

Interfaces y Clases Internas

ELO329: Diseño y Programación Orientados a
Objetos

¿Qué queremos decir con interfaces y clases internas?

- NOTA: El término interfaz aquí NO se refiere a las interfaces gráficas (ya viene ...).
- Aquí las **interfaces** son una manera de describir qué debería hacer una clase sin especificar el cómo.
- Faz =>entre caras => Interfaz, plural interfaces
- Las **clases internas** son clases anidadas dentro de otras clases o métodos.
- Interfaces y clases internas son recursos esenciales en el manejo de ***interfaces gráficas*** en Java. Será nuestro próximo tópico.

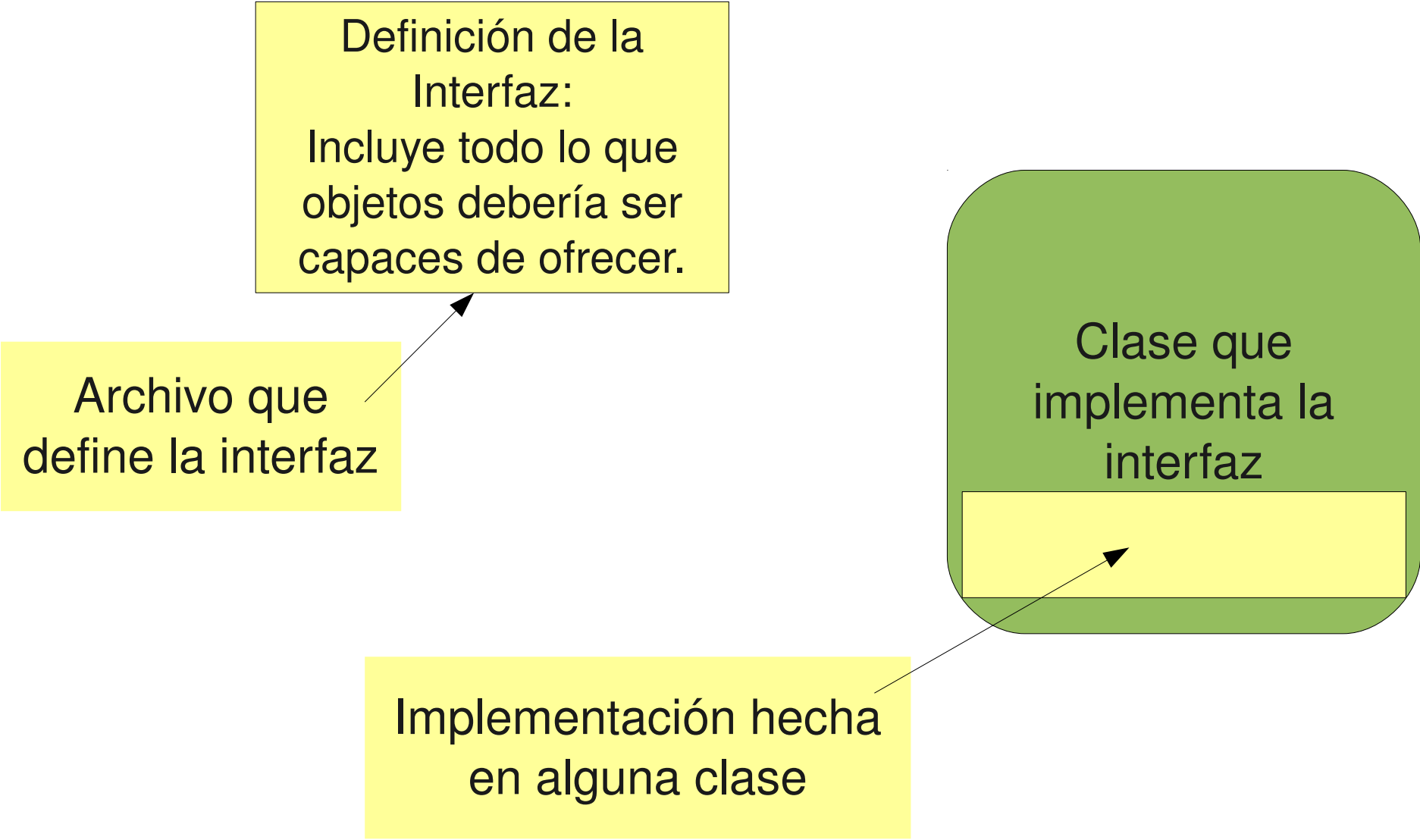
Interfaces

- Una **interfaz** es la descripción de uno o más servicios (métodos) que posteriormente alguna clase puede implementar (y por ende ofrecer).
- Por ejemplo, si un alumno sabe alemán, tenemos idea de lo que él es capaz. Además de ser persona (herencia) él cumple la interfaz “interprete de alemán”. También podríamos decir que él **es un** “interprete de alemán” (la misma relación que en herencia).
- **Gracias a las interfaces podemos crear métodos genéricos (como ordenar) que manipulen objetos y les exijan a éstos sólo funcionalidades mínimas requeridas (como poder compararse).**
- Lo mismo se puede pensar para personas que tienen certificación Java: esto es, algunas personas cumplen la interfaz programador Java..

Interfaces

Definición de la Interfaz:
Incluye todo lo que objetos debería ser capaces de ofrecer.

Archivo que define la interfaz



Implementación hecha en alguna clase

Clase que implementa la interfaz

Interfaces (cont.)

- En Java cada clase puede tener sólo una clase base (en Java no hay herencia múltiple).
- Cuando hay relación **es-un** con más categorías del mundo real, usamos herencia con una de ellas e interfaces para exhibir un comportamiento como el esperado por las otras.
- Se **cumple también el principio de sustitución**.
Instancias de la clase que **implementa** una Interfaz pueden ser usadas donde se espera una instancia de la interfaz. Es similar a usar una instancias de una subclase cuando se espera un objeto de la clase base.

