

3

Construcción de la Solución



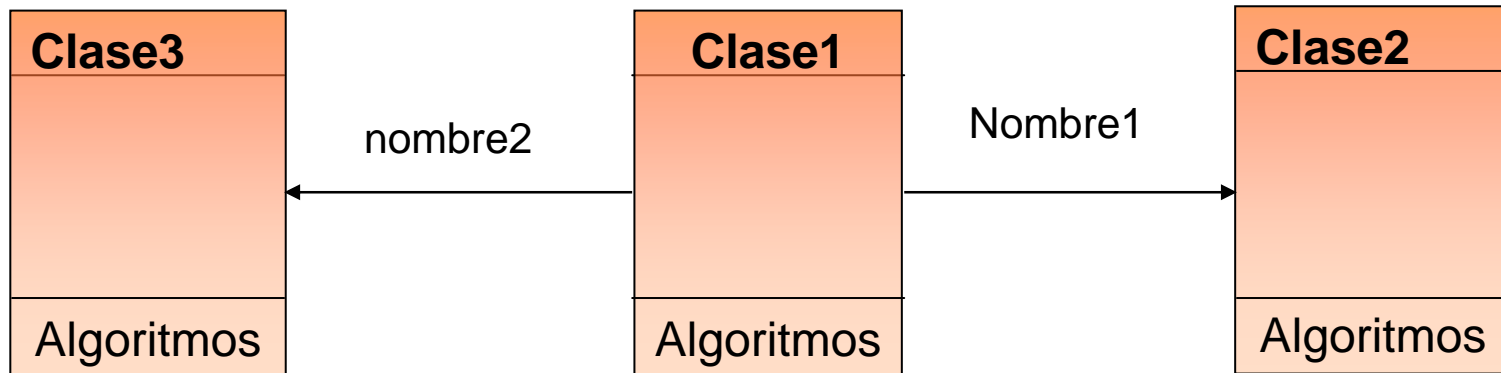
Clases y Objetos

Qué es una Clase?

- Son los elementos que definen la estructura de un programa.
- Son entidades del mundo del problema.

En un programa completo, los algoritmos están dentro de las clases, y ellas son las que establecen la manera en que los algoritmos colaboran para resolver el problema global

Programa

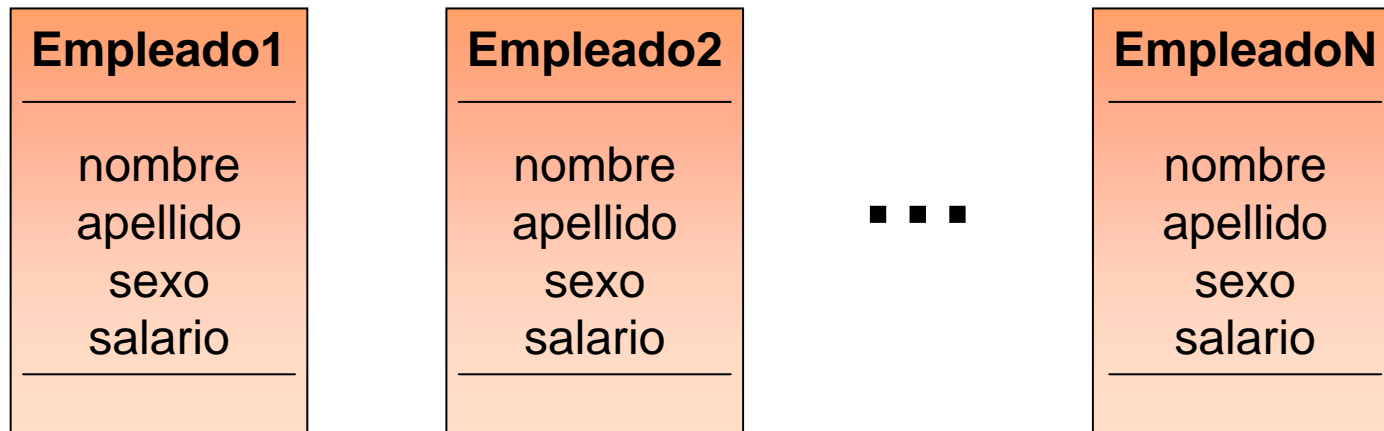


Clases y Objetos

- Objeto

Por cada clase hay una entidad del mundo del problema. Qué pasa si hay varias “instancias” de una de las entidades?

