Programación 4

Implementación

Manejo de Objetos

Contenido

- Introducción
- Referencias
- Objetos Compartidos
- Copia de Objetos
- Destrucción de Objetos

Introducción

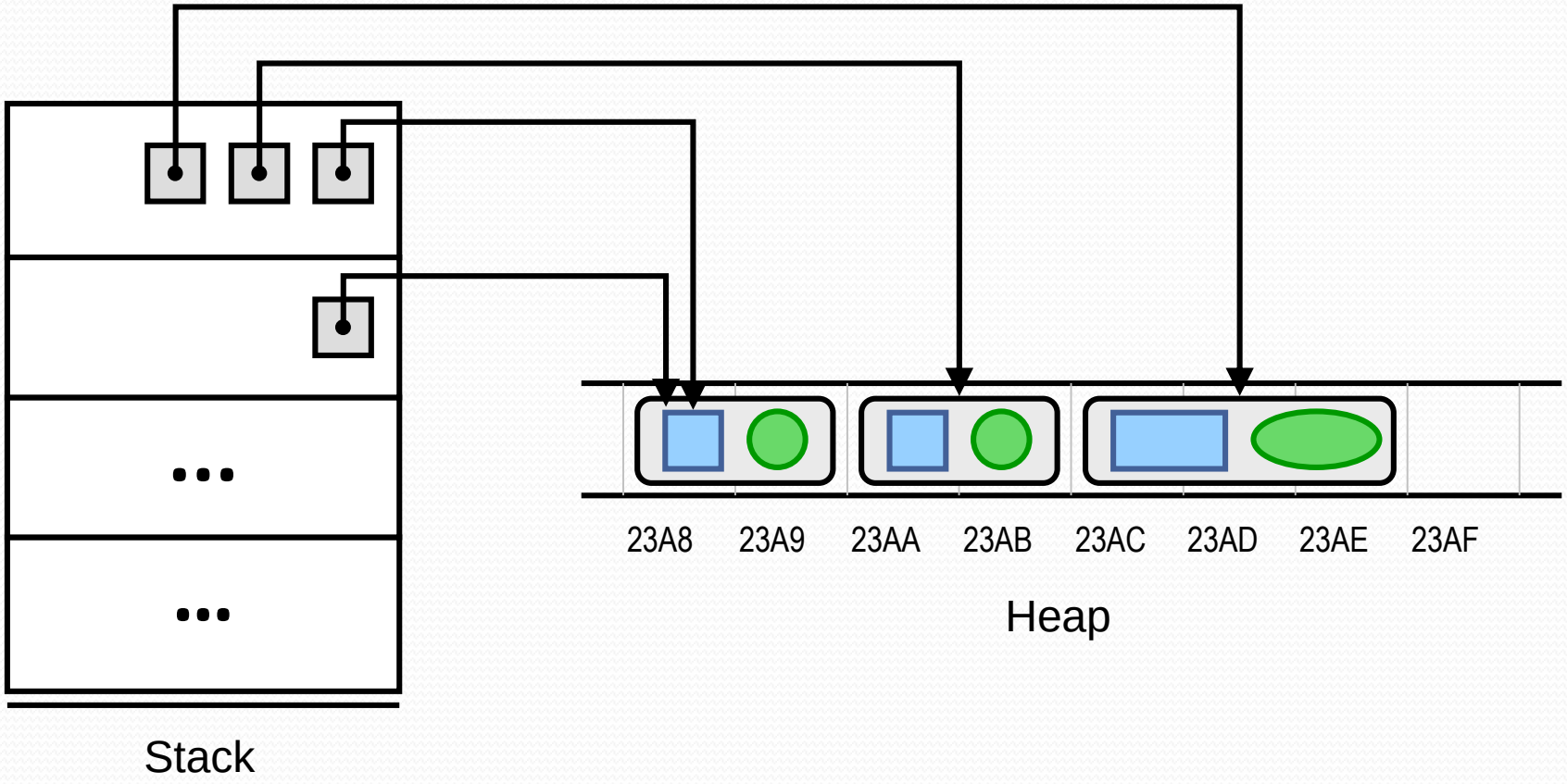
- Los objetos son manipulados a través de referencias
- Dependiendo de cómo los lenguajes de programación las implementen aplican ciertas consideraciones tanto a la manipulación como a la destrucción de objetos
- La identidad requiere que los objetos sean compartidos
- Esto hace que las copias necesiten ser examinadas en detalle

