APRENDERAPROGRAMAR.COM

EJEMPLO EJERCICIO RESUELTO CON POLIMORFISMO, SOBREESCRITURA DE MÉTODOS Y HERENCIA JAVA. CÓDIGO (CU00691B)

Sección: Cursos
Categoría: Curso “Aprender programación Java desde cero”

Fecha revisión: 2029

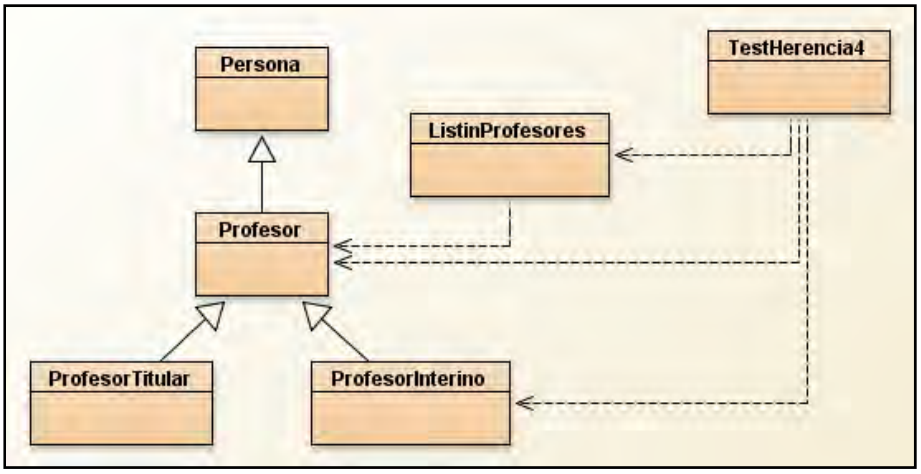
Resumen: Entrega nº91 curso Aprender programación Java desde cero.

Autor: Alex Rodríguez

EJERCICIO EJEMPLO RESUELTO POLIMORFISMO, HERENCIA, SOBREESCRITURA.

En apartados anteriores del tutorial hemos visto conceptos como herencia en Java, polimorfismo y sobreescritura de métodos. Vamos a plantear y desarrollar un ejercicio donde, a partir de un diagrama de clases, definimos el código que usa todos los conceptos estudiados.

Analiza el siguiente diagrama de clases. En él se pueden observar relaciones de herencia y relaciones de uso:



Trata de definir el código de las clases, estableciendo las relaciones de herencia y uso entre ellas. Trata de crear una clase con el método main (TestHerencia4) donde de alguna manera crees objetos de los distintos tipos y hagas uso de ellos, por ejemplo crea profesores interinos y titulares y luego recórrelos con un for extendido donde el tipo sea Profesor (uso del polimorfismo). Luego compáralo con las explicaciones y soluciones que damos a continuación.

En la solución que hemos planteado nosotros, en el tipo Profesor hemos incluido un método denominado mostrarDatos() que muestra los datos propios de un objeto Profesor. Luego, en las subclases ProfesorInterino y ProfesorTitular hemos sobreescrito el método mostrarDatos() de modo que en este caso únicamente muestra los datos específicos de los subtipos.

Por último, en la clase ListinProfesores simulamos un listín que admite todo tipo de profesores mediante un ArrayList que usa objetos de tipo Profesor, y que permite listar los profesores mediante un método *listar()* que lo que hace es invocar el método *mostrarDatos()* de los objetos contenidos en la lista. Si el método utilizado se basara en el tipo declarado en el código, *listar()* siempre nos devolvería