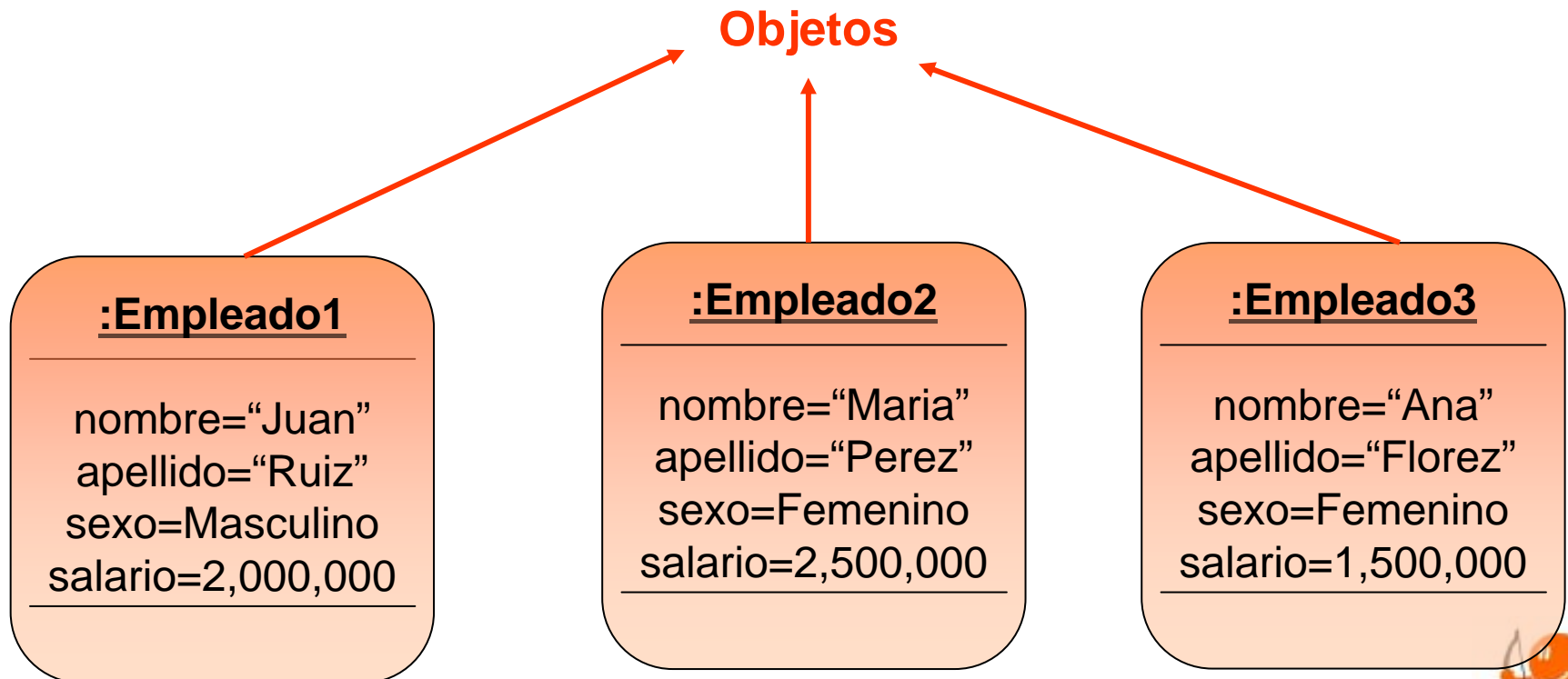
Pensemos por ejemplo en el programa para manejar “UN” empleado de la clase pasada... Si nos pidieran ahora un programa para manejar todos los empleados de una empresa



Clases y Objetos

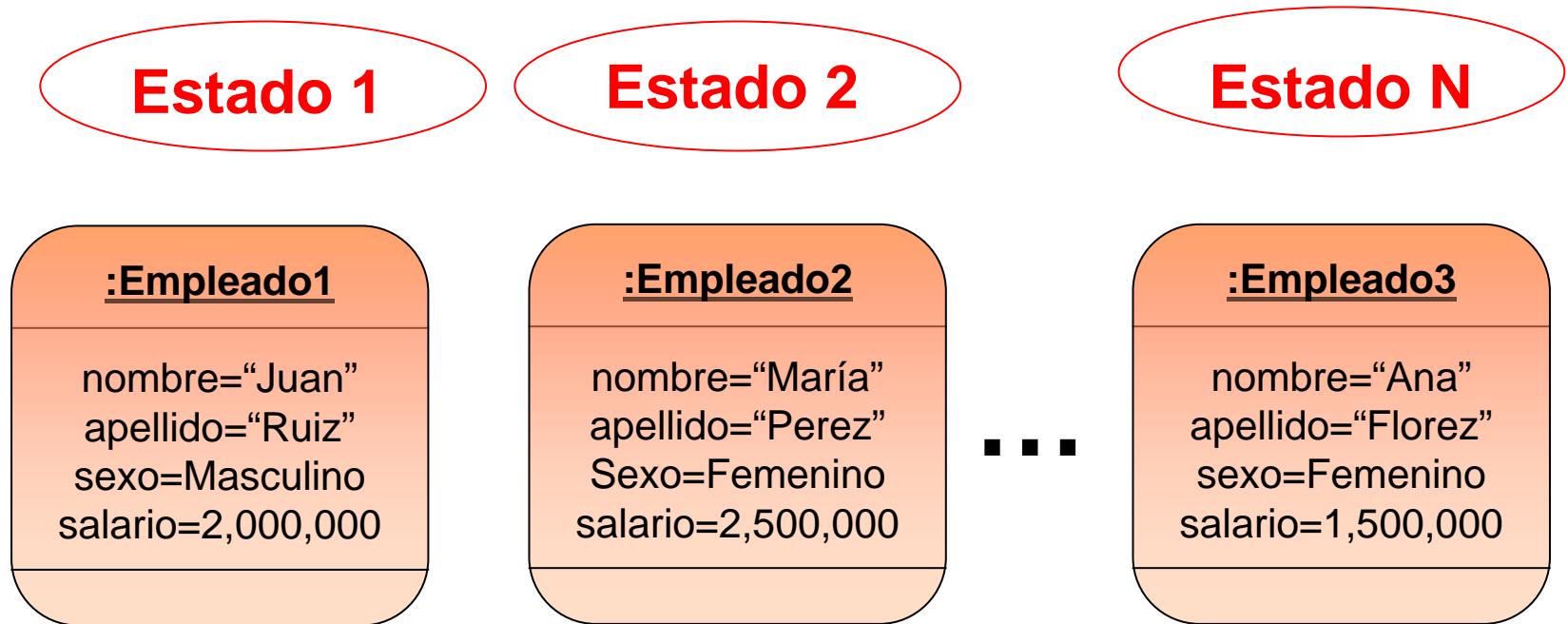
Aunque todos los empleados tienen las mismas características (Nombre, Apellido, etc.), cada uno tiene valores distintos para ellas (cada uno va a tener un nombre y apellido diferente).

