

Programación Orientada a Objetos

Tema 3. Relaciones entre clases. Parte I

José Fidel Argudo Argudo Francisco Palomo Lozano
Inmaculada Medina Buló Gerardo Aburrizaga García



Versión 1.0

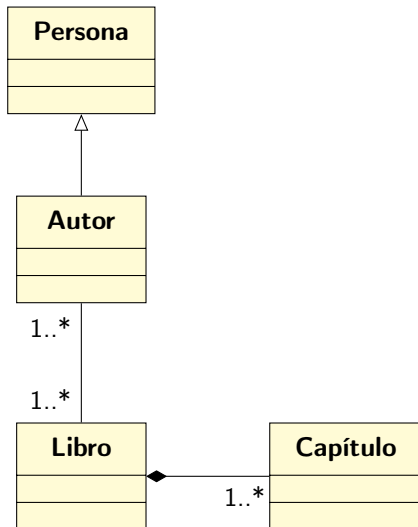


Índice

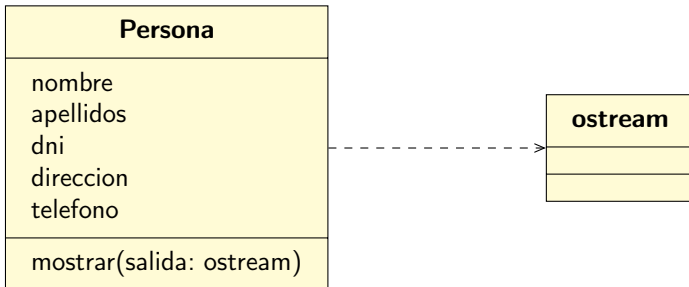
- 1 Descripción general
- 2 Dependencia
- 3 Asociaciones

Descripción general

- Dependencia
- Asociación (cardinalidad, multiplicidad, navegabilidad)
 - Agregación
 - Composición
- Generalización (simple y múltiple)
- Realización



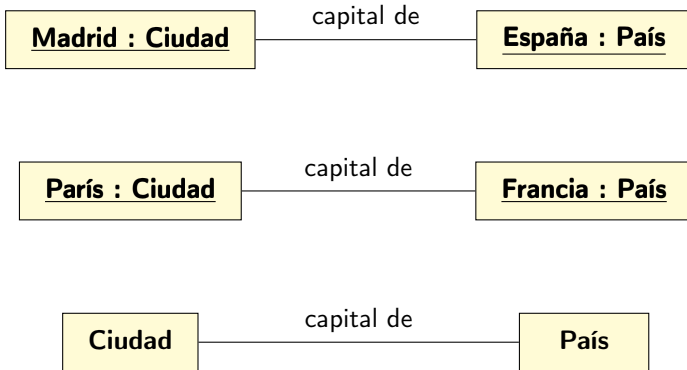
Dependencia



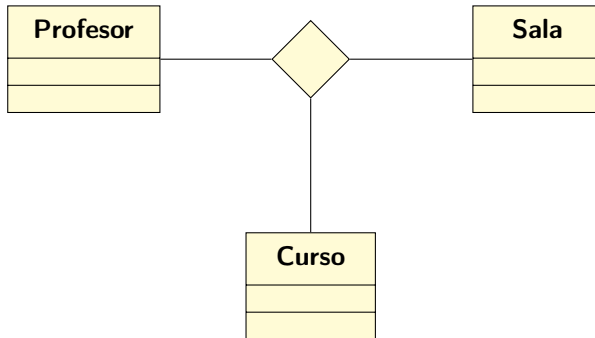
Implementación

No hay que tomar ninguna decisión de diseño al implementar una relación de dependencia. Normalmente, sólo se requiere la inclusión del fichero de cabecera de la clase usada para la definición de la clase dependiente.

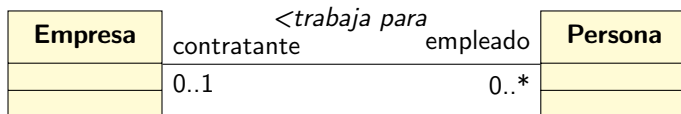
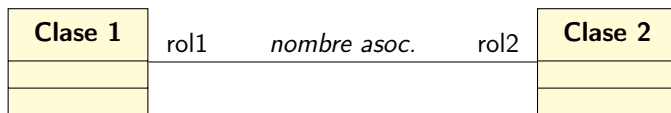
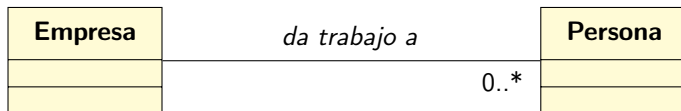
Asociación binaria



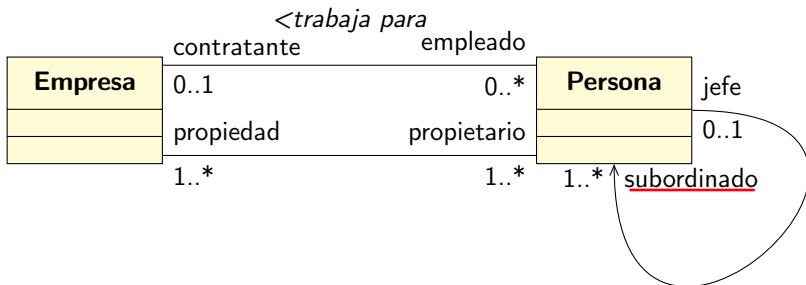
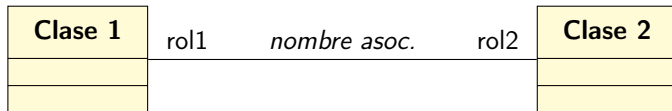
Asociación ternaria



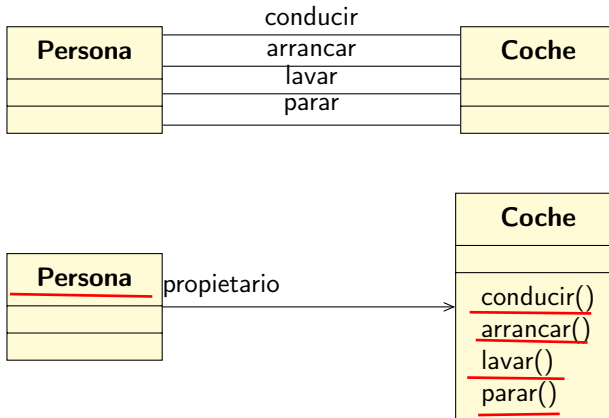
Nombres de función en la asociación



Nombres de función necesarios en una asociación



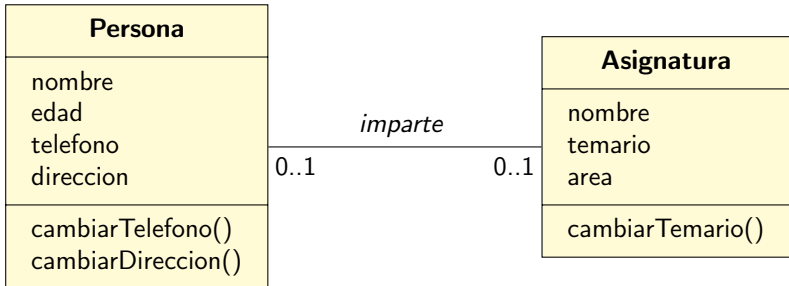
Asociaciones frente a mensajes



Implementación de asociaciones

Alternativas

- 1 Añadir miembros a las clases relacionadas:
 - Atributos que hacen referencia explícita a objetos de la otra clase.
 - Métodos para enlazar los objetos y recorrer los enlaces.
- 2 Clases de asociación.



