

PROGRAMACIÓN ORIENTADA A OBJETOS

Elaborado por: Juan Miguel Guanira Erazo

PUCP

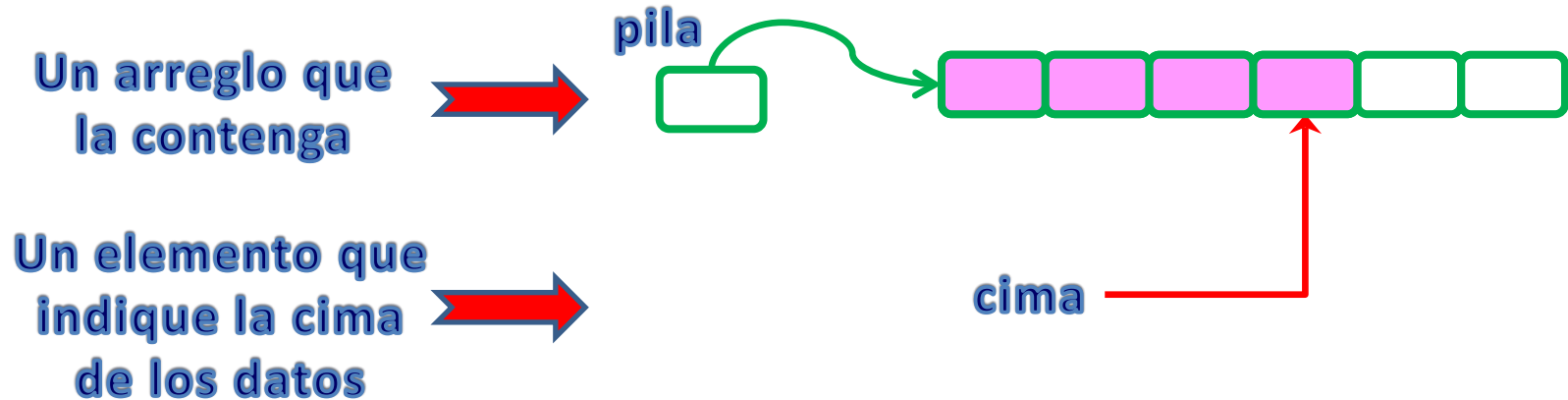
INTRODUCCIÓN

La "Programación orientada a objetos" es un paradigma de programación que aparece a principios de los años 1980.

Buscaba corregir ciertas deficiencias de la programación tradicional.

En la programación tradicional, los datos de un programa y la funcionalidad del mismo son tratados por separado. La POO junta los datos y la funcionalidad en un solo elemento.

Por ejemplo si queremos implementar un programa que maneje una pila de datos, la programación tradicional por un lado define la estructura de datos que la contendrá y los elementos que ayudarán a manipularla:



Y por otro lado se maneja la funcionalidad del programa o las funciones que manejarán la pila:

Una función “push” para apilar un dato

Una función “pop” para desapilar un dato

Una función para verificar la pila está vacía

Una función para verificar si se llenó la pila

Etc.

Finalmente los dato y la funcionalidad del programa se juntan a la hora de llamar a las funciones a través de los parámetros:

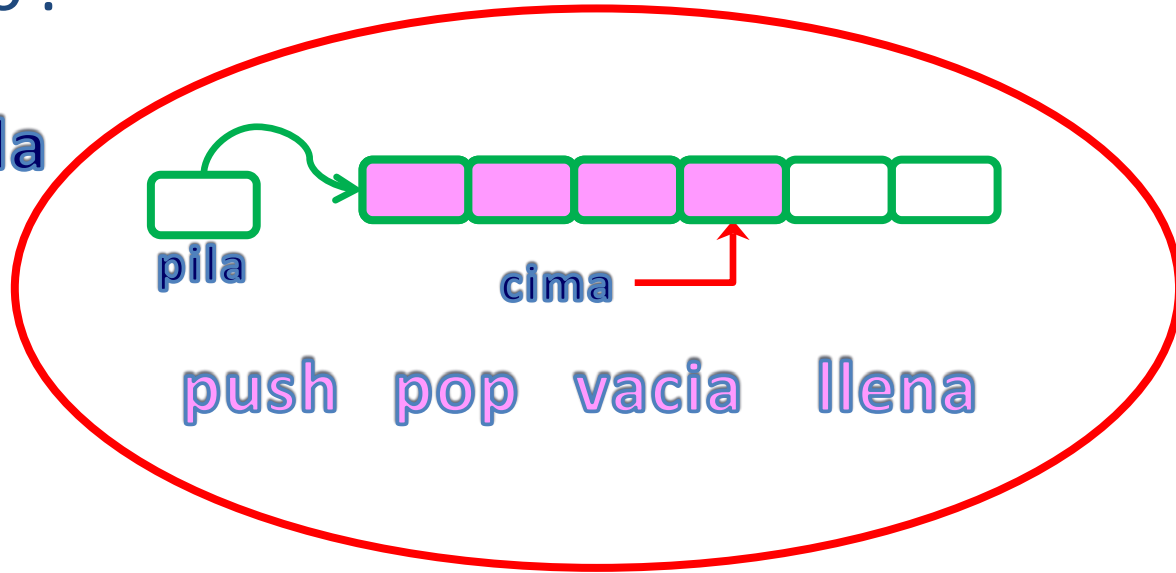
```
push(pila, cima, dato);
```

```
dato = pop(pila, cima);
```

Etc.

