

Proceso de implementación a partir de un diseño.

Semana 6

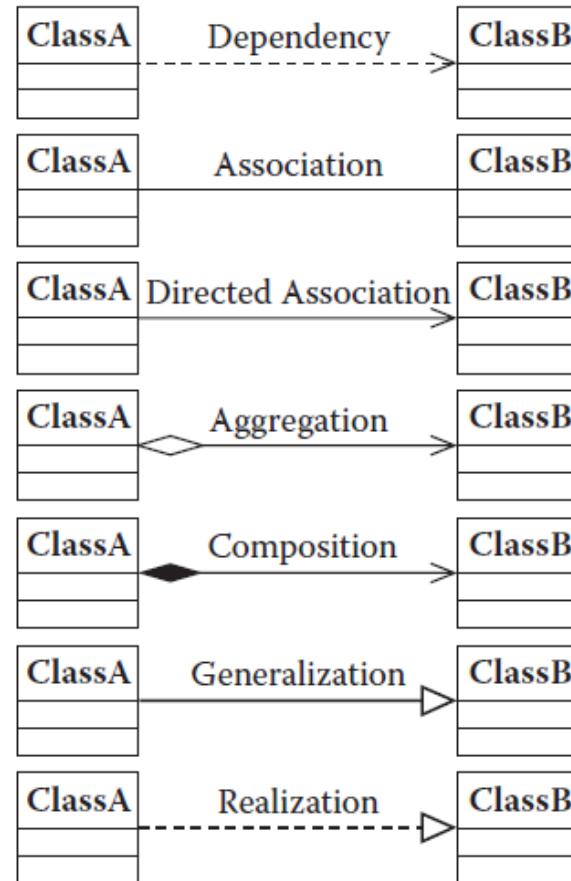
Agenda

- Traduciendo un diagrama de clase en código fuente y viceversa
- Traduciendo un diagrama de secuencia en código fuente y viceversa

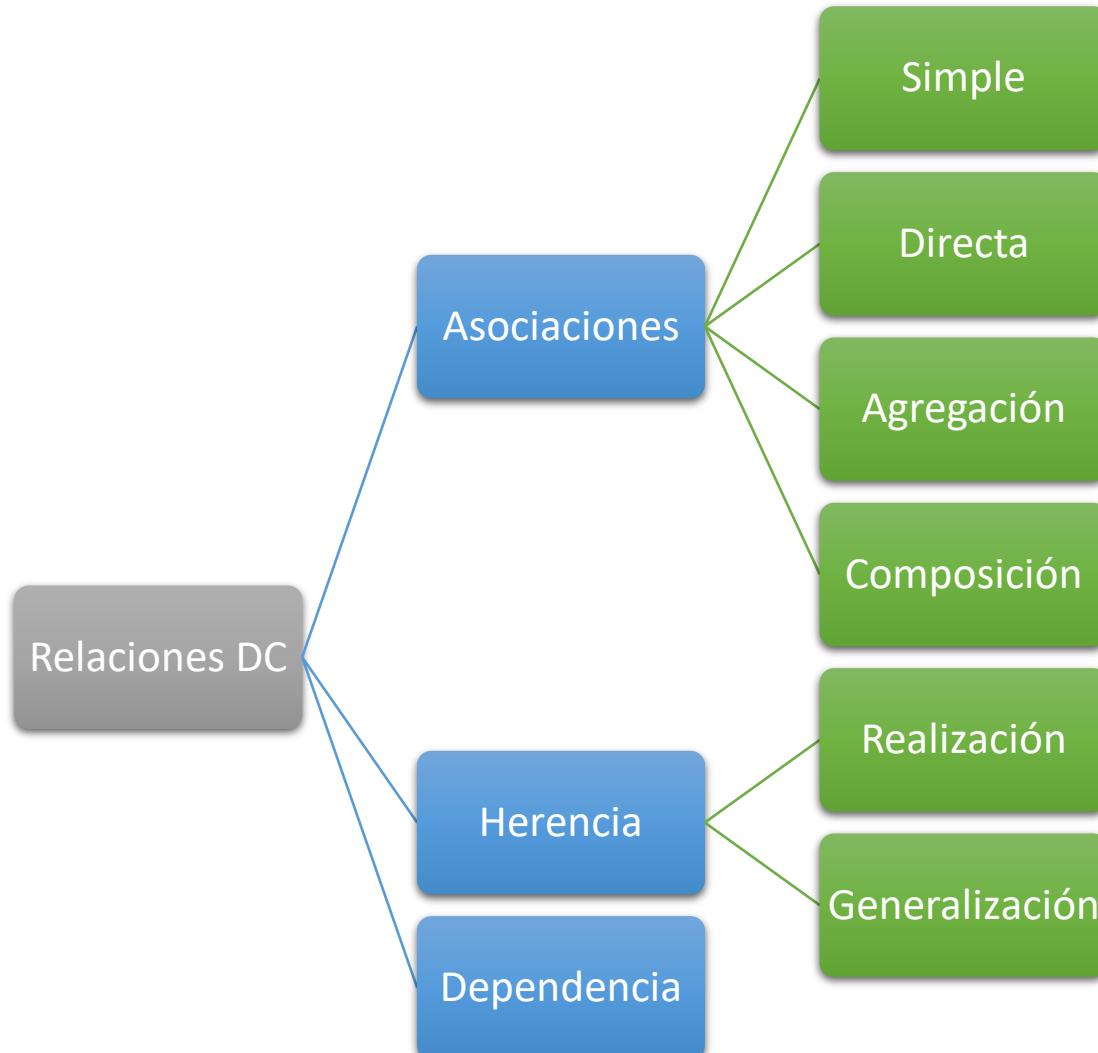
Traduciendo un diagrama de clase en código fuente y viceversa

