Programación II

Tema 7

jtorresmat@nebrija.es

Índice

Introducción

- Polimorfismo:
  - Del griego “polys”+”morfo” -> Cualidad de tener muchas formas
  - De todo lo que hemos visto hasta ahora sobre POO... ¿a qué conceptos vistos en PII os suena esta definición?
    - Sobrecarga
    - Plantillas

Introducción

- Principales características de C++:
  - Abstracción
  - Encapsulamiento
  - Herencia
  - Polimorfismo

Introducción

- Principales características de C++:
  - *Abstracción*:
    - Mecanismo de diseño en POO
    - Aplicarla nos permite extraer un conjunto de entidades, datos y comportamientos comunes para almacenarlos en clases

Introducción

- Principales características de C++:
  - *Encapsulamiento*: cada clase tendrá una interfaz con la que se comunicará con el exterior (parte pública). Y también tendrá una serie funciones miembro privadas/protegidas con las que realizará operaciones sobre las variables miembro de la clase. (Es un mecanismo de protección de los datos)

Introducción

- Principales características de C++:
  - *Herencia*: mecanismo que nos permite crear clases derivadas (concretas) a partir de clases base (generales). Añadiendo o extendiendo funcionalidades a medida que avanzamos hacia la concretización.

Introducción

- Principales características de C++:
  - *Polimorfismo*: nos permite tener distintas implementaciones de un mismo método dependiendo de la clase en la que se realice. Es decir, podemos acceder a una variedad de métodos distintos con el mismo nombre mediante el mismo mecanismo de acceso.

Introducción

- Principales características de C++:
  - *Polimorfismo*:
    - En el ej. de los vehículos podremos decir al vehículo que se desplace sin importarnos como lo hace.
      - Coche: llave, giro, embragar, meter marcha, etc…
      - Avión: encendido turbinas, bajada de flags, etc...

Introducción

- Principales características de C++:
  - *Polimorfismo*:
    - En C++ lo conseguimos mediante:
      - Definición clases derivadas (herencia)
      - Funciones virtuales
      - Punteros a objetos

Índice

Polimorfismo estático

- Polimorfismo: pretende dotar de una misma interfaz a entidades de distintos tipos
  - Estático: en tiempo de compilación
    - Sobrecarga de funciones
    - Plantillas
  - Dinámico: en tiempo de ejecución
    - Funciones virtuales

Polimorfismo estático

- Polimorfismo:
  - Sobrecarga de funciones:

```
#include <iostream>
void funcion(int entero){std::cout<<"Funcion con parametro entero"<<std::endl;}
void funcion(bool digital){std::cout<<"Funcion con parametro booleano"<<std::endl;}

int main()
{
   funcion(-8); //Muestra: Funcion con parametro entero
   funcion(true);//Muestra: Funcion con parametro booleano

   return 0;
}
```

  - ***funcion*** es polimórfica(tiene 2 formas)

Polimorfismo estático

- Polimorfismo:
  - Sobrecarga de funciones miembro:

```
#include <iostream>
struct cosa{
   void funcion(int p1){std::cout<<"int"<<std::endl;}
   void funcion(bool p1){std::cout<<"bool"<<std::endl;}};
int main()
{
   cosa MiCosa;
   MiCosa.funcion(3);//Muestra: int
   MiCosa.funcion(false);//Muestra: bool
   return 0;
}
```

  - ***funcion*** es polimórfica(tiene 2 formas)

Polimorfismo estático

- Polimorfismo:
  - Plantillas: más de lo mismo…

```
#include <iostream>
template <typename T>
void funcion(T) {std::cout<<"Lo que quieras..." <<std::endl; }
int main()
{
   int **punt=nullptr;

   funcion("Hola");//Muestra: Lo que quieras...
   funcion(8);//Muestra: Lo que quieras...
   funcion(punt);//Muestra: Lo que quieras...

   return 0;
}
```

Polimorfismo estático

Ejemplo:

Implementar función permutar templatizada. La función debe intercambiar el valor de dos variables sin usar std::swap

Código

Índice

Funciones virtuales

- Polimorfismo:
  - Clases derivadas:

    ejemplo con overriding u superposición

    ejemplo clase derivada con sobrecarga

Hasta ahora desde la clase base sólo podemos acceder a datos y funciones de la clase base, los datos y funciones propias de los objetos de clases derivadas serán inaccesibles. Esto es debido a que el compilador decide en tiempo de compilación que métodos y atributos están disponibles para cada clase.

Funciones virtuales

- Polimorfismo:
  - Clases derivadas:

El polimorfismo se basa en poder acceder a miembros de una clase derivada desde un puntero a su clase base

Ejemplo

Funciones virtuales

- Polimorfismo: pretende dotar de una misma interfaz a entidades de distintos tipos
  - Estático: en tiempo de compilación
    - Sobrecarga de funciones
    - Plantillas
  - Dinámico: en tiempo de ejecución
    - Funciones virtuales

Funciones virtuales

- Polimorfismo:
  - Funciones virtuales: Una función virtual es una función miembro de una clase base, que será posteriormente definida en la clase hija.

