7.F. Polimorfismo.

1. Polimorfismo.

1.3. Limitaciones de la ligadura dinámica.

Como has podido comprobar, el **polimorfismo** se basa en la utilización de **referencias** de un tipo más "amplio" (**superclases**) que los objetos a los que luego realmente van a apuntar (**subclases**). Ahora bien, existe una importante **restricción** en el uso de esta capacidad, pues el tipo de referencia limita cuáles son los métodos que se pueden utilizar y los atributos a los que se pueden acceder.

No se puede acceder a los **miembros específicos** de una **subclase** a través de una **referencia** a una **superclase**. Sólo se pueden utilizar los miembros declarados en la **superclase**, aunque la definición que finalmente se utilice en su ejecución sea la de la **subclase**.

Veamos un ejemplo: si dispones de una clase **A** que es subclase de **B** y declaras una variable como referencia un objeto de tipo **B**. Aunque más tarde esa variable haga referencia a un objeto de tipo **A** (**subclase**), los miembros a los que podrás acceder sin que el compilador produzca un error serán los miembros de **A** que hayan sido heredados de **B** (**superclase**). De este modo, se garantiza que los métodos que se intenten llamar van a existir cualquiera que sea la subclase de **B** a la que se apunte desde esa referencia.

En el ejemplo de las clases **Persona**, **Profesor** y **Alumno**, el **polimorfismo** nos permitiría declarar variables de tipo **Persona** y más tarde hacer con ellas referencia a objetos de tipo **Profesor** o **Alumno**, pero no deberíamos intentar acceder con esa variable a métodos que sean específicos de la clase **Profesor** o de la clase **Alumno**, tan solo a métodos que sabemos que van a existir seguro en ambos tipos de objetos (métodos de la **superclase Persona**).

Ejercicio resuelto

Haz un pequeño programa en Java en el que se declare una variable de tipo `Persona`, se pidan algunos datos sobre esa persona (**nombre**, **apellidos** y si es **alumno** o si es **profesor**), y se muestren nuevamente esos datos en pantalla, teniendo en cuenta que esa variable no puede ser instanciada como un objeto de tipo `Persona` (es una **clase abstracta**) y que tendrás que instanciarla como `Alumno` o como `Profesor`. Recuerda que para poder recuperar sus datos necesitarás hacer uso de la **ligadura dinámica** y que tan solo deberías acceder a métodos que sean de la **superclase**.

Solución:

Si tuviéramos diferentes variables referencia a objetos de las clases `Alumno` y `Profesor`tendrías algo así:

```
Alumno obj1;
```

```
Profesor obj2:
```

```
...
```

```
// Si se dan ciertas condiciones el objeto será de tipo Alumno y lo tendrás en obj1
```

```
System.out.printf ("Nombre: %s\n", obj1.getNombre());
```

```
// Si se dan otras condiciones el objeto será de tipo Profesor y lo tendrás en obj2
```

```
System.out.printf ("Nombre: %s\n", obj2.getNombre());
```

Pero si pudieras tratar de una manera más genérica la situación, podrías intentar algo así:

```
Persona obj;
```

```
// Si se dan ciertas condiciones el objeto será de tipo Alumno y por tanto lo instanciarás como tal
```

```
obj = new Alumno (<parámetros>);
```

```
// Si se otras condiciones el objeto será de tipo Profesor y por tanto lo instanciarás como tal
```

```
obj = new Profesor (<parámetros>);
```

De esta manera la variable `obj` obj podría contener una referencia a un objeto de la **superclase** `Persona` de **subclase** `Alumno` o bien de **subclase** `Profesor` (**polimorfismo**).

Esto significa que independientemente del tipo de **subclase** que sea (`Alumno` o `Profesor`), podrás invocar a métodos de la **superclase** `Persona` y durante la ejecución se resolverán como métodos de alguna de sus **subclases**.

```
//En tiempo de compilación no se sabrá de qué subclase de Persona será obj.
```

```
//Habrá que esperar la ejecución para que el entorno lo sepa e invoque al método adecuado.
```

```
System.out.printf ("Contenido del objeto usuario: %s\n", stringContenidoUsuario);
```

Por último recuerda que debes de proporcionar **constructores** a las **subclases** `Alumno` y `Profesor` que sean "compatibles" con algunos de los **constructores** de la **superclase** `Persona`, pues al llamar a un **constructor** de una **subclase**, su formato debe coincidir con el de algún **constructor** de la **superclase** (como debe suceder en general con cualquier método que sea invocado utilizando la **ligadura dinámica**).