Relaciones en Diagramas de Clases

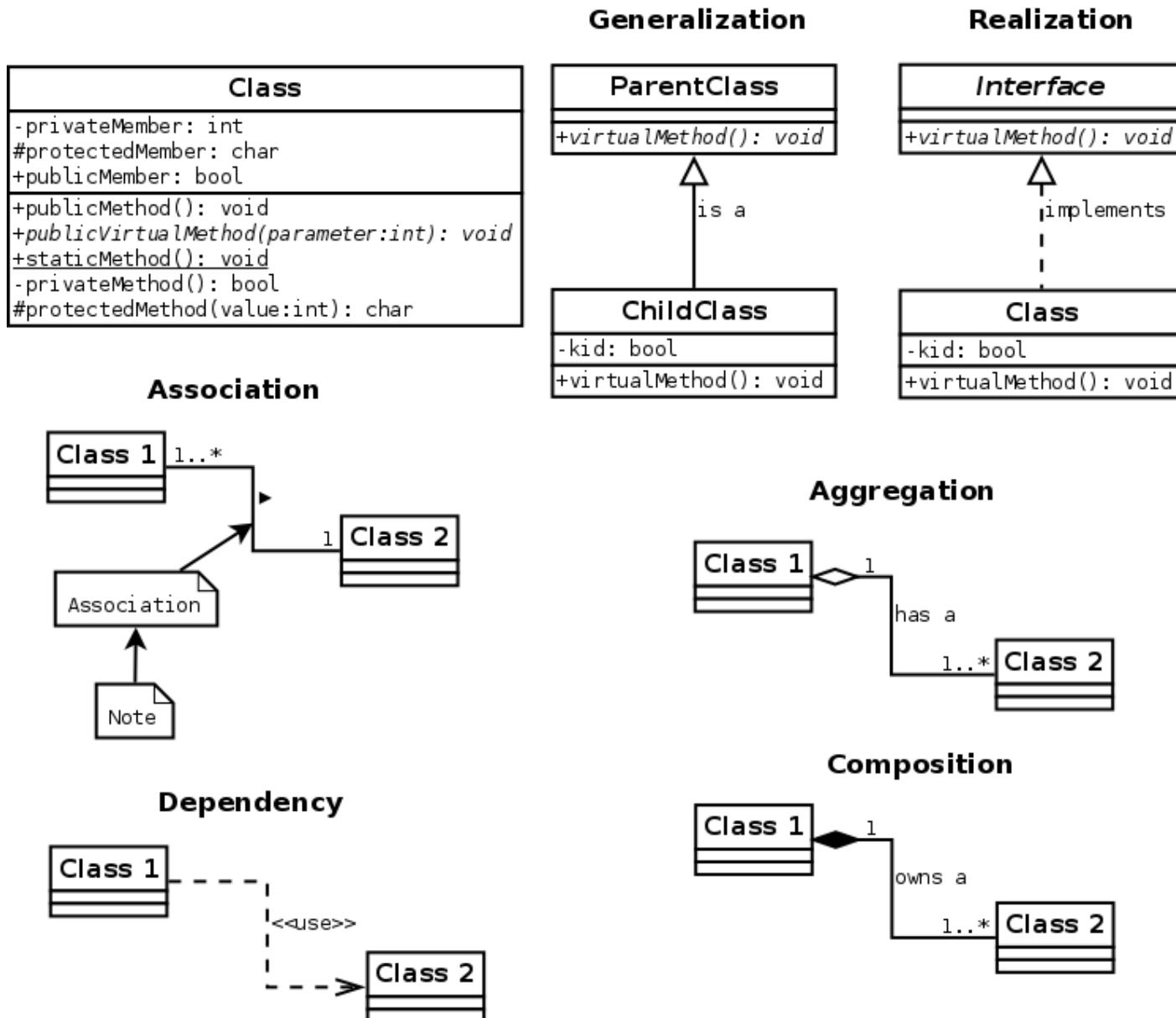
- Las clases deben estar relacionadas con otras.



