Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2022-2

01 de Septiembre de 2022
Actividad Sumativa

Actividad Sumativa 1

Programación Orientada a Objetos I y II

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AS1/`
- **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

Tras años estudiando los conceptos de herencia y polimorfismo, y cómo se pueden aplicar a la vida real, el científico loco Hernán encontró una forma de crear dinosaurios. Este descubrimiento captó de inmediato la atención del prestigioso DCCentral Zoo, conocido por la preocupación de su staff respecto a la felicidad y el cuidado de sus animales.

Figura 1: Científico Hernan en el DCCentral Zoo

El DCCentral Zoo cuenta con diversas atracciones que reciben visitantes a diario. Para evaluar la factibilidad de incorporar animales híbridos y abrir nuevas atracciones han contratado a un equipo de expertos en distintas áreas. Tú, como estudiante de IIC2233, has recibido la misión de crear una simulación para ver cómo se desarrollaría el zoológico. ¡Este proyecto depende de ti!.

Flujo del programa

Al ejecutar el archivo `main.py` se instancia la clase `DCCentralZoo`, encargada de llevar a cabo la simulación, y en la que se instanciaran las diferentes clases que heredan de `Atraccion`. Cada instancia de la clase `Atraccion` es la que se encargará de crear los diferentes animales del zoológico. Finalmente, se utiliza el método `empezar()` del DCCentralZoo, que en primer lugar creará Atracciones y Animales de manera aleatoria para después comenzar a generar los distintos eventos y simular el paso de las semanas.

Importante: Dado que hay varias clases en el programa, **te recomendamos que leas todo el enunciado antes de comenzar a programar.** Además, **se incluye un diagrama de clases en la sección Anexo al final del enunciado**, que puede ayudarte a entender las relaciones entre ellas.

Archivos

En el directorio de la actividad encontrarás los siguientes archivos:

Código

- No modificar `main.py`: Este es el archivo principal del programa. Puedes ejecutarlo para probar la simulación.
- Modificar `atracciones.py`: Contiene la clase abstracta `Atraccion` y las clases que heredan de ella; `GranjaHerbivoros` y `PaseoCarnivoros`.
- Modificar `fauna.py`: Contiene la clase abstracta `Animal` y las clases que heredan de ella; `Carnivoro`, `Herbivoro` y `Omnivoro`.
- Modificar `parque.py`: Contiene la clase `DCCentralZoo`.
- No modificar `parametros.py`: Contiene los parametros necesarios para el funcionamiento del programa.

Datos

Para la simulación se hace uso del archivo `especimenes.csv`. En este archivo encontraras información de los animales del DCCentral Zoo. La primera linea del archivo corresponde al nombre de las columnas y el resto de las lineas contienen los atributos de los animales, separados por comas, de la siguiente forma:

```
Tipo,Especie
```

NO debes modificar este archivo. La función para cargarlo ya se encuentra lista.

Parte 1: Modelación de la Fauna 🐒

En esta parte deberás completar las clases `Animal`, `Carnivoro`, `Herbivoro` y `Omnivoro` del archivo `fauna.py`. Deberás trabajar con una clase abstracta y definir correctamente la herencia, métodos y atributos de las demás clases. Recuerda que en el anexo del enunciado hay un diagrama de clases que puedes usar de apoyo.

Clase Animal

- Modificar `class Animal`: Es una clase **abstracta** que modela a los animales del DCCentral Zoo. De esta clase heredan las subclases `Carnivoro`, `Herbivoro` y `Omnivoro`.

- Modificar `def __init__(self, especie: str, **kwargs)`: Inicializador de la clase `Animal` que recibe la especie en formato `str` y se asigna al atributo `self.especie`. Además, deberás crear el atributo `self.ganancia_actual` con un valor inicial de 0.
- Modificar `def alimentarse(self)`: Es un método **abstracto**.
- Modificar `def exhibicion(self)`: Es un método **abstracto**.
- No modificar `def __str__(self) -> str`: Este método se encarga de retornar el animal en formato `str`, especificando la especie.

