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

9 de Septiembre de 2021
Actividad Sumativa

Actividad Sumativa 1

OOP 2

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

Con el gradual retorno a la presencialidad, comienzas a planificar las miles de actividades que harás con tus amigues de región una vez que vengan a Santiago, y como primera elección, decides que deberán ir al recién inaugurado parque de atracciones DCCoaster. Para convencer a todo el mundo que este es el primer lugar que deben visitar juntes, decides terminar de implementar una famosa simulación conocida como DCCoaster Tycoon, programa que simula el funcionamiento del parque de atracciones DCCoaster durante un día, para lo que va haciendo que todes les asistentes recorran el parque, subiéndose a sus atracciones favoritas.

Figura 1: Logo DCCoaster Tycoon

Flujo del programa

Al ejecutar el script `main.py`, este carga la información de las atracciones y personas utilizado las funciones `cargar_personas()` y `cargar_atracciones()` e inicializa las clases correspondientes. Luego, inicializa una instancia de `Parque` con las atracciones y personas cargadas anteriormente, junto con iniciar la simulación por horas del parque.

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 al final del enunciado**, que puede ayudarte a entender las relaciones entre ellas.

Archivos

Archivos de datos

- `asistentes.csv`: En este archivo se encuentran todos los datos de los asistentes al Parque.
- `atracciones.csv`: En este archivo se encuentran todos los datos de las atracciones del Parque.

Archivos de código

En el directorio de la actividad encontrarás los siguientes archivos

- `main.py`: Este es el archivo principal del programa. No debes modificarlo
- `parque.py`: Contiene la clase `Parque`. Debes modificarlo
- `atracciones.py`: Contiene la clase `Atraccion` y las que heredan de esta; `AtraccionFamiliar`, `AtraccionAdrenalinica`, `AtraccionAcuatica`, `AtraccionTerrorifica`, `CasaEmbrujada` y `MontanaAcuatica`. Debes modificarlo
- `personas.py`: Contiene la clase `Persona` y las que heredan de esta; `Adulto` y `Nino`. Debes modificarlo
- `cargar_datos.py`: En este archivo encontrarás todas las funciones encargadas de la lectura de archivo e instanciación de las personas y atracciones. No debes modificarlo
- `parametros.py`: Este archivo contiene los parámetros necesarios para el funcionamiento del programa. No debes modificarlo

Parte 1: Asistentes de DCCoaster

En esta parte, deberás completar las clases `Persona`, `Adulto` y `Nino`, las cuales se encuentran definidas en el archivