«Talento Tech»

Back-End

Java

Clase 05

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase 5: Programación Orientada a Objetos (POO)

Índice

Objetivos de la clase.

- Comprender las diferencias entre la programación estructurada y la programación orientada a objetos.
- Entender qué son las clases, objetos y atributos y cómo representan entidades y datos del mundo real en el código.
- Aprender a crear clases y objetos en Java, dando vida a entidades como productos y clientes.
- Distinguir entre métodos de instancia y métodos de clase (estáticos), entendiendo cuándo usar cada tipo.
- Conocer el concepto de constructores y métodos especiales que inicializan los objetos de manera correcta y consistente.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Paradigmas de programación: Estructurada vs. POO.

En la **programación estructurada**, el código se organiza principalmente en funciones y estructuras de control. Los datos suelen ser manipulados de forma más procedural, y la lógica se centra en la secuencia de pasos para resolver un problema. Esto funciona bien en programas pequeños, pero a medida que el proyecto crece, se vuelve difícil mantener la coherencia y el orden.

En la **programación orientada a objetos**, el foco se pone en entidades llamadas objetos, que combinan datos (atributos) y comportamientos (métodos). El programa se modela como una interacción entre estos objetos. Algunas ventajas de la POO:

- **Encapsulamiento:** Cada objeto controla su propio estado interno, exponiendo sólo las operaciones necesarias.
- **Abstracción:** Ocultar la complejidad interna de los objetos, presentando una interfaz sencilla.
- **Herencia:** Posibilidad de crear nuevas clases basadas en otras existentes.
- **Polimorfismo:** Objetos de distintas clases pueden ser tratados de la misma manera si comparten una interfaz común.

Problema: complejidad creciente en el proyecto.

Contexto: Silvia pide agregar nuevas categorías de productos con distintas reglas de descuento. Matías y Sabrina ven que su enfoque estructurado actual requiere condicionales y funciones específicas para cada tipo. Con POO, podrían crear una clase base `Producto` y luego clases derivadas especializadas, o al menos encapsular la lógica del producto en su propia clase, facilitando modificaciones y expansiones.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clases, objetos y atributos.

Una **clase** es un plano o plantilla que define las características (atributos) y las acciones (métodos) de algo. Un **objeto** es una instancia concreta de una clase. Si la clase es la receta, el objeto es el pastel resultante.

Los **atributos** son variables que almacenan el estado de un objeto. Por ejemplo, la clase `Producto` podría tener atributos como `nombre`, `precio`, `cantidadEnStock`.

```
public class Producto {
    String nombre;
    double precio;
    int cantidadEnStock;
}
```

Problema: representación de productos y clientes.

Contexto: Silvia quiere que el sistema trate cada producto como una entidad independiente, con su propio precio y stock. Además, desea que cada cliente y clienta tenga su nombre, su mail y un historial de compras.

Solución Propuesta:

Crear la clase `Producto` y la clase `Cliente`:

```
public class Cliente {
   String nombre;
   String email;
}
```

De este modo, podemos crear objetos `Producto` y `Cliente` con datos coherentes y fácilmente accesibles.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Creación de clases y objetos en Java.

Para crear un objeto a partir de una clase, usamos `new` y el constructor de la clase:

```
Producto cafe = new Producto();
cafe.nombre = "Café Premium Molido";
cafe.precio = 250.0;
cafe.cantidadEnStock = 100;
```

Esto crea un objeto `cafe` con sus atributos definidos. La separación es clara: `Producto` es el tipo (clase), `cafe` es el objeto (instancia).

Problema: instanciar productos desde el catálogo.

Contexto: Antes cargábamos los productos en listas de strings. Ahora queremos crear objetos `Producto` y guardarlos en un `ArrayList<Producto>`, para poder acceder a sus atributos y métodos directamente.

Solución:

```
ArrayList<Producto> catalogo = new ArrayList<>();
Producto teVerde = new Producto();
teVerde.nombre = "Té Verde Orgánico";
teVerde.precio = 180.0;
teVerde.cantidadEnStock = 50;
catalogo.add(teVerde);
```

Así el catálogo maneja objetos ricos en información, no sólo cadenas aisladas.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Métodos de instancia y de clase o *estáticos.*

Los **métodos de instancia** son aquellos que operan sobre un objeto específico. Por ejemplo, `descontarStock(int cantidad)` en la clase `Producto` afectará el stock de ese objeto particular.

Los **métodos de clase (estáticos)** no requieren una instancia para ser usados. Son útiles para lógica general, por ejemplo, `Producto.calcularImpuesto(precio)` podría ser un método estático si no depende de ningún estado interno de un objeto.

```
public class Producto {
    String nombre;
    double precio;
    int cantidadEnStock;

    public void descontarStock(int cantidad) {
        this.cantidadEnStock -= cantidad;
    }

    public static double calcularImpuesto(double precio) {
        return precio * 0.21; // 21% de impuestos
    }
}
```

Problema: lógica compartida vs. lógica específica de cada objeto.

