

**UNIVERSIDAD
CENFOTEC**

**PROGRAMACIÓN
ORIENTADA A OBJETOS**

**Bachillerato en ingeniería de
software**

Universidad Cenfotec



**universidad
cenfotec_**
tecnologías digitales

Temas

- Jerarquía de objetos
- Enlace-Asociación
- Agregación-Composición

Jerarquías de objetos

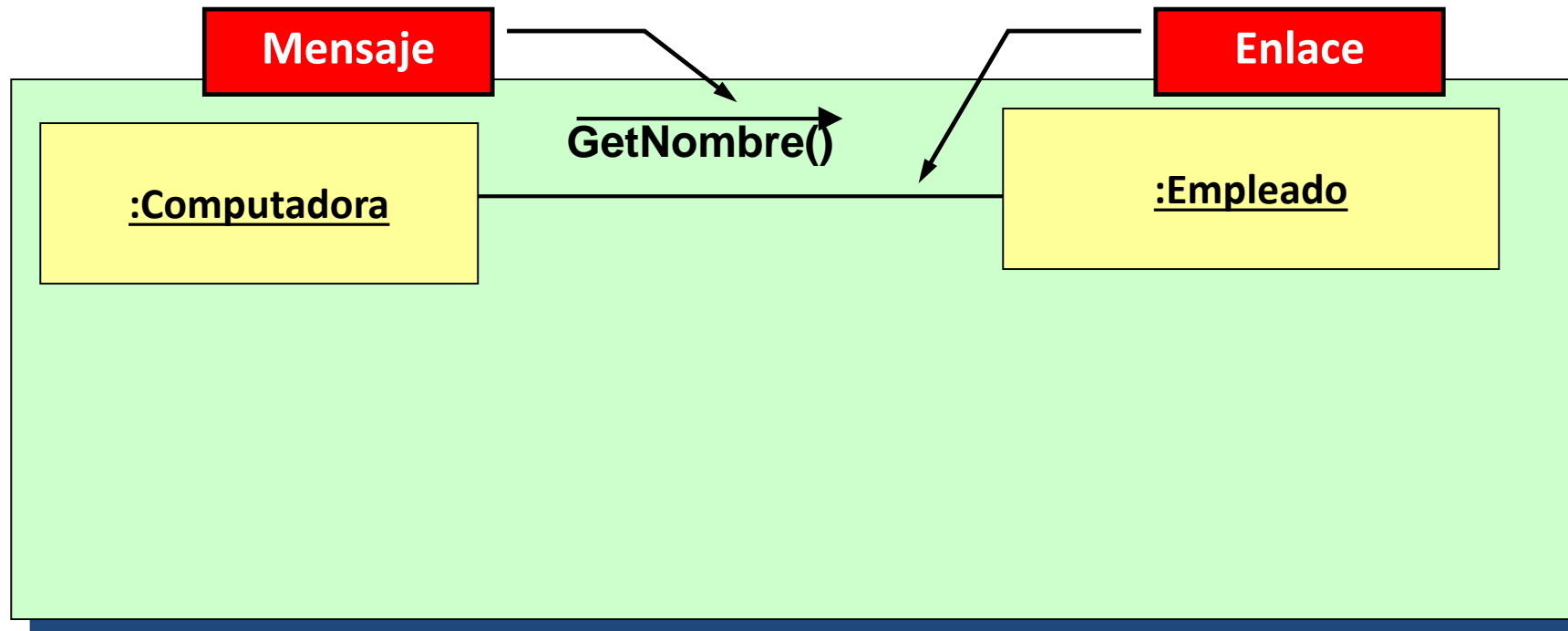
Jerarquía

- Uno de los conceptos más importantes en la orientación a objetos es el de jerarquía
- Establecer una jerarquía permite clasificar u organizar las abstracciones que forman parte de un problema.
- Las dos jerarquías más importantes en un sistema complejo son
 - Estructura de objetos (jerarquía de partes)
 - Estructura de clases (jerarquía de clases)

Jerarquía de objetos: Enlace-Asociación

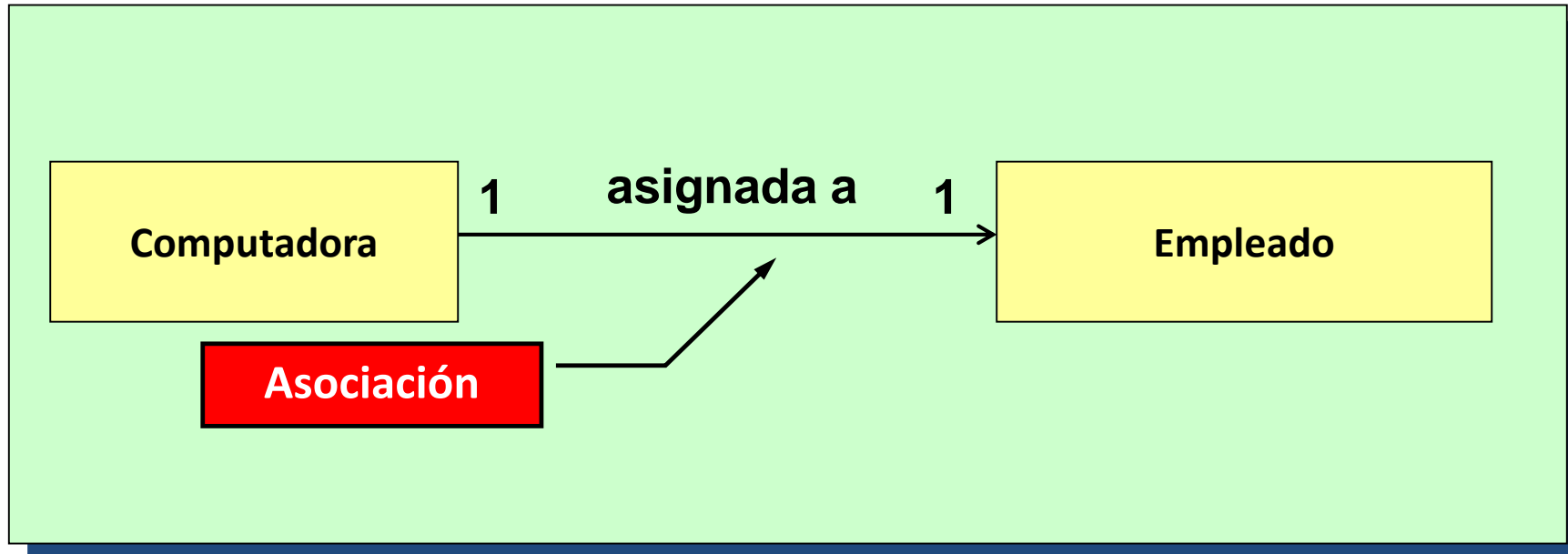
Enlace - Asociación

- Un enlace denota la asociación específica por la cual un objeto (el cliente) utiliza los servicios de otro objeto (el servidor), o a través de la cual un objeto puede comunicarse con otro.



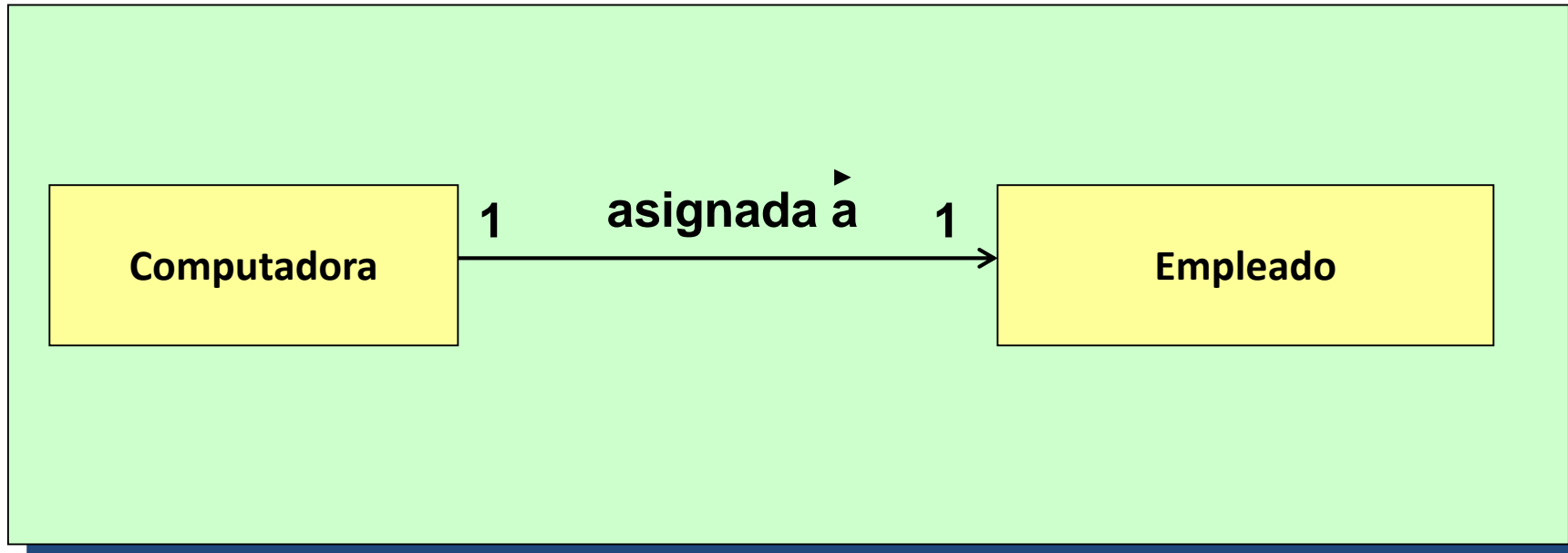
Enlace - Asociación

- La asociación es una relación de uso en la cuál dos elementos están ligados el uno al otro. Es una relación estructural, en la que la clase A usa una instancia de la clase B o de la clase C, y viceversa. Esta relación implica que se puede navegar de una clase A hacia una clase B



Enlace - Asociación

- Tanto los enlaces como las asociaciones se representan mediante arcos, generalmente etiquetados con el nombre de la asociación.
- Es importante que las asociaciones se pueden leer en cualquier dirección.