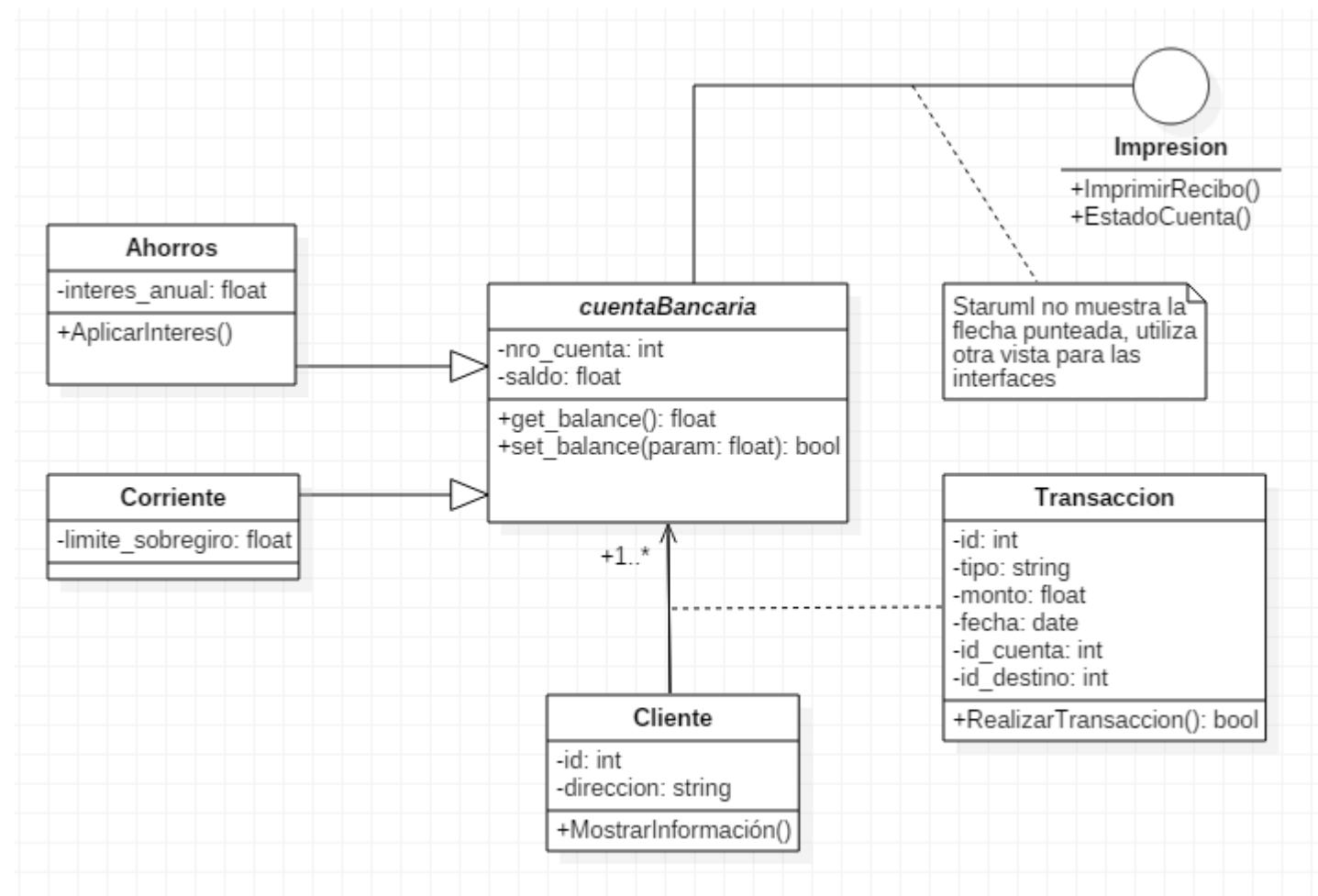
Relaciones en DC



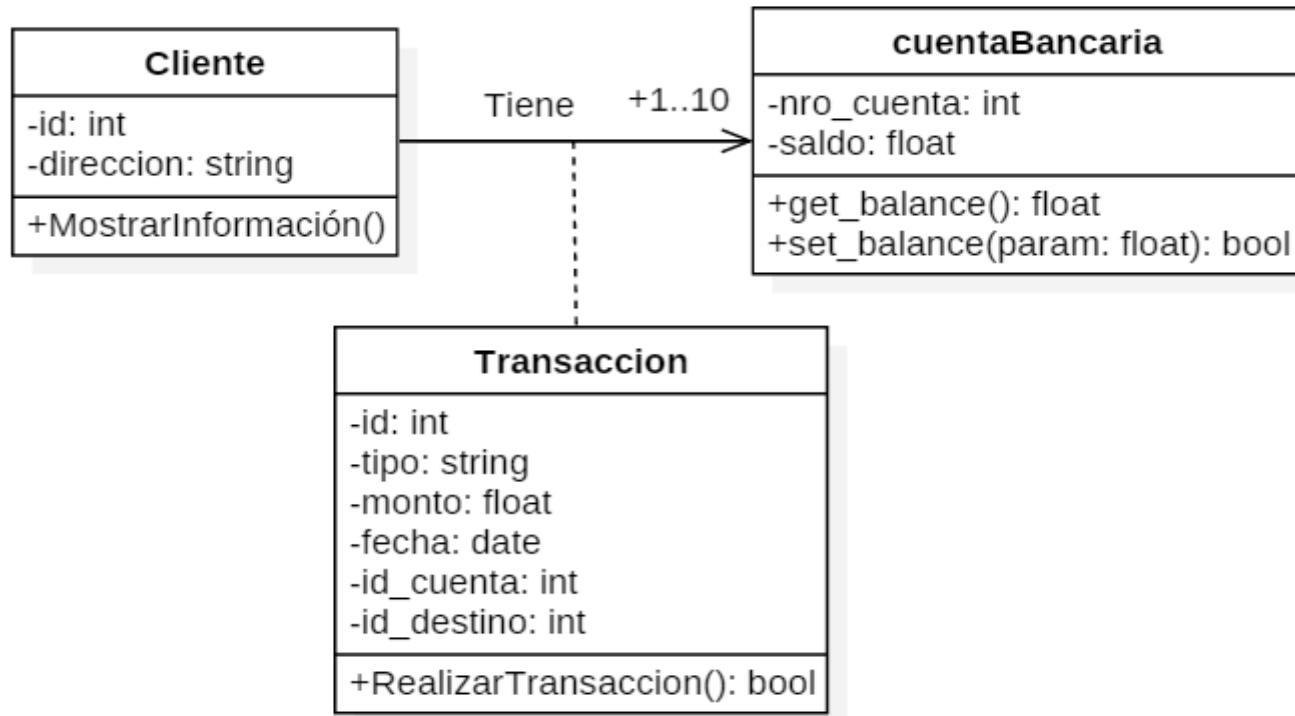
UML Class Diagram Cheat Sheet



Ejemplo de Diagrama de clases



Asociación



Código - Asociación

```
1 import java.util.*;
2
3 /**
4  * 
5  */
6
7 public abstract class CuentaBancaria implements Impresion {
8
9     /**
10      * Default constructor
11      */
12     public CuentaBancaria() {
13
14         private int nro_cuenta;
15
16         private float saldo;
17
18     }
19
20     public float get_balance() {
21         // TODO implement here
22         return 0.0f;
23     }
24
25     public boolean set_balance(float param) {
26         // TODO implement here
27         return null;
28     }
29
30     public void ImprimirRecibo() {
31         // TODO implement here
32     }
33
34     public void EstadoCuenta() {
35         // TODO implement here
36     }
37
38 }
39 }
```

```
1 import java.util.*;
2
3 /**
4  * 
5  */
6
7 public class Cliente {
8
9     /**
10      * Default constructor
11      */
12     public Cliente() {
13
14         private int id;
15
16         private String direccion;
17
18         public ArrayList<CuentaBancaria> cuentas;
19
20
21     public void MostrarInformación() {
22         // TODO implement here
23     }
24
25
26 }
```

