Laboratorio de Computación II

Programación orientada a objetos: Constructores y destructores + Sobrecarga

Atributos y métodos

- Muy resumidamente, una clase nos permite crear un objeto con atributos encapsulados y métodos que nos permiten interactuar con ellos.

- Éstos métodos se llaman según la necesidad de realizar las tareas que la clase provee.
  - Ejemplo: objCliente.cargar(), objExamen.getNota(), etc

- Pero.. ¿cómo controlamos lo que el objeto tiene al crearse? o ¿lo que el objeto hace al destruirse?

Constructores y destructores

Constructor

Método que se ejecuta solo cuando el objeto se crea, es decir, cuando se instancia o cuando se crea a partir de memoria dinámica con new/malloc.

Destructor

Método que se ejecuta solo cuando el objeto se elimina, es decir, cuando finaliza su alcance o se ejecuta un delete/free en el uso de memoria dinámica.

Constructor

- Método de clase que se ejecuta solo cuando se instancia el objeto.

- Debe llamarse obligatoriamente igual a la clase y no puede devolver un valor de retorno. Ni siquiera void.

```
class Auto{
  private:
    float velocidad;
  public:
     Auto(){
       velocidad = 0;
     }
};
```

Constructor de la clase Auto que asigna cero a la propiedad velocidad. De esta manera, no puede haber ningún objeto del tipo Auto con basura al momento de la instancia.

Destructor

- Un método que se ejecuta solo al momento de la destrucción de un objeto. Es decir, cuando finaliza su alcance o se libera la memoria de un puntero.

```
class Auto{
  private:
    float velocidad;
  public:
     ~Auto(){
       cout << "Auto eliminado";
       cout << " a " << velocidad;
       cout << " kms/h";
     }
};
```

Destructor de la clase Auto que muestra la velocidad a la que iba al eliminarse.

¿Cuándo utilizarlos?

- Aunque parezca una obviedad, cuando necesito que algo se ejecute obligatoriamente al crear un objeto o al eliminar un objeto.

- Por ejemplo, una clase vectorEnteros que debe recibir obligatoriamente el tamaño para pedir memoria (constructor) y liberar la memoria al no necesitarlo más (destructor).

Múltiples constructores

- Se puede tener todos los constructores que sean necesarios, siempre y cuando se diferencien por los parámetros que reciben.

¿Cóoooomo?

¿Se puede tener una función que se llame igual que otra?

Sí, se llama sobrecarga y la regla es que deben diferenciarse por los parámetros que reciben.

Sobrecarga de funciones y/o métodos

- Es la capacidad que permite un lenguaje de diferenciar dos funciones con el mismo nombre a partir de los parámetros que recibe. *

```
/* Devuelve un número al azar
entre 1 y max */
int alAzar (int max){
  return 1 + rand() % max;
}

/* Devuelve un número al azar
entre min y max */
int alAzar (int min, int max){
  return min + rand() % (max-min);
}
```

Sin preocuparnos por la complejidad del cálculo del número al azar, podemos ver que dos funciones llamadas iguales se diferencian por sus parámetros.

Si el programador llama a alAzar con un parámetro, el compilador asume que llama a la primera y sortea el número entre 1 y el valor enviado.

Si el programador llama a alAzar con dos parámetros, el compilador asume que llama a la segunda y sortea el número entre los parámetros enviados.

Sobrecarga de constructores

```
class Auto{
  private:
    float velocidad;
  public:
     Auto(){
       velocidad = 0;
     }
     Auto(int v_inicial){
       velocidad = v_inicial;
     }
};
int main(){
  Auto rayo;
  Auto sally(100);
  return 0;
}
```

Siguiendo la lógica anterior:

Al estar sobrecargado el método de construcción Auto() y Auto(int). En main tenemos dos instancias del mismo objeto pero que llaman a métodos distintos.

Por eso, el objeto *rayo* tiene su velocidad al iniciarse con un valor de 0.
Por otro lado, el objeto *sally* toma el valor de 100 al iniciar ya que se llama al constructor que recibe un parámetro y lo asigna a la velocidad.

Ejemplo práctico

Vamos a crear una clase Cadena.

Clase Cadena

- Debe tener dos constructores:
  - El primero debe recibir el tamaño, asignar memoria dinámica y asignar el caracter terminador al principio.
  - El segundo debe recibir un vector de char, asignar memoria dinámica y copiar el vector a la cadena.
- Destructor:
  - Debe liberar la memoria dinámica (si corresponde).

- Otros métodos:
  - getCadena() debe devolver la cadena.
  - setCadena(char *) debe asignar la cadena enviada hasta cubrir el tamaño del objeto.
  - getLargo() debe devolver el tamaño real de la cadena.