


ALGORITMOS Y PROGRAMACIÓN

Clase 3

Abstracción de datos (tipos de datos abstractos)

Temario

- Abstracción y encapsulamiento.
 - Tipos de datos standard
 - Tipos definidos por el usuario
 - Abstracción de datos:
 - Concepto
 - Aplicación: Tipos abstractos de Datos (TAD)
- 

Repasando: Tipo de dato

- El *tipo de dato* de una variable define el conjunto de valores posibles que se pueden almacenar en la misma y las operaciones que pueden aplicarse sobre dichos datos.
- También definen las operaciones entre diferentes tipos de datos y la conversión entre uno y otro.
- Todos los lenguajes de programación proveen un conjunto mínimo de datos predefinidos, llamados *tipos de datos estandar*.

Tipo de dato standard o predefinidos

Tipo de dato	Dominio	Operaciones
byte	0..255	suma, resta, multiplicación, división, resto
int	-2.147.483.648...2.147.483.647	
float	1.5×10^{-45} a 3.4×10^{38} con precisión de 7 dígitos	
bool	true, false	and, or, not
char	Letras, dígitos, símbolos, caracteres especiales	Concatenación para formar un string
string	Secuencia de caracteres	Concatenación, indexación

Tipo de dato standard o predefinidos

- El programador *no conoce cómo están almacenados los datos ni cómo están implementadas las operaciones* que se les pueden aplicar.
- La estructura de datos subyacente y la implementación de las operaciones están ocultas al programador. Ambas cosas son transparentes al usuario.
 - Los usa directamente a partir de saber para qué sirven y cuáles son las operaciones permitidas.
 - No puede modificar su estructura interna ni el código de implementación de cada operación.

Tipo de dato definido por el usuario

- Los *tipos definidos por el usuario* permiten modelar conceptos específicos del dominio del problema.
- El programador es quien los diseña:
 - *eligiendo la estructura de datos o representación interna* más adecuada e
 - *implementando las operaciones* que se pueden aplicar sobre los mismos.
- Los constructores *struct, class, interface y enum* en C# son los que se utilizan para crear tipos definidos por el usuario, o sea, tipos de datos personalizados.

Tipo de dato definido por el usuario

- Ejemplo: Lista de apellidos
 - Estructura interna: se utiliza un ArrayList de strings
 - Operaciones: agregar un apellido, borrar un apellido, busca un apellido, etc..
-
- La estructura interna y la implementación de las operaciones son visibles.
 - Si el programador desea modificar su estructura interna, puede hacerlo, pero debe modificar también la implementación de las operaciones de manipulación.

Abstracción de datos

Combinando las características de los tipos estándar y los definidos por el usuario nacen los Tipos Abstractos de Datos (TAD) :

Si diseñamos y definimos *nuevos tipos de datos* pero *ocultando tanto su representación interna* como *la implementación de sus operaciones* mediante un mecanismo de *encapsulamiento*, obtenemos un

TIPO ABSTRACTO DE DATOS.

Tipo abstracto de datos

- Un tipo abstracto de datos (TAD), especifica:
 - Cuál es el conjunto de valores que pueden ser almacenados en una variable de dicho tipo
 - Cuáles son las operaciones que pueden aplicarse sobre dichos valores
- **Pero no detalla cuál es su estructura interna (cómo se almacenan dichos datos en la memoria) ni la implementación de las operaciones, es decir ambas cosas quedan ocultas al usuario.**

Tipo abstracto de datos

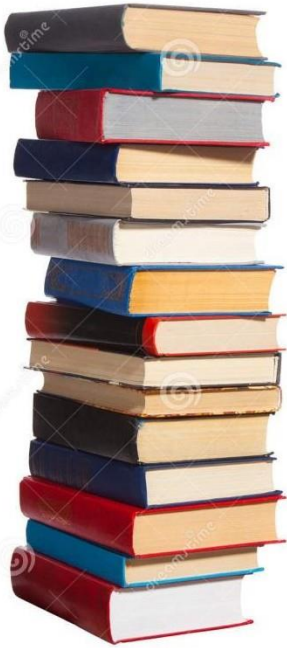
- La abstracción de datos consiste en enfocarse solo en las características esenciales de un objeto o entidad y en las operaciones que pueden aplicarse sobre dicha entidad (su funcionalidad), independientemente de cómo se lo implemente.
- Separa el diseño de la implementación. Por esta razón todo TAD tiene una *interfaz pública y una privada*.

