Laboratorio de Computación II

Programación orientada a objetos
Parte 1

Programación orientada a objetos

- Es un paradigma de programación en el que predominan entidades (objetos) que interactúan entre sí.
- Al resolver un problema complejo, contamos con elementos necesarios para la solución: cliente, factura, producto, venta, devolución, pago, etc.
- Cada elemento mencionado tiene sus características que lo identifican y comportamiento. Ejemplo: Un cliente compra (comportamiento) y una factura está impaga (característica)

Programación orientada a objetos

Algo innovador que trabajaremos con la POO es que las entidades tendrán atributos (variables) y comportamiento (métodos).

```
class Persona{
  private:
    char apellido[50];
    char nombre[50];
    int dia, mes, anio;
    float altura;
  public:
    bool hablar();
    void dormir(int horas);
    bool saltar(float alto);
    bool correr(float metros);
};
```


Una clase define un molde de cómo representar para nuestro programa una entidad.
Pero ese molde está vacío, no contiene información ni puede hacer cosas hasta que no generemos una instancia de la clase.
Una instancia es un objeto, es decir, una variable de Persona.

Objeto

- Un objeto es una instancia de una clase que posee atributos y puede realizar acciones.

- Si en nuestro programa queremos representar a dos personas podemos instanciar dos objetos de la clase Persona.

```
Persona humano1;
Persona humano2;
```

Atributo

Una característica de la POO es que los atributos de una clase no deberían ser accesibles desde fuera de ella (privados). Esto se llama encapsulamiento. Para tal fin utilizaremos diferentes modificadores de acceso (private, public, protected, friend).

```
class Auto{
  private:
    float velocidad;
};

int main(){
  Auto coche;
  coche.velocidad = 150;
}
```

La clase Auto tiene la velocidad encapsulada. Quiere decir que es privada y no se puede modificar desde afuera de la clase.

De la manera en que está desarrollada no es posible modificar el valor de la velocidad.

Métodos

- En la POO los objetos tienen comportamiento. Éstos pueden acceder a los atributos de la clase y modificarlos si es necesario. Los métodos pueden ser privados o públicos. Pero recordar que si un método es privado entonces está encapsulado.

- ¿Cómo podría un objeto de la clase Auto aumentar su velocidad?.

Métodos

Acelerar sería la acción (pública) para poder aumentar la velocidad (privada). ¿Cómo hace el auto para ir más rápido? Eso queda dentro de la abstracción de la caja negra que es una clase.

Podemos modificar la velocidad del auto mediante el método acelerar que requiere se le indique en qué magnitud debe hacerse la aceleración.

¿Puede un auto ir a una velocidad menor a 0?

```
class Auto{
  private:
    float velocidad;
  public:
    void acelerar(float valor){
       velocidad += valor;
    }
};

int main(){
 Auto coche;
 coche.acelerar(50);
 coche.acelerar(100);
}
```

Métodos

Acelerar sería la acción (pública) para poder aumentar la velocidad (privada). ¿Cómo hace el auto para ir más rápido? Eso queda dentro de la abstracción de la caja negra que es una clase.

Podemos modificar la velocidad del auto mediante el método acelerar que requiere se le indique en qué magnitud debe hacerse la aceleración. Pero sólo si este valor es positivo.

¿Permite esta clase saber a qué velocidad marcha el auto?

```
class Auto{
  private:
    float velocidad;
  public:
    void acelerar(float valor){
      if (valor > 0)
        velocidad += valor;
    }
};

int main(){
 Auto coche;
 coche.acelerar(50);
 coche.acelerar(100);
}
```

Setters y getters

Métodos de clase que permiten establecer y obtener valores del objeto

- Es muy común que quiera que ciertos atributos puedan ser modificados o accedidos desde afuera de la clase. Los métodos set y get son métodos que permiten realizar esas acciones.

- No tienen ninguna particularidad especial. Es una convención en la que los métodos set asignan información a los atributos y los get obtienen información de un atributo.

Setters y getters

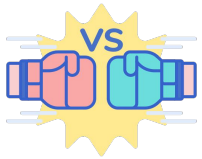
```
class Auto{
  private:
    float velocidad;
    int anioFabricacion;
  public:
    void acelerar(float valor){
      if (valor > 0) velocidad += valor;
    }
    void setAnioFabricacion(int anio){
      if (anio > 1900) anioFabricacion = anio;
    }
    float getVelocidad(){
      return velocidad;
    }
};
int main(){
 Auto coche;
 coche.setAnioFabricacion(2020);
 coche.acelerar(100);
 cout << "Velocidad: " << coche.getVelocidad();
}
```

Una clase Auto cuyos métodos permiten establecer el año de fabricación, permiten aumentar la velocidad mediante el método acelerar y obtener la velocidad actual.

Sin embargo, no puede frenar.

Structs vs Clases

```
int main(){
    struct Auto a;
    cargar_auto(&a);
    mostrar_auto(a);
    guardar_auto(a);
}
```




```
int main(){
  Auto a;
  a.cargar();
  a.mostrar();
  a.guardar();
}
```

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.

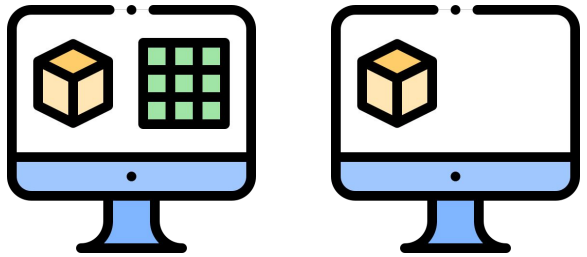
```
class Auto{
  private:
    float velocidad;
  public:
     ~Auto(){
       cout << "Auto eliminado";
       cout << " a " << velocidad;
       cout << " kms/h";
     }
};
```

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 (constructor) o al eliminar un objeto (destructor).

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



Ejemplos en C/C++

Atribuciones a obras de terceros

- Iconos diseñados por DinosoftLabs from Flaticon
- Iconos diseñados por Flat Icons from Flaticon
- Iconos diseñados por Freepik from Flaticon