Referencias

- En tiempo de ejecución los objetos no son alojados en el stack sino en el heap
- La forma de acceder a un objeto es mediante referencias
- Una referencia es una variable (tipada) que es alojada en el stack (o en el heap si está dentro de un objeto) tal que
  - No identifica a ningún objeto (*void*)
  - Identifica a un objeto particular de una determinada clase (*attached*)

Referencias (2)

- Ejemplo:



Referencias (3)

- En algunos lenguajes de programación las referencias se implementan explícitamente
  - En C++ las referencias se implementan mediante punteros

```
ClaseA *aPtr;
// referencia a un objeto de clase A

ClaseA aObj;
// variable en el stack
```

Referencias (4)

- Otros lenguajes manejan referencias en forma implícita
  - En Java y C# no es posible definir objetos en el stack sino únicamente referencias

```
ClaseA a;
// 'a' es una referencia a un objeto
// de ClaseA y no un objeto
```

Referencias (5)

- Hacer que una referencia sea *void*
  - En C++: `a = NULL`
  - En Java y C#: `a = null`
- Hacer que una referencia sea *attached* (en cualquiera de los tres lenguajes)
  - Crear un objeto y adjuntar la referencia a él

```
ClaseA *a = new A(); // en el heap
ClaseA *b = null;
```

  - Adjuntar la referencia a un objeto obtenido a través de otra referencia

```
a = b;
```

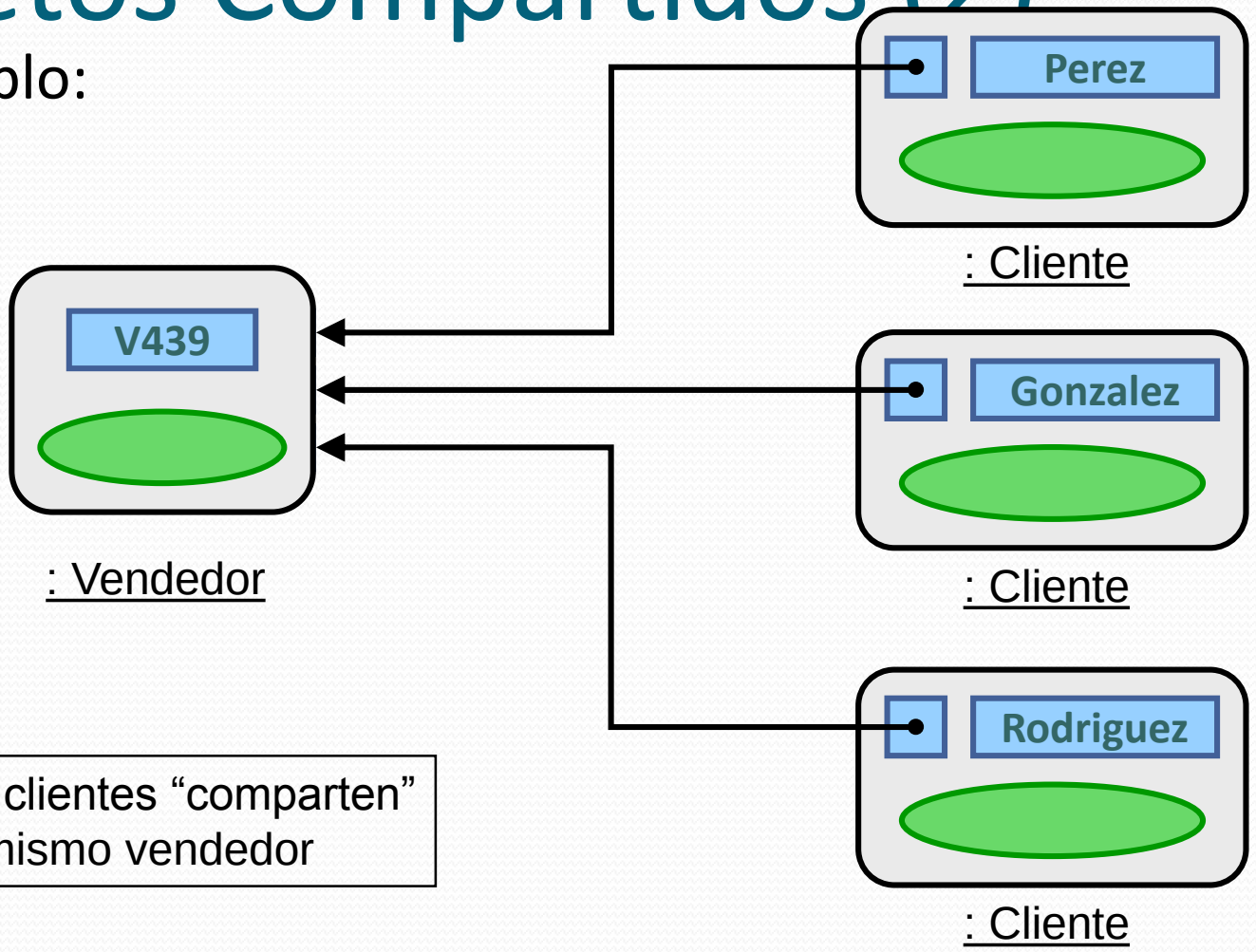
Asignación de referencias (punteros)

Objetos Compartidos

- En sistemas orientados a objetos es usual que un objeto sea “conocido” por otros varios objetos
- La identidad requiere que dichos objetos referencien al mismo objeto y no a copias de él
- Eso implica que el objeto será “compartido” por otros
- Esto se logra teniendo en cada objeto una referencia al objeto compartido

Objetos Compartidos (2)

- Ejemplo:



Los tres clientes “comparten” el mismo vendedor

Copia de Objetos

- La identidad y la necesidad de compartir objetos hace que en general no sea correcto copiar objetos
- Recordar que una copia de un objeto es otro objeto que luego de la copia tiene propiedades iguales a las del original
- A partir de la copia ambos elementos evolucionan independientemente

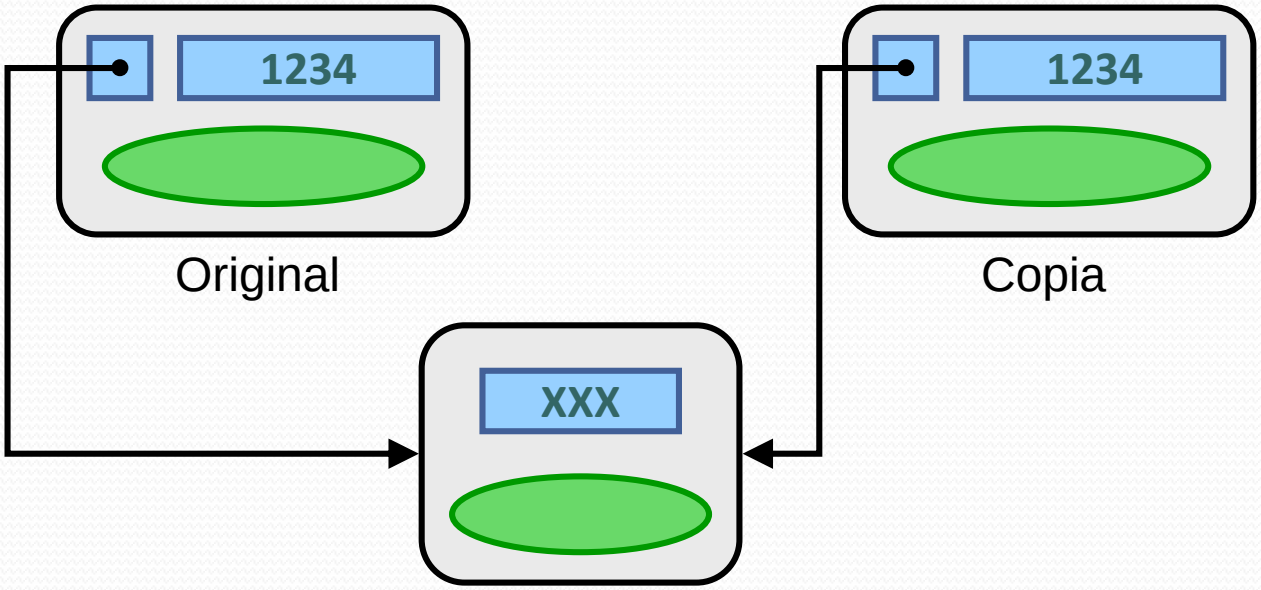
Copia de Objetos (2)