Tipo abstracto de datos

- La **interfaz privada** de un TAD define la vista interna, solo visible por el diseñador, que es quién desarrolla e implementa el nuevo TAD.
- En cambio la **interfaz pública** define el conjunto de operaciones permitidas y detalla para que sirve cada operación. Es la parte visible del TAD: lo que conoce el usuario.
- **Sólo se puede consultar o modificar los datos almacenados en un TAD con las operaciones definidas en su interfaz pública.**

Ejemplo TAD Pila

- ¿Qué es una Pila?

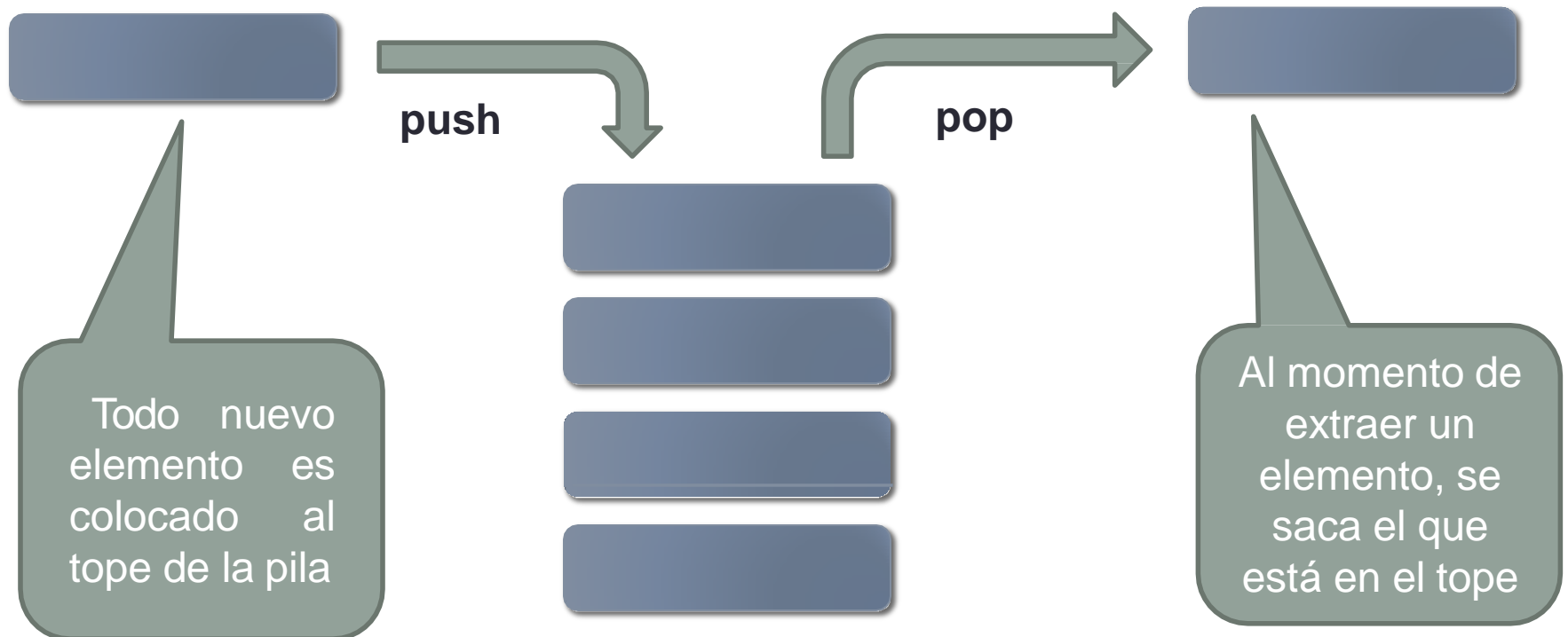


TAD Pila: para qué sirve

- Una pila es una colección que contiene múltiples elementos.
- Presenta una política de manipulación de sus elementos de tipo LIFO (“last in, first out” en inglés), “el último en entrar es el primero en salir”, porque el elemento añadido en último lugar es el primero que se extrae y se procesa.

