1. Ejercicios

EJERCICIO 1: ASTROS

Define una jerarquía de clases que permita almacenar datos sobre los planetas y satélites (lunas) que forman parte del sistema solar. Algunos atributos que necesitaremos almacenar son:

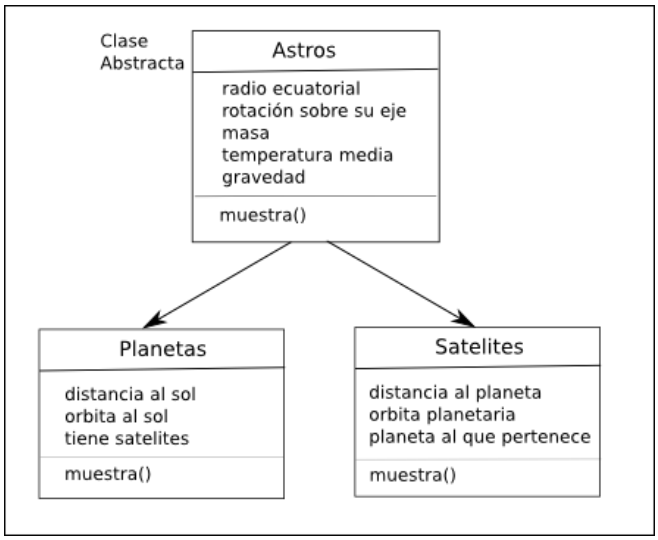
- Masa del cuerpo.
- Diámetro medio.
- Período de rotación sobre su propio eje.
- Período de traslación alrededor del cuerpo que orbitan.
- Distancia media a ese cuerpo.
- etc.

Define las clases necesarias conteniendo:

- Constructores.
- Métodos para recuperar y almacenas atributos.
- Método para mostrar la información del objeto.

Define un método, que dado un objeto del sistema solar (planeta o satélite), imprima toda la información que se dispone sobre el mismo (además de su lista de satélites si los tuviera).

El diagrama UML sería:



Ayuda:

Una posible solución sería crear una lista de objetos, insertar los planetas y satélites (directamente mediante código o solicitándolos por pantalla) y luego mostrar un pequeño menú que permita al usuario imprimir la información del astro que elija.

EJERCICIO 2: MASCOTAS

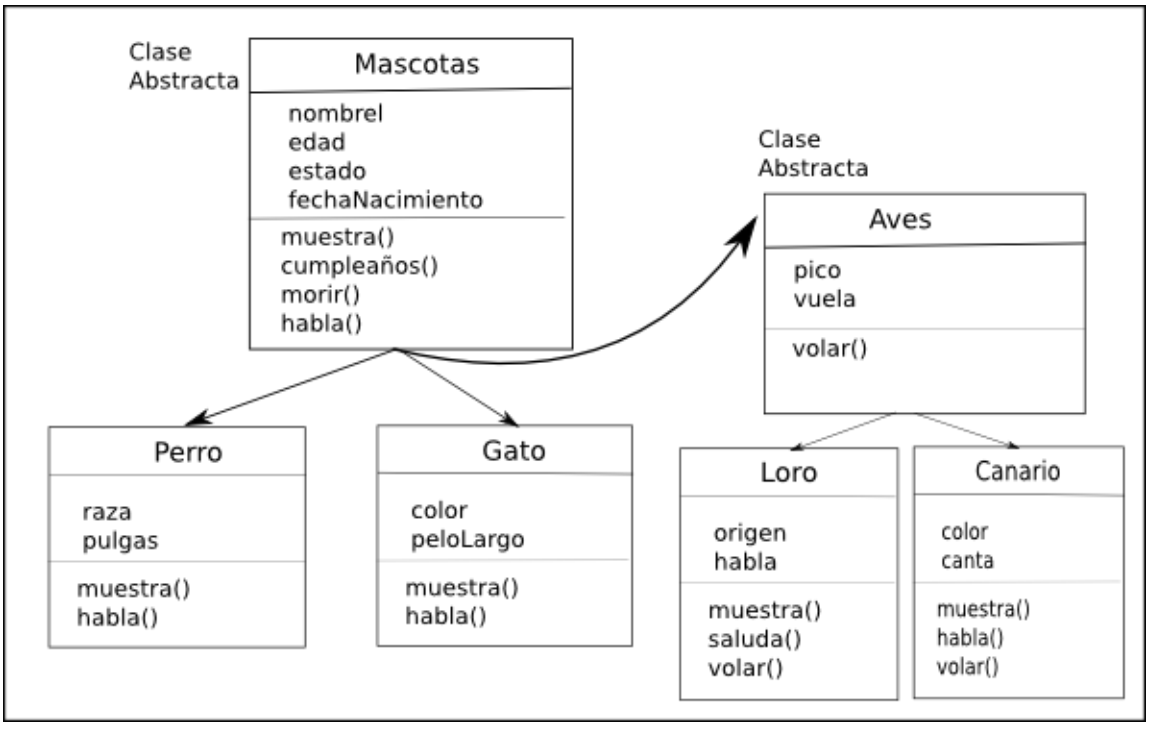
Implementa una clase llamada **Inventario** que utilizaremos para almacenar referencias a todos los animales existentes en una tienda de mascotas.

