

Programación Avanzada

Implementación

Generación de Código

Contenido

- Introducción
- Implementación de una Colaboración:
 - Implementación de la Estructura
 - Implementación de la Interacción
- Sugerencias

Introducción

- Propósito: realizar la implementación de una parte del diseño (una colaboración)
- El resultado es código fuente en la forma de Elementos de Implementación
- Una tarea de implementación se enfoca en obtener cierta funcionalidad (al implementar la realización de un caso de uso) que implica la implementación de diferentes elementos de diseño que contribuyan a dicha funcionalidad

Implementación de una Colab.

- Para implementar una colaboración (que realice caso/s de uso):
 - Implementar la estructura de la colaboración
 - Implementar interfaces
 - Implementar clases
 - Implementar atributos
 - Implementar operaciones
 - Implementar relaciones
 - Implementar generalizaciones
 - Implementar realizaciones
 - Implementar asociaciones
 - Implementar las interacciones de la colaboración
 - Implementar métodos

Implementar la Estructura

Implementar Interfaces

- Las interfaces se implementan directamente a partir del DCD
- Las operaciones se obtienen de la propia especificación de la interfaz
- **Advertencia:** algunos lenguajes de programación no proveen una construcción para implementar directamente interfaces:
 - En esos casos se suele implementar una clase abstracta, sin atributos y con todas sus operaciones abstractas

Implementar la Estructura

Implementar Interfaces (2)

```
class IControladorUsuario { // interfaz
public:
    virtual bool login(string usuario, string password) = 0;
    virtual void logout() = 0;
    virtual bool actualizarPassword(string passwordAntiguo,
                                    string passwordNuevo) = 0;
    virtual set<DataPermiso> permisos() = 0;
    virtual set<DataUsuario> getAmigos() = 0;
    virtual DataUsuario getPerfil() = 0;
    // ...

    virtual ~IControladorUsuario(){}; // virtual y vacío
};
```

Implementar la Estructura

Implementar Clases

- La implementación de las clases se hace en forma directa a partir del DCD
- Los lenguajes de programación orientados a objetos incluyen una construcción para este fin (la clase)
- Los atributos y operaciones se obtienen de la propia especificación de la clase
 - Se incluyen los constructores y destructor
 - También las operaciones de acceso y/o modificación de los atributos

Implementar la Estructura

Implementar Clases (2)

```
class Tweet {
private:
    Usuario *usuario;
    string contenido;
public:
    // constructores
    Tweet(Usuario *u, contenido c);

    // métodos de acceso
    Usuario getUsuario();
    string getContenido();

    // otros métodos
    void agregarHashtag(string hashtag);
    /* ... */
private:
    static set<string> parsearHashtags(string s);
    /* ... */
public:
    virtual ~Tweet();
};
```


Implementar la Estructura

Implementar Relaciones

- Las relaciones entre elementos de diseño empleadas son:
 - Generalizaciones
 - Realizaciones
 - Asociaciones
 - Dependencias

Relaciones – Generalizaciones

- Las generalizaciones se obtienen directamente del DCD
- Los lenguajes de programación orientados a objetos proveen una construcción para esto
 - En la declaración de la clase se especifica su ancestro (muchos lenguajes permiten sólo uno)
- Ejemplos:
 - Java: `class Jornalero extends Empleado`
 - C++: `class Jornalero: public Empleado`
 - C#: `class Jornalero: Empleado`

Implementar la Estructura

Relaciones – Realizaciones

- Las realizaciones también se obtienen directamente del DCD
- Los lenguajes de programación que no proveen interfaces tampoco proveen realizaciones
 - En la declaración de la clase se especifica la(s) interfaz(ces) que realiza
 - En C++ se utiliza una generalización
- Ejemplos:
 - Java: `class` ControladorUsuario `implements` ControladorUsuario
 - C#: `class` ControladorUsuario : ControladorUsuario
 - C++: `class` ControladorUsuario : `public` ControladorUsuario

