Introducción a la Programación Orientada a Objetos

Unidad: Programación Orientada a Objetos (POO)

Introducción a la Programación

De lo estructurado a lo orientado a objetos

Problema: demasiados datos sueltos

- Hasta ahora, para manejar información de personas usamos:
  - Varios arreglos paralelos, por ejemplo:
  - `nombres[i]`, `telefonos[i]`, `emails[i]`
- Este enfoque funciona en problemas pequeños, pero:
  - Es fácil confundir índices o mezclar datos.
  - Las funciones reciben muchos parámetros relacionados.
  - Agregar un nuevo dato (por ejemplo, `etiqueta` o `direccion`) implica modificar varias partes del programa.
- Idea clave para esta unidad:
  - En lugar de pensar en "arreglos sueltos de datos", vamos a pensar en **entidades completas**: un Contacto.
  - A cada entidad la representaremos con una **clase** y crearemos **objetos**.

Limitaciones del enfoque previo

Arreglos + funciones por separado

- Cuando trabajamos con varios arreglos paralelos:
  - La información de una persona está "repartida" en distintas estructuras.
  - Es fácil cometer errores de índice (`i`) y desordenar los datos.
- Las funciones crecen en cantidad de parámetros:
  - `imprimirContacto(nombres, telefonos, emails, i)`
  - Se vuelve difícil leer, entender y mantener el código.
- Agregar nuevos datos complica el programa:
  - Hay que modificar la definición de los arreglos.
  - Hay que ajustar todas las funciones que usan esos datos.
- Pregunta guía:
  - *¿No sería mejor agrupar todos estos datos y operaciones en una sola unidad lógica?*

Clase vs Objeto

El plano y la casa

- **Clase**
  - Es una plantilla o modelo que describe:
    - Qué datos tendrá un tipo de entidad (atributos).
    - Qué operaciones se pueden realizar sobre ella (métodos).
  - Ejemplo: una clase `Contacto` describe qué información tiene un contacto.
- **Objeto**
  - Es una instancia concreta de una clase.
  - Cada objeto tiene sus propios valores para los atributos.
  - Ejemplo: el contacto *"Ana López"* con su teléfono y correo.
- Analogía:
  - Clase $\rightarrow$ plano de una casa.
  - Objeto $\rightarrow$ una casa construida a partir de ese plano.
- En Java:
  - Definimos la clase una vez.
  - Podemos crear muchos objetos a partir de ella usando `new`.

Estructura básica de una clase en Java

Ejemplo: clase `Contacto`

- En Java, una clase se define con la palabra clave `class`:
  - Agrupa los **atributos** (datos) y los **métodos** (operaciones).
- Para nuestra agenda, usaremos una clase `Contacto` como modelo para representar cada persona en la agenda.

```
class Contacto {
    // Atributos: datos del contacto

    // Metodos: operaciones que podemos hacer con el contacto
}
```

Atributos y métodos de `Contacto`

Datos y acciones en una misma unidad

- **Atributos** (datos) que queremos guardar de cada contacto:
  - `nombre`, `telefono`, `email`, `etiqueta`.
- **Métodos** (acciones) que puede realizar un contacto en nuestro programa:
  - Por ejemplo, mostrar un resumen de su información.
- Cada **objeto** `Contacto` tendrá sus propios valores para estos atributos.

```
class Contacto {
    String nombre;
    String telefono;
    String email;
    String etiqueta;

    void imprimirResumen() {
        System.out.println(nombre + " - " + telefono);
    }
}
```

Encapsulamiento: protegiendo los datos

Por qué no todo debe ser `public`

- Si dejamos todos los atributos como `public`:
  - Cualquier parte del programa puede cambiarlos libremente.
  - No hay control sobre los valores que se asignan.
  - Es más difícil garantizar que los datos sean coherentes.
- Ejemplos de problemas:
  - Teléfonos vacíos o con formato incorrecto.
  - Correos no válidos.
  - Etiquetas nulas o sin sentido.
- Idea clave de la POO:
  - Los datos se **encierran** dentro del objeto.
  - Se accede a ellos a través de **métodos** que pueden validar y controlar cambios.

