CAPITULO 9. ALGO MAS SOBRE HERENCIA

9.2 Tipo estático y tipo dinámico

Denominamos ***tipo estático*** al tipo declarado de una variable en el código fuente, la representación estática del programa.

Denominamos ***tipo dinámico*** al tipo del objeto almacenado en una variable porque depende de su asignación en tiempo de ejecución, el comportamiento dinámico del programa.

Si con una variable invocamos un método que el tipo dinámico tiene, pero el estático no, el compilador informa de un error porque cuando controla los tipos usa el tipo estático. El tipo dinámico se conoce, frecuentemente, sólo en tiempo de ejecución por lo que el compilador no tiene otra opción más que usar el tipo estático cuando quiere hacer alguna verificación de tipos en tiempo de compilación.

9.3 Sobrescribir

Sobreescritura (redefinición). Una subclase puede sobrescribir la implementación de un método. Para hacerlo la subclase declara un método con la misma signatura que la superclase pero con un cuerpo diferente. El método que sobrescribe tiene precedencia cuando se invoca sobre objetos de la subclase.

9.4 Búsqueda dinámica del método

El control de tipos que realiza el compilador es sobre el tipo estático, pero en tiempo de ejecución los métodos que se ejecutan son los que corresponden al tipo dinámico.

Veamos con más detalle cómo se invocan los métodos. Este procedimiento se conoce como *búsqueda de método, ligadura de método* o *despacho de método*.

Suponga que tenemos un objeto de clase DVD almacenado en una variable v1 declarada de tipo DVD. La clase DVD tiene un método imprimir y no tiene declarada ninguna superclase. Esta es una situación muy simple que no involucra herencia ni polimorfismo. Luego, ejecutamos la sentencia v1.imprimir ();

l. Se accede a la variable v1.
2. Se encuentra el objeto almacenado en esa variable (siguiendo la referencia).
3. Se encuentra la clase del objeto (siguiendo la referencia «es instancia de»).
4. Se encuentra la implementación del método imprimir en la clase y se ejecuta.

A continuación, vemos la búsqueda de un método cuando hay herencia. El escenario es similar al anterior, pero esta vez la clase DVD tiene una superclase, Elemento, y el método imprimir está definido sólo en la superclase. Ejecutamos la misma sentencia. La invocación al método comienza de manera similar: se ejecutan nuevamente los pasos l al 3 del escenario anterior pero luego continúa de manera diferente:

4. No se encuentra ningún método imprimir en la clase DVD.
5. Dado que no se encontró ningún método que coincida, se busca en la superclase un método que coincida. Si no se encuentra ningún método en la superclase, se busca en la siguiente superclase (si es que existe). Esta búsqueda continúa hacia arriba por toda la jerarquía de herencia de la clase hasta Object o hasta que se encuentre definitivamente un método. Tenga en cuenta que, en tiempo de ejecución, debe encontrarse definitivamente un método que coincida, de lo contrario la clase no habría compilado.
6. En nuestro ejemplo, el método imprimir es encontrado en la clase Elemento y es el que será ejecutado.

Este escenario ilustra la manera en que los objetos heredan los métodos. Cualquier método que se encuentre en la superclase puede ser invocado sobre un objeto de la subclase y será correctamente encontrado y ejecutado.

Ahora llegamos al escenario más interesante: la búsqueda de métodos con una variable polimórfica y un método sobrescrito. El escenario nuevamente es similar al anterior pero existen dos cambios:

- El tipo declarado de la variable v1 ahora es Elemento, no DVD.
- El método imprimir está definido en la clase Elemento y redefinido (o sobrescrito) en la clase DVD.

Los pasos que se siguen para la ejecución del método son exactamente los mismos pasos 1 al 4 del primer escenario.

Es importante hacer algunas observaciones:

- No se usa ninguna regla especial para la búsqueda del método en los casos en los que el tipo dinámico no sea igual al tipo estático. El comportamiento que observamos es un resultado de las reglas generales.

