Programación Avanzada

Diseño

Diagramas de Comunicación

Contenido

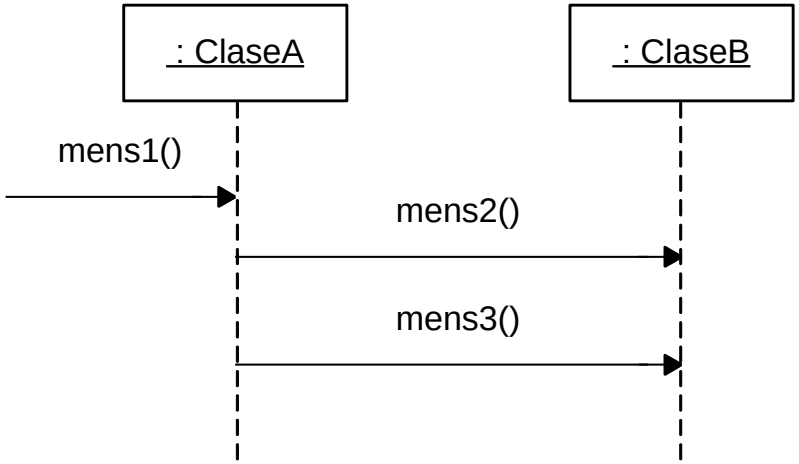
- Diagramas de Interacción
- Notación
- Reuso de Elementos de Diseño

Diagramas de Interacción

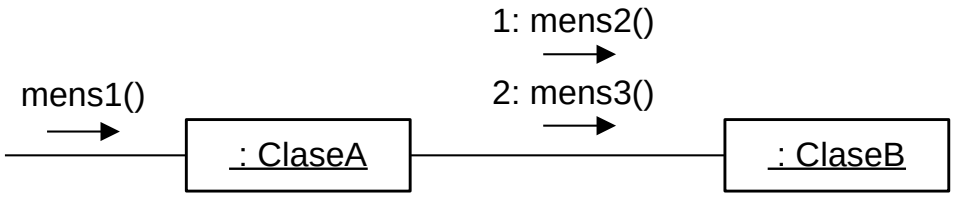
- UML incluye los **diagramas de interacción** que sirven para mostrar ejemplos de cómo ciertos objetos interactúan a través de mensajes para la realización de tareas
- Existen varios tipos de diagramas de interacción que son semánticamente equivalentes entre sí, en particular:
  - **Diagramas de Secuencia**
  - **Diagramas de Comunicación**

Diagramas de Interacción (2)

- Un Diagrama de Secuencia

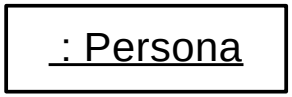


- Su Diagrama de Comunicación equivalente

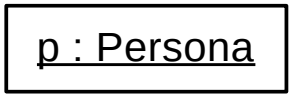


Notación
Instancias

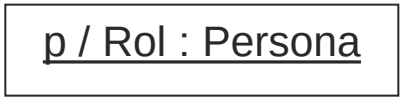
- Las instancias se representan igual que en los diagramas de instancias
- Corresponden a una instancia “cualquiera” de una cierta clase o interfaz (no a una instancia real)



Sin nombre



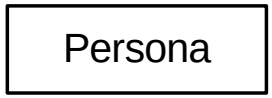
Con nombre



Cuando existen varias formas de acceder a esa instancia

Notación
Clases

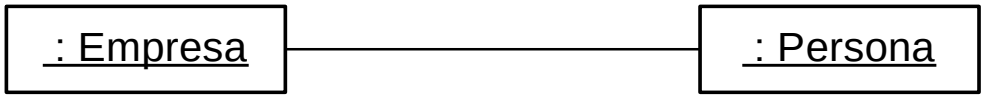
- Las clases se representan con el nombre de la clase dentro de un rectángulo
- Corresponden a una clase no a una instancia



