```
ArrayList<Mamifero> mamiferos = new ArrayList<Mamifero>();
ArrayList<Anfibio> anfibios = new ArrayList<Anfibio>();
ArrayList<Pez> peces = new ArrayList<Pez>();
ArrayList<Reptil> reptiles = new ArrayList<Reptil>();
ArrayList<Ave> aves = new ArrayList<Ave>();

//Agregar a los Arraylists
mamiferos.add(m1);
mamiferos.add(m2);
mamiferos.add(m3);
anfibios.add(an1);
anfibios.add(an2);
anfibios.add(an3);
aves.add(a1);
aves.add(a2);
aves.add(a3);
reptiles.add(r1);
reptiles.add(r2);
reptiles.add(r3);
peces.add(p1);
peces.add(p2);
peces.add(p3);
```

```
Scanner scanear = new Scanner(System.in);
String usuarioNombre = "", usuarioEspecie = "", usuarioAlimento = "", usuarioComportamiento = "";
int key, usuarioTotal;
do {
    System.out.println("-------Zoologico-------");
    System.out.println("1. Mostrar lista mamiferos\n2. Mostrar lista aves\n3. Mostrar lista reptiles\r
    System.out.print("6. Agregar un nuevo animal\n0. Salir\nSeleccion: ");
    key = scanear.nextInt();
    switch (key) {
        case 1:
            System.out.println(mamiferos);
            break;
        case 2:
            System.out.println(aves);
            break;
        case 3:
            System.out.println(reptiles);
            break;
        case 4:
            System.out.println(anfibios);
            break;
        case 5:
            System.out.println(peces);
            break;
        case 6:
            System.out.print("Nombre: ");
            usuarioNombre = scanear.nextLine();
```

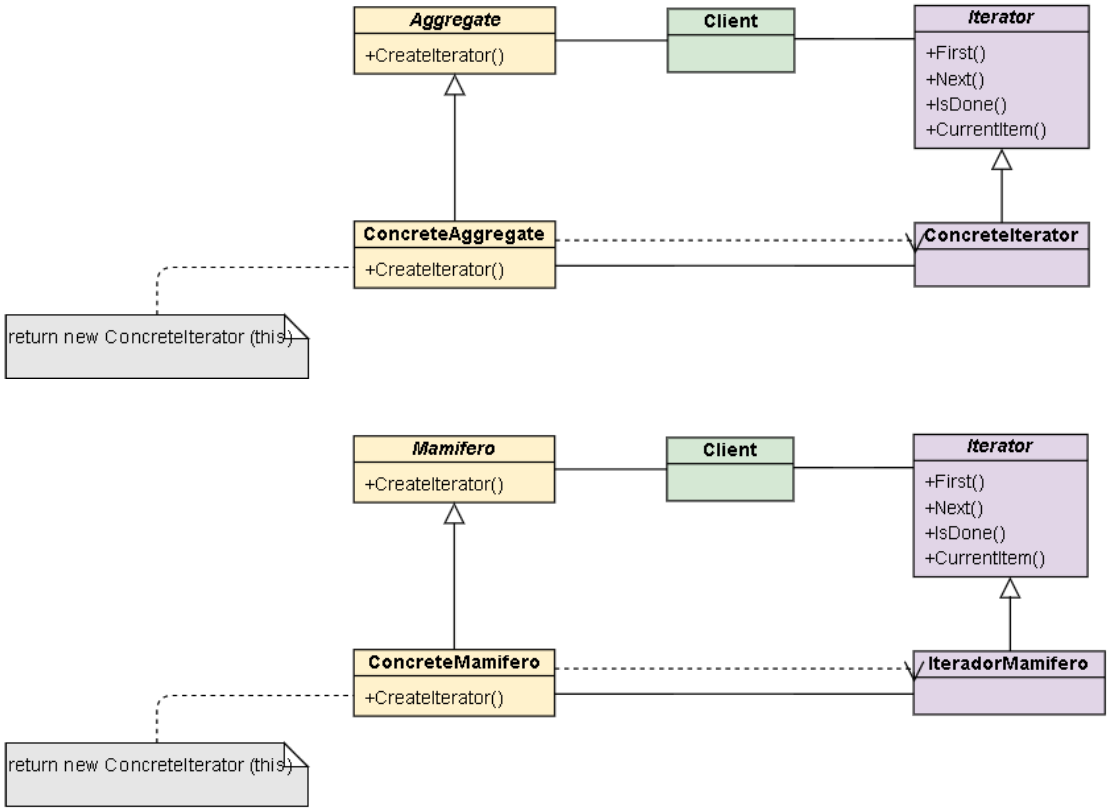
Objetivo:

Proporcionar un modo de acceder secuencialmente a los elementos de la lista, sin exponer su estructura. En este caso, un ArrayList de animales. Es especialmente útil cuando trabajamos con estructuras de datos complejas, ya que nos permite recorrer sus elementos mediante un Iterador.

Motivación:

El patrón surge del deseo de acceder a los elementos de un contenedor de objetos (en este caso, varios ArrayLists) sin exponer su representación interna. Además, es posible que se necesite más de una forma de recorrer la estructura siendo para ello necesario crear modificaciones en la clase.

Diagrama de Clases:



- **Client**: Actor que utiliza al Iterator.
- **Aggregate**: Interface que define la estructura de las clases que pueden ser iteradas.
- **ConcreteAggregate**: Clase que contiene la estructura de datos que deseamos iterar (cualquiera de los ArrayList declarados en el código).
- **IIterator**: Interface que define la estructura de los iteradores, la cual define los métodos necesarios para poder realizar la iteración sobre el ConcreteAggregator.
- **ConcreteIterator**: Implementación de un iterador concreto, el cual hereda de IIterator para implementar de forma concreta cómo iterar un ConcreteAggregate.