TAD Pila: qué operaciones admiten

- Las operaciones básicas que se pueden realizar con una pila son apilar (**push**) y desapilar (**pop**).



TAD Pila: interfaz pública

- Las operaciones apilar (push) y desapilar (pop) constituyen las operaciones mínimas necesarias para manipular una pila.
 - new: **crea** una pila vacía en memoria
 - apilar(elem): **agrega** a elem en el tope de la pila
 - desapilar(): **saca** y **retorna** el elemento del tope de la pila
- Otras operaciones:
 - tope(): **devuelve** el elemento en el tope de la pila **sin sacarlo** de la pila.
 - estaVacía(): devuelve **true** si la pila no tiene elementos

TAD Pila. Ejemplo de uso

- ¿Cómo se usa? A través de las operaciones elementales detalladas en su interfaz pública.

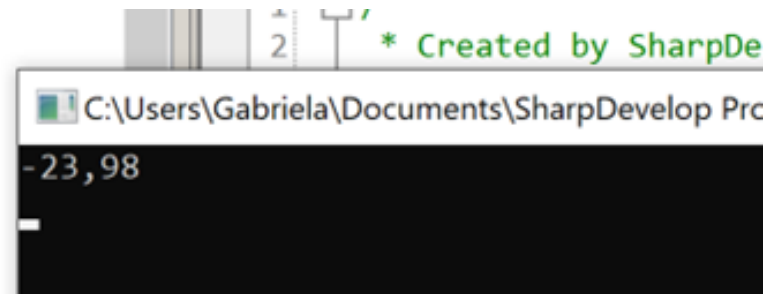
```
static public void Main(){  
    Pila p = new Pila(); /* crea la pila en memoria */  
    p.apilar(4);          /* agrega 1 elemento en la pila */  
    p.apilar("Hola");  
    p.apilar(-23.98);  
    Console.WriteLine(p.desapilar()); /* desapila el elemento  
tope y lo retorna */  
    Console.ReadKey(true); }  
}
```

Usamos las
operaciones sin
saber cómo
están
implementadas

TAD Pila.

Ejemplo de uso

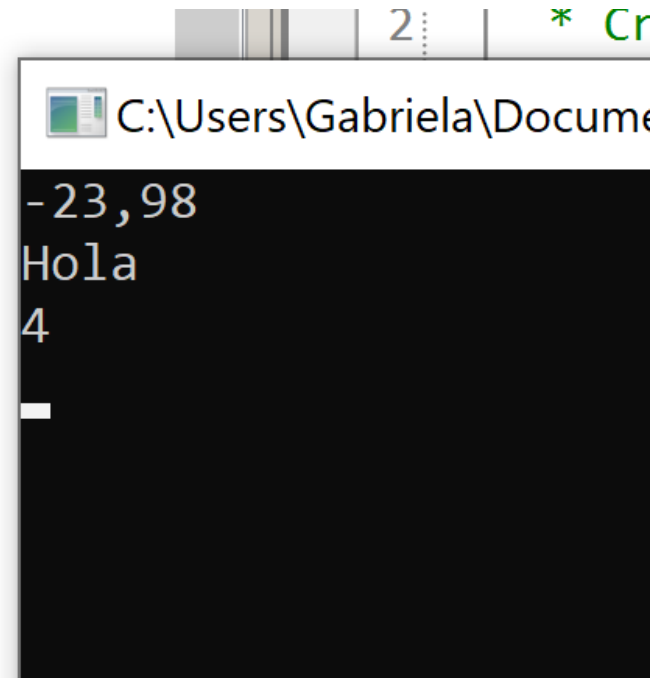
```
static public void Main(){  
    Pila p = new Pila(); /* crea la pila en memoria*/  
    p.apilar(4);          /*agrega 1 elemento en el tope */  
    p.apilar("Hola");  
    p.apilar(-23.98);  
    Console.WriteLine(p.desapilar()); /*desapila el elemento  
tope y lo retorna*/  
    Console.ReadKey(true);
```