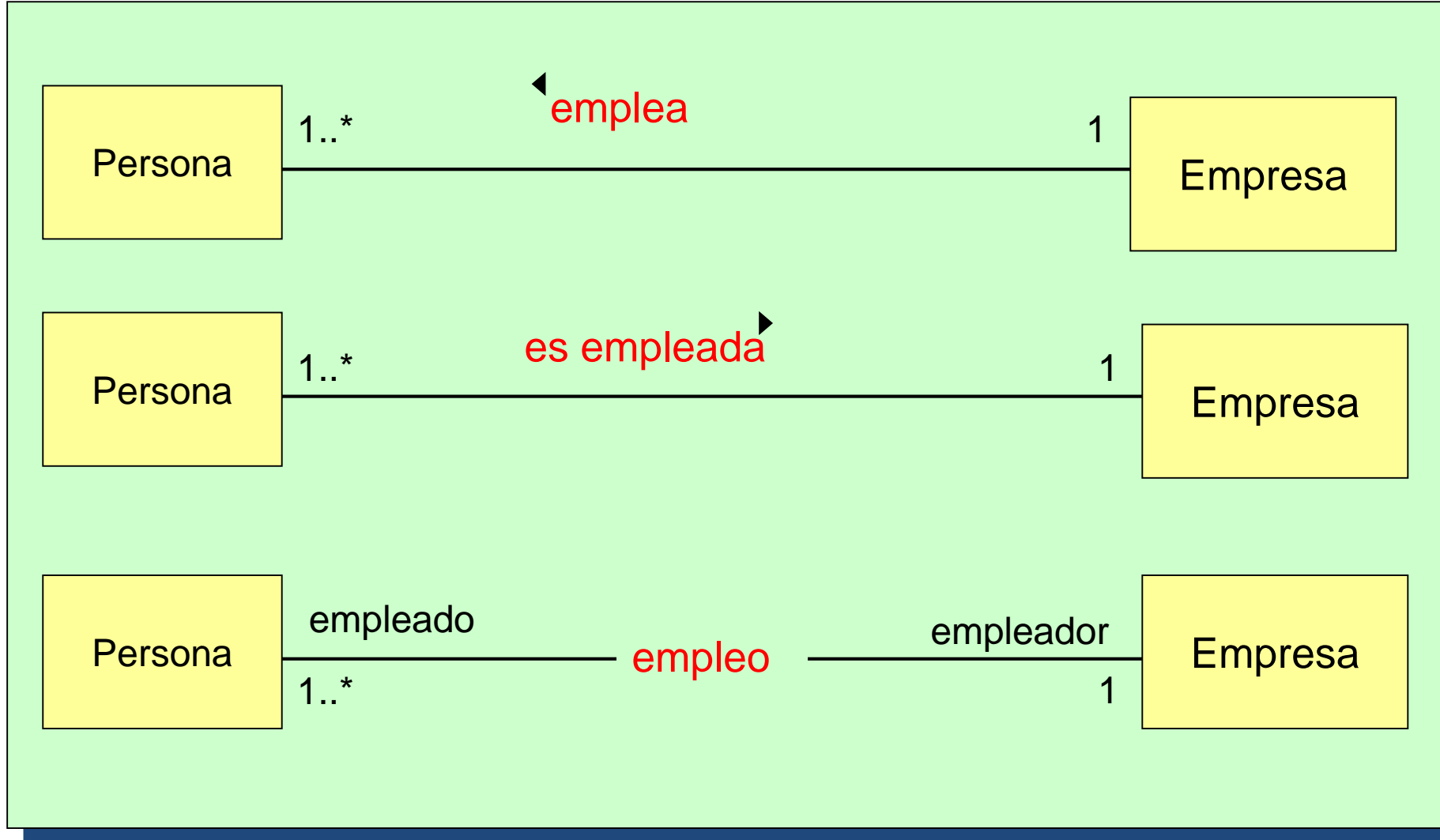
Asociación

- Una asociación puede ser nombrada de tres formas:
- Leyéndola “de izquierda a derecha”:
 - Una persona **es empleada por** una empresa
- Leyéndola “de derecha a izquierda”:
 - Una empresa **emplea a** una persona .

Asociación

- Una asociación se representa en UML por una línea que une los dos conceptos.
- Los roles pueden indicarse en cada extremo de la línea.
- La multiplicidad se indica en cada extremo: uno (1), muchos (*), uno o mas (1..*), etc.

