UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA UNIVERSITARIA DE INFORMÁTICA
LENGUAJES, PROYECTOS Y SISTEMAS INFORMÁTICOS
Luis Fernández Muñoz
setillo@eui.upm.es

Programción Orientada a Objetos en Java

Tema 6. POLIMORFISMO

ÍNDICE

8. POLIMORFISMO

DEFINICIÓN: término de origen griego que significa “muchas formas”.
Ej.: Una persona puede pagar con tarjeta o con efectivo
Ej.: Una empresa de transporte realiza ventas de billetes por ventanilla o a través de una máquina
Ej.: Un sistema operativo imprime a través de drivers de impresora para cada modelo
Ej.: Un navegador muestra textos, imágenes, videos, …, con muy diversos formatos

Limitaciones en Programación Orientada a Objetos:
- no se contempla que algo cambie de forma.
- no se contempla que algo sea dos cosas a la vez.

Ej.: Una persona de ventanilla NO se convierte en máquina expendedora
Ej.: Una persona NO es a la vez una máquina expendedora.
Ej.: Simplemente, un billete lo puede vender una persona o una máquina expendedora en cada momento que se vende un billete

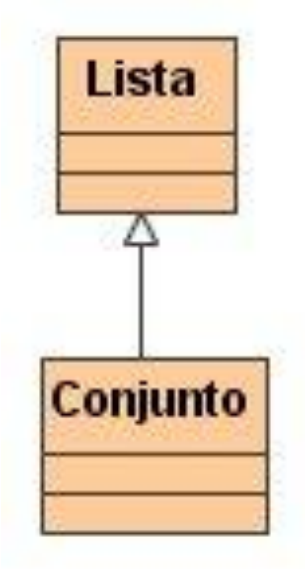
8. POLIMORFISMO

DEFINICIÓN: el polimorfismo es una relajación del sistema de tipos, de tal manera que una referencia a una clase (atributo, parámetro o declaración local o elemento de un vector) acepta direcciones de objetos de dicha clase y de sus clases derivadas (hijas, nietas, …).

Por tanto:

- el polimorfismo exige la existencia de una jerarquía de clasificación;
- las jerarquías de clasificación NO exigen tratamientos polimórficos;

Ej.: En un punto dado, existen listas, en otro punto, existen conjuntos y, en otro punto, pueden existir indiferentemente listas o conjuntos



8. POLIMORFISMO

```
Ej.:   Lista lista = new Lista();
       Conjunto conjuntoMal  = new Lista(); // ERROR 1
       Conjunto conjunto  = new Conjunto();
       Lista coleccion;
       …
       coleccion = new Lista(); // POLIMORFISMO
       …
       coleccion = new Conjunto(); // POLIMORFISMO
       …
```

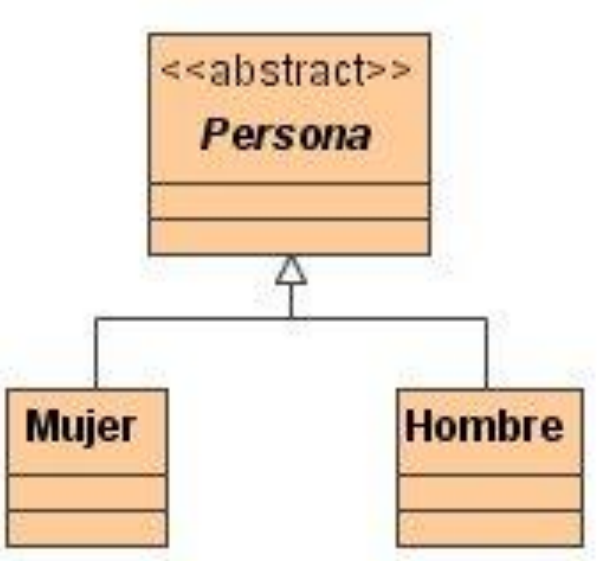
ERROR 1: Incompatibilidad de tipos

- *donde un objeto se crea de una clase y siempre será de esa clase mientras que la referencia coleccion puede tener, en un momento dado, una dirección de un objeto lista o de un objeto conjunto*

8. POLIMORFISMO

• con la incorporación del polimorfismo, tiene sentido declarar referencias a clases abstractas con la intención de almacenar direcciones de objetos de clases concretas derivadas, NO de clases abstractas que no son instanciables;

Ej.: A lo largo de la historia, en este mundo existe la jerarquía de personas: mujeres y hombres. Pero en ciertos momento y en ciertos países, no existe polimorfismo: el lugar de una mujer no lo puede ocupar un hombre y el de un hombre no lo puede ocupar una mujer. Mientras que, en otros momentos u otros países, el lugar de una persona es indiferentemente ocupado por una mujer o un hombre.



