Programación Orientada a Objetos

Atributos-Mensajes

Contenidos

- Variables de instancia
- Mensajes o métodos
- Encapsulamiento
- Creación
- Destrucción

Un poco de notación

Antes de empezar con un lenguaje específico, un poco de pseudo-codigo

Persona A → A es de tipo Persona, que puede ser un tipo o clase

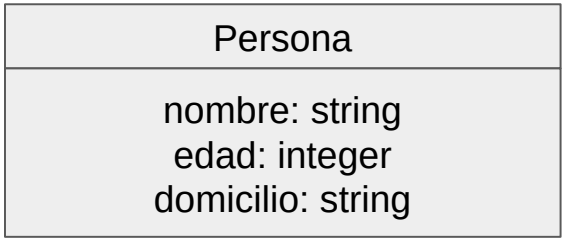
A->nombre → accede al atributo o propiedad nombre del objeto A, similar a A['nombre'] si fuera un arreglo

Notación para funciones:

```
function nombre (parametro, &parametro) {
        codigo….
        return VALOR
}
```

Variables de instancia o propiedades

Supongamos un Objeto como una estructura de datos o un arreglo hetereogéneo:




```
dimension A(3);

A[“nombre”] = “Juan”;
A[“edad”] = 20;
A[“domicilio”] = “San Martin 100”;
```

Para modificar los datos, necesita una función:

```
function CambiarNombre(&A, otro) {
        A[“nombre”] = otro;
}

CambiarNombre(A, “Pedro”);
```

Mensaje

Conceptualmente es solicitarle algo a un objeto

- Auto->abrirPuerta()
- Persona->decimeTuNombre()
- Nave->despegar()

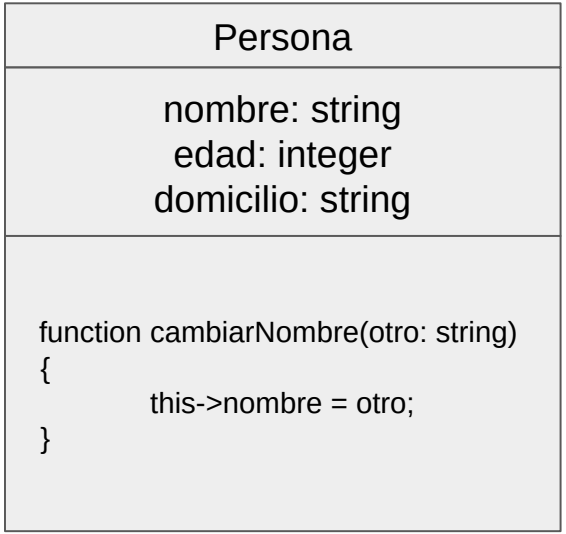
Pueden llevar un parámetro

- Circulo->cambiarColor(“rojo”)
- Auto->acelerar(“100”)

```
function NombreClase::nombreMensaje (parametro, &parametro) {

        codigo….

        return VALOR

}
```

Mensajes o métodos

Supongamos un Objeto como una estructura de datos o un arreglo hetereogéneo, pero que “porta” sus propias funciones:




```
Persona A, B
B->nombre = “Luis”;
A->nombre = “Juan”;
A->edad = 20;
A->domicilio = “San Martin 100”;

A->cambiarNombre(“Pedro”);
```

Encapsulamiento

Acceso directo:

- Requiere conocer la estructura, el tipo de datos
- Quedan expuestos a manipulación incorrecta

Acceso por método o mensaje:

- Oculta la estructura interna o implementación

Surge concepto de **Encapsulamiento**

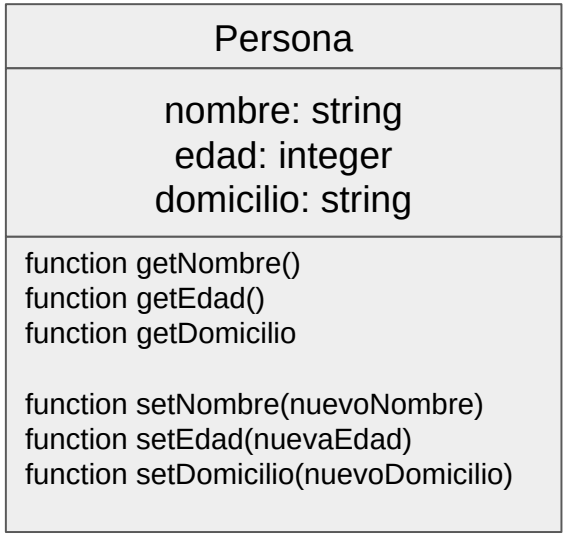
Encapsulamiento

Métodos de lectura *get*:

- getNombre
- getEdad
- getDomicilio

Métodos de actualización *set*

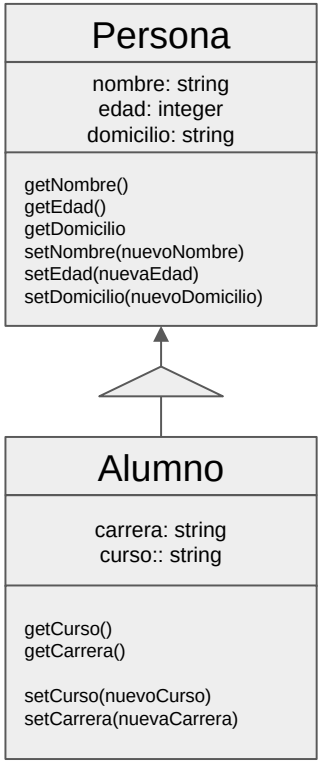
- setNombre
- setEdad
- setDomicilio




```
function getNombre() {
        return this->nombre;
}
```

```
function setNombre(nuevoNombre) {
        this->nombre = nuevoNombre;
}
```

Encapsulamiento y Herencia



Alumno A

A->setCurso(“POO”)

A->setCarrera(“T. S. en A., D. y P. de A.”)

A->setNombre(“Andres”)

A->setEdad(25)

Reutilización !

Acceso a propiedades

Público

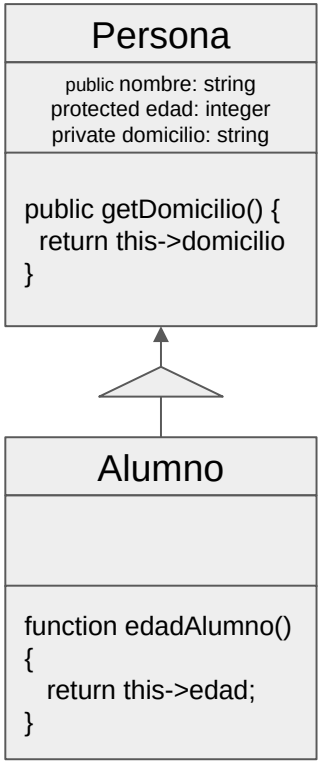
Cualquiera puede accederlo

Protegido

La clase y subclase directa

Privado

Solo la clase



Alumno A

mostrar A->nombre

mostrar A->edad → Falla!!

mostrar A->edadAlumno()

mostrar A->domicilio → Falla !

mostrar A->getDomicilio()

Acceso a métodos

Público

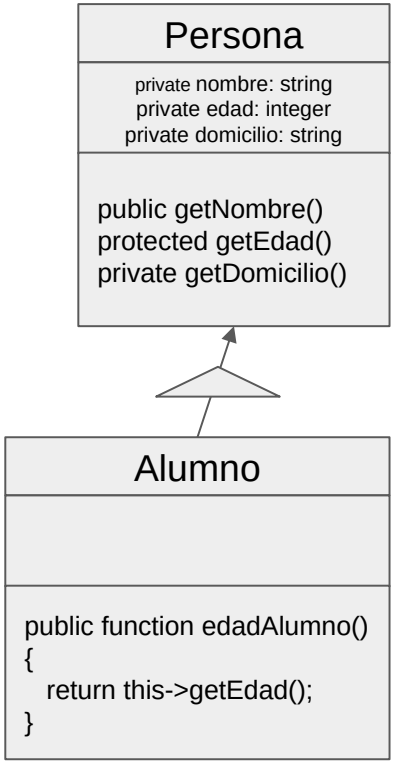
Cualquiera puede accederlo

Protegido

La clase y subclase directa

Privado

Solo la clase



Alumno A

mostrar A->nombre → Falla !!

mostrar A->getNombre()