Asociación 1-a-1 Persona-Asignatura: Esquema básico

```
1 class Persona {
2 public:
3     //...
4     void imparte(Asignatura& asignatura); // enlaza con asignatura
5     Asignatura& imparte() const; // objeto Asignatura enlazado
6 private:
7     //...
8     Asignatura* asignatura; // enlace con objeto Asignatura
9 };

11 class Asignatura {
12 public:
13     //...
14     void impartida(Persona& persona); // enlaza con persona
15     Persona& impartida() const; // objeto Persona enlazado
16 private:
17     //...
18     Persona* persona; // enlace con objeto Persona
19 };
```

Asociación 1-a-1 Persona-Asignatura (persona.h)

```
1 #ifndef PERSONA_H_
2 #define PERSONA_H_
3 #include <string>
4 class Asignatura; // declaración adelantada
5 class Persona {
6 public:
7     Persona(std::string nombre, /* ... */ std::string direccion);
8     // ...
9     void mostrar() const;
10    void imparte(Asignatura& asignatura);
11    Asignatura& imparte() const;
12    void mostrarAsignatura() const;
13 private:
14     std::string nombre;
15     // ...
16     std::string direccion;
17     Asignatura* asignatura;
18 };
19 #endif // PERSONA_H
```

Asociación 1-a-1 Persona-Asignatura (persona.cpp)

```
1  #include "persona.h"
2  #include "asignatura.h"
3  #include <iostream>
4  #include <string>
5  using namespace std;

6  // Constructor
7  Persona::Persona(string nombre, /* ..., */ string direccion):
8      nombre(nombre), /* ..., */ direccion(direccion),
9      asignatura(nullptr)
10 {}

11 // Muestra los datos de una persona
12 void Persona::mostrar() const
13 {
14     cout << "Nombre:_" << nombre << "\n"
15         // ...
16         << "Dirección:_" << direccion << endl;
17 }
```

Asociación 1-a-1 Persona-Asignatura (persona.cpp)

```
21 // Asociación: una persona imparte una asignatura
22 void Persona::imparte(Asignatura& asignatura)
23 {
24     this->asignatura = &asignatura;
25 }

27 Asignatura& Persona::imparte() const
28 {
29     return *asignatura;
30 }

32 // Muestra la asignatura impartida por una persona
33 void Persona::mostrarAsignatura() const
34 {
35     if (!asignatura)
36         cout << "No imparte ninguna asignatura" << endl;
37     else
38         asignatura->mostrar();
39 }
```

Asociación 1-a-1 Persona-Asignatura (asignatura.h)

```
1 #ifndef ASIGNATURA_H_
2 #define ASIGNATURA_H_
3 #include <string>
4 class Persona; // declaración adelantada
5 class Asignatura {
6 public:
7     Asignatura(std::string nombre, /* ..., */ std::string area);
8     // ...
9     void mostrar() const;
10    void impartida(Persona& persona);
11    Persona& impartida() const;
12    void mostrarPersona() const;
13 private:
14     std::string nombre;
15     // ...
16     std::string area;
17     Persona* persona;
18 };
19 #endif // ASIGNATURA_H
```

Asociación 1-a-1 Persona-Asignatura (asignatura.cpp)

```
1  #include "asignatura.h"
2  #include "persona.h"
3  #include <iostream>
4  #include <string>
5  using namespace std;

7  // Constructor
8  Asignatura::Asignatura(string nombre, /* ..., */ string area):
9      nombre(nombre), /* ..., */ area(area),
10     persona(nullptr)
11 {}

13 // Muestra los datos de una asignatura
14 void Asignatura::mostrar() const
15 {
16     cout << "Asignatura:␣" << nombre << "\n"
17         // ...
18         << "Área:␣" << area << endl;
19 }
```


Asociación 1-a-1 Persona-Asignatura (asignatura.cpp)

```
21 // Asociación: una asignatura es impartida por una persona
22 void Asignatura::impartida(Persona& persona)
23 {
24     this->persona = &persona;
25 }

27 Persona& Asignatura::impartida() const
28 {
29     return *persona;
30 }

32 // Muestra la persona que imparte una asignatura
33 void Asignatura::mostrarPersona() const
34 {
35     if (!persona)
36         cout << "No es impartida por ninguna persona" << endl;
37     else
38         persona->mostrar();
39 }
```

Asociación 1-a-1 Persona-Asignatura (prueba.cpp)

```
1  #include "persona.h"
2  #include "asignatura.h"

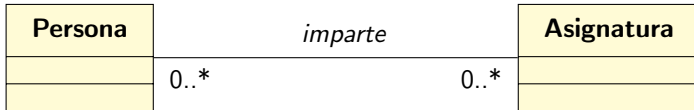
4  int main() {
5      Persona genaro("Genaro_López_Sánchez",
6                      // ...
7                      "C/_Chile_s/n");
8      Asignatura mtp("Metodología_y_Tecnología_de_la_Prog.",
9                      // ...
10                     "Lenguajes_y_Sistemas_Informáticos");
11     genaro.mostrar(); genaro.mostrarAsignatura();
12     // Genaro imparte MTP
13     genaro.imparte(mtp);
14     genaro.mostrar(); genaro.mostrarAsignatura();
15     mtp.mostrar(); mtp.mostrarPersona();
16     // MTP es impartida por Genaro
17     mtp.impartida(genaro);
18     mtp.mostrar(); mtp.mostrarPersona();
19     Persona p(mtp.impartida()); p.mostrar();
20 }
```

Plantilla para asociación 1-a-1

```
1 class A {
2 public:
3     //...
4     void asocia(B&);    // enlaza con objeto B
5     B& asocia() const; // objeto B enlazado
6 private:
7     //...
8     B* b; // enlace con objeto B
9 };

11 class B {
12 public:
13     //...
14     void asocia(A&);    // enlaza con objeto A
15     A& asocia() const; // objeto A enlazado
16 private:
17     //...
18     A* a; // enlace con objeto A
19 };
```

Asoc. varios-a-varios Persona–Asignatura: Esquema básico



```
1 class Persona {
2 public:
3     typedef set<Asignatura*> Asignaturas;
4     // ...
5     void imparte(Asignatura& asignatura); // enlaza con asignatura
6     const Asignaturas& imparte() const;
7     // objetos Asignatura enlazados
8 private:
9     // ...
10    Asignaturas asignaturas; // enlaces con objetos Asignatura
11 };
```

Asoc. varios-a-varios Persona–Asignatura: Esquema básico

```
12 class Asignatura {
13 public:
14     typedef set<Persona*> Personas;
15     //...
16     void impartida(Persona& persona); // enlaza con persona
17     const Personas& impartida() const; // objetos Persona enlazados
18 private:
19     //...
20     Personas personas; // enlaces con objetos Persona
21 };
```

Asoc. varios-a-varios Persona–Asignatura (persona.h)

```
1 #ifndef PERSONA_H_
2 #define PERSONA_H_
3 #include <string>
4 #include <set>
5 class Asignatura; // declaración adelantada
6 class Persona {
7 public:
8     typedef std::set<Asignatura*> Asignaturas;
9     Persona(std::string nombre, /* ... */ std::string direccion);
10    // ...
11    void mostrar() const;
12    void imparte(Asignatura& asignatura);
13    const Asignaturas& imparte() const;
14    void mostrarAsignaturas() const;
15 private:
16    std::string nombre;
17    // ...
18    std::string direccion;
19    Asignaturas asignaturas;
20 };
21 #endif // PERSONA_H
```

Asoc. varios-a-varios Persona–Asignatura (persona.cpp)

```
1  #include "persona.h"
2  #include "asignatura.h"
3  #include <iostream>
4  #include <string>
5  using namespace std;

7  // Constructor
8  Persona::Persona(string nombre, /* ..., */ string direccion):
9      nombre(nombre), /* ..., */ direccion(direccion)
10 {}

12 // Muestra los datos de una persona
13 void Persona::mostrar() const
14 {
15     cout << "Nombre:_" << nombre << "\n"
16         // ...
17         << "Dirección:_" << direccion << endl;
18 }
```

Asoc. varios-a-varios Persona–Asignatura (persona.cpp)

```
20 // Asociación: una persona imparte una asignatura
21 void Persona::imparte(Asignatura& asignatura)
22 { asignaturas.insert(&asignatura); }

24 // Asignaturas impartidas por una persona
25 const Persona::Asignaturas& Persona::imparte() const
26 { return asignaturas; }

28 // Muestra las asignaturas impartidas por una persona
29 void Persona::mostrarAsignaturas() const
30 {
31     if (asignaturas.empty())
32         cout << "No imparte ninguna asignatura" << endl;
33     else
34         for (Persona::Asignaturas::const_iterator
35              i = asignaturas.begin(); i != asignaturas.end(); ++i)
36             (*i)->mostrar();
37 }
```