TAD Pila. Ejemplo de uso

- Para procesar todos los elementos de una Pila

```
static public void Main(){  
    Pila p= new Pila();  
    p.apilar(4);  
    p.apilar("Hola");  
    p.apilar(-23.98);  
  
    while(! p.estaVacia())  
        Console.WriteLine(p.desapilar());  
    Console.ReadKey(true);  
}
```

A screenshot of a Windows console window. The title bar shows the file path "C:\Users\Gabriela\Documents\2...". The console output displays the elements of the stack being popped: "-23,98", "Hola", and "4", each on a new line. A white cursor is visible on the line following the last output.

```
C:\Users\Gabriela\Documents\2...  
-23,98  
Hola  
4  
_
```

TAD Pila. Ejemplo de uso

- En ambos ejemplos pudimos utilizar el TAD sin tener idea de su estructura interna ni cómo están implementadas las operaciones básicas de manejo de la pila.
- Pudimos abstraer el qué del cómo: usamos el TAD pila sabiendo para qué sirve, no cómo lo lleva a cabo.

TAD PILA

Para qué se puede usar el TAD Pila:

- Invertir colecciones
- Determinar palíndromos
- Evaluación de operaciones aritméticas postfija
- Ejecución de soluciones recursivas.
- Historial de navegación en los browser de internet.
- Manejo de la operación "deshacer".
- Simular situaciones de la vida real donde se utilizan pilas de elementos.

Puesta en común

Escriba una función que reciba una pila de elementos enteros y dos valores: ***valor_a_buscar*** y ***valor_a_reemplazar*** y que devuelva una pila donde, toda ocurrencia de ***valor_a_buscar*** sea reemplazada por ***valor_a_reemplazar***. La pila devuelta debe volver con sus valores en el mismo orden en que estaban.

NOTA: Use la [interfaz pública](#) dada previamente

```
static Pila reemplazo(Pila pilaOr, int buscar, int reemplazar)
{
    Pila temporal = new Pila();
    while(!pilaOr.estaVacia()) {
        int v = (int)(pilaOr.desapilar());
        if(v == buscar)
            temporal.apilar(reemplazar)
        else
            temporal.apilar(v);
    }
```

→ **Recorremos la pila recibida por parámetro.**

→ **Sacamos un elemento de la pila.**

Si el valor sacado es igual al buscado, ponemos en la pila el valor de reemplazo.

/* La pila temporal tiene todos los valores invertidos. */

```
    Pila nueva = new Pila();
    while(!temporal.estaVacia()) {
        int elem= (int)(temporal.desapilar());
        nueva.apilar(elem);
    }

    return nueva;
}
```

