Programación II

Tema 3

jtorresmat@nebrija.es

Índice

Constructor y destructor

- Son dos funciones miembro especiales de una clase
- Se encargan de asegurar que la creación y la destrucción de un objeto de una clase se realizan de forma correcta
    - Constructor -> creación del objeto
    - Destructor -> destrucción del objeto

Constructor y destructor

- Constructor:
  - La clase es un tipo de datos -> podemos inicializar las variables que generemos (que son los objetos)
  - Ej. inicialización tipos de datos:
    - int MiEntero=8;
    - std::string Frase{“Hola, parece que hace buena tarde”};
  - ¿Cómo inicializamos un objeto?
    - ¿Así?
    - Inicializa los atributos, pero los datos están “expuestos”... ¿Encapsulamiento? :(

Constructor y destructor

- Constructor:
  - ¿Tenemos alguna forma de inicializar los atributos manteniendo el principio de encapsulación?
    - Si, haciendo uso del constructor.
  - Definición: función miembro de una clase que se llama de modo automático al crear un objeto. Se encarga de inicializar sus variables miembro.

Constructor y destructor

- Constructor:
  - El/los constructores de una clase son unos métodos especiales que sirven para inicializar un objeto al mismo tiempo que se declara.
  - Características:
    - Tienen el mismo nombre de la clase
    - No tienen valor de retorno (tampoco **void**)
    - Suelen ser públicos
    - Es el primer método que se llama al declarar un objeto de esa clase

Constructor y destructor

- Constructor:
  - Más características:
    - Una clase puede tener varios constructores (sobrecarga)
    - Si no se define ninguno el compilador nos proporciona uno por defecto
  - Ejemplos:
    - Inicialización objeto con constructor
    - Varios constructores
    - Ningún constructor

Constructor y destructor

- Constructor:
  - Constructor por defecto:
    - Lo crea el compilador si la clase no tiene definido uno.
    - Si la clase tiene un constructor definido e intentas usar el constructor por defecto dará error (el compilador sólo lo crea si no hay uno)

Constructor y destructor

- Constructor copia:
  - Es un constructor especial que se ejecuta cuando un objeto se asigna a otro.
  - Realiza una copia literal de los atributos de un objeto sobre el otro.
  - Recibe por parámetro una referencia constante a un objeto de la misma clase.
  - El constructor copia se llama siempre que se haga copia de un objeto (incluyendo cuando el objeto se devuelve de una función o cuando se pasa como parámetro por copia).
  - Ejemplo

Constructor y destructor

- Destructor:
  - Definición: función miembro de una clase a la que se le llama cuando el objeto va a dejar de existir
  - Características:
    - Se llama *~NombreClase*()
    - Es único (no admite sobrecarga)
    - No tiene argumentos ni retorno
    - Si no se define uno el compilador te “asigna uno de oficio”

Constructor y destructor

- Destructor:
  - ¿Cuando deja de existir un objeto?
    - Si el objeto es local -> al terminar el ámbito
    - Si el objeto es global -> al terminar el programa
    - Si el objeto se ha creado con **new** -> con el **delete**

Ejemplo

Constructor y destructor

- Ejercicios propuestos:
  - Ejercicio 1 de la colección sin lambdas
  - Ejercicio 2 de la colección

https://github.com/Nebrija-Programacion/Programacion-II/blob/master/EJERCICIOS.md

Índice

Gestión de errores con excepciones

- Hemos visto que con getter/setter podemos hacer gestión de errores, mostrando mensajes por pantalla, limitando los datos, etc… Pero no es lo habitual.
- La forma correcta de gestionar errores es mediante el uso de excepciones.

