Laboratorio de Computación II

Programación orientada a objetos

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;
```

Atributos

- 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).

Ejemplo:

```
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.

Así como está no hay manera de 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 privados 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.

¿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

- 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* obtiene 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 cuyo objeto permite establecer el año de fabricación, permite 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();
}
```

Otras características

- Construcción y destrucción: La capacidad de realizar acciones al crearse y al destruirse el objeto de manera automática.

- Composición y agregación: Diferentes maneras de relacionar clases.

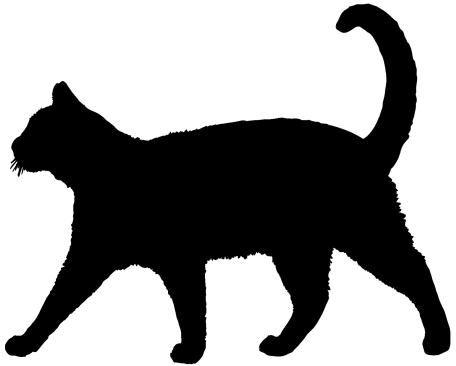
- Herencia: Poder crear una clase derivada a partir de una clase base de la cual obtendrá todos los rasgos.

- Polimorfismo: La posibilidad de que varios objetos de diferentes clases, pero con una base común, se comporten diferentes ante una misma acción.

Pero las personas y los autos son aburridos.

¿Quiénes sí son divertidos?

Los michis



Clase Gato

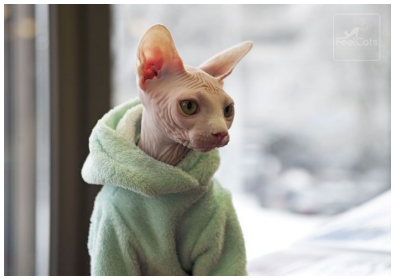
- Antes de diseñar la clase Gato, debemos preguntarnos: ¿Por qué razón nuestro sistema necesita una clase Gato?
    - ¿Es un juego protagonizado por un gato?
    - ¿Es un registro de una mascota para un sistema de veterinaria?
    - ¿Es un sistema que registra los gatos disponibles para su adopción?

- ¿Qué características tendrá un gato en nuestro sistema?

- ¿Qué acciones realizará un gato en nuestro sistema?

Clase Gato

- Un gato tendrá:
  - Nombre
  - Día, mes y año de nacimiento
  - Tipo de gato
  - Peso
  - Altura
  - Colores



- Un gato podrá:
  - Cargarse
  - Mostrarse
  - Establecer y obtener sus atributos