Utilización avanzada de clases

Adolfo Sanz De Diego

Junio de 2011

1 ¿Diseño incorrecto?

- ¿Cómo sabemos si nuestro diseño es incorrecto?
  - las clases son difíciles de cambiar
  - las clases son difíciles de reutilizar
  - las clases son difíciles de usar
  - las clases tienen código repetido
  - las clases dejen de funcionar sin motivo aparente

2 Mejorarlo: principios SOLID

S	SRP	(The **S**ingle Responsibility Principle o **Principio de Responsabilidad Única**)	Las clases se diseñan sólo para un propósito.	Tom DeMarco	1

O	OCP	(The **O**pen/Closed Principle o **Principio Abierto/Cerrado**)	Las clases debe permitir ser extendidas, sin necesidad de ser modificadas.	Bertrand Meyer	1

L	LSP	(The **L**iskov Substitution Principle o **Principio de Sustitución de Liskov**)	Las clases padre siempre deben poder ser sustituidas por sus clases hijas y viceversa.	Barbara Liskov	1

I	ISP	(The **I**nterface Segregation Principle o **Principio de Segregación de Interfaces**)	Las clases no deben ser forzados a depender de interfaces con métodos que no utilizan.	Robert C. Martin	1

D	DIP	(The **D**ependency Inversion Principle o **Principio de Inversión de Dependencias**)	Los clientes deben delegar la gestión de las instancias de objetos a las librerias.	Martin Fowler	2

3 ¿Cómo utilizar nuestras clases?

- **Colaboración**
  - Cuando 2 o más clases colaboran de distintas formas (ver teoría POO)
- **Herencia**
  - Cuando una clase hija extiende una clase padre.
- Error común
  - Aplicar herencia para todo.

4 Herencia

- En Java no hay herencia múltiple pero una clase puede implementar varias Interfaces.
- Las **Interfaces** son clases que sólo definen métodos (comportamientos) y que obligan a las clases hijas a implementarlos.
- Las **clases abstractas** son clases que no se pueden instanciar, osea que sólo se pueden heredar. Implementan varios métodos, y definen otros como abstractos, obligando a las clases hijas a implementarlos.

5 Clases padre y clases hijas

- El operador **this** hace referencia a la instancia concreta del objeto.
- El operador **super** hace referencia a la clase padre.
- Los métodos no marcados como final y/ como private, se pueden sobreescribir en las clases hijas (**polimorfismo**).