En la programación orientada a objetos se busca juntar los datos y la funcionalidad en un solo elemento :

objetoPila



La manipulación de la pila se realizará mediante operaciones de la siguiente forma:

```
objetoPila.push(dato);
```

```
dato = objetoPila.pop();
```

Etc.

A estas operaciones se les denomina “mensajes”.

CONCEPTOS

CLASES Y OBJETO:

A pesar que muchos los confunden, se tratan de dos elementos muy diferentes. En términos simples podemos decir que:

Las clases son el equivalente a los tipos de datos, mientras que los objetos son las variables en la programación tradicional.

ATRIBUTOS O DATOS MIEMBRO:

Son los elementos que definen el estado de un objeto.

Por ejemplo en el objeto pila sus atributos son el arreglo que contiene los datos y el elemento que marca la cima .

MÉTODOS O FUNCIONES MIEMBRO:

Son los elementos que definen la funcionalidad del objeto.

Por ejemplo en el objeto pila sus método son push, pop, etc.

MENSAJE:

Es la orden que se le da a un objeto para que realice una acción o cambie de estado.

Por ejemplo un mensaje sería:

```
objetoPila.push(dato);
```

ENCAPSULAMIENTO Y OCULTACIÓN:

El encapsulamiento se refiere a que todos los elementos que la conforman están inmersos en una “capsula” u “objeto”.

El encapsulamiento permite ocultar los atributos de una clase.

POR QUÉ OCULTAR LOS ATRIBUTOS:

Pensemos en la definición una clase para manipular una pila en la que el diseñador ha pensado que la mejor opción es la de definir un arreglo como contenedor de la pila.

Esta clase podría ser utilizada en muchas aplicaciones y proyectos.

POR QUÉ OCULTAR LOS ATRIBUTOS:

Qué pasaría si el diseño permite manipular los atributos directamente, esto es:

```
objetoPila.pila[cima] = dato;  
objetoPila.cima++;
```

Es muy probable que todos los proyectos funcionen correctamente.

POR QUÉ OCULTAR LOS ATRIBUTOS:

Pero, qué pasaría si luego de un tiempo, el diseñador se diera cuenta que quizá es mejor que el contenedor del pila fuera una lista ligada y prueba que esto es así.

Pues estaría muy tentado a elaborar una segunda versión de la clase pila, pero ahora con una lista ligada.

POR QUÉ OCULTAR LOS ATRIBUTOS:

Pero, qué pasaría con todas aquellas aplicaciones que utilizaron los atributos directamente:

```
objetoPila.pila[cima] = dato;
```

Pues, ninguna de ellas podría beneficiarse del nuevo diseño porque el manejo de un arreglo es muy diferente al de una lista. Tendrían que conformarse y seguir usando la versión obsoleta, o modificar su aplicación para que se adapte a esta nueva versión.

POR QUÉ OCULTAR LOS ATRIBUTOS:

Sin embargo, qué hubiera pasado si desde el principio el diseñador impedía el uso directo de los atributos, esto es no se puede usar:

```
objetoPila.pila[cima] = dato;
```

Sino que si se quiere colocar un dato en la pila, obligatoriamente se debería hacer mediante los métodos, esto es `objetoPila.push(dato);`

POR QUÉ OCULTAR LOS ATRIBUTOS:

Pues la forma cómo se coloca un dato en la pila sería totalmente transparente para el usuario de esta clase.

Al usuario no le debe interesar cómo se hace la operación, sino que la operación haga lo que él necesita y de la mejor manera posible.

POR QUÉ OCULTAR LOS ATRIBUTOS:

Si se sigue esta forma de trabajo, actualizar la clase por otra mejor no traería ninguna complicación para las aplicaciones que usaban la clase original ya que seguirían usando el mismo método y de la misma forma, porque es el método el que manipula los atributos, el usuario no.

“Por esta razón siempre debemos ocultar los atributos de una clase”.

HERENCIA:

Es una propiedad de la POO por medio de la cual se puede crear una clase a partir de otra.

Esto quiere decir que una clase puede adquirir todos los elementos de otra clase, sin tener que volverlos a escribir, incluso sin tener el código fuente.

También se puede modificar el comportamiento de algunos métodos y agregar otros a la nueva clase.

POLIMORFISMO:

Es una propiedad que permite que una misma acción aplicada a diferentes objetos produzca comportamientos diferentes.

Por ejemplo la operación de presionar dos veces el botón izquierdo de un mouse.