Asoc. varios-a-varios Persona–Asignatura (asignatura.h)

```
1 #ifndef ASIGNATURA_H_
2 #define ASIGNATURA_H_
3 #include <string>
4 #include <set>
5 class Persona; // declaración adelantada
6 class Asignatura {
7 public:
8     typedef std::set<Persona*> Personas;
9     Asignatura(std::string nombre, /* ..., */ std::string area);
10    // ...
11    void mostrar() const;
12    void impartida(Persona& persona);
13    const Personas& impartida() const;
14    void mostrarPersonas() const;
15 private:
16    std::string nombre;
17    // ...
18    std::string area;
19    Personas personas;
20 };
21 #endif // ASIGNATURA_H
```

Asoc. varios-a-varios Persona–Asignatura (asignatura.cpp)

```
1  #include "asignatura.h"
2  #include "persona.h"
3  #include <iostream>
4  #include <string>
5  using namespace std;

7  // Constructor
8  Asignatura::Asignatura(string nombre, /* ..., */ string area):
9      nombre(nombre), /* ..., */ area(area)
10 {}

12 // Muestra los datos de una asignatura
13 void Asignatura::mostrar() const
14 {
15     cout << "Asignatura:␣" << nombre << "\n"
16         // ...
17         << "Área:␣" << area << endl;
18 }
```

Asoc. varios-a-varios Persona–Asignatura (asignatura.cpp)

```
20 // Asociación: una asignatura es impartida por varias personas
21 void Asignatura::impartida(Persona& persona)
22 { personas.insert(&persona); }

24 // Personas que imparten una asignatura
25 const Asignatura::Personas& Asignatura::impartida() const
26 { return personas; }

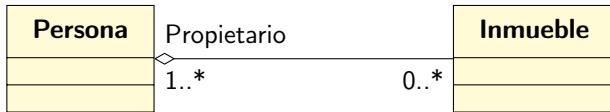
28 // Muestra las personas que imparten una asignatura
29 void Asignatura::mostrarPersonas() const
30 {
31     if (personas.empty())
32         cout << "No es impartida por ninguna persona" << endl;
33     else
34         for (Asignatura::Personas::const_iterator i = personas.begin();
35              i != personas.end(); ++i)
36             (*i)->mostrar();
37 }
```

Plantilla para asociación varios-a-varios

```
1  class A {
2  public:
3      typedef set<B*> Bs;
4      //...
5      void asocia(B&);           // enlaza con objeto B
6      const Bs& asocia() const; // objetos B enlazados
7  private:
8      //...
9      Bs bs; // enlaces con objetos B
10 };

12 class B {
13 public:
14     typedef set<A*> As;
15     //...
16     void asocia(A&);           // enlaza con objeto A
17     const As& asocia() const; // objetos A enlazados
18 private:
19     //...
20     As as; // enlaces con objetos A
21 };
```

Agregación y composición



Implementación de agregaciones

Agregaciones

Se sigue el mismo esquema que con las asociaciones, teniendo en cuenta que normalmente son unidireccionales.

Composiciones

Puesto que la existencia de un objeto componente no tiene sentido independiente de la existencia del agregado, los enlaces en un agregado/compuesto se pueden implementar mediante atributos que sean directamente del tipo de los componentes, en lugar de que sean punteros o referencias a dichos componentes.

Composición y delegación de operaciones


```
1  #ifndef LISTA_H_
2  #define LISTA_H_
3  #include <deque>

5  class Lista {
6  public:
7      bool vacia() const;
8      int primero() const;
9      int ultimo() const;
10     void insertarPrincipio(int e);
11     void insertarFinal(int e);
12     void eliminarPrimero();
13     void eliminarUltimo();
14     void mostrar() const;
15 private:
16     std::deque<int> l;
17 };

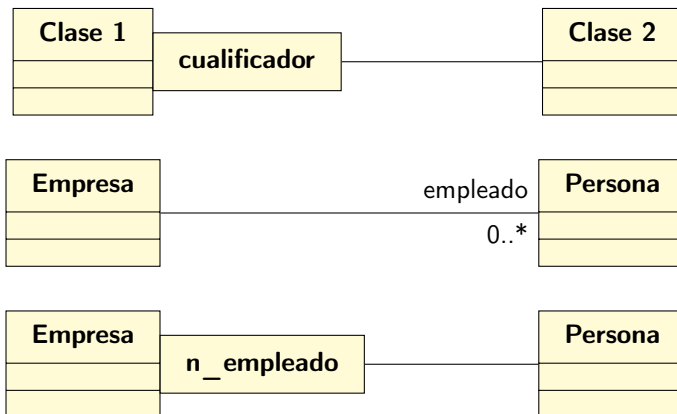
19 #endif // LISTA_H_
```

Composición y delegación de operaciones

```
1  #ifndef PILA_H_
2  #define PILA_H_
3  #include "lista.h"
4  class Pila {
5  public:
6      bool vacia() const;
7      int cima() const;
8      void apilar(int e);
9      void desapilar();
10     void mostrar() const;
11 private:
12     Lista l;
13 };
14 // Delegación de operaciones
15 inline bool Pila::vacia() const { return l.vacia(); }
16 inline int Pila::cima() const { return l.primer(); }
17 inline void Pila::apilar(int e) { l.insertarPrincipio(e); }
18 inline void Pila::desapilar() { l.eliminarPrimer(); }
19 inline void Pila::mostrar() const { l.mostrar(); }
20 #endif // PILA_H_
```

PILA  Lista

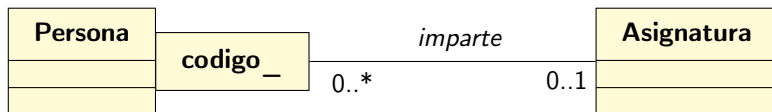
Asociación calificada



Implementación de asociaciones calificadas

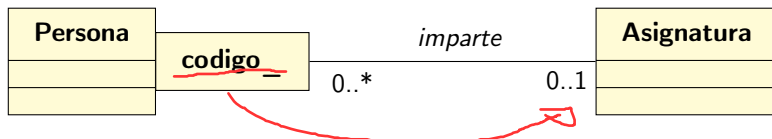
- Si la asociación tiene multiplicidad uno, se puede implementar en forma de diccionario (aplicación entre un conjunto de claves y un conjunto de valores). $MAP \in O(\log n)$
- Si tiene multiplicidad varios, se puede implementar como un diccionario multivalor (aplicación que a cada clave puede asociar más de un valor).

Asoc. varios-a-varios Persona–Asignatura calificada en un extremo: Esquema básico



```
1 class Persona {
2   public:
3     typedef std::map<string, Asignatura*> AsignaturasCalificadas;
4     // ...
5     void impone(Asignatura& asignatura); // enlaza con asignatura
6     const AsignaturasCalificadas& impone() const; // asignaturas
7                                     // enlazadas ordenadas por código
8   private:
9     AsignaturasCalificadas asignaturas; // enlaces calificados con
10                                         // código de asignatura
11 };
```

Asoc. varios-a-varios Persona–Asignatura calificada en un extremo: Esquema básico




```
13 class Asignatura {
14     public:
15     typedef std::set<Persona*> Personas;
16     // ...
17     string codigo() const;
18     void impartida(Persona& persona); // enlaza con persona
19     const Personas& impartida() const; // personas enlazadas
20 private:
21     string codigo_;
22     // ...
23     Personas personas; // enlaces con personas
24 };
```

Asoc. varios-a-varios Persona–Asignatura calificada en un extremo (persona.h)

```
1 #ifndef PERSONA_H_
2 #define PERSONA_H_
3 #include <string>
4 #include <map>

6 class Asignatura; // declaración adelantada

8 class Persona {
9 public:
10     typedef std::map<string, Asignatura *> AsignaturasCalificadas;
11     Persona(std::string nombre, /* ... */ std::string direccion);
12     // ...
13     void mostrar() const;
14     void imparte(Asignatura& asignatura);
15     const AsignaturasCalificadas& imparte() const;
16     void mostrarAsignaturas() const;
```



Asoc. varios-a-varios Persona–Asignatura calificada en un extremo (persona.h)

```
17 private:
18     std::string nombre;
19     // ...
20     std::string direccion;
21     AsignaturasCalificadas asignaturas;
22 };
23 #endif // PERSONA_H
```