Interfaces (cont.)

- No se permite crear instancias (objetos) de una Interfaz. Por la misma razón que no se puede crear instancias de clases abstractas, no se tienen la implementaciones.

`new InterfazX();` ✘

- Todos los métodos de una Interfaz son públicos. No es necesario indicarlo.
- Pueden incluir constantes. En este caso son siempre `public static final`.

Interfaces: Aspectos sintácticos

- Debemos atender dos cosas:
 - Si la interfaz no existe, debemos definirla.
 - Luego debemos implementar la interfaz en alguna clase.

Definición de una interfaz: En un archivo de nombre Comparable.java, poner:

```
public interface Comparable{  
    int compareTo (Object other);  
}
```

Implementación de una interfaz: En otro archivo implementar la interfaz:

```
class Employee implements Comparable {  
    ....  
    public int compareTo(Object other) {  
        ....// implementación  
    }  
}
```

Ejemplo: uso de interfaces

- Consideremos la extensión de la clase Employee para que podamos ordenar arreglos de empleados según su salario.
- La interfaz Comparable ya está definida en el lenguaje, luego sólo debemos implementarla.
- Ver EmployeeSortTest.java
- Ver documentación de clase Arrays e interfaz Comparable. Notar el métodos genérico sort de la clase Arrays.

Clases Internas

- Cuando sólo necesitamos crear una instancia de un objeto, podemos definir la clase dentro de otra clase.
- Son clases definidas dentro de otras (anidadas).
- Se permite el acceso a miembros de la clase anfitriona – incluso si son privados.
- Se usan como mecanismo de encapsulamiento. No son visibles desde fuera.
- Son muy útiles para reducir código fuente cuando la clase sólo genera instancias locales.
- Son comunes en el desarrollo de interfaces gráficas.

Clases Internas (Cont.)

- Las clases internas existen sólo para el compilador, ya que éste las transforma en clases regulares separando la clase externa e interna con signo \$.
- La máquina virtual no distingue la clases internas.
- También se pueden definir al interior de un método.