Clase Carnívoro

- Modificar `class Carnivoro`: Es una clase que hereda de `Animal`. Modela a los animales carnívoros del DCCentral Zoo.
  - Modificar `def __init__(self, ferocidad: int, **kwargs)`: Inicializador de la clase `Carnivoro`. Llama al inicializador de la superclase y le entrega los argumentos necesarios. Además, este inicializador recibe la ferocidad en formato `int` y lo asigna al atributo `self.ferocidad`. Finalmente ,suma el parámetro `GAN_CARNIVORO` al atributo `self.ganancia_actual`.
  - Modificar `def alimentarse(self)`: Método que representa al animal comiendo. Primero se llama al método de su superclase. Luego incrementa la ferocidad del animal por `INCREMENTO_FEROCIDAD` e imprime el siguiente mensaje:

    `f"Animal {self.especie} se come un kilogramo de Carne"`
  - Modificar `def exhibicion(self)`: . Primero se llama al método de su superclase. Luego aumenta el atributo `self.ganancia_actual` en:

$$\texttt{self.ferocidad} \times \texttt{ponderador} \tag{1}$$

    Donde ponderador es un numero entero aleatorio[1] entre `MIN_EX_CARNIVORO` y `MAX_EX_CARNIVORO`.

Clase Herbívoro

- Modificar `class Herbivoro`: Es una clase que hereda de `Animal`. Modela a los animales herbívoros del DCCentral Zoo.
  - Modificar `def __init__(self, especie: str, adorabilidad: int)`: Inicializador de la clase `Herbivoro`. Llama al inicializador de la superclase y le entrega los argumentos necesarios. Además, recibe la adorabilidad en formato `int` y la asigna al atributo `self.adorabilidad`. Finalmente, suma el parámetro `GAN_HERBIVORO` al atributo `self.ganancia_actual`.
  - Modificar `def alimentarse(self)`: Método que representa al animal comiendo. Primero se llama al método de su superclase. Luego incrementa la adorabilidad del animal por `INCREMENTO_ADORABILIDAD` e imprime el siguiente mensaje:

    `f"Animal {self.especie} se come un kilogramo de vegetales"`
  - Modificar `def exhibicion(self)`: Primero se llama al método de su superclase. Luego aumenta el atributo `self.ganancia_actual` en:

$$\texttt{self.adorabilidad} \times \texttt{ponderador} \tag{2}$$

    Donde ponderador es un número entero aleatorio entre `MIN_EX_HERBIVORO` y `MAX_EX_HERBIVORO`.

[1] Puedes utilizar la función `randint()` de la librería `random`

Clase Omnívoro

- Modificar `class Omnivoro`: Es una clase que hereda de las clases `Carnivoro` y `Herbivoro`. Los animales de esta clase tienen tanto adorabilidad como ferocidad, y comen de todo.
  - Modificar `def __init__(self, **kwargs)`: Inicializador de la clase `Omnivoro`. Recibe los argumentos en forma de `**kwargs` y se encarga de inicializar las superclases.
  - Modificar `def alimentarse(self)`: Método que representa al animal comiendo. Como un omnívoro puede comer carne y vegetales, este método debe llamar a los métodos `alimentarse` de las clases `Carnivoro` y `Herviboro`.
  - Modificar `def exhibicion(self)`: Cómo un omnívoro, su exhibición genera ganancias como si fuera un animal carnívoro y uno herbívoro. Por lo tanto, este método llama al método `exhibicion` de las clases `Carnivoro` y `Herviboro`.

Parte 2: Atracciones ✨

Clase Atracción

En esta parte deberás completar las clases `Atraccion`, `GranjaHerbivoros`, y `GranjaCarnivoros` del archivo `atracciones.py`. Deberás definir correctamente todos los métodos y trabajar con las *properties* que se solicitan.

