Universidad Mariano Gálvez

Centro Regional: Boca del Monte

Facultad: Ingeniería en sistemas

Curso: Programación II

Sección: A

Implementar alguno de los principios solid

Liskov

Nombres: Elmer Yúnior Roblero Santos

Pedro Javier De León Coy

Wilmer Orlando García Salazar

Números de carnets: 7690-23-14847

7690-22-8894

7690-16-13523

Fecha:25/08/2024

Guatemala, de 2024

1. **Superclase: Animal**
    - **Descripción**: Esta es la clase base o superclase que define un comportamiento general para todos los animales, a través del método makeSound (). Es el punto de partida en la jerarquía de clases.
    - **Relación**: Todas las subclases heredan de Animal. En el diagrama, esto se representa por líneas etiquetadas con "extends", que conectan Animal con cada subclase.

2. Subclases: Perro, Gato, Ave, Pez, León

- **Descripción**: Estas son clases específicas que extienden la funcionalidad de Animal. Cada una sobrescribe el método makeSound () para proporcionar un comportamiento específico:
    - Perro (class) - Emite "Bark".
    - Gato (class) - Emite "Meow".
    - Ave (class) - Emite "Chirp".
    - Pez (class) - Emite "Blub".
    - Lean (class) - Emite "Roar".
- **Relación**: Estas subclases están conectadas a Animal con la etiqueta "extends", indicando que heredan de Animal.

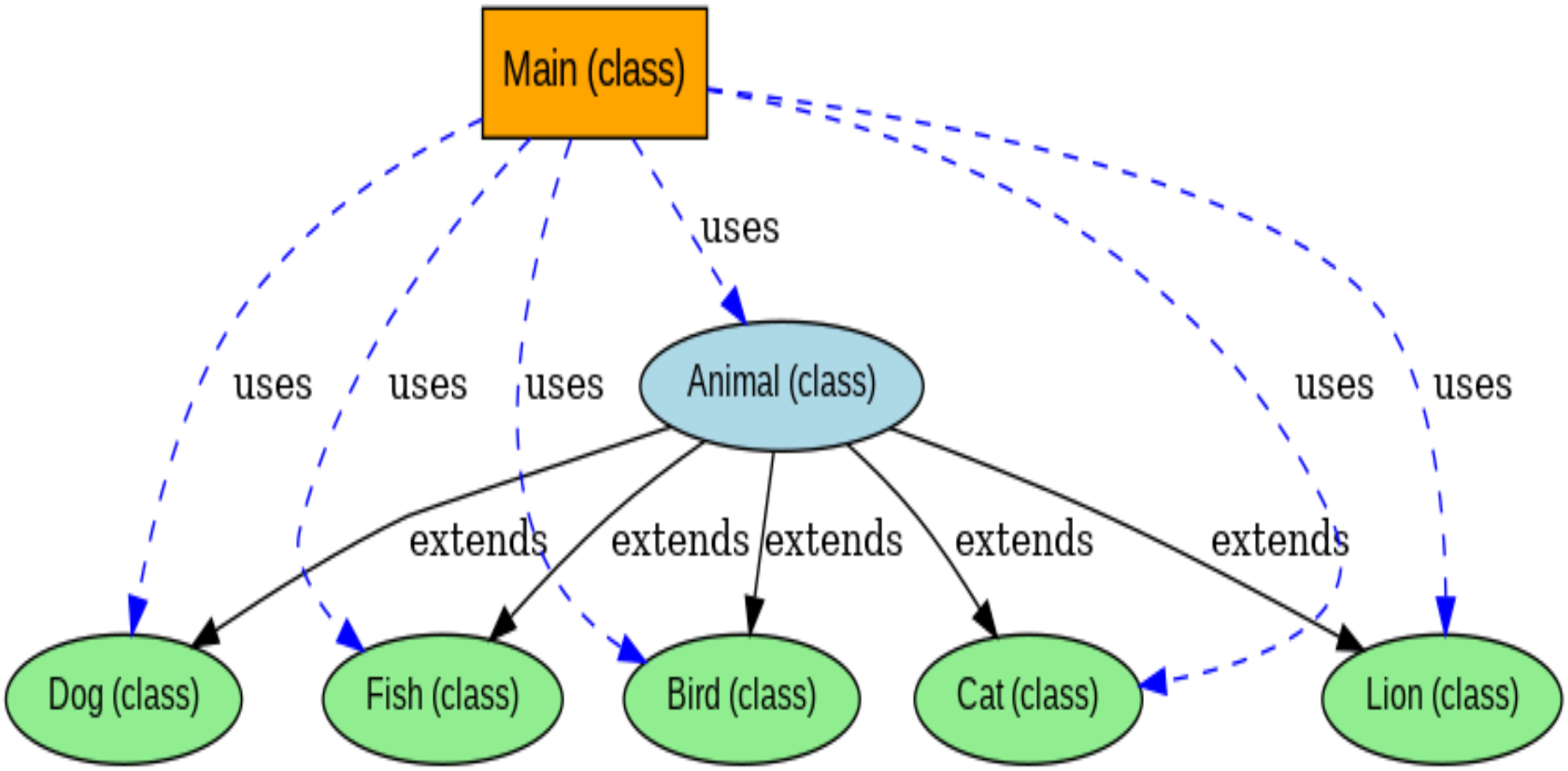
3. Clase Cliente: Main (class)

- **Descripción**: Esta es la clase que utiliza las instancias de Animal y sus subclases. En el método main (), se crea una instancia de cada subclase (Perro, Gato, etc.) y se le pasa al método someMethod () que espera un Animal. Este método llama a makeSound (), que invoca el comportamiento específico de la subclase.
- **Relación**: En el diagrama, Main está conectada tanto a Animal como a cada una de sus subclases con la etiqueta "uses". Esto indica que la clase Main utiliza (o invoca métodos de) Animal y sus subclases.

Propósito del Diagrama:

El diagrama muestra cómo el **Principio de Sustitución de Liskov** permite que cualquier subclase de Animal pueda ser utilizada en lugar de la superclase Animal sin cambiar la funcionalidad del programa. En otras palabras, una instancia de Perro, Gato, Ave, Pez, o León puede sustituir una instancia de Animal en cualquier lugar del código, y el programa seguirá comportándose correctamente.

Este principio es fundamental en la programación orientada a objetos para garantizar la flexibilidad y escalabilidad del código. Al seguir LSP, el código es más fácil de mantener, extender y probar.


Main (class)
uses
uses
uses
uses
Animal (class)
uses
uses
extends
extends
extends
extends
extends
Dog (class)
Fish (class)
Bird (class)
Cat (class)
Lion (class)