8. POLIMORFISMO

```
Ej.:   Mujer mujer = new Mujer();
       Mujer mujerMal = new Hombre(); // ERROR 1
       Mujer mujerPeor = new Persona(); // ERROR 2
       Hombre hombre = new Hombre();
       Hombre hombreMal = new Mujer(); // ERROR 1
       Hombre hombrePeor = new Persona(); // ERROR 2
       Persona personaBien = new Mujer(); // POLIMORFISMO
       Persona personaGuay = new Hombre(); // POLIMORFISMO
       Persona personaMal = new Persona(); // ERROR 2
```

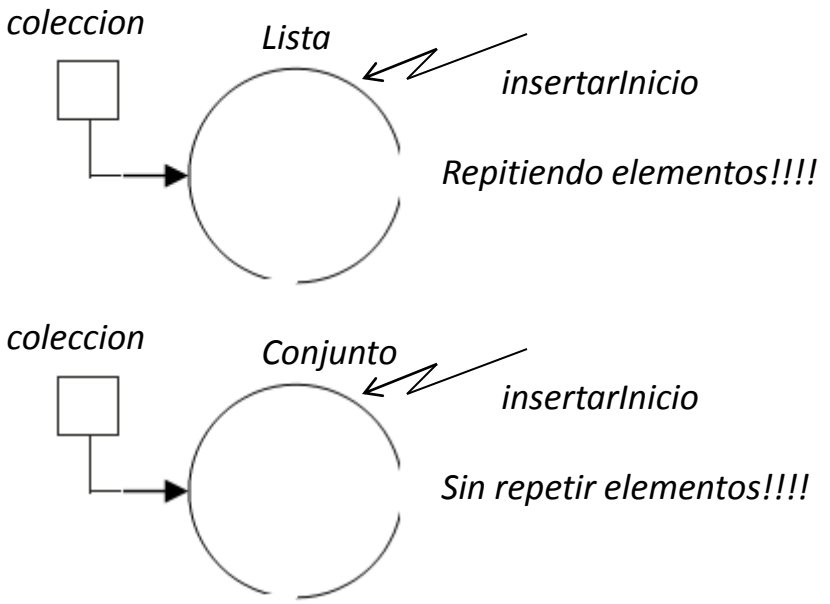
ERROR 1: Incompatibilidad de tipos
ERROR 2: Instanciación de una clase abstracta

- *donde no se contempla que una mujer se convierta en hombre o un hombre en mujer; ni se contempla que alguien sea mujer y hombre a la vez; lo único que se contempla es que una referencia a persona apunte a un objeto de la clase mujer u hombre*

8. POLIMORFISMO

Comportamiento: cuando se lanza un mensaje a un objeto a través de una referencia polimórfica se ejecuta el método prescrito en la clase del objeto que recibe el mensaje.

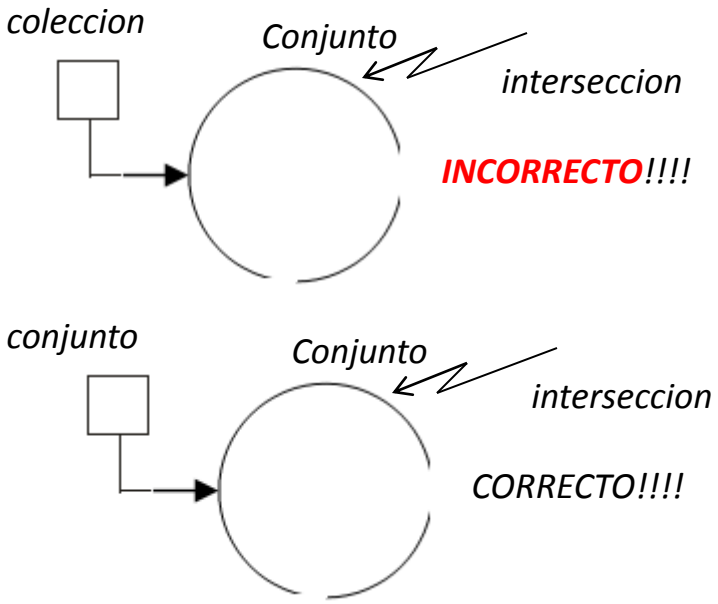
Ej.: En un punto dado, existen listas, en otro punto, existen conjuntos y, en otro punto, pueden existir indiferentemente listas o conjuntos