Funciones virtuales

- Polimorfismo:
  - Funciones virtuales:(ejemplo)

```
#include <iostream>
class Figura{
public:
        Figura(){};
        virtual ~Figura(){}
        virtual float getArea(){std::cout << "No implementado\n";return 0;}};

class Cuadrado:Figura{
public:
        float lado;
        Cuadrado(float a):lado{a} {}
        float getArea(){return lado*lado;}};
```

Funciones virtuales

- Polimorfismo:
  - Funciones virtuales: (ejemplo, ¿qué está pasando?)
    - Figura tiene una función virtual getArea()(palabra reservada **virtual**)
    - Figura por tener función virtual necesita destructor virtual (**virtual** ~***Figura***(){})
    - Cuadrado implementa una función *getArea*(). Y la implementa de forma "normal" ya que no usa la palabra reservada **virtual**.
    - Para este código ¿Que muestra por pantalla?

Funciones virtuales

- Polimorfismo:
  - Funciones virtuales: para el ejemplo anterior no parece que aporte mucho la aplicación de la función virtual y el polimorfismo… :S Pero poco a poco... ¡¡¡Vamos a intentar dárselo!!!

Índice

Polimorfismo dinámico

- Polimorfismo: hace lo siguiente (importante que se entienda...):
  - Si tengo un puntero a una clase padre, e inicializo el puntero como tipo de la clase hija -> puedo llamar a alguna de las funciones virtuales del padre y se ejecutará la implementación de la hija.
  - Ejemplo

Polimorfismo dinámico

- Polimorfismo:
  - ¿Y qué pasa si no hay redefinición de la metodo virtual?
  - Ejemplo

Polimorfismo dinámico

- Ejemplo aplicación polimorfismo:
  - Para las clases figura, cuadrado y triángulo de los ejemplos anteriores hacer un vector de punteros a figuras y “cargarlo” con un puntero a triángulo y otro a cuadrado
  - De tal manera que al recorrerlo y llamar al método getArea() de el resultado correcto para cada caso
  - Código

Polimorfismo dinámico

- Polimorfismo - Clases abstractas:
  - Clase abstracta: es una clase que tiene al menos una función virtual pura.
  - Función virtual pura: es una función virtual que no se implementa en la clase base/padre (sólo se declara) y que por lo tanto debe ser implementada en todas las clases hijas.

Polimorfismo dinámico

- Polimorfismo - Clases abstractas:
  - Función virtual pura: ejemplo

```
class Figura{
public:
    Figura(){};
    virtual float getArea() const  = 0;
};
```

  - Función virtual pura: se declara haciéndola = 0
  - Se utilizan cuando no tiene sentido implementarla en la clase base, y queremos "forzar" a que se implemente en todas las derivadas

Polimorfismo dinámico

- Polimorfismo - Clases abstractas:
  - Clase abstracta: es una clase que tiene al menos una función virtual pura.
    - No se puede instanciar un objeto de una clase abstracta: ejemplo
  - Hay que definir la función virtual de la clase base en todas las clases derivada que hereden de ella (si no las clases derivadas serán abstractas también): ejemplo

Polimorfismo dinámico

- Polimorfismo - Clases abstractas:
  - ¿Para qué vale?
  - De entrada parece que crear una clase de la que no podemos instanciar objetos puede servir de poco… ¿verdad? pero...
  - Son imprescindibles para un buen diseño POO
  - Habitualmente las clases superiores de la jerarquía de clases de un diseño de POO son abstractas, siendo las derivadas las que van concretando la funcionalidad

Polimorfismo dinámico

- Polimorfismo - Downcasting:
  - Las funciones derivadas pueden ampliar la funcionalidad de la clase base (añadir miembros que no están en la base)
  - Pero… ¿Cómo podemos acceder a esas funciones desde la base? Intento explicar con un ejemplo
  - Mediante downcasting podremos acceder
    - Se trata de convertir un tipo de la clase padre en un tipo de la clase derivada para poder acceder así a sus miembros

Polimorfismo dinámico

- Polimorfismo - Downcasting:
  - Podemos hacer downcasting siempre que:
    - La clase derivada herede de la base
    - La clase derivada a la que queremos acceder a través de la clase base sea del tipo que decimos
  - Sintaxis para nuestro ejemplo:

```
Triangulo* t = dynamic_cast<Triangulo*>(elem.get());
```

puntero clásico de tipo clase derivada

puntero inteligente de tipo clase base

- Ejemplo

Polimorfismo dinámico

- Polimorfismo - Downcasting:
  - ¿Qué pasa si intentamos convertir cosas que no pueden ser convertidas? por ej.

```
int main(){

        auto c = std::make_unique<Cuadrado>(3);
        Triangulo* t = dynamic_cast<Triangulo*>(c.get());
}
```

    - Compila porque la conversión se realiza en tiempo de ejecución. Pero… la conversión no es posible, ya que no cumplimos las reglas del downcasting

