APRENDERAPROGRAMAR.COM

CLASES Y MÉTODOS ABSTRACTOS EN JAVA. ABSTRACT CLASS. CLASES ABSTRACTAS DEL API. EJEMPLOS Y EJERCICIOS. (CU00695B)

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

Fecha revisión: 2029

Resumen: Entrega nº95 curso Aprender programación Java desde cero.

Autor: Alex Rodríguez

CLASES Y MÉTODOS ABSTRACTOS EN JAVA. ABSTRACT CLASS.

Supongamos un esquema de herencia que consta de la clase Profesor de la que heredan ProfesorInterino y ProfesorTitular. Es posible que todo profesor haya de ser o bien ProfesorInterino o bien ProfesorTitular, es decir, que no vayan a existir instancias de la clase Profesor. Entonces, ¿qué sentido tendría tener una clase Profesor?

El sentido está en que una superclase permite unificar campos y métodos de las subclases, evitando la repetición de código y unificando procesos. Ahora bien, una clase de la que no se tiene intención de crear objetos, sino que únicamente sirve para unificar datos u operaciones de subclases, puede declararse de forma especial en Java: como clase abstracta. La declaración de que una clase es abstracta se hace con la sintaxis **public abstract class NombreDeLaClase { … }**. Por ejemplo *public abstract class Profesor*. Cuando utilizamos esta sintaxis, no resulta posible instanciar la clase, es decir, no resulta posible crear objetos de ese tipo. Sin embargo, sigue funcionando como superclase de forma similar a como lo haría una superclase "normal". La diferencia principal radica en que no se pueden crear objetos de esta clase.

Declarar una clase abstracta es distinto a tener una clase de la que no se crean objetos. En una clase abstracta, no existe la posibilidad. En una clase normal, existe la posibilidad de crearlos aunque no lo hagamos. El hecho de que no creemos instancias de una clase no es suficiente para que Java considere que una clase es abstracta. Para lograr esto hemos de declarar explícitamente la clase como abstracta mediante la sintaxis que hemos indicado. Si una clase no se declara usando *abstract* se cataloga como "clase concreta". En inglés abstract significa "resumen", por eso en algunos textos en castellano a las clases abstractas se les llama resúmenes. Una clase abstracta para Java es una clase de la que nunca se van a crear instancias: simplemente va a servir como superclase a otras clases. No se puede usar la palabra clave *new* aplicada a clases abstractas. En el menú contextual de la clase en BlueJ simplemente no aparece, y si intentamos crear objetos en el código nos saltará un error.

A su vez, las clases abstractas suelen contener métodos abstractos: la situación es la misma. Para que un método se considere abstracto ha de incluir en su signatura la palabra clave *abstract*. Además un método abstracto tiene estas peculiaridades:

a) **No tiene cuerpo** (llaves): sólo consta de signatura con paréntesis.

b) Su signatura **termina con un punto y coma.**

c) **Sólo puede existir dentro de una clase abstracta.** De esta forma se evita que haya métodos que no se puedan ejecutar dentro de clases concretas. Visto de otra manera, si una clase incluye un método abstracto, forzosamente la clase será una clase abstracta.

d) Los métodos abstractos **forzosamente habrán de estar sobreescritos en las subclases.** Si una subclase no implementa un método abstracto de la superclase tiene un método no ejecutable, lo que la fuerza a ser una subclase abstracta. Para que la subclase sea concreta habrá de implementar métodos sobreescritos para todos los métodos abstractos de sus superclases.

Un método abstracto para Java es un método que nunca va a ser ejecutado porque no tiene cuerpo. Simplemente, un método abstracto referencia a otros métodos de las subclases. ¿Qué utilidad tiene un método abstracto? Podemos ver un método abstracto como una palanca que fuerza dos cosas: la primera, que no se puedan crear objetos de una clase. La segunda, que todas las subclases sobreescriban el método declarado como abstracto.