Contexto: Queremos aplicar impuestos a todos los productos, pero el cálculo es el mismo para todos. Esto sugiere un método estático. Sin embargo, descontar stock es propio de cada producto, entonces un método de instancia es más adecuado.

Solución:

- Usar `calcularImpuesto(precio)` como estático.
- Usar `descontarStock(int cantidad)` como instancia.

Esto deja clara la responsabilidad: la lógica común a todos los productos (calcular impuestos) va en un método estático; la lógica que modifica el estado de un producto (descontar stock) es un método de instancia.

Constructores y métodos especiales.

Un **constructor** es un método especial que se invoca al crear un objeto con `new`. Se utiliza para inicializar los atributos del objeto de una forma coherente desde el principio. Si no definimos constructores, Java provee uno por defecto sin parámetros, pero es buena práctica crearlos cuando necesitamos asegurarnos de que el objeto nazca con datos válidos.

```
public class Producto {
    String nombre;
    double precio;
    int cantidadEnStock;

    // Constructor con parámetros
    public Producto(String nombre, double precio, int cantidadEnStock) {
        this.nombre = nombre;
        this.precio = precio;
        this.cantidadEnStock = cantidadEnStock;
    }
}
```

Al crear un producto:

```
Producto cafe = new Producto("Café Premium Molido", 250.0, 100);
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Problema: inicialización correcta de un producto.

Contexto: Silvia insiste en que no puede existir un producto sin nombre, precio o stock. Antes se asignaban los atributos manualmente después de crear el objeto, corriendo el riesgo de olvidarlo o dejarlos en un estado inválido.

Solución:

Con un constructor, nos aseguramos de que cada producto nazca con datos completos:

```
Producto nuevoProducto = new Producto("Chocolate Amargo 80%", 300.0,
30);
```

De esta forma, no hay forma de tener un producto "incompleto".

Situación inicial en TechLab.

Silvia, la Product Owner, observa que el código del proyecto de e-commerce comienza a ser difícil de mantener: tenemos datos de productos, clientes, descuentos y otras lógicas dispersas en funciones y variables globales. Matías y Sabrina notan la necesidad de un cambio de paradigma. En lugar de manipular datos de manera aislada, quieren "dar vida" a estos datos creando entidades llamadas objetos, cada uno con atributos (datos) y comportamientos (métodos).

En suma, el equipo se prepara para migrar hacia un enfoque orientado a objetos. Esto facilitará la representación del dominio (productos, clientes, pedidos) de forma más natural, haciendo el código más legible y flexible ante futuros cambios.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Ejercicios prácticos.

1. **Creación de clases y objetos:**
   - Crea una clase `Cliente` con atributos `nombre` y `email`.
   - Instanciá un objeto `Cliente` y asignale valores a sus atributos.
2. **Métodos de instancia:**
   - En la clase `Producto`, agrega un método `mostrarInformacion()` que imprima nombre, precio y stock.
   - Llamá a este método desde el `main` para varios objetos de `Producto`.
3. **Métodos:**
   - Agregá un método en `Producto` que calcule un descuento general para todos los productos de un 10%.
   - Probalo con distintos precios.
4. **Constructores:**
   - Creá un constructor para `Cliente` que reciba el `nombre` y `email`.
   - Creá varios clientes con este constructor y mostralos por pantalla.

Materiales y recursos adicionales

- Documentación Oficial de Java: Clases y Objetos
- Conceptos básicos de OOP en Java (Oracle Tutorials)
- Videos recomendados en YouTube sobre OOP en Java y mejores prácticas.

Preguntas para reflexionar

- ¿Cómo ayuda la POO a manejar la complejidad a medida que el proyecto de e-commerce crece?
- ¿Qué diferencias notás entre datos sueltos y datos encapsulados en objetos con atributos y métodos?
- ¿En qué casos usarías un método estático en lugar de uno de instancia?

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Próximos pasos

En la próxima clase, profundizaremos en las relaciones entre objetos, la herencia, la reutilización de código y otras características avanzadas de la POO. Mientras tanto, experimentá creando tus propias clases, instanciando objetos y aplicando métodos, para familiarizarte con este poderoso paradigma.

¡Felicitaciones por llegar hasta aquí! La Programación Orientada a Objetos abre un mundo de posibilidades para estructurar tus aplicaciones y, en TechLab, esto marcará la diferencia en calidad y mantenibilidad de nuestro software.

Buenos Aires Ciudad