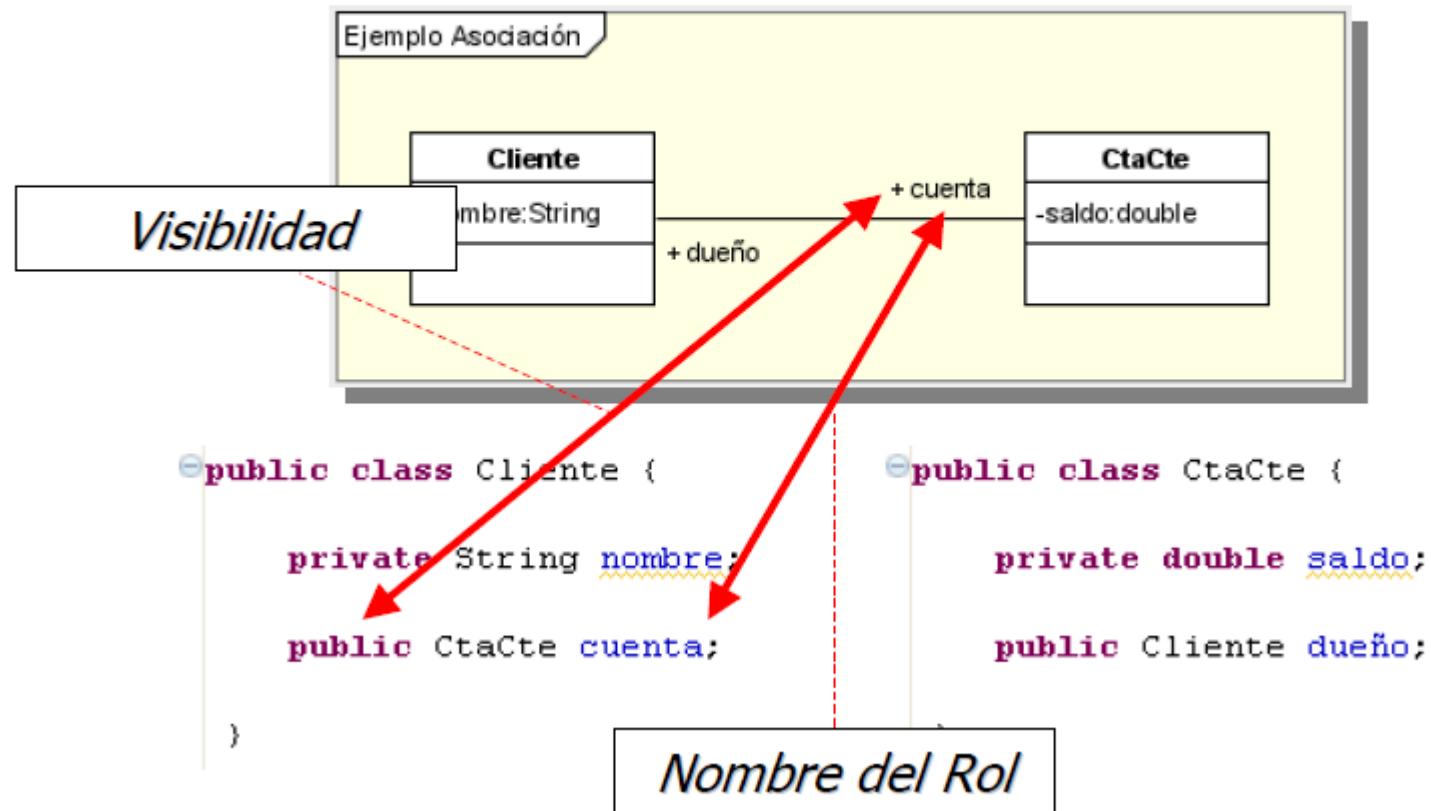
Clase de asociación

- La clase de asociación se instancia al ejecutar la relación entre Cliente y CuentaBancaria.
- RealizarTransaccion () asocia los clientes y las cuentas.

```
1  import java.util.*;  
2  
3 4 ▼ /**  
5   *  
6   */  
7 ▼ public class Transaccion {  
8  
9 ▼   /**  
10    * Default constructor  
11    */  
12   public Transaccion() {  
13   }  
14  
15   private int id;  
16  
17   private String tipo;  
18  
19   private float monto;  
20  
21   private Date fecha;  
22  
23   private int id_cuenta;  
24  
25   private int id_destino;  
26  
27 ▼   /**|  
28   * @return  
29   */  
30 ▼   public boolean RealizarTransaccion(int id_cuenta, int id_destino) {  
31       // TODO implement here  
32       return null;  
33   }  
34  
35 }
```

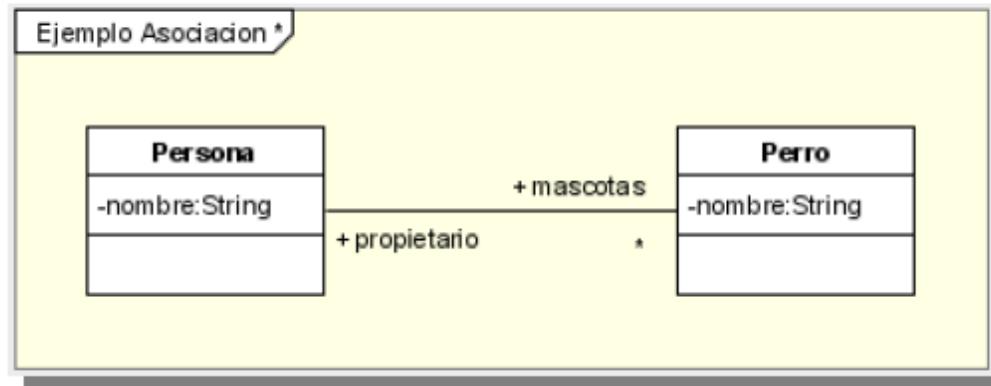
Bidireccionalidad

- Bidireccional con multiplicidad 0..1 o 1



Bidireccionalidad

- Bidireccional con multiplicidad *



```
public class Persona {

    private String nombre;

    public java.util.Collection mascotas = new java.util.TreeSet();

}
```



```
public class Perro {

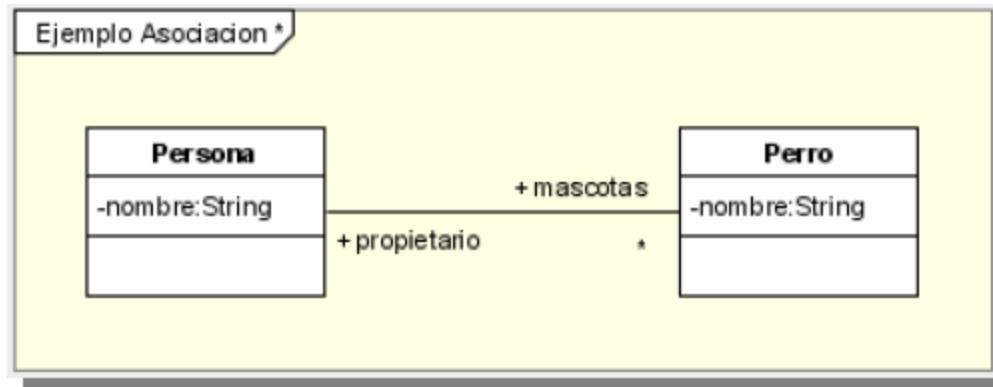
    private String nombre;

    public Persona propietario;

}
```

Bidireccionalidad

- Bidireccional con multiplicidad *



```
public class Persona {

    private String nombre;

    public java.util.Collection mascotas = new java.util.TreeSet();
}
```

Decisión de Implementación

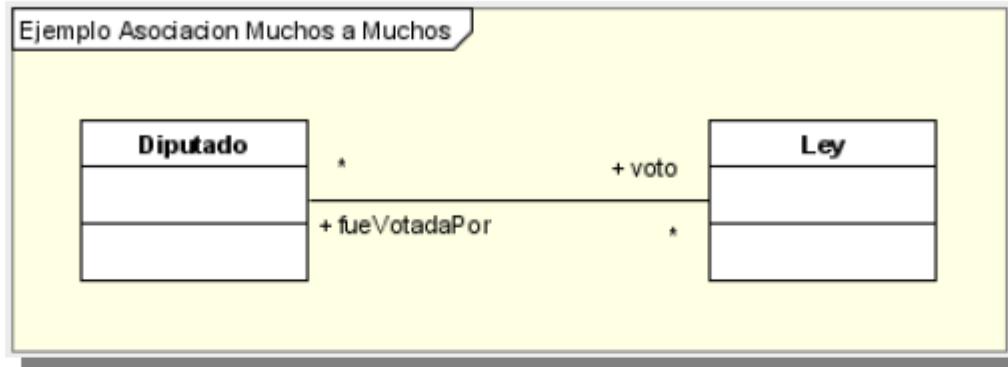
```
public class Perro {

    private String nombre;

    public Persona propietario;
}
```

Bidireccionalidad

- Bidireccional con multiplicidad *



```
public class Diputado {

    public java.util.Collection voto = new java.util.TreeSet();

}

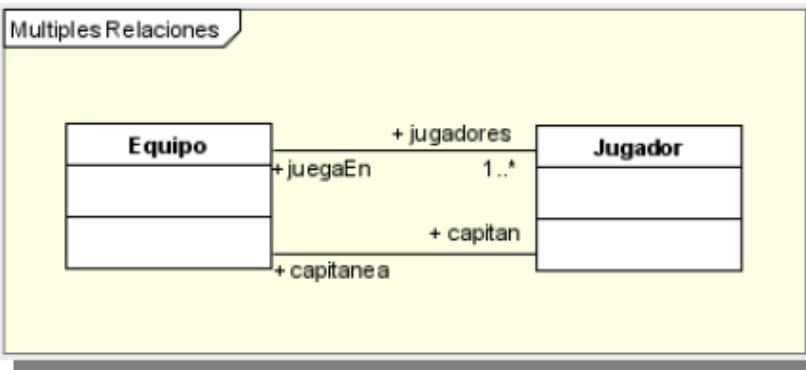
public class Ley {

    public java.util.Collection fueVotadaPor = new java.util.TreeSet();

}
```

Múltiples relaciones

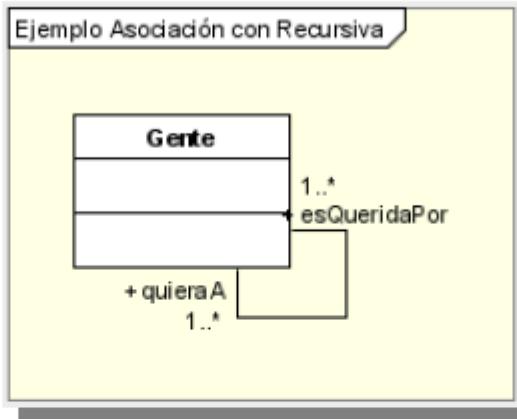
- ¿Con más de una relación?



```
@public class Equipo {  
  
    public Jugador capitana;  
  
    public java.util.Collection jugadores = new java.util.TreeSet();  
  
}  
  
@public class Jugador {  
  
    public Equipo capitanea;  
  
    public Equipo juegaEn;  
  
}
```

