todoSobreLasAsociaciones.pdf

victxrms

Programación Orientada a Objetos

2º Grado en Ingeniería Informática

Escuela Superior de Ingeniería
Universidad de Cádiz

Todo sobre las asociaciones

(O al menos todo lo que yo se)

Programación Orientada a Objetos

TODO SOBRE LAS ASOCIACIONES

UNIDIRECCIONAL

SE GUARDA B EN A

RELACIÓN 1 A 1

SE GUARDA UN PUNTERO AL OBJETO DE LA OTRA CLASE

```
class A {          class B {
  B* b_              A* a_
}                  }
```

FUNCIONES REQUERIDAS

INSERTAR → `void A::insertar(const B& b) { b_ = &b; }`

OBSERVADORA → `B A::devolver() const { return *b_ }`

RELACIÓN 1 A N

EN EL LADO DE 1 SE GUARDA UN SET DE PUNTEROS A OBJETOS DE N Y EN ESTE GUARDAMOS UN PUNTERO AL OTRO

```
class A {              class B {
  set<B*> bs_            A* a_
}                      }
```

FUNCIONES REQUERIDAS

INSERTAR → `void A::insertar(const B& b) { bs_.insert(&b) }`

OBSERVADORA → `set<B*> A::devolver() const { return bs_ }`

PARA TI ESTE PORTÁTIL ES GENIAL

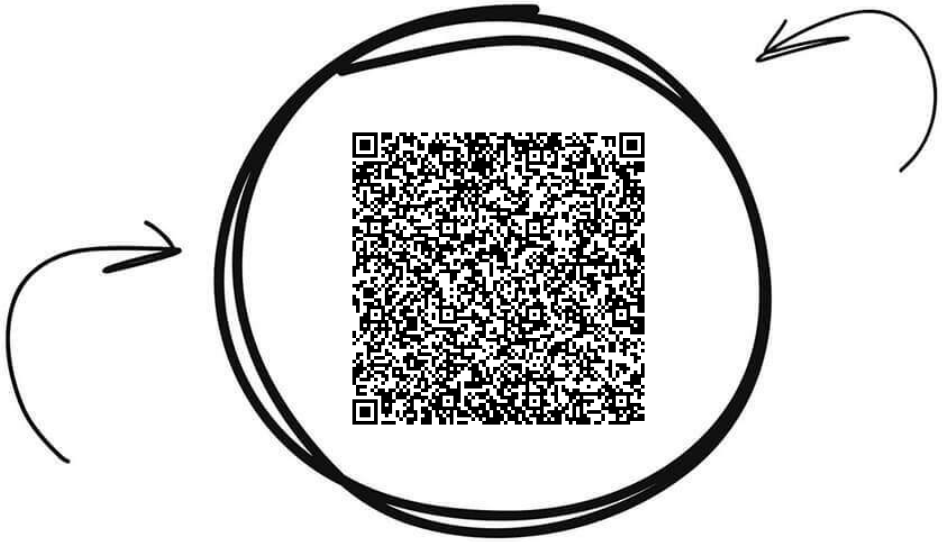
Para tu padre... está en oferta

Pásale este QR
a los Reyes magos...
o a tu padre

GAMING - MSI Stealth 16 Studio

Queridos Reyes Magos, este año he sido excepcionalmente bueno, y tengo en mente un regalo que podría hacer que mis tareas escolares se conviertan en épicas batallas de conocimiento. Me encantaría recibir un portátil ultraligero que, aunque suene a ciencia ficción, esté equipado con todo lo que un amante de los videojuegos como yo podría desear: pantalla UHD+, tarjeta gráfica hasta RTX4070, procesador i9, ¡y un peso menor a 2kg para llevarlo a donde sea! el color no me importa, podéis elegeri en blanco o azul marino.

Programación Orientada a Obj...



