Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2021-2

2 de Septiembre de 2021
Actividad Formativa

Actividad Formativa 1

Programación Orientada a Objetos I

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AF1/`
- **Hora del *push*:** 16:40

Importante: Antes de comenzar, comprueba que Git este funcionando correctamente en tu repositorio privado. Para esto, **sube los archivos base de la actividad de inmediato** (*add*, *commit*, *push*). Se espera que en esta actividad (así como en las demás actividades y tareas) utilices Git a lo largo de **todo tu desarrollo** como una herramienta, no sólo como un método de entrega. Es por esto que recomendamos enfáticamente que vayas subiendo tus cambios constantemente (***push***), ya que **problemas de último minuto** relacionados con la entrega y Git **no serán considerados**.

Introducción

¡Ding dong! Suena el timbre del nuevo Hotel para Mascotas del DCC. Los ayudantes de Programación Avanzada han tenido que reunirse para preparar la Tarea 1 del curso, por lo que en su ausencia te piden que cuides a sus mascotas virtuales.

¡Eres el nuevo **DCCuidador** de mascotas! Deberás encargarte de atender adecuadamente las necesidades específicas de cada uno de los animalitos que queden a tu cuidado.

Figura 1: Logo

Flujo del programa

El programa consiste en una simulación del Hotel, partiendo en un **Menú de Inicio**, que dará la opción de ingresar al juego. Al seleccionar la opción *Ingresar*, se instanciará un objeto de la clase **Hotel**, la entidad principal del programa que está a cargo de controlar la simulación. Con lo anterior se cargan los datos de las mascotas desde una base de datos almacenada en el archivo `mascotas.csv`, que podrán ser un **perro**, **gato** o **conejo**. Con dicha información se generan mascotas y se escoge un grupo aleatorio que serán los huéspedes iniciales del Hotel, y se mostrarán desde el **Menú de Mascotas**.

Seleccionar una mascota en el **Menú de Mascotas** dirige hacia el **Menú Cuidados**, donde puedes realizar distintas acciones sobre esta mascota para mejorar sus atributos de **saciedad**, y **entretención**, y así en conjunto aumentar su **satisfacción** que depende de una combinación de los ambos atributos. Seleccionar una opción en el **Menú Cuidados** hará que el Hotel llame al método respectivo de la mascota.

El Hotel podrá realizar todas las acciones que quiera sobre las mascotas huéspedes, siempre y cuando como cuidador tengas energía suficiente para hacerlo. Cuando la energía no te alcance para realizar alguna acción, se imprimirá un mensaje que lo indique y se deberá seleccionar la opción *Pasar de día* en el **Menú Cuidados** para continuar con el día siguiente de la simulación. Iniciar un nuevo día recargará la energía del cuidador, y además, las mascotas cuya satisfacción esté muy baja abandonarán el establecimiento. También podrán aparecer nuevas mascotas que se hospeden en el Hotel, y todos los huéspedes actuales perderán una cantidad aleatoria de sus atributos de saciedad y entretención.

La simulación termina si tres o más mascotas abandonan el Hotel en un mismo día, o si la cantidad de mascotas huéspedes es igual o menor a tres. ¿Cuántos días puedes mantener funcionando el Hotel?

Archivos

Archivo de datos

- `mascotas.csv`: En este archivo encontrarás los datos de todas las mascotas que pueden llegar a ser clientes del Hotel. La primera línea del archivo corresponde a los encabezados de la información que se está dando, mientras que las líneas siguientes contienen las características de una mascota, separadas por coma, de la forma:

  `Nombre,Especie,Raza,Dueño,Saciedad,Entretención`

  donde los valores se interpretan de esta manera:

`Nombre`	corresponde al nombre de la mascota
`Especie`	corresponde a la especie de la mascota (perro, gato o conejo)
`Raza`	corresponde a la raza de la mascota
`Dueño`	corresponde al nombre del dueño de la mascota
`Saciedad`, `Entretención`	valores entre 0 y 100 que indican los niveles de cada característica de la mascota al momento de ingresar al hotel, donde 100 es el máximo (y un buen estado) mientras que 0 es el mínimo (mal estado).

  Puedes suponer que los datos del archivo `mascotas.csv` están siempre correctos. No es necesario hacer verificaciones adicionales.

Archivos de código

En el directorio de la actividad encontrarás los siguientes archivos con código:

- `cargar_datos.py`: Aquí encontrarás la función relacionada con el cargado de datos de las mascotas, de nombre `cargar_mascotas`. **Debes modificarlo**
- `mascota.py`: Este archivo contiene la clase `Mascota` con sus clases hijas correspondientes a `Perro`, `Gato` y `Conejo`. **Debes modificarlo**
- `hotel.py`: Aquí encontrarás la clase `Hotel`. **Debes modificarlo**
- `comida.py`: Aquí encontrarás la clase `Comida`. **No debes modificarlo**
- `main.py`: Este es el archivo principal. Aquí se encuentran las clases que controlan el flujo del programa. Debes correrlo para iniciar la simulación, y **te ayudará a probar tu código.** **No debes modificarlo**
- `parametros.py`: Este archivo contiene parámetros para la ejecución del programa. Se importa en `main.py`, `mascota.py` y `hotel.py` y a través de él puedes acceder a los valores de estos parámetros. **No debes modificarlo**

Parte I: Cargando datos

En esta primera parte deberás trabajar en `cargar_datos.py` para definir una función que se encargue de obtener los datos desde `mascotas.csv` y crear instancias de las diferentes clases de mascotas, es decir, si la línea del archivo corresponde a la información de un perro, se deberá instanciar un objeto de la clase `Perro` con sus atributos.