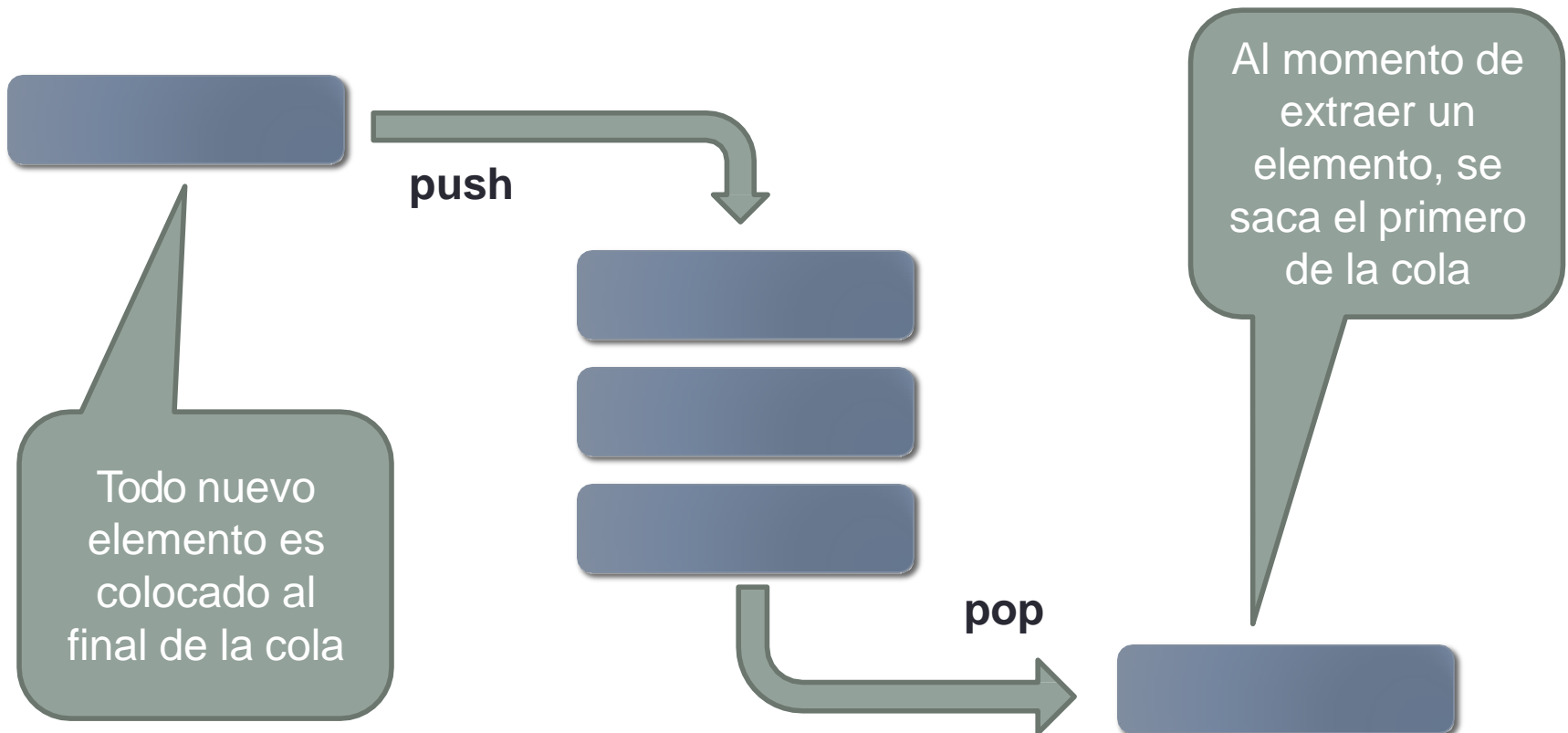
Usamos una tercer pila para "dar vuelta" los valores que están en temporal y la retornamos.

TAD Cola: para qué sirve

- Una cola es una colección que contiene múltiples elementos.
- Presenta una política de manipulación de sus elementos del tipo FIFO (“first in, first out” en inglés), “primero en entrar primero en salir”, porque el elemento añadido en primer lugar es el primero que se extrae y se procesa.

TAD Cola

- Las operaciones básicas que se pueden realizar con una cola son encolar (**push** o **enqueue**) y desencolar (**pop** o **dequeue**).



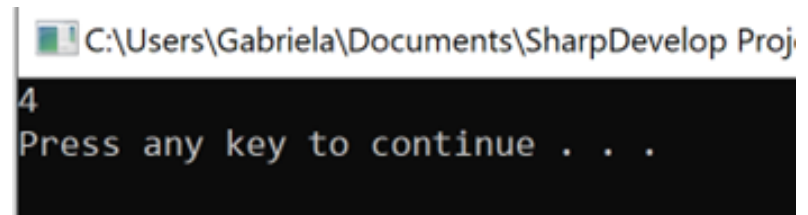
TAD Cola:interfaz de usuario

- Las operaciones encolar (push) y desencolar (pop) constituyen las operaciones mínimas necesarias para manipular una cola.
 - new: **crea** una cola vacía sin elementos
 - encolar(elem): **agrega** a elem en el tope de la cola (al final)
 - desencolar(): **elimina y retorna** el primer elemento de la cola
- Otras operaciones:
 - tope (): **devuelve** el elemento que está en el principio de la cola **sin sacarlo**.
 - estaVacía(): devuelve **true** si la cola no tiene elementos

TAD Cola

- ¿Cómo se usa?

