Herencia y Otros en C++

Dr. Juan Bekios Calfa

http://jbekios.ucn.cl

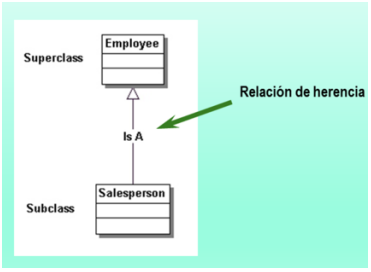
juan.bekios@ucn.cl

Herencia en C++

- La herencia es una especialización entre clases. Una clase especializa la estructura o comportamiento definido en una (herencia simple) o más (herencia múltiple) clases.
- La nueva clase es una subclase (***derivada***) de una super clase existente (***base***).
- Una subclase típicamente aumenta o redefine la estructura existente y el comportamiento de su superclase.

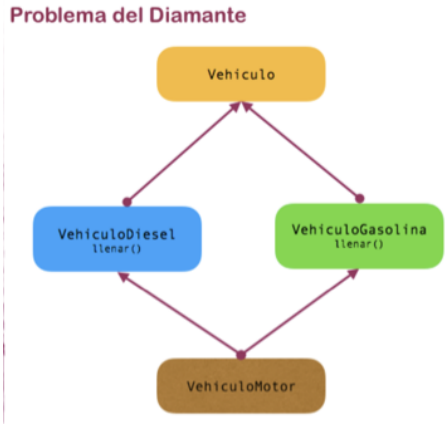
Base

Derivada



Problemas de Herencia Múltiple

Herencia múltiple permite unir comportamiento de varias clases, lo cual se puede ver genial … pero hay problemas



¿Cuál método "llenar" se usará? Este problema se puede heredar hacia todas las subclases …

¿Quieres saber más?

Nathanael Schärli, Stéphane Ducasse, Oscar Nierstrasz and Andrew Black, **"Traits: Composable Units of Behavior,"** *Proceedings ECOOP 2003 (European Conference on Object-Oriented Programming)*, LNCS, vol. 2743, Springer Verlag, July 2003, pp. 248-274.

¡NO a la "Herencia Implementacional"!

Una subclase no debería violar el sentido de la súper clase, pues un objeto de una sub-clase sigue siendo un objeto de la súper clase. Por ejemplo:

- Base (Súper-clase) implementa una pila a través de una LinkedList
- Derived (Sub-clase) hereda de una pila y modifica el método "add" para agregar al inicio de la lista (= cola)

¿Cuál es el problema acá?

Ejemplo de Herencia (1)

```
#include<iostream>

using namespace std;

class Animal {
    public:
        Animal() {cout <<"Animal"<<endl;}
        void eat() { std::cout << "I'm eating generic food."<<endl; }
};

class Cat : public Animal {
    public:
        Cat() {cout <<"Cat"<<endl;}
        void eat() { std::cout << "I'm eating a rat."<<endl; }
};
```

```
int main() {
  Animal* animal = new Animal();
  Cat* cat = new Cat();

  animal->eat();
  cat->eat();

  return 0;
}
```

¿Qué muestra por pantalla?
¿Por qué?

https://repl.it/@PaulLeger/clase41

Ejemplo de Herencia (2)

```
#include<iostream>

using namespace std;

class Animal {
    public:
        Animal() {cout <<"Animal"<<endl;}
        void eat() { std::cout << "I'm eating generic food."<<endl; }
};

class Cat : public Animal {
    public:
        Cat() {cout <<"Cat"<<endl;}
        void eat() { std::cout << "I'm eating a rat."<<endl; }
};

void eats(Animal* a) {
  a->eat();
}
```

```
int main() {
  Animal* animal = new Animal();
  Cat* cat = new Cat();

  eats(animal);
  eats(cat);

  return 0;
}
```

¿Qué muestra por pantalla?
¿Por qué?