Aquí aparece el concepto de OBJETO, una instancia de una clase que tiene sus propios valores para cada uno de los atributos y los define



Clases y Objetos

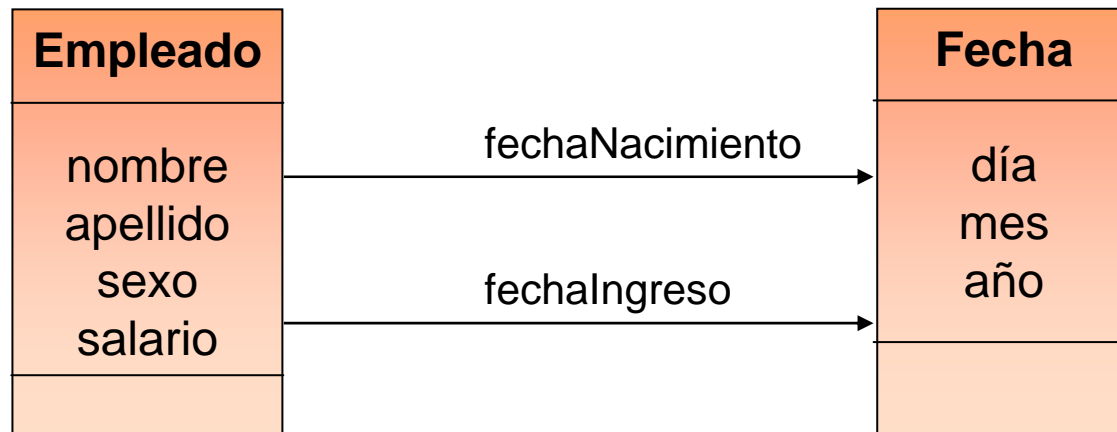
El conjunto de valores de los atributos se denomina el **ESTADO DEL OBJETO**.



Para diferenciar las clases de los objetos, se puede decir que **una clase define un tipo de elemento del mundo**, mientras que **un objeto representa un elemento individual**



Diagrama de Clases



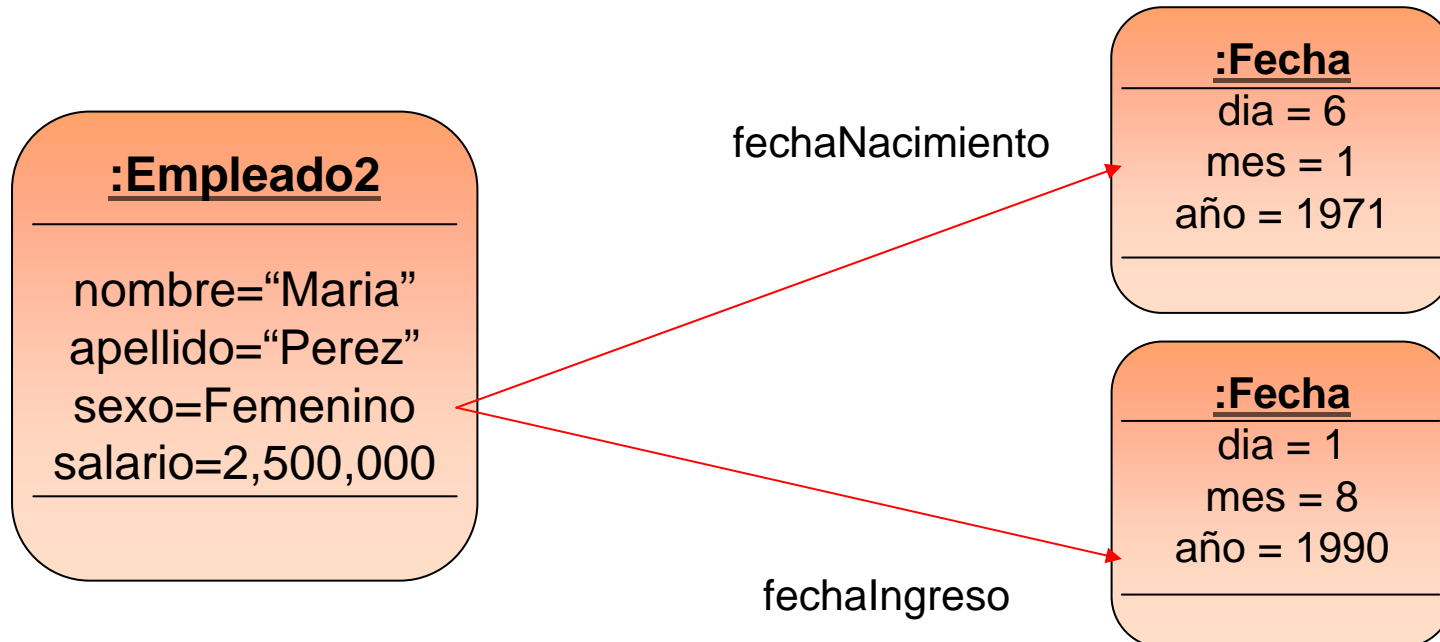
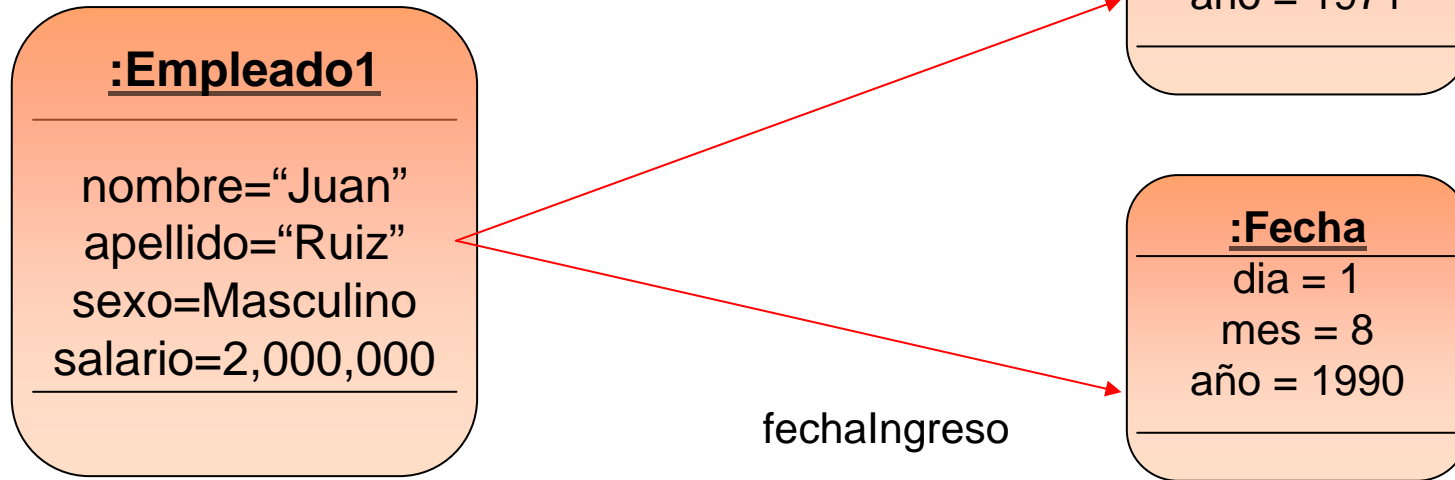
Cada objeto de la clase Empleado tendrá un valor para cada uno de sus atributos y un objeto para cada asociación