- Modificar `class Atraccion`: Es una clase **abstracta** que modela las diferentes atracciones del `DCCentralZoo`. De esta clase heredan las subclases `GranjaHerviboros` y `PaseoCarnivoros`.
  - No modificar `def __init__(self, numero: int)`: Inicializador de la clase `Atraccion`. Recibe el número identificador asignado a la atracción y define un atributo `especies_disponibles`, el cual es un diccionario con las especies que puede crear. Este diccionario se cargará al correr el método `cargar_especies`.
  - No modificar `def cargar_especies(self, ruta_archivo: str)`: Lee el archivo `especimenes.csv` y carga a un diccionario las posibles especies que pueden ser creadas según su tipo (que serán las llaves del diccionario). Pueden ser carnivoro, herbivoro u omnivoro.
  - No modificar `def alimentar_animales(self)`: Itera por cada animal de la lista `self.animales` y llama al método `alimentarse()` de cada uno.
  - Modificar `def visitantes(self) -> int`: Es una **property** dinámica, que debe tener el siguiente comportamiento:
    - `getter`: Retorna un numero de visitantes aleatorios con `randint()`. Al llamar esta ultima función se le debe entregar como rango los elementos de la lista `VISITANTES` del modulo `parametros`.
  - Modificar `def recaudacion(self) -> int`: Es una **property** dinámica que debe tener el siguiente comportamiento:
    - `getter`: Retornara el dinero obtenido por la atracción en una semana, este se calcula de la siguiente manera:
      1. Crea una variable `dinero` para realizar las sumas.
      2. Itera por cada animal en la lista `self.animales`. En cada iteración primero llama al método `.alimentar()` del animal, para finalmente sumar el monto `animal.ganancia_actual` al dinero actual.

3. Luego de terminada la iteración el dinero se multiplica por el atributo `self.visitantes` y el parámetro `MULTIPLICADOR_RECAUDACION`
4. Luego calcula una probabilidad con el método `random()` del modulo `random`. Sí esta probabilidad es mayor que `PROBABILIDAD_EVENTO`, se debe sumar el monto entregado por el método `self.evento()` al dinero actual
5. Finalmente se retorna el dinero calculado

- Modificar `def crear_animales(self)`: Es un método **abstracto**.
- Modificar `def __str__(self)`: Es un método **abstracto**.
- Modificar `def evento(self)`: Es un método **abstracto**.

Clase Granja Herbívoros

- Modificar `class GranjaHerbivoros`: Es una clase que hereda de `Atraccion` y representa el lugar donde estarán los animales herbívoros y omnívoros.
  - Modificar `def __init__(self, *args, **kwargs)`: Método que llama al inicializador de la superclase con los argumentos necesarios.
  - No modificar `def crear_animales(self)`: Método que elige de forma aleatoria entre crear un animal herbívoro o un omnívoro. Luego de escoger, consulta las especies disponibles y crea una instancia de la clase correspondiente.
  - Modificar `def __str__(self) -> str`: Método que retorna una representación de la atracción en la forma:

    ```
    f"Granja de Herbivoros {self.numero}"
    ```

  - Modificar `def evento(self) -> str`: Primero imprime el siguiente mensaje:

    ```
    f"\nEVENTO {self}: AVISTAMIENTO DE BRACHIOSAURUS\n "
    ```

    Luego retorna la parámetro `EVENTO_HERBIVOROS`.

Clase Paseo Carnívoros

- Modificar `class PaseoCarnivoros`:
  - Modificar `def __init__(self, *args, **kwargs)`: Método que llama al inicializador de la superclase con los argumentos necesarios.
  - No modificar `def crear_animales(self)`: Método que elige de forma aleatoria entre crear un animal carnívoro o un omnívoro. Luego de escoger, consulta las especies disponibles y crea una instancia de la clase correspondiente.
  - Modificar `def __str__(self) -> str`: Método que retorna una representación de la atracción en la forma:

    ```
    f"Paseo de Carnivoros {self.numero}"
    ```

  - Modificar `def evento(self) -> str`: Primero imprime el siguiente mensaje:

    ```
    f"\nEVENTO {self}: SE ALIMENTARA AL TYRANOSAURUS\n "
    ```

    Luego retorna el parámetro `EVENTO_CARNIVORO`.