Asoc. varios-a-varios Persona–Asignatura calificada en un extremo (persona.cpp)

```
1  #include "persona.h"
2  #include "asignatura.h"
3  #include <iostream>
4  #include <string>
5  using namespace std;

7  // Constructor
8  Persona::Persona(string nombre, /* ..., */ string direccion):
9      nombre(nombre), /* ..., */ direccion(direccion)
10 {}

12 // Muestra los datos de una persona
13 void Persona::mostrar() const
14 {
15     cout << "Nombre:_" << nombre << "\n"
16         // ...
17         << "Dirección:_" << direccion << endl;
18 }
```

Asoc. varios-a-varios Persona–Asignatura calificada en un extremo (persona.cpp)

```

20 // Asociación: una persona imparte una asignatura
21 void Persona::imparte(Asignatura& asignatura)
22 { asignaturas.insert(make_pair(asignatura.codigo(),
23                               &asignatura)); }

25 // Asignaturas impartidas por una persona
26 const Persona::AsignaturasCalificadas& Persona::imparte() const
27 { return asignaturas; }

29 // Muestra las asignaturas impartidas por una persona
30 void Persona::mostrarAsignaturas() const {
31     if (asignaturas.empty())
32         cout << "No imparte ninguna asignatura" << endl;
33     else
34         for (Persona::AsignaturasCalificadas::const_iterator
35             i = asignaturas.begin(); i != asignaturas.end(); ++i)
36             (i->second)->mostrar();
37 }

```

Handwritten notes:

- Red checkmark and "ej" next to line 21.
- Red "Bucle de rangos" (range loop) next to line 35.
- Red "AUTO i: iter" (AUTO i: iterator) next to line 35.
- Red "for(pair<string, Asignatura*> a: asignaturas) xa.second->mostrar();" next to line 36.

Asoc. varios-a-varios Persona–Asignatura calificada en un extremo (asignatura.h)

```
1 #ifndef ASIGNATURA_H_
2 #define ASIGNATURA_H_
3 #include <string>
4 #include <set>

6 class Persona; //declaración adelantada

8 class Asignatura {
9 public:
10     typedef std::set<Persona*> Personas;
11     Asignatura(std::string codigo, /* ..., */ std::string area);
12     // ...
13     string codigo() const;
14     void mostrar() const;
15     void impartida(Persona& persona);
16     const Personas& impartida() const;
17     void mostrarPersonas() const;
```

Asoc. varios-a-varios Persona–Asignatura calificada en un extremo (asignatura.h)

```
18 private:
19     std::string codigo_;
20     // ...
21     std::string area;
22     Personas personas;
23 };
24 #endif // ASIGNATURA_H
```

Asoc. varios-a-varios Persona–Asignatura calificada en un extremo (asignatura.cpp)

```
1 #include "asignatura.h"
2 #include "persona.h"
3 #include <iostream>
4 #include <string>
5 using namespace std;
6 // Constructor
7 Asignatura::Asignatura(string codigo, /* ..., */ string area):
8     codigo(codigo), /* ..., */ area(area) {}

10 // Devuelve el código de la asignatura
11 string codigo() const { return codigo_; }

13 // Muestra los datos de una asignatura
14 void Asignatura::mostrar() const {
15     cout << "Asignatura:␣" << nombre << "\n"
16         // ...
17         << "Área:␣" << area << endl;
18 }
```

Asoc. varios-a-varios Persona–Asignatura calificada en un extremo (asignatura.cpp)

```
20 // Asociación: una asignatura es impartida por varias personas
21 void Asignatura::impartida(Persona& persona)
22 { personas.insert(&persona); }

24 // Personas que imparten una asignatura
25 const Asignatura::Personas& Asignatura::impartida() const
26 { return personas; }

28 // Muestra las personas que imparten una asignatura
29 void Asignatura::mostrarPersonas() const
30 {
31     if (personas.empty())
32         cout << "No es impartida por ninguna persona" << endl;
33     else
34         for (Asignatura::Personas::const_iterator
35              i = personas.begin(); i != personas.end(); ++i)
36             (*i)->mostrar();
37 }
```

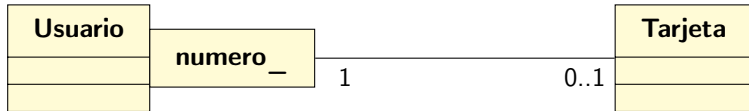
Plantilla para asociación varios-a-varios calificada en un extremo

```
1 class A {  
2 public:  
3     typedef map<C, B*> BsCalificadas;  
4     // ...  
5     void asocia(B&); // enlaza con objeto B  
6     const BsCalificadas& asocia() const; // objetos B enlazados  
7 private:  
8     BsCalificadas bs; // enlaces con objetos B  
9 };
```

Plantilla para asociación varios-a-varios calificada en un extremo

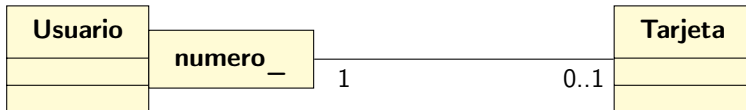
```
11 class B {  
12 public:  
13   typedef set<A*> As;  
14   //...  
15   void asocia(A&);           // enlaza con objeto A  
16   const As& asocia() const; // objetos A enlazados  
17 private:  
18   C c;           // calificador  
19   //...  
20   As as;         // enlaces con objetos A  
21 };
```

Asoc. varios-a-varios Usuario–Tarjeta calificada en un extremo



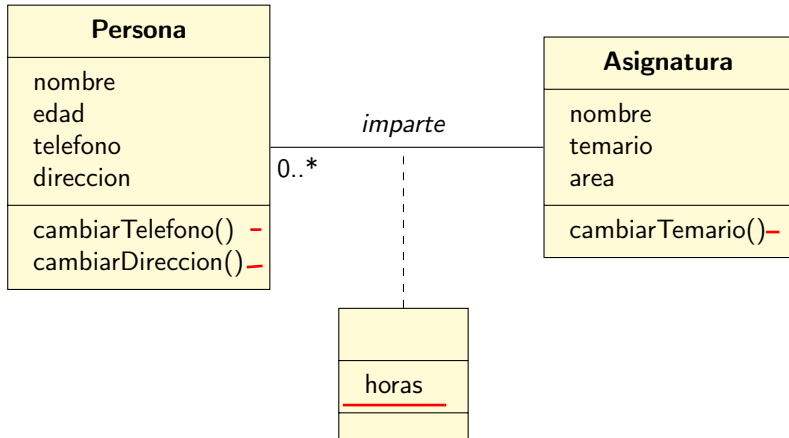
1 `class Usuario {`
 2 `public:`
 3 `typedef std::map<Numero, Tarjeta*> Tarjetas;`
 4 `//...`
 5 `void es_titular_de(Tarjeta&); // enlaza con tarjeta`
 6 `void no_es_titular_de(Tarjeta&); // desenlaza tarjeta`
 7 `const Tarjetas& tarjetas() const; // tarjetas del usuario`
 8 `// ordenadas por número`
 9 `~Usuario(); // destruye todas las tarjetas del usuario`
 10 `private:`
 11 `// ...`
 12 `Tarjetas tarjetas_; // enlaces con tarjetas mediante número`
 13 `};`

Asoc. varios-a-varios Usuario–Tarjeta calificada en un extremo

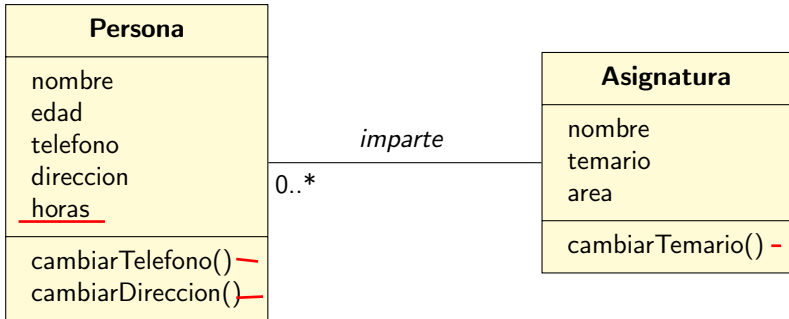


```
1 class Tarjeta {
2 public:
3   Tarjeta(const Numero&, Usuario& /*, ...*/); // enlaza con su titular
4   ~Tarjeta(); // desenlaza la tarjeta de su titular
5   // ...
6   Numero numero() const;
7   const Usuario* titular() const; // usuario enlazado
8 private:
9   // ...
10  Numero numero_;
11  const Usuario* titular_; // enlace con titular
12 };
```