Ejemplo sin encapsulamiento

Atributos public en Contacto

- En este diseño, cualquier parte del programa puede modificar los atributos directamente.

```
class Contacto {
    public String nombre;
    public String telefono;
    public String email;
    public String etiqueta;
}
class Main {
    public static void main(String[] args) {
        Contacto c = new Contacto();
        c.nombre = "Ana Lopez";
        c.telefono = "";                // telefono vacio
        c.email = "correo-no-valido"; // email incorrecto
        c.etiqueta = null;             // sin etiqueta
    }
}
```

`private`, `public` y acceso controlado

Patrón getter/setter

- Buen diseño en POO:
  - Los **atributos** suelen ser `private`.
  - Los **métodos** que usamos desde afuera suelen ser `public`.
- Para acceder a los datos usamos:
  - **Getters**: devuelven el valor de un atributo.
  - **Setters**: permiten cambiar el valor de forma controlada.
- Ventajas:
  - Podemos validar antes de aceptar un cambio.
  - Podemos cambiar la implementación interna sin modificar el código que usa la clase.

Encapsulamiento en Contacto

Atributos private y métodos de acceso

```
class Contacto {
    private String nombre;
    private String telefono;

    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nuevoNombre) {
        nombre = nuevoNombre;
    }
    public String getTelefono() {
        return telefono;
    }
    public void setTelefono(String nuevoTelefono) {
        // Aqui podriamos validar el formato del telefono
        telefono = nuevoTelefono;
    }
}
```

Constructores: iniciar bien un objeto

Crear contactos con datos coherentes

- Un **constructor** es un método especial que:
  - Se ejecuta al crear un objeto con `new`.
  - Sirve para inicializar los atributos del objeto.
- Características:
  - Tiene el mismo nombre que la clase.
  - No tiene tipo de retorno (ni siquiera `void`).

```
class Contacto {
    private String nombre;
    private String telefono;

    // Constructor
    public Contacto(String n, String t) {
        nombre = n;
        telefono = t;
    }
}
```

Uso de `this` en el constructor

Distinguir atributos de parámetros

- Es común usar los mismos nombres para:
  - Atributos: `nombre`, `telefono`.
  - Parámetros del constructor: `nombre`, `telefono`.
- La palabra clave `this` se refiere al **objeto actual**:
  - `this.nombre` → atributo del objeto.
  - `nombre` → parámetro del método.

```
class Contacto {
    private String nombre;
    private String telefono;

    public Contacto(String nombre, String telefono) {
        this.nombre = nombre;
        this.telefono = telefono;
    }
}
```

Ejemplo: creando contactos en main

Objetos con constructor y método de salida

```
class Contacto {
    private String nombre;
    private String telefono;
    public Contacto(String nombre, String telefono) {
        this.nombre = nombre;
        this.telefono = telefono;
    }
    public void imprimirResumen() {
        System.out.println(nombre + " - " + telefono);
    }
}
class Main {
    public static void main(String[] args) {
        Contacto c1 = new Contacto("Ana Lopez", "7777777");
        Contacto c2 = new Contacto("Carlos Perez", "8888888");
        c1.imprimirResumen();
        c2.imprimirResumen();
    }
}
```

Resumen de la sesión

Introducción a POO con la agenda de contactos

- Problema detectado:
  - Manejar personas con arreglos paralelos complica el código.
  - Difícil de mantener y propenso a errores.
- Nuevas ideas introducidas:
  - **Clase** como modelo (`Contacto`).
  - **Objeto** como instancia concreta.
  - **Atributos** y **métodos** en una misma unidad.
- Buenas prácticas vistas:
  - Encapsulamiento: atributos `private`, métodos `public`.
  - Uso de **getters** y **setters** para acceso controlado.
  - Uso de **constructores** y de `this` para inicializar objetos.
- Para reflexionar antes de la próxima sesión:
  - *¿Cómo podríamos guardar varios contactos usando la clase `Contacto`?*
  - *¿Qué clase podría encargarse de gestionar toda la agenda?*