

Polimorfismo

Polimorfismo

- El polimorfismo es una habilidad de tener varias formas; por ejemplo, la clase Jefe tiene acceso a los métodos de la clase Empleado.
- Un objeto tiene sólo un forma.
- Una variable tiene muchas formas, puede apuntar a un objeto de diferentes maneras.
- En Java hay una clase que es la clase padre de todas las demás: java.lang.Object.
- Un método de esta clase (por ejemplo: toString()) que convierte cualquier elemento de Java a cadena
- de caracteres), puede ser utilizada por todos.

Polimorfismo

- Java permite apuntar a un objeto con una variable definida como tipo de clase padre.

```
Empleado e = new Jefe ();
```

- Sólo se puede acceder a las partes del objeto que pertenecen a la clase Empleado; las partes específicas de la clase Jefe no se ven. Este efecto se consigue porque, para el compilador, e es sólo una variable de tipo Empleado, no Jefe.

```
e.departamento = "Finanzas";  
//Incorrecto
```

El operador instanceof redundante

```
public class Empleado extends Object  
public class Jefe extends Empleado  
public class Contractor extends Empleado
```

```
public void método (Empleado e){  
    if (e instanceof Jefe) {  
        // Obtiene beneficios por su salario  
    }  
    else if (e instanceof Contractor) {  
        //Obtiene tarifa por horas  
    }  
    else {  
        //empleos temporales  
    }  
}
```

Conversión de objetos

Utiliza instanceof para verificar el tipo de objeto.

Restablecer la funcionalidad total de un objeto mediante una conversión.

Comprobar la conversión apropiada con:

La conversión hacia clases superiores en la jerarquía se hace implícitamente (con una asignación).

Conversión de objetos

Conversión hacia abajo, han de ser hacia subclases y el compilador las comprueba.

El tipo del puntero se comprueba en tiempo de ejecución, cuando hay errores. Estos errores en tiempo de ejecución se llaman excepciones.

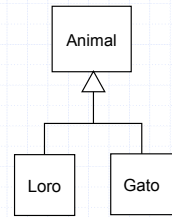
Conversión de objetos

```
public void método (Empleado e){
    if (e instanceof Jefe) {
        Jefe m = (Jefe) e;
        System.out.println("Este es el
        director de " +
        m.departamento);
    }
    //resto de la función
}
```

Si no se hace la conversión y se intenta acceder a e.departamento el compilador no encuentra el miembro departamento en la clase Empleado.

Casting

- Casting automático
Loro c = new Loro();
Animal a = c;
- Se necesita casting explícito
Animal a = new Loro();
Loro c = (Loro) a;
- Error de compilación:
Loro c = new Loro();
Gato d = (Gato) c;



Sobreescritura de métodos

Una subclase puede modificar los métodos que ha heredado del padre.

Una subclase puede crear un método con diferente funcionalidad al método del padre, pero con el mismo:

- Nombre
- Tipo de retorno
- Lista de argumentos

Recordar que los métodos, con el mismo nombre y lista de argumentos distinta, dentro de la misma clase, se denomina sobrecarga. La lista de argumentos es lo que indica al compilador que se invoca

Sobreescritura de métodos

```
public class Empleado {
    String nombre;
    int salario;
    public String getDetails () {
        return "Nombre: " + nombre + "\n" + "Salario: " +
        salario; }
}

public class Jefe extends Empleado{
    String departamento;
    public String getDetails() {
        return "Nombre: " + nombre + "\n" + "Departamento: "
        + departamento;
    }
}
```

Sobreescritura de métodos

Llamada de métodos virtuales.

```
Empleado e = new Jefe;
e.getDetails ();
```

Se comprueba el tipo de la referencia estática (Empleado) en tiempo de compilación y el tipo de la referencia dinámica (Jefe) en tiempo de ejecución.

¿Cual de los getDetails() se ejecutará, el de la clase Empleado o el de la clase Jefe?.

Sobreescritura de métodos

Se obtiene el comportamiento del método asociado a la clase de la variable y no el comportamiento asociado al tipo de la variable en el compilador. Por tanto se ejecuta e.getDetails() ejecutando el del tipo real del objeto, es decir el método de Jefe.

A este comportamiento se le suele llamar invocación de métodos virtuales.

Hay una diferencia muy importante entre C++ y Java. En C++, sólo se modifica el comportamiento

si se define el método como virtual. En los lenguajes OO puros, esto es normal, pero con ello C++ gana tiempo en ejecución.

Sobreescritura de métodos

Para poder sobreescribir métodos en las clases descendientes se debe verificar:

- El tipo de retorno de los dos métodos ha de ser igual.
- El método de la subclase no puede ser menos accesible que el de la clase padre.
- El método de la subclase no puede provocar más excepciones que el método del padre.

Sobreescritura de métodos

```
public class Padre {
    public void método () { }
}
public class Hijo extends Padre {
    public void método () { }
    //no puede ser private
}
public class OtraClase{
    public void otroMetodo(){
        Padre p1 = new Padre();
        Padre p2 = new Hijo();
        p1.método();
        p2.método();
    }
}
```

Sobreescritura de métodos

La inicialización de los objetos es muy estructurada.

Cuando un objeto se inicializa:

- Se reserva memoria y se inicializa con el valor null.
- Se implementan las inicializaciones explícitas, de cada clase de la jerarquía.
- Se llama a los constructores de cada clase de la jerarquía.

Las últimas dos etapas se realizan en cada una de las clases de la jerarquía, empezando por arriba.

Java llama siempre al constructor de la clase padre, antes de ejecutar el constructor de la clase hija.

Sobreescritura de métodos

En muchas ocasiones, el constructor por defecto se utiliza en la clase padre.

Super o this tienen que estar en la primera línea de los métodos constructores.

```
public class Empleado {
    String nombre;
    public Empleado (String n){
        nombre = n;
    }
}
public class Jefe extends Empleado {
    String departamento;
    public Jefe (String s, String d) {
        super(s);
        departamento = d;
    }
}
```