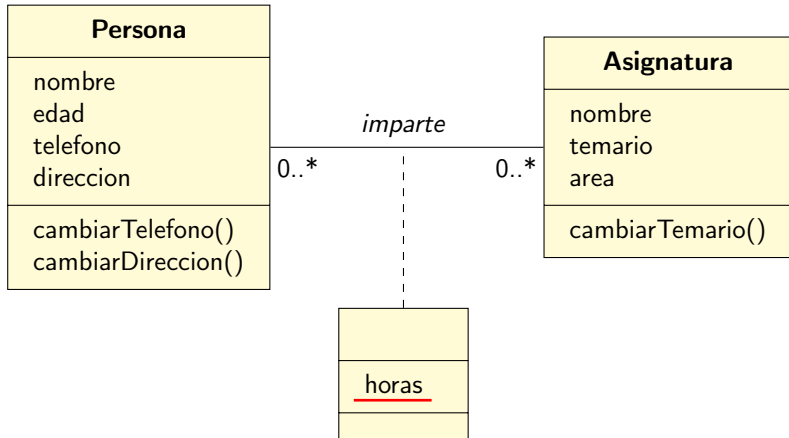

Atributos de enlace



Sin atributos de enlace



Atributos de enlace necesarios



Asoc. varios-a-varios Persona–Asignatura con atributos de enlace: Esquema básico

```
1 class Persona {  
2 public:  
3     typedef map<Asignatura*, int> Asignaturas;  
4     //...  
5     void imparte(Asignatura& asignatura, int horas);  
6     const Asignaturas& imparte() const;  
7 private:  
8     //...  
9     Asignaturas asignaturas; // enlaces y sus atributos  
10 };
```

Asoc. varios-a-varios Persona–Asignatura con atributos de enlace: Esquema básico

```
12 class Asignatura {  
13 public:  
14     typedef map<Persona*, int> Personas;  
15     // ...  
16     void impartida(Persona& persona, int horas);  
17     const Personas& impartida() const;  
18 private:  
19     // ...  
20     Personas personas; // enlaces y sus atributos  
21 };
```

Asoc. varios-a-varios Persona–Asignatura con atributos de enlace (persona.h)

```
1  #ifndef PERSONA_H_
2  #define PERSONA_H_
3  #include <string>
4  #include <map>

6  class Asignatura; // declaración adelantada

8  class Persona {
9  public:
10     typedef std::map<Asignatura *, int> Asignaturas;
11     Persona(std::string nombre, /* ... */ std::string direccion);
12     // ...
13     void mostrar() const;
14     void imparte(Asignatura& asignatura, int horas);
15     const Asignaturas& imparte() const;
16     void mostrarAsignaturas() const;
```

Asoc. varios-a-varios Persona–Asignatura con atributos de enlace (persona.h)

```
17 private:
18     std::string nombre;
19     // ...
20     std::string direccion;
21     Asignaturas asignaturas;
22 };
23 #endif // PERSONA_H
```

Asoc. varios-a-varios Persona–Asignatura con atributos de enlace (persona.cpp)

```
1  // ...

3  void Persona::imparte(Asignatura& asignatura, int horas)
4  { asignaturas.insert(std::make_pair(&asignatura, horas)); }

6  const Persona::Asignaturas& Persona::imparte() const
7  { return asignaturas; }


9  void Persona::mostrarAsignaturas() const {
10     if (asignaturas.empty())
11         cout << "No se imparte ninguna asignatura" << endl;
12     else
13         for (Persona::Asignaturas::const_iterator
14             i = asignaturas.begin(); i != asignaturas.end(); ++i) {
15             (i->first)->mostrar();
16             cout << "con " << i->second
17                 << " horas semanales" << endl;
18         }
19 }
```


Asoc. varios-a-varios Persona–Asignatura con atributos de enlace (asignatura.h)

```
1 #ifndef ASIGNATURA_H_
2 #define ASIGNATURA_H_
3 #include <string>
4 #include <map>

6 class Persona; // declaración adelantada

8 class Asignatura {
9 public:
10     typedef std::map<Persona*, int> Personas;
11     Asignatura(std::string nombre, /* ..., */ std::string area);
12     // ...
13     void mostrar() const;
14     void impartida(Persona& persona, int horas);
15     const Personas& impartida() const;
16     void mostrarPersonas() const;
```

 u otra clase, si fuera una clase de atributos de enlace

Asoc. varios-a-varios Persona–Asignatura con atributos de enlace (asignatura.h)

```
17 private:
18     std::string nombre;
19     // ...
20     std::string area;
21     Personas personas;
22 };
23 #endif // ASIGNATURA_H
```

Asoc. varios-a-varios Persona–Asignatura con atributos de enlace (asignatura.cpp)

```
1 // ...

3 void Asignatura::impartida(Persona& persona, int horas)
4 { personas.insert(std::make_pair(&persona, horas)); }

6 const Asignatura::Personas& Asignatura::impartida() const
7 { return personas; }

9 void Asignatura::mostrarPersonas() const {
10     if (personas.empty())
11         cout << "No se impartida por ninguna persona" << endl;
12     else
13         for (Asignatura::Personas::const_iterator
14             i = personas.begin(); i != personas.end(); ++i) {
15             (i->first)->mostrar();
16             cout << "con" << i->second
17                 << " horas semanales" << endl;
18         }
19 }
```

Asoc. varios-a-varios Persona–Asignatura con atributos de enlace (prueba.cpp)

```
1 #include <iostream>
2 #include "persona.h"
3 #include "asignatura.h"
4 int main() {
5     Persona marisa("Marisa", /* ... */ "C/_Argentina_s/n");
6     Asignatura ada("ADAI", /* ... */ "LSI");
7     Asignatura poo("POO", /* ... */ "LSI");

9     marisa.imparte(ada, 5); ada.impartida(marisa, 5);
10    marisa.imparte(poo, 8); poo.impartida(marisa, 8);
11    marisa.mostrarAsignaturas();
12 }
```

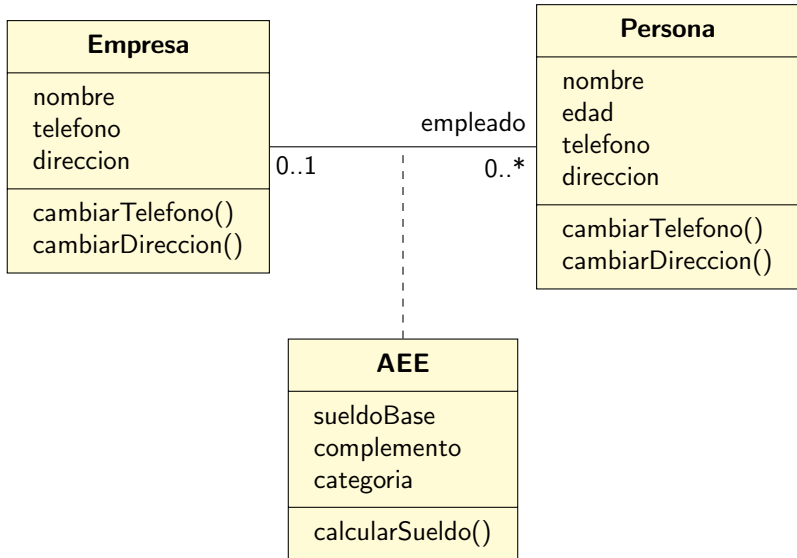
Plantilla para asociación varios-a-varios con atributos de enlace

```
1 class A {  
2 public:  
3     typedef map<B*, C> Bs;  
4     //...  
5     void asocia(B&, C);    // enlaza con objeto B y atributo C  
6     const Bs& asocia() const; // objetos B enlazados  
7 private:  
8     //...  
9     Bs bs; // enlaces y sus atributos  
10 };
```

Plantilla para asociación varios-a-varios con atributos de enlace

```
12 class B {  
13 public:                                Podemos sustituir C por C* para llamar directamente al objeto C  
14     typedef map<A*, C> As;  
15     //...  
16     void asocia( A&, C);    // enlaza con objeto A y atributo C  
17     const As& asocia() const; // objetos A enlazados  
18 private:  
19     //...  
20     As as; // enlaces y sus atributos  
21 };
```