Implementar la Estructura

Relaciones – Asociaciones

- Los lenguajes de programación generalmente no proveen una construcción específica para la implementación de asociaciones
- Para que una clase A pueda estar asociada a una clase B se suele incluir un atributo en A
 - Este atributo no pertenece al conjunto de atributos definidos en el diseño por lo que se lo denomina “pseudoatributo”
- A través del pseudoatributo una instancia de A puede mantener una referencia a otra de B y así implementar el link

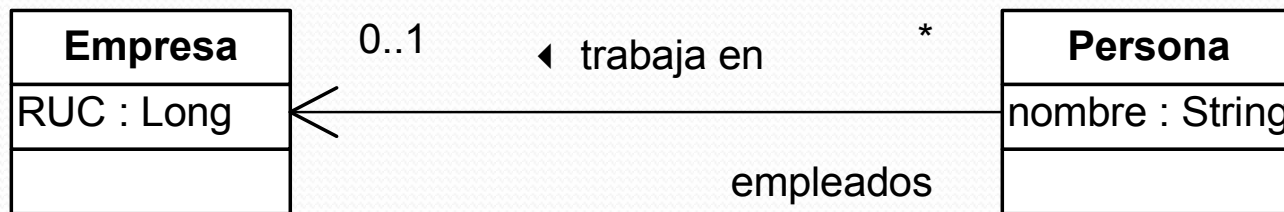
Implementar la Estructura

Relaciones – Asociaciones (2)

- Se define un pseudoatributo en A solamente si la asociación es navegable hacia B
- El tipo de un pseudoatributo para A depende de la clase B, pero también de la multiplicidad en el extremo de la asociación del lado de B
- Se distinguen dos casos dependiendo del máximo de dicha multiplicidad:
 - Si el máximo es 1 (0..1 ó 1)
 - El pseudoatributo es de tipo **B ***
 - De lo contrario (0..*; 1..*; etc.)
 - El pseudoatributo es de tipo colección (`ICollection`; `set`; `map<T, B>`, etc.) dependiendo de las necesidades de navegación

Relaciones – Asociaciones (3)

- Caso 1:

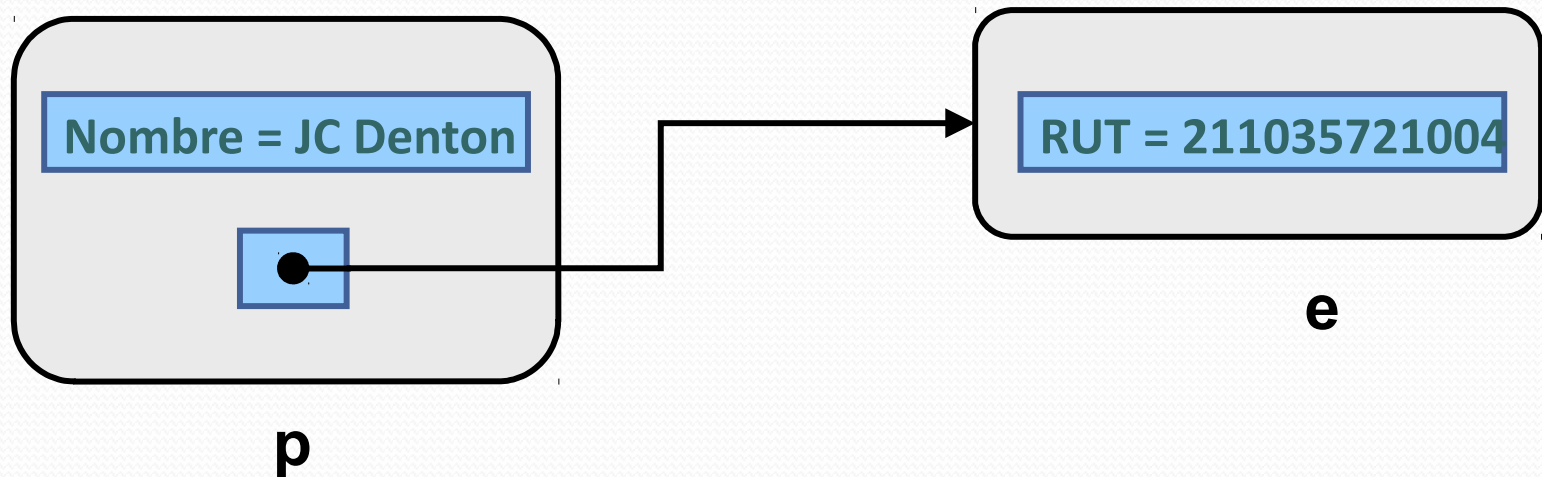


- Ejemplo en C++

```
class Persona {
    private:
        String nombre;
        Empresa * empresa; // pseudoatributo
    public:
        ...
};
```

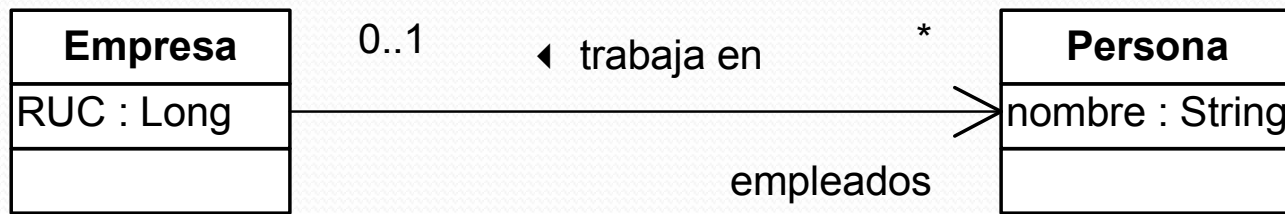
Relaciones – Asociaciones (4)

- Caso 1 (cont.):
 - Una persona puede tener una referencia a una empresa



Relaciones – Asociaciones (5)

- Caso 2:

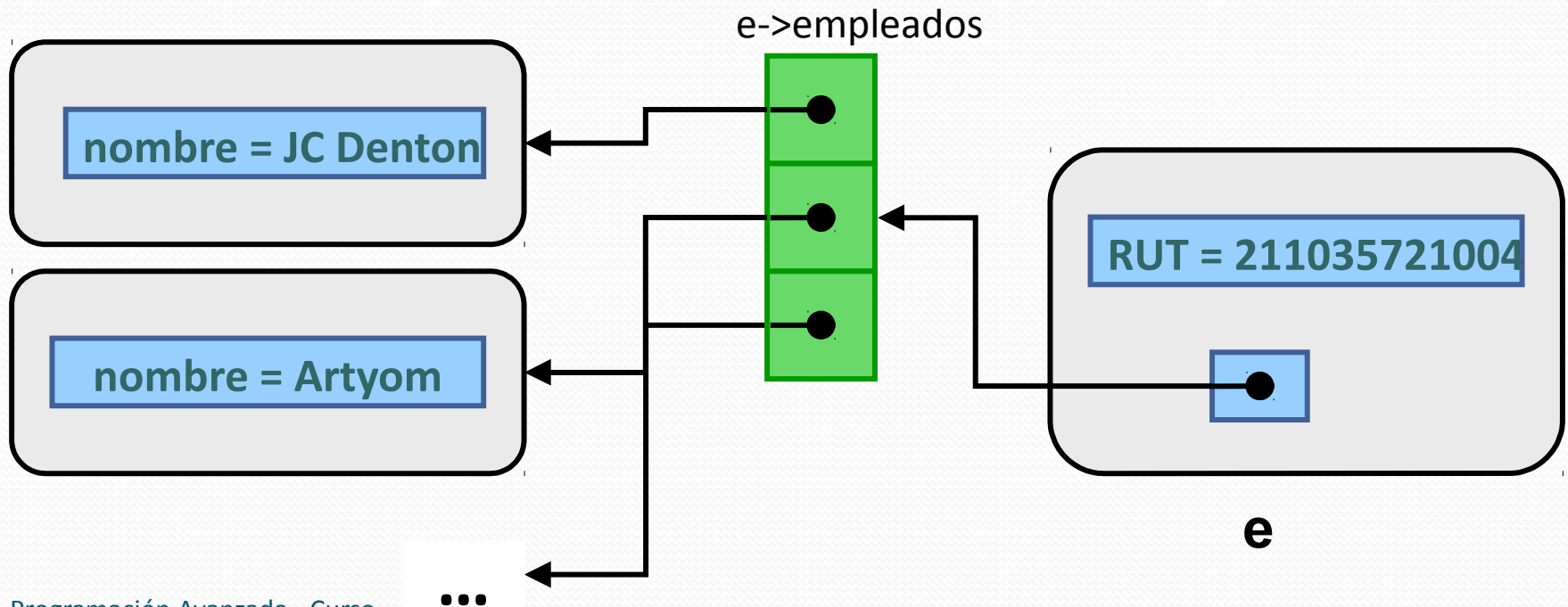


- Ejemplo en Java

```
class Empresa {
    private long RUC;
    private Lista *empleados; // pseudoatributo
    ...
};
```

Relaciones – Asociaciones (6)

- Caso 2 (cont.):
 - Una empresa tiene una colección de referencias a personas



Relaciones – Asociaciones (7)

- Caso 2 (cont.):
 - La elección de la estructura de datos que implemente la colección se realiza en función de simplicidad, disponibilidad, requerimientos de eficiencia, etc.
 - En casos en que el extremo de asociación tenga aplicada la restricción {ordered} es necesario utilizar una colección que permita ordenamiento
 - Muchos ambientes de programación cuentan con bibliotecas de clases con diferentes tipos de colecciones predefinidas. En C++ está STL

Implementar la Estructura

Relaciones – Dependencias

- Las dependencias se declaran en la definición de un elemento para tener visibilidad sobre otros
- Esto se hace cuando en el DCD existe una dependencia desde un elemento A hacia otro B
 - Una asociación navegable, una generalización y una realización son también formas de dependencia
- En C++ se utiliza `#include`
- En Java se utiliza `import`
- En C# se utiliza `using`

Implementar las Interacciones

- La implementación de la estructura conduce a la definición de los elementos de diseño junto con sus relaciones
- Las clases incluyen sus operaciones pero no los métodos asociados
 - Esto significa que no existen invocaciones implementadas por lo que aún no hay comportamiento
- A partir de los diagramas de interacción se extrae información para implementar los métodos