Cada Empleado será representado con tres objetos de tipo:

- Clase Empleado
- Clase Fecha
- Clase Fecha



Diagrama de Objetos



Tipos de instrucciones

- Instrucción de retorno
 - Para devolver un resultado como solución del problema puntual
 - Se representa con un “return”

```
public int darNumeroLlamadas( )  
{  
    return numeroLlamadas;  
}
```



Tipos de instrucciones

- Instrucción de llamada (o invocación) de un Método
 - Para usar métodos de la misma clase
 - Para usar métodos de un objeto de otra clase con el cual existe una asociación.

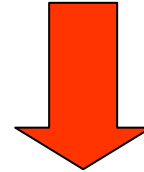


Invocación de un método de la misma clase

- Se hace para construir métodos complejos a partir de métodos mas simples que ya están escritos.

Empleado
nombre apellido sexo salario
<code>int calcularSalarioAnual()</code> <code>int calcularImpuesto()</code>

Ejemplo: calcular el monto de los impuestos que debe pagar el empleado en un año. Los impuestos se calculan como el 19.5% del total de salarios recibidos en un año



Vamos a descomponer el problema en dos métodos:

- Cálculo del valor total del **salario anual**
- Cálculo del **monto del impuesto**, que usa el método anterior



Ejemplo de invocación de un método de la misma clase

```
public class Empleado  
{
```

```
...
```

```
public int calcularSalarioAnual( )  
{  
    return(salario * 12 );  
}
```

```
public int calcularImpuesto( )  
{  
    return( calcularSalarioAnual( ) * 19.5 / 100 );  
}
```

```
}
```

Empleado
nombre apellido sexo salario
int calcularSalarioAnual() int calcularImpuesto()

Llamado al método calcularSalarioAnual

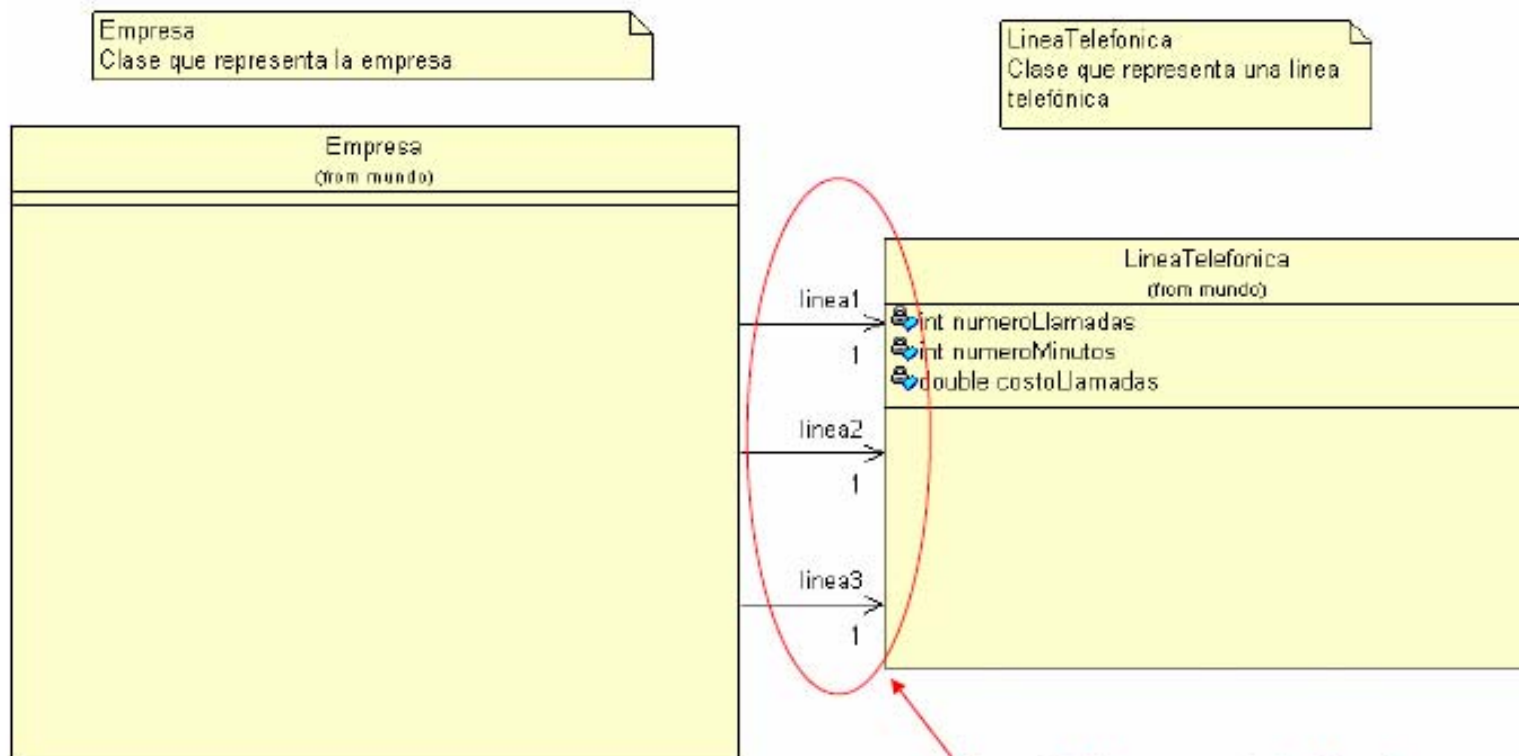


Invocación de un método de un objeto de otra clase con el cual existe una asociación

- Se hace cuando se necesita obtener o modificar alguna información de un objeto de otra clase con el cual existe una asociación.