POLIMORFISMO:

Si la acción la hacemos sobre un ícono en el escritorio del computador el resultados será muy diferente al de hacerlo sobre una palabra escrita en el MS Word.

El primer caso se abrirá una ventana, en el segundo se marcará toda la palabra.

Y si lo hacemos en el margen izquierdo en el MS Word se marcará todo el párrafo.

INSTANCIA e INSTANCIAR:

El término “Instancia” o “instancia de una clase” se utiliza frecuentemente para denominar también a los objetos.

En ese sentido el término “Instanciar” se utiliza para indicar que se está creando un objeto.

INSTANCIA e INSTANCIAR:

Si se tiene una clase denominada “ClaseA” y se quiere definir un objetos de esa clase, entonces al escribir:

`ClaseA objetoA;`

Diremos que estamos instanciando la clase “ClaseA” y que “objetoA” es una instancia de esa clase.

INSTANCIA e INSTANCIAR:

Pero si escribimos lo siguiente:

```
ClaseA *ptrObj;
```

No podemos decir que se ha instanciado la clase porque `ptrObj` no es un objeto sino un puntero.

Se instanciará cuando hagamos lo siguiente:

```
ptrObj = new ClaseA;
```

La variable referenciada será la instancia de la clase.

IMPLEMENTACIÓN Y USO DE UNA CLASE

EJEMPLO:

Vamos a implementar una clase para manipular un “Rectangulo”. El rectángulo estará definido por su base y altura y permitirá determinar su área y perímetro.

Primero vamos a hacerlo de manera “manual” es decir que vamos a escribir todo el código. Luego veremos algunas herramientas propias del NetBeans para hacer un poco más fácil el proceso.



Pasemos al programa

EL PUNTERO this

```
class CRectangulo{  
    private:  
        double base;  
        double altura;  
    public:  
        void SetBase(double);  
        ...  
};
```

```
void SetBase(double b){  
    base = b;  
};
```


ClaseRectangulo

base altura

SetBase area

SetAltura

...

ClaseRectangulo R1,
R2,R3,...;

Segmento de Código

Segmento de Datos

R1



base altura SetBase SetAltura ...

ClaseRectangulo

base altura

SetBase area

SetAltura

...

ClaseRectangulo R1,
R2,R3,...;

R1.SetBase(3);

R2.SetBase(5);

Segmento de Código

```
void SetBase (double b){  
    base = b;  
}
```

```
void SetBase (double b){  
    base = b;  
}
```

Segmento de Datos

R2

R1



base altura

base altura

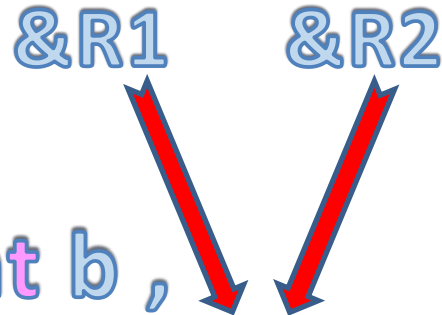
La solución a este problema la da el
puntero “this”

```
R1.SetBase(3);
```

```
R1.SetBase(3, &R1);
```

```
void CRectangulo::SetBase(int b){  
    base = b;  
}
```

```
void CRectangulo::SetBase(int b ,  
                           CRectangulo *this){  
    this -> base = b;  
}
```



The diagram illustrates the concept of a pointer to 'this' in a C++ member function. Two labels, &R1 and &R2, are positioned above the second function signature. Two red arrows point from these labels to the *this parameter in the function signature, indicating that these variables represent the address of the object calling the function.

Definición:

“this” es un puntero que se crea automáticamente en toda función miembro de una clase. Este puntero recibe la dirección del objeto que invoca al método.



Regresemos al
programa

CONSTRUCTORES Y DESTRUCTORES

CONSTRUCTOR

Definición:

Un “Constructor” es un método especial de una clase que permite inicializar los atributos del objeto instanciado.

CONSTRUCTOR

Características:

1. Se ejecuta automáticamente cuando se instancia la clase.
2. Debe tener el mismo nombre que la clase.
3. Puede tener argumentos pero no puede devolver valores.
4. Se puede sobrecargar.

CONSTRUCTOR

Tipos:

1. **Constructor por defecto** es aquel que no posee argumentos.
2. **Constructor** propiamente dicho es aquel que posee argumentos.
3. **Constructor copia** es aquel que inicializa al objeto a través de otro objeto.

DESTRUCTOR

Definición:

Un “Destructor” es otro método especial de una clase. Permite eliminar todo el desecho que queda en memoria cuando ya no se requiere al objeto.

DESTRUCTOR

Características:

1. Se ejecuta automáticamente cuando se sale del ámbito del objeto.
2. Debe tener el mismo nombre que la clase pero anteponiéndole la tilde '˜'.
3. No tiene argumentos ni puede devolver valores.
4. No se puede sobrecargar. Es único.



Pasemos al programa