Clase Persona

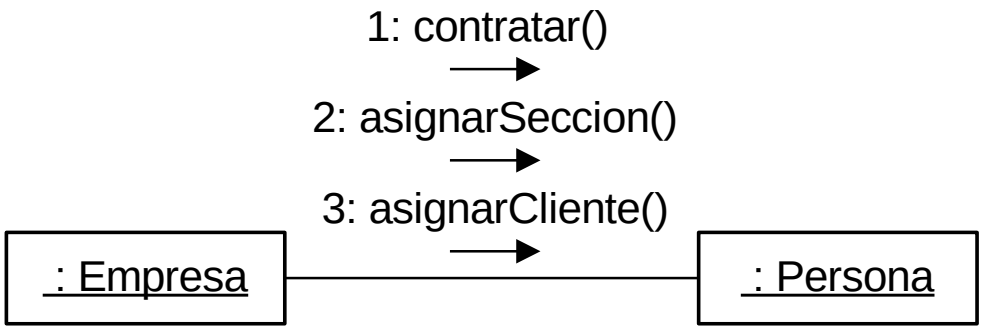
Notación
Links

- Representa una conexión entre instancias que indica navegabilidad y visibilidad entre ellas
- Establece una relación de cliente/servidor entre las instancias



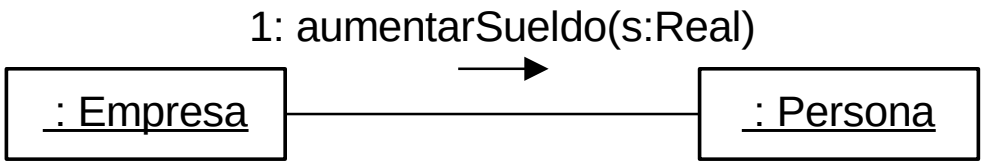
Notación
Mensajes

- Los mensajes son representados mediante una flecha etiquetada
- Un mensaje está asociado a un link y tiene asignado un número de secuencia que determina el orden de ocurrencia



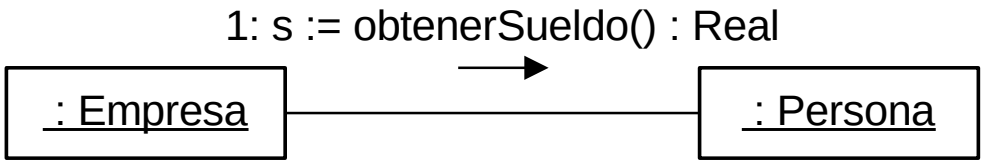
Notación
Parámetros

- Los parámetros se muestran entre paréntesis a la derecha del nombre del mensaje
- Se puede mostrar además su tipo



Notación
Tipo de Retorno

- El valor de retorno puede ser mostrado a la izquierda del mensaje, con un := en medio
- Se puede mostrar además el tipo del valor de retorno



Notación
Sintaxis de Mensajes

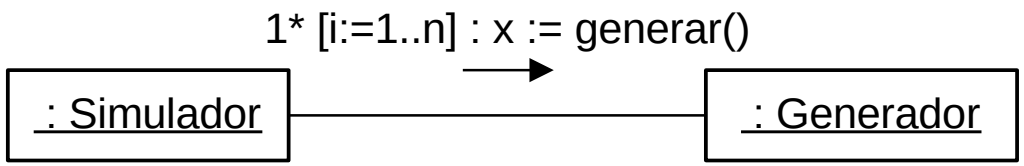
- La sintaxis de los mensajes es la siguiente:

```
[ret :=] mensaje([param [: TipoParam]]) [: TipoRet]
```

- Donde:
  - `ret` almacena el resultado de la operación (opcional)
  - `mensaje` es el nombre del mensaje enviado (y de la operación invocada)
  - `param` son argumentos usados en el envío
  - `TipoParam` es el tipo de cada parámetro (opcional)
  - `TipoRet` es el tipo del recorrido de la operación (opcional)

