7.F. Polimorfismo.

1. Polimorfismo.

1.1. Concepto de polimorfismo.

El **polimorfismo** consiste en la capacidad de poder utilizar una referencia a un objeto de una determinada clase como si fuera de otra clase (en concreto una **subclase**). Es una manera de decir que una clase podría tener varias (poli) formas (morfismo).

Un método "**polimórfico**" ofrece la posibilidad de ser distinguido (saber a qué clase pertenece) en **tiempo de ejecución** en lugar de en **tiempo de compilación**. Para poder hacer algo así es necesario utilizar métodos que pertenecen a una **superclase** y que en cada **subclase** se implementan de una forma en particular. En **tiempo de compilación** se invocará al método sin saber exactamente si será el de una subclase u otra (pues se está invocando al de la **superclase**). Sólo en **tiempo de ejecución** (una vez instanciada una u otra **subclase**) se conocerá realmente qué método (de qué **subclase**) es el que finalmente va a ser invocado.

Esta forma de trabajar te va a permitir hasta cierto punto "desentenderte" del tipo de objeto **específico** (**subclase**) para centrarte en el tipo de objeto **genérico** (**superclase**). De este modo podrás manipular objetos hasta cierto punto "desconocidos" en tiempo de compilación y que sólo durante la ejecución del programa se sabrá exactamente de qué tipo de objeto (**subclase**) se trata.

El **polimorfismo** ofrece la **posibilidad de que toda referencia a un objeto de una superclase pueda tomar la forma de una referencia a un objeto de una de sus subclases**. Esto te va a permitir escribir programas que procesen objetos de clases que formen parte de la misma jerarquía como si todos fueran objetos de sus **superclases**.

El **polimorfismo** puede llevarse a cabo tanto con **superclases** (abstractas o no) como con **interfaces**.

Dada una **superclase X**, con un método **m**, y dos **subclases A** y **B**, que redefinen ese método **m**, podrías declarar un objeto **O** de tipo **X** que en durante la **ejecución** podrá ser de tipo **A** o de tipo **B** (algo desconocido en **tiempo de compilación**). Esto significa que al invocarse el método **m** de **X** (**superclase**), se estará en realidad invocando al método **m** de **A** o de **B** (alguna de sus **subclases**). Por ejemplo:

```
// Declaración de una referencia a un objeto de tipo X

ClaseX obj;  // Objeto de tipo X (superclase)

...

// Zona del programa donde se instancia un objeto de tipo A (subclase) y se le asigna a la referencia obj.

// La variable obj adquiere la forma de la subclase A.
obj = new ClaseA ();

...

// Otra zona del programa.

// Aquí se instancia un objeto de tipo B (subclase) y se le asigna a la referencia obj.

// La variable obj adquiere la forma de la subclase B.
obj = new ClaseB ();

...

// Zona donde se utiliza el método m sin saber realmente qué subclase se está utilizando.

// (Sólo se sabrá durante la ejecución del programa)
obj.m ()  // Llamada al método m (sin saber si será el método m de A o de B).
```

...

Imagina que estás trabajando con las clases `Alumno` y `Profesor` y que en determinada zona del código podrías tener objetos, tanto de un tipo como de otro, pero eso sólo se sabrá según vaya discurriendo la ejecución del programa. En algunos casos, es posible que un determinado objeto pudiera ser de la clase `Alumno` y en otros de la clase `Profesor`, pero en cualquier caso serán objetos de la clase `Persona`. Eso significa que la llamada a un método de la clase `Persona` (por ejemplo `devolverContenidoString`) en realidad será en unos casos a un método (con el mismo nombre) de la clase `Alumno` y, en otros, a un método (con el mismo nombre también) de la clase `Profesor`. Esto será posible hacerlo gracias a la **ligadura dinámica**.

Autoevaluación

El polimorfismo ofrece la posibilidad de que toda referencia a un objeto de una clase A pueda tomar la forma de una referencia a un objeto de cualquier otra clase B. ¿Verdadero o Falso?

- ○ Verdadero
- ○ Falso