TP 2 – Programación II

1 - La herencia es un tipo de jerarquía de abstracciones en la que existen una serie de

subclases (abstracciones especializadas) que van a heredar características de una o varias superclases (abstracciones generalizadas).

El objetivo fundamental de la herencia es, como el de la POO en general, la economía

de expresión (reutilización).

Entre las ventajas de la herencia múltiple tenemos que es un mecanismo muy potente

que nos proporciona mucha más flexibilidad.

2 - la herencia se representa mediante una flecha con punta de triángulo vacío que apunta desde la subclase hacia la superclase.

3 - Es recomendable utilizar la herencia cuando se tiene una relación entre dos clases (clase hija – clase padre) y se desea compartir códigos o atributos entre ambos.

4 - Los interfaces son clases especiales que pueden contener nombres de funciones, pero nunca sus implementaciones.

5 - Se representa al igual que una clase en un rectángulo, con el nombre de la interface en cursiva y entre signos de mayor y menor (<<Interface>>).

6 - Las interfaces en Java son útiles cuando necesitas establecer contratos comunes, lograr la abstracción, admitir la herencia múltiple, implementar el polimorfismo o permitir diferentes implementaciones de métodos comunes.

7 - Mientras que una interfaz define un contrato que las clases deben, una clase abstracta proporciona una implementación parcial y puede contener tanto métodos abstractos como concretos. La elección entre una interfaz y una clase abstracta depende de los requisitos específicos de diseño y del grado de implementación común que necesites compartir entre las subclases.

8 - Clases y objetos serán para nosotros conceptos similares a tipos y variables. Es decir, la relación que existe entre un objeto y la clase a la que pertenece (la clase de la que es una instancia) es análoga a la que hay entre una variable y el tipo del que es declarada.

9 / 10 -

Asociación:

Representación: Se muestra como una línea sólida entre las clases involucradas, con posiblemente un símbolo de multiplicidad en cada extremo para indicar el número de instancias relacionadas.

Se utiliza cuando dos clases están relacionadas de manera general, pero ninguna clase es parte de la otra.

Composición:

Representación: Se muestra como una línea sólida con un rombo lleno en el extremo de la clase que contiene la otra clase.

Se utiliza cuando una clase es parte de otra clase y no tiene sentido existir independientemente de esa clase

Agregación:

Representación: Similar a la composición, pero el rombo en el extremo de la clase que contiene está abierto en lugar de lleno.

Se utiliza cuando una clase es parte de otra clase, pero puede existir independientemente. La diferencia principal entre composición y agregación es que en la agregación, la parte puede existir sin la clase principal.

Herencia:

Representación: Se muestra como una línea sólida con una flecha sólida en el extremo de la clase derivada apuntando a la clase base.

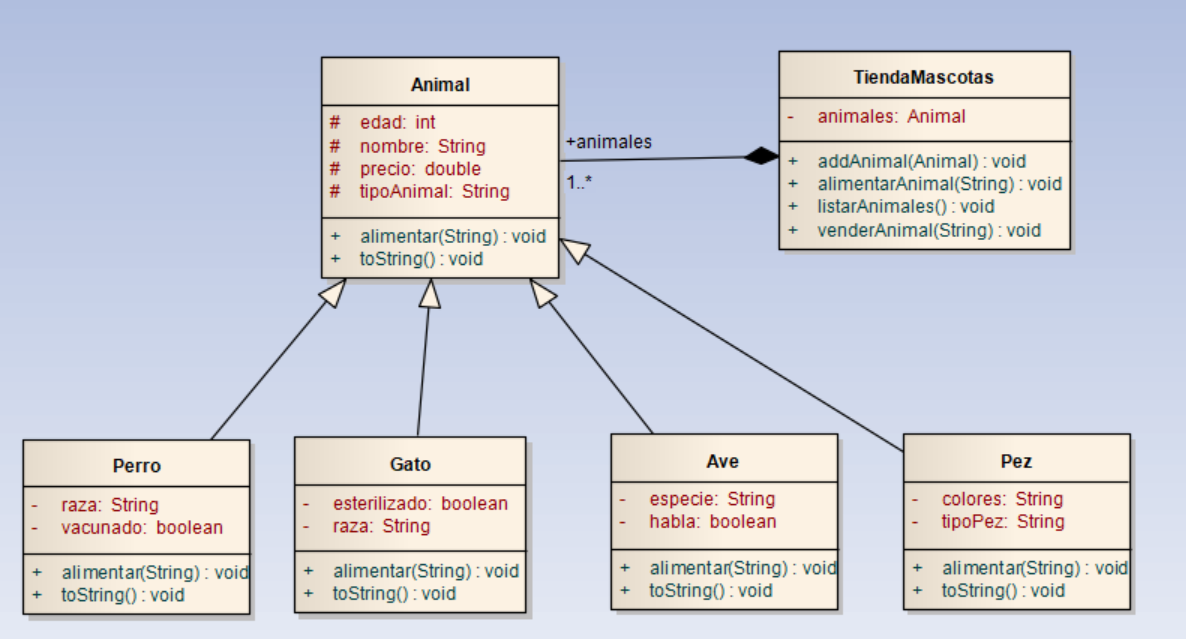
Se utiliza cuando una clase comparte características comunes con otra clase y puede extender o especializar su funcionalidad.

Implementación:

Representación: Se muestra como una línea punteada con una flecha sólida en el extremo de la clase que implementa la interfaz.

Se utiliza cuando una clase implementa una interfaz, es decir, proporciona una implementación para los métodos definidos en esa interfaz.

UML Ejercicio 1



UML Ejercicio 2