Sintaxis tipo: *abstract public/private/protected TipodeRetorno/void (parámetros …);*

Por ejemplo: *abstract public void generarNomina (int diasCotizados, boolean plusAntiguedad);*

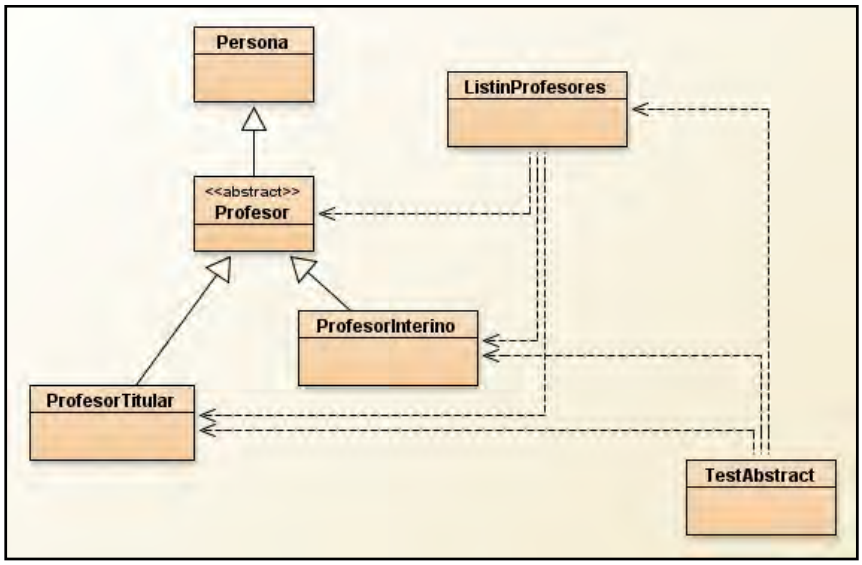
Que un método sea abstracto tiene otra implicación adicional: que podamos invocar el método abstracto sobre una variable de la superclase que apunta a un objeto de una subclase de modo que el método que se ejecute sea el correspondiente al tipo dinámico de la variable. En cierta manera, podríamos verlo como un método sobreescrito para que Java comprenda que debe buscar dinámicamente el método adecuado según la subclase a la que apunte la variable.

¿Es necesario que una clase que tiene uno o más métodos abstractos se defina como abstracta? Sí, si declaramos un método abstracto el compilador nos obliga a declarar la clase como abstracta porque si no lo hiciéramos así tendríamos un método de una clase concreta no ejecutable, y eso no es admitido por Java.

¿Una clase se puede declarar como abstracta y no contener métodos abstractos? Sí, una clase puede ser declarada como abstracta y no contener métodos abstractos. En algunos casos la clase abstracta simplemente sirve para efectuar operaciones comunes a subclases sin necesidad de métodos abstractos. En otros casos sí se usarán los métodos abstractos para referenciar operaciones en la clase abstracta al contenido de la sobreescritura en las subclases.

¿Una clase que hereda de una clase abstracta puede ser no abstracta? Sí, de hecho esta es una de las razones de ser de las clases abstractas. Una clase abstracta no puede ser instanciada, pero pueden crearse subclases concretas sobre la base de una clase abstracta, y crear instancias de estas subclases. Para ello hay que heredar de la clase abstracta y anular los métodos abstractos, es decir, implementarlos.

Vamos a ver un ejemplo basado en el siguiente esquema:



En este diagrama de clases vemos cómo hemos definido una clase abstracta denominada Profesor. BlueJ la identifica señalando <<abstract>> en la parte superior del icono de la clase. Sin embargo, hereda de la clase Persona que no es abstracta, lo cual significa que puede haber instancias de Persona pero no de Profesor.

El test que hemos diseñado se basa en lo siguiente: ProfesorTitular y ProfesorInterino son subclases de la clase abstracta Profesor. ListinProfesores sirve para crear un ArrayList de profesores que pueden ser tanto interinos como titulares y realizar operaciones con esos conjuntos. El listín se basa en el tipo estático Profesor, pero su contenido dinámico siempre será a base de instancias de ProfesorTitular o de ProfesorInterino ya que Profesor es una clase abstracta, no instanciable. En la clase de test creamos profesores interinos y profesores titulares y los vamos añadiendo a un listín. Posteriormente, invocamos el método imprimirListin, que se basa en los métodos toString de las subclases y de sus superclases mediante invocaciones sucesivas a super.