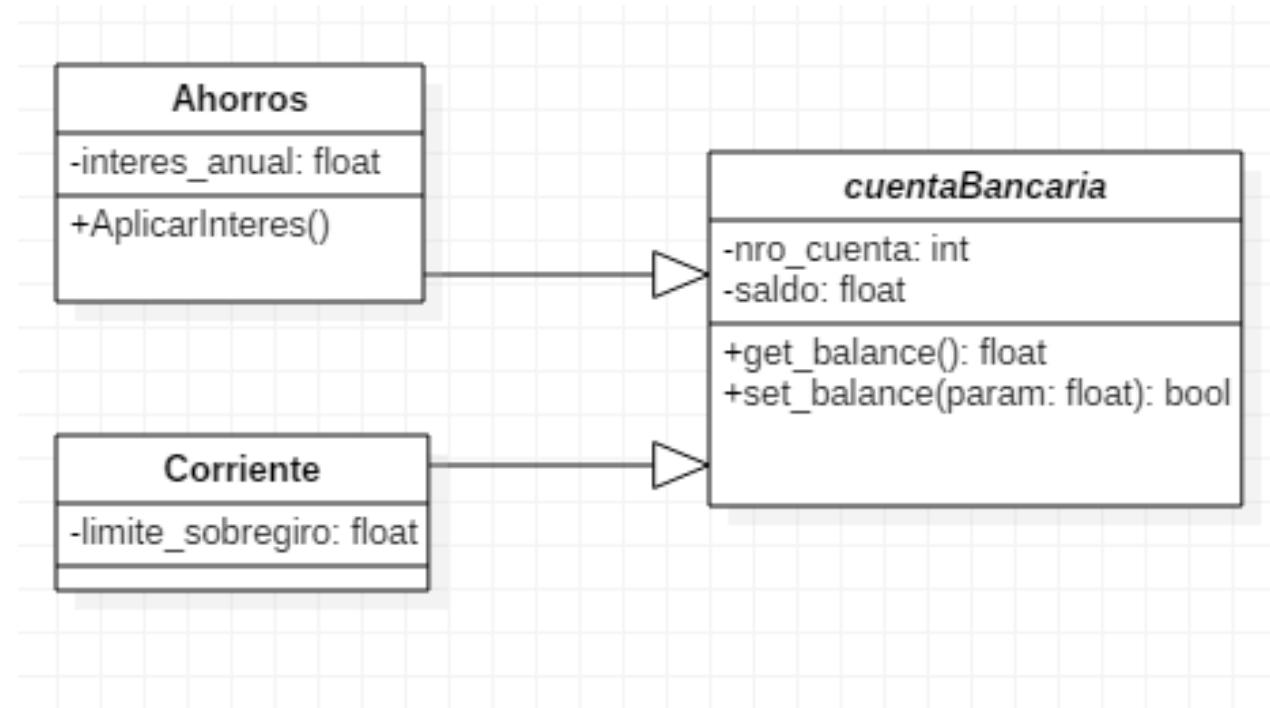
Asociación con recursividad

- ¿Y con esto?



```
@public class Gente {  
  
    public java.util.Collection quieraA = new java.util.TreeSet();  
  
    public java.util.Collection esQueridaPor = new java.util.TreeSet();  
}
```

Ejemplo - Generalización



Generalización

```
1 import java.util.*;  
2  
3  
4 /**  
5 *  
6 */  
7 public abstract class CuentaBancaria implements Impresion {  
8  
9 /**  
10 * Default constructor  
11 */  
12 public CuentaBancaria() {  
13 }  
14  
15 private int nro_cuenta;  
16  
17 private float saldo;  
18  
19  
20 public float get_balance() {  
21 // TODO implement here  
22 return 0.0f;  
23 }  
24  
25 public boolean set_balance(float param) {  
26 // TODO implement here  
27 return null;  
28 }  
29  
30 public void ImprimirRecibo() {  
31 // TODO implement here  
32 }  
33  
34 public void EstadoCuenta() {  
35 // TODO implement here  
36 }  
37  
38  
39 }
```

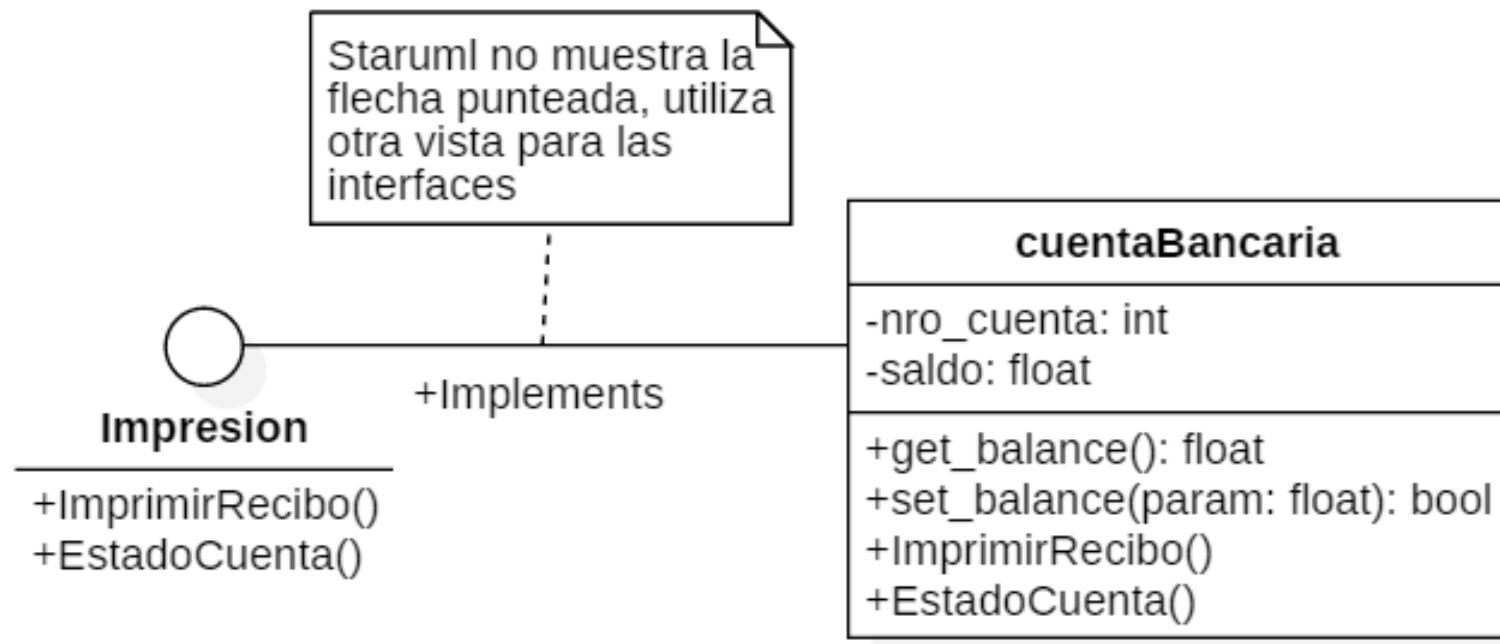
```
1 import java.util.*;  
2  
3  
4 /**  
5 *  
6 */  
7 public final class Ahorros extends CuentaBancaria {  
8  
9 /**  
10 * Default constructor  
11 */  
12 public Ahorros() {  
13 }  
14  
15 private float interes_anual;  
16  
17 public void AplicarInteres() {  
18 // TODO implement here  
19 }  
20  
21 }
```



En el DC el nombre de la clase está en cursivas
esto se traduce como clase abstracta

Ejemplo - Realización

- Impresión es una **Interface**
- CuentaBancaria **implementa** esa interface y por lo tanto desarrolla los métodos **ImprimirRecibo()** y **EstadoCuenta()**



