APRENDERAPROGRAMAR.COM

SOBREESCRIBIR MÉTODOS EN JAVA. TIPO ESTÁTICO Y DINÁMICO. LIGADURA. MÉTODOS POLIMÓRFICOS. (CU00690B)

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

Fecha revisión: 2029

Resumen: Entrega nº90 curso Aprender programación Java desde cero.

Autor: Alex Rodríguez

SOBREESCRIBIR MÉTODOS EN JAVA. MÉTODOS POLIMÓRFICOS.

En apartados anteriores del curso hemos visto conceptos como herencia en Java, jerarquías de herencia entre clases, polimorfismo y variables polimórficas. Vamos a centrarnos ahora en las variables. Ya sabemos que en Java las variables que representan un objeto no son el objeto mismo, sino referencias al objeto.

Cuando hablamos de una variable que apunta a un objeto podemos distinguir, debido a la herencia, dos tipos:

a) El tipo declarado para la variable en el código fuente: se llama **tipo estático**.

b) El tipo del objeto al que apunta la variable en un momento dado (en tiempo de ejecución): puede ser el tipo declarado o un subtipo del tipo declarado, y se llama **tipo dinámico**.

Ejemplo: *ProfesorInterino profesorInterino43 = new ProfesorInterino();*
Persona p1 = new Persona(); *p1 = profesorInterino43;*

El tipo estático de p1 es Persona, mientras que tras la ejecución de la tercera instrucción su tipo dinámico pasa a ser ProfesorInterino porque el objeto al que pasa a apuntar es a un profesor interino. Digamos que inicialmente el contenedor de p1 es una caja de tipo “Persona” y que en la tercera línea la caja pasa a ser de tipo “ProfesorInterino”.

Ejemplo: *Persona p1 = new ProfesorInterino();*

El tipo estático es Persona y el tipo dinámico es ProfesorInterino. ¿Por qué? Porque estamos creando una variable de tipo Persona e inmediatamente asignándole de forma dinámica un objeto anónimo de tipo ProfesorInterino. Sin embargo, no podemos invocar un método exclusivo de ProfesorInterino sobre p1 porque el compilador se basa en los tipos estáticos para hacer comprobaciones.

El compilador controla los tipos basándose en el tipo estático (declarado): no conoce si en un momento dado una variable está apuntando a un subtipo. Debido a esto, no podremos llamar a un método de un subtipo si la variable corresponde a una superclase, aunque la variable apunte a la subclase, porque el compilador no llega a conocer a qué tipo apunta la variable en tiempo de ejecución. Como veremos a continuación, esto es una limitación relativa, porque en los métodos que

denominamos “sobreescritos” sí se llega a identificar si un método corresponde a un supertipo o a un subtipo.

Decimos que un método está sobreescrito cuando está presente, exactamente con la misma signatura, en una subclase y en una superclase. En esta situación un objeto de la subclase tiene dos métodos con el mismo nombre y la misma signatura: uno heredado de la superclase y otro definido en su propia estructura. Una subclase puede sobreescribir la implementación de un método. Para hacerlo, la subclase declara un método con la misma signatura que la superclase, pero con un cuerpo diferente (hacerlo con el mismo cuerpo sería repetitivo y no tendría mucho sentido). El método sobreescrito tiene precedencia cuando se invoca sobre objetos de la subclase. Por el contrario, para los objetos de la subclase el método de la superclase pierde visibilidad.

Si como hemos dicho, el compilador se basa en el tipo estático para su trabajo, podríamos pensar que si invocamos a un objeto de tipo estático “superclase” y tipo dinámico “subclase” con un método sobreescrito, el método que se utilice sería el propio de la superclase. Pero sin embargo, esto no es así: el control de tipos del compilador se basa en los tipos estáticos pero en tiempo de ejecución los métodos que se ejecutan dan preferencia al tipo dinámico. Es decir, **en tiempo de ejecución Java está constantemente “buscando” (“ligando” o “despachando”) el método que corresponda en función del tipo dinámico** al que apunta una variable. Si el método invocado no se encuentra definido en el tipo dinámico de que se trate, Java busca el método en la superclase para tratar de ejecutarlo. Si no lo encuentra en la superclase, continúa subiendo niveles de clase hasta alcanzar a la clase Object. Si en la clase Object tampoco se encontrara un método con ese nombre, el programa no llegaría a compilar. Decimos que los métodos en Java son polimórficos porque una misma llamada en un programa puede dar lugar a la ejecución de distintos métodos según el tipo dinámico de una variable. Ya tenemos una segunda forma de expresión del polimorfismo en Java: además de polimorfismo en variables, hablamos de polimorfismo en métodos.

> ***Tenemos que distinguir dos momentos temporales en un programa, la compilación durante la cual el compilador realiza una serie de verificaciones y transforma el código fuente en código máquina, y la ejecución, durante la cual una variable puede cambiar de tipo debido al polimorfismo que admite Java. El compilador solo conoce un tipo: el tipo estático o declarado en el código fuente. No obstante, durante la ejecución la máquina virtual Java es capaz de identificar el tipo dinámico de las variables que apuntan a objetos y asignar un método u otro en función de ese tipo dinámico.***

Esto tiene ciertos riesgos. Si pensamos que hemos sobreescrito un método para que haga una tarea específica por nosotros requerida, pero no es así o lo hemos borrado, aparentemente el programa no tendrá problemas de compilación si se encuentra el método en una superclase. Sin embargo, los resultados pueden no ser los deseados y esto generarnos un quebradero de cabeza al no saber determinar con precisión por qué ocurre lo que ocurre. Debido a ello, muchos programadores tienen por costumbre dentro de los primeros pasos al definir una clase, sobreescribir algunos métodos como *toString* y *equals* para esos métodos respondan en la clase exactamente como se pretende y no de acuerdo a una especificación de una superclase que puede no ser adecuada.

EJERCICIO

Responde a las siguientes preguntas:

Supongamos la superclase Vehiculo que representa a cualquier tipo de vehículo y la subclase Taxi que representa a un tipo de vehículo concreto.

a) ¿Un objeto de tipo estático declarado Taxi puede contener a un objeto Vehiculo en tiempo de ejecución?

b) ¿Un objeto de tipo estático declarado Vehiculo puede contener a un objeto Taxi en tiempo de ejecución?

c) Escribe el código de una clase Vehiculo con los atributos matricula (String) y numeroDeRuedas (int), constructor, métodos getters y setters y método toString() para mostrar la información de un vehículo.

d) Escribe el código de una clase Taxi que herede de vehículo y que además de los atributos de Vehiculo lleve un atributo adicional nombreDelConductor (String) y numeroDePlazas (int), constructor, métodos getters y setters y método toString() para mostrar la información de un Taxi.

e) Escribe el código de una clase test con el método main que cree un objeto cuyo tipo es Vehiculo, instanciado como Taxi. Establece valores para sus atributos y usa el método toString(). ¿Qué método toString() resulta de aplicación, el propio de la clase Vehiculo o el propio de la clase Taxi? ¿Por qué?

Puedes comprobar si tus respuestas son correctas consultando en los foros aprenderaprogramar.com.

Próxima entrega: CU00691B

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