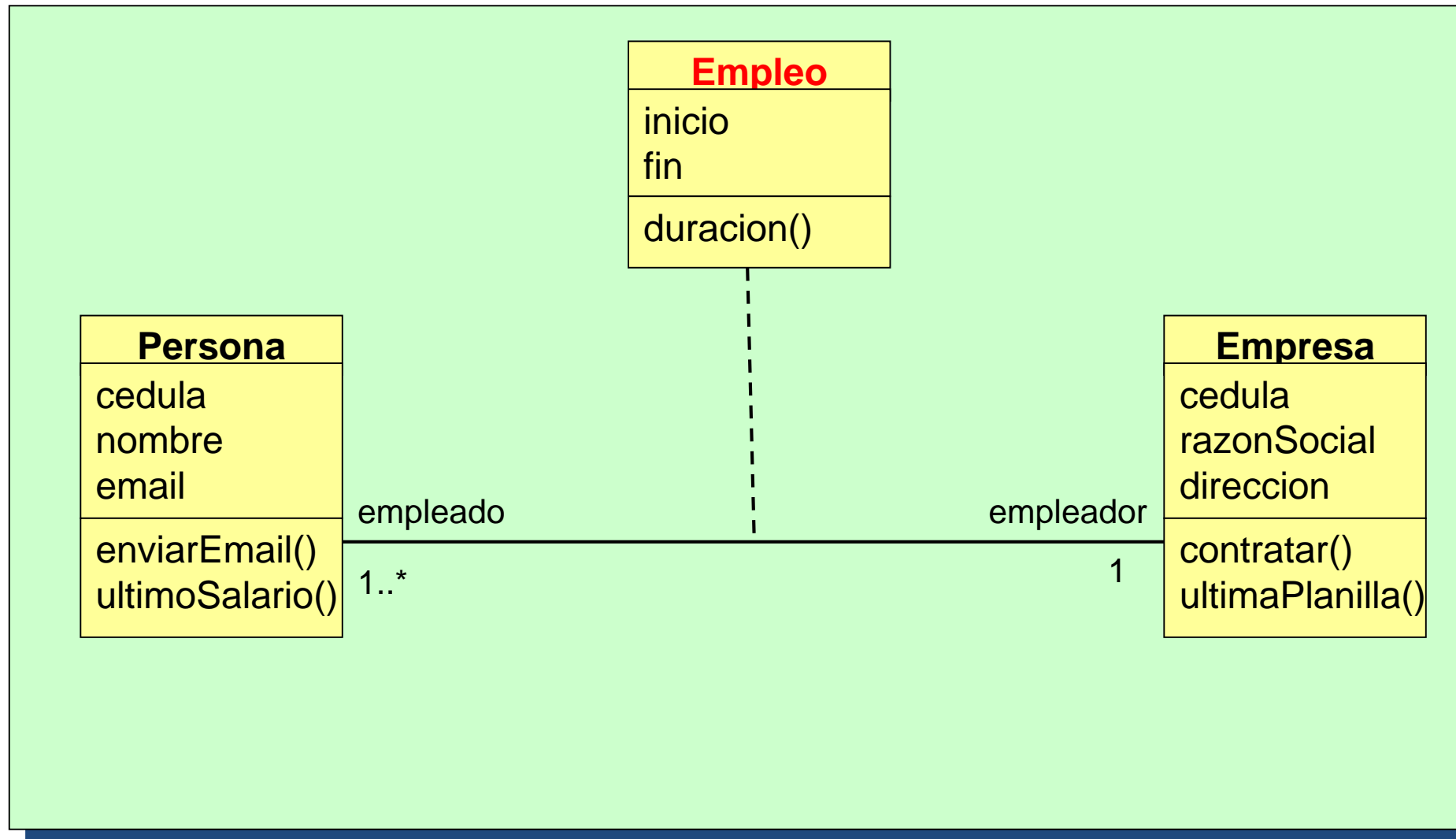
Asociación



Nota importante de diseño

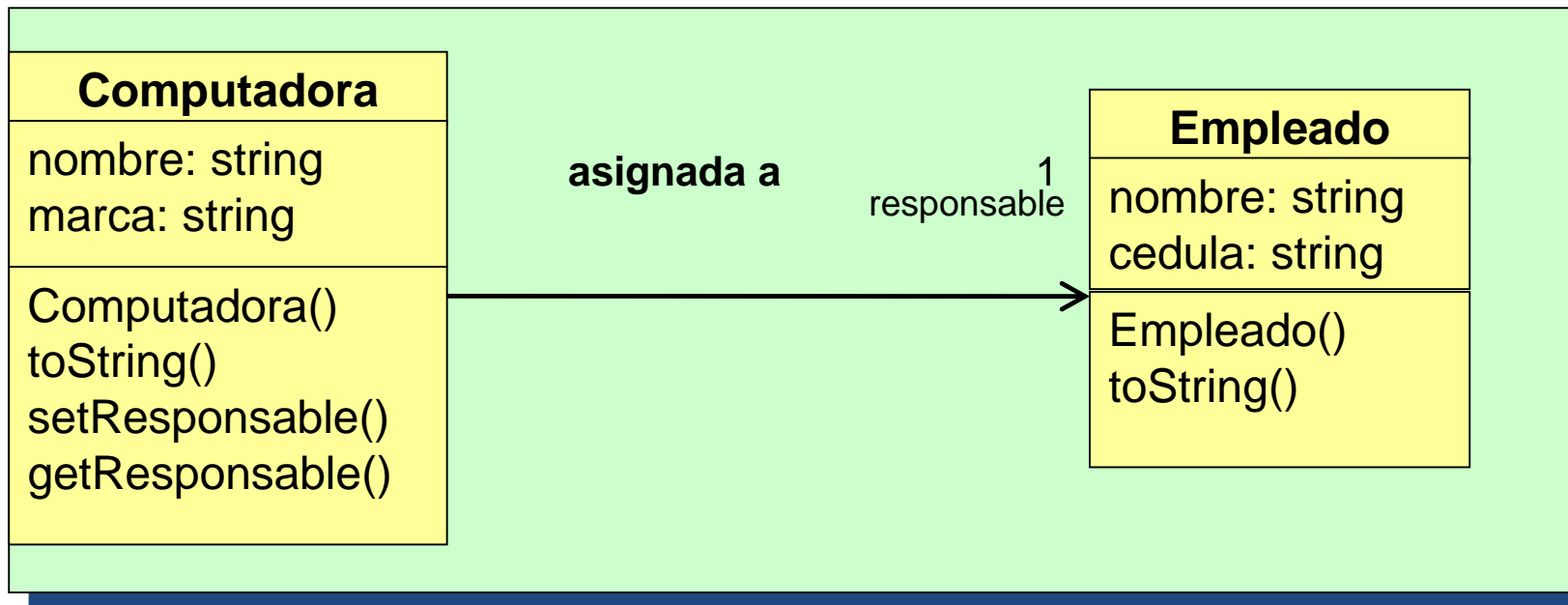
- Si una relación tiene atributos y operaciones propios entonces puede constituirse en una clase.
- Por ejemplo, si se quiere registrar las fechas en que inició y terminó la relación de empleo entre la persona y la empresa, la asociación **Empleo** puede constituirse en una clase.

Asociación



Ejercicio resuelto

- Suponga que tiene el siguiente problema:
- Notas de diseño:
 - vea como la relación se lee “una computadora está asignada a un responsable”
 - vea como en la clase computadora hay un get y un set para un atributo llamado responsable.
 - vea como la flecha apunta de la clase cliente (la que necesita la información u operación) a la clase servidora (la que provee el servicio), pero el atributo se pone en la clase cliente.
 - vea que el atributo NO se pone en clase, sino que está implícito en la relación (responsable es un atributo de tipo empleado que va en la clase computadora)



Asociación

```
public class Empleado
{ private String mnombre;
  private String mcedula;

  —
  public Empleado(string pnombre,string pcedula){
    setNombre(pnombre);
    setCedula(pcedula)
  }

  public string getNombre()
  { return mnombre;}
  . . .
}
```

Asociación

```
Empleado unResp= new Empleado("Juan Perez","208");  
Computadora unaMaq=new Computadora("234","Sony");  
unaMaq.setResponsable(unResp);
```

```
public class Computadora  
{ private String mnumSerie;  
  private String mmarca;  
  Empleado mresponsable;  
—  
  public Computadora(String pnumSerie,String pmarca){  
    setNumSerie(pnumSerie);  
    setMarca(pmarca);}   
  
  public Computadora(String pnumSerie,String pmarca, Empleado pemp){  
    setNumSerie(pnumSerie);  
    setMarca(pmarca);  
    setResponsable(pemp);}   
  
  public void setResponsable(Empleado pemp)  
  { mresponsable=pemp;}  
  ...  
}
```


Asociación

```
Empleado unEmp= new Empleado("Juan Perez","208");  
Computadora unaMaq=new Computadora("234","Sony");  
unaMaq.setResponsable(unEmp);
```

Ahora, es posible obtener el nombre del responsable

```
public String getNombreResponsable(){  
    return mresponsable.getNombre();  
}  
}
```

```
out.println(unaMaq.getNombreResponsable());
```

Jerarquía de objetos: Composición y Agregación

Agregación

- Como se mencionó, la agregación establece un tipo de relación todo/parte.
- La agregación puede o no denotar contención física.
- En muchos casos la relación todo/parte será más conceptual y por tanto menos directa que la contención física.