Gestión de errores con excepciones

- Ejemplo en setter:

```
class Persona{
public:
 ...
 void setEdad(unsigned short int _edad){
      if (_edad > 150 || _edad <0){
         cout << "Hay un error con la edad indicada: " << _age
              << " . La pongo a 0"
              << endl;
         age = 0;
      }else{//Todo bien...
         age = _age;
      }
  }
   ...
};
```

Gestión de errores con excepciones

- Ejemplo con setter:
  - ¿”Estamos bien” con el ejemplo?
    - Y si nuestro programa no fuera de consola?
    - Y si queremos volver a pedir la edad, porque el 0 no nos vale? ¿Cómo avisamos a nuestro programa?
    - Debemos evitar que los getter/setter se comuniquen por consola (como ocurre con el resto de funciones/metodos). Las funciones deben interaccionar mediante parámetros de entrada y valores de retorno (programación funcional)
  - “Estaremos mejor” con excepciones :)

Gestión de errores con excepciones

- Excepciones:
  - Definición: Son situaciones anómalas durante la ejecución del código (no sólo errores). Si no las gestionamos el programa generalmente terminará de forma abrupta (no se guardarán buffers, no se cerrarán de forma controlada determinados recursos, … en definitiva dejaremos de controlar nuestro sistema …)

Gestión de errores con excepciones

- Excepciones:
  - Ejemplo excepción sin gestión.
  - Las excepciones son mensajes de “socorro, algo va mal” a nuestro código que pueden ser “escuchadas” por el.
  - La gestión de excepciones consiste en transferir el control del programa desde donde ha ocurrido un error a otra donde se pueda gestionar

Gestión de errores con excepciones

- Ejemplo del "setter" con excepciones

```
class Persona{
public:
 ...
 void setEdad(unsigned short int _edad){
      if (_edad > 150 || _edad <0){
         throw string{"La edad no puede ser >150 ni <0"};
         age = 0; //Esta instrucción se llega a ejecutar???
      }else{//Todo bien...
         age = _age;
      }
  }
  ...
};
```

- Está lanzando una excepción tipo string.

Gestión de errores con excepciones

- ¿Cómo gestionamos esa excepción?
  - Bloque "try-catch"

```
try {
    //Instrucciones a "vigilar"
}catch (std::string mensaje) {
    //Gestión de excepciones
}
```

  - Dentro del try "código a vigilar"
  - Dentro del catch "comprobación excepción"

Gestión de errores con excepciones

- Ejemplo 1
- Ejemplo 2

Gestión de errores con excepciones

- Un bloque try, puede tener varios catch asociados. Importante la prioridad (de error concreto a general).
- Dentro del bloque try debería existir una expresión throw explícita **(throw "Loquesea":)** o implícita (generada por otro elemento)
- Cuando se produce una excepción se busca un catch apropiado de su bloque, si no se encuentra se retrocede al anterior hasta encontrarlo (Ejemplo)

Gestión de errores con excepciones

- Los tipos de expresión del **throw** y del **catch** deben coincidir (ojo, fuertemente tipado… no es lo mismo un int que un unsigned int)
- Si no se encuentra **catch** adecuado se sale del programa (como si fuera un error sin gestión, con sus mismos problemas...)
  - Para evitar esto existe un catch general (**catch(...)**) Ejemplo

Índice

Ejemplos

- Ejemplo:
  - Atrapar una excepción para error out_of_range al intentar acceder a una posición de un array mayor que su tamaño.

    Enunciado.

    Solución.

Ejercicio propuesto

- Ejercicio 3 de la colección:
  - Crear una clase con:
    - Una variable miembro decimal
    - Funciones miembro:
      - dividePor(float otroNum);
      - multipPor(float otroNum);
      - sumaA(float otroNum);
    - Crea las excepciones que consideres oportunas
  - El programa debe:
    - Pedir al usuario el número
    - Pedir al usuario la operación que desea realizar
    - Pedir al usuario el segundo número a operar
    - Mostrar resultado por pantalla

Documentación en GitHub

- https://github.com/Nebrija-Programacion/Programacion-II
  - Constructores:

    (https://github.com/Nebrija-Programacion/Programacion-II/blob/master/temario/clasesIII.md)

  - Control de excepciones:

    (https://github.com/Nebrija-Programacion/Programacion-II/blob/master/temario/excepciones.md)

Solución ejercicios

- Ejercicio 1 colección
- Ejercicio 2 colección