Notación
Iteración

- Las iteraciones se indican mediante un asterisco (*) a continuación del numero de secuencia del mensaje
- Esto expresa que el mensaje es enviado en forma repetida (en un loop) al receptor



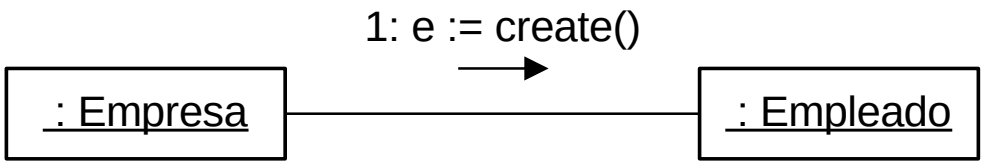

```
class Simulador {
  Generador gen;

  void unaOper() {
    for (i from 1 to n) {
      x = gen.generar();
    }
}
```

Notación

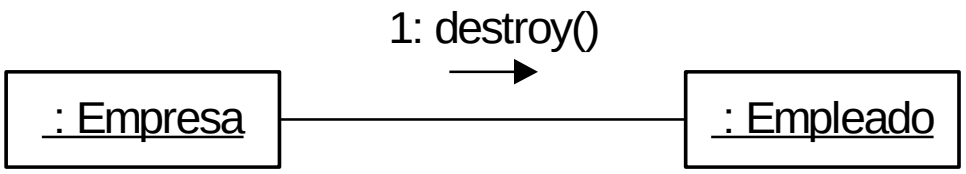
Creación de Instancias

- La forma de ilustrar la creación de una instancia es enviando el mensaje **create**
- Este mensaje puede incluir parámetros
- Lo usual es especificar un nombre para la instancia para poder utilizarla después



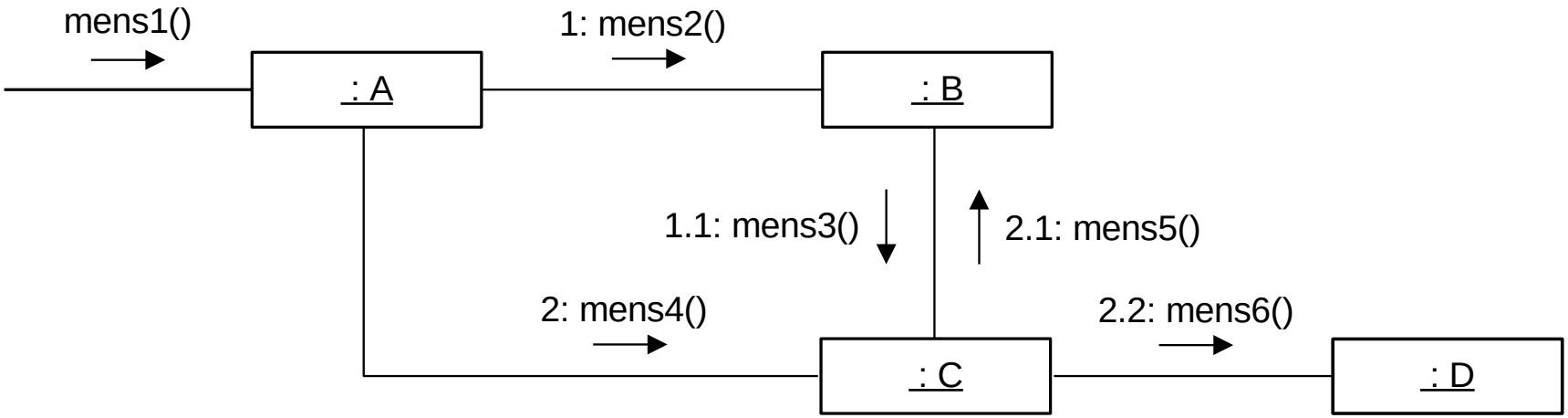
Notación
Destrucción de Instancias

- La forma de ilustrar explícitamente la destrucción de una instancia es enviando el mensaje **destroy**
- Previamente, debe eliminarse todo link que exista con esa instancia