los datos de los objetos Profesor. Sin embargo, como veremos, esto no es así: cuando la variable apunta a un subtipo, el método invocado en tiempo de ejecución es el propio del subtipo, mientras que cuando la variable apunta a un tipo sí se invoca el método propio del tipo. Por eso decimos que Java hace una búsqueda dinámica del método: el método que se usa depende del tipo dinámico del objeto. Escribe este código:

```
//Código de la clase Persona ejemplo aprenderaprogramar.com
public class Persona {
   private String nombre; private String apellidos; private int edad;

   public Persona() { nombre = ""; apellidos = ""; edad = 0; }
   public Persona (String nombre, String apellidos, int edad) {
      this.nombre = nombre; this.apellidos = apellidos; this.edad = edad; }
   public String getNombre() { return nombre;  }
   public String getApellidos () { return apellidos;  }
   public int getEdad() { return edad;  }
} //Cierre de la clase
```

```
public class Profesor extends Persona {  //Ejemplo aprenderaprogramar.com
   private String IdProfesor;

   public Profesor () {      super();
      IdProfesor = "Unknown";}

   public Profesor (String nombre, String apellidos, int edad) {
      super(nombre, apellidos, edad);
      IdProfesor = "Unknown";   }

   public void setIdProfesor (String IdProfesor) { this.IdProfesor = IdProfesor;   }
   public String getIdProfesor () { return IdProfesor;   }
   public void mostrarDatos() {
      System.out.println ("Datos Profesor. Profesor de nombre: " + getNombre() + " " +  getApellidos() +
 " con Id de profesor: " + getIdProfesor() );   }
} //Cierre de la clase ejemplo aprenderaprogramar.com
```

```
import java.util.Calendar; //Ejemplo aprenderaprogramar.com
public class ProfesorInterino extends Profesor {

   private Calendar FechaComienzoInterinidad;
   public ProfesorInterino(Calendar fechaComienzaInterinidad) {
      super();
      FechaComienzoInterinidad = fechaComienzaInterinidad; }
   public ProfesorInterino (String nombre, String apellidos, int edad, Calendar fechaComienzaInterinidad) {
      super(nombre, apellidos, edad);
      FechaComienzoInterinidad = fechaComienzaInterinidad; }

   public Calendar getFechaComienzoInterinidad () { return FechaComienzoInterinidad; }
   public void mostrarDatos() { System.out.println("Datos ProfesorInterino. Comienzo interinidad: " +
 FechaComienzoInterinidad.getTime().toString() );  }
} //Cierre de la clase
```

```
import java.util.ArrayList; //Ejemplo aprenderaprogramar.com
public class ListinProfesores{
   private ArrayList <Profesor> listinProfesores;

   //Constructor
   public ListinProfesores () {
      listinProfesores = new ArrayList <Profesor> (); }

   //Métodos
   public void addProfesor (Profesor profesor) {
      listinProfesores.add(profesor); }    // Cierre método addProfesor

   public void listar() {
      System.out.println ("Se procede a mostrar los datos de los profesores existentes en el listín");
      for (Profesor tmp: listinProfesores) {      //Uso de for extendido
         tmp.mostrarDatos(); }
   } //Cierre método
} //Cierre de la clase
```

```
import java.util.Calendar; //Ejemplo aprenderaprogramar.com
public class TestHerencia4 {
   public static void main (String [ ] Args) {

      Profesor profesor1 = new Profesor ("Juan", "Hernández García", 33);
      profesor1.setIdProfesor("Prof 22-387-11");

      Calendar fecha1 = Calendar.getInstance();
      fecha1.set(2019,10,22); //Los meses van de 0 a 11, luego 10 representa noviembre
      ProfesorInterino interino1 = new ProfesorInterino("José Luis", "Morales Pérez", 54, fecha1);

      ListinProfesores listin1 = new ListinProfesores ();
      listin1.addProfesor(profesor1);
      listin1.addProfesor(interino1);
      listin1.listar(); } //Cierre del main

} //Cierre de la clase
```

```
Se procede a mostrar los datos de los profesores existentes en el listín

Datos Profesor. Profesor de nombre: Juan Hernández García con Id de profesor: Prof 22-387-11

Datos ProfesorInterino. Comienzo interinidad: Fri Nov 22 12:28:44 CET 2019
```

No hemos incluido el código de ProfesorTitular porque no lo utilizamos en el test. Si observamos el código de la clase ProfesorInterino vemos que el método mostrarDatos() está sobreescrito respecto al de su superclase Profesor. El método mostrarDatos() de la clase Profesor muestra nombre, apellidos e id de profesor, mientras que el método mostrarDatos() de la clase ProfesorInterino muestra la fecha de comienzo de la interinidad. En la clase ListinProfesores definimos un tipo que almacena objetos de tipo Profesor y su método listar() invoca el método mostrarDatos(). En la clase TestHerencia4 creamos un objeto Profesor y un objeto ProfesorInterino e introducimos ambos en un objeto de tipo ListinProfesores. ¿Qué ocurre cuando invocamos a listar() en el objeto donde tenemos una colección

con objetos de la superclase (Profesores) y objetos de la subclase (ProfesoresInterinos)? Que según sea el tipo dinámico del objeto, se usa el método mostrarDatos() con mayor cercanía. En el caso del objeto ProfesorInterino, se usa el método propio de los profesores interinos, aunque hayamos dicho que el ArrayList contiene objetos Profesor y aunque el bucle que invoca al método mostrarDatos() indique que el tipo que se usa es Profesor. Esto es debido a que toda variable de tipo Profesor es polimórfica y admite objetos de distintos tipos. A la hora de diseñar y dar nombres a las clases debes usar la lógica de la herencia y evitar nombres o relaciones entre clases que resulten contradictorios o contrarios a lo que sería lógico en el mundo real.

tmp.mostrarDatos() puede dar lugar a la ejecución del método de la clase ProfesorInterino, ProfesorTitular o Profesor, dependiendo del tipo dinámico al que apunte la variable.

Los objetos heredan los métodos de abajo hacia arriba, es decir, siempre tienen preferencia los métodos sobreescritos. Java hace una búsqueda dinámica del método aplicable empezando por el más próximo al tipo y escalando sucesivamente los supertipos hasta encontrar un método con la denominación especificada.

Próxima entrega: CU00692B

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:

http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=68&Itemid=188