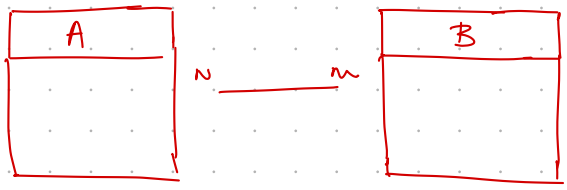
Banco de apuntes de la

WUOLAH

Comparte estos flyers en tu clase y consigue más dinero y recompensas

1. Imprime esta hoja
2. Recorta por la mitad
3. Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes
4. Llévate dinero por cada descarga de los documentos descargados a través de tu QR

RELACIÓN N A M



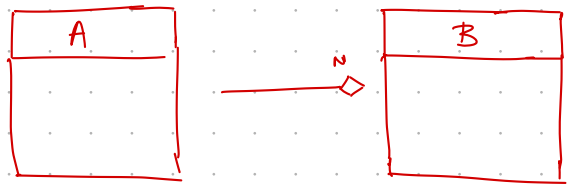
SE GUARDA UN SET DE PUNTEROS A OBJETOS DE LA OTRA CLASE

```
class A {                    class B {
  set<B*> bs_                  set<A*> as_
}                            }
```

FUNCIONES REQUERIDAS

IGUAL QUE PARA LA CLASE A DEL TIPO ANTERIOR

AGREGACIÓN



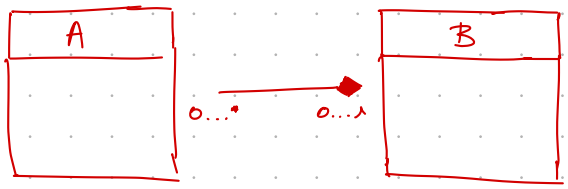
IGUAL QUE EL CASO ANTERIOR PERO PUEDE SER UNIDIRECCIONAL. HAY QUE TENER EN CUENTA LA MULTIPLICIDAD EN AMBOS LADOS

```
class A {                    class B {
  set<B*> bs_                }
}
```

FUNCIONES REQUERIDAS

IGUAL QUE PARA LA CLASE A DEL TIPO ANTERIOR

COMPOSICIÓN



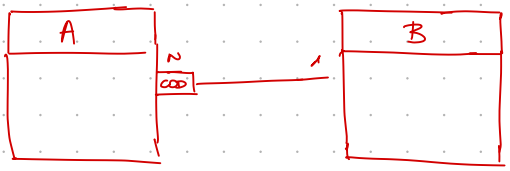
LA CLASE DEL OBJETO COMPUESTO TIENE UN OBJETO DE LA CLASE COMPONENTE

```
class A {                    class B {
}                              A a_
                             }
```

FUNCIONES REQUERIDAS

CONSTRUCTOR → `B ( A a) . a_(a) {}`

CALIFICADOR



LA CLASE CON EL CALIFICADOR TENDRÁ UN MAP CUYA CLAVE ES EL CALIFICADOR Y LA OTRA TIENE UN ATRIBUTO PARA ALMACENAR EL CALIFICADOR

```
class A {
  map<cod, B*> bs_
}
```

```
class B {
  cod c_
  set<A*> a
}
```

FUNCIONES REQUERIDAS

INSERTAR →
```
void A::insertar(const B& b) {
  bs_.insert(make_pair(b.cod(), &b))
}
```

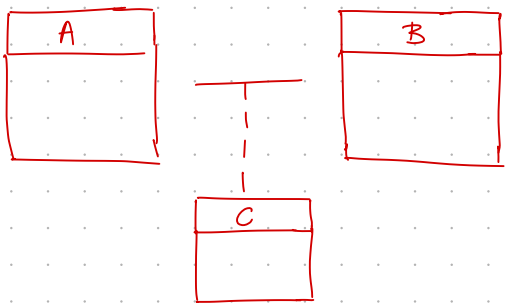
OBSERVADORA →
```
map<cod,B*> A::devolver() const {
  return bs_
}
```

ATRIBUTOS DE ENLACE

RELACIÓN 1 A 1 → LO PONEMOS EN LA PARTE PRIVADA DE UNA DE LAS DOS CLASES

RELACIÓN 1 A N → LO PONEMOS EN LA PARTE QUE TENGA N O CREAMOS UNA CLASE DE ASOCIACIÓN

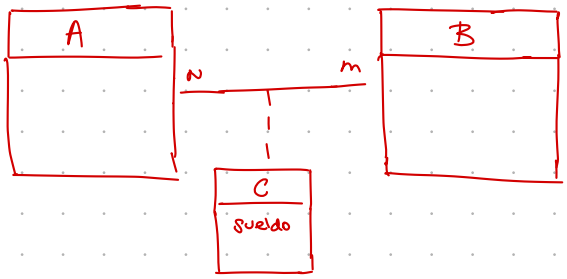
RELACIÓN N A M → HACEMOS UNA CLASE DE ASOCIACIÓN O AÑADIMOS UN MAP




```
class A {
  map<B*, C*> bc_
}
```

```
class B {
  map<A*, C*> ac_
}
```

CLASE DE ASOCIACIÓN



GUARDAMOS EN UNA CLASE AUX LOS ATRIBUTOS DE ENLACE Y EN LA CLASE DE ASOCIACIÓN GUARDAMOS UN MAP DE MAP DE DIRECTA E INVERSA Y SUS MÉTODOS PARA HACER LAS RELACIONES

```
class A {        class B {        class C {
   ...              ...              int sueldo
}                }                }

class asociacion {
    map<A*, map<B*, C*>> Directa
    map<B*, map<A*, C*>> Inversa
}
```

FUNCIONES REQUERIDAS

INSERTAR →

```
void A::insertar (const A& a, const B& b, const C& c)
    Directa[&a].insert(make_pair(&b, &c))
    Inversa[&b].insert(make_pair(&a, &c))
}
```

OBSERVADORA →

```
map<B*, C*> asociacion::devolver (const A& a) const {
    map<A*, map<B*, C*>>::const_iterator i = Directa.find(&a)
    if (i != Directa.end())
        return i->second;
    else
        return map<B*, C*>();
}

map<A*, C*> asociacion::devolver (const B& b) const {
    map<B*, map<A*, C*>>::const_iterator i = Inversa.find(&b)
    if (i != Inversa.end())
        return i->second;
    else
        return map<A*, C*>();
}
```

- Si A tuviera multiplicidad 1

```
class Asociacion {
    map<A*, map<B*, C*>>  Directa
    map<B*, pair<A*, C*>>  Inversa
};
```

Funciones requeridas

Insertar →
```
void A::insertar(const A& a, const B& b, const C& c)
    Directa[&a].insert(make_pair(&b, &c));
    Inversa[&b].first = &a;  Inversa[&b].second = &c;
}
```

Observadora →
```
pair<A*, C*> asociacion::devolver(const B& b) const {
    map<B*, pair<A*, C*>>::const_iterator i = Inversa.find(&b)
    if (i != Inversa.end())
        return i->second;
    else
        return pair<A*, C*>();
}
```

- Si NO hubiera atributos de enlace y multiplicidad N a M

```
class Asociacion {
    map<A*, set<B*>>   Directa
    map<B*, set<A*>>   Inversa
};
```

Funciones requeridas

Insertar →
```
void A::insertar(const A& a, const B& b)
    Directa[&a].insert(&b)
    Inversa[&b].insert(&a)
}
```

Observadora →
```
set<B*> asociacion::devolver(const A& a) const {
    map<A*, set<B*>>::const_iterator i = Directa.find(&a)
    if (i != Directa.end())
        return i->second;
    else
        return set<B*>();
}
```

Observadora →
```
set<A*> asociacion::devolver(const B& b) const {
    map<B*, set<A*>>::const_iterator i = Inversa.find(&b)
    if (i != Inversa.end())
        return i->second;
    else
        return set<A*>();
}
```

- Si NO hubiera atributos de enlace y multiplicidad 1 a M

```
class Asociacion {
    map<A*, set<B*>>   Directa
    map<B*, A*>        Inversa
};
```