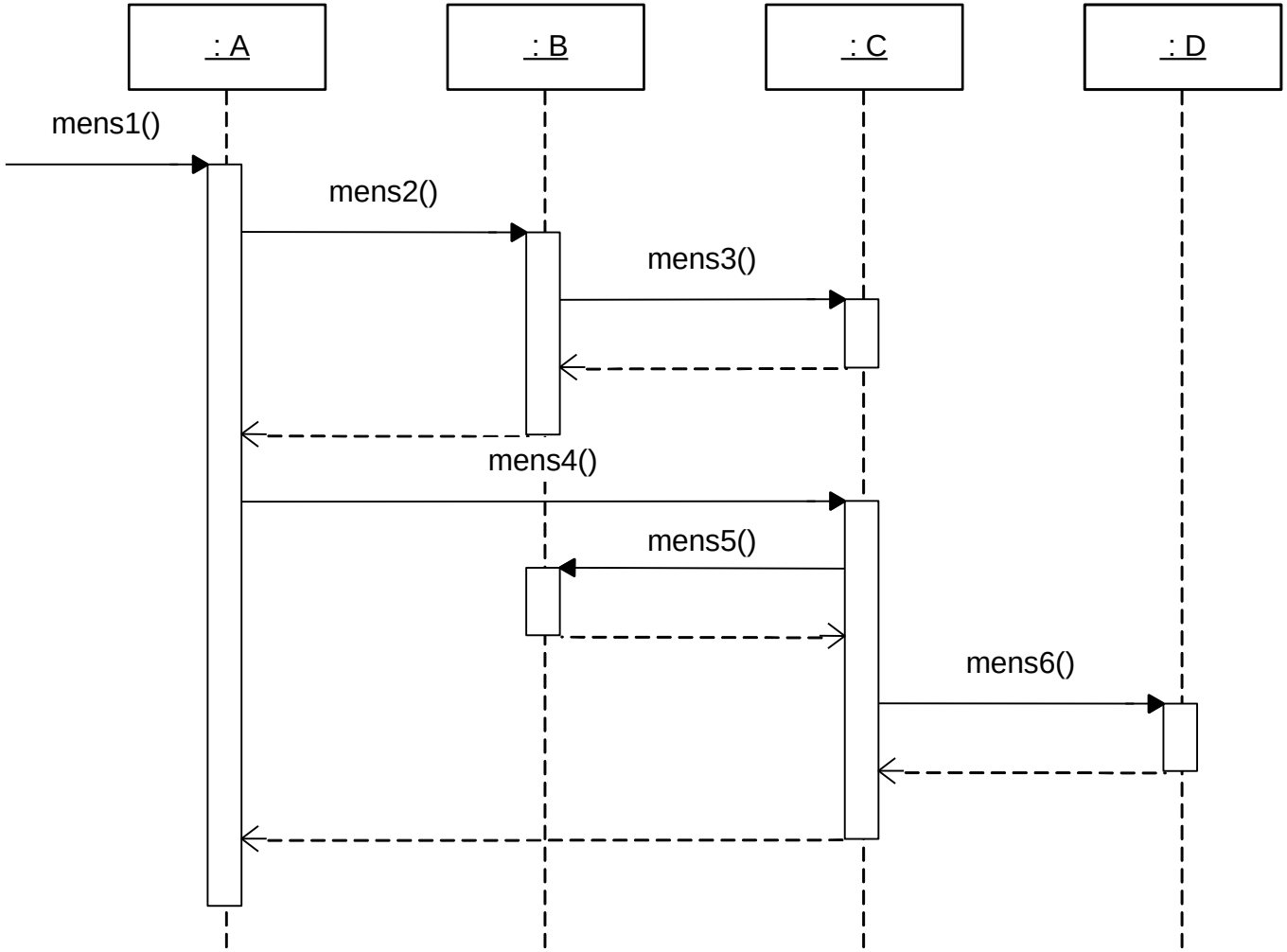


Notación
Números de Secuencia

- El orden de ocurrencia de los mensajes viene dado por los números de secuencia
- El mensaje que inicia la interacción generalmente no es numerado