Ejemplo de Clase Interna

```
class BankAccount {  
    public BankAccount(double initialBalance) {  
        balance = initialBalance;  
    }  
  
    public void start(double rate)  
    {  
        ActionListener adder = new InterestAdder(rate);  
        Timer t = new Timer(1000, adder);  
        t.start();  
    }  
    private double balance;  
  
    private class InterestAdder implements ActionListener {  
        public InterestAdder(double aRate) {  
            rate = aRate;  
        }  
  
        public void actionPerformed(ActionEvent event) {  
            double interest = balance * rate / 100;  
            balance += interest;  
            NumberFormat formatter  
                = NumberFormat.getCurrencyInstance();  
            System.out.println("balance=" +  
                + formatter.format(balance));  
        }  
        private double rate;  
    }  
}
```

Ver InnerClassTest.java

Clase interna dentro de un método

```
class BankAccount {  
    public BankAccount(double initialBalance) {  
        balance = initialBalance;  
    }  
    public void start(double rate) {
```

```
        class InterestAdder implements ActionListener {  
            public InterestAdder(double aRate) {  
                rate = aRate;  
            }  
  
            public void actionPerformed(ActionEvent event) {  
                double interest = balance * rate / 100;  
                balance += interest;  
                NumberFormat formatter = NumberFormat.getCurrencyInstance();  
                System.out.println("balance=" + formatter.format(balance));  
            }  
            private double rate;  
        }  
    }
```

```
        ActionListener adder = new InterestAdder(rate);  
        Timer t = new Timer(1000, adder);  
        t.start();  
    }  
  
    private double balance;  
}
```

Ver: InnerClassMethodTest.java

Clases internas anónimas

- ¿Para qué definir una clase si sólo deseamos proveer una implementación a los métodos de una interfaz?
- Cuando necesitamos sólo una instancia de una clase que implementa una interfaz, no necesitamos darle un nombre. Decimos que tal clase es interna y anónima.
- Ver `AnonymousInnerClassTest.java`

