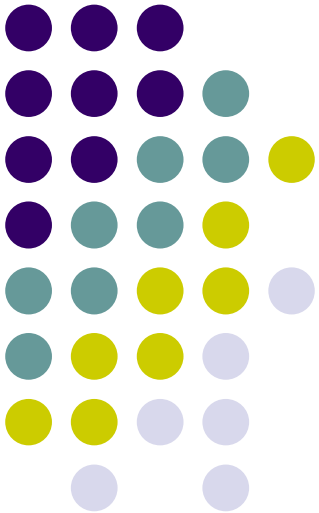
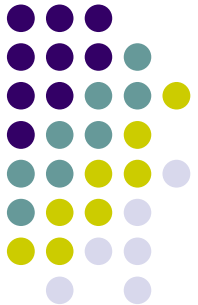
Metodologías de Diseño y Programación

Implementación

Generación de Código

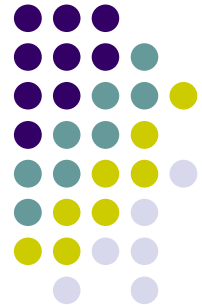


Contenido



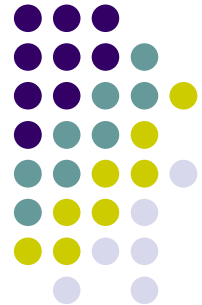
- Introducción
- Modelo de Implementación
- Implementación de una colaboración
- Sugerencias

Introducción



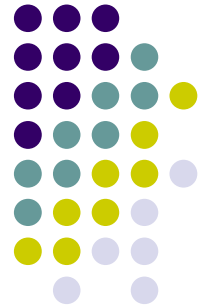
- Propósito: realizar la implementación de una parte del diseño (una colaboración)
- El resultado es código fuente en la forma de Elementos de Implementación
- Una tarea de implementación se enfoca en obtener cierta funcionalidad (al implementar la realización de un caso de uso) que implica la implementación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

Modelo de Implementación



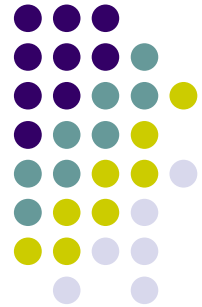
- El Modelo de Implementación representa la composición física de la implementación
- Está expresada principalmente en términos de Elementos de Implementación
- Éstos son típicamente elementos físicos como archivos, pero también directorios
  - Archivos de código (fuentes, binarios, ejecutables)
  - Archivos de datos y configuración
- Dichos elementos pueden organizarse en Subsistemas de Implementación

Modelo de Implementación (2)



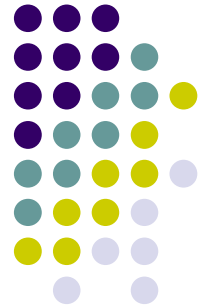
- Contenido:
  - **Introducción:** Breve descripción que sirve como introducción al modelo
  - **Subsistemas de implementación:** Conjuntos de Elementos de Implementación, definen una jerarquía
  - **Elementos de implementación:** Todos los archivos que conforman la implementación del sistema, contenidos en los subsistemas

Modelo de Implementación (3)



- Contenido (cont.):
  - **Relaciones:** Las relaciones del modelo entre elementos de implementación, contenidas en los subsitemas
  - **Diagramas:** Representación de los elementos del modelo (p.e. dependencias estáticas entre fuentes, dependencias de tiempo de ejecución entre ejecutables)

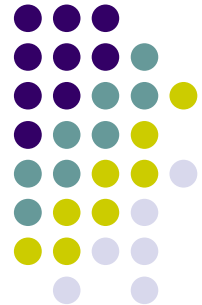
Implementación de una Colab.



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

Implementar la Estructura

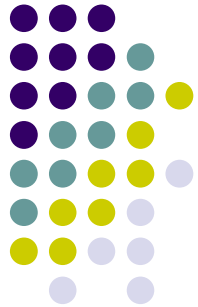
Implementar Clases



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

Implementar la Estructura

Implementar Clases (2)

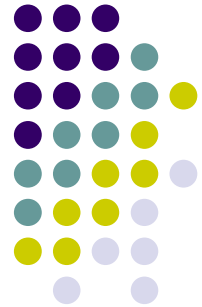


- Ejemplo en C++

```
class Empleado {
   private:
      String nombre;        //atributo
   public:
      Empleado();           //constructor por defecto
      Empleado(String);     //constructor comun
      ~Empleado();          //destructor
      String getNombre(); //operacion de acceso
      virtual float getPago() = 0; // op. abstracta
};
```

Implementar la Estructura

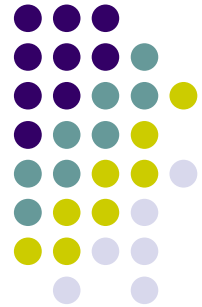
Implementar Interfaces



- Las interfaces también 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)



- Ejemplo en Java:

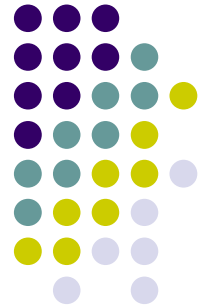
```
public interface IRetiroHandler {
    public void identificacion(int,String);
    public void seleccionarCuenta(int);
    ...
}
```

- Ejemplo en C++:

```
class IRetiroHandler {
    public:
        virtual void identificacion(int,String) = 0;
        virtual void seleccionarCuenta(int) = 0;
        ...
}
```

Implementar la Estructura

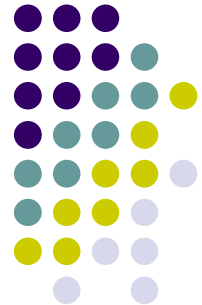
Implementar Relaciones



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

Implementar la Estructura

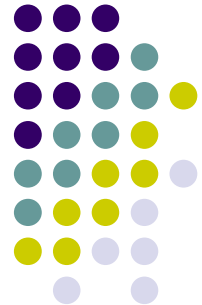
Relaciones – Generalizaciones



- 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 Jornaleo : public Empleado`
  - C#: `class Jornalero : Empleado`

Implementar la Estructura

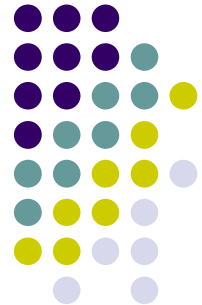
Relaciones – Realizaciones



- Las realizaciones se 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 ATM implements IRetiro`
  - C#: `class ATM : IRetiro`

Implementar la Estructura

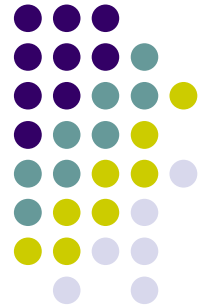
Relaciones – Asociaciones



- 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

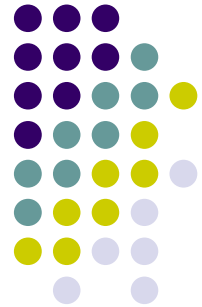
Implementar la Estructura

Relaciones – Asociaciones (2)