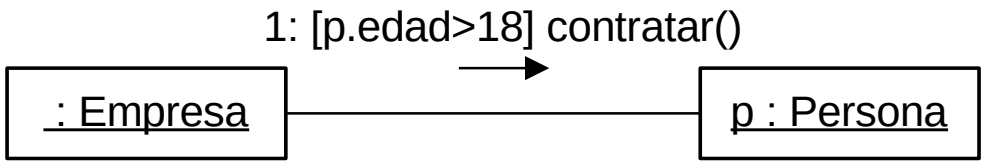


Notación
Números de Secuencia (2)



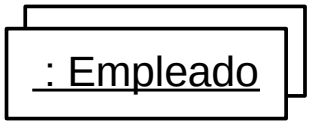
Notación
Mensajes Condicionales

- Un mensaje condicional es enviado únicamente si su guarda es satisfecha
- La guarda se muestra entre paréntesis rectos ([ ]) a la izquierda del mensaje



Notación
Colecciones

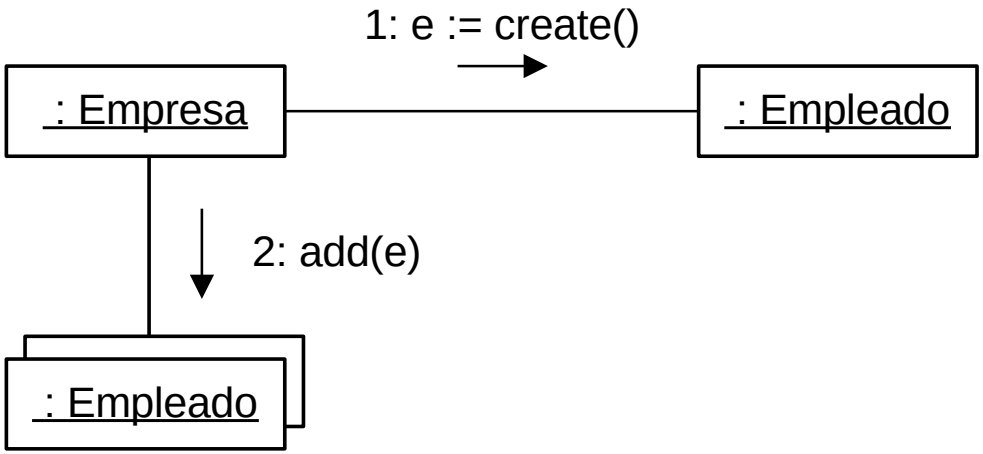
- Los multiobjetos de los diagramas de interacción representan una colección de objetos de una cierta clase



Colección de instancias
de la clase Empleado

Notación
Mensajes a Colecciones

- Un mensaje a una colección representa un mensaje al objeto colección mismo
- No un broadcast a todos los elementos contenidos en él