Clases de asociación



Clase de asociación Empresa–Persona (AEE.h)

```
1 #include <map>
2 #include "empresa.h"
3 #include "persona.h"

5 class Salarío {
6 public:
7     // ...
8     calcularSueldo();
9 private:
10     double sueldoBase,
11           complemento;
12     int categoria;
13 };
```

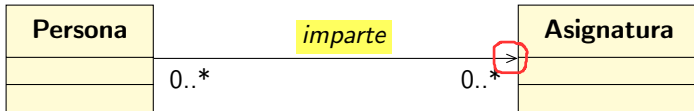
No aparece en el diagrama de clases
Nos la inventamos para guardar los
atributos de la clase AEE

Clase de asociación Empresa–Persona (AEE.h)

```
15 class AEE {
16 public:
17 → void asocia(Empresa& e, Persona& p);
18 → void asocia(Persona& p, Empresa& e);
19 const std::map<Persona*, Salario*>* asocia(Empresa& y) const;
20 // Devuelve un puntero nulo si y no tiene empleados
21 const std::pair<Empresa*, Salario*>* asocia(Persona& x) const;
22 // Devuelve un puntero nulo si x no es empleado de empresa alguna
23 private:
24     typedef std::map<Empresa*, std::map<Persona*, Salario*> > AD;
25     typedef std::map<Persona*, std::pair<Empresa*, Salario*> > AI;
26     AD empresa-empleado;
27     AI empleado-empresa;
28 };
```

2 diccionarios para una relacion bidireccional

Clase de asociación Persona–Asig. (persona-asignatura-1.h)



```
1 #include <map>
2 #include "persona.h"
3 #include "asignatura.h"

5 class PersonaImparteAsignatura {
6 public:
7     void asocia(Persona& p, Asignatura& a);
8     void mostrarAsignaturas(Persona& p) const;
9 private:
10    bool esta(Persona& p, Asignatura& a) const;
11    typedef std::multimap<Persona*, Asignatura*> A;
12    typedef A::const_iterator I;
13    A asociacion;
14 };
```

Clase de asoc. Persona-Asig. (persona-asignatura-1.cpp)

```
1  #include <iostream>
2  #include "persona-asignatura-1.h"

4  bool
5  PersonaImparteAsignatura::esta(Persona& p, Asignatura& a) const
6  { std::pair<I, I> rango = asociacion.equal_range(&p);
7    for (I i = rango.first; i != rango.second; ++i)
8      if (i->second == &a) return true;
9    return false; } *i es el par persona asignatura
                       i->second es el segundo valor del par

11 void PersonaImparteAsignatura::asocia(Persona& p, Asignatura& a)
12 { if (!esta(p, a)) asociacion.insert(std::make_pair(&p, &a)); }

14 void
15 PersonaImparteAsignatura::mostrarAsignaturas(Persona& p) const
16 { std::pair<I, I> rango = asociacion.equal_range(&p);
17   if (rango.first == rango.second)
18     std::cout << "No imparte ninguna asignatura." << endl;
19   else for (I i = rango.first; i != rango.second; ++i)
20     (i->second)->mostrar(); }
```

Clase de asociación Persona–Asig. (prueba.cpp)

```
1  #include <iostream>
2  #include "persona-asignatura-1.h"

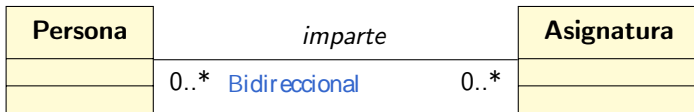
4  int main() {
5      Persona genaro("Genaro_López_Sánchez", "C/_Chile_s/n");
6      Persona felisa("Felisa_González_Pérez", "C/_Bolivia_s/n");
7      Asignatura mtp("MTP", "LSI");
8      Asignatura poo("POO", "LSI");

10     PersonaImparteAsignatura d;
11     d.asocia(genaro, mtp); // genaro.imparte(mtp);
12     d.asocia(felisa, poo); // felisa.imparte(poo);

14     std::cout << "Listado_de_asignaturas_que_imparte_cada_persona\n"
15                 << "-----\n";
16     genaro.mostrar();
17     d.mostrarAsignaturas(genaro);
18     felisa.mostrar();
19     d.mostrarAsignaturas(felisa);
20 }
```

aquí guardaremos los enlaces persona asig

Clase de asociación Persona–Asignatura



Clase de asociación Persona–Asig. (persona-asignatura-2.h)

```
1  #include <map>
2  #include "persona.h"
3  #include "asignatura.h"

5  class PersonaImparteAsignatura {
6  public:
7      void asocia(Persona& p, Asignatura& a);
8      void asocia(Asignatura& a, Persona& p);
9      void mostrarPersonas(Asignatura& a) const;
10     void mostrarAsignaturas(Persona& p) const;
11 private:
12     bool esta(Persona& x, Asignatura& y) const;
13     typedef std::multimap<Persona*, Asignatura*> AD;
14     typedef std::multimap<Asignatura*, Persona*> AI;
15     typedef AD::const_iterator ID;
16     typedef AI::const_iterator II; } Por comodidad
17     AD directa;
18     AI inversa;
19 };
```

Clase de asoc. Persona-Asig. (persona-asignatura-2.cpp)

```
1  #include <iostream>
2  #include "persona-asignatura-2.h"

4  bool
5  PersonaImparteAsignatura::esta(Persona& p, Asignatura& a) const
6  { std::pair<PersonaImparteAsignatura::ID,
7      PersonaImparteAsignatura::ID>
8      rango = directa.equal_range(&p);
9      for (PersonaImparteAsignatura::ID
10         i = rango.first; i != rango.second; ++i)
11         if (i->second == &a) return true;
12     return false;
13 }

15 void PersonaImparteAsignatura::asocia(Persona& p, Asignatura& a)
16 { if (!esta(p, a)) {
17     directa.insert(std::make_pair(&p, &a));
18     inversa.insert(std::make_pair(&a, &p));
19 }
20 }
```

dobles enlaces
persona asignat
asignat persona

Clase de asoc. Persona-Asig. (persona-asignatura-2.cpp)

```
22 void PersonaImparteAsignatura::asocia(Asignatura& a, Persona& p)
23 { associa(p, a); }

25 void
26 PersonaImparteAsignatura::mostrarAsignaturas(Persona& p) const
27 {
28     std::pair<PersonaImparteAsignatura::ID,
29             PersonaImparteAsignatura::ID>
30         rango = directa.equal_range(&p);
31     if (rango.first == rango.second)
32         std::cout << "No imparte ninguna asignatura." << endl;
33     else
34         for (PersonaImparteAsignatura::ID
35             i = rango.first; i != rango.second; ++i)
36             (i->second)->mostrar();
37 }
```


Clase de asoc. Persona-Asig. (persona-asignatura-2.cpp)

```
39 void
40 PersonaImparteAsignatura::mostrarPersonas(Asignatura& a) const
41 {
42     std::pair<PersonaImparteAsignatura::II,
43             PersonaImparteAsignatura::II>
44         rango = inversa.equal_range(&a);
45     if (rango.first == rango.second)
46         std::cout << "No es impartida por ninguna persona." << endl;
47     else
48         for (PersonaImparteAsignatura::II
49             i = rango.first; i != rango.second; ++i)
50             (i->second)->mostrar();
51 }
```

Clase de asociación Persona–Asig. (persona-asignatura-3.h)

```
1 #include <map>
2 #include <set>
3 #include "persona.h"
4 #include "asignatura.h"

6 class PersonaImparteAsignatura {
7 public:
8     void asocia(Persona& p, Asignatura& a);
9     void asocia(Asignatura& a, Persona& p);
10    void mostrarPersonas(Asignatura& a) const;
11    void mostrarAsignaturas(Persona& p) const;
12 private:
13     std::map<Persona*, std::set<Asignatura*> > directa;
14     std::map<Asignatura*, std::set<Persona*> > inversa;
15 };
```

