Programación Orientada a Objetos en Java

Curso 2006 - 2007

Tema 4 – Herencia y Polimorfismo

Gonzalo Méndez Pozo
Dpto. de Ingeniería de Software e Inteligencia Artificial
Universidad Complutense de Madrid

Herencia

- Tipo especial de relación entre clases
- Es uno de los aspectos que distinguen el Paradigma de Orientación a Objetos frente a otros paradigmas
- Mecanismo que, bien utilizado, facilita la modificabilidad y reutilización de los diseños y el código

Herencia

- ¿En qué consiste?
  - Existen dos clases, a las que llamaremos padre (superclase o clase base) e hija (subclase o clase derivada)
  - Al igual que las herencias en la vida real, la clase hija pasa a tener lo que tiene la clase padre
    - Atributos
    - Métodos
  - Un objeto de la clase hija es también un objeto de la clase padre

Herencia

- ¿En qué consiste?
  - En la clase hija se definen las diferencias respecto de la clase padre
- ¿Para qué se usa?
  - Para extender la funcionalidad de la clase padre
  - Para especializar el comportamiento de la clase padre

Herencia

- La herencia modifica el mecanismo de paso de mensajes
- Cuando a un objeto de una clase **C** se le pasa un mensaje **M**, se busca un método **M** en la clase **C**:
  - Si existe un método **M** en la clase **C**, se ejecuta ese método y termina el proceso
  - Si en la clase **C** no hay ningún método **M**, se busca éste en la superclase de **C**
  - Si en la superclase de **C** existe un método **M**, se ejecuta ese método y termina el proceso
  - Si en la superclase de **C** no hay ningún método **M**, se busca en las superclases de la superclase hasta que o bien se encuentra y se ejecuta o no se encuentra en ninguna de las superclases, de forma que el objeto no entiende ese mensaje **M**, dando como resultado el consiguiente error

Herencia

- Ventajas
  - Se ahorra código
  - Permite reutilizar código extendiendo su funcionalidad
- Desventajas
  - Se ahorra código
  - Se introduce una fuerte dependencia en la clase hija respecto a la clase padre
  - Puede dificultar la reutilización
  - Un cambio en la clase padre puede tener efectos imprevistos en las clases hijas
  - Un objeto de una clase hija puede tener un comportamiento inconsistente con lo esperado de un objeto de la clase padre
  - Se establece una jerarquía o clasificación. Si cambia el criterio de clasificación puede acarrear muchas modificaciones

Herencia en Java

- Se indica usando la palabra reservada extends
  class Punto3D extends Punto2D
- Visibilidad:
  - Los miembros privados de la superclase no son visibles desde la subclase
  - Los miembros públicos de la superclase son visibles y siguen siendo públicos en la subclase
- Se puede acceder a los miembros de la superclase usando la palabra reservada super
- Si una clase se declara como final no se puede heredar de ella
- En java, todas las clases heredan implícitamente de la clase Object.

Herencia en Java

```
class Punto2D
{
    private int x,y;

    Punto2D (int x, int y)
    {
        this.x=x; this.y=y;
    }

    public void pintar ()
    {
        …
    }
}
```

```
final class Punto3D extends Punto2D
{
    private int z;

    Punto3D (int x, int y, int z)
    {
        super (x,y); this.z=z;
    }

    public void pintar()
    {
        super.pintar();
        …
    }
}
```


Clases Abstractas

- En ciertos casos, una clase se diseña directamente para ser extendida por un conjunto de subclases
- En estos casos suele interesar no implementar alguno de sus métodos, pues no tiene ningún significado en la clase base.
- Es necesario declarar tanto la clase como los métodos no implementados como abstractos a través de la palabra reservada abstract
- Una clase abstracta es, por tanto, aquella que tiene alguno de sus métodos definido pero no implementado


Clases Abstractas

- Podemos crear una clase Animal a partir de la cual crearemos otras clases que definan animales concretos
- (Casi) todos los animales emiten algún sonido, pero no hay ninguno común para todos los animales
- Cada subclase reimplementará el método sound() como le convenga

```
public abstract class Animal
{
      private String nombre;
      public abstract void sound();

      public Animal (String nombre)
      {
            this.nombre=
                  new String(nombre);
      }
      …
}
```


Interfaces

- A veces nos interesa que todos los métodos de una clase abstracta sean abstractos
  - Lo usamos para obligar a que todas las subclases reimplementen esos métodos
- Para estos casos, Java proporciona unas clases especiales llamadas interfaces
- Se declaran como interface, no como class
- Representan el concepto de clase abstracta pura
- Una clase declarada como interfaz no puede tener ningún método implementado


Herencia de Interfaz

- Para heredar de una interfaz se usa la palabra reservada implements

  public class MiClase implements Serializable
- Reglas:
  - Una interfaz puede heredar de otra interfaz
  - Una clase (abstracta o no) puede heredar de una interfaz
  - Una interfaz NO puede heredar de una clase
- Es un mecanismo muy usado en Java

Más sobre Herencia en Java

- Sólo se puede hacer herencia de implementación (extends) de una clase
  - No se crean problemas de referencias circulares o alternativas a un método con la misma declaración
- Se puede hacer herencia de interfaz (implements) de todas las interfaces que se quiera

Polimorfismo

- Gracias a la herencia, se puede interpretar que un objeto de una subclase es también un objeto de una superclase
- El polimorfismo es un mecanismo que se aprovecha de la herencia (especialmente de interfaz) para manejar indistintamente objetos de las subclases como si fuesen objetos de la clase base, sin preocuparse por la clase en concreto a la que pertenecen
- Interesa utilizarlo cuando un comportamiento varía en función del tipo de algo


Polimorfismo

- Se declaran atributos, parámetros o variables de la clase base
- Se les asignan objetos de alguna de las subclases
- Estamos seguros de que se pueden usar todos los métodos declarados en la clase base
- Si necesitamos usar métodos de las subclases es necesario hacer un cast


Polimorfismo

- La utilización del cast aumenta la posibilidad de hacer conversiones erróneas, por lo que es mejor evitarlo
- Se puede preguntar por la clase a la que pertenece un objeto:
  - instanceof
  - objeto.getClass().getName()
- Es una mala idea. En general, preguntar por la clase de un objeto implica un diseño malo y problemas para modificar el diseño y el código