8. POLIMORFISMO

Limitación: cuando se lanza un mensaje a un objeto a través de una referencia polimórfica , éste debe estar contemplado en el interfaz de la clase de la que se declaró la referencia sin contemplar los posibles métodos añadidos en la clase del objeto apuntado.

Ej.: En un punto dado, existen listas, en otro punto, existen conjuntos y, en otro punto, pueden existir indiferentemente listas o conjuntos



8. POLIMORFISMO

FORMALIZACIÓN:

Enlace*:* es la asociación entre un elemento de un lenguaje de programación y una de sus características;
Ej.: una variable y su nombre, su valor, su dirección, su tipo, …; una constante y su nombre, su valor, su dirección, su tipo, …; una expresión y su número de operadores, su valor evaluado, su tipo, …

Tipos de Enlaces:

• *Enlace estático:* aquel enlace que se puede resolver analizando el código, o sea, en tiempo de compilación;
Ej.: una variable y su nombre, su tipo, …; una constante y su nombre, su valor, su tipo, …; una expresión y su número de operadores, su tipo, …

• *Enlace dinámico:* aquel enlace que NO se puede resolver analizando el código sino que se resuelve en tiempo de ejecución;
Ej.: una variable y su valor, …; una expresión y su valor evaluado

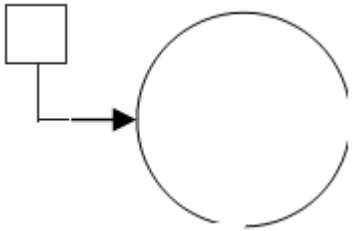
8. POLIMORFISMO

FORMALIZACIÓN:

Polimorfismo: es un enlace dinámico entre una referencia y la clase de objeto apuntado por la referencia;

```
Ej.:   Lista coleccion;
       …
       … coleccion …
```

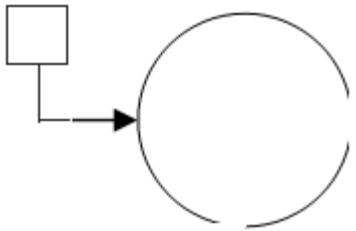
coleccion



¿de la clase Lista o Conjunto?

```
Ej.:   Persona persona;
       …
       … persona …
```

persona



¿de la clase Mujer u Hombre?

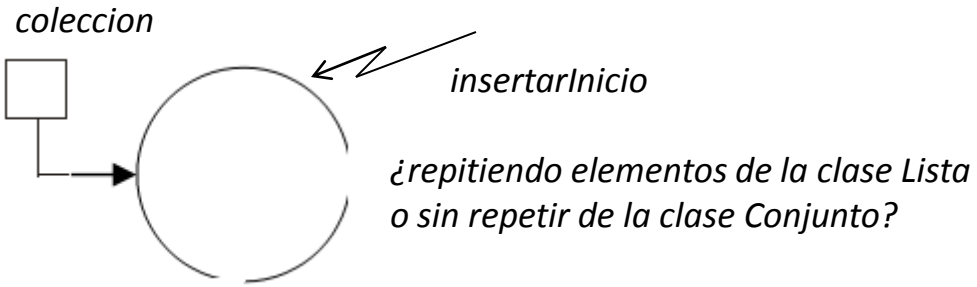
8. POLIMORFISMO

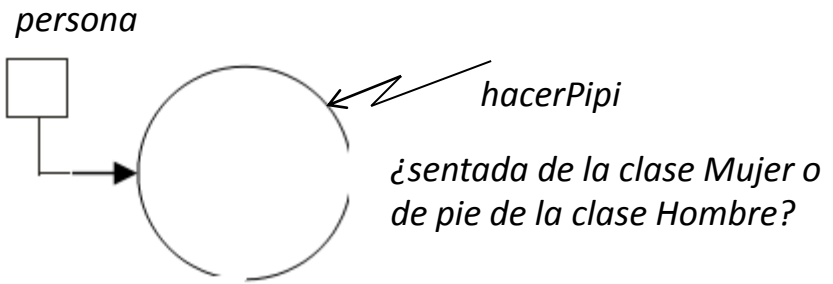
FORMALIZACIÓN:

Polimorfismo: consecuentemente a la formalización anterior, es un enlace dinámico entre un mensaje y el método que se ejecuta;

```
Ej.:   Lista coleccion;
       …
       … coleccion.insertarInicio …
```

```
Ej.:   Persona persona;
       …
       … persona.hacerPipi …
```





9. POLIMORFISMO VS SOBRECARGA

FORMALIZACIÓN:

Sobrecarga: es un enlace estático entre entre un mensaje y el método que se ejecuta;

```
Ej.: class A {
        public void m()
        public void m(A a)
        public void m(B b)
        public A m(B b, C c)
        public B m(A a, C c)
     }

     class B {
     }
```

```
Ej.: class C {
}
…
A a;
B b;
C c;
a.m(b, c).m();
b = a.m(a, c);
…
```

10. CONVERSIÓN DE TIPOS

Conversión de Direcciones a Objetos:

• *Conversión ascendente (upcast):* cuando se transforma una dirección de un objeto de una clase a una dirección del objeto pero de una clase ascedente (padre, abuela, …) *;*

• *Conversión descendente (downcast):* cuando se transforma una dirección de un objeto de una clase a una dirección del objeto pero de una clase derivada (hija, nieta, …) *;*

Cualquier otra conversión produce error en tiempo de ejecución (conversión de una dirección de un objeto de una clase a otra clase fuera de la rama ascendente o del árbol descendente: tíos, hermanos, clases ajenas a la jerarquía, …)

10. CONVERSIÓN DE TIPOS

Conversión de tipos:

- *Conversión implícita*: por conversión ascendente cuando se asigna una dirección de un objeto de una clase a una referencia declarada a una clase ascendente;
- *Conversión explícita*: por conversión descendente a través del operador de conversión de tipos (*cast*);

```
(<tipo>) <direccion>
```

```
Lista lista = new Lista();
Conjunto conjunto = new Conjunto()
Lista coleccion = new Conjunto();
Conjunto resultado;
resultado = conjunto.interseccion(coleccion); // ERROR
resultado = conjunto.interseccion((Conjunto) coleccion);
resultado = coleccion.interseccion(conjunto); // ERROR
resultado = ((Conjunto) coleccion).interseccion(conjunto);
((Conjunto) lista).interseccion(conjunto); // ERROR DE
                                           // EJECUCION
```

11. BENEFICIOS DEL POLIMORFISMO

ABSTRACCIÓN: “Entonces es el receptor del mensaje el que determina cómo se interpretará el mensaje y no lo hará el emisor. El emisor sólo necesita conocer qué comportamiento puede desarrollar el otro objeto, no qué clase de objeto cree que es y, por tanto, qué método realiza en cada instante el comportamiento. Esto es una herramienta extremadamente importante para permitirnos desarrollar sistemas flexibles. De esta manera, sólo tenemos especificado qué ocurre pero no cómo ocurrirá. Mediante esta forma de delegar qué ocurrirá, se obtiene un sistema flexible y resistente a las modificaciones” *[Jacobson (creador de los Casos de Uso), 1992]*

!!! No se puede preguntar por la clase de un objeto polimórfico!!!

11. BENEFICIOS DEL POLIMORFISMO

EXTENSIBILIDAD: "Emplear las consultas de tipo durante la ejecución para implantar un enunciado de conmutación – estructura de control de flujo CASE o IF-THEN-ELSE encadenados – en un campo de tipo destruye toda la modularidad de un programa y anula los objetivos de la programación orientada a objetos. También es propensa a errores; […] La experiencia demuestra que los programadores que se formaron con lenguajes como Pascal o C encuentran esta trampa muy difícil de resistir. Una razón es que este estilo requiere menos premeditación […]; en este contexto, semejante falta de premeditación muchas veces no es más que una chapuza."
[Stroustrup (creador de C++), 1993]

!!! No se puede preguntar por la clase de un objeto polimórfico!!!