Laboratorio de Computación II

Programación orientada a objetos
Herencia simple

Características de la POO

- Construcción y destrucción: La capacidad de realizar acciones al crearse y al destruirse el objeto de manera automática.
- Encapsulamiento: Restricción del acceso a elementos (propiedades y métodos) que no deban accederse desde fuera de clase. Pueden accederse si se les desarrolla un método público para tal fin.
- 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 diferente ante una misma acción.

Herencia

Tipo de relación entre clases que permite que una clase A herede atributos y métodos de una clase B.

```
class Animal{
  protected:
    Fecha nacimiento;
    char tipo[50];
  public:
     Animal();
     void mostrar();
};
class Mascota:public Animal{
  private:
    char nombre[50];
    Persona duenio;
  public:
    void mostrar();
};
```


Como se puede observar, una Mascota está compuesta por un nombre y dueño de la clase Persona. Pero, como una Mascota *es un* Animal también incorporará los atributos y comportamientos de esa clase.
Nótese como la Mascota no tiene un Animal (composición) sino que es un Animal (herencia).

Especificador de acceso protected

```
class Animal{
  protected:
    Fecha nacimiento;
    char tipo[50];
  public:
    Animal();
    Fecha getNacimiento();
    void setNacimiento(Fecha);
    char* getTipo();
    void setTipo(char *);
};
```

El encapsulamiento sigue estando presente incluso en la herencia. Si bien una clase derivada recibe todos los atributos y comportamientos de la clase base. No podrá acceder directamente a aquellos que sean privados.
Para ello existe el especificador de acceso protected. Siendo inaccesible desde afuera de la clase salvo para las que heredan de ella.

Acceso a los miembros de la clase base

```
void Mascota::mostrar(){
  cout << "Nacimiento: ";
  nacimiento.mostrar();
  cout << "Tipo: ";
  cout << tipo;
  cout << "Nombre: ";
  cout << nombre;
  cout << "Dueño: ";
  duenio.mostrar();
}

int main(){
  Mascota michi;
  // Seteo de datos
  michi.mostrar();
}
```

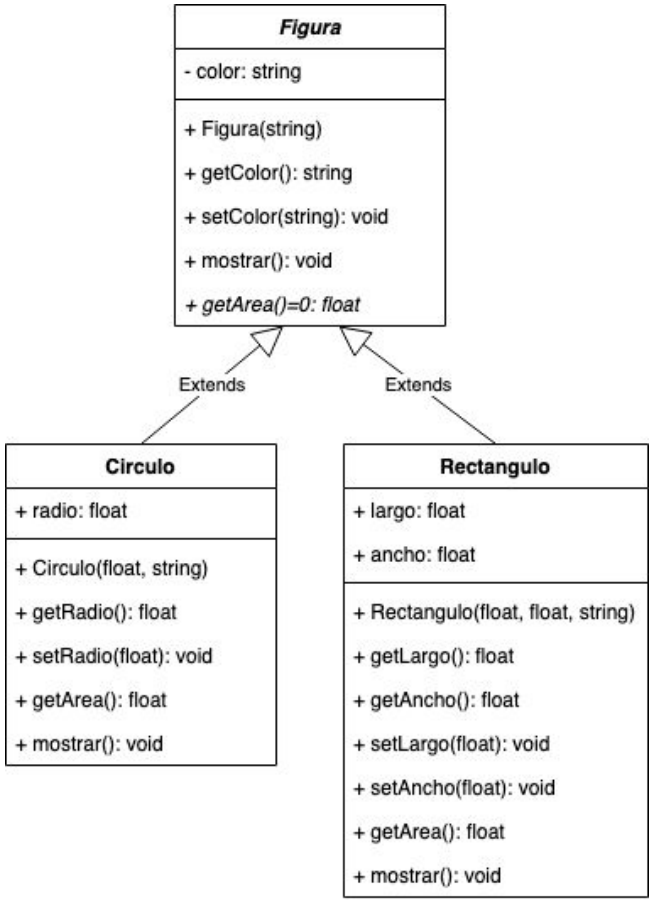
Dentro de la clase Mascota se puede acceder a los atributos protegidos (no privados) directamente como si fuesen atributos propios ya que, conceptualmente, son atributos de la clase derivada desde el momento en que se hereda de la clase base.
El método mostrar ejemplifica eso, aunque, hay una manera más elegante de resolver esto.