• El método que se encuentra primero y que se ejecuta está determinado por el tipo dinámico, no por el tipo estático. En otras palabras, el hecho de que el tipo declarado de la variable v1 ahora es Elemento no tiene ningún efecto. La instancia con la que estamos trabajando es de la clase DVD, y esto es todo lo que cuenta.
• Los métodos sobrescritos en las subclases tienen precedencia sobre los métodos de las superclases. Dado que la búsqueda de método comienza en la clase dinámica de la instancia (al final de la jerarquía de herencia) la última redefinición de un método es la que se encuentra primero y la que se ejecuta.
• Cuando un método está sobrescrito, sólo se ejecuta la última versión (la más baja en la jerarquía de herencia). Las versiones del mismo método en cualquier superclase no se ejecutan automáticamente.

9.5 Llamada a super en métodos

Para llamar al método sobrescrito de la superclase desde la subclase escribiremos:

```
public void <NombreMetodo>(){
        super.<NombreMetodo>();
…}
```

Al contrario que las llamadas a super en los constructores, el nombre del método de la superclase está explícitamente establecido. Una llamada a super en un método siempre tiene la forma super.<nombre-del-método>(<parámetros>).

Nuevamente, y en contra de la regla de las llamadas a super en los constructores, la llamada a super en los métodos puede ocurrir en cualquier lugar dentro de dicho método. No tiene por qué ocurrir en la primera sentencia.

Al contrario que en las llamadas a super en los constructores, no se genera automáticamente ninguna llamada a super y tampoco se requiere ninguna llamada a super, es completamente opcional. De modo que el comportamiento por defecto presenta el efecto de un método de una subclase ocultando completamente (sobrescribiendo) la versión de la superclase del mismo método.

9.6 Método polimórfico

Método polimórfico. Las llamadas a métodos en Java son polimórficas. El mismo método puede invocar en diferentes momentos diferentes métodos dependiendo del tipo dinámico de la variable usada para hacer la invocación.

9.7 Métodos de Object

La superclase universal Object implementa algunos métodos que, por tanto, forman parte de todos los objetos. El más interesante de estos métodos es toString.

El propósito del método toString es crear una cadena de representación de un objeto. Esto es útil para cualquier objeto que pueda ser representado textualmente en la interfaz de usuario pero también es de ayuda para todos los otros objetos; por ejemplo, los objetos pueden ser fácilmente impresos con fines de depuración de un programa. En Object el valor de retorno de toString muestra simplemente el nombre de la clase del objeto y un número que representa la dirección de memoria donde el objeto está almacenado

Cada objeto en Java tiene un método **toString** que puede usarse para devolver un String de su representación. Típicamente, para que resulte útil un objeto debe sobrescribir este método.

Los métodos System.out.print y System.out.println son especiales con respecto a esto: si el parámetro de uno de estos métodos no es un objeto String, el método invoca automáticamente al método toString de dicho objeto.

9.8 Acceso protegido

Los lenguajes orientados a objetos frecuentemente definen un nivel de acceso que está entre medias de la restricción completa del acceso privado y la total disponibilidad del acceso público. En Java este nivel de acceso se denomina ***acceso protegido***.

La declaración de un campo o un método como **protegido** (protected) permite su acceso directo desde las subclases (directas o indirectas).

El acceso protegido permite acceder a los campos o a los métodos dentro de una misma clase y desde todas las subclases, pero no desde otras clases.

Mientras que el acceso protegido puede aplicarse a cualquier miembro de una clase, generalmente se reserva para los métodos y los constructores; no es frecuente aplicarlo en los campos porque debilitaría el encapsulamiento. Siempre que sea

posible, los campos modificables de las superclases deberían permanecer privados. Sin embargo, existen casos válidos ocasionales en los que es deseable el acceso directo desde una subclase. La herencia representa una forma mucho más cerrada de acoplamiento que una relación normal de cliente.

La herencia vincula las clases de manera muy cercana y la modificación de la superclase puede romper fácilmente la subclase. Este punto debiera tenerse en consideración cuando se diseñan las clases y sus relaciones.