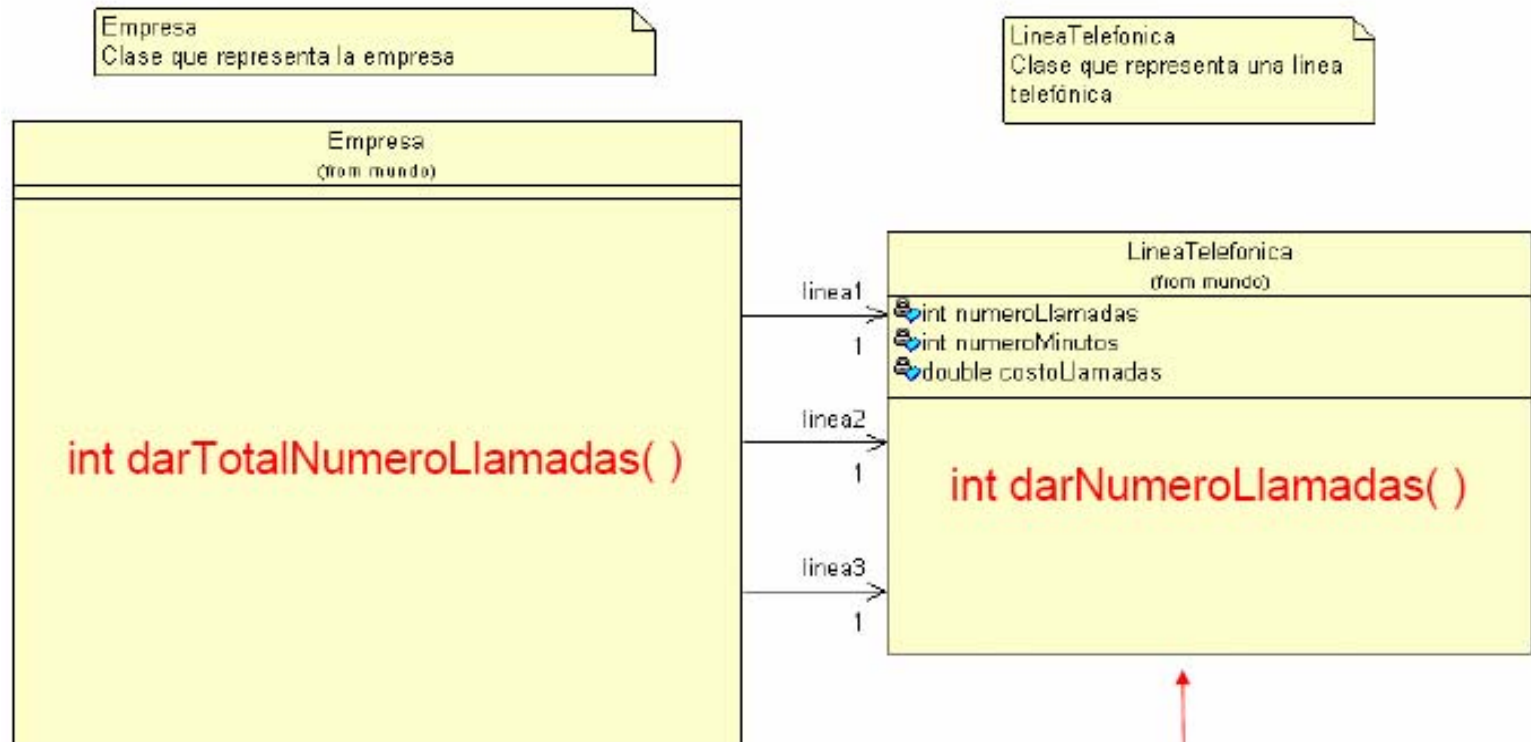
Ejemplo



Hay 3 líneas telefónicas
(asociaciones)



