7.E. Interfaces.

1. Interfaces.

1.4. Implementación de interfaces.

Como ya has visto, todas las clases que implementan una determinada **interfaz** están obligadas a proporcionar una **definición (implementación) de los métodos de esa interfaz**, adoptando el modelo de comportamiento propuesto por ésta.

Dada una **interfaz**, cualquier clase puede especificar dicha **interfaz** mediante el mecanismo denominado **implementación de interfaces**. Para ello se utiliza la palabra reservada `implements`:

```
class NombreClase implements NombreInterfaz {
```

De esta manera, la clase está diciendo algo así como "**la interfaz indica los métodos que debo implementar, pero voy a ser yo (la clase) quien los implemente**".

Es posible indicar varios nombres de **interfaces** separándolos por comas:

```
class NombreClase implements NombreInterfaz1, NombreInterfaz2,... {
```

Cuando una clase implementa una **interfaz**, tiene que redefinir sus métodos nuevamente con **acceso público**. Con otro tipo de acceso se producirá un **error de compilación**. Es decir, que del mismo modo que no se podían restringir permisos de acceso en la **herencia de clases**, tampoco se puede hacer en la **implementación de interfaces**.

Una vez implementada una **interfaz** en una clase, los métodos de esa interfaz tienen exactamente el mismo tratamiento que cualquier otro método, sin ninguna diferencia, pudiendo ser invocados, heredados, redefinidos, etc.

En el ejemplo de los depredadores, al definir la clase **León**, habría que indicar que implementa la **interfaz Depredador**:

```
class Leon implements Depredador {
```

Y en su interior habría que implementar aquellos métodos que contenga la **interfaz**:

```
void localizar (Animal presa) {
```

```
// Implementación del método localizar para un león
```

```
...
```

```
}
```

En el caso de clases que pudieran ser a la vez **Depredador** y **Presa**, tendrían que implementar ambas interfaces, como podría suceder con la clase **Rana**:

```
class Rana implements Depredador, Presa {
```

Y en su interior habría que implementar aquellos métodos que contengan ambas **interfaces**, tanto las de **Depredador** (**localizar**, **cazar**, etc.) como las de **Presa** (**observar**, **huir**, etc.).

Autoevaluación

¿Qué palabra reservada se utiliza en Java para indicar que una clase va a definir los métodos indicados por una interfaz?

- ○ implements.
- ○ uses.
- ○ extends.
- ○ Los métodos indicados por una interfaz no se definen en las clases pues sólo se pueden utilizar desde la propia interfaz.

Ejercicio resuelto

Haz que las clases `Alumno` y `Profesor` implementen la interfaz `Imprimible` que se ha escrito en el ejercicio anterior.

Solución:

La primera opción que se te puede ocurrir es pensar que en ambas clases habrá que indicar que implementan la interfaz Imprimible y por

tanto definir los métodos que ésta incluye: devolverContenidoString, devolverContenidoHashtable y devolverContenidoArrayList.

Si las clases Alumno y Profesor no heredaran de la misma clase habría que hacerlo obligatoriamente así, pues no comparten superclase y precisamente para eso sirven las interfaces: para implementar determinados comportamientos que no pertenecen a la estructura jerárquica de herencia en la que se encuentra una clase (de esta manera, clases que no tienen ninguna relación de herencia podrían compartir interfaz).

Pero en este caso podríamos aprovechar que ambas clases sí son subclases de una misma superclase (heredan de la misma) y hacer que la interfaz Imprimible sea implementada directamente por la superclase (Persona) y de este modo ahorrarnos bastante código. Así no haría falta indicar explícitamente que Alumno y Profesor implementan la interfaz Imprimible, pues lo estarán haciendo de forma implícita al heredar de una clase que ya ha implementado esa interfaz (la clase Persona, que es padre de ambas).

Una vez que los métodos de la interfaz estén implementados en la clase Persona, tan solo habrá que redefinir o ampliar los métodos de la interfaz para que se adapten a cada clase hija específica (Alumno o Profesor), ahorrándonos tener que escribir varias veces la parte de código que obtiene los atributos genéricos de la clase Persona.

