Informe de Implementación

Luis Carlos Cortez Guzmán

Descripción de la Clase Abstracta

Clase Abstracta `Animal`

La clase abstracta `Animal` está diseñada para establecer un marco común para todos los animales en nuestro modelo. Esta clase actúa como una plantilla que define un método abstracto `emitirSonido()` y un método concreto `dormir()`.

Métodos de la Clase Abstracta:

1. `emitirSonido()`: Método abstracto que debe ser implementado por todas las subclases. Este método no tiene implementación en la clase `Animal`, obligando a cada subclase a proporcionar su propia versión específica del sonido que hace el animal.

2. `dormir()`: Método concreto que imprime un mensaje genérico indicando que el animal está durmiendo. Este método puede ser utilizado tal cual o sobrescrito por las subclases para proporcionar más detalles específicos sobre el comportamiento de dormir de cada animal.

El diseño de esta clase abstracta facilita la creación de nuevas subclases de animales, asegurando que todas compartan un comportamiento común mientras permiten la especificación de comportamientos únicos.

Descripción de Subclases

Se han creado tres subclases concretas: `Leon`, `Elefante` y `Canguro`. Cada una de estas subclases implementa el método abstracto `emitirSonido()` de una manera que refleja el sonido característico del animal, y sobrescribe el método `dormir()` para proporcionar detalles específicos sobre el hábitat o características físicas del animal.

Clase `Leon`

- Implementación del método `emitirSonido()`: Imprime un mensaje indicando el rugido característico del león.

- Sobrescritura del método `dormir()`: Proporciona detalles sobre el león durmiendo en su hábitat específico.

Leon.java > ...

```java
class Leon extends Animal {
    private String habitat;
    private String nombre;

    public Leon(String nombre, String habitat) {
        this.nombre = nombre;
        this.habitat = habitat;
    }

    @Override
    void emitirSonido() {
        System.out.println("El león " + nombre + " ruge: ¡Roaaar!");
    }

    @Override
    void dormir() {
        System.out.println("El león " + nombre + " está durmiendo en " + habitat + ".");
    }
}
```

Clase `Elefante`

- Implementación del método `emitirSonido()`: Imprime un mensaje indicando el barrito característico del elefante.

- Sobrescritura del método `dormir()`: Proporciona detalles sobre el elefante durmiendo, incluyendo su peso.

Elefante.java > ...

```java
class Elefante extends Animal {
    private double peso;
    private String nombre;

    public Elefante(String nombre, double peso) {
        this.nombre = nombre;
        this.peso = peso;
    }

    @Override
    void emitirSonido() {
        System.out.println("El elefante " + nombre + " barrita: ¡Prrrrr!");
    }

    @Override
    void dormir() {
        System.out.println("El elefante " + nombre + " está durmiendo. Pesa " + peso + " kg.");
    }
}
```

Clase `Canguro`

- Implementación del método `emitirSonido()`: Imprime un mensaje indicando el sonido característico del canguro.

- Sobrescritura del método `dormir()`: Proporciona detalles sobre el canguro durmiendo, incluyendo su altura.

Canguro.java > ...

```java
class Canguro extends Animal {
    private double altura;
    private String nombre;

    public Canguro(String nombre, double altura) {
        this.nombre = nombre;
        this.altura = altura;
    }

    @Override
    void emitirSonido() {
        System.out.println("El canguro " + nombre + " hace un sonido característico: ¡Click click!");
    }

    @Override
    void dormir() {
        System.out.println("El canguro " + nombre + " está durmiendo. Mide " + altura + " metros.");
    }
}
```

Problemas Encontrados y Soluciones

Problema: Al principio, fue un desafío decidir qué métodos deberían ser abstractos y cuáles concretos en la clase `Animal`. Quería asegurar que todas las subclases compartieran ciertos comportamientos comunes pero también necesitaba flexibilidad para comportamientos específicos.

Solución: decidí hacer que `emitirSonido()` fuera un método abstracto, forzando a cada subclase a proporcionar su propia implementación. Para `dormir()`, cree un método concreto con un mensaje genérico, que podría ser sobrescrito por las subclases para proporcionar detalles específicos. Este enfoque equilibró la necesidad de comportamiento común y específico.

Demostración del Código

A continuación se presenta el código de demostración, que crea instancias de cada animal y llama a sus métodos `emitirSonido()` y `dormir()`.

Código de Demostración

PruebaAnimales.java > ...

```java
public class PruebaAnimales {
    Run | Debug
    public static void main(String[] args) {
        // Crear instancias de cada animal
        Animal leon = new Leon(nombre:"Simba", habitat:"la sabana");
        Animal elefante = new Elefante(nombre:"Dumbo", peso:5000);
        Animal canguro = new Canguro(nombre:"Joey", altura:1.8);

        // Probar el león
        System.out.println(x:"Pruebas para el León:");
        leon.emitirSonido();
        leon.dormir();
        System.out.println();

        // Probar el elefante
        System.out.println(x:"Pruebas para el Elefante:");
        elefante.emitirSonido();
        elefante.dormir();
        System.out.println();

        // Probar el canguro
        System.out.println(x:"Pruebas para el Canguro:");
        canguro.emitirSonido();
        canguro.dormir();
        System.out.println();
    }
}
```

Salida Esperada

```
Pruebas para el León:
El león Simba ruge: ¡Roaaar!
El león Simba está durmiendo en la sabana.

Pruebas para el Elefante:
El elefante Dumbo barrita: ¡Prrrrr!
El elefante Dumbo está durmiendo. Pesa 5000.0 kg.

Pruebas para el Canguro:
El canguro Joey hace un sonido característico: ¡Click click!
El canguro Joey está durmiendo. Mide 1.8 metros.
```

Conclusión

El diseño de la clase abstracta `Animal` y sus subclases `Leon`, `Elefante` y `Canguro` demuestra cómo compartir comportamientos comunes y especificar comportamientos únicos para cada tipo de animal. La clase abstracta proporciona una base sólida y extensible para añadir más animales en el futuro, asegurando una estructura coherente y fácil de mantener. La implementación de métodos abstractos y concretos facilita tanto la reutilización del código como la especificidad necesaria para representar adecuadamente cada animal.