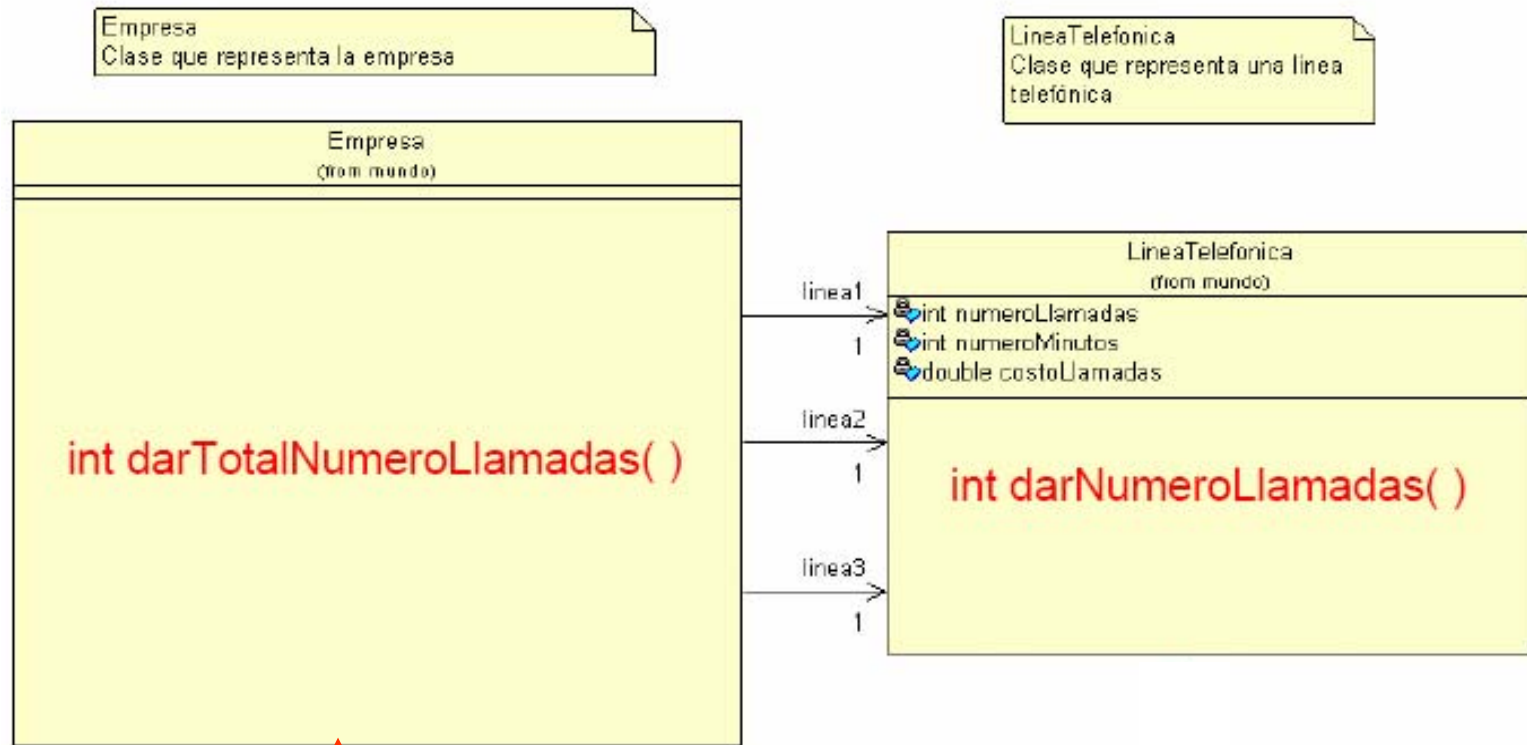
Ejemplo



En la clase LineaTelefonica existe el método darNumeroLlamadas



Ejemplo



El método darTotalNumeroLlamadas de la clase Empresa DEBE invocar el método darNumeroLlamadas de la clase LineaTelefonica



Ejemplo

LineaTelefonica
int numeroLlamadas int numeroMinutos double costoLlamadas
int darNumeroLlamadas()

```
public class LineaTelefonica
{
    ...
    public int darNumeroLlamadas( )
    {
        return( );
    }
}
```



Ejemplo

Empresa
<code>int darNumeroLlamadas()</code>

- El número total de llamadas de la empresa es la suma del número de llamadas de la linea1 + el número de llamadas de la linea2 + el número de llamadas de la linea3



Ejemplo

public class Empresa

{

...

public int darTotalNumeroLlamadas()

{

return(linea1.darNumeroLlamadas() +
linea2.darNumeroLlamadas() +
linea3.darNumeroLlamadas());

}

}

Empresa

int darNumeroLlamadas()



Ejemplo

```
public class Empresa
```

```
{
```

```
...
```

```
public int darTotalNumeroLlamadas( )
```

```
{
```

```
return( linea1.darNumeroLlamadas( ) +
```

```
        linea2.darNumeroLlamadas( ) +
```

```
        linea3.darNumeroLlamadas( ));
```

```
}
```

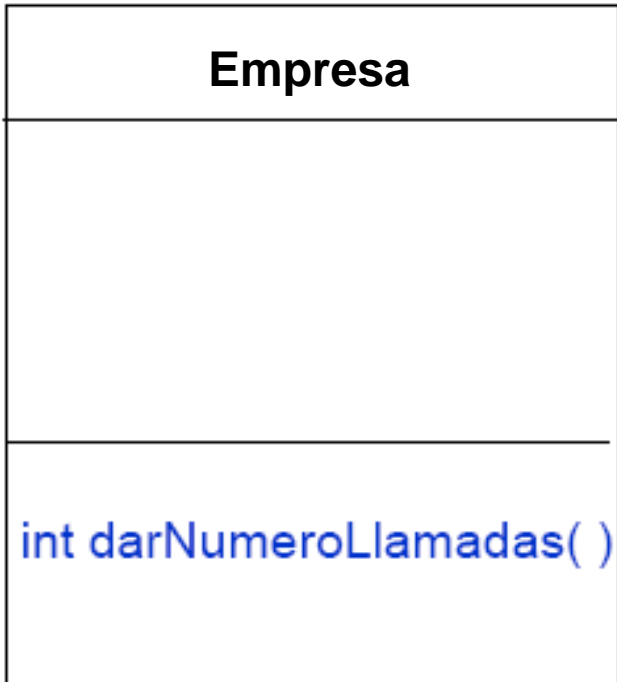
```
}
```

**Nombre de la
Asociación**

punto

**Nombre del
método de la
asociación**

**Parámetros
(ninguno en
este caso)**



Ejemplo con parámetros

```
Public class LineaTelefonica
{
    ...
    public void agregarLlamadaLocal( int minutos )
    {
        numeroLlamadas= numeroLlamadas+ 1;
        numeroMinutos= numeroMinutos+ minutos;
        costoLlamadas= costoLlamadas+ ( minutos * 35 );
    }
}
```



Ejemplo con parámetros

```
Public class Empresa
```

```
{
```

```
    public void agregarLlamadaLocalLinea1( int minutos)
```

```
    {
```

```
        linea1.agregarLlamadaLocal( minutos);
```

```
    }
```

```
}
```

Nombre de la
Asociación

Nombre del Método
de la Asociación

Parámetros



Llamando métodos con parámetros

- Cuándo necesita parámetros un método?

- Cuando la información que tiene el objeto en sus atributos no es suficiente para resolver el problema

- Cómo se declara un parámetro?