1. Clase Persona.

Indicamos que se va a implementar la interfaz Imprimible:

public abstract class Persona implements Imprimible {

...

Definimos el método devolverContenidoHashtable a la manera de como debe ser implementado para la clase Persona. Podría quedar, por ejemplo, así:

public Hashtable devolverContenidoHashtable () {

// Creamos la Hashtable que va a ser devuelta

Hashtable contenido= new Hashtable ();

// Añadimos los atributos de la clase

SimpleDateFormat formatoFecha = new SimpleDateFormat("dd/MM/yyyy");

String stringFecha= formatoFecha.format(this.fechaNacim.getTime());

contenido.put ("nombre", this.nombre);

contenido.put ("apellidos", this.apellidos);

contenido.put ("fechaNacim", stringFecha);

// Devolvemos la Hashtable

return contenido;

}

Del mismo modo, definimos también el método devolverContenidoArrayList:

public ArrayList devolverContenidoArrayList () { ... }

Y por último el método devolverContenidoString:

public String devolverContenidoString () { ... }

2. Clase Alumno.

Esta clase hereda de la clase Persona, de manera que heredará los tres métodos anteriores. Tan solo habrá que redefinirlos para que, aprovechando el código ya escrito en la superclase, se añada la funcionalidad específica que aporta esta subclase.

public class Alumno extends Persona {

...

Como puedes observar no ha sido necesario incluir el implements Imprimible, pues el extends Persona lo lleva implícito dado que Persona ya implementaba ese **interfaz**. Lo que haremos entonces será llamar al método que estamos redefiniendo utilizando la referencia a la **superclase super**.

El método devolverContenidoHashtable podría quedar, por ejemplo, así:

public Hashtable devolverContenidoHashtable () {

// Llamada al método de la superclase

Hashtable contenido= **super.devolverContenidoHashtable()**;

// Añadimos los atributos específicos de la clase

contenido.put ("grupo", this.salario);

contenido.put ("notaMedia", this.especialidad);

// Devolvemos la Hashtable rellena

return contenido;

}

3. **Clase** Profesor.

En este caso habría que proceder exactamente de la misma manera que con la clase Alumno: redefiniendo los métodos de la interfaz **Imprimible** para añadir la funcionalidad específica que aporta esta **subclase**.

La solución completa será:

Imprimible.java (Interfaz)

/*

* Interfaz Imprimible

*/

package ejemplointerfazimprimible;

```
import java.util.Hashtable;
import java.util.ArrayList;

/**
 *
 * Interfaz Imprimible
 */
public interface Imprimible {
        String devolverContenidoString ();
        ArrayList devolverContenidoArrayList ();
        Hashtable devolverContenidoHashtable ();

}
```