Clase de asoc. Persona-Asig. (persona-asignatura-3.cpp)

```
1  #include <iostream>
2  #include "persona-asignatura-3.h"

4  void PersonaImparteAsignatura::asocia(Persona& p, Asignatura& a)
5  { directa[&p].insert(&a); inversa[&a].insert(&p); }

7  void PersonaImparteAsignatura::asocia(Asignatura& a, Persona& p)
8  { asocia(p, a); }

10 void PersonaImparteAsignatura::mostrarAsignaturas(Persona& p) const
11 { std::set<Asignatura*>::const_iterator i;
12   for (i = directa[&p].begin(); i != directa[&p].end(); ++i)
13     (*i)->mostrar();
14 }

16 void PersonaImparteAsignatura::mostrarPersonas(Asignatura& a) const
17 { std::set<Persona*>::const_iterator i;
18   for (i = inversa[&a].begin(); i != inversa[&a].end(); ++i)
19     (*i)->mostrar();
20 }
```

Clase de asociación Persona–Asig. (prueba.cpp)

```
1  int main()
2  {
3      Persona genaro("Genaro_López_Sánchez", "C/_Chile_s/n");
4      Persona marisa("Marisa_Gómez_Jiménez", "C/_Argentina_s/n");
5      Asignatura mtp("MTP", "LSI");
6      Asignatura poo("POO", "LSI");
7      PersonaImparteAsignatura d;
8      d.asocia(genaro, mtp); // d.asocia(mtp, genaro);
9      d.asocia(poo, genaro);
10     d.asocia(poo, felisa);
11     cout << "Listado_de_asignaturas_que_imparte_cada_persona\n"
12           << "-----\n";
13     genaro.mostrar(); d.mostrarAsignaturas(genaro);
14     felisa.mostrar(); d.mostrarAsignaturas(felisa);
15     cout << "Listado_de_personas_que_imparten_cada_asignatura\n"
16           << "-----\n";
17     mtp.mostrar(); d.mostrarPersonas(mtp);
18     poo.mostrar(); d.mostrarPersonas(poo);
19 }
```

Clase de asociación Persona–Asig. (persona-asignatura.h)

```
1 #include <map>
2 #include <set>
3 #include "persona.h"
4 #include "asignatura.h"

6 class PersonaImparteAsignatura {
7 public:
8     void asocia(Persona& p, Asignatura& a);
9     void asocia(Asignatura& a, Persona& p);
10     std::set<Asignatura*> asociados(Persona& x) const;
11     std::set<Persona*> asociados(Asignatura& y) const;
12 private:
13     std::map<Persona*, std::set<Asignatura*> > directa;
14     std::map<Asignatura*, std::set<Persona*> > inversa;
15 };
```

Clase de asoc. Persona–Asignatura (persona-asignatura.cpp)

```
1  #include <iostream>
2  #include "persona_asignatura.h"
3  using namespace std;

5  void PersonaImparteAsignatura::asocia(Persona& p, Asignatura& a)
6  {
7      directa[&p].insert(&a);
8      inversa[&a].insert(&p);
9  }

11 void PersonaImparteAsignatura::asocia(Asignatura& a, Persona& p)
12 {  asocia(p, a); }
```

Clase de asoc. Persona–Asignatura (persona-asignatura.cpp)

```
14 set<Asignatura*>
15 PersonaImparteAsignatura::asociados(Persona& p) const
16 {
17     map<Persona*, set<Asignatura*>>::
18     const_iterator i = directa.find(&p);
19     if (i != directa.end())
20         return i->second;
21     else
22         return set<Asignatura*>();
23 }
24 set<Persona*>
25 PersonaImparteAsignatura::asociados(Asignatura& a) const
26 {
27     map<Asignatura*, set<Persona*>>::
28     const_iterator i = inversa.find(&a);
29     if (i != inversa.end())
30         return i->second;
31     else
32         return set<Persona*>();
33 }
```

Diagram illustrating the flow of data in the first function:

- The parameter `p` (highlighted in yellow) is passed to the `find` method of the `map` (highlighted in yellow).
- The `const_iterator` `i` (highlighted in yellow) is used to access the `second` member of the map (highlighted in yellow).
- The `const` keyword (highlighted in yellow) is used to declare the function as constant.
- The `const` keyword (highlighted in yellow) is used to declare the function as constant.

Clase de asociación Persona–Asignatura (prueba.cpp)

```
1  template <typename T>
2  ostream& operator <<(ostream& fs, const set<T*>& c)
3  {  typename set<T*>::const_iterator i;
4     for (i = c.begin(); i != c.end(); ++i)
5         fs << **i;
6     return fs;
7  }

9  ostream& operator <<(ostream& fs, const Persona& p)
10 {  p.mostrar(fs); return fs; }

12 ostream& operator <<(ostream& fs, const Asignatura& a)
13 {  a.mostrar(fs); return fs; }

15 int main() {
16     Persona genaro("Genaro_López_Sánchez", "C/_Chile_s/n");
17     Persona marisa("Marisa_Gómez_Jiménez", "C/_Argentina_s/n");
18     Persona felisa("Felisa_González_Pérez", "C/_Bolivia_s/n");

20     Asignatura mtp("MTP", "LSI");
```

****i**
un asterisco para el puntero
y otro es el del iterador

Clase de asociación Persona–Asignatura (prueba.cpp)

```
21  Asignatura ada("ADA", "LSI");
22  Asignatura poo("POO", "LSI");

24  PersonaImparteAsignatura d;
25  d.asocia(genaro, mtp);
26  d.asocia(mtp, genaro);
27  d.asocia(poo, genaro);
28  d.asocia(poo, felisa);

30  cout << "Listado de asignaturas que imparte cada persona\n"
31       << "-----\n";
32  cout << genaro << d.asociados(genaro) << endl;
33  cout << marisa << d.asociados(marisa) << endl;
34  cout << felisa << d.asociados(felisa) << endl;
35  cout << "Listado de personas que imparten cada asignatura\n"
36       << "-----\n";
37  cout << mtp << d.asociados(mtp) << endl;
38  cout << ada << d.asociados(ada) << endl;
39  cout << poo << d.asociados(poo) << endl;
40 }
```

Clase de asociación Persona–Asignatura (prueba.cpp)

Salida del programa

```
1 Listado de asignaturas que imparte cada persona
2 -----
3 Nombre: Genaro López Sánchez
4 Dirección: C/ Chile s/n
5 Asignatura: P00
6 Área: LSI
7 Asignatura: MTP
8 Área: LSI

10 Nombre: Marisa Gómez Jiménez
11 Dirección: C/ Argentina s/n

13 Nombre: Felisa González Pérez
14 Dirección: C/ Bolivia s/n
15 Asignatura: P00
16 Área: LSI
```

Clase de asociación Persona–Asignatura (prueba.cpp)

Salida del programa (cont.)

```
17 Listado de personas que imparten cada asignatura
18 -----
19 Asignatura: MTP
20 Área: LSI
21 Nombre: Genaro López Sánchez
22 Dirección: C/ Chile s/n

24 Asignatura: ADA
25 Área: LSI

27 Asignatura: P00
28 Área: LSI
29 Nombre: Felisa González Pérez
30 Dirección: C/ Bolivia s/n
31 Nombre: Genaro López Sánchez
32 Dirección: C/ Chile s/n
```

Clase genérica de asociación (asoc.h)

Plantilla bidireccional

```
1  #include <map>
2  #include <set>

4  // Clase genérica de asociación bidireccional .

6  template <typename X, typename Y>
7  class AsociacionBidireccional {
8  public:
9      void asocia(X& x, Y& y);
10     void asocia(Y& y, X& x);
11     std::set<Y*> asociados(X& x) const;
12     std::set<X*> asociados(Y& y) const;
13 private:
14     std::map<X*, std::set<Y*> > directa;
15     std::map<Y*, std::set<X*> > inversa;
16 };
```

Clase genérica de asociación (asoc.h)

```
18 // Asocia bidireccionalmente dos objetos.
19 template <typename X, typename Y>
20 void AsociacionBidireccional<X, Y>::asocia(X& x, Y& y)
21 {
22     directa[&x].insert(&y);
23     inversa[&y].insert(&x);
24 }

26 template <typename X, typename Y>
27 inline void AsociacionBidireccional<X, Y>::asocia(Y& y, X& x)
28 { associa(x, y); }
```