Parte 3: Modelación del DCCentral Zoo 🏞️

En esta parte deberás completar la clase `DCCentralZoo` ubicada en el archivo `zoologico.py`. Esta es la encargada de manejar, instanciar y acceder al resto de las clases.

Clase DCCentral Zoo

- Modificar `class DCCentralZoo`: Clase que representa al parque zoológico y lleva la simulación. Es la encargada de crear atracciones y hacer que estas creen animales. Además de llevar la cuenta de la recaudación y visitas.
  - No modificar `def __init__(self, semanas_maximas: int)`: Inicializador del zoológico. Recibe un `int` que representa las semanas máximas de la simulación. También posee atributos que permiten monitorear los visitantes semanales y totales, además de recaudación semanal y total. Uno de sus atributos es `recaudacion_construccion`, que representa la recaudación necesaria en una semana para poder construir una nueva atracción. Finalmente inicializa dos listas: `atracciones_herbivoros` y `atracciones_carnivoros` donde son guardadas las diferentes atracciones.
  - No modificar `def alimentar_animales(self)`: Debe iterar por los animales de cada atracción en los atributos `self.atracciones_herbivoros` y `self.atracciones_carnivoros`. Llama al método `alimentar_animales()` de cada animal.
  - Modificar `def calcular_estadisticas(self) -> tuple`: Debes iterar por cada atracción para obtener los valores de recaudación y visitantes. Calcula los valores totales de cada dato y los retorna como una tupla de la forma `(visitantes totales, recaudación total)`.
  - No modificar `def construir_atraccion(self, tipo: str)`: Recibe un tipo que puede ser `"herbivoros"` o `"carnivoros"` y según este instanciará una `GranjaHerbivoros` o un `PaseoCarnivoros`. Finalmente guarda la instancia creada en la lista correspondiente.
  - No modificar `def poblar_zoo(self)`: Debe iterar en el rango de los parámetros `N_PASEOS` para construir los `PaseoCarnivoros` y `N_GRANJAS` para construir los `GranjaHerbivoros`. En cada iteración se llama al método `self.construir_atraccion()` con sus argumento pertinente.
  - No modificar `def incubar_animales(self) -> int`: Itera por cada atracción e instancia un número arbitrario de animales. Finalmente retorna el número de nuevos animales creados.
  - No modificar `def reset(self)`: Suma la recaudación semanal a la total y los visitantes semanales a los totales. Fija en 0 todos los atributos semanales.
  - No modificar `def empezar(self)`: Método que maneja la simulación. Primero construye las atracciones iniciales. Después entra en el ciclo de iteraciones según el máximo de semanas. Se instancian los nuevos animales para luego calcular las estadísticas semanales. Se revisa si se supera la meta para crear una nueva atracción. Posteriormente se reinician los atributos semanales y pasa a la siguiente semana. Cuando este proceso termina muestra las estadísticas finales.

Notas

- La recolección de la actividad se hará en la rama principal (main) de tu repositorio.
- Recuerda verificar que **no** estés trabajando en el Syllabus. Debes hacer la entrega en tu repositorio personal.
- Se recomienda completar la actividad en el orden del enunciado.
- 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.
- Puedes probar tu código ejecutando el archivo `main.py`.
- Recuerda hacer `git pull` antes de comenzar a trabajar para asegurarte de que tu repositorio esté sincronizado con GitHub.

Requerimientos

- (2.50 pts) Modelación de Animales
  - (0.50 pts) Modelar correctamente la Clase Animal.
  - (0.75 pts) Modelar correctamente la Clase Carnivoro.
  - (0.75 pts) Modelar correctamente la Clase Herbivoro.
  - (0.50 pts) Modelar correctamente la Clase Omnivoro.
- (2.50 pts) Modelación de Atracciones
  - (1.50 pts) Modelar correctamente la Clase Atraccion.
  - (0.50 pts) Modelar correctamente la Clase GranjaHerbivoros.
  - (0.50 pts) Modelar correctamente la Clase PaseoCarnivoros.
- (1.0 pts) Modelación de Parque
  - (1.0 pts) Modelar correctamente la Clase DCCentralZoo.