- En determinadas situaciones es aceptable la copia de objetos
- Distinguiremos tres casos:
  - Objetos que implementan data values
  - Objetos que son instancias de clases del diseño
  - Objetos que representan colecciones
- A su vez distinguimos dos enfoques de realizar copias de objetos:
  - Copia plana
  - Copia en profundidad

Copia de Objetos (3)

- Copia plana:
  - Su resultado es un objeto exactamente igual al original, incluyendo sus referencias
  - Ejemplo:

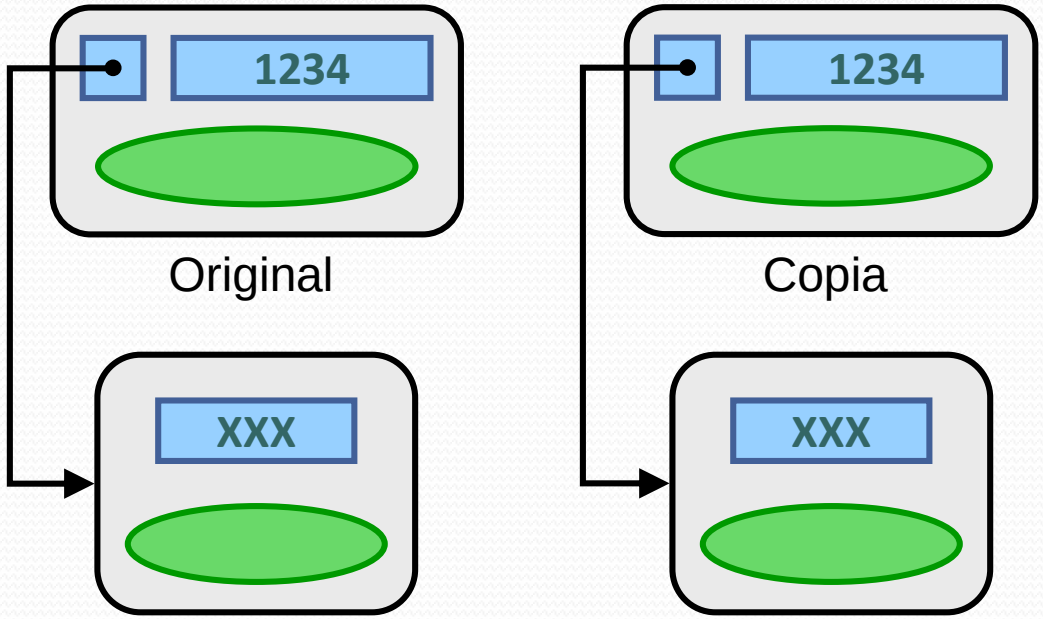
La copia es 100% igual al original



Copia de Objetos (4)

- Copia en profundidad:
  - El objeto resultante es exactamente igual al original, salvo las referencias
  - Ejemplo:

Los objetos referenciados son copiados en profundidad



Copia de Objetos (5)

- Copia de Data Values:
  - Algunos Data Types deben ser implementados mediante clases por lo cual sus instancias serán formalmente objetos
  - Estos objetos se pueden copiar dado que
    - No tienen identidad (ya que son data values)
    - Se desea disponer de un ejemplar diferente en cada lugar donde se lo requiere
  - La copia de data values se realiza **en profundidad**

Copia de Objetos (6)