Por otro lado, en la clase ListinProfesores hemos definido el método importeTotalNominaProfesorado() que se basa en un bucle que calcula la nómina de todos los profesores que haya en el listín (sean interinos o titulares) mediante el uso de un método abstracto: importeNomina(). Este método está definido como *abstract public float importeNomina ();* dentro de la clase abstracta profesor, e implementado en las clases ProfesorInterino y ProfesorTitular. El aspecto central de este ejemplo es comprobar cómo una clase abstracta como Profesor nos permite realizar operaciones conjuntas sobre varias clases, **ahorrando código y ganando en claridad** para nuestros programas. Escribe este código:

```
public class Persona { //Código de la clase Persona ejemplo aprenderaprogramar.com
      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;  }
public String toString() {   Integer datoEdad = edad;
return "-Nombre: ".concat(nombre).concat(" -Apellidos: ").concat(apellidos).concat(" -Edad: ").concat(datoEdad.toString() ); }
} //Cierre de la clase
```

En la clase Persona transformamos edad en un Integer para poder aplicarle el método toString(). De otra manera no podemos hacerlo por ser edad un tipo primitivo. Escribe este código:

```
public abstract class Profesor extends Persona {
    // Campo de la clase ejemplo aprenderaprogramar.com
   private String IdProfesor;

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

   // Métodos
   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() );   }
   public String toString () { return super.toString().concat(" -IdProfesor: ").concat(IdProfesor); }
   abstract public float importeNomina ();  // Método abstracto
} //Cierre de la clase
```

Hemos declarado la clase Profesor como abstracta. De hecho, tenemos un método abstracto (definido como *abstract* y sin cuerpo), lo cual de facto nos obliga a declarar la clase como abstracta. El método sobreescrito toString llama al método toString de la superclase y lo concatena con nuevas cadenas. Como clases que heredan de Profesor tenemos a ProfesorTitular y ProfesorInterino:

```
public class ProfesorTitular extends Profesor {
       // Constructor ejemplo aprenderaprogramar.com
       public ProfesorTitular(String nombre, String apellidos, int edad, String id) {
        super(nombre, apellidos, edad, id); }
       public float importeNomina () { return 30f * 43.20f; }  //Método abstracto sobreescrito en esta clase
} //Cierre de la clase
```

```
import java.util.Calendar;

public class ProfesorInterino extends Profesor {
   // Campo de la clase ejemplo aprenderaprogramar.com
   private Calendar fechaComienzoInterinidad;

// Constructores
   public ProfesorInterino (Calendar fechaInicioInterinidad) {
      super();     fechaComienzoInterinidad = fechaInicioInterinidad; }

   public ProfesorInterino (String nombre, String apellidos, int edad, String id, Calendar fechaInicioInterinidad) {
       super(nombre, apellidos, edad, id);
       fechaComienzoInterinidad = fechaInicioInterinidad; }
   public Calendar getFechaComienzoInterinidad () { return fechaComienzoInterinidad; } //Método
   public String toString () { // Sobreescritura del método
 return super.toString().concat (" Fecha comienzo interinidad: ").concat (fechaComienzoInterinidad.getTime().toString()); }

   public float importeNomina () { return 30f * 35.60f ; } //Método abstracto sobreescrito en esta clase
} //Cierre de la clase
```

```
import java.util.ArrayList; import java.util.Iterator;
public class ListinProfesores {
   private ArrayList <Profesor> listinProfesores; //Campo de la clase
   public ListinProfesores () {  listinProfesores = new ArrayList <Profesor> ();  } //Constructor
   public void addProfesor (Profesor profesor) {  listinProfesores.add(profesor);  } //Método
   public void imprimirListin() {  //Método
      String tmpStr1 = ""; //String temporal que usamos como auxiliar
      System.out.println ("Se procede a mostrar los datos de los profesores existentes en el listín \n");
      for (Profesor tmp: listinProfesores) {        System.out.println (tmp.toString () );
                                          if (tmp instanceof ProfesorInterino) { tmpStr1 = "Interino";} else { tmpStr1 = "Titular"; }
       System.out.println("-Tipo de este profesor:"+tmpStr1+" -Nómina de este profesor: "+(tmp.importeNomina())+ "\n");}
   } //Cierre método imprimirListin
   public float importeTotalNominaProfesorado() {
      float importeTotal = 0f; //Variable temporal que usamos como auxiliar
      Iterator<Profesor> it = listinProfesores.iterator();
      while (it.hasNext() ) { importeTotal = importeTotal + it.next().importeNomina(); }
      return importeTotal;
   } //Cierre del método importeTotalNominaProfesorado
} //Cierre de la clase ejemplo aprenderaprogramar.com
```