Anexo

A continuación se adjunta el diagrama de clases de la actividad.

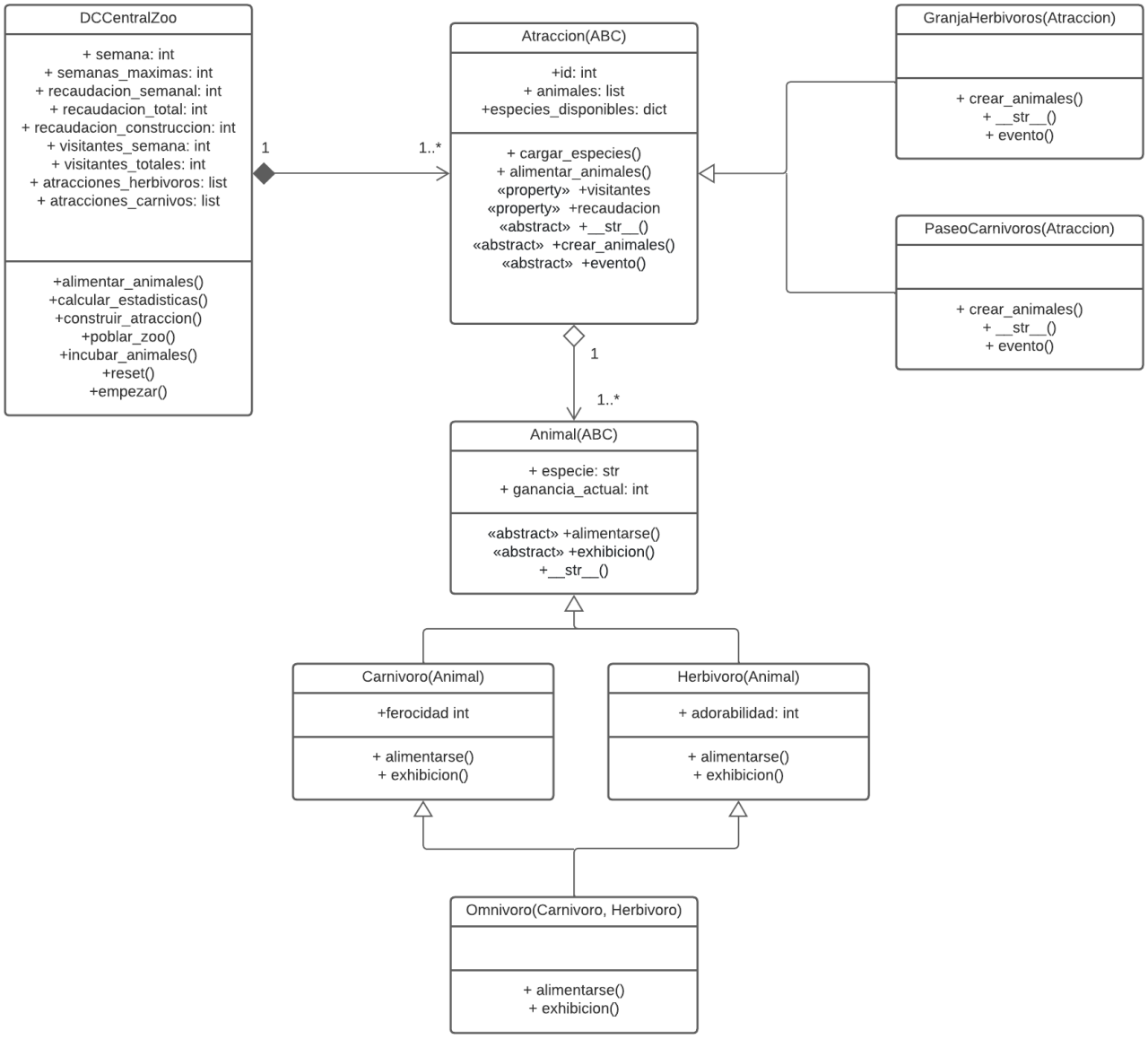

Figura 2: Diagrama de clases