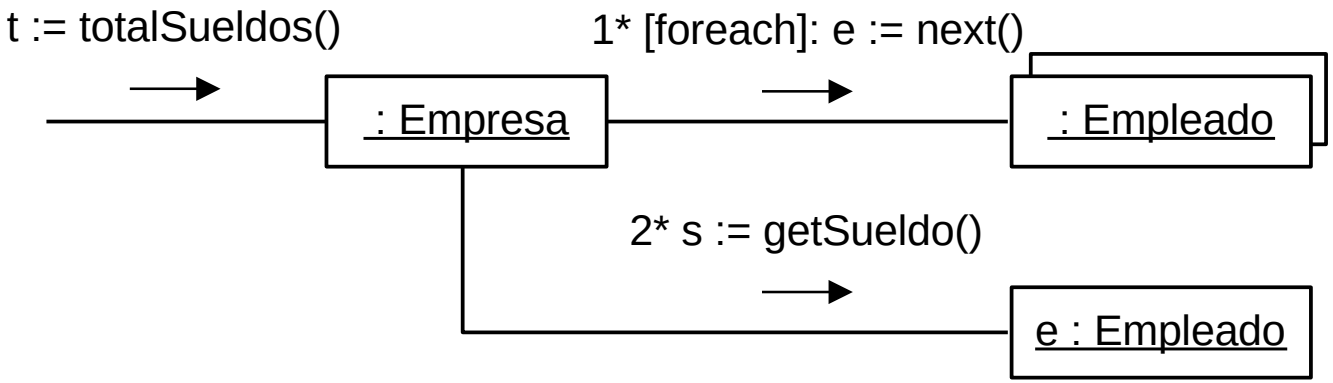
Notación
Responsabilidad de Colecciones

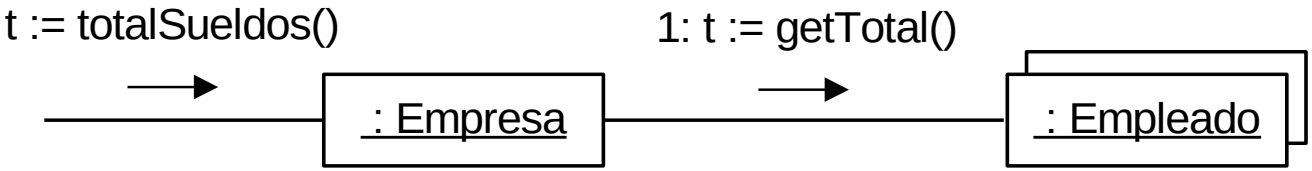
- Las colecciones serán tratadas como meros contenedores de objetos por lo que no tendrán otra responsabilidad más que esa
- Proveerán solamente operaciones que permitan administrar los objetos contenidos
- En general las interfaces de **Diccionario** (add, remove, find, member, etc.) e **Iterador** (next, etc.) son suficientes para las colecciones

Notación
Responsabilidad de Colecciones (2)

- `add(o:Tipo)` Agrega la instancia **`o`** a la colección del tipo **`Tipo`**
- `remove(o:Tipo)` Remueve la instancia **`o`** de la colección del tipo **`Tipo`**. No elimina la instancia
- `find(c:Clave):Tipo` Retorna la instancia con clave **`c`** de tipo **`Clave`**
- `member(o:Tipo):Boolean` Devuelve un booleano indicando si la instancia **`o`** de tipo **`Tipo`** existe o no en la colección
- `next():Tipo` Devuelve el próximo elemento en la colección. Se supone que la colección está ordenada