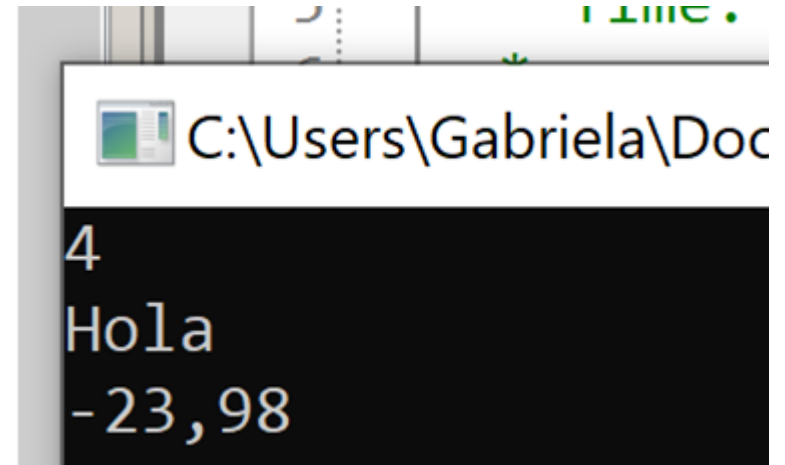
```
static public void Main(){  
    Cola c= new Cola(); /*crea una cola vacía*/  
    c.encolar(4); /* agrega un elemento al final*/  
    c.encolar("Hola");  
    c.encolar(-23.98);  
    Console.WriteLine(c.desencolar()); /*saca el 1er  
    elemento y lo retorna*/  
    Console.ReadKey(true);  
}
```



TAD Cola

- Para procesar todos los elementos de una Cola

```
static public void Main(){  
    Cola c = new Cola();  
    c.encolar(4);  
    c.encolar("Hola");  
    c.encolar(-23.98);  
  
    while(! c.estaVacia())  
        Console.WriteLine(c.desencolar());  
    Console.ReadKey(true);  
}
```



TAD COLA

Cuáles pueden ser los usos del TAD Cola:

- Representar colas de personas de la vida real
 - Banco
 - Correo
 - Oficina de atención
- En informática
 - Cola de impresión
 - Cola de procesos

Ventajas del uso de TADs

- **Abstracción:** Los usuarios de un TAD no necesitan conocer los detalles de su implementación. Solo saben para qué sirve y qué hace cada operación.
- **Reutilización:** un TAD puede ser utilizado en distintos desarrollos de aplicaciones.
- **Corrección:** el código de un TAD, probado y testeado, disminuye la presencia de errores en un desarrollo.

Implementación de TADs

- Para implementar un TAD el lenguaje de programación debe proveer dos mecanismos:
 - **Encapsulamiento:** a través del cual se codifica la estructura interna junto con la implementación de las operaciones
 - **Ocultamiento:** a través del cual quedan inaccesibles e invisibles la estructura interna y la implementación de las operaciones.

No todos los lenguajes de programación proveen ambos mecanismos.

Implementación de TADs

- En C# los TADs estarán implementados cada uno en una **clase**.
- La abstracción es la base de la programación orientada a objetos (POO).
- Cuando se modela en POO, se crean **clases**, a través de las cuales se representan **objetos** del mundo real en base a sus **atributos o características** y a su **funcionalidad**.
- La clase **encapsula y oculta** la estructura interna de los objetos y la implementación de las operaciones que se les pueden aplicar.

Abstracción en POO

- Un programa en el POO es un conjunto de objetos que se envían mensajes entre sí solicitándose la ejecución de una tarea o función específica.
- El programador desarrolla el programa conociendo las características de los objetos y sus funcionalidades, pero sin saber cómo están almacenados los datos ni cómo están implementadas sus operaciones.

En POO la abstracción es el mecanismo que nos permite utilizar un objeto SOLO sabiendo lo que hace, SIN importar como lo hace.