- `def cargar_mascotas(ruta)`: Recibe la ruta a un archivo `.csv` y retorna una lista con todas las instancias de mascotas. **Debes modificarlo**

  La lista que retorna debe quedar de la siguiente manera:

```
lista_mascotas = [mascota_1, mascota_2, mascota_3, ...]
```

donde cada objeto `mascota` es una instancia de `Perro`, `Gato`, ó `Conejo`.

Una vez completada esta parte, puedes ejecutar `main.py` y no debe producir errores.

Parte II: Modelando entidades

Antes de poder comenzar con la simulación del Hotel, es importante definir las entidades que formarán parte de ella. Estas son representadas por medio de las clases en `mascota.py` y `comida.py`, las que deberás completar en base a los siguientes requerimientos:

- `class Mascota`: Representa a una mascota. Incluye los siguientes métodos: **Debes modificarlo**
  - `def __init__(self, nombre, raza, dueno, saciedad, entretencion)`: Este es el inicializador de la clase, y debe asignar los siguientes atributos: **No debes modificarlo**

`self.nombre`	Un `str` que representa el nombre de la mascota.
`self.raza`	Un `str` que representa la raza de la mascota.
`self.dueno`	Un `str` que representa el nombre del dueño de la mascota.
`self._saciedad`	Un `int` que representa que tan satisfecha se encuentra la mascota, es importante que no pueda ser negativo y tampoco más grande que 100. **Debe implementarse como una property**
`self._entretencion`	Un `int` que representa cuan entretenida se encuentra la mascota, es importante que no pueda ser negativo y tampoco más grande que 100. **Debe implementarse como una property**.

- `def satisfaccion(self)`: *Property* que calcula la satisfacción de la mascota, como una ponderación de sus atributos `self._saciedad` y `self._entretencion`. No debes modificarlo
  De la siguiente manera:

$$satisfaccion = \lfloor saciedad/2 \rfloor + \lfloor entretencion/2 \rfloor$$

- `def comer(self, comida)`: Esta función se llamará cada vez que quieras alimentar a la mascota. Recibe una instancia de `Comida` y deberás hacer uso de `random.random()` para calcular si la comida está vencida. Para esto, deberás comparar el resultado del `random()` con el atributo `probabilidad_vencer` del objeto `comida`. Si el resultado de `random()` es menor, entonces la comida ha vencido y deberás restar el valor de calorías de la comida a la saciedad de la mascota. En caso que la comida no esté vencida, deberás agregar el valor de calorías de la comida a la saciedad de la mascota. En cada caso, debes imprimir un mensaje indicando lo que pasó. Por ejemplo:

```
print(f"La comida estaba vencida! A {self.nombre} le duele la pancita :(")
print(f"{self.nombre} está comiendo {comida.nombre}, que rico!")
```

  Debes modificarlo

- `def pasear(self)`: Esta función se llamará cada vez que quieras pasear a la mascota. Ya viene implementada. No debes modificarlo

- `def __str__(self)`: Este método se llamará cada vez que se imprima en pantalla una instancia del objeto. Puedes elegir cómo mostrar los valores, pero al menos deben contener `nombre` y la `satisfacción` de la mascota. Por ejemplo:

```
Nombre: Copito de Nieve
Saciedad: 30
Entretención: 90
Satisfacción: 61
```

  Debes modificarlo

- `class Perro`: Representa la mascota de especie **Perro**. Deberás completarla de manera que herede de la clase `Mascota` y además definir dentro de ella un atributo `self.especie = "PERRO"` y un método `saludo()` que imprima un saludo específico para su especie (por ejemplo: `"guau guau"`). Debes modificarlo

- `class Gato`: Representa la mascota de especie **Gato**. Deberás completarla de manera que herede de la clase `Mascota` y además definir dentro de ella un atributo `self.especie = "GATO"` y un método `saludo()` que imprima un saludo específico para su especie (por ejemplo: `"miau miau"`). Debes modificarlo

- `class Conejo`: Representa la mascota de especie **Conejo**. Deberás completarla de manera que herede de la clase `Mascota` y además definir dentro de ella un atributo `self.especie = "CONEJO"` y un método `saludo()` que imprima un saludo específico para su especie (por ejemplo: `"chillidos"`). Debes modificarlo

Dentro del archivo `comida.py` se encuentra:

- `class Comida`: Representa el alimento que se le podrá dar a las mascotas dentro del Hotel. Contiene los siguientes métodos: No debes modificarlo
  - `def __init__(self, nombre, calorias, probabilidad_vencer)`: Inicializador de la Clase, contiene los siguientes atributos:

`self.nombre`	Un `str` con el nombre del alimento.
`self.calorias`	Un `int` con las calorías que se le sumarán o restarán a la saciedad de la mascota cuando se le dé comida.
`self.probabilidad_vencer`	Un `float` con la probabilidad de que la comida esté vencida.