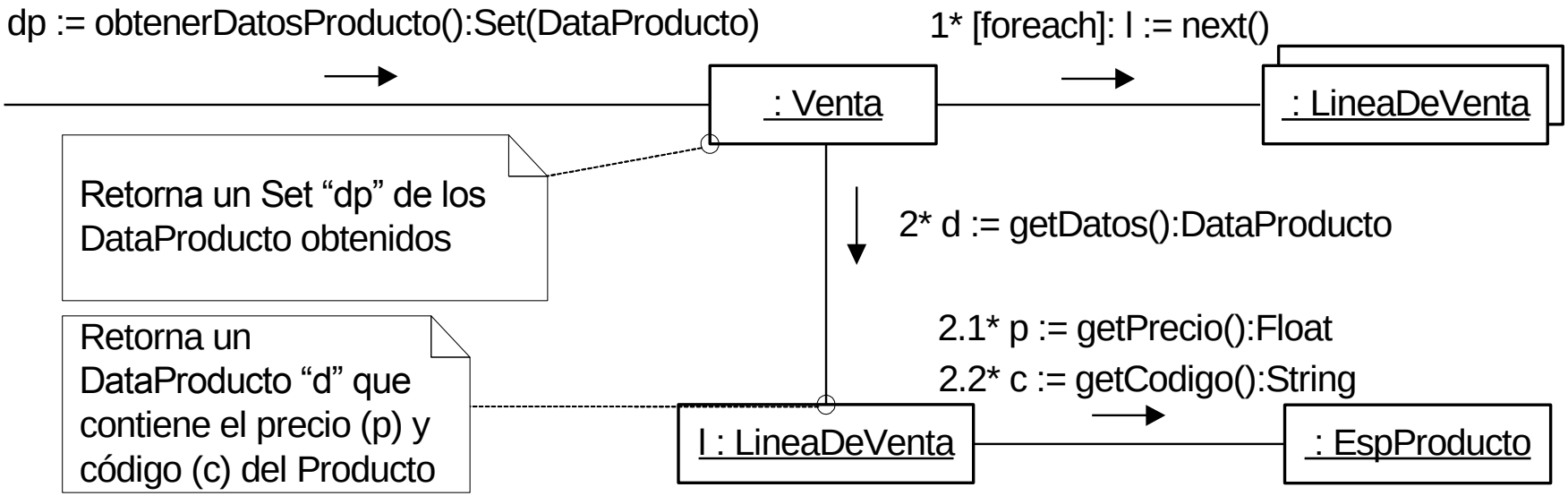
Notación
Resp. de Colecciones - Ejemplo





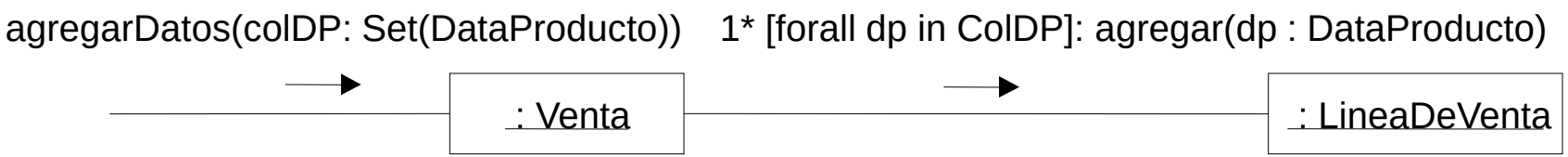
Notación
Datatypes

- El procesamiento de datatypes (construcción, envío de mensajes) no se muestra gráficamente: se utilizan notas

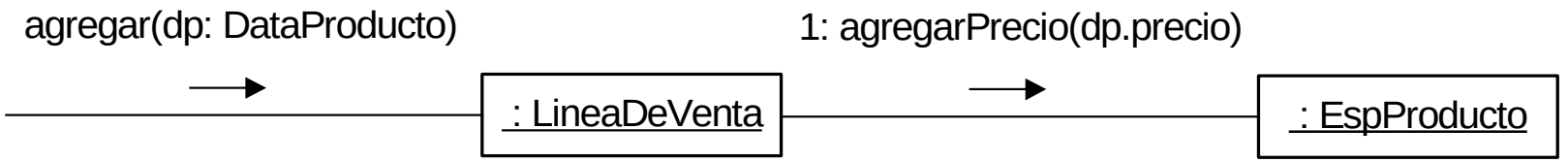


Notación
Datatypes (2)

- Es posible iterar sobre los elementos de una colección de datatypes: *forall dt in ColDT*



- Es posible acceder a los elementos de un datatype utilizando el operador “.”



Reuso de Elementos de Diseño

- Se busca reutilizar los elementos de diseño generados de una iteración a otra
  - En particular: clases, operaciones y atributos
- Esto apunta a generar iterativamente el diseño y no “reinventar la rueda” cada vez
- El diseño debe ser consistente de una iteración a otra. Es decir, si un elemento de diseño cambia, no puede quedar información inconsistente en otra parte del diseño

Diagramas de Comunicación
Errores Comunes

- Suponer la existencia de links nunca generados
- Enviar un mensaje a un multiobjeto que implique el procesamiento con todos los objetos contenidos en él
- No especificar qué sucede con mensajes que aparentan ser triviales
- Representar datatypes como instancias