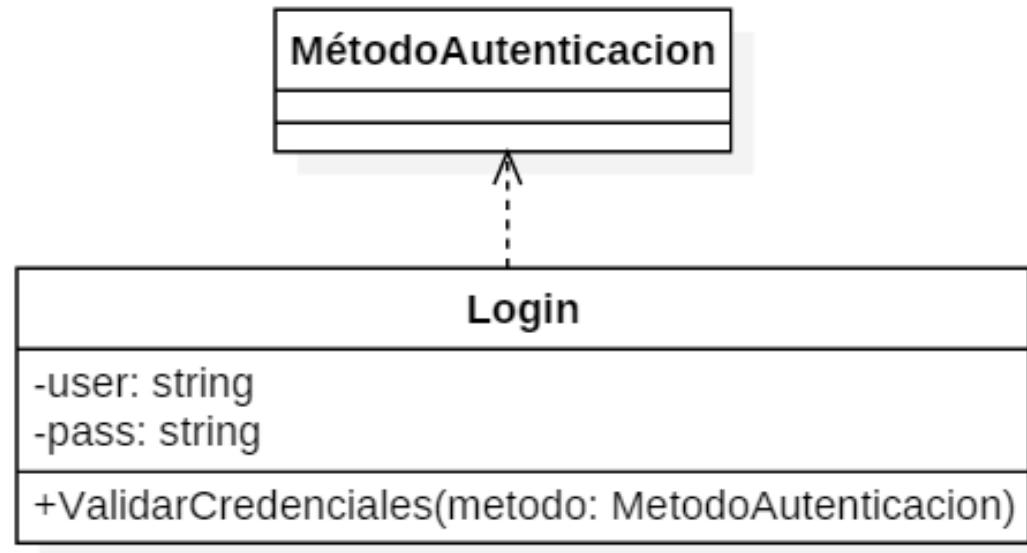
Realización

```
1  import java.util.*;
2
3
4  /**
5   *
6   */
7  public abstract class CuentaBancaria implements Impresion {
8
9    /**
10     * Default constructor
11     */
12    public CuentaBancaria() {
13    }
14
15    private int nro_cuenta;
16
17    private float saldo;
18
19
20    public float get_balance() {
21        // TODO implement here
22        return 0.0f;
23    }
24
25    public boolean set_balance(float param) {
26        // TODO implement here
27        return null;
28    }
29
30    public void ImprimirRecibo() {
31        // TODO implement here
32    }
33
34    public void EstadoCuenta() {
35        // TODO implement here
36    }
37
38
39 }
```

```
1  import java.util.*;
2
3
4  /**
5   *
6   */
7  public interface Impresion {
8
9    public void ImprimirRecibo();
10
11    public void EstadoCuenta();
12
13 }
```

Dependencia

- Indica que los cambios sobre la clase B pueden afectar a la clase A.
- ClassA usa los servicios de ClassB.
- En Java se podría representar con un import.

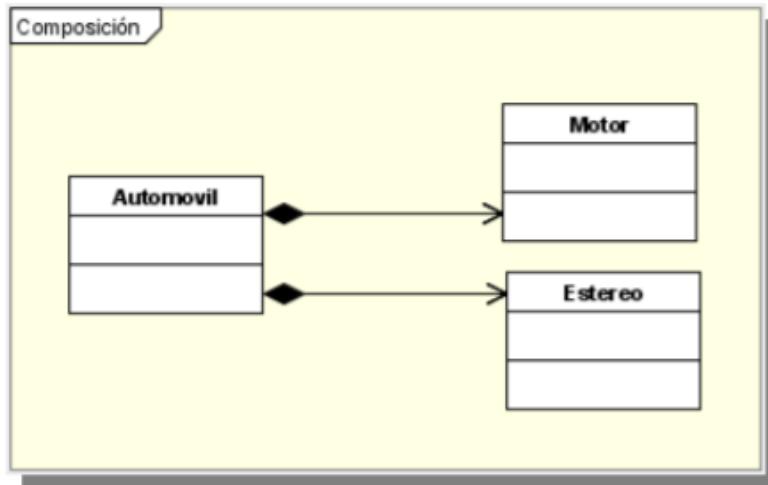


Dependencia

```
1 import java.util.*;
2 import authentication.MetodoAutenticacion;
3 /**
4  * 
5  */
6 public class Login {
7
8
9    /**
10     * Default constructor
11     */
12    public Login() {
13    }
14
15    /**
16     *
17     */
18    private string user;
19
20    /**
21     *
22     */
23    private string pass;
24
25    /**
26     * @param metodo
27     */
28    public void ValidarCredenciales(MetodoAutenticacion metodo) {
29        // TODO implement here
30    }
31
32 }
```

```
1 import java.util.*;
2
3 package authentication;
4 /**
5  * 
6  */
7 public class MetodoAutenticacion {
8
9
10   /**
11    * Default constructor
12    */
13   public MetodoAutenticacion() {
14   }
15
16 }
```

EjemplIFICACIÓN de composición

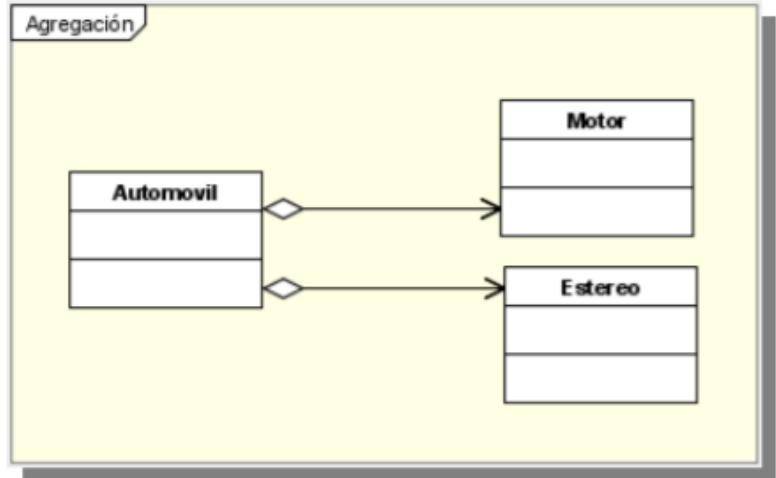


```
public class Automovil {

    public Estereo estereo;
    public Motor motor;

    public Automovil() {
        estereo = new Estereo();
        motor = new Motor();
    }
}
```