Polimorfismo dinámico

- Polimorfismo - Downcasting:
  - En t (resultado de la conversión) tendremos un nullptr -> Antes de hacer uso del puntero tendremos que chequearlo para "evitar problemas" (segmentation fault)

```
int main(){
        auto c = std::make_unique<Cuadrado>(3);
        Triangulo* t = dynamic_cast<Triangulo*>(c.get());

        //if(t) //Comprueba que t apunta a algo, es equivalente a if(t!=nullptr)
        if(t!=nullptr){t->getAltura();}
        return 0;
}
```

Polimorfismo dinámico

- Polimorfismo - Downcasting:

  Si únicamente queremos utilizar punteros inteligentes debemos usar

  http://www.cplusplus.com/reference/memory/dynamic_pointer_cast/

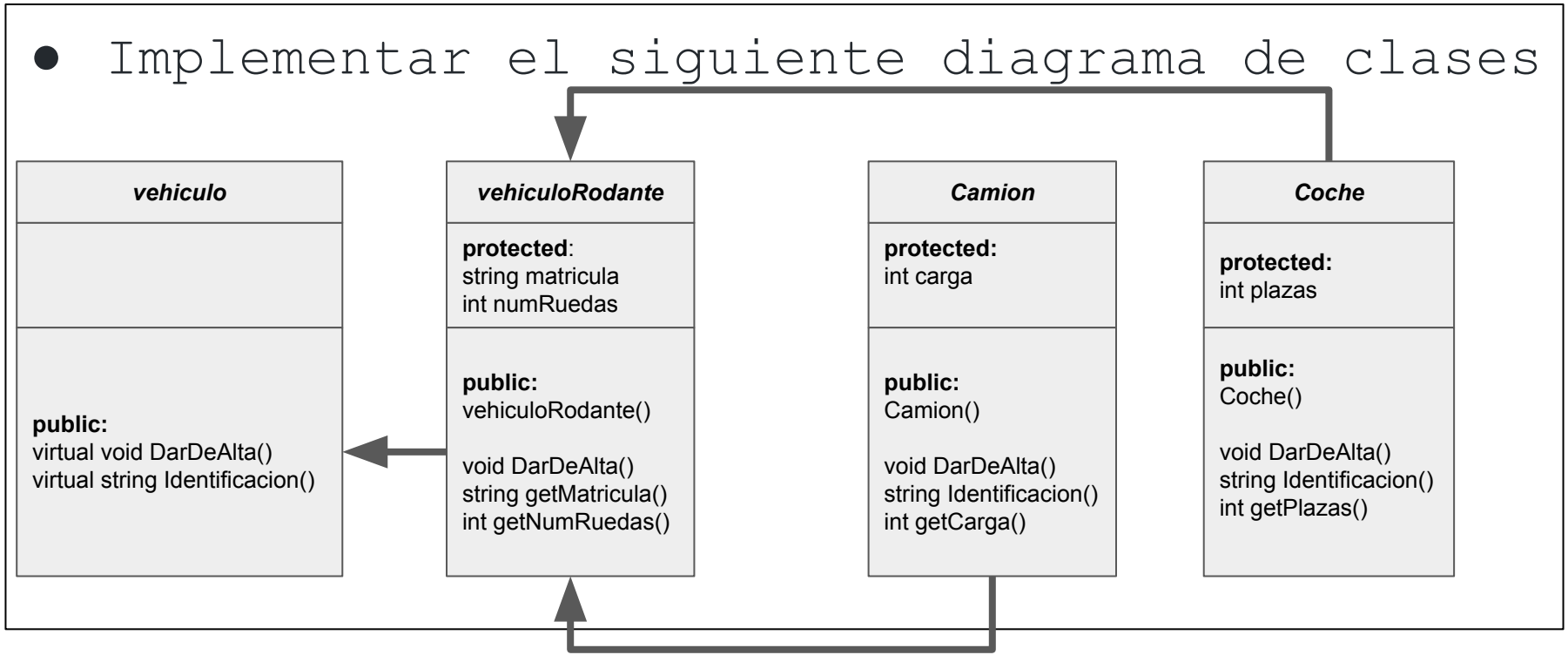
GitHub

- Polimorfismo:
  - Funciones virtuales y polimórficas
  - Funciones virtuales puras y clases abstractas
  - Downcasting

Índice

Ejemplos

- Implementar el siguiente diagrama de clases



Ejemplos

- Los métodos DarDeAlta de cada clase deben pedir al usuario por consola las variables miembro de su clase
- Los métodos Identificación deben devolver un string con la palabra “camion” o “coche” según el caso
- Los getter devolver las variables miembro correspondientes

Ejemplos

- Se desea que el programa principal gestione un menú para dar de alta coches y camiones, que deben ser almacenados en un vector de vehiculosRodantes. El menú también debe dar la opción de salir del programa
- Antes de finalizar el programa, se desea mostrar por consola todos los datos relativos a los vehículosRodantes almacenados en el vector: Código