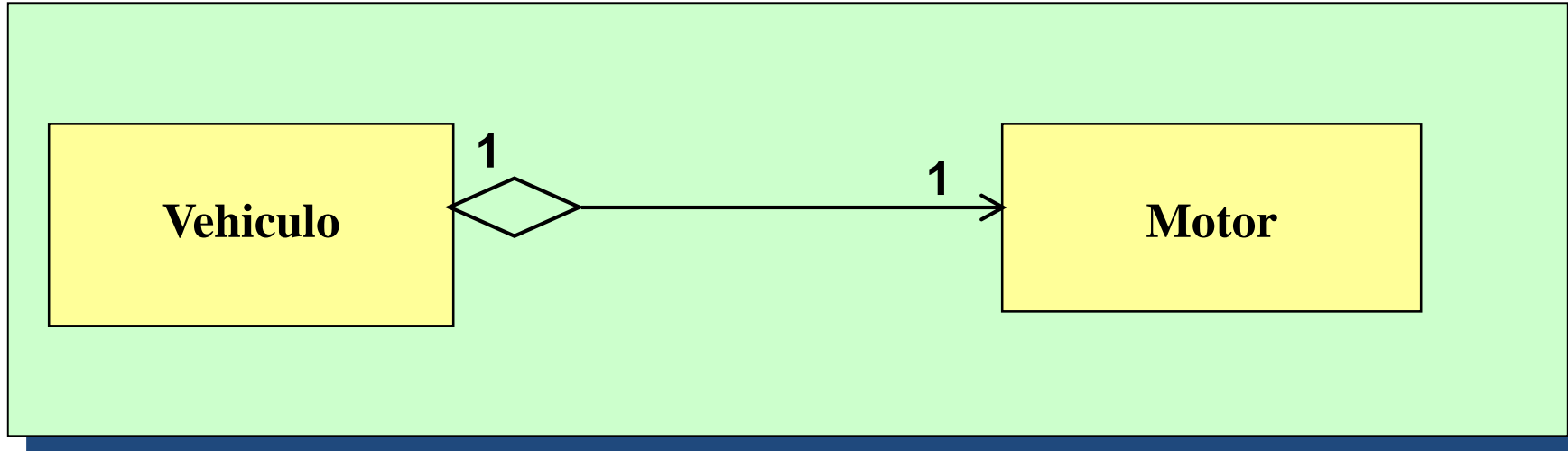
Agregación

- La relación de agregación puede leerse de la siguiente manera:
 - el compuesto **tiene-un** componente
- La agregación **simple** puede implementarse incluyendo en la clase compuesta **un atributo** de la clase componente.
- La **múltiple** puede implementarse incluyendo en la clase compuesta un colección (**arreglo, ArrayList, etc.**) de la clase componente

Agregación

- La agregación existe si hay una relación de contención, o de todo / parte.
- La agregación existe si la relación entre el todo y la parte no es exclusiva.
- La agregación existe si la vida del todo y la parte son independientes: si se elimina el todo, las partes existen.

Agregación



Agregación

```
public class Motor
{ private String mserie;
  private int mnumCilindros;

  —
  public Motor(string pserie,string pcilindros){
    setSerie(pserie);
    setCilindros(pcilindros);
  }

  public string getSerie()
  { return mserie;}
  ...
}
```

Agregación

```
public class Vehiculo
{ private String mnumSerie;
  private String mmarca;
  private Motor mmotor;
```

```
—
  public Vehiculo(String pnumSerie,String pmarca){
    setNumSerie(pnumSerie);
    setMarca(pmarca);
    mmotor=new Motor(pnumSerie,4); //Este constructor tiene que crear el motor
  }
```

```
  public Vehiculo(String pnumSerie,String pmarca, Motor pmotor){
    setNumSerie(pnumSerie);
    setMarca(pmarca);
    setMotor(pmotor);}
  ...
}
```

```
Vehiculo miCarro= new Vehiculo("123","Toyota");
out.println(miCarro.getSerieMotor());
```


Agregación

```
class Vehiculo
{ private String mnumSerie;
  private String mmarca;
  Motor mmotor;
```

```
—
  public Vehiculo(String pnumSerie,String pmarca){
    setNumSerie(pnumSerie);
    setMarca(pmarca);
    mmotor=new Motor(pnumSerie,4); //Este constructor tiene que crear el motor
  }
```

```
  public Vehiculo(String pnumSerie,String pmarca, Motor pmotor){
    setNumSerie(pnumSerie);
                                     setMarca(pmarca);
    setMotor(pmotor);}
```

```
  ...
}
```

```
Motor unMotor = new Motor ("456",6);
Vehiculo miCarro= new Carro("123","Toyota", unMotor);
out.println(miCarro.getSerieMotor());
```

Composición

- La relación de composición puede leerse de la siguiente manera:
 - el componente **es-parte-de** un compuesto.
- La composición **simple** puede implementarse incluyendo en la clase compuesta **un atributo** de la clase componente.
- La **múltiple** puede implementarse incluyendo en la clase compuesta un colección (**arreglo, ArrayList, etc.**) de la clase componente

Composición

- La composición (agregación compuesta) se representa en UML por una línea que une el compuesto con el componente, con un **rombo relleno** en el extremo del compuesto.
- La multiplicidad se indica en cada extremo: uno (1), muchos (*), uno o mas (1..*), etc.
- La navegabilidad se indica con **una flecha**. Usualmente desde el compuesto puede navegarse hacia (recuperarse) el componente.

Composición

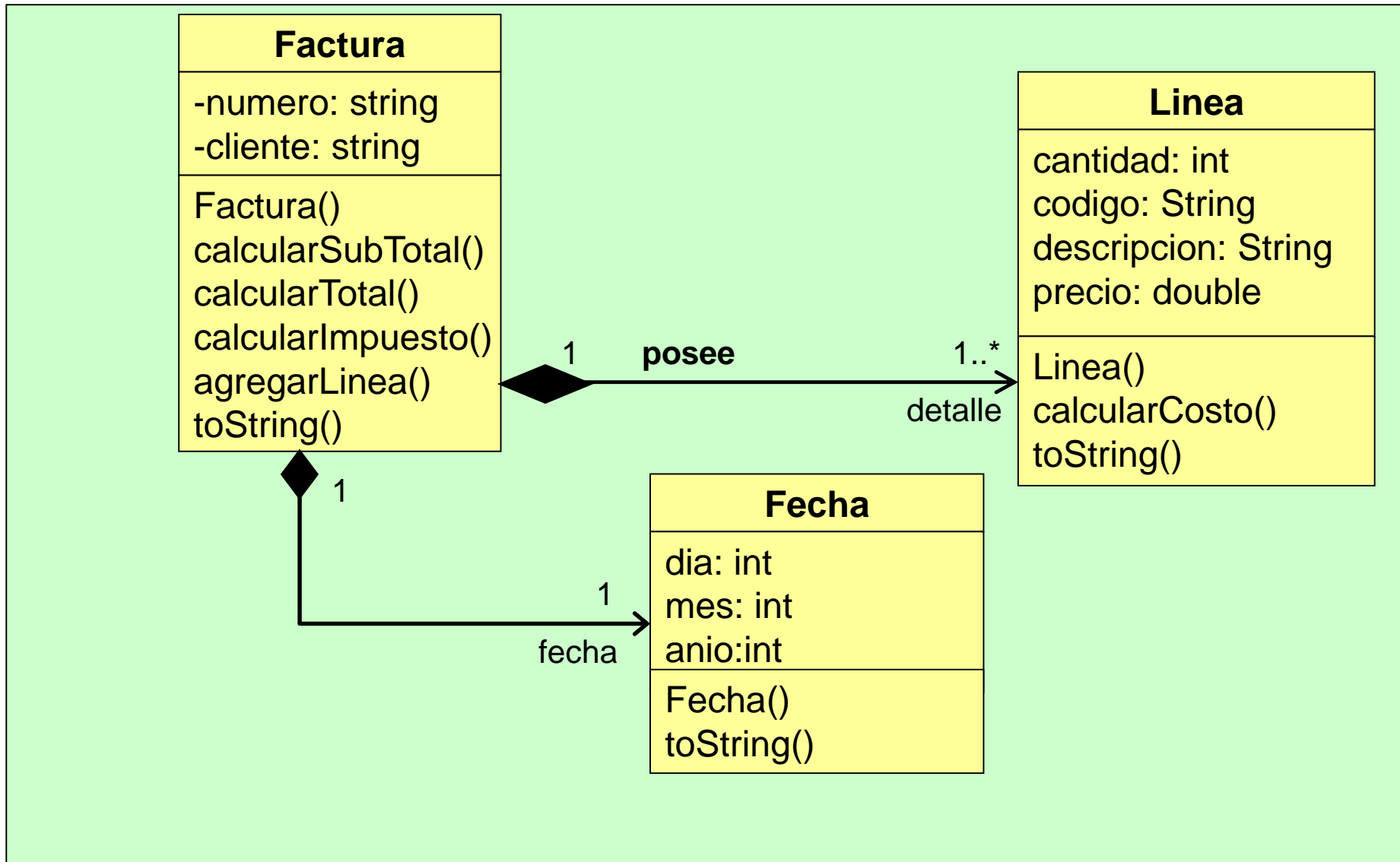
- La composición existe si hay una relación de contención, o de todo / parte.
- La composición existe si la relación entre el todo y la parte no exclusiva, es decir no puede tener ninguna relación con ninguna otra clase.
- La composición existe si la vida del todo y la parte son independientes: si se elimina el todo, las partes se eliminan también.