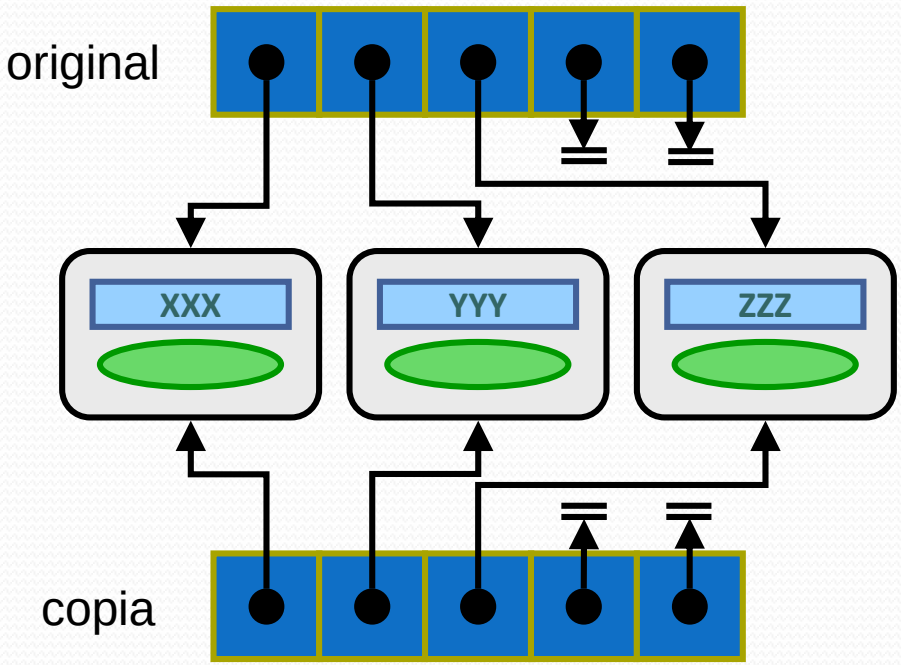
- Copia de Objetos:
  - Los objetos si tienen identidad
  - En caso de requerir a uno desde más de un lugar se debe compartirlo (no es aceptable copiarlo)
  - Como regla general NO se debe copiar objetos
  - Existen casos controlados donde es posible realizar copias de objetos

Copia de Objetos (7)

- Copia de Colecciones:
  - El caso de las colecciones es particular porque pueden involucrar
    - Una estructura de datos (sin identidad)
    - Objetos (con identidad)
  - Las colecciones de data values se tratan como el caso de los data values (en profundidad)
  - En casos en que sea necesario otra colección igual a la original se debe copiar solamente la estructura de datos (plana)

Copia de Objetos (8)

- Copia de Colecciones (cont.)
  - La copia de colecciones de objetos se realiza en forma **plana**
  - Ejemplo:



Destrucción de Objetos

- Los objetos alojados en el *heap* permanecen allí hasta que el programa termina (a diferencia de aquellos alojados en el *stack*)
- Cuando un objeto ya no es de utilidad se lo suele retirar del *heap* para liberar la memoria
- Existen dos enfoques para ello:
  - Automático, mediante el llamado *Garbage Collector* (Java, C#)
  - Manual (C++)
    - Hay librerías que implementan Garbage Collection en C++ (libgc)

Destrucción de Objetos (2)

- Garbage Collector:
  - Forma parte del ambiente de ejecución del lenguaje de programación
  - Corre en paralelo con el programa
    - Busca objetos en el *heap* tales que no exista ninguna referencia adjunta a ellos
    - Cuando encuentra un objeto tal lo elimina y libera la memoria que éste ocupa
  - Permite al programador solicitar memoria sin tener que preocuparse por “devolverla”

Destrucción de Objetos (3)

- Destrucción Manual:
  - Este enfoque es más complejo y delicado
  - Requiere que el programador explícitamente libere la memoria ocupada por un objeto
  - Problemas frecuentes
    - Memoria inaccesible
    - Referencias colgantes

Destrucción de Objetos (4)

- Destrucción Manual (cont.)
  - **Memoria inaccesible**: esto ocurre cuando un objeto no tiene ninguna referencia adjunta a él
  - Ejemplo:

```
void memLeak(){
    Empleado * e;
    e = new Jornalero();
}
```

La única referencia adjunta al jornalero recién creado se perdió cuando se llega a la llave de cierre. En consecuencia el jornalero queda inaccesible.

Destrucción de Objetos (5)

- Destrucción Manual (cont.)
  - **Referencias colgantes**: esto ocurre cuando un objeto es compartido y se destruye a través de una de las referencias
  - Ejemplo:

```
Empleado *colgante(){
    Empleado * e1, *e2;
    e1 = new Jornalero();
    e2 = e1;
    delete e2;
    return e1;
}
```

Moraleja: ¡¡cuidado al destruir objetos compartidos!!

Destrucción de Objetos (6)

- Enfoques para la destrucción manual de objetos
  - Desarrollar una estrategia particular: en función de las particularidades del problema el programador “sabe” cuándo y cómo eliminar un objeto en forma segura (típicamente quien crea el objeto será quien lo destruya).
    - Hay programas que permiten detectar memory leaks, referencias colgantes pero son muy lentos (valgrind)
  - No destruir objetos: No es aplicable en sistemas donde se cree una gran cantidad de objetos
  - Utilizar contadores de referencias: cada objeto contiene un contador de referencias adjuntas a él. No soluciona los ciclos de referencias.