Consultores y Modificadores (Getters y Setters)

Programación y Diseño Orientado a Objetos (PDOO)

📋 Tabla de Contenidos

Introducción

En **Programación y Diseño Orientado a Objetos (PDOO)**, los **Consultores (Getters)** y **Modificadores (Setters)** son métodos utilizados para interactuar con los atributos (el estado interno) de un objeto, manteniendo el principio de **Encapsulamiento**.

Consultores (Getters)

Métodos encargados de **devolver el valor de un atributo**.

Características Clave

Característica	Descripción
Función Principal	Leer el estado de un objeto
Flexibilidad	Pueden devolver el valor directo, modificado, o una copia
Ámbito	De instancia (atributos de objeto) o de clase (atributos estáticos)
Regla de Oro	✅ Solo crear los que sean **estrictamente necesarios**

Convenciones de Nombres

Lenguaje	Convención	Ejemplo

Java	getAtributo()	getNombre()
Ruby	atributo	nombre

Ejemplo en Java

```
public class Persona {
    // Atributo de instancia
    private String nombre;
    // Consultor de instancia (Getter)
    public String getNombre() {
        return nombre;
    }
    // Consultor de clase (Atributo estático)
    private static final int MAYORIA_EDAD = 18;
    public static int getMayoriaEdad() {
        return MAYORIA_EDAD;
    }
}
```

Ejemplo en Ruby

```
class Persona
    # Uso de abreviatura para crear el consultor implícito
    attr_reader :nombre # Crea el método 'nombre'
    def initialize(nombre)
        @nombre = nombre
    end
    # Uso del consultor
    # p = Persona.new("Adam")
    # puts p.nombre
end
```

Modificadores (Setters)

Métodos encargados de **modificar o establecer el valor de un atributo**.

Características Clave

Característica	Descripción
Función Principal	Escribir o cambiar el estado de un objeto
Validación	⚠ Controlar las restricciones sobre el atributo
Ámbito	De instancia o de clase
Regla de Oro	⚠ Solo crear los que sean **estrictamente necesarios**

Convenciones de Nombres

Lenguaje	Convención	Ejemplo
Java	setAtributo(...)	setNombre("...")
Ruby	atributo=	nombre = "..."

Ejemplo en Java

```
public class Persona {
   private int edad;
   // Modificador de instancia (Setter)
   public void setEdad(int nuevaEdad) {
      // Validación: Controlar restricciones
      if (nuevaEdad > 0) {
         this.edad = nuevaEdad;
      } else {
         System.out.println("Error: La edad debe ser positiva.");
      }
   }
}
```

Ejemplo en Ruby

```
class Persona
   # Uso de abreviatura para crear consultor y modificador a la vez
   attr_accessor :nombre # Crea 'nombre' (getter) y 'nombre=' (setter)
   # Modificador de instancia explícito (Convención 'atributo=')
   def edad=(nueva_edad)
      # Aquí se añaden las comprobaciones
      @edad = nueva_edad if nueva_edad > 0
   end
end
```

Problemática de Referencias

En lenguajes como **Java** (objetos) y **Ruby** (todos los objetos), las variables contienen **referencias (punteros)**.

 ADVERTENCIA: Si un atributo interno es un objeto **mutable** (como java.util.Date o GregorianCalendar), devolver su referencia directamente a través de un consultor o aceptarla directamente en un modificador **rompe el encapsulamiento**.

El Problema del Encapsulamiento Roto

Al devolver la referencia, el código externo puede modificar el objeto a través de esa referencia, cambiando el estado interno del objeto **sin pasar por el modificador ni sus validaciones**.

Ejemplo del Problema (Java - GregorianCalendar es Mutable)

 Código Problemático:

```
// Clase Persona
public GregorianCalendar getFechaNacimiento() {
   // ¡PROBLEMA! Devuelve la referencia directa
   return fechaNacimiento;
}
```

Consecuencia:

```
// Código externo
Persona juan = new Persona(new GregorianCalendar(1989, 10, 28)); // Noviembre 28, 1989
// 1. Obtenemos la referencia interna
GregorianCalendar lectura = juan.getFechaNacimiento();
// 2. Modificamos el objeto A TRAVÉS de la referencia externa
lectura.set(1985, 5, 13); // Cambiado a Junio 13, 1985
// El estado interno de 'juan' ha cambiado, ¡aunque no se usó el setter!
// Esto rompe el encapsulamiento y el control.
```

Solución: Copias Defensivas

Para objetos mutables, la solución es usar **Copias Defensivas**:

En...	Acción
Consultor (Getter)	Devolver una **copia** del objeto en lugar de la referencia original
Modificador (Setter)	Guardar una **copia** del objeto recibido en lugar de la referencia

☑ Implementación Correcta

```
public class Persona {
   private GregorianCalendar fechaNacimiento;
   // Getter con Copia Defensiva
   public GregorianCalendar getFechaNacimiento() {
      // Se devuelve una COPIA, la referencia original queda protegida
```

```
        return (GregorianCalendar) fechaNacimiento.clone();
    }
    // Setter con Copia Defensiva
    public void setFechaNacimiento(GregorianCalendar nuevaFecha) {
        // Se almacena una COPIA, así si 'nuevaFecha' se modifica fuera, no afecta al objeto interno
        this.fechaNacimiento = (GregorianCalendar) nuevaFecha.clone();
    }
}
```

📊 Resumen de Buenas Prácticas

Práctica	Descripción	Importancia
Minimizar Getters/Setters	Solo crear los estrictamente necesarios	⭐ Crítico
Validación en Setters	Controlar restricciones del dominio	⭐ Crítico
Copias Defensivas	Para objetos mutables (fechas, colecciones)	⭐ Importante
Encapsulamiento	Proteger el estado interno del objeto	⭐ Crítico
Nombres Consistentes	Seguir convenciones del lenguaje	⭐ Recomendado

---

Adam Bourbahh Romero ~ PDOO