Esta clase debe cumplir con los siguientes requisitos:

- En la tienda existirán 4 tipos de animales: perros, gatos, loros y canarios.
- Los animales deben almacenarse en un ArrayList privado dentro de la clase **Inventario**.
- La clase debe permitir realizar las siguientes acciones:
  - Mostrar la lista de animales (solo tipo y nombre, 1 línea por animal).
  - Mostrar todos los datos de un animal concreto.
  - Mostrar todos los datos de todos los animales.
  - Insertar animales en el inventario.
  - Eliminar animales del inventario.
  - Vaciar el inventario.

Implementa las demás clases necesarias para la clase Inventario.

El diagrama UML sería:



Ampliación:

Como alternativa a mostrar los datos de los animales, sobrescribe el método **toString()** de la clase Object en las clases correspondientes.

EJERCICIO 3: BANCO

Vamos a hacer una aplicación que simule el funcionamiento de un banco. Crea una clase **CuentaBancaria** con los atributos: **iban** y **saldo**. Implementa métodos para:

- Consultar los atributos.
- Ingresar dinero.
- Retirar dinero.
- Traspasar dinero de una cuenta a otra.

Para los tres últimos métodos puede utilizarse internamente un método privado más general llamado **añadir(...)** que añada una cantidad (positiva o negativa) al saldo.

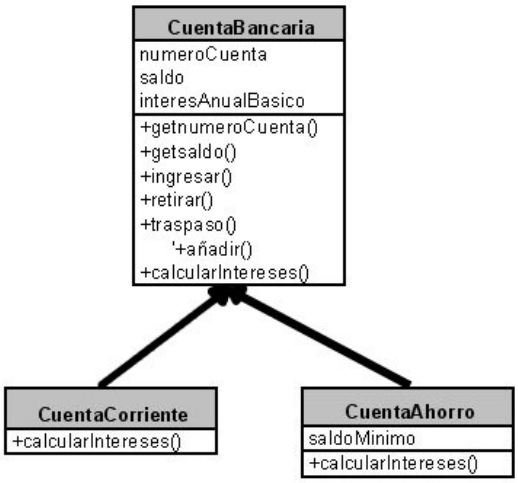
También habrá un atributo común a todas las instancias llamado **interesAnualBasico**, que en principio puede ser constante. La clase tiene que ser ***abstracta*** y debe tener un método **calcularIntereses()** que se dejará sin implementar. También puede ser útil implementar un método para mostrar los datos de la cuenta.

De esta clase heredarán dos subclases: **CuentaCorriente** y **CuentaAhorro**. La diferencia entre ambas será la manera de calcular los intereses:

- A la primera se le incrementará el saldo teniendo en cuenta el interés anual básico.
- La segunda tendrá una constante de clase llamada **saldoMinimo**. Si no se llega a este saldo el interés será la mitad del interés básico. Si se supera el saldo mínimo el interés aplicado será el doble del interés anual básico.