https://repl.it/@PaulLeger/clase42

C++ no es igual a Java

Ejemplo de Herencia (3)

```cpp
#include<iostream>

using namespace std;

class Animal {
    public:
        Animal() {cout <<"Animal"<<endl;}
        virtual void eat() { std::cout << "I'm eating generic food."<<endl; }
};

class Cat : public Animal {
    public:
        Cat() {cout <<"Cat"<<endl;}
        void eat() { std::cout << "I'm eating a rat."<<endl; }
};

void eats(Animal* a) {
  a->eat();
}
```

```cpp
int main() {
  Animal* animal = new Animal();
  Cat* cat = new Cat();

  eats(animal);
  eats(cat);

  return 0;
}
```

https://repl.it/@PaulLeger/clase43

¿Qué muestra por pantalla?
¿Por qué?

Para que funcione C++ como Java, debe usar virtual

Ejemplo de Herencias (4): Figuras

Realice un diagrama de clases del siguiente código.

```
class Figure{
…
}

class Triangle: public Figure {
…};

class Rectangle: public Figure {
…};

class Group: public Figure {
 private:
 Figure *figures[MAX];
…};
```

Clases Abstractas

- Las clases que **no pueden tener** instancias se denominan clases abstractas.
- Una **clase abstracta** es escrita con el propósito de que sus subclases de compartir comportamientos, estados y tipos, a través de completar la implementación de sus (usualmente) métodos incompletos.
- C++ permite al desarrollador impedir que un método asociado a una clase abstracta sea invocado directamente, mediante la inicialización de su declaración en cero. Tal método se denomina una función virtual pura; el lenguaje prohíbe la creación de instancias cuya clase exporta tales funciones.

Clases Abstractas

- Por ejemplo:

  virtual void Draw()**=0;**

- La clase más generalizada en una estructura de clases se llama ***la Clase Base***. Algunos lenguajes incluyen una clase base, que sirve como la superclase fundamental de todas las clases.
- Por ejemplo:

  Java: Object

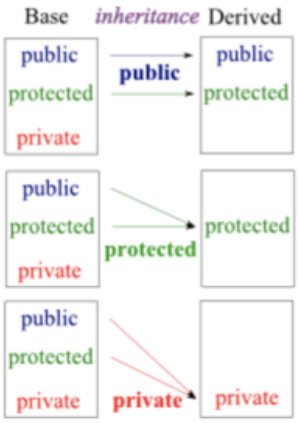
  Smalltalk: Object

```
class Figure {

protected:
  Point** vertices;
  int size;

public:

  Figure() {…
  }

  Figure(int nargs,...) {…
  };

  int getSize() {…
  }

  double getPerimeter() {…
  }

  virtual void draw() = 0;

  ~Figure() {…
  }
};
```

Clases Abstractas

Una clase puede ser derivada a partir de una clase existente usando:

```
class nombre-clase: [public | protected |private] nombre-clase-base{

  //declaraciones

};
```

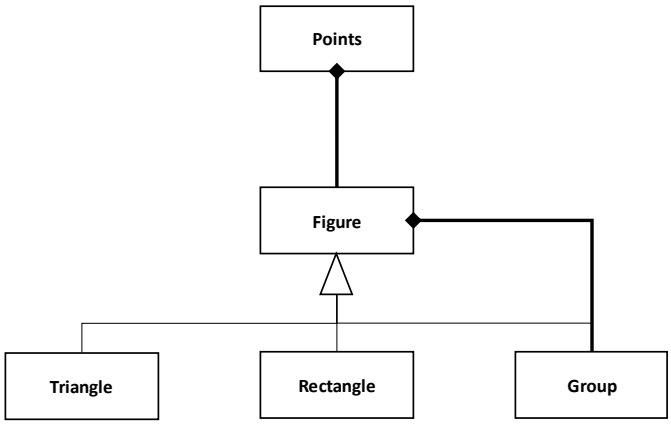


Si quieres seguir un estilo Java, use “public”