Implementar las Interacciones

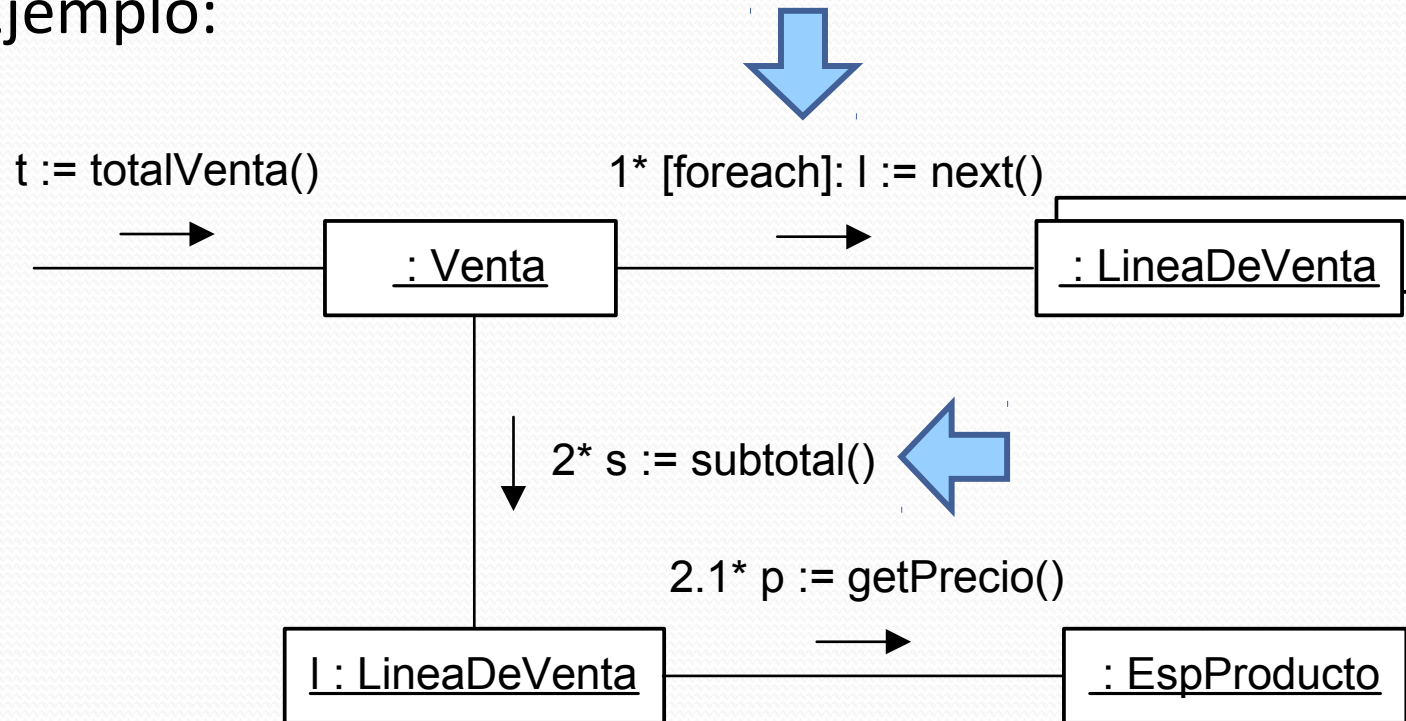
Implementar Métodos

- Un diagrama de comunicación no tiene como objetivo servir de pseudocódigo
- Sin embargo generalmente ilustra la lógica general de las operaciones
- Al implementar el método asociado a la operación `op()` en una clase A:
 - Se busca el diag. de comunicación que incluya un mensaje `op()` llegando a una instancia de A
 - La interacción anidada en ese mensaje debería resultar de ayuda para implementar el método

Implementar las Interacciones

Implementar Métodos (2)

- Ejemplo:



Para implementar el método asociado a `totalVenta()` observamos la interacción anidada en el mensaje (en el primer nivel) en el diagrama de comunicación

Implementar las Interacciones

Implementar Métodos (3)

- Ejemplo (cont.):

```
class Venta {  
private:  
    ICollection *lineas;  
public:  
    float totalVenta();  
    virtual ~Venta();  
};
```

```
float Venta::totalVenta() {  
    float total = 0;  
  
    for(IIterator *it = lineas->getIterator(); it->hasCurrent(); it->next()) {  
        LineaDeVenta *ldv = dynamic_cast<LineaDeVenta *>(it->getCurrent());  
        total += ldv->subtotal();  
    }  
    return total;  
}
```


Sugerencias

- Antes de implementar una clase desde cero es recomendable considerar si código existente puede ser reutilizado o adaptado
- Comprender en qué lugar de la arquitectura encaja la implementación ayuda a:
 - Identificar oportunidades de reuso
 - Asegurar que el código nuevo sea coherente con el del resto del sistema

Sugerencias (2)

- Orden de implementación de las clases:
 - Las clases deben ser implementadas comenzando por las menos acopladas y finalizando por las más acopladas
 - De esta forma las clases van disponiendo de todos los elementos necesarios para su implementación
 - Esto permite que al terminar de implementar una clase se pueda testear inmediatamente