ProfesorTitular y ProfesorInterino se han definido como clases concretas que heredan de la clase abstracta Profesor. Ambas clases redefinen (obligatoriamente han de hacerlo) el método abstracto importeNomina() de la superclase. El método sobreescrito toString() de la clase ProfesorInterino llama al método toString() de la superclase y lo concatena con nuevas cadenas. El cálculo de importeNomina en ambas clases es una trivialidad: hemos incluido un cálculo sin mayor interés excepto que el de ver el funcionamiento de la implementación de métodos abstractos. ProfesorTitular lo hemos dejado con escaso contenido porque aquí lo usamos solo a modo de ejemplo de uso de clases abstractas y herencia. Su único cometido es mostrar que existe otra subclase de Profesor. Por otro lado, en la clase ListinProfesores tenemos un ejemplo de uso de *instanceof* para determinar qué tipo (ProfesorInterino o ProfesorTitular) es el que porta una variable Profesor. Iteramos con clase declarada Profesor y clases dinámicas ProfesorTitular y ProfesorInterino. Dinámicamente se determina de qué tipo es cada objeto y al invocar el método abstracto importeNomina() **Java determina si debe utilizar el método propio de un subtipo u otro.** En *imprimirListin* llegamos incluso a mostrar por pantalla de qué tipo es cada objeto usando la sentencia *instanceof* para determinarlo. Escribe y ejecuta el código del test:

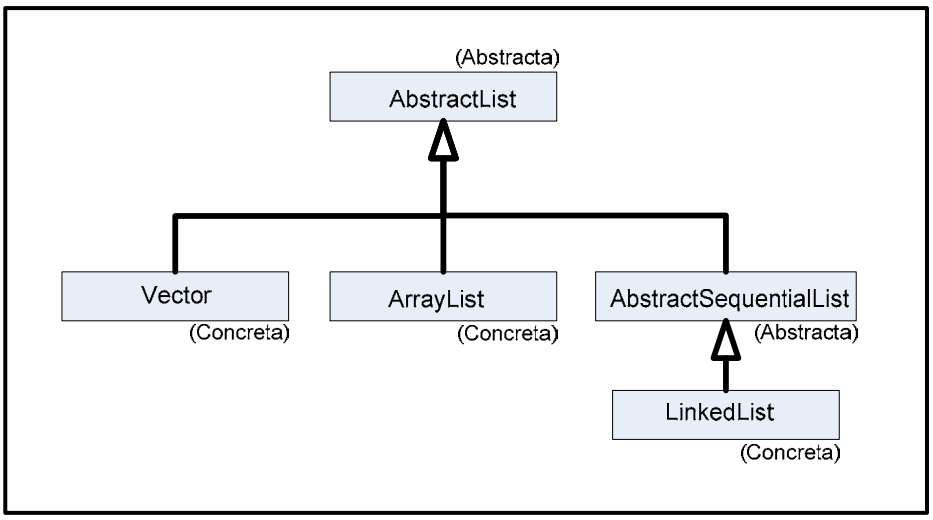
```
import java.util.Calendar; //Ejemplo aprenderaprogramar.com
public class TestAbstract {
   public static void main (String [ ] Args) {
      Calendar fecha1 = Calendar.getInstance();
      fecha1.set(2019,10,22); //Los meses van de 0 a 11, luego 10 representa noviembre
      ProfesorInterino pi1 = new ProfesorInterino("José", "Hernández López", 45, "45221887-K", fecha1);
      ProfesorInterino pi2 = new ProfesorInterino("Andrés", "Moltó Parra", 87, "72332634-L", fecha1);
      ProfesorInterino pi3 = new ProfesorInterino ("José", "Ríos Mesa", 76, "34998128-M", fecha1);
      ProfesorTitular pt1 = new ProfesorTitular ("Juan", "Pérez Pérez", 23, "73-K");
      ProfesorTitular pt2 = new ProfesorTitular ("Alberto", "Centa Mota", 49, "88-L");
      ProfesorTitular pt3 = new ProfesorTitular ("Alberto", "Centa Mota", 49, "81-F");
      ListinProfesores listinProfesorado = new ListinProfesores ();
      listinProfesorado.addProfesor (pi1); listinProfesorado.addProfesor(pi2); listinProfesorado.addProfesor (pi3);
      listinProfesorado.addProfesor (pt1); listinProfesorado.addProfesor(pt2); listinProfesorado.addProfesor (pt3);
      listinProfesorado.imprimirListin();
      System.out.println ("El importe de las nóminas del profesorado que consta en el listín es " +
      listinProfesorado.importeTotalNominaProfesorado()+ " euros");
   } }  //Cierre del main y cierre de la clase
```