Herencia privada y protegida no es comúnmente implementada en los lenguajes

Implementado Ejemplo de Herencia (4): Figuras

- Toda figura está compuesta de un conjunto de vertices, y cada figura puede calcular su perimetro y ser dibujada
- El perimetro es simplemente la suma de las distancia de sus lados, pero el dibujado es reponsabilidad de cada clase
- Además tenemos una figura Group, que contiene varias figuras y puede agregar más figuras y grupos



Clases Punto y Figura

```
class Point {
  private:
  int x,y;

  public:

  Point() {}

  Point(int x, int y) {
    this->x = x;
    this->y = y;
  }

  int getX() {
    return x;
  }

  int getY() {
    return y;
  }

  double getDistance(Point *p) {
    return sqrt(pow(x-p->x,2) + pow(y-p->y,2));
  }
};
```

```
class Figure {

protected:
  Point** vertices;
  int size;

public:

  Figure() {
    size = 0;
  }

  int getSize() {
    return size;
  }

  double getPerimeter() {
    double s = 0;
    for (int i = 0; i < size - 1; ++i) {
      s += vertices[i]->getDistance(vertices[i+1]);
    }
    return s + vertices[size-1]->getDistance(vertices[0]);
  }

  virtual void draw() = 0;

  ~Figure() {
    cout << "deleting Figure
    delete[] vertices;
  }
};
```

Pointers of pointers????

https://repl.it/@PaulLeger/clase44

```
Figure(int nargs,...) {
  size = nargs;
  vertices = new Point*[size];

  va_list args;
  va_start(args, nargs);

  for (int i = 0; i < size; ++i) {
    Point *p = va_arg(args, Point*);
    vertices[i] = p;
  }
  va_end(args);
};
```

Metodo abstracto

Figure con variables números de argumentos

Clases Triangulo y Rectángulo

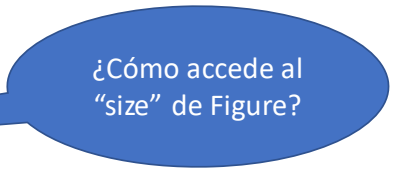
```
class Triangle: public Figure {

public:
  Triangle(Point *p1, Point *p2, Point *p3): Figure(3,p1,p2,p3){}

  void draw() {
    cout << "Triangle" << endl;
  }
};

class Rectangule: public Figure {
public:
  Rectangule(Point *p1, Point *p2, Point *p3, Point *p4): Figure(4,p1,p2,p3,p4) {}

  void draw() {
    cout << "Rectangule" << endl;
  }
};
```

Triangulo y Rectángulo llaman al constructor de Figura
Con un número fijo de parámetro

```
class Group: public Figure {
private:
  int size;
  Figure *figures[MAX];

public:
  Group(int nargs,...) {
    size = nargs;

    va_list args;
    va_start(args, nargs);

    for (int i = 0; i < size; ++i) {
      Figure *f = va_arg(args, Figure*);
      figures[i] = f;
    }
    va_end(args);
  }

  void add(Figure* f) {
    figures[size++] = f;
  }

  void draw() {
    for (int i = 0; i < size; ++i) {
      figures[i]->draw();
    }
  }

};
```

Group no llama explícitamente al constructor de Figura, pero implícitamente llama al constructor vacío de figura

Redefine "size" y permite dibujar cada figura.



Ejercicio

- Al código de Figura, agregar cuadrado:
  - Este hereda de Rectangulo
  - Redefinir el método draw
  - [Propuesto]: Implementar el método "void showArea", el cual muestra por pantalla el calculo del área del cuadrado

Polimorfismo:
Volviendo a ejemplo de los animales

- En Smalltalk y Java, cualquier método de la superclase puede ser redefinido en una subclase.

- C++: funciones miembros que son declaradas como **virtual** permiten el polimorfismo. *¡C++ es especial!*