Material descargable

JAVA Programación Orientada a Objetos

egg

Programación Orientada a Objetos

La programación orientada a objetos (POO) es un paradigma de programación que se basa en el uso de "objetos" y sus interacciones para diseñar aplicaciones y programas de software. Los objetos son instancias de "clases", las cuales pueden incluir variables de instancia, métodos (funciones) y otros datos necesarios para su funcionamiento.

Clases e instancias

En la Programación Orientada a Objetos (POO), **una clase representa un modelo o plantilla que define las características (atributos o campos) y comportamientos (métodos) que deben tener los objetos de un tipo específico.** Una clase consiste en un conjunto de instrucciones que especifican cómo deben construirse las instancias de dicha clase.

Una instancia de una clase es un objeto concreto que se ha creado utilizando la plantilla de esa clase, mediante el operador **new**. Cada objeto instanciado tiene su propio conjunto de valores para los atributos definidos en la clase. Además, cada instancia puede acceder a los métodos que se han definido en la clase.

Veamos un ejemplo:

Clase

```
public class Perro {
    // Atributos
    private String nombre;
    private String raza;
    private int edad;

    // Constructor
    public Perro(String nombre, String raza, int edad) {
        this.nombre = nombre;
        this.raza = raza;
```

```
        this.edad = edad;
    }
}
```

Instancia

```
public class Main {
    public static void main(String[] args) {
        // Crear una instancia de la clase Perro
        Perro miPerro = new Perro("Max", "Labrador", 3);
    }
}
```

En este ejemplo, la clase Perro define las características de un perro. Luego, en la instancia miPerro, se crea un objeto específico de tipo Perro con el nombre "Max", la raza "Labrador" y la edad 3 años.

Palabra clave *this* y método Constructor

En Java, la palabra clave "this" se emplea como una referencia al objeto actual dentro de una clase. Un constructor, por otro lado, es un método especial diseñado para inicializar objetos de una clase. Este método lleva el mismo nombre que la clase y no tiene un tipo de retorno, ni siquiera void.

El propósito principal de un constructor es establecer los valores iniciales de los campos de un objeto cuando se crea dicho objeto. Si no se declara explícitamente un constructor en una clase Java, el compilador proporciona automáticamente un constructor sin parámetros, conocido como constructor predeterminado o constructor por defecto. Este constructor simplemente crea una instancia de la clase con los valores predeterminados de sus campos: 0 para tipos numéricos, false para booleanos y null para referencias de objetos.

```
public class Persona {
   String nombre;// propiedad de los objetos de la clase
   public Persona() {}// constructor por defecto, no hace falta declararlo
}
```

Sin embargo, si decides declarar uno o más constructores con parámetros en tu clase, el compilador no proporcionará un constructor por defecto. En tal caso, necesitarás proporcionar un constructor sin parámetros explícitamente si aún

deseas poder crear objetos sin pasar ningún argumento. Este comportamiento se conoce como "sobrecarga del constructor", que es un tipo de "sobrecarga de métodos".

Recuerda que la sobrecarga de métodos ocurre cuando se definen varios métodos con el mismo nombre pero con diferentes listas de parámetros. En tiempo de ejecución, la JVM determina qué método debe invocarse basándose en la cantidad y tipo de argumentos pasados.

La palabra clave "this" también puede utilizarse en los constructores para hacer referencia al objeto que se está construyendo. Además, puedes emplear "this" seguido de paréntesis para invocar a otro constructor en la misma clase, lo cual se conoce como "llamada al constructor" o "delegación al constructor". Por ejemplo:

```java
public class Persona {
    private String nombre;
    private Integer edad;

    // constructor por defecto
    public Persona() {
        this("Desconocido", 0);
    }

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

En este ejemplo, la llamada "this()" en el constructor por defecto invoca al constructor que toma nombre y edad como parámetros. Es importante destacar que estas llamadas al constructor deben ser la primera línea en el constructor. Así, al crear una nueva instancia de Persona sin parámetros, se llamará al constructor por defecto, que a su vez invocará al constructor con parámetros, pasándole "Desconocido" como nombre y 0 como edad, evitando que las propiedades se inicialicen con null. Mientras que, al crear una nueva instancia con parámetros, se llamará directamente al constructor con parámetros, permitiendo la inicialización personalizada de la instancia.

Modificador static

En Java, **una variable estática** es una variable que pertenece a la clase en lugar de a las instancias individuales de esa clase. Esto significa que solo hay una copia de la variable estática, independientemente del número de objetos creados a partir de la clase. En otras palabras, todas las instancias de la misma clase comparten la misma variable estática. Este concepto resulta útil cuando se necesita una variable que conserve su valor y sea común a todas las instancias de la clase.

Similarmente, un método estático es un método que pertenece a la clase en lugar de a una instancia específica de esa clase. Los métodos estáticos no pueden acceder a variables de instancia ni a métodos de instancia, ya que estos requieren una instancia de la clase para existir. Los métodos estáticos suelen utilizarse para realizar operaciones que no dependen de datos específicos del objeto.

```
public class Clase {
    Integer variableNoEstatica;
    static Integer variableEstatica;

    public static void metodoEstatico() {

    }

    public void metodoNoEstatico() {

    }
}
```

Es importante mencionar que las variables estáticas se declaran dentro de una clase pero fuera de cualquier método, constructor o bloque. No pueden declararse dentro de métodos, constructores o bloques de código a menos que estén dentro de un bloque estático, en cuyo caso están limitadas a ese bloque. Para invocar métodos y variables estáticas, se deben utilizar directamente con el nombre de la clase, por ejemplo, Clase.metodoEstatico() o Clase.variableEstatica.

Métodos getters y settters

Los métodos getters y setters son esenciales para la encapsulación y la ocultación de datos en Java. Estos métodos se utilizan para obtener (getters) y establecer (setters) el valor de un atributo privado de una clase. En el siguiente ejemplo de la clase Persona, los atributos nombre y edad son privados, lo que significa que solo pueden ser accedidos y modificados dentro de la propia clase.

```
public class Persona {
    // Los atributos son privados, lo que significa que solo pueden ser
accedidos directamente dentro de esta clase.
    private String nombre;
    private int edad;

    // Este es el constructor de la clase.
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    // Un método getter para el nombre.
    public String getNombre() {
        return nombre;
    }

    // Un método setter para el nombre.
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    // Un método getter para la edad.
    public int getEdad() {
        return edad;
    }

    // Un método setter para la edad.
    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

Estos métodos permiten un acceso controlado a los datos de la clase, lo que mejora la seguridad y la robustez del código al evitar el acceso directo a los atributos privados desde fuera de la clase. Los getters y setters permiten establecer y obtener los valores de los atributos de forma segura y coherente, contribuyendo así a un diseño más modular y mantenible del código.