

POLIMORFISMO

Repaso del ejemplo de herencia:

Un consultor es un trabajador, esa relación es un... es la herencia. En realidad, podemos utilizar referencias de una superclase para referenciar (almacenar) objetos de una subclase. Por ejemplo, podemos utilizar referencias de trabajador para hacer referencia a empleado o consultores además de para hacer referencias de trabajadores.

Por ejemplo, podemos tener un método que trabaje con la clase base y con las heredadas, un método que salude a cualquier trabajador podemos recibir un trabajador como argumento y saludarlo (ya que todos tienen en método getnombre()). O para fichar la entrada de la hora de un trabajador sea del tipo que sea.

REFERENCIAS Y SUBCLASES

- ▶ Una subclase puede ser accedida a través de una referencia de una superclase.
- ▶ Esto es muy útil, sobre todo, para usar como atributos de métodos.

```
public static void saludar(Trabajador t) {  
    System.out.println("Hola, " + t.getNombre());  
}
```

Se complica cuando tenemos métodos con los mismos nombres en distintas clases (padre e hijo), esto es útil como el cálculo de la paga, pero si tenemos referencias de trabajador y llamamos a calcular la paga, sería un problema, para solucionarlo se usa el polimorfismo.

OCULTACIÓN DE MÉTODOS

- ▶ Si una subclase añade un método con mismo nombre y firma que otro de la clase base, oculta a este.
- ▶ ¿Qué sucede en caso de el uso de referencias de clase base, pero instanciación de objetos derivados?

```
Trabajador empleado;  
empleado = new Empleado("Larry Ellison", "Presidente",  
                        "Redwood", "", "", 100000.0, 1000.0);  
empleado.calcularPaga();
```

Java escoge en tiempo de ejecución el método correcto.

POLIMORFISMO

- ▶ Java escoge, en tiempo de ejecución, el tipo de objeto.
- ▶ Si ese tipo ha ocultado un método de la superclase, llama a la *concreción*.
- ▶ En otro caso, llama al método de la clase base.

En el ejemplo, si añadimos el método siguiente a trabajador:

```
public double calcularPaga() {  
    return SALARIO_BASE;  
}
```

Ya las 3 clases tienen un método `calcularPaga`, en `Empleado` y `Consultor` al implementar un método con el mismo nombre, están ocultando el método de `Trabajador`. (se llama igual en las 3 clases)

```
public static void main(String[] args) {  
  
    Trabajador trabajador;  
    Trabajador empleado;  
    Trabajador consultor;  
  
    trabajador = new Trabajador("Bill Gates", "Presidente", "Redmond", "", "");  
    empleado = new Empleado("Larry Ellison", "Presidente", "Redwood", "", "", 100000.0, 1000.0);  
    consultor = new Consultor("Steve Jobs", "Consultor Jefe", "Cupertino", "", "", 20, 1000.0);  
  
    saludar(trabajador);  
    saludar(empleado);  
    saludar(consultor);  
}
```

Saludar, funciona si le paso cualquiera de los 3 objetos. En mi clase principal tengo el método `saludar` (como las funciones que hacíamos antes):

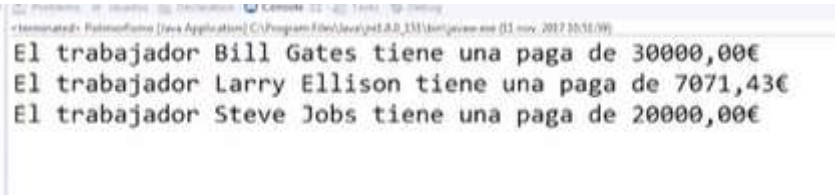
```
public static void saludar(Trabajador t) {  
    System.out.println("Hola, " + t.getNombre());  
}  
  
public static void imprimirNombreYPaga(Trabajador t) {  
    System.out.printf("El trabajador %s tiene una paga de %.2f€ %n", t.getNombre(), t.calcularPaga());  
}
```

```
Hola, Bill Gates  
Hola, Larry Ellison  
Hola, Steve Jobs
```

Si saludamos, todo correcto, pero y con `CalcularPaga`, si llamamos al método y pasamos como argumento los 3 objetos,

```
imprimirNombreYPaga(trabajador);  
imprimirNombreYPaga(empleador);  
imprimirNombreYPaga(consultor);
```

Vamos a verlo:



El trabajador Bill Gates tiene una paga de 30000,00€
El trabajador Larry Ellison tiene una paga de 7071,43€
El trabajador Steve Jobs tiene una paga de 20000,00€

En tiempo de ejecución Java ha detectado el método de cada uno lo ha solucionado de forma correcta, eso es el polimorfismo.