Programación II

Tema 2

jtorresmat@nebrija.es

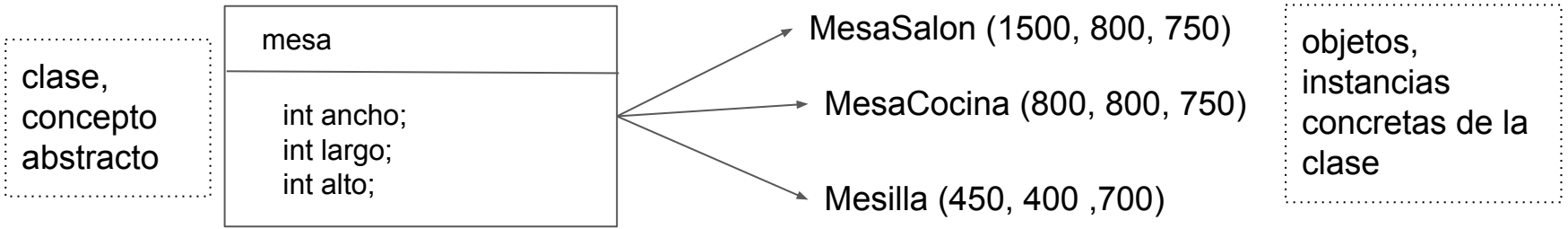
Índice

Clases

- Hasta ahora hemos usado C++ “casi casi” como si fuera C.
  - Durante Programación I hemos hecho constantes “actos de fe” para usar tipos de datos de la biblioteca estándar
- En Programación II vamos a entender esos pendientes, vamos a empezar a hacer uso de la programación orientada a objetos (POO)
- Y qué mejor que empezar por un objeto… :)
  - Objeto es una unidad de información que engloba datos y procedimientos para tratar esos datos
- Hasta ahora (Programación I) nuestros programas tenían datos y funciones para trabajar sobre esos datos, por separado. Pero con POO un programa será un conjunto de objetos, y cada objeto contendrá (encapsulará) datos y funciones.

Clases

- Esos objetos provendrán de clases. Al igual que ocurre en la vida real, en la que existen por ejemplo muchas mesas (la del laboratorio, la del comedor, la de la cafetería, la de vuestro compañero, la mía, …) podemos agruparlas bajo el concepto abstracto de mesa.
- De tal manera que una mesa concreta, por ejemplo la mesa de mi cocina es una instancia de ese concepto abstracto general mesa



Clases

- ¿Qué aporta trabajar con objetos? ¿qué ventajas tiene?
  - Permite reutilizar código. Si tenemos programada una clase botón, podremos instanciar los objetos que necesitemos para nuestra interfaz.
  - Abstracción: un programa será más sencillo de leer y comprender ya que nos permitirá ocultar los detalles de la implementación dejando visibles sólo las partes más relevantes.
  - Modificabilidad: con un programa estructurado en clases/objetos es más fácil realizar cambios (facilita el mantenimiento del programa).
  - Fiabilidad: al dividir el programa en partes más pequeñas será más fácil realizar pruebas y aislar errores.

Clases

- Eso sí….
  - Hay nuevos tipos de datos y nuevas características del lenguaje que debemos aprender -> Toca estudiar... ¡¡¡más aún!!!
  - Hay que aprender a pensar de una forma diferente a la que pensamos en la programación tradicional. Tenemos que aprender a abstraer características para poder empezar a pensar en objetos.

Clases

- Clases:
  - Son la unidad básica de encapsulación en C++ y permiten la creación de objetos
  - ¿Qué son? Un nuevo tipo de datos (estilo estructuras*)
  - Definición (sintaxis): class Etiqueta{ ... };
    - Recomendable: primera letra nombre en mayúscula.
  - Generalmente incluye datos y código para trabajar con ellos: métodos y atributos.
  - Una clase es una abstracción lógica, una declaración
  - El objeto es una instancia de una clase

*Con particularidades que ya veremos

Clases

- Podemos decir que una clase es un mecanismo nuevo para poder definir nuevos tipos de datos como hacíamos en Programación I con las estructuras
    - Ejemplo n° complejo con estructura
    - Ejemplo n° complejo con clase
- Declaración clase:

```
class Etiqueta{
    [nivel de acceso 1:]
        declaración de miembros de la clase
    [nivel de acceso 2:]
        declaración de más miembros de la clase
};
```

Clases

- Ejemplo en detalle:
  - El nombre de la clase es: Persona
  - Tiene un constructor
  - Tiene atributos (variables de la clase):
    - Nombre
    - Edad
    - Altura
  - Por pasos…
    - “public” (palabra reservada indica que lo que viene después es de acceso público. Podremos acceder desde nuestro código)

