POO Tema 1. Encapsulación: tipos de datos, clases y objetos

Martín González Dios

2 de enero de 2025

Consejos generales:

1. El método main suele tener muy pocas líneas de código.
2. La clase principal no debe tener ningún atributo.

1. Encapsulación

Asegura la **integridad de los datos**, limitando los trozos de código del programa que pueden acceder a unos datos dados.
Idealmente no existe ningún fragmento de código que pueda acceder a todos los datos del programa.
Se usa el **principio de ocultación**, en virtud del cual solo un fragmento de código puede "ver" un conjunto de datos dado.

2. Clases

Los tipos de datos son las entidades que se quieren representar en el programa, estas **entidades** se conceptualizan como **clases**. Todos los tipos de datos son clases, y los **valores concretos** de esas clases se denominan **objetos**.
En Java existen 2 especiaes de tipos de datos:

1. Tipos de **datos primitivos** (int, float, char, ...)
2. **Clases**: están vinculados a las entidades de alto nivel del programa.

2.1. Encapsulación en las clases

Las clases contienen 2 tipos de código:

1. **Variables o atributos**: características que definen la entidad representada por la clase.
2. **Funciones o métodos**: acceden a los atributos para permitir la lectura y escritura, y para implementar funcionalidades.

Para solventar el problema de la integridad de datos las únicas funciones que pueden acceder a los atributos son las funciones de la clase.

2.2. Tipos de acceso

Se pueden aplicar a **atributos y a métodos**.

1. Público.
   - **Métodos públicos**: pueden ser invocados desde cualquier método de cualquier clase del programa.
   - **Atributos públicos**: puede ser leídos o escritos desde cualquier método de cualquier clase del programa.
2. Privado
   - **Métodos privados**: solamente pueden ser invocados desde métodos de la clase a la que pertenecen.
   - **Atributos privados**: solamente se pueden leer o escribir desde métodos de la clase a la que pertenecen.
3. Acceso a paquete.
   - **Métodos de acceso a paquete**: solamente pueden invocarlos los métodos de las clases que pertenecen al mismo paquete de la clase en la que se encuentra el método, se comportan como privados para cualquier otro método.
   - **Atributos de acceso a paquete**: solamente se pueden leer o escribir desde los métodos de las clases que pertenecen al mismo paquete de la clase en la que se encuentra el atributo, se comportan como privados para cualquier otro método.
4. Protegido.
   - **Métodos protegidos**: pueden ser invocados por los métodos de las clases que pertenecen al mismo paquete de la clase y por los métodos de las subclases de la clase.
   - **Atributos protegidos**: pueden ser leídos o escritos por los métodos de las clases que pertenecen al mismo paquete de la clase del atributo y por los métodos de las subclases de la clase en la que se encuentra el atributo.

Generalmente los **atributos** deben ser **privados** y los **métodos públicos**. Se pueden **usar métodos privados** para facilitar la codificación de los métodos.

2.3. Getters y setters

- **Getters**: métodos que devuelven el valor que tienen los atributos en cada momento.
- **Setters**: métodos que escriben el valor de los atributos.

Cada **atributo** de la clase tiene **un único** getter/setter.
Buenas praxis:

1. Los **setters** deben incluir una **condición** para evitar la escritura de null como valor de los atributos.
2. Generalmente **todos los atributos** tienen que tener **getter y setter**, salvo que solamente tengan interés o sean usados como parte de la programación de los métodos de la clase.

2.4. Constructores

Si el **atributo** es un **dato de tipo primitivo** se realiza **reserva de memoria** y se le asigna un valor inicial (dependiendo del tipo).
Si el **atributo no es un dato primitivo** no se le reserva automáticamente memoria, se le asigna un **valor inicial null**, que en la práctica se trata como una variable que no existe. La **reserva de memoria** se hace invocando a los **constructores** a través del **operador new**.

Un constructor es un método que se invoca para:

1. **Reservar memoria** para un objeto de la clase.
2. Reservar memoria para los atributos de dicho objeto.
3. Asignar **valores iniciales** a los atributos del objeto.

Un constructor se invoca un única vez para cada objeto.

Constructor por defecto: proporcionado por Java automáticamente.

- No tiene argumentos.
- Inicializa los atributos a sus valores por defecto.
- Tiene el mismo efecto que especificar un constructor sin argumentos con cuerpo vacío.

Buenas praxis:

1. **No** está bien no definir constructores y **usar el constructor por defecto de Java**.
2. Se deben proporcionar constructores que tienen como **argumentos** los **valores de los atributos que es obligatorio inicializar** para crear un objeto.

2.5. Métodos funcionales

Acceden directamente a los valores de los atributos.
Se crean cuando los **datos necesarios** para realizar las operaciones **son** los **valores de los atributos de dicha clase**. Los métodos se invocan desde los objetos de las clases con el **operador "."**.
Pueden tener varias implementaciones, se dice entonces que el **método** está **sobrecargado**. La firma del método debe ser diferente.

2.6. Método toString

Devuelve la **representación en texto de un objeto**, este método lo tienen todas las clases de Java, ya que lo heredan de la clase Object. La implementación por defecto no aporta información, es **necesario reimplementar el método**.

Buenas praxis:

1. Todas las clases que necesiten **mostrar características por consola** deberían **reimplementar toString**.
2. No es necesario incluir todos los valores de todos los **atributos**, sirve con los que se **consideren relevantes**.

3. Registros (Records)

Clase cuyas **instancias** son **inmutables**, es decir, **no pueden cambiar los valores de los atributos en tiempo de ejecución**.

1. Todos sus **atributos** son **privados**.
2. Todo atributo tiene su **getter correspondiente**.
3. Los atributos **no tienen setters**, ni ningún otro método que pueda modificar los valores de los atributos.
4. Tienen un constructor, denominado **constructor canónico**, con sus correspondientes argumentos para cada uno de los atributos.
5. El toString incluye el nombre de la clase y el nombre de los atributos con sus valores.