- En la signature del método se define el tipo del dato del parámetro y se le asocia un nombre

- Cómo se utiliza el valor de un parámetro

- Basta con utilizar el nombre del parámetro en el cuerpo del método de la misma manera que se utilizan los atributos



Llamando métodos con parámetros

- Se puede utilizar el parámetro por fuera del cuerpo del método?
 - Aquel que hace la llamada del método, cómo hace para definir los valores de los parámetros ?
- NO, en ningún caso
 - En el momento de hacer la llamada, se deben pasar tantos valores como parámetros está esperando el método. Esos valores pueden ser constantes (por ejemplo, 500), atributos del objeto que hace la llamada (por ejemplo, costoLlamadas), parámetros del método desde el cual se hace la llamada (por ejemplo, minutos) o expresiones que mezclen los tres anteriores (por ejemplo, costoLlamadas* minutos * 500)



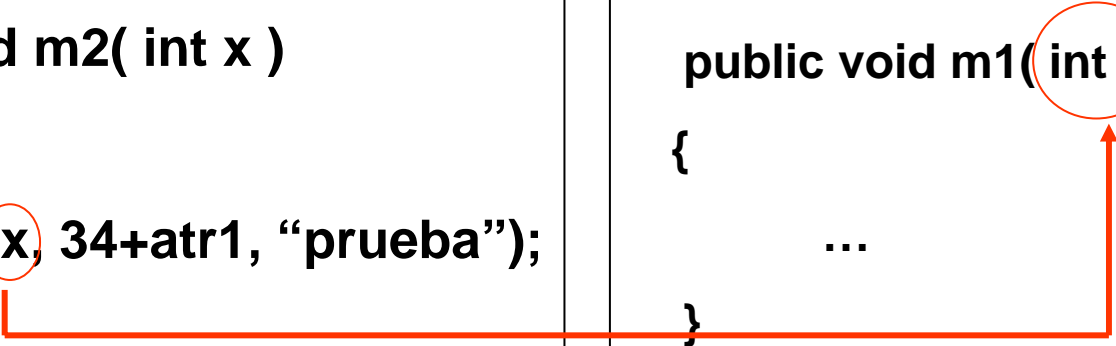
Llamando métodos con parámetros

■ Cómo se hace la relación entre esos valores y los parámetros?

• Los valores se deben pasar teniendo en cuenta el orden en el que se declararon los parámetros

```
public class C2
{
    private int atr1;
    private C1 obj;
    public void m2( int x )
    {
        obj.m1( x, 34+atr1, "prueba");
    }
}
```

```
public class C1
{
    ...
    public void m1( int a, int b, String c )
    {
        ...
    }
}
```



Llamando métodos con parámetros

■ Cómo se hace la relación entre esos valores y los parámetros?

• Los valores se deben pasar teniendo en cuenta el orden en el que se declararon los parámetros

```
public class C2
{
    private int atr1;
    private C1 obj;
    public void m2( int x )
    {
        obj.m1( x, 34+atr1, "prueba");
    }
}
```

```
public class C1
{
    ...
    public void m1( int a, int b, String c )
    {
        ...
    }
}
```



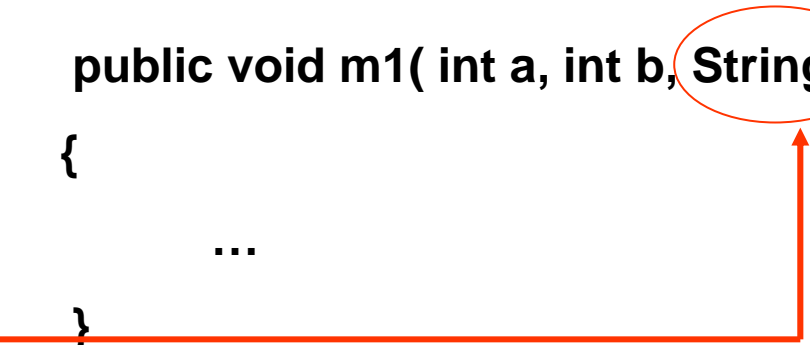
Llamando métodos con parámetros

■ Cómo se hace la relación entre esos valores y los parámetros?

• Los valores se deben pasar teniendo en cuenta el orden en el que se declararon los parámetros

```
public class C2
{
    private int atr1;
    private C1 obj;
    public void m2( int x )
    {
        obj.m1( x, 34+atr1, "prueba");
    }
}
```

```
public class C1
{
    ...
    public void m1( int a, int b, String c )
    {
        ...
    }
}
```



Diseño de la Solución



Entradas y Salidas de la etapa de Diseño



Salidas de la etapa de Diseño

Interfaz de Usuario

Es la parte de la solución que permite que los usuarios interactúen con el Programa. A través de ésta, el usuario puede utilizar las operaciones del programa que implementan los requerimientos Funcionales.

Arquitectura de la Solución

Diseño de alto nivel en el que aparecen a grandes rasgos los elementos que conforman la solución.

Diseño de las Clases

El objetivo es mostrar los detalles de cada una de las clases que van a hacer parte del programa. Para esto se utiliza el Diagrama de Clases de UML.



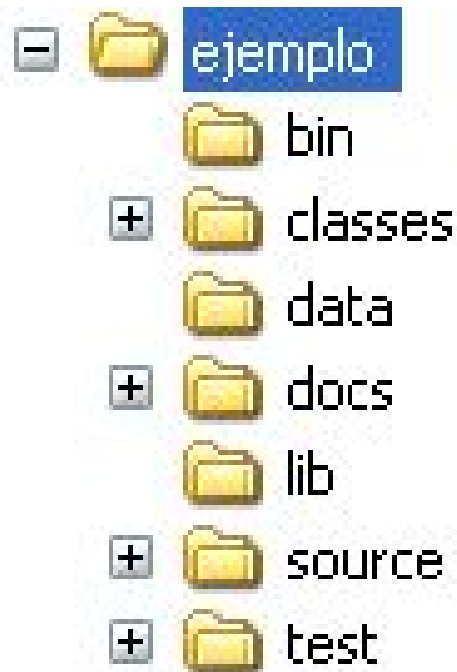
Construcción de la Solución

Visión Global



Estructura de directorios

Un ejemplo tiene la siguiente estructura de directorios:



Todos estos directorios deben estar presentes aún si no tienen contenido. A continuación se describe el propósito de cada directorio.