Composición

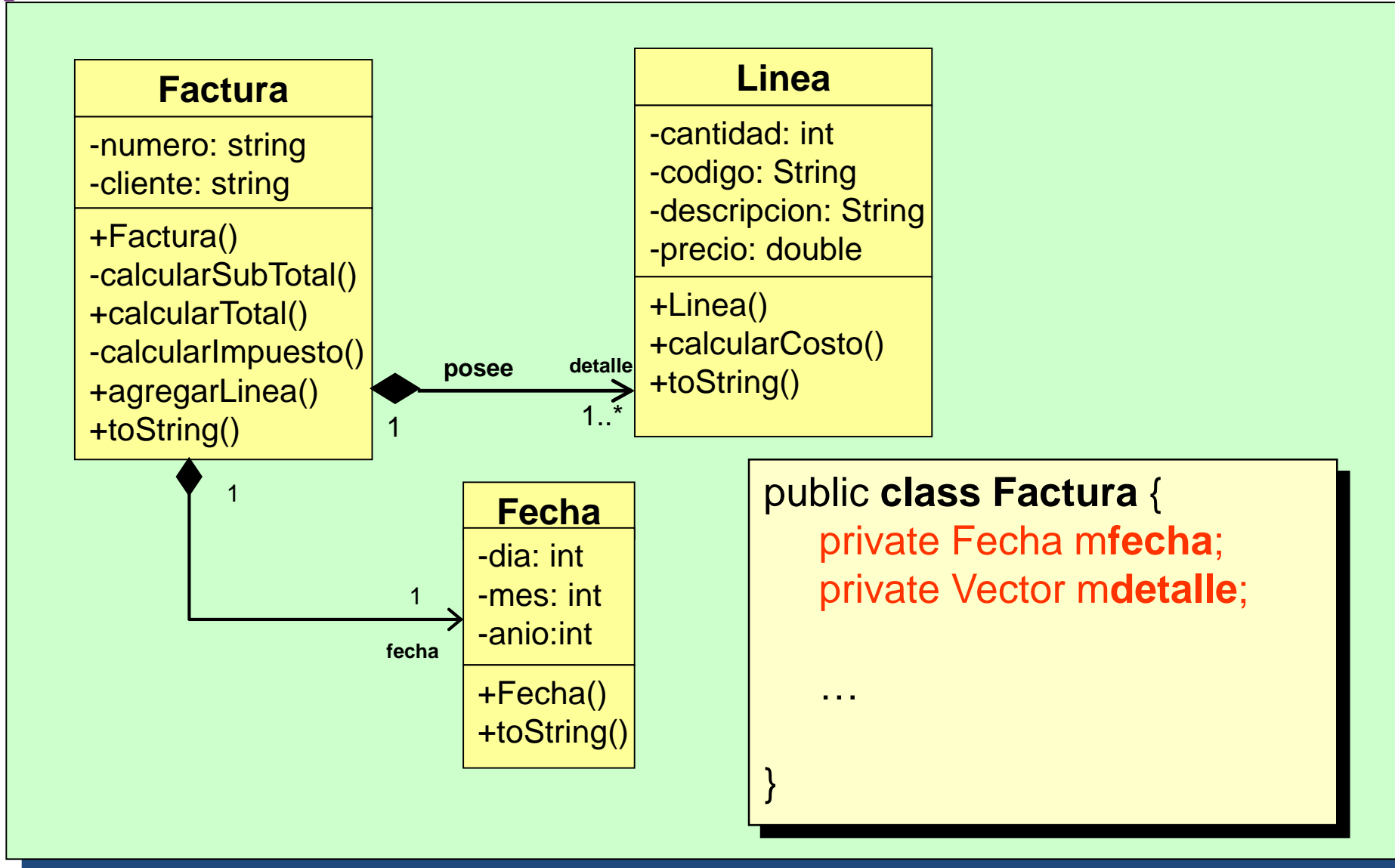
Factura

Joyería La Perla			No. 0065433	
Cliente: Juan Pérez			12 / 05 / 2002	
cant.	código	descripción	precio	costo
2	3702	camisas	5300	10600
1	2805	cepillo	8200	8200
1	1531	corbata	4100	4100
			subtotal	22900
			impuesto	2977
			total	25877

