PARADIGMAS DE PROGRAMACIÓN
Licenciatura en Informática
Programador Universitario

# UNIDAD III
# PARADIGMAS DE PROGRAMACIÓN
## Ejemplo Pila(item) en C++

## ADT Pila(item)

**Sintaxis**

```
PV: → Pila
Push: Pila x item → Pila

Top: Pila → item U {indefinido}

Pop: Pila → Pila

esPilaVacia: Pila → Bool
Pertenece: Pila x item → Bool
sonIguales: Pila x Pila → Bool
```

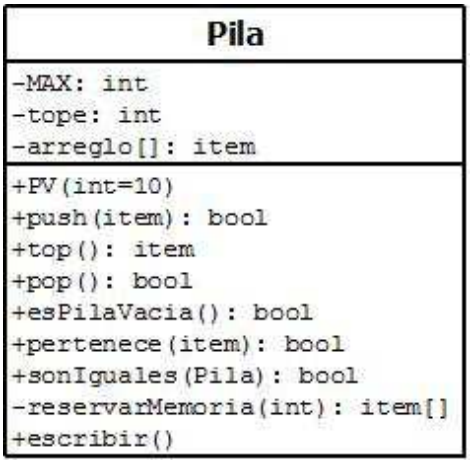
# ADT Pila(item)

**Semántica** **P, Q ∈ Pila, x, k ∈ item**

```
Top(PV) ≡ indefinido
Top(Push(P, x)) ≡ x

Pop(PV) ≡ PV
Pop(Push(P, x)) ≡ P

esPilaVacia(PV) ≡ true
esPilaVacia(Push(P,x)) ≡ false

Pertenece(PV, k) ≡ false
Pertenece(Push(P, x), k) ≡ x==k OR Pertenece(P, k)

sonIguales(PV, PV) ≡ true
sonIguales(Push(P, x), PV) ≡ false
sonIguales(PV, Push(Q, k)) ≡ false
sonIguales(Push(P, x), Push(Q, k)) ≡ x==k AND
                                    sonIguales(P,Q)
```

3

# Clase Pila en UML




4

# Clase Pila en C++

**Pila.h**

```
class Pila{
    int tope;
    item *arreglo;
    int MAX;
    item* reservarMemoria(int tama);
public:
    void PV(int =10);
    bool push(int item);
    int  top();
    bool pop();
    bool esPilavacia();
    bool pertenece(item k);
    bool sonIguales(Pila &Q);
    void escribir();
};
```

- Palabra reservada del Lenguaje
- Inicio de la declaración de los miembros de la clase
- Datos Miembros
- Miembros Privados (default). Sólo se puede acceder a ellos por medio de los métodos de la clase
- Declaración de Funciones Miembros. Sus implementaciones se proporcionan después
- Miembros Públicos. Forman la interfaz de la clase Pila
- Fin de la clase

5

# Programa Principal

OBJETOS IMPLÍCITOS

```
#include <iostream>
#include "Pila.h"

using namespace std;

int main() {
    Pila P, Q;
    P.PV(5);
    Q.PV(8);
    // ...
    Q.pop();

    cout<<"Tope de P: "<<P.top()<< endl;

    if(P.sonIguales(Q))
        cout<<"Las pilas son iguales"<<endl;
    else
        cout<<"Las pilas son distintas"<<endl;
    return 0;
}
```

6

# Implementación de los métodos

```
class Pila{
    int tope;
    item *arreglo;
    int MAX;
    item* reservarMemoria(int tama);
public:
    void PV(int dim = 10){
        MAX = dim >0 ? dim : 10;
        tope=-1;
        arreglo = reservarMemoria(MAX);
    }
    ...
};
```

Implementación **dentro del alcance de la clase**

**Operador de resolución de alcance**

Implementación **fuera del alcance de la clase**

```
void Pila::PV(int dim)
{
    MAX = dim >0 ? dim : 10;
    tope=-1;
    arreglo = reservarMemoria(MAX);
}
```

7

# Implementación de los métodos

```
bool Pila ::push(int item){
    bool resultado=  false;
    if(tope+1 <MAX){
        tope++;
        arreglo[tope] = item;
        resultado =true;
    }
    return resultado;
}
```

**Pila.cpp**

```
bool Pila:: esPilavacia(){
    return tope==-1;
}
```

```
int Pila:: top(){
    if(!esPilavacia())
        return arreglo[tope];
    else
        return indefinido;
}
```

8

# Implementación de los métodos

**Pila.cpp**

```
bool Pila::sonIguales(Pila &Q){
    bool resultado=true;
    if (tope!=Q.tope)
        resultado=false;
    else{
        int i=0;
        while(i<=tope && resultado){
            if(arreglo[i]!=Q.arreglo[i])
                resultado=false;
            i++;
        }
    }
    return resultado;
}
```

```
bool Pila::pertenece(item k){
    int i=tope;
    while(i>=0 && arreglo[i]!=k)
        i--;
    return i>=0;

}
```

# Implementación de los métodos

**Pila.cpp**

```
bool Pila:: pop(){
    bool resultado=false;
    if(tope>=0){
        tope--;
        resultado = true;
    }
    return resultado;
}
```

```
item * Pila:: reservarMemoria(int dim){
    item *reserva = new item[dim];
    if(reserva==NULL){
        cout<<"Problema: no se pudo realizar la reserva";
    }
    return reserva;
}
```

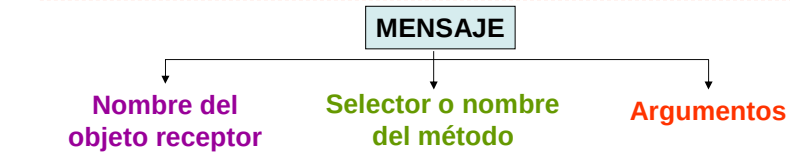
10

# Implementación de Métodos EXTERNOS

```
void cargarPila(Pila &P){
    int cant;
    item elemento;
    cout << "Ingrese la cantidad de items: " <<endl;
    cin >> cant;
    if(cant <=0)
        cant=10;
    cout << "Ingrese los items de la Pila: " <<endl;
    while(cant>0){
        cin >> elemento;
        P.push(elemento);
        cant--;
    }
}
```

# Métodos y Mensajes

Los métodos se invocan enviando mensajes a los objetos, cuya ejecución representa el conjunto de operaciones a realizar para la resolución del problema.




Ejemplo en C++

```
// Agrega un elemento al objeto Pila P
P.push(elemento);
```

Mensaje

Objeto receptor del mensaje

Selector

Argumento del método

12

# Programa Principal

```
#include <iostream>
#include "Pila.h"

using namespace std;

void cargarPila(Pila &P);

int main() {
    Pila P, Q;
    P.PV(5);
    Q.PV(8);

    cargarPila(P);
    cargarPila(Q);

    Q.pop();

    cout<<"Tope de P: "<<P.top()<< endl;

    if(P.sonIguales(Q))
        cout<<"Las pilas son iguales"<<endl;
    else
        cout<<"Las pilas son distintas"<<endl;
    return 0;
}
```

MENSAJES

13

# Constructores y Destructor

```
class Pila{

    ...
 public:

    Pila(int dim=10);
    Pila(const Pila &);
    ...
    ~Pila();
};
```

**CONTRUCTOR**

Método especial de una clase que se invoca automáticamente cada vez que se crea un objeto de dicha clase

**DESTRUCTOR**

Método especial de una clase que se invoca automáticamente cada vez que el objeto sale de alcance

14