Ejemplificación de Agregación



```
public class Automovil {

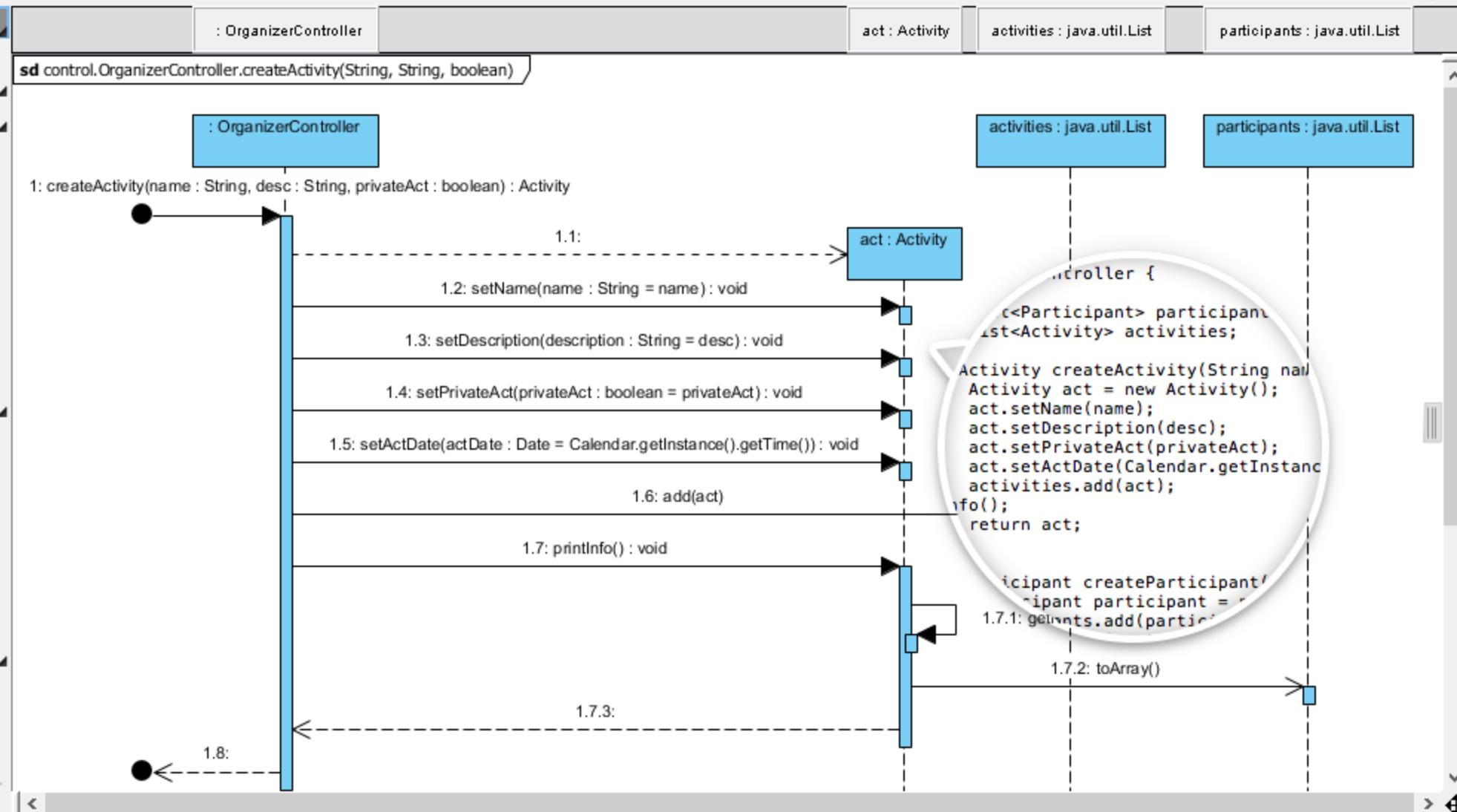
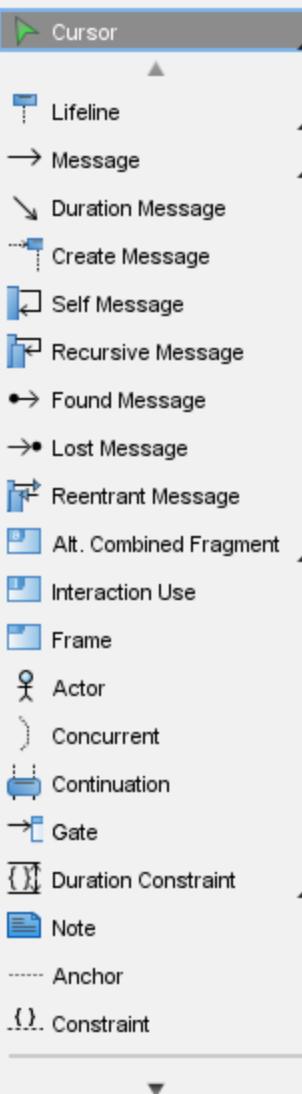
    public Estereo estereo;
    public Motor motor;

    public Automovil() {
    }

    public void ensamblar(Estereo e, Motor m) {
        estereo = e;
        motor = m;
    }
}
```

Traduciendo un diagrama de secuencia en código fuente y viceversa

control.OrganizerController.createActivity(String, String, boolean)



Antes de finalizar

Puntos para recordar

- Recordar cómo ir de un diagrama de clase a código fuente y viceversa.
- Recordar cómo ir de un diagrama de secuencia a código fuente y viceversa.

Lectura adicional

- Rumbaugh, Jacobson and Booch, “The Unified Modeling Language Reference Manual”
 - Chapters 1, 2,3, 4, 5 y 8
- Perdita Stevens with Rob Pooley, “Using UML”
 - Chapters 1, 2,3, 5, 6, 7, 8 y 9
- Pressman and Maxin , “Software Engineering”
 - Appendix 1: An Introduction to UML
 - Appendix 2: Object-Oriented Concepts

Próxima sesión

- Patrones de diseño