Composición

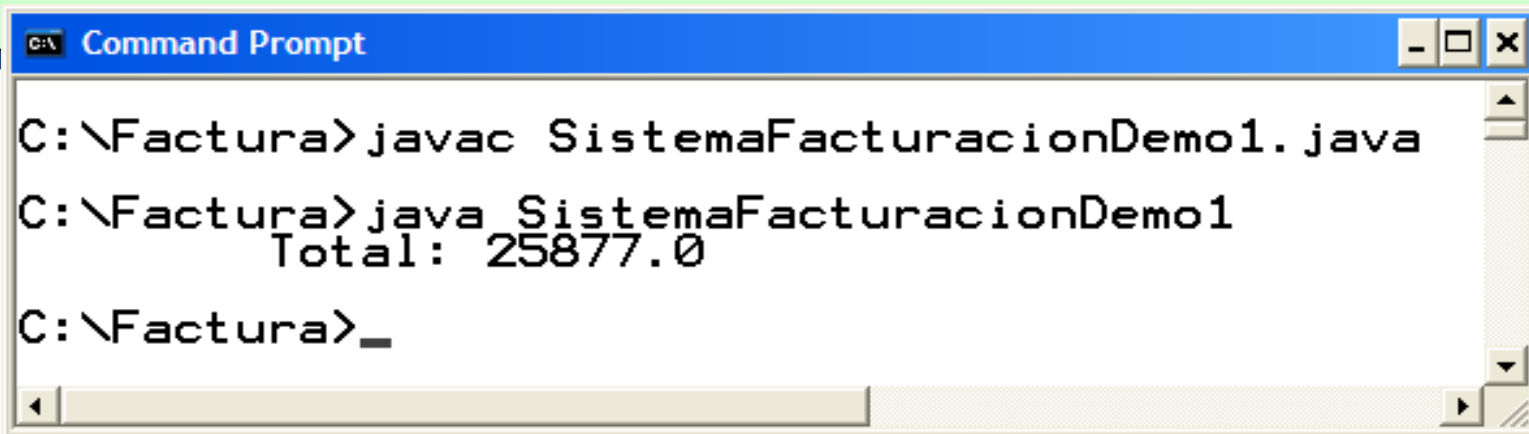


Composición



Composición

```
public class SistemaFacturacionDemo1 {  
    public static void main( String[] args ){  
        Factura factura;  
  
        factura = new Factura("0065433","Juan Perez",12,5,2002);  
        factura.agregarLinea(2,"3702","camisas",5300);  
        factura.agregarLinea(1,"2805","cepillo",8200);  
        factura.agregarLinea(1,"1531","corbata",4100);  
        System.out.println("Total: " + factura.calcularTotal());  
        //System.out.println(factura.toString());  
    }  
}
```

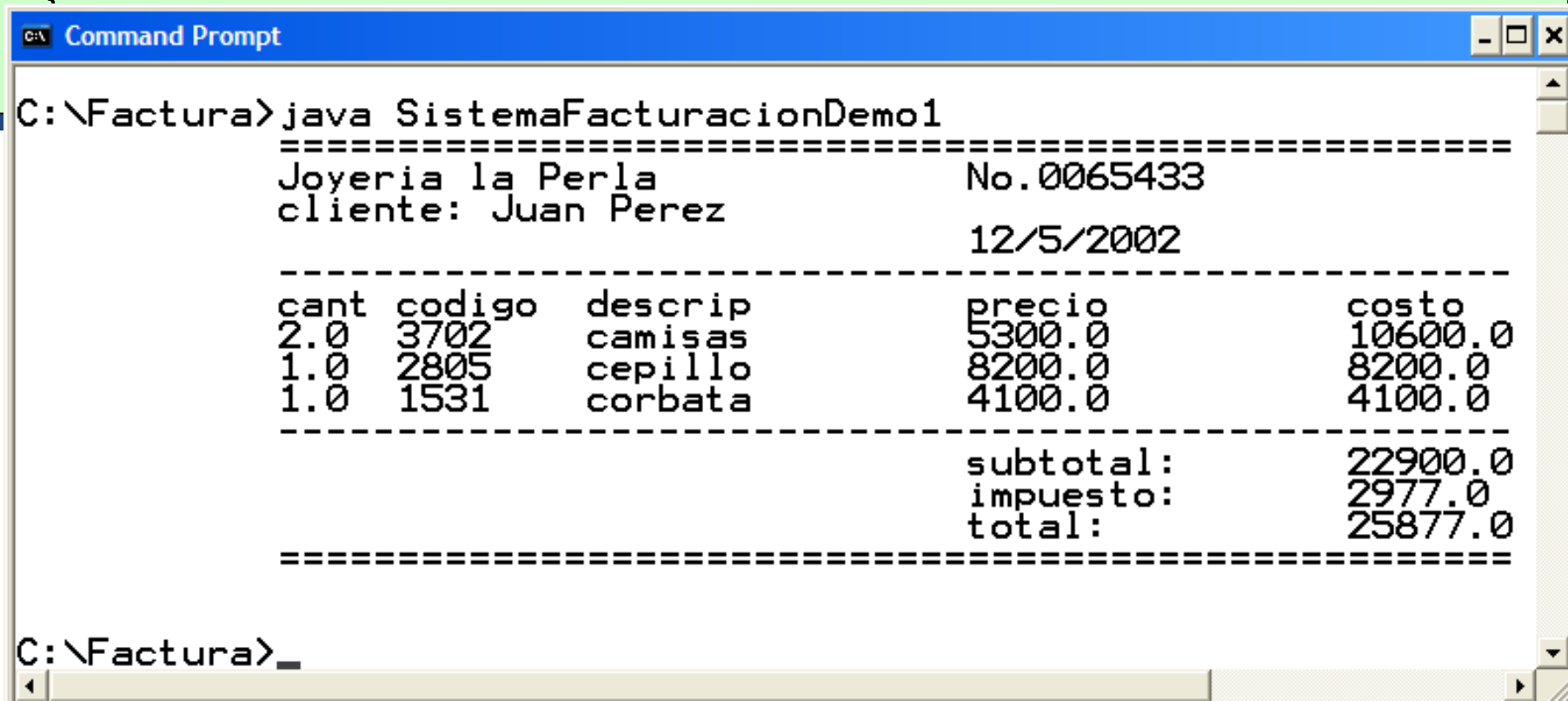


The screenshot shows a Windows Command Prompt window with the following text:

```
C:\Factura>javac SistemaFacturacionDemo1.java  
C:\Factura>java SistemaFacturacionDemo1  
Total: 25877.0  
C:\Factura>_
```