Persona.java

```
/*
 * Clase Persona
 */
package ejemplointerfazimprimible;
import java.util.GregorianCalendar;
import java.util.Hashtable;
import java.util.ArrayList;
import java.util.Enumeration;
import java.text.SimpleDateFormat;

/**
 * Clase Persona
 */
public abstract class Persona implements Imprimible {
    protected String nombre;
    protected String apellidos;
    protected GregorianCalendar fechaNacim;
    // Constructores
    // -------------
    // Constructor
    public Persona (String nombre, String apellidos, GregorianCalendar fechaNacim) {
        this.nombre= nombre;
        this.apellidos= apellidos;
        this.fechaNacim= (GregorianCalendar) fechaNacim.clone();
```

```
}

// Métodos get
// ----------

// Método getNombre
protected String getNombre (){
   return nombre;
}

// Método getApellidos
protected String getApellidos (){
   return apellidos;
}

// Método getFechaNacim
protected GregorianCalendar getFechaNacim (){
   return this.fechaNacim;
}

// Métodos set
// ----------

// Método setNombre
protected void setNombre (String nombre){
   this.nombre= nombre;
}

// Método setApellidos
protected void setApellidos (String apellidos){
   this.apellidos= apellidos;
}

// Método setFechaNacim
protected void setFechaNacim (GregorianCalendar fechaNacim){
   this.fechaNacim= fechaNacim;
}

// Implementación de los métodos de la interfaz Imprimible
```

```
// ------------------------------------------------------------

// Método devolverContenidoHashtable
public Hashtable devolverContenidoHashtable () {
    // Creamos la Hashtable que va a ser devuelta
    Hashtable contenido= new Hashtable ();

    // Añadimos los atributos específicos
    SimpleDateFormat formatoFecha = new SimpleDateFormat("dd/MM/yyyy");
    String stringFecha= formatoFecha.format(this.fechaNacim.getTime());
    contenido.put ("nombre", this.nombre);
    contenido.put ("apellidos", this.apellidos);
    contenido.put ("fechaNacim", stringFecha);

    // Devolvemos la Hashtable
    return contenido;
}

// Método devolverContenidoArrayList
public ArrayList devolverContenidoArrayList () {
    ArrayList contenido= new ArrayList ();
    SimpleDateFormat formatoFecha = new SimpleDateFormat("dd/MM/yyyy");
    String stringFecha= formatoFecha.format(this.fechaNacim.getTime());

    contenido.add(this.nombre);
    contenido.add (this.apellidos);
    contenido.add(stringFecha);

    return contenido;
}



// Método devolverContenidoString
public String devolverContenidoString () {
    String contenido= Persona.HashtableToString(this.devolverContenidoHashtable());
    return contenido;
}

// Métodos estáticos (herramientas)
```

```
        // --------------------------------

        // Método HashtableToString
        protected static String HashtableToString (Hashtable tabla) {
           String contenido;
           String clave;
           Enumeration claves= tabla.keys();

           contenido= "{";
           if (claves.hasMoreElements()) {
              clave= claves.nextElement().toString();
              contenido= contenido + clave + "=" + tabla.get(clave).toString();
           }
           while (claves.hasMoreElements()) {
                 clave= claves.nextElement().toString();
                 contenido += ",";
                 contenido= contenido.concat (clave) ;
                 contenido= contenido.concat ("=" + tabla.get(clave));
           }
           contenido= contenido + "}";

           return contenido;
        }
}
```

Alumno.java

```
/*
 * Clase Alumno.
 */
package ejemplointerfazimprimible;
import java.util.*;
import java.text.*;

/**
 *
 * Clase Alumno
 */
public class Alumno extends Persona  {
        protected String grupo;
```

```
protected double notaMedia;



// Constructor
// -----------
public Alumno (String nombre, String apellidos, GregorianCalendar fechaNacim, String grupo, double notaMedia) {
   super (nombre, apellidos, fechaNacim);
   this.grupo= grupo;
   this.notaMedia= notaMedia;
}

// Métodos get
// ------------

// Método getGrupo
public String getGrupo (){
   return grupo;
}

// Método getNotaMedia
public double getNotaMedia (){
   return notaMedia;
}

// Métodos set
// -----------

// Método setGrupo
public void setGrupo (String grupo){
   this.grupo= grupo;
}

// Método setNotaMedia
public void setNotaMedia (double notaMedia){
   this.notaMedia= notaMedia;
}

// Redefinición de los métodos de la interfaz Imprimible
// -----------------------------------------------------
```

```
    // Método devolverContenidoHashtable
    @Override
    public Hashtable devolverContenidoHashtable () {
        // Llamada al método de la superclase
        Hashtable contenido= super.devolverContenidoHashtable();
        // Añadimos los atributos específicos
        contenido.put ("grupo", this.grupo);
        contenido.put ("notaMedia", this.notaMedia);
        // Devolvemos la Hashtable rellena
        return contenido;
    }

    // Método devolverContenidoArray
    @Override
    public ArrayList devolverContenidoArrayList () {
        // Llamada al método de la superclase
        ArrayList contenido= super.devolverContenidoArrayList ();
        // Añadimos los atributos específicos
        contenido.add(this.grupo);
        contenido.add (this.notaMedia);
        // Devolvemos el ArrayList relleno
        return contenido;
    }

    // Método devolverContenidoString
    @Override
    public String devolverContenidoString () {
        // Aprovechamos el método estático para transformar una Hashtable en String
        String contenido= Persona.HashtableToString(this.devolverContenidoHashtable());
        // Devolvemos el String creado.
        return contenido;
    }
}
```

