Programación en Java – Clase 6

Introducción a la programación orientada a objetos

La programación orientada a objetos (de aquí en más la llamaremos POO) es un paradigma de programación que se basa en la idea de modelar el mundo real a través de la creación de objetos, los cuales son entidades que encapsulan datos y comportamientos relacionados.

En la POO, los objetos son instancias de clases, las cuales actúan como plantillas o moldes para crear objetos. Una clase define las características y comportamientos comunes que tienen los objetos de un determinado tipo. Por ejemplo, si estamos modelando un sistema de gestión de una biblioteca, podríamos tener una clase llamada "Libro" que define las propiedades y métodos comunes a todos los libros, como el título, autor, año de publicación, etc.

Los principales conceptos de la POO son:

- **Clases:** Son las plantillas para crear objetos. Definen las propiedades (atributos) y comportamientos (métodos) que tienen los objetos de ese tipo.

- **Objetos:** Son las instancias de una clase. Representan entidades concretas que poseen estados y comportamientos.

- **Encapsulamiento:** Es el mecanismo que permite ocultar los detalles internos de un objeto y exponer solo los métodos necesarios para interactuar con él. Se logra mediante la declaración de atributos privados y el uso de métodos públicos (getters y setters) para acceder y modificar dichos atributos.

- **Herencia:** Es un mecanismo que permite crear nuevas clases a partir de clases existentes, heredando sus propiedades y comportamientos. Permite establecer jerarquías y relaciones entre clases.

- **Polimorfismo:** Es la capacidad de que un objeto pueda tomar diferentes formas y comportarse de manera distinta según el contexto. Permite tratar diferentes objetos de manera uniforme a través de interfaces comunes.

La programación orientada a objetos ofrece ventajas como la reutilización de código, la modularidad, la organización y estructuración del código, y la capacidad de abstraer conceptos complejos en objetos más manejables.

Java es un lenguaje de programación orientado a objetos y se basa en estos conceptos. Proporciona un amplio soporte para la implementación de POO, lo que lo hace muy adecuado para el desarrollo de aplicaciones complejas y escalables.

Clases y objetos

Una clase es una plantilla o molde para la creación de objetos. Representa un conjunto de atributos (propiedades) y métodos (comportamientos) comunes que tendrán los objetos que se creen a partir de ella.

En términos más sencillos, una clase es una estructura que define cómo se deben crear y comportar los objetos. Podemos considerarla como un plano o una descripción de cómo se construirá un objeto y qué cosas puede hacer.

Una clase contiene la definición de sus atributos, que son variables que representan las características o datos que posee un objeto. Por ejemplo, si estamos modelando una clase "Persona", sus atributos podrían ser el nombre, la edad, el género, etc.

Además de los atributos, una clase también contiene los métodos, que son las acciones o comportamientos que puede realizar un objeto. Estos métodos definen qué puede hacer el objeto y cómo interactuar con él. Por ejemplo, en la clase "Persona", podríamos tener métodos como "caminar", "hablar" o "trabajar".

La clase actúa como una plantilla o molde a partir de la cual se crean objetos. Cuando se crea un objeto utilizando la clase, se dice que se está instanciando la clase. Cada objeto creado a partir de una clase tendrá sus propios valores para los atributos, pero compartirá los mismos métodos definidos en la clase.

Sobre los constructores

En Java, un constructor es un método especial que se utiliza para inicializar objetos de una clase. Se llama automáticamente cuando se crea una instancia (un objeto) de la clase mediante el operador `new`.

Algunas características importantes de los constructores en Java son:

1. Tienen el mismo nombre que la clase: El nombre del constructor debe ser exactamente igual al nombre de la clase en la que se encuentra.
2. No tienen tipo de retorno explícito: A diferencia de otros métodos, los constructores no tienen un tipo de retorno declarado. No se utiliza la palabra clave `void` ni ningún otro tipo.
3. Pueden recibir parámetros: Los constructores pueden tener parámetros, lo que permite inicializar los atributos del objeto con valores específicos al momento de la creación.
4. Pueden ser sobrecargados: Al igual que otros métodos, los constructores pueden ser sobrecargados, lo que significa que una clase puede tener varios constructores con diferentes parámetros. Esto brinda flexibilidad al crear objetos y permite diferentes formas de inicialización.