Clases

- **//constructor** (es un comentario)
- constructor (método con el mismo nombre que la clase)
- atributos (variables miembro)

- Vamos a instanciar un objeto persona:

```
Persona alumno("Euclides", 22, 185);
```

Al declarar una variable de una clase lo primero que hace el compilador es llamar al constructor

Clases

- Características del constructor:
  - Tiene el mismo nombre que la clase
  - No devuelve ningún valor
  - Puede tomar o no parámetros de entrada.
- En nuestro caso particular toma 3 parámetros de entrada que utilizamos para inicializar los atributos de la clase Persona.
- Podremos crear tantas instancias de la clase como queramos es un “tipo de datos”
- ¿Cómo accedemos a los atributos?

```
std::cout<<"Hola "<<alumno.nombre<<
        ". Tienes "<<alumno.edad<<"años"<<
        ", y mides "<<alumno.altura<<"cm."<<std::endl;
```

Clases

Ejemplo: Partiendo del ejemplo anterior (clase Persona), crear un programa que pregunte al usuario si quiere dar de alta una persona, y en caso afirmativo que pida la información.

Ejercicio propuesto: Modificar el ejemplo para que mientras no se diga lo contrario se vaya añadiendo personas al vector de personas. Y antes de finalizar muestre por pantalla la información almacenada en el mismo por el terminal.

Clases

¿En el ejemplo anterior podríamos modificar los atributos de la persona creada? -> Cambiar nombre

¿En el ejemplo anterior podríamos asignar valores que estén fuera del rango del tipo de variable declarada? -> Edad negativa

¿Daría error la ejecución del programa?

Clases

No daría error, pero…

Nuestro código es “vulnerable” todos los datos son públicos (se pueden modificar desde cualquier parte del código)

La edad no tiene sentido que sea negativa o superior a un valor determinado

Y encima el resultado es “impredecible” al ser tipo unsigned

¿Qué podríamos hacer?

Usar métodos getter y setter:

- Típicamente tienen nombre getNombreVariable(), setNombreVariable()
- Nos sirven para escribir y leer su valor.

Clases

Ejemplo con getter y setter

¿Podríamos seguir escribiendo en los atributos?

```
grupo.at(0).edad=500;

std::cout<<"Hola "<<grupo.at(0).nombre<<
        ". Tienes "<<grupo.at(0).edad<<"años"<<
        ", y mides "<<grupo.at(0).altura<<"cm."<<std::endl;
```

Clases

- Miembros públicos y privados (niveles de acceso):
  - Públicos son de libre acceso (**public:**)
  - Privados sólo son accesibles desde dentro de la clase (**private:**)

Añadimos en el ejemplo anterior la palabra reservada private: (antes de los atributos)

¿Qué pasará? -> Todo lo que sea private no será accesible "desde fuera" de la clase

Clases

- Encapsulación:
  - Para una clase los únicos miembros accesibles desde el exterior son los declarados como públicos.
  - Los miembros privados sólo son accesibles desde el interior de la clase, no son accesibles desde fuera (otras funciones y clases exteriores a la clase)
  - Los niveles de acceso son el mecanismo que debemos usar para conseguir el encapsulamiento. Que no es otra cosa que conseguir que cada objeto sea “autónomo”, evitando que lo que ocurra dentro del él sea visible al exterior. Protegemos la información que contiene controlando el flujo de información (cada objeto responderá a determinadas peticiones y proporcionará determinada información al exterior)

Clases

- Encapsulación:
  - Mediante este concepto conseguimos que cada objeto sea una especie de caja negra. De tal manera que desde el exterior un objeto sea un ente que responde a una serie de mensajes públicos (interfaz de la clase)
  - En una clase el nivel de acceso de sus miembros por defecto será privado
    - En una estructura el nivel de acceso de sus miembros por defecto es publico.

Documentación en GitHub

- https://github.com/Nebrija-Programacion/Programacion-II
  - Clases: (https://github.com/Nebrija-Programacion/Programacion-II/blob/master/temario/clases.md)
    - Funciones miembro
    - Variables miembro
  - Miembros públicos y privados (https://github.com/Nebrija-Programacion/Programacion-II/blob/master/temario/clasesII.md)

Índice

- Introducción
- Clases
- Ejemplos

Ejemplos

1. Clase que describa un vehículo rodante (abstracción).

2. Programa que tenga 2 clases, triángulo (base y alt.) y rectángulo (lado1, lado2):
   - Pida por pantalla base y alt al usuario. Muestre área.
   - Pida por pantalla lados rectángulo. Muestre área y perímetro.

Código