Profesor.java

```
/*
 * Clase Profesor
 */
```

```
package ejemplointerfazimprimible;

/**

 */
import java.util.*;
import java.text.*;

/**
 *
 * Clase Profesor
 */
public class Profesor extends Persona {
        String especialidad;
        double salario;

    // Constructor
    // -----------

    public Profesor (String nombre, String apellidos, GregorianCalendar fechaNacim, String especialidad, double salario) {
       super (nombre, apellidos, fechaNacim);
       this.especialidad= especialidad;
       this.salario= salario;
    }

    // Métodos get
    // -----------

    // Método getEspecialidad
    public String getEspecialidad (){
       return especialidad;
    }

    // Método getSalario
    public double getSalario (){
       return salario;
    }

    // Métodos set
```

```
// ----------

// Método setSalario
public void setSalario (double salario){
    this.salario= salario;
}

// Método setESpecialidad
public void setESpecialidad (String especialidad){
    this.especialidad= especialidad;
}



// Redefinición de los métodos de la interfaz Imprimible
// ------------------------------------------------------

// Método devolverContenidoHashtable
@Override
public Hashtable devolverContenidoHashtable () {
    // Llamada al método de la superclase
    Hashtable contenido= super.devolverContenidoHashtable();
    // Añadimos los atributos específicos
    contenido.put ("salario", this.salario);
    contenido.put ("especialidad", this.especialidad);
    // Devolvemos la Hashtable rellena
    return contenido;
}

// Método devolverContenidoArrayList
@Override
public ArrayList devolverContenidoArrayList () {
    // Llamada al método de la superclase
    ArrayList contenido= super.devolverContenidoArrayList ();
    // Añadimos los atributos específicos
    contenido.add(this.salario);
    contenido.add (this.especialidad);
    // Devolvemos el ArrayList relleno
    return contenido;
}
```

```
        // Método devolverContenidoString
        @Override
        public String devolverContenidoString () {
            // Aprovechamos el método estático para transformar una Hashtable en String
            String contenido= Persona.HashtableToString(this.devolverContenidoHashtable());
            // Devolvemos el String creado.
            return contenido;
        }
}
```

EjemploInterfazImprimible.java

```
/*
 * Ejemplo de utilización de clases que implementan la interfaz Imprimible.
 */
package ejemplointerfazimprimible;

import java.util.GregorianCalendar;
import java.util.ArrayList;
import java.util.Hashtable;

/**
 *
 * Ejemplo de utilización de clases que implementan la interfaz Imprimible
 */
public class EjemploInterfazImprimible {

    /**
     * Clase principal
     */
    public static void main(String[] args) {
        String stringContenidoAlumno, stringContenidoProfesor;
        Hashtable hashtableContenidoAlumno, hashtableContenidoProfesor;
        ArrayList listaContenidoAlumno, listaContenidoProfesor;

        // Creación de objetos alumno y profesor
        Alumno al1= new Alumno ("Juan", "Torres Mula", new GregorianCalendar (1990, 10, 6), "1DAM-B", 7.5);
        Profesor prof1= new Profesor  ("Antonio", "Campos Pin", new GregorianCalendar (1970, 8, 15), "Informática", 2000);
```

```
        // Obtención del contenido del objeto alumno a través de los métodos del interfaz Imprimible
        stringContenidoAlumno= al1.devolverContenidoString();
        hashtableContenidoAlumno= al1.devolverContenidoHashtable();
        listaContenidoAlumno= al1.devolverContenidoArrayList();

        // Obtención del contenido del objeto profesor a través de los métodos del interfaz Imprimible
        stringContenidoProfesor= prof1.devolverContenidoString();
        hashtableContenidoProfesor= prof1.devolverContenidoHashtable();
        listaContenidoProfesor= prof1.devolverContenidoArrayList();

        // Impresión en pantalla del contenido del objeto alumno a través de las estructuras obtenidas
        System.out.printf ("Contenido del objeto alumno: %s\n", stringContenidoAlumno);

        // Impresión en pantalla del contenido del objeto alumno a través de las estructuras obtenidas
        System.out.printf ("Contenido del objeto profesor: %s\n", stringContenidoProfesor);


    }
}
```