Herencia Múltiple

Prof. Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Programación y Diseño Orientado a Objetos

Doble Grado en Ingeniería Informática
y Administración y Dirección de Empresas

(Curso 2023-2024)

Créditos

- Las siguientes imágenes e ilustraciones son libres y se han obtenido de:
  - Emojis, `https://pixabay.com/images/id-2074153/`
  -  `https://medias.maisonsdumonde.com/image/upload/q_auto,f_auto/w_2000/img/estanteria-de-metal-negro-y-abeto-con-reloj-1000-0-31-188836_1.jpg`
- El resto de imágenes e ilustraciones son de creación propia, al igual que los ejemplos de código

Objetivos

- Comprender en qué consiste la herencia múltiple
- Entender los problemas que puede ocasionar
- Conocer alternativas

Contenidos

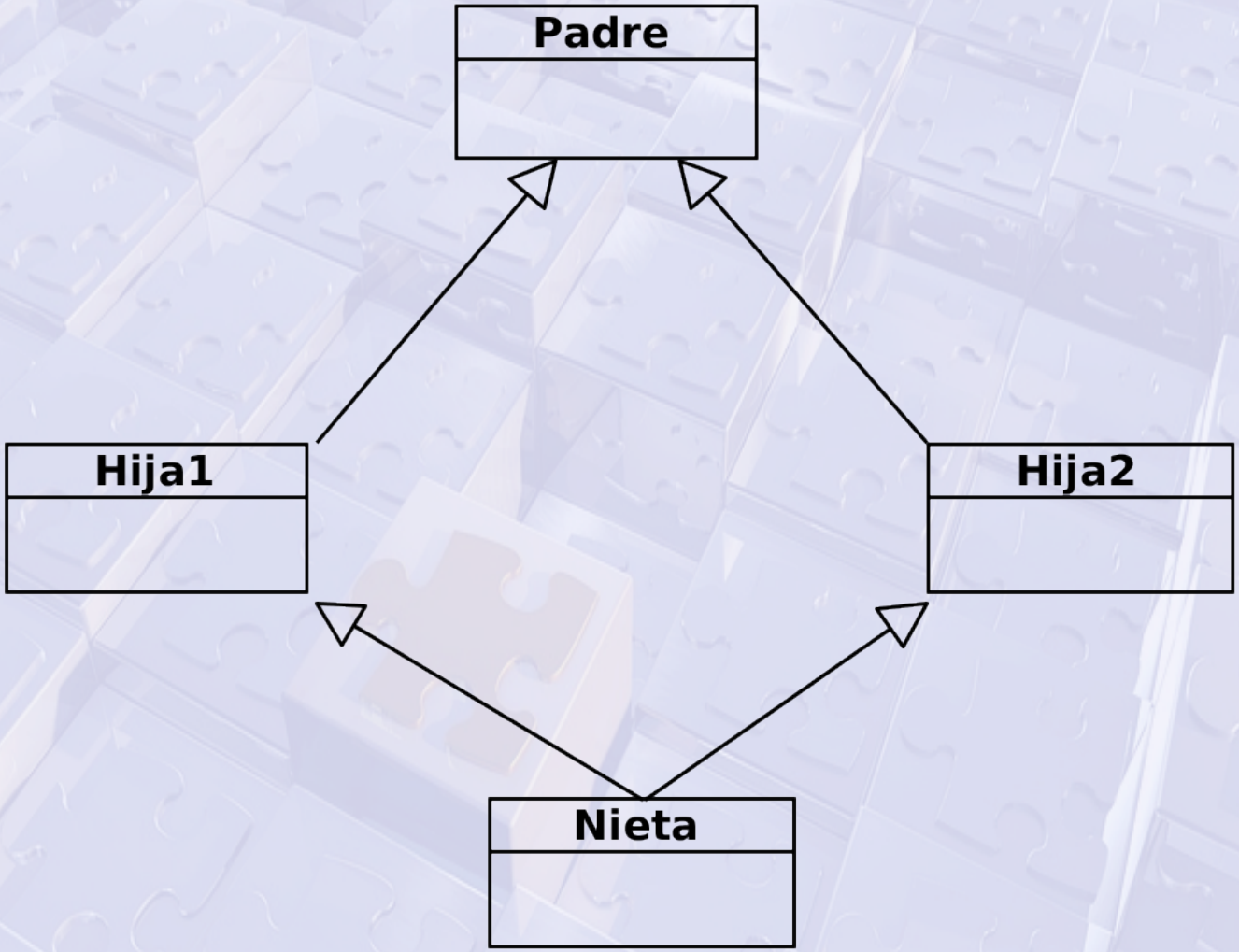
Herencia múltiple

- Se produce cuando una clase es descendiente de más de una superclase
- Permite representar problemas donde un objeto tiene propiedades según criterios distintos (clasificado según criterios distintos)
- Presenta problemas de implementación y pocos lenguajes la soportan (Ej. C++ y Python)
- Java y Ruby no tienen herencia múltiple