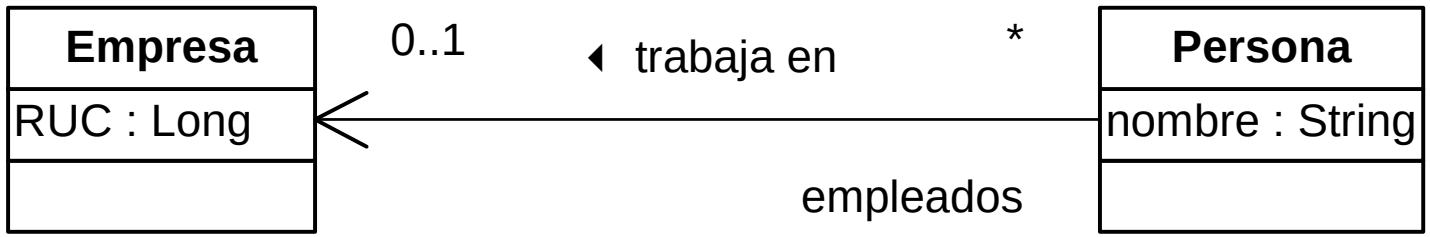


- Se define un pseudoatributo en A solamente si la asociación es navegable hacia B
- El tipo de un pseudoatributo para A depende de la clase B, pero también de la multiplicidad en el extremo de la asociación del lado de B
- Se distinguen dos casos dependiendo del máximo de dicha multiplicidad
  - El máximo es 1: el pseudoatributo es de tipo B
  - El máximo es mayor que 1 (típicamente *): el pseudoatributo es de tipo Coleccion(B)

Implementar la Estructura
Relaciones – Asociaciones (3)



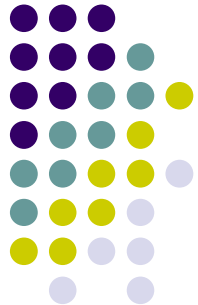
- Caso 1



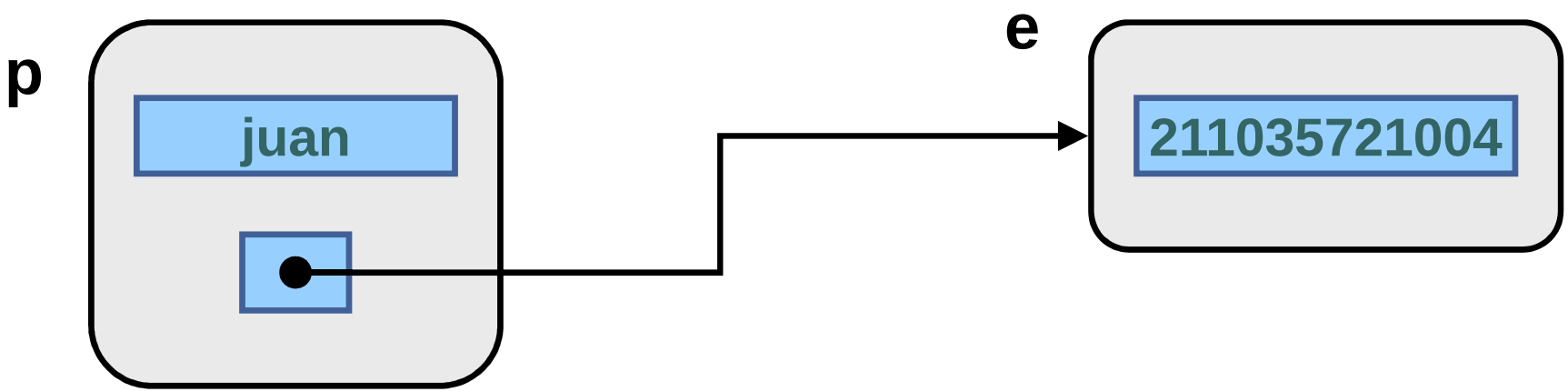
  - Ejemplo en C++

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

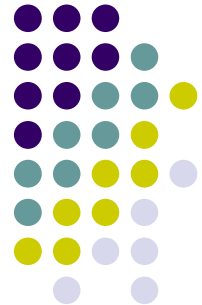
Implementar la Estructura
Relaciones – Asociaciones (4)



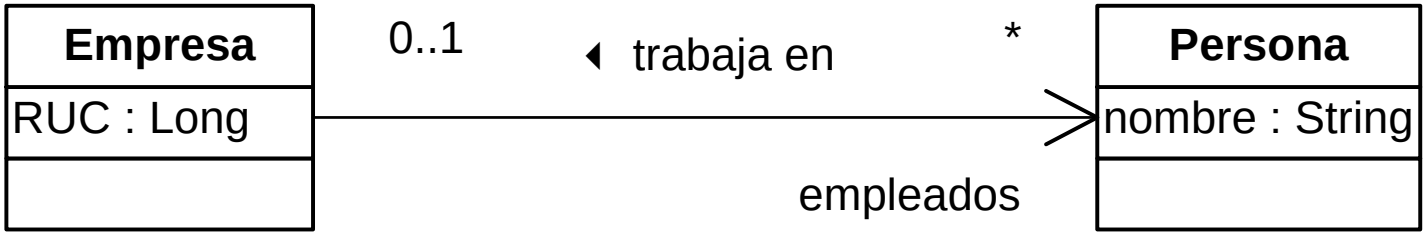
- Caso 1 (cont.)
  - Una persona puede tener una referencia a una empresa



Implementar la Estructura
Relaciones – Asociaciones (5)



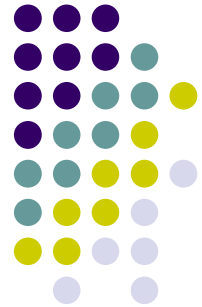
- Caso 2



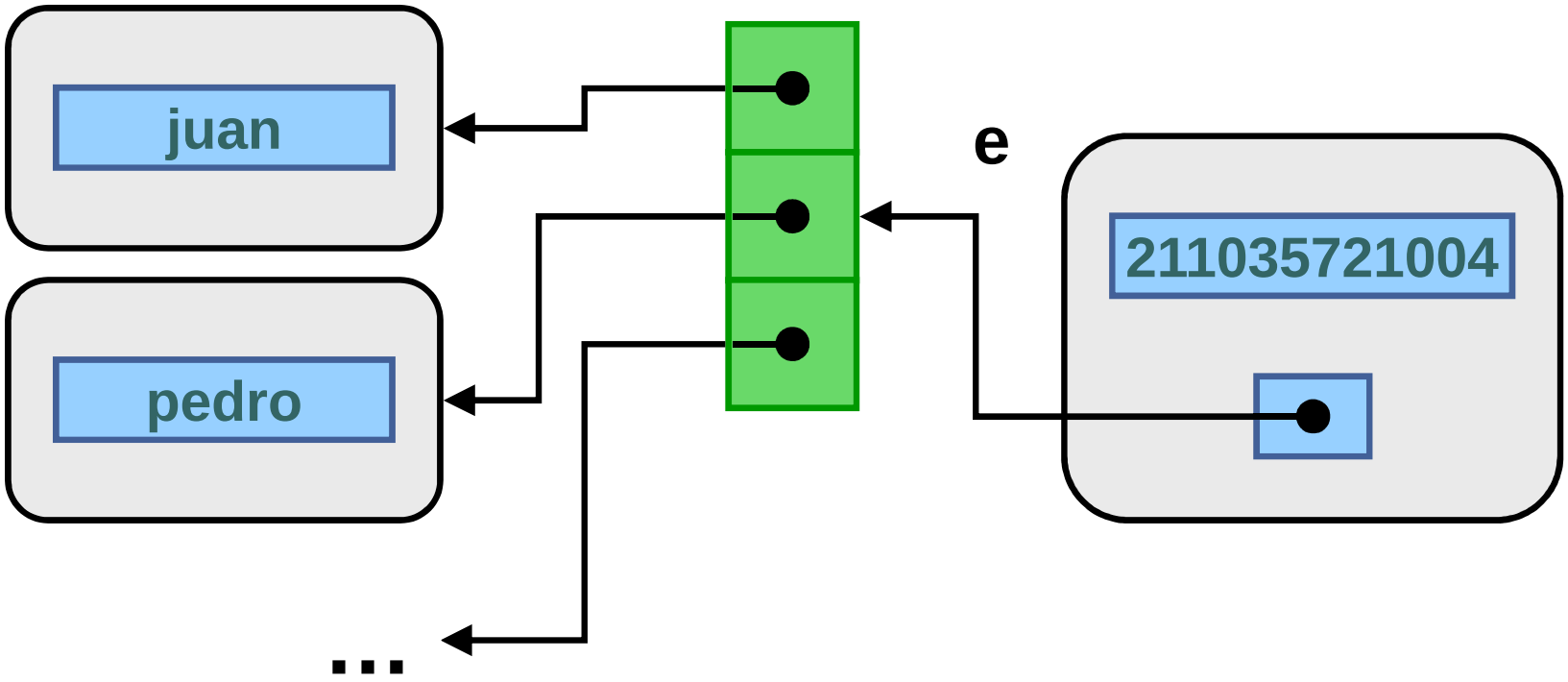
- Ejemplo en Java

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

Implementar la Estructura
Relaciones – Asociaciones (6)

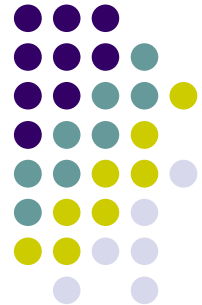


- Caso 2 (cont.)
  - Una empresa tiene una colección de referencias a personas



Implementar la Estructura

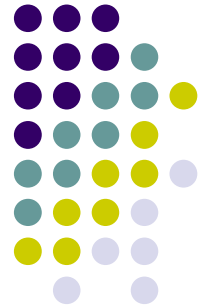
Relaciones – Asociaciones (7)



- 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 lineal con operaciones de acceso a los elementos por posición
  - Muchos ambientes de programación cuentan con bibliotecas de clases con diferentes tipos de colecciones predefinidas

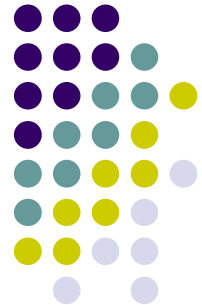
Implementar la Estructura

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` y en C# `using`
  - Sólo para elementos de otros paquetes de diseño