Estructura de directorios

Directorio	Contenido
bin	Comandos ejecutables que permiten la realización de las diferentes tareas de interacción del ejemplo. Estos ejecutables están pensados para la interacción con el ejemplo por fuera de eclipse.
classes	Clases compiladas del ejemplo. La estructura interna corresponde a la estructura de paquetes utilizada para el ejemplo.
data	Donde se guardan los datos persistentes o de muestra para la ejecución del ejemplo, en caso de haberlos.
docs	Documentación del ejemplo. Incluye la descripción general del ejemplo, su especificación, modelaje y diseño.
lib	Reunión de librerías (jars) necesarias para la compilación y ejecución del ejemplo. También incluye el empaquetado del ejemplo mismo.
source	Código fuente del ejemplo. La estructura interna corresponde a la estructura de paquetes utilizada.
test	Todos los elementos relacionados con las pruebas del ejemplo



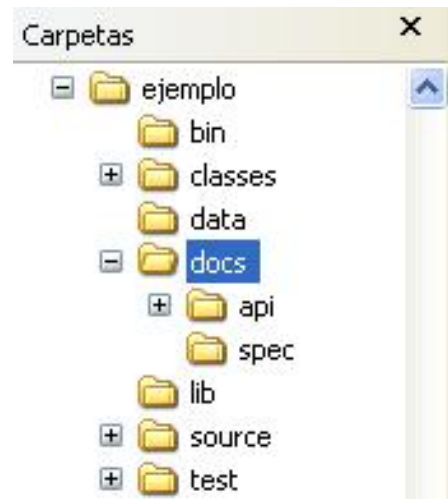
Directorio bin

bin	
clean.bat	Limpia los archivos generados en la construcción (build) del ejemplo
build.bat	Compila el código fuente del ejemplo y lo empaqueta en un jar, colocándolo en el directorio lib
run.bat	Ejecuta el ejemplo con el jar que se creo en la etapa de construcción
cleanTest.bat	Limpia los archivos generados en la construcción (buildTest) de las pruebas del ejemplo
buildTest.bat	Compila el código fuente de las pruebas del ejemplo y lo empaqueta en un jar, colocándolo en el directorio test/lib
runTest.bat	Ejecuta las pruebas y presenta la pantalla gráfica de resultados de junit
doc.bat	Genera la documentación javadoc del ejemplo y la coloca en el directorio docs/api



Directorio docs

docs



api



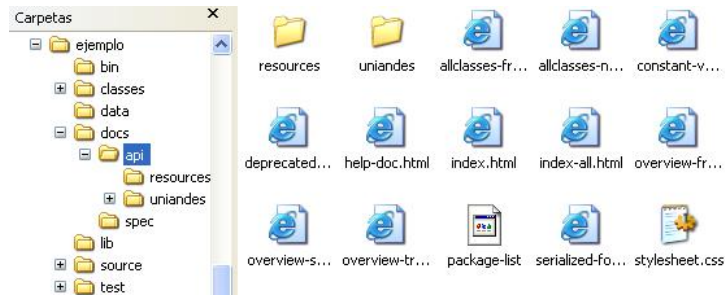
spec

Contiene la documentación del ejemplo. Se divide en dos subdirectorios: spec y api



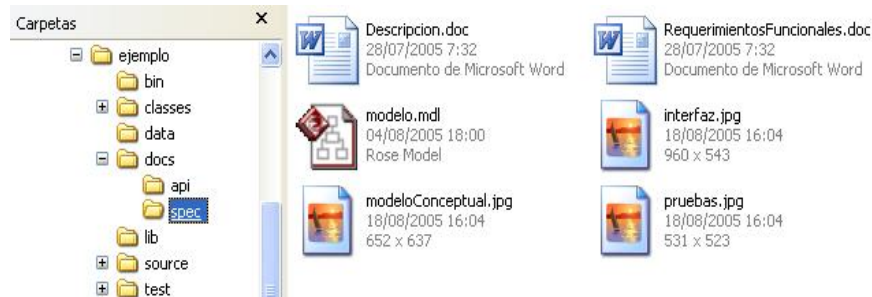
Directorio docs

api



Donde se alojará la documentación generada con javadoc para el ejemplo. El directorio api debe estar presente pero la documentación se debe generar cuando se necesite con el bat indicado y no se guarda en el repositorio.

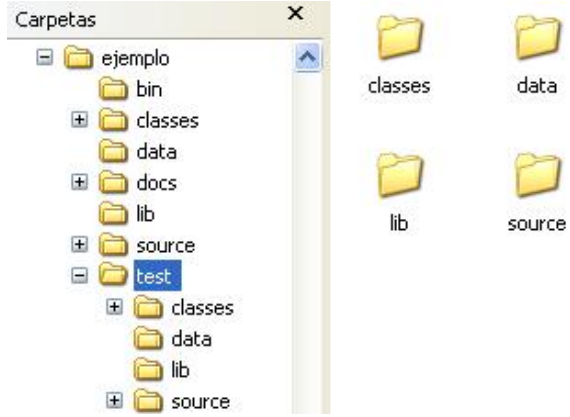
spec



Donde encontramos los documentos: Descripcion.doc, RequerimientosFuncionales.doc y Modelo.mdl. Además se tienen tres imágenes (jpg) de los diagramas contruidos en Modelo.mdl con el fin de facilitar su consulta.



Directorio test

test	
En test encontramos una estructura interna de directorios con elementos similares a la de la raíz del ejemplo	
source	Contiene el código fuente de las clases que implementan las pruebas.
classes	Contiene las clases de pruebas compiladas.
lib	Contiene las librerías necesarias para la compilación y ejecución de las pruebas, incluye el empaquetamiento de las clases de pruebas.
data	Contiene los datos utilizados en la ejecución de las pruebas