Composición

```
public class SistemaFacturacionDemo1 {  
    public static void main( String[] args ){  
        Factura factura;  
  
        factura = new Factura("0065433","Juan Perez",12,5,2002);  
        ...  
        System.out.println(factura.toString());  
    }  
}
```



```
C:\Factura>java SistemaFacturacionDemo1  
=====No.0065433=====  
Joyeria la Perla  
cliente: Juan Perez  
12/5/2002  
-----  
cant  codigo  descrip  precio  costo  
2.0   3702    camisas  5300.0  10600.0  
1.0   2805    cepillo  8200.0  8200.0  
1.0   1531    corbata  4100.0  4100.0  
-----  
subtotal: 22900.0  
impuesto: 2977.0  
total: 25877.0  
=====
```

C:\Factura>_

Composición

```
public class Factura {  
    private String mnumero;  
    private String mcliente;  
    private Fecha mfecha;  
    private Vector mdetalle;  
    /* Constructor */  
    public Factura( String pnumero,  
                   String pcliente,  
                   int pdia,  
                   int pmes,  
                   int panio ){  
        setNumero(pnumero);  
        setCliente(pcliente);  
        mfecha = new Fecha(pdia,pmes,panio);  
        mdetalle = new Vector();  
    }
```

Composición a través
de una colección

Composición

/* Subtotal de la factura, i.e. el total sin impuesto */

```
private float calcularSubtotal( ){
```

```
    Linea lineaDetalle;
```

objeto local de tipo
Linea

```
    float subtotal;
```

```
    subtotal = 0;
```

```
    for(int i = 0; i < detalle.size(); i++){
```

```
        lineaDetalle = (Linea) mdetalle.get(i);
```

```
        subtotal = subtotal + lineaDetalle.calcularCosto();
```

```
    }
```

```
    return subtotalL;
```

```
}
```

/* Impuesto de la factura (13% del subtotal) */

```
private float calcularImpuesto( ){
```

```
    float impuesto;
```

```
    impuesto = calcularSubtotal()* 13/100;
```

```
    return impuesto;
```

```
}
```

Factura

- numero: string
- cliente: string
- fecha: Fecha
- detalle: Vector

- +Factura()
- calcularSubTotal()
- +calcularTotal()
- calcularImpuesto()
- +agregarLinea()
- +toString()

Composición: Factura

```
/* Total de la factura (subtotal mas impuesto) */  
public float calcularTotal( ){  
    float total;  
    total = calcularSubtotal() + calcularImpuesto();  
    return total;  
}  
  
/* Agrega una linea de detalle a la factura */  
public void agregarLinea( int pcantidad,  
                           String pcodigo,  
                           String pdescripcion,  
                           float pprecio ){  
    mdetalle.add( new Linea(pcantidad,pcodigo,  
                              pdescripcion, pprecio));  
}
```

Composición: Factura

```
/* Version en texto de todos los datos de la factura. */
public String toString() {
    String msg;
    Linea lineaD;
    msg = "===== " + "\n";
    msg = msg + "Joyeria la Perla";
    msg = msg + "\t\t" + "No. " + numero + "\n";
    msg = msg + "cliente: " + cliente + " ";
    msg = msg + "\t" + mfecha.toString() + "\n";
    msg = msg + "-----" + "\n";
    msg = msg + "cant" + "\t" + "codigo" + "\t" + "descrip" +
        "\t" + "precio" + "\t" + "costo" + "\n";
    for(int i = 0; i < mdetalle.size(); i++){
        lineaD = (Linea) mdetalle.get(i);
        msg = msg + lineaD.toString() + "\n";
    }
    msg = msg + "\t\t\t" + "-----" + "\n";
    msg = msg + "\t\t\t" + "subtotal:" + this.calcularSubtotal() + "\n";
    msg = msg + "\t\t\t" + "impuesto:" + this.calcularImpuesto() + "\n";
    msg = msg + "\t\t\t" + "total  : " + this.calcularTotal() + "\n";
    msg = msg + "===== " + "\n";
    return msg;
}
}
```

Composición

```
public class Fecha {  
    private int mdia;  
    private int mmes;  
    private int manio;  
    /* Constructor */  
    public Fecha( int pdia,  
                  int pmes,  
                  int panno ){  
        setDia(pdia);  
        setMes(pmes);  
        setAnio(panno);  
    }  
    public String toString( ){  
        String msg;  
        msg = mdia + "/" + mmes + "/" + manio;  
        return msg;  
    }  
}
```

Composición

```
public class Linea {  
    private float mcantidad;  
    private String mcodigo;  
    private String mdescpcion;  
    private float mprecio;  
  
    /* Constructor */  
    public Linea( float pcantidad,  
                  String pcodigo,  
                  String pdescpcion,  
                  float pprecio ){  
        setCantidad(pcantidad);  
        setCodigo(pcodigo);  
        setDescription(pdscpcion);  
        setPrecio(pprecio);  
    }  
}
```

Linea
-cantidad: int -codigo: String -descripcion: String -precio: float
+Linea() +calcularCosto() +toString()

Composición

```
/* Costo de una línea (cantidad por el precio) */  
public float calcularCosto( ){  
    return (mcantidad * mprecio);  
}
```

```
/* Version en texto con todos los datos de la linea */  
public String toString( ){  
    String msg;  
    msg = mcantidad + "\t";  
    msg = msg + mcodigo + "\t";  
    msg = msg + mdescripcion + "\t";  
    msg = msg + mprecio + "\t";  
    msg = msg + calcularCosto();  
    return msg;  
}  
}
```


Tarea

- Reemplazar los atributos código, descripción y precio por la clase Producto.
- Encontrar la relación entre línea y producto y justificarla, mediante la entrega de un UML (OJO ALGUNOS ELEGIDOS LO VAN A PRESENTAR)
- En el código ejemplo se usa la clave Vector. Esa clase ya está depreciada. Debe usar ArrayList.
- Debe usar la arquitectura del curso (capas, paquetes, jar e ui).
- La impresión debe ser igual a la mostrada.
- Debe usar la clase Fecha incluida en el diagrama, no puede usar LocalDate.
- Diez puntos sobre el examen final si lo entrega bien el día miércoles, incluyendo que el código sea copia fiel del UML y que se siga el estándar de UML visto en clase.



¿?

**UNIVERSIDAD
CENFOTEC**

**PROGRAMACIÓN
ORIENTADA A
OBJETOS**

**Bachillerato en ingeniería de
software**

Universidad Cenfotec

¡Muchas gracias!



**universidad
cenfotec_**
tecnologías digitales