Comprueba el resultado de ejecución. El resultado del test nos muestra que operamos exitosamente sobre las dos clases usando abstracción:

```
Se procede a mostrar los datos de los profesores existentes en el listín
-Nombre: José -Apellidos: Hdez López -Edad: 45 -IdProfesor: 45221887-K Fecha comienzo interinidad: Fri Nov 22 11:55:28 CET 2019
-Tipo de este profesor: Interino -Nómina de este profesor: 1068.0
-Nombre: Andrés -Apellidos: Moltó Parra -Edad: 87 -IdProfesor: 72332634-L Fecha comienzo interinidad: Fri Nov 22 11:55:28 CET 2019
-Tipo de este profesor: Interino -Nómina de este profesor: 1068.0
-Nombre: José -Apellidos: Ríos Mesa -Edad: 76 -IdProfesor: 34998128-M Fecha comienzo interinidad: Fri Nov 22 11:55:28 CET 2019
-Tipo de este profesor: Interino -Nómina de este profesor: 1068.0
-Nombre: Juan -Apellidos: Pérez Pérez -Edad: 23 -IdProfesor: 73-K
-Tipo de este profesor: Titular -Nómina de este profesor: 1296.0
-Nombre: Alberto -Apellidos: Centa Mota -Edad: 49 -IdProfesor: 88-L
-Tipo de este profesor: Titular -Nómina de este profesor: 1296.0
-Nombre: Alberto -Apellidos: Centa Mota -Edad: 49 -IdProfesor: 81-F
-Tipo de este profesor: Titular -Nómina de este profesor: 1296.0
El importe de las nóminas del profesorado que consta en el listín es 7092.0 euros
```

CLASES ABSTRACTAS EN EL API DE JAVA

Java utiliza clases abstractas en el API de la misma forma que podemos nosotros usarlas en nuestros programas. Por ejemplo, la clase AbstractList del paquete java.util es una clase abstracta con tres subclases:



Como vemos, entre las subclases dos de ellas son concretas mientras que una todavía es abstracta. En una clase como AbstractList algunos métodos son abstractos, lo que obliga a que el método esté sobreescrito en las subclases, mientras que otros métodos no son abstractos.

Sobre un objeto de una subclase, llamar a un método puede dar lugar a:

a) La ejecución del método **tal y como esté definido en la subclase.**

b) La búsqueda del método **ascendiendo por las superclases** hasta que se encuentra y puede ser ejecutado. Es lo que ocurrirá por ejemplo con toString() si no está definido en la subclase.

EJERCICIO

Declara una clase abstracta Legislador que herede de la clase Persona, con un atributo provinciaQueRepresenta (tipo String) y otros atributos. Declara un método abstracto getCamaraEnQueTrabaja. Crea dos clases concretas que hereden de Legislador: la clase Diputado y la clase Senador que sobreescriban los métodos abstractos necesarios. Crea una lista de legisladores y muestra por pantalla la cámara en que trabajan haciendo uso del polimorfismo.

Para comprobar si tu código es correcto puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU00696B

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