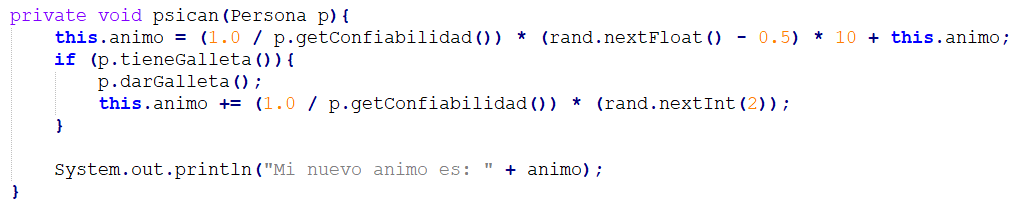
Laboratorio #1

**Objetivos que se buscan del estudiante:**

* Crea objetos e invoca métodos para modelar un sistema.
* Implementa de un programa principal que cumpla con un objetivo.
* Aplica correctamente los modificadores de visibilidad para encapsular sus clases.
* Presenta su diseño en un diagrama de clases UML básico.

**Introducción**

Usted trabaja en una perrera. Much@s voluntari@s se acercan a la perrera para ofrecer su tiempo y cuidar a l@s animales. Sin embargo, ha habido muchas quejas de voluntari@s mordid@s por perros. Esforzándose por evitar estos accidentes y enseñar al público a tratar con perr@s desconocidos, usted y su equipo deciden implementar un entrenador de voluntari@s.

Su veterinari@ residente sale con un(a) programador(a), quien les ha provisto de un método pre-programado que simula ciertos aspectos de la psicología canina habitual en un perro callejero:

Les han advertido que, para que el método funcione, es necesario:

* Que el perro tenga una propiedad de tipo Random llamada rand, y que debe inicializarse al instanciar al perro, así: rand = new Random();
* Que el perro importe, al inicio del archivo, la clase Random del paquete util de Java, usando la siguiente instrucción: import java.util.Random;

Como su vet no sabe nada sobre programación, no puede proveer más información sobre estas instrucciones. Investigue con un par de búsquedas en Google, si necesita más información.

Usted debe modelar e implementar el resto del sistema. El sistema debe representar a un perro que presenta tres comportamientos: ladrar, morder y mover la cola. El perro debe poseer como propiedad numérica un estado de ánimo, inicialmente con valor entre 5 y -5. También debe poseer un umbral numérico con valor negativo entre 0 y -20, que determina si el ánimo del perro es suficientemente malo para morder. Ambos deben ser inicializados al azar al instanciarse el perro.

**Nota**:

La clase Random provee un método llamado nextFloat() que devuelve un número pseudoaleatorio entre 0.0 y 1.0. Si queremos que nuestro número sea aleatorio entre -5 y 5, ¿cómo le hacemos? Complete la fórmula siguiente reemplazando las variables x e y por los valores necesarios:

float miNumero = rand.nextFloat() \* x – y;

Podemos poner a una instancia de Perro a interactuar con una persona. Cada vez que el perro interactúa, ejecuta, al principio, su método de simulación de psicología canina para determinar y asignar un nuevo valor a su estado de ánimo. Luego revisa su estado de ánimo resultante y decide:

* Morder si su estado de ánimo está por debajo de su umbral establecido.
* Ladrar si su estado de ánimo es negativo, pero sobre el umbral.
* Mover la cola si su estado de ánimo es positivo.

Todas estas acciones deben desplegar un mensaje en pantalla que describan la acción y luego deben provocar una reacción en la persona con la que se interactúa.

El sistema también debe representar a una persona. Como propiedades, la persona debe tener cierta confiabilidad (numérica) que es inicializada con un valor aleatorio entre 0 y 10 al instanciar una persona. Observe que, para que el valor sea aleatorio, el archivo Persona también necesita java.util.Random. La persona tiene la característica de poseer, o no, una galleta. Cuando la persona desea mejorar el ánimo del perro saca una galleta, lo cual altera correspondientemente el valor de su propiedad galletera. Claro que, luego de interactuar, la persona se queda sin galleta (porque se la da al perro).

Por último, la persona tiene un comportamiento de reacción que varía según la interacción que haya tenido con el perro. Elija una reacción para cada acción del perro y haga que se despliegue como un mensaje de texto en la pantalla.

Finalmente, su sistema debe proveer un *driver program* con una instancia de perro y una de persona, y debe presentar un menú que ofrezca las siguientes opciones:

1. Nueva persona: reemplaza la instancia actual que representa al (la) voluntari@ en el sistema. El objetivo es cambiar la confiabilidad de la persona para cambiar las pruebas.
2. Nuevo perro: reemplaza la instancia actual que representa al can simulado. El objetivo es cambiar el umbral y ánimo inicial del perro para cambiar las pruebas.
3. Sacar galleta: hace que la persona saque una galleta.
4. Interactuar: hace que el perro interactúe con la persona.
5. Salir: muestra la cantidad de interacciones y de estas cuántas veces ha mordido un perro, luego un mensaje de despedida y termina el programa.

Un@ de sus compañer@s de equipo le ha provisto de un *driver program* parcial. Ya presenta la funcionalidad de menú, pero le falta que las opciones realicen las acciones requeridas y le faltan comentarios. Complete el *driver program* provisto y agréguele la documentación necesaria.

**Ejercicios**

Ejercicio 1: responda a las siguientes preguntas.

* ¿Qué propiedades y métodos tendrá cada clase?
* ¿Qué tipo deben tener las propiedades y métodos de cada clase?
* ¿Cuáles deben ser los modificadores de visibilidad de los miembros en cada clase?
* ¿Qué parámetros serán requeridos por los métodos en sus clases?
* ¿Cómo proveerá de valores iniciales a sus objetos? ¿Qué valores iniciales les asignará?

Ejercicio 2: diseñe clases para Perro y Persona. Considere los detalles especificados y agregue o modifique lo que necesite. Desarrolle un diagrama de clases que muestre los miembros de cada clase y que establezca una asociación entre ellas. Incluya en su diagrama al *driver program*. Use anotaciones (recuadros de texto) en su diagrama para aclarar cualquier información que considere necesaria.

Ejercicio 3: programe las clases Perro y Persona. Complete el *driver program* provisto para que use las clases programadas y cumpla con los requerimientos funcionales del ejercicio.

**Material a entregar en Canvas**

* Archivo .pdf con las respuestas al ejercicio 1.
* Archivo .jpg o .png que presente el diagrama de clases requerido.
* Archivos **.java** con el código que implementa las clases y el *driver program* requerido.

**Evaluación**

* **(30 puntos) Ejercicio 1. Sus respuestas presentan un diseño coherente y funcional, siguiendo los principios de encapsulación.**
* **(30 puntos) Ejercicio 2. Su diagrama usa la sintaxis UML adecuada y refleja lo respondido en el ejercicio 1.**
* **(30 puntos) Ejercicio 3. El *driver program* y las clases compilan sin errores y el *driver program* cumple con los objetivos establecidos. Las clases funcionan como esperado.**
* **(10 puntos) El código presenta comentarios, encabezados y tabulación adecuada.**