Solución completa (ficheros):

Imprimible.java

```
/*
 * Interfaz Imprimible
 */
package ejemplopolimorfismopersona;

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

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

}
```

Persona.java

```
/*
 * Clase Persona
 */
package ejemplopolimorfismopersona;

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 ejemplopolimorfismopersona;
import java.util.*;
import java.text.*;

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



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

        public Alumno (String nombre, String apellidos, GregorianCalendar fechaNacim) {
            super (nombre, apellidos, fechaNacim);
            // Valores por omisión para un alumno: Grupo "GEN" y nota media de 0.
            this.grupo= "GEN";
            this.notaMedia= 0;
        }
```

```
// 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 ejemplopolimorfismopersona;

/**

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

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

        // Constructores
        // -------------

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

        public Profesor (String nombre, String apellidos, GregorianCalendar fechaNacim) {
            super (nombre, apellidos, fechaNacim);
            // Valores por omisión para un profesor: especialidad "GEN" y sueldo de 1000 euros.
            this.especialidad= "GEN";
            this.salario= 1000;
        }

        // 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;
        }
}
```

EjemploPolimorfismo.java

```
/*
 * Ejemplo de utilización del polimorfismo y la ligadura dinámica.
 */
package ejemplopolimorfismopersona;

import java.util.GregorianCalendar;
import java.util.Date;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.text.SimpleDateFormat;
import java.text.ParseException;
/**
 *
 * Ejemplo de utilización del polimorfismo y la ligadura dinámica.
 */
public class EjemploPolimorfismo {

    /**
     * Clase principal
     */
    public static void main(String[] args) {
        String stringContenidoUsuario;

        String nombre= null, apellidos= null, tipo= null;
```

```
Persona usuario= null;
GregorianCalendar fecha= null;

// PRESENTACIÓN
// ------------
System.out.printf ("PRUEBA DE USO DEl POLIMORFISMO Y LA LIGADURA DINÁMICA. \n");
System.out.printf ("-----------------------------------------------------\n\n");

// ENTRADA DE DATOS
// ----------------
// Nombre
System.out.print("Nombre del usuario: ");
try {
    nombre= lecturaTeclado();
}
catch (Exception e) {
    System.err.println(e.getMessage());
}

// Apellidos
System.out.print("Apellidos del usuario: ");
try {
    apellidos= lecturaTeclado();
}
catch (Exception e) {
    System.err.println(e.getMessage());
}

// Fecha de nacimiento
boolean fechaValida= true;
do {
    String stringFecha= null;
    SimpleDateFormat formatoFecha= null;
    Date dateFecha= null;

    System.out.print("Fecha de nacimiento del usuario (formato DD/MM/AAAA): ");
    try {
        stringFecha= lecturaTeclado();
    }
```

```
    catch (Exception e) {
        System.err.println(e.getMessage());
    }

    // Conversión del texto en fecha
    formatoFecha = new SimpleDateFormat("dd/MM/yyyy");
    try {
        dateFecha= formatoFecha.parse(stringFecha);
    } catch (ParseException e)  {
        fechaValida= false;
    }
    fecha= new GregorianCalendar ();
    fecha.setTime(dateFecha);

} while (!fechaValida);

// ¿Alumno o Profesor?
do {
    System.out.println("¿Es alumno(A) o profesor(P)?");
    try {
        tipo= lecturaTeclado();
    }
    catch (Exception e) {
        System.err.println(e.getMessage());
    }
    if (tipo.equals("P") || tipo.equals("p")) tipo="profesor";
    else if (tipo.equals("A") || tipo.equals("a")) tipo="alumno";
    else tipo="X";

} while (tipo.equals("X"));

// Creación del objeto usuario (desconocido en tiempo de compilación)
// Sabemos que será subclase de Persona, pero no sabemos si será Alumno o Profesor
// (dependerá de la ejecución)
if (tipo.equals("profesor")) {
    usuario= new Profesor (nombre, apellidos, fecha);
}
else if (tipo.equals("alumno")) {
    usuario=  new Alumno (nombre, apellidos, fecha);
```

```
        } else {

        }

        // Obtención del contenido del objeto usuario a través del método devolverContenidoString.
        // El método que se va a ajecutar aún no se sabe cuál es (ligadura dinámica), pues
        // este objeto usuario no sabemos si será Alumno o Profesor. Tan solo sabemos que será de la
        // superclase Persona. En tiempo de ejecución se sabrá de qué tipo de subclase se trata y será
        // también en ese momento cuando el entorno de ejecución pueda resolver qué método se ejecuta
        // (el de método devolverContenidoString de la clase Alumno o el de la clase Profesor)

        stringContenidoUsuario= usuario.devolverContenidoString();

        // Impresión en pantalla del contenido del objeto usuario a través de la estructura obtenida
        System.out.printf ("Contenido del objeto usuario: %s\n", stringContenidoUsuario);

    }


    //---------------------------------------------------------------
    // MÉTODO lecturaTeclado: Captura de una cadena de teclado
    //---------------------------------------------------------------
    private static String lecturaTeclado () throws Exception {
        try {
            InputStreamReader inputStreamReader = new InputStreamReader(System.in);
            BufferedReader reader = new BufferedReader(inputStreamReader);
            String line = reader.readLine();
            return line;
        }
        catch (Exception e) {
            throw e;

        }

    }
}
```