Clase genérica de asociación (asoc.h)

```
30 // Devuelve el conjunto de enlaces asociados a un objeto.
31 template <typename X, typename Y>
32 std::set<Y*> AsociacionBidireccional<X, Y>::asociados(X& x) const
33 {
34     typename std::map<X*, std::set<Y*>>::
35         const_iterator i = directa.find(&x);
36     if (i != directa.end()) return i->second;
37     else return std::set<Y*>();
38 }

40 template <typename X, typename Y>
41 std::set<X*> AsociacionBidireccional<X, Y>::asociados(Y& y) const
42 {
43     typename std::map<Y*, std::set<X*>>::
44         const_iterator i = inversa.find(&y);
45     if (i != inversa.end()) return i->second;
46     else return std::set<X*>();
47 }
```

Clase genérica de asociación (prueba2.cpp)

```
1  int main() {
2      Persona genaro("Genaro_López_Sánchez", "C/_Chile_s/n");
3      Persona marisa("Marisa_Gómez_Jiménez", "C/_Argentina_s/n");
4      Asignatura mtp("MTP", "LSI");
5      Asignatura poo("POO", "LSI");

7      AsociacionBidireccional<Persona, Asignatura> imparte;
8      imparte.asocia(genaro, mtp);
9      imparte.asocia(poo, genaro);
10     imparte.asocia(poo, marisa);

12     cout << "Listado_de_asignaturas_que_imparte_cada_persona\n";
13         << "-----\n";
14     cout << genaro << imparte.asociados(genaro) << endl;
15     cout << marisa << imparte.asociados(marisa) << endl;
16     cout << "Listado_de_personas_que_imparten_cada_asignatura\n";
17         << "-----\n";
18     cout << mtp << imparte.asociados(mtp) << endl;
19     cout << poo << imparte.asociados(poo) << endl;
20 }
```

Clase genérica de asociación (prueba2.cpp)

Salida del programa

```
1 Listado de asignaturas que imparte cada persona
2 -----
3 Nombre: Genaro López Sánchez
4 Dirección: C/ Chile s/n
5 Asignatura: P00
6 Área: LSI
7 Asignatura: MTP
8 Área: LSI

10 Nombre: Marisa Gómez Jiménez
11 Dirección: C/ Argentina s/n
12 Asignatura: P00
13 Área: LSI
```

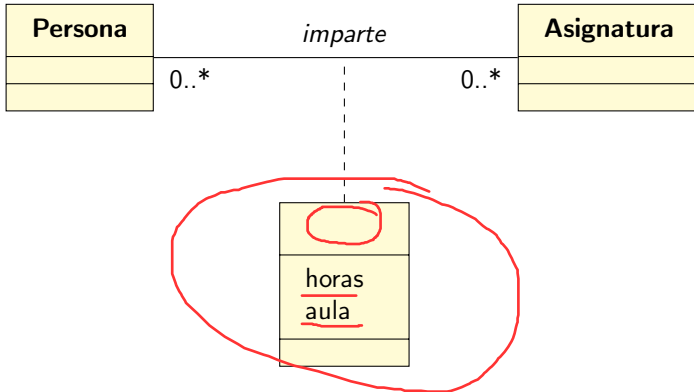

Clase genérica de asociación (prueba2.cpp)

Salida del programa (cont.)

```
14 Listado de personas que imparten cada asignatura
15 -----
16 Asignatura: MTP
17 Área: LSI
18 Nombre: Genaro López Sánchez
19 Dirección: C/ Chile s/n

21 Asignatura: P00
22 Área: LSI
23 Nombre: Marisa Gómez Jiménez
24 Dirección: C/ Argentina s/n
25 Nombre: Genaro López Sánchez
26 Dirección: C/ Chile s/n
```

Clase de asoc. Persona–Asignatura con atributos de enlace

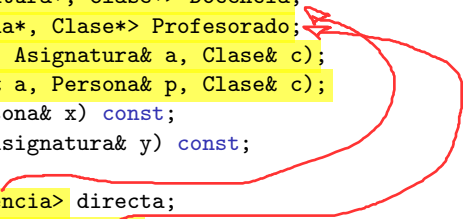


Clase de asociación Persona–Asignatura con atributos de enlace (persona_asignatura.h)

```
1 #include <map>
2 #include "persona.h"
3 #include "asignatura.h"
4 #include "clase.h"

6 class Persona_Asignatura {
7 public:
8     typedef std::map<Asignatura*, Clase*> Docencia;
9     typedef std::map<Persona*, Clase*> Profesorado;
10    void asocia(Persona& p, Asignatura& a, Clase& c);
11    void asocia(Asignatura& a, Persona& p, Clase& c);
12    Docencia asociados(Persona& x) const;
13    Profesorado asociados(Asignatura& y) const;
14 private:
15    std::map<Persona*, Docencia> directa;
16    std::map<Asignatura*, Profesorado> inversa;
17 };
```

Map de A y Map de P y C



Clase de asociación Persona–Asignatura con atributos de enlace (clase.h)

```
1  //-----
2  // clase.h
3  // Definición de la clase de los atributos
4  // de enlace.
5  //-----
6  #include <string>

8  class Clase {
9  public:
10     //...
11 private:
12     int horas;
13     string aula;
14 };
```

Clase de asociación Persona–Asignatura con atributos de enlace (persona_asignatura.cpp)

```
1 void
2 Persona_Asignatura::asocia(Persona& p, Asignatura& a, Clase& c)
3 {
4     directa[&p].insert(std::make_pair(&a, &c));
5     inversa[&a].insert(std::make_pair(&p, &c));
6 }

8 void
9 Persona_Asignatura::asocia(Asignatura& a, Persona& p, Clase& c)
10 { associa(p, a, c); }
```

Clase de asociación Persona–Asignatura con atributos de enlace (persona_asignatura.cpp)

```
12 Persona_Asignatura::Docencia
13 Persona_Asignatura::asociados(Persona& p) const
14 {
15     std::map<Persona*, Docencia>::
16     const_iterator i = directa.find(&p);
17     if (i != directa.end()) return i->second;
18     else return Docencia();
19 }

21 Persona_Asignatura::Profesorado
22 Persona_Asignatura::asociados(Asignatura& a) const
23 {
24     std::map<Asignatura*, Profesorado>::
25     auto const_iterator i = inversa.find(&a);
26     if (i != inversa.end()) return i->second;
27     else return Profesorado();
28 }
```

Clase genérica de asociación con atributos de enlace (asoc_atr_enlace.h)

```
1  #include <map>
2  using std::map;
3  using std::make_pair;

5  template <typename X, typename Y, typename Z>
6  // X e Y: clases asociadas
7  // Z: clase de los atributos de enlace
8  class AsociacionBidireccional {
9  public:
10     void asocia(X& x, Y& y, Z& z);
11     void asocia(Y& y, X& x, Z& z);
12     map<Y*, Z*> asociados(X& x) const;
13     map<X*, Z*> asociados(Y& y) const;
14 private:
15     map<X*, map<Y*, Z*> > directa;
16     map<Y*, map<X*, Z*> > inversa;
17 };
```

Plantilla bidireccional varios-varios atribo enlace

Clase genérica de asociación con atributos de enlace (asoc_atr_enlace.h)

```
19 // Asocia bidireccionalmente dos objetos.
20 template <typename X, typename Y, typename Z>
21 void AsociacionBidireccional<X, Y, Z>::asocia(X& x, Y& y, Z& z)
22 {
23     directa[&x].insert(make_pair(&y, &z));
24     inversa[&y].insert(make_pair(&x, &z));
25 }

27 template <typename X, typename Y, typename Z> inline
28 void AsociacionBidireccional<X, Y, Z>::asocia(Y& y, X& x, Z& z)
29 { asocia(x, y, z); }
```


Clase genérica de asociación con atributos de enlace (asoc_atr_enlace.h)

```
31 // Devuelve el conjunto de enlaces asociados a un objeto.
32 template <typename X, typename Y, typename Z>
33 map<Y*,Z*> AsociacionBidireccional<X, Y, Z>::asociados(X& x) const
34 {
35     map<X*, map<Y*, Z*>>::const_iterator i = directa.find(&x);
36     if (i != directa.end()) return i->second;
37     else return map<Y*, Z*>();
38 }

40 template <typename X, typename Y, typename Z>
41 map<X*,Z*> AsociacionBidireccional<X, Y, Z>::asociados(Y& y) const
42 {
43     map<Y*, map<X*, Z*>>::const_iterator i = inversa.find(&y);
44     if (i != inversa.end()) return i->second;
45     else return map<X*, Z*>();
46 }
```