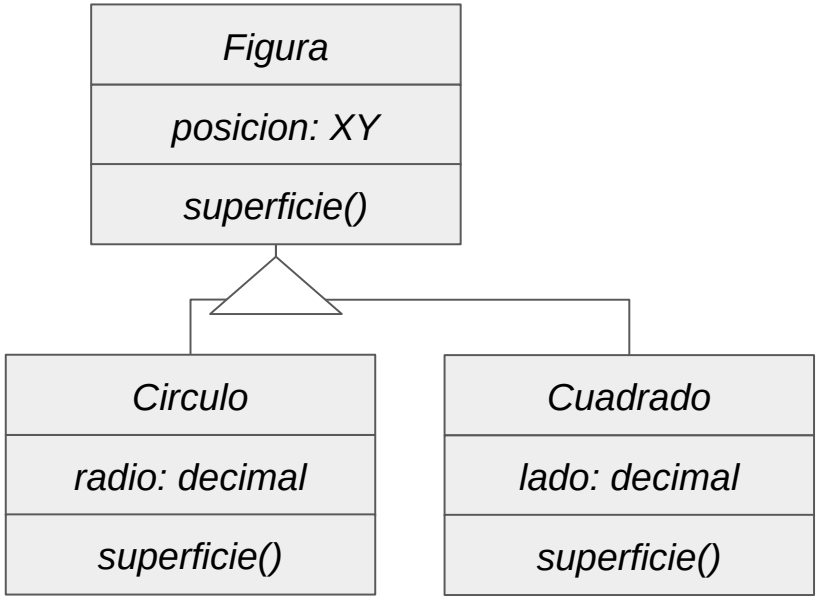
mostrar A->getEdad() → Falla

mostrar A->edadAlumno()

mostrar A->getDomicilio() → Falla!

Polimorfismo

Responder distinto al mismo mensaje




```
function Cuadrado::superficie() {
        return this->lado * this->lado
}
```

```
function Circulo::superficie() {
        return this->radio * this->radio *
PI;
}
```

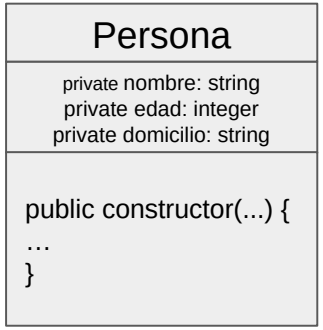
Creación de objetos

Según el lenguaje existen distintas formas. Pero tienen el mismo principio:

Iniciar las propiedades internas

A = *new* Persona()

A = new Persona(“Julio”, 25, “San Martin 100”)




```
function Persona::Constructor(nombreOtro, edadOtra, domicilioOtro) {
        this->nombre = nombreOtro;
        this->edad = edadOtra;
        this->domicilio = domicilioOtro;
}
```

Destrucción de objetos

Libera la memoria para un objeto. Puede realizar acciones necesarias.

Tiene sentido para estructuras de conjuntos de objetos

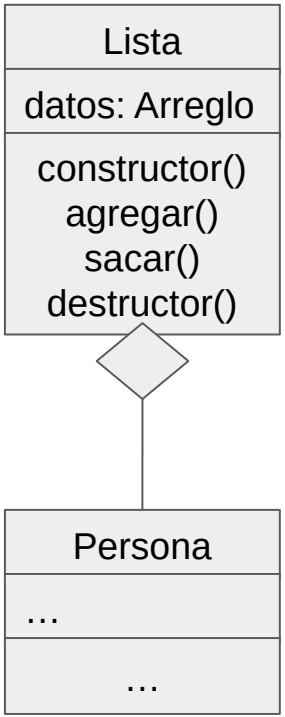
P = *new* Persona(“Pedro”, 20, “Belgrano 10”)

…

P->delete()

```
function Persona::Destructor() {
        mostrar “Adios “ + this->getNombre()
}
```

Destrucción de objetos




```
function Lista::destructor() {
        Para i = 1 Hasta this->datos->longitud {
                Persona P = this->datos[ i ];
                P->destructor();
        }
}
```

Resumen

Vimos un objeto como

- Una estructura de datos, “similar” a un arreglo
- Con sus propias funciones de acceso
- Mecanismo de ocultamiento
- Su propio creador y destructor

Próxima Clase

- Igualdad e identidad.
- Lenguajes de POO.
- Introducción a PHP.