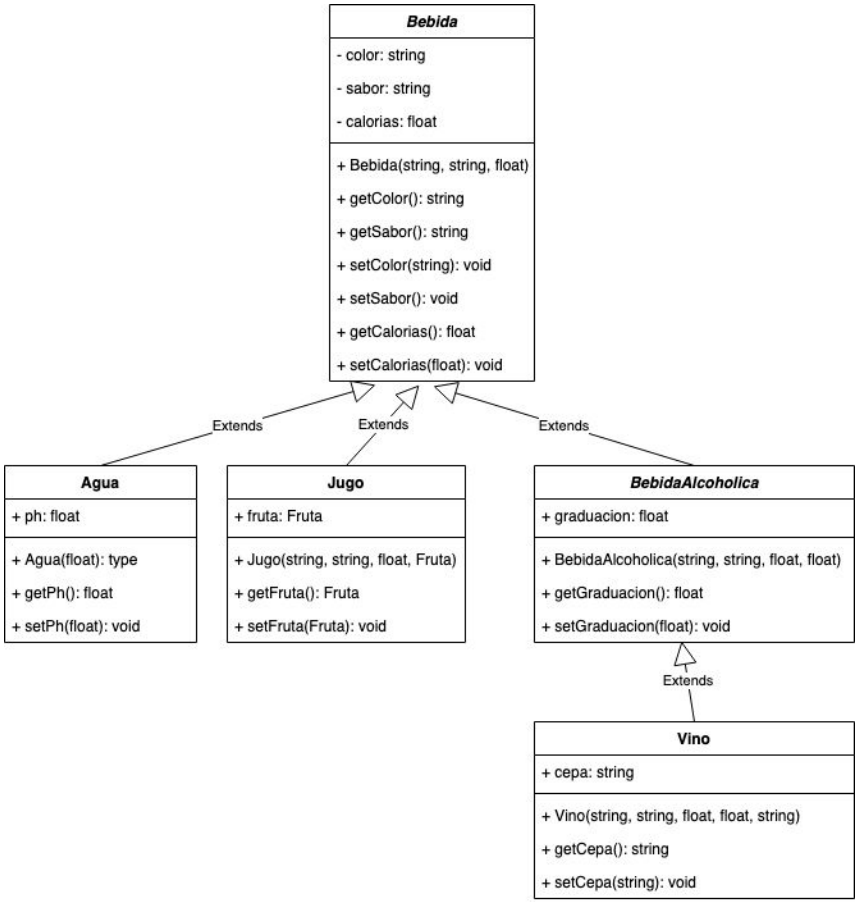
Acceso a los miembros de la clase base

```
void Animal::mostrar(){
  cout << "Nacimiento: ";
  nacimiento.mostrar();
  cout << "Tipo: ";
  cout << tipo;
}
void Mascota::mostrar(){
  Animal::mostrar();
  cout << "Nombre: ";
  cout << nombre;
  cout << "Dueño: ";
  duenio.mostrar();
}
int main(){
  Mascota michi;
  // Seteo de datos
  michi.mostrar();
}
```

El método mostrar de Animal define cómo se muestra un animal de forma general.
Al invocar el método mostrar de Mascota, en principio, se ejecuta al método mostrar del componente Animal que hay en Mascota para luego mostrar los atributos propios de la mascota.
En main, el resultado es el mismo que el ejemplo anterior.
Este abordaje es mucho más escalable que el anterior.

Otros ejemplos de herencia





Ejemplos en C/C++