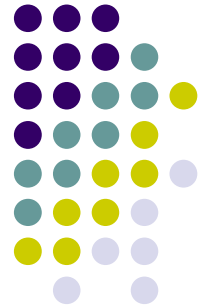
Implementar las Interacciones



- 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

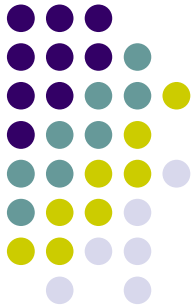
Implementar las Interacciones

Implementar Métodos

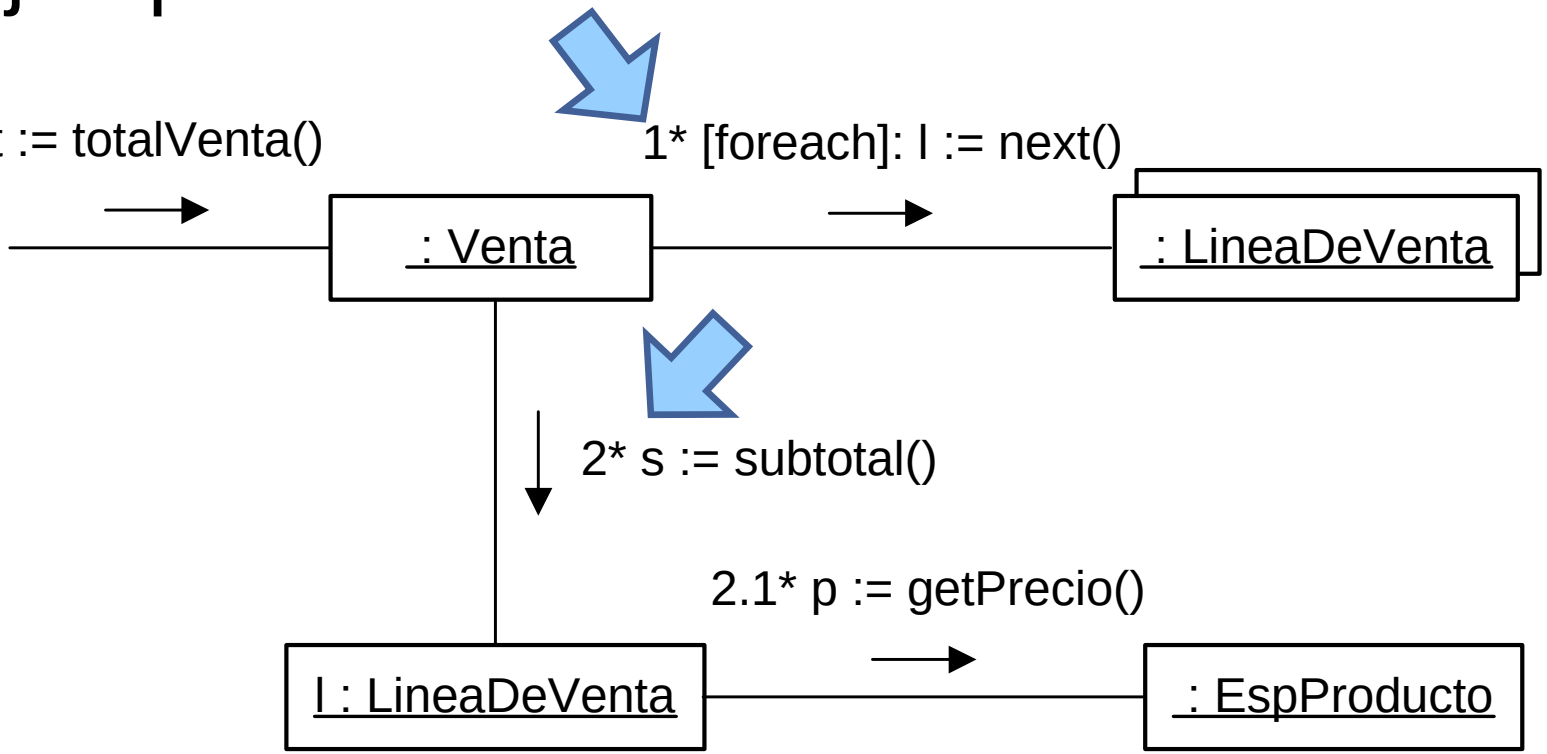


- Un diagrama de colaboración no tiene como objetivo servir de pseudocódigo
- Sin embargo generalmente ilustra la lógica general de varias operaciones
- Al implementar el método asociado a la operación `op()` en una clase A
  - Se busca un mensaje `op()` llegando a una instancia de A
  - La interacción anidada en ese mensaje debe ser de ayuda para implementar el método

Implementar las Interacciones
Implementar Métodos (2)



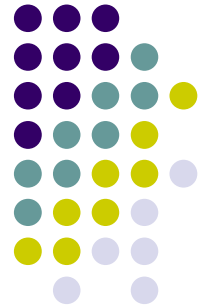
- Ejemplo



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

Implementar las Interacciones

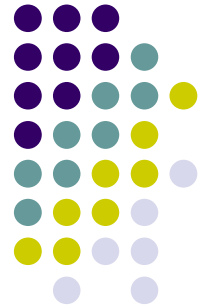
Implementar Métodos (2)



- Ejemplo (cont.)

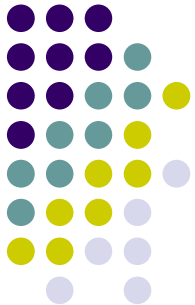
```
class Venta {
   public float totalVenta() {
      float total = 0;
      LineaDeVenta ldv;
      IteratorLineaDeVenta it = lineas.getIterator();

      while (it.hasCurrent()) {
         ldv = it.current();
         total = total + ldv.subtotal();
         it.next();
      }
      return total;
}
```

Sugerencias



- Antes de implementar una clase desde cero es recomendable considerar si código existente puede ser reusado 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 disponen de todos los elementos necesarios para su implementación
  - Esto permite que al terminar de implementar una clase se pueda testear inmediatamente

Sugerencias (3)