Tecnología de la Programación 14-15

Tema 5. Polimorfismo y vinculación dinámica

Alberto Díaz
Purificación Arenas, Yolanda García,
Marco Antonio Gómez, Simon Pickin

Polimorfismo

- Hemos visto hasta ahora:
  - **Sobrecarga**
    - dos métodos tienen el mismo nombre, mismo tipo de retorno pero difieren en los parámetros (número y/o tipo).
  - **Sobreescritura o redifinición**
    - Una clase hija redefine un método de su superclase.
- **Variable polimórfica:** Una variable se declara de un tipo (clase) pero durante la ejecución puede contener valores de distinto tipo (subclases).

Polimorfismo

- Supongamos las clases A y B

```
public class A { ...  }
public class B extends A { ... }
```

- Una variable de la superclase A puede contener referencias a objetos de la subclase B

```
A a;
a = new B();
```

- El polimorfismo implica que toda variable tiene un **tipo estático** y otro **dinámico**.
- El *tipo estático* es el tipo asociado en la declaración.
- El *tipo dinámico* es el tipo del objeto contenido dentro de una variable.
- A es el tipo estático de la variable a y B es el tipo dinámico de a.
- El tipo estático siempre se determina en tiempo de compilación
- El tipo dinámico en general sólo se puede conocer en tiempo de ejecución y puede cambiar ⇒ polimorfismo inclusivo

Polimorfismo

- Dada una variable referencia x de la clase X, Java permite *invocar a los métodos* y *acceder a los atributos* conocidos para el *tipo estático* de x

```
public class A
{ ... public void incX() { x++; } ... }
public class B extends A
{ ... public void incZ() { z++; } ... }

A a = new A();
a.incX(); // Ok
a.incZ(); // error
B b = new B();
b.incX(); // Ok
b.incZ(); // Ok
```

Polimorfismo

- Java sólo permite *asignar* una expresión a una variable referencia de tipo X si el tipo de la expresión es X o una subclase de X

```
B b = new A();
// error, el objeto no pertenece a la clase B

A a = new B(); // Ok
...
B b = a;
// error, el tipo estático de a no es una subclase de B
```


Polimorfismo y casting

- Un objeto de la clase A se puede convertir a una referencia de la clase B mediante un *cast*, si su tipo dinámico es B

```
A a = new B();
B b = (B)a;
b.incZ();         // Ok
( (B)a ).incZ(); // Ok
```

- Si la clase B (que hereda de A) contiene algún método no declarado en la clase A, ni en ninguna de sus posibles superclases, entonces hay que hacer casting
  - En el ejemplo para poder invocar a incZ() con una referencia de la clase A a un objeto de la clase B

Polimorfismo y casting

- No todo objeto de la clase A se puede convertir a la clase B

```
A a = new A();
...
B b = (B)a;
// Ok, en compilación, pero
// error en tiempo de ejecución
```

- Java comprueba durante la ejecución todas las *conversiones explícitas* (casts)
- Si el objeto no pertenece a la clase a la cual se pretende convertir, entonces se produce una *excepción*

- La conversión manual de tipos (casting) en tiempo de ejecución se debe evitar.

Vinculación dinámica

- Existe un problema cuando se combina polimorfismo con redifinición

```
public class A {
      private int x, y;
      public void print() { System.out.println(x + " " + y); }
class B extends A {
      private int z;
      void print() {  super.print();  System.out.println(z);  } }
```

- ¿Qué método se invoca en el siguiente caso?

```
A a = new B(1, 2, 3);
a.print(); // ?
```

Vinculación dinámica

- ¿Qué método se invoca en el siguiente caso?

```
A a = new B(1, 2, 3);
a.print(); // ?
```

- Se invoca el método definido para el tipo dinámico de la variable a, es decir, B => 1 2 3
- Enlazar el nombre de un método con el código que se ejecutará para un objeto determinado se denomina *vinculación dinámica*
- El método que finalmente se invocará en general sólo se conoce durante la *ejecución* y no durante la *compilación*

Vinculación dinámica

- La regla general de búsqueda es

> Se ejecuta el método correspondiente al tipo dinámico más específico que se pueda aplicar según los parámetros del método

Vinculación dinámica: ejemplo

```
public class Animal
{
   ...
}
public class Perro extends Animal
{
   public void comer();
}
public class Gato extends Animal
{
   public void comer();
}
```


Vinculación dinámica: ejemplo

```
public class Test
{
    public static void main (String args[])
    {
       Perro p = new Perro();
       Gato g = new Gato();
       Animal granja[] = new Animal[2];
       granja[0] = p;
       granja[1] = g;
       for (int i = 0;i < 2; i++)
       {
              granja1[i].comer();
       }
    }
}
```

Polimorfismo con Interfaces: ejemplo

```
interface Animal
{
   public void habla();
}

interface Bestia
{
   public void grita();
}
```


Polimorfismo con Interfaces: ejemplo

```
public class Vaca implements Animal,Bestia
{
    public void habla()
    {
        System.out.println("Soy una vaca");
    }
    public void grita()
    {
        System.out.println("Que Soy una vaca!!!!!!");
    }
}
public class Perro implements Animal,Bestia
{
    public void habla() {...}  public void grita() {...}
}
public class Gato implements Animal,Bestia{. . . }
```

Polimorfismo con Interfaces: ejemplo

```
public class Test
{
    public static void main (String args[])
    {
       Perro p = new Perro();
        Gato g = new Gato();
        Vaca v = new Vaca();
        Animal granja1[] = new Animal[3];
        granja1[0] = p; granja1[1] = g; granja1[2] = v;
        Bestia granja2[] = new Bestia[3];
        granja2[0] = p; granja2[1] = g; granja2[2] = v;
        for (int i = 0;i<3; i++)
        {
               granja1[i].habla(); granja2[i].grita();
        }
    }
}
```


Casting y tipo de un objeto en ejecución

- No todo objeto de la clase A se puede convertir a la clase B

```
A a = new A();
...
B b = (B)a;
// Ok, en compilación, pero
// error en tiempo de ejecución
```

- Si el objeto no pertenece a la clase a la cual se pretende convertir, entonces se produce una *excepción*
- Antes de hacer un casting se puede preguntar por el tipo del objeto en tiempo de ejecución.
- Podemos hacerlo de dos formas:
  - Utilizando getClass(), heredado de la clase Object:
    - Devuelve la clase exacta de un objeto en tiempo de ejecución.
  - Utilizando (obj instance of NombreClase):
    - Comprueba si el objeto de tipo o tiene un tipo compatible con la clase C.

Ejemplo: Clase Object: método equals()

- Redefinición de equals // por defecto sólo referencias

```
public class Otra
{
    private int x;
    public boolean equals (Object obj)
    {
        if (this == obj) return true;
        if (obj == null) return false;
        if (this.getClass() != obj.getClass()) return false;

        Otra otra = (Otra) obj;
        return otra.x == this.x;
    }
}
```


Tipo de un objeto en ejecución

- Dada una jerarquía de clases:

```
public class A {…}
public class B extends A {…}
public class C extends B {…}
```

```
A a = new B();
(a instanceof B)?
(a instanceof A)?
(a instanceof C)?
(a.getClass()==A.class)?
(a.getClass()==B.class)?
```