Veamos un ejemplo:

```
public class Persona {
    private String nombre;
    private int edad;

    // Constructor con parámetros
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    // Constructor sin parámetros (por defecto)
    public Persona() {
        this.nombre = "Sin nombre";
        this.edad = 0;
    }
```

En este ejemplo, la clase `Persona` tiene dos constructores: uno con parámetros que permite inicializar el nombre y la edad de la persona al momento de la creación del objeto, y otro sin parámetros que establece valores predeterminados. Cada vez que se crea un objeto `Persona` utilizando `new Persona(...)`, se invoca el constructor correspondiente para realizar la inicialización.

Los constructores son una parte importante de la programación orientada a objetos, ya que permiten establecer un estado inicial coherente para los objetos y garantizar que estén en un estado válido desde el principio.

Sobre el encapsulamiento

El encapsulamiento en Java es uno de los conceptos fundamentales de la programación orientada a objetos (POO). Es un principio que busca ocultar los detalles internos de una clase y brindar un acceso controlado a sus atributos y métodos desde el exterior.

El encapsulamiento se logra mediante la declaración de los atributos de una clase como privados (private) y el uso de métodos públicos (getters y setters) para acceder y modificar dichos atributos. Esto significa que los atributos privados no pueden ser accedidos directamente desde fuera de la clase, sino que se hace a través de los métodos públicos proporcionados.

Clave del encapsulamiento

1. **Protección de los datos:** Al declarar los atributos como privados, se evita que otros componentes externos modifiquen o accedan directamente a los datos internos de la clase. Esto ayuda a mantener la integridad de los datos y evita posibles manipulaciones no deseadas.

2. **Control de acceso:** El uso de métodos públicos para acceder y modificar los atributos permite tener un control más preciso sobre cómo se accede a los datos y qué validaciones o lógica adicional se puede aplicar antes de realizar cambios en ellos.

3. **Flexibilidad:** El encapsulamiento permite modificar los atributos internos de una clase sin afectar el código que utiliza dicha clase externamente. Esto proporciona una mayor flexibilidad para realizar cambios internos sin romper la funcionalidad externa.
4. **Mejorar el mantenimiento y depuración:** Al encapsular los atributos y métodos de una clase, se facilita el mantenimiento y la depuración del código. Los cambios o correcciones pueden hacerse en un solo lugar (en los métodos), sin necesidad de modificar directamente todos los puntos de acceso a los atributos.

Veamos un ejemplo:

```
public class Persona {
    private String nombre;
    private int edad;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getEdad() {
        return edad;
    }
```

```
    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

En este ejemplo, la clase "Persona" tiene dos atributos privados, "nombre" y "edad". Los métodos públicos getNombre, setNombre, getEdad y setEdad permiten acceder y modificar estos atributos de manera controlada desde fuera de la clase.

Acerca de "this"

En Java, `this` es una palabra clave que se utiliza para referirse al objeto actual dentro de una clase. Representa una referencia al objeto en el que se está ejecutando el código en ese momento.

La palabra clave "this" se utiliza principalmente en algunas situaciones:

1. Referencia a los atributos de la clase: Cuando hay un nombre de variable o atributo en la clase que tiene el mismo nombre que un parámetro o variable local, se puede utilizar "this" para referirse al atributo de la clase y evitar la ambigüedad.

Por ejemplo:

```
public class Persona {
    private String nombre;

    public void setNombre(String nombre) {
        this.nombre = nombre; // "this.nombre" se refiere al atributo "nombre" de la clase
    }
}
```

2. Llamada a constructores: "this" se puede utilizar para llamar a otro constructor dentro de la misma clase. Esto se conoce como "llamada al constructor" y se utiliza para reutilizar código común en diferentes constructores de la misma clase.

Por ejemplo:

```
public class Persona {
    private String nombre;
    private int edad;

    public Persona() {
        this("Jose Perez", 20); // Llamada al constructor con parámetros
    }

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
}
```