  - `def __str__(self)`: Este método se llamará cuando se imprima una instancia de comida.

Dentro del archivo `hotel.py` se encuentra:

- `class Hotel`: Representa la entidad que administra el cuidado de las mascotas. Contiene los métodos:
  - `def __init__(self)`: Inicializador de la clase, contiene los siguientes atributos: No debes modificarlo

`self.__energia`	Un `int` que representa la cantidad de energía actual del cuidador. Es importante que cuando se modifique la energía el nuevo valor sea siempre positivo y menor o igual a `self.max_energia`. **Debe implementarse como una property**.
`self.__dias`	Un `int` que funciona como contador del progreso de la simulación. Debe verificar que se mantenga siempre positivo y con un incremento ascendente. **Debe implementarse como una property**.
`self.max_energia`	Un `int` que representa la energía del cuidador al inicio de cada día.
`self.mascotas:`	Una `list` que contiene instancias de las clases `Perro`, `Gato` y `Conejo`.
`self.funcionando:`	Un `bool` que indica si la simulación está en curso o se ha acabado.
`self.comidas`	Una `list` que contiene instancias de la clase `Comida`.

  - `def hotel_en_buen_estado(self)`: Se llama cada vez que pasa un día, para verificar si se cumplen las condiciones de término de la simulación. No debes modificarlo
  - `def imprimir_estado(self, nuevo_valor)`: Imprime en consola información importante del estado del Hotel. Debe imprimir algo como: Debes modificarlo

```
Día: 1
Energía cuidador: 50 / 100
Mascotas hospedadas: 7
```

  - `def recibir_mascota(self, mascota)`: Este método agrega un objeto de cualquiera de las subclases de `Mascota` a la lista `self.mascotas`, e imprime alguna de sus características en pantalla. No debes modificarlo
  - `def despedir_mascota(self, mascota)`: Contrario al método anterior, este remueve una instancia de `Mascota` desde `self.mascotas`, e imprime un mensaje en consola informándolo.

No debes modificarlo

- `def imprimir_mascotas(self)`: Este método invoca el método `__str__` de cada una de las instancias de `Mascota` en `self.mascotas`. No debes modificarlo
- `def nuevo_dia(self)`: Este método simula el paso de un día. Primero debe verificar las condiciones de término llamando a `self.hotel_en_buen_estado()`; si está en buen estado se imprime que comienza un nuevo día y se aumenta `self.__dias` en una unidad, se actualiza `self.__energia` a `self.max_energia` y además se disminuyen los atributos `entretencion` y `saciedad` de cada mascota en `self.mascotas` de forma aleatoria[1].
  De lo contrario, se modifica el valor de `self.funcionando` a `False` y se imprime un mensaje indicando que la simulación ha finalizado junto con los días transcurridos. Debes modificarlo
- `def revisar_energia(self)`: Este método revisa si existe suficiente energía para realizar cuidados sobre la mascota. No debes modificarlo
- `def pasear_mascota(self, mascota)`: Este método invoca el método `pasear()` de una `mascota` en `self.mascotas` y resta `COSTO_ENERGIA_PASEAR`, de `parametros.py`, a `self.__energia`. No debes modificarlo
- `def alimentar_mascota(self, mascota)`: Este método invoca el método `comer()` de `mascota`, y resta `COSTO_ENERGIA_ALIMENTAR`, de `parametros.py`, a `self.__energia`. Debes modificarlo

Simulación

Una vez definidas las entidades, puedes ejecutar el archivo `main.py`. En este se ejecuta todo el código que desarrollaste, y te permitirá dar vida al Hotel para mascotas mediante una simulación. No hay nada más que desarrollar aquí y puedes usarlo para probar que tu código funciona correctamente.

Notas

- Recuerda que la ubicación de tu entrega es en tu **repositorio personal**. Verifica que no estés trabajando en el **Syllabus**.
- Se recomienda completar la actividad en el orden del enunciado.
- Para las *properties* pueden crear nuevos métodos o modificar los existentes si creen que es necesario.
- Si aparece un error inesperado, ¡léelo! Intenta interpretarlo.
- Siéntete libre de agregar nuevos `print` en cualquier lugar de tu código para encontrar errores. Es una herramienta muy útil.

Objetivos de la actividad

- (0.5 pts) Implementar correctamente la función `cargar_mascotas` del archivo `cargar_datos.py`.
- (1.0 pts) Implementar correctamente la herencia y properties de las clases `Perro`, `Gato` y `Conejo` en `mascotas.py`.

[1] Para disminuir los atributos de forma aleatoria puedes ocupar randint() entre un rango de números que estimes conveniente.