Problemas comunes

- Colisión de nombres de métodos y/o atributos
- Problema del diamante:
  - Provoca duplicidad en los elementos heredados
- No hay que olvidar que para que tenga sentido debe haber una relación *es-un* de la clase descendiente con todos sus ascendientes
  - La reutilización de código existente en varias clases no es por si solo un criterio para establecer relaciones de herencia múltiple

Problema del diamante



Ejemplo del problema del diamante

C++: Herencia múltiple. Problema del diamante

```
1  class Persona
2  {
3    private:
4      string nombre;
5    public:
6      Persona()  {cout<<"Creada Persona SIN INICIALIZAR" <<endl;}
7      Persona(string n) {cout<<"Creada Persona e inicializada" <<endl; nombre = n;}
8      string getNombre() {return nombre;}
9      void setNombre(string n) {nombre = n;}
10 };
11
12 class Docente: public Persona
13 {
14   public:
15     Docente(string n): Persona(n) {}
16     void presentaDocente() {cout<<"Soy el docente "<<getNombre()<<endl;}
17 };
18
19 class Investigador: public Persona
20 {
21   public:
22     Investigador(string n): Persona(n) {}
23     void presentaInvestigador() {cout<<"Soy el investigador "<<getNombre()<<endl;}
24 };
```

Ejemplo del problema del diamante

C++: Herencia múltiple. Problema del diamante

```
1 class Profesor: public Docente, public Investigador
2 {
3   public:
4     Profesor(string n): Docente(n), Investigador(n){}
5     void presentaProfesor(){presentaDocente(); presentaInvestigador();}
6     void modificaNombre(string n) {cout<<"Me modifico el nombre"<<endl; setNombre(n);}
7  // Error: hay ambigüedad
8     void modificaNombre(string n) {cout<<"Me cambio de nombre"<<endl; Docente::setNombre(n);}
9 };
10
11 int main(int argc,char **argv) {
12   Profesor *p = new Profesor("Ana");
13   // Creada Persona e inicializada
14   // Creada Persona e inicializada
15
16   p->presentaProfesor();
17   // Soy el docente Ana
18   // Soy el investigador Ana
19
20   p->modificaNombre("Juan");
21   p->presentaProfesor();
22   // Soy el docente Juan
23   // Soy el investigador Ana
24 }
```

Ejemplo del problema del diamante

C++: Herencia múltiple. Solución C++ a la duplicidad de atributos

```
1  class Persona
2  {
3    private:
4      string nombre;
5    public:
6      Persona()  {cout<<"Creado A SIN INICIALIZAR" <<endl;}
7      Persona(string n) {cout<<"Creada Persona e inicializada" <<endl; nombre = n;}
8      string getNombre() {return nombre;}
9      void setNombre(string n) {nombre = n;}
10 };
11
12 class Docente: virtual public Persona
13 {
14   public:
15     Docente(string n): Persona(n) {}
16     void presentaDocente() {cout<<"Soy el docente "<<getNombre()<<endl;}
17 };
18
19 class Investigador: virtual public Persona
20 {
21   public:
22     Investigador(string n): Persona(n) {}
23     void presentaInvestigador() {cout<<"Soy el investigador "<<getNombre()<<endl;}
24 };
```

Ejemplo del problema del diamante

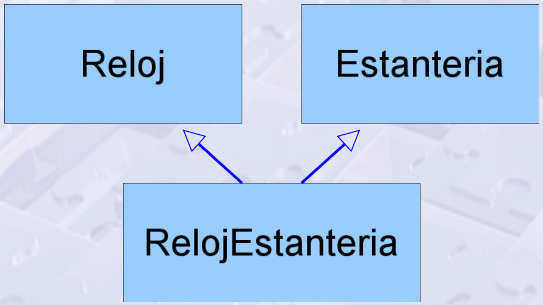
C++: Herencia múltiple. Solución C++ a la duplicidad de atributos

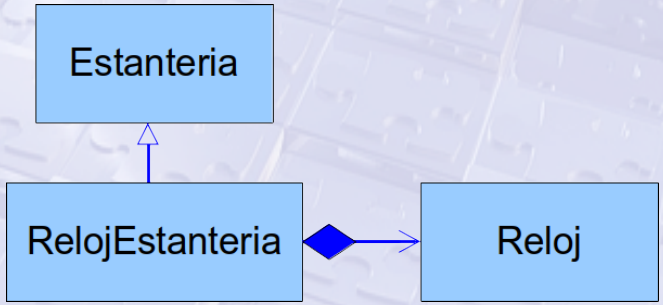
```
1 class Profesor: public Docente, public Investigador
2 {
3   public:
4     Profesor(string n): Persona(n), Docente(n), Investigador(n){}
5     void presentaProfesor(){presentaDocente(); presentaInvestigador();}
6     void modificaNombre(string n) {cout<<"Me modifico el nombre"<<endl; setNombre(n);}
7     // Ya no hay ambigüedad
8 };
9
10 int main(int argc,char **argv) {
11   Profesor *p = new Profesor("Ana");
12   // Creada Persona e inicializada
13   // Creada Persona e inicializada
14
15   p->presentaProfesor();
16   // Soy el docente Ana
17   // Soy el investigador Ana
18
19   p->modificaNombre("Juan");
20   p->presentaProfesor();
21   // Soy el docente Juan
22   // Soy el investigador Juan
23
24   Docente *d = new Docente("Pepe");
25   d->presentaDocente();
26   // Las instancias de Docente tambien tienen el atributo Persona::nombre
27 }
```

Alternativas

- Composición
  - Sustituir una o varias relaciones de herencia por composición
- Interfaces Java
  - Se pueden realizar varias interfaces Java y heredar de una superclase
- Mixins de Ruby
  - Permiten incluir código proveniente de varios módulos como parte de una clase

Ejemplo de composición

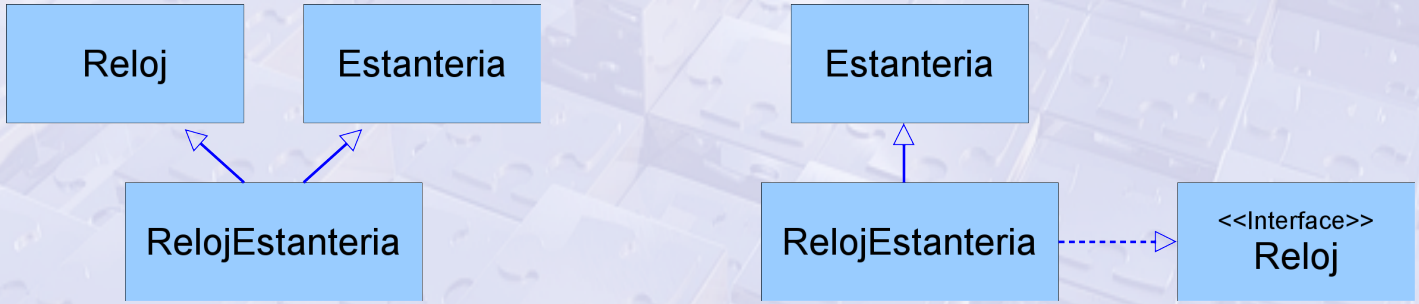




Java: Ejemplo de composicion

```
1  class Estanteria { . . . }
2
3  class Reloj { . . . }
4
5  class RelojEstanteria extends Estanteria {
6    private Reloj reloj;
7
8    RelojEstanteria () {
9      super();
10     reloj = new Reloj();
11   }
12
13   // Los métodos de Estanteria se heredan
14
15   void setHora (Hora h) {      // Los métodos de Reloj se definen
16     reloj.setHora (h);         // se implementan reenviando el mensaje al atributo
17   }
18 }
```

Ejemplo con interfaces Java



Java: Ejemplo con interfaces Java

```
1 class Estanteria { . . . }
2
3 interface Reloj { public void setHora (Hora h); public Hora getHora (); }
4
5 class RelojEstanteria extends Estanteria implements Reloj {
6
7   RelojEstanteria () {
8     super();
9   }
10
11  // Los métodos de Estanteria se heredan
12
13  // Se implementan los métodos de la interfaz
14
15  public void setHora (Hora h) { . . . }
16  public Hora getHora () { . . . }
17 }
```

Ejemplo de mixin de Ruby

Ruby: Ejemplo de mixin de Ruby

```ruby
module Volador
  def volar
    puts "Volando"
  end
end

module Nadador
  def nadar
    puts "Nadando"
  end
end

class Ejemplo
  def metodo
    puts "Método propio"
  end

  include Volador # Añadimos todo el módulo
  include Nadador # Añadimos todo el módulo
end

e=Ejemplo.new
e.metodo
e.volar
e.nadar
```

Herencia Múltiple

Prof. Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Programación y Diseño Orientado a Objetos

Doble Grado en Ingeniería Informática
y Administración y Dirección de Empresas

(Curso 2023-2024)