Implementa una clase principal con función main para probar el funcionamiento de las tres clases: Crea varias cuentas bancarias de distintos tipos, pueden estar en un ArrayList si lo deseas; prueba a realizar ingresos, retiradas y transferencias; calcula los intereses y muéstralos por pantalla; etc.

El diagrama UML sería:



Ampliación

En la clase principal aplica el concepto de polimorfismo: CuentaBancaria puede ser CuentaCorriente o CuentaAhorro.

EJERCICIO 4: ESTADÍSTICAS

Dada la siguiente interface:

```
public interface Estadisticable {
	double minimo();
	double maximo();
	double media();
	int cuantos();
}
```

Se pide:

1. Diseña la clase ArrayListDoubleEstadistica que tenga un ArrayList de double y que implemente la interface Estadísticas. Piensa qué métodos debe añadir la clase que no están en la interface.
2. Implementa la clase ArrayDoubleEstadistica, con un array (no ArrayList) de double, que implemente la interface *Estadisticable*.
3. En la clase principal, define un objeto de cada una de las clases anteriores, añade valores a sus listas y utiliza los métodos de la interface que implementan para mostrar los respectivos valores mínimos, máximos y la media de sus elementos.

EJERCICIO 5: FIGURAS

Implementa una **interface** llamada **iFigura2D** que declare los métodos:

- double perimetro(): Para devolver el perímetro de la figura
- double area(): Para devolver el área de la figura
- void escalar(double escala): Para escalar la figura (aumentar o disminuir su tamaño). Solo hay que multiplicar los atributos de la figura por la escala (> 0).
- void imprimir(): Para mostrar la información de la figura (atributos, perímetro y área) en una sola línea.

Existen 4 tipos de figuras.

- **Cuadrado**: Sus cuatro lados son iguales.
- **Rectángulo**: Tiene ancho y alto.
- **Triángulo**: Tiene ancho y alto.
- **Círculo**: Tiene radio.

Crea las 4 clases de figuras de modo que implementen la *interface* iFigura2D. Define sus métodos.

Crea una clase ProgramaFiguras con un main en el que realizar las siguientes pruebas:

- Crea un ArrayList figuras.
- Añade figuras de varios tipos.
- Muestra la información de todas las figuras.
- Escala todas las figuras con escala = 2.
- Muestra de nuevo la información de todas las figuras.

- Escala todas las figuras con escala = 0.1.
- Muestra de nuevo la información de todas las figuras.

EJERCICIO 6: COMPARACIONES

Si queremos que los objetos de una clase se puedan comparar, esa clase debería definir los métodos *esMayor*, *esMenor* y *esIgual*. En lugar de implementarlos en esa clase, el objetivo es definirlos en una interfaz y hacer que cada clase que necesite poder comparar sus objetos implemente esa interfaz:

1. Utiliza la interfaz de la API de Java: **Comparable** para comparar los objetos de la clase Departamento. Por tanto, la clase Departamento implementará la interfaz *Comparable*. Habrá que implementar el método *compareTo* haciendo que un Departamento sea mayor que otro si tiene más empleados. En caso de igualdad, será mayor quien tenga el nombre mayor que el otro (habrá que llamar al *compareTo* de la clase *String*).
2. Sobrescribe el método ***equals*** de la clase *Object* para indicar que dos departamentos son iguales si tienen el mismo número de empleados e igual nombre.
3. Sobrescribe el método **toString** para mostrar los datos del departamento, por ejemplo:

```
Departamento { nombre = 'Ventas', numEmpleados = 10}
```

En el programa principal crear un ArrayList de departamentos y pon algunos. Muestra el ArrayList, ordénalo y vuelve a mostrarlo.

2. Bibliografía

Ejercicios prácticos: Unidad 8 Programación Orientada a Objetos I Ejercicios (CEEDCV) de Lionel Tarazón.

Ejercicios de Espe Micó i Joan Gerard Camarena (IES Jaume II el Just, Tavernes de la Valldigna).