe proporcionar la implementación de los métodos descritos en la interface.
- Cada clase es libre para determinar los detalles de su implementación.
- Dos clases podrían implementar la misma interfaz de diferentes formas pero cada clase tendría los mismos métodos.

1. INTERFACES

- Cabe señalar que antes de JDK 8 una interfaz no podía definir la implementación de ningún método.
- Por lo tanto, antes de JDK 8, una interface podía definir sólo qué, pero no cómo.
- JDK 8 cambió esto y hoy es posible añadir una implementación por defecto al método de una interfaz.
- Por lo que ahora es posible que una interface especifique algún comportamiento.
- Sin embargo, esto debe verse como una característica especial y no se debe perder de vista la intención original para la creación de interfaces.

1. INTERFACES

- En resumen, como regla general, crearemos y usaremos interfaces en las que no existan método por defecto.
- La sintaxis básica sería:

```
modificador-acceso interface name {
        type var1 = value;
        type var2 = value;
        ....
        type varN = value;
        ret-type method-name1(param-list);
        ret-type method-name2(param-list);
         ...
        ret-type method-nameN(param-list);
}
```

1. INTERFACES

- En la manera tradicional de entender las interfaces, los métodos son declarados únicamente con el valor de retorno y su signatura.
- Por lo que son, en esencia, métodos abstractos.
- En una interface, los métodos son public implícitamente.
- Las variables declaradas en una interface no son variables de instancia. Son implícitamente **public**, **final**, y **static** y deben ser inicializadas. Por lo tanto, son esencialmente constantes.
- Ejemplo:

```
public interface Series {

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

    void reset(); // restart

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

2. Implementando interfaces

- Una vez que una interface ha sido definida, una o más clases pueden implementar de esta interface.
- Para implementar de una interface se incluye la palabra **implements** en la definición de la clase y luego se crearán los métodos requeridos por la interface.
- Cuando una clase implementa una interface, la sintaxis a utilizar es la siguiente:

class *classname* extends *superclass* **implements *interface*** {

    // class-body

}

** En este ejemplo, la clase también está heredando de una superclase.

2. Implementando interfaces

- Para implementar de más de una interfaz, las interfaces se separan por una coma.
- Los métodos que implementan una interface deben ser públicos y su signatura debe ser exactamente la misma que la que se ha especificado en la definición de la interface.

2. Implementando interfaces

- Ejemplo:

```
class ByTwos implements Series {
    int start;
    int val;

    ByTwos() {
        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;
    }
}
```

2. Implementando interfaces

- Ejemplo de uso:

```
class SeriesDemo {
    public static void main(String args[]) {
        ByTwos ob = new ByTwos();
        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());
        }
    }
}
```

2. Implementando interfaces

- La salida sería:

```
Next value is 2
Next value is 4
Next value is 6
Next value is 8
Next value is 10

Resetting
Next value is 2
Next value is 4
Next value is 6
Next value is 8
Next value is 10

Starting at 100
Next value is 102
Next value is 104
Next value is 106
Next value is 108
Next value is 110
```

2. Implementando interfaces

- Las subclases que implementan de interfaces pueden implementar además métodos propios que no estén indicados en la interface.

```
class ByTwosBis implements Series {

    int start;
    int val;
    int prev;

    ByTwosBis() {
        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;
    }
}
```

3. Variables en interfaces

- Como ya hemos mencionado, se pueden declarar variables en una interface, pero son implicitamente **public**, **static** y **final**.
- Se usan a menudo ya que es muy típico hacer uso de varios valores constantes a lo largo de las clases como tamaños de arrays, enumeraciones, etc.
- Los programas grandes normalmente hacen uso de varios valores constantes que describen cosas como el tamaño de la matriz, diversos límites, valores especiales, etc.
- Dado que un programa grande generalmente se mantiene en una cantidad de archivos fuente separados, debe haber una forma conveniente de hacer que estas constantes estén disponibles para cada archivo.

3. Variables en interfaces

- En Java, las variables de interfaz ofrecen una solución.
- Para definir un conjunto de constantes compartidas por varios ficheros, crea una interface de variables que contenga dichas constantes, sin ningún método.
- Cada fichero que necesite acceder a esas constantes, simplemente implementará la interface.
- Ejemplo:

```
interface IConstant {

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

3. Variables en interfaces

```
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] + " ");
            }
        }
    }
}
```

4. Herencia en interfaces

- Una interface puede heredar de otra usando la palabra reservada **extends**.
- Cuando una clase implementa de una interface que hereda de otra interface, la clase debe proveer implementaciones para todos los métodos requeridos por la cadena de herencia de la interface.
- Ejemplo:

```
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
```

4. Herencia en interfaces

- Ejemplo:

```
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();
  }
}
```

Actividad

- Realiza los ejercicios del 10 al 13 de la Hoja de Ejercicios