Ejemplo: Clase Anónima

```
class BankAccount
{
    public BankAccount(double initialBalance) {
        balance = initialBalance;
    }
    //Única instancia

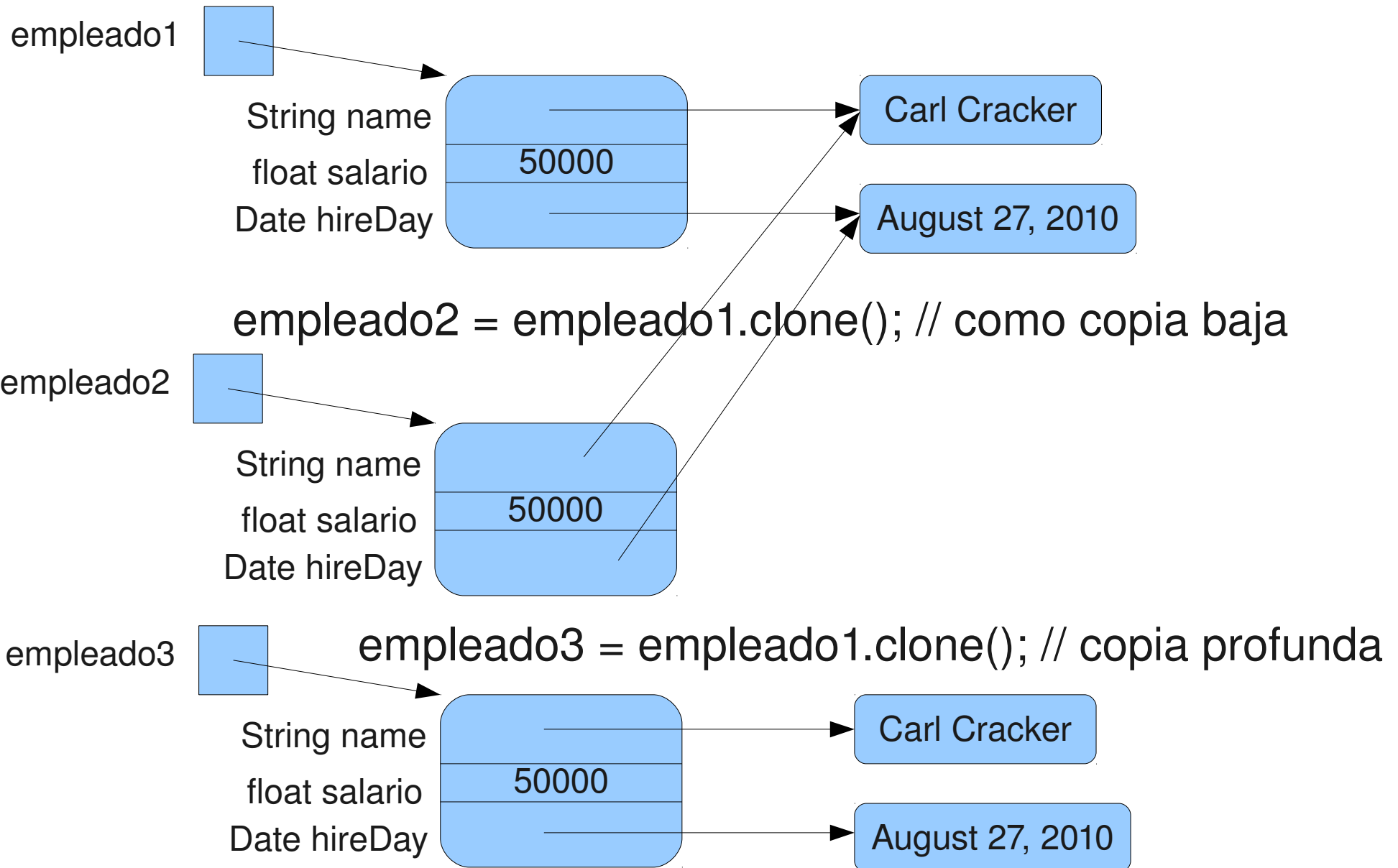
    public void start(final double rate) {
        ActionListener adder = new ActionListener() { // Implementación
            public void actionPerformed(ActionEvent event) {
                double interest = balance * rate / 100;
                balance += interest;
                NumberFormat formatter = NumberFormat.getCurrencyInstance();
                System.out.println("balance=" + formatter.format(balance));
            }
        };

        Timer t = new Timer(1000, adder);
        t.start();
    }
    //Ver
    private double balance;
}
AnonymousInnerClassTest.java
```

Método clone() en Object (revisitado)

- El método clone() existe con acceso protegido en la clase Object.
- Para invocarlo sobre un objeto se requiere que la clase del objeto implemente la interfaz Cloneable, lo cual significa que debemos redefinir el método clone.
- Para generar un clone correcto, debemos hacerlo invocando el método clone de la clase Object.
- El método clone de Object crea y retorna un objeto con igual estructura del objeto llamado e inicializa todos sus campos con el mismo contenido de los campos del objeto llamado.
- Los contenidos de cada campo no son clonados (hasta aquí se le llama copia baja), luego para una copia completa (profunda) se debe llamar el método clone con cada atributo.

Copia baja v/s copia profunda



Implementación de clone (copia profunda)

- La implementación típica es como sigue:

```
class Employee implements Cloneable {  
    public Object clone() { // redefinición de clone  
        try { // el manejo de excepciones de revisará más adelante  
            Employee c = (Employee) super.clone(); // no usamos constructor  
            c.hireDay = hireDay.clone();  
            return c;  
        } catch (CloneNotSupportedException e ) {  
            return null;  
        }  
    }  
    .....  
    private String name;  
    private float salary;  
    private Date hireDay;  
}
```

Hasta aquí copia baja

Necesarios para copia profunda

Ver CloneTest.java