APRENDERAPROGRAMAR.COM

# CONCEPTO DE POLIMORFISMO Y VARIABLES POLIMÓRFICAS EN JAVA. JERARQUÍA DE TIPOS. EJEMPLOS. (CU00688B)

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

Fecha revisión: 2029

**Resumen:** Entrega nº88 curso Aprender programación Java desde cero.

Autor: Alex Rodríguez

**TIPOS Y SUBTIPOS. POLIMORFISMO Y VARIABLES POLIMÓRFICAS**

Al igual que se forma una jerarquía de clases, el hecho de que las clases definan tipos hace que la herencia dé lugar a una jerarquía de tipos. El tipo que se define mediante una subclase se dice que es **un subtipo del tipo definido en su superclase.**


Los supertipos pueden usarse para definir operaciones que admitan objetos de distintos subtipos. Por ejemplo, podemos crear una colección que admita objetos de distintos subtipos:

ArrayList <Profesor> ListadoProfesores = new ArrayList <Profesor>(); supone que admitimos en el listado de profesores a todos los subtipos de Profesor, por tanto podemos incluir en el listado tanto a profesores interinos como a profesores titulares. No necesitamos dos listados diferentes para cada tipo de profesor. Podemos tener un listado único si así lo deseamos. Esto nos permite unificar las operaciones con listados de profesores en una sola clase sin tener que duplicar código para profesores titulares y para profesores interinos. Por ejemplo mostrar los componentes de la lista, añadir componentes a la lista, etc. van a ser operaciones comunes tanto para profesores titulares como interinos.

Una variable que apunta a un objeto de un supertipo puede contener objetos de ese supertipo (si es que es coherente que existan) o de cualquier subtipo en escalas dependientes dentro de la jerarquía de tipos. Así resultarían válidas declaraciones como estas (suponiendo que se admiten constructores sin parámetros):

*Persona p1 = new Persona();*

*Persona p1 = new Profesor();*

*Persona p3 = new ProfesorInterino();*

***Al uso de variables de subtipos en lugares donde se espera (o se admite) un objeto de un supertipo se le denomina "sustitución". Los lenguajes orientados a objetos trabajan con el principio de sustitución: los tipos hijos pueden sustituir a los tipos padres. Sin embargo, la operación contraria no es posible: un tipo padre no puede ocupar el lugar de un tipo hijo.***

Esto sería erróneo: *ProfesorInterino p1 = new Persona();*. La persona puede ser un profesor interino o no: existe la incertidumbre de que lo sea o no lo sea. Java no puede saber si la persona es profesor interino o no, por lo que diremos que esta asignación no es válida en Java. También sería erróneo declarar *ProfesorInterino p1 = new ProfesorTitular();*. Obviamente esto no tiene sentido.

Para el API de Java, también es aplicable lo indicado: *List <String> miListado = new ArrayList <String> ();* es válido porque ArrayList es un subtipo de List. En ocasiones declararemos un tipo List sin tener predefinido si lo vamos a implementar en un subtipo ArrayList o en otro subtipo como LinkedList. A su vez, un método que espere un objeto de tipo List admitirá recibir tanto un ArrayList como un LinkedList.

Java decimos que se basa en el **polimorfismo**, entre otras razones porque una variable que apunta a un objeto admite distintas formas de ese objeto: las formas definidas por la superclase y las clases que extienden a la superclase. Es decir, una variable que apunta a un objeto es polimórfica porque admite distintos tipos de objetos, no solo uno. Una variable de tipo Persona admite tanto objetos de tipo Persona, como objetos de tipo Profesor, como objetos de tipo ProfesorInterino. O visto de otra manera, podríamos decir que un ProfesorInterino es una forma de Profesor, pero también una forma de Persona, y también una forma de Object. Las variables pueden ser polimórficas. El polimorfismo también se concreta de otras maneras en Java, como veremos más adelante.

La posibilidad de uso de variables polimórficas reduce la cantidad de código que es necesario escribir y facilita la reusabilidad. Por ejemplo no necesitamos dos bucles:

```
for (ProfesorInterino tmp : listado) {    …    }
```

```
for (ProfesorTitular tmp : listado) {   …   }
```

Sino solo uno:

```
for (Profesor tmp: listado) {   …   }
```

La variable listado, que apuntará a un objeto tipo ArrayList o similar, puede contener profesores interinos o profesores titulares, pero gracias al polimorfismo el bucle nos funciona en ambos casos, porque admite tanto profesores como cualquiera de los subtipos de profesores.

**EJERCICIO**

Amplía el código del programa Java que planteamos como ejercicio en la entrega CU00687 de este curso, relativo a la gestión de una empresa agroalimentaria, teniendo en cuenta que la empresa gestiona envíos a través de diferentes medios, y un envío puede contener cierto número de productos frescos, refrigerados o congelados. Añade al código:

a) Una clase EnvioDeProductos que represente un envío de productos como colección de objetos que admite el polimorfismo.

b) Crear una clase testHerencia4 con el método main donde se creen: dos productos frescos, tres productos refrigerados y cinco productos congelados (2 de ellos congelados por agua, otros 2 por agua y 1 por nitrógeno). Crear un envío que represente la agrupación de los anteriores productos. Mostrar por pantalla la información del número de productos que componen el envío y recorrer los productos del envío usando iterator para mostrar la información (valor de los atributos) de cada uno de ellos.

Puedes comprobar si tu respuesta es correcta consultando en los foros aprenderaprogramar.com.

**Próxima entrega:** CU00689B

**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