ESTRUCTURAS DE DATOS

NOTAS SOBRE C++

Herencia y polimorfismo

Manuel Montenegro Montes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

Herencia

Heredar de una clase

```
class Rectangulo {
public:
  Rectangulo(double ancho, double alto): ancho(ancho), alto(alto) { }

  double area() { return ancho * alto; }
  double perimetro() { return 2 * ancho + 2 * alto; }

protected:
  double ancho, alto;
};

class Cuadrado: public Rectangulo {
public:
  Cuadrado(double lado): Rectangulo(lado, lado) { }
};
```

Ejemplo

```
Rectangulo *r;
double ancho, alto;
cin >> ancho >> alto;

if (ancho == alto) {
  r = new Cuadrado(ancho);
} else {
  r = new Rectangulo(ancho, alto);
}

double area = r->area();
double perimetro = r->perimetro();
cout << "Area: " << area << endl;
cout << "Perímetro: " << perimetro << endl;

delete r;
```

Polimorfismo y métodos virtuales

Nuevo método: dibujar()

```
class Rectangulo {
public:
   ...
  void dibujar() {
    std::cout << "Rectángulo de ancho " << ancho << " y alto " << alto << std::endl;
  }
protected:
  double ancho, alto;
};

class Cuadrado: public Rectangulo {
public:
  Cuadrado(double lado): Rectangulo(lado, lado) { }

  void dibujar() {
    std::cout << "Cuadrado de lado " << ancho << std::endl;
  }
};
```

Ejemplo

```
Rectangulo *r;
double ancho, alto;
cin >> ancho >> alto;

if (ancho == alto) {
  r = new Cuadrado(ancho);
} else {
  r = new Rectangulo(ancho, alto);
}

r→dibujar();

delete r;
```

```
1.2 4.5
Rectángulo de ancho 1.2 y alto 4.5
```

```
1.2 1.2
Rectángulo de ancho 1.2 y alto 1.2
```

Vinculación estática vs dinámica

- C++ determina a qué método llamar en base al tipo del objeto sobre el que se realiza la llamada.

```
Rectangulo *r;
double ancho, alto;
cin >> ancho >> alto;

if (ancho == alto) {
  r = new Cuadrado(ancho);
} else {
  r = new Rectangulo(ancho, alto);
}

r→dibujar();

delete r;
```

`r` es de tipo puntero a `Rectangulo`
Por tanto el compilador determina que `r→dibujar()` llama al método `dibujar` de `Rectangulo`.

Vinculación estática vs dinámica

- Si se realiza **vinculación dinámica**, decimos al compilador que se compruebe, en tiempo de ejecución, la clase a la que pertenece el objeto, y se llame al método correspondiente a esa clase, independientemente del tipo.
- Por defecto, en C++ se utiliza vinculación estática.
- Por defecto, en Java se utiliza vinculación dinámica.

Habilitar la vinculación dinámica

```
class Rectangulo {
public:
   ...
  virtual void dibujar() {
    std::cout << "Rectángulo de ancho " << ancho << " y alto " << alto << std::endl;
  }
protected:
  double ancho, alto;
};

class Cuadrado: public Rectangulo {
public:
  Cuadrado(double lado): Rectangulo(lado, lado) { }

  virtual void dibujar() {
    std::cout << "Cuadrado de lado " << ancho << std::endl;
  }
};
```

Ejemplo

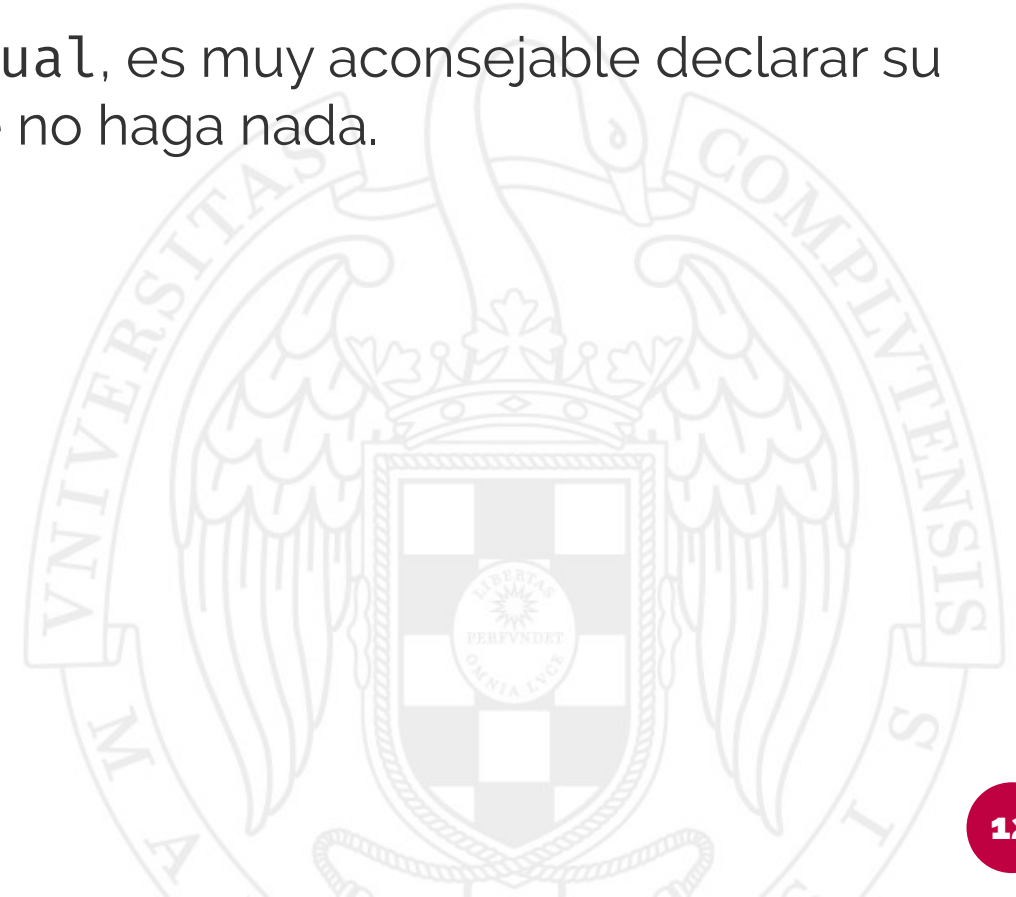
```
Rectangulo *r;
double ancho, alto;
cin >> ancho >> alto;

if (ancho == alto) {
  r = new Cuadrado(ancho);
} else {
  r = new Rectangulo(ancho, alto);
}

r→dibujar();

delete r;
```

```
1.2 4.5
Rectángulo de ancho 1.2 y alto 4.5
```

```
1.2 1.2
Cuadrado de lado 1.2
```

Reglas generales

- Cualquier método que sea susceptible de ser reescrito debe declararse como `virtual`.
- Si una clase tiene un método `virtual`, es muy aconsejable declarar su destructor como `virtual`, aunque no haga nada.



Métodos abstractos

```
class Figura {
public:
  virtual double area() = 0;
  virtual double perimetro() = 0;
  virtual void dibujar() = 0;

  virtual ~Figura() { }

};


class Rectangulo: public Figura {  ...  }
```

- Los métodos abstractos han de ser virtuales.
- Si una clase tiene un método abstracto, la clase es abstracta.
  - No pueden crearse instancias de `Figura`.

Otras diferencias con Java

- En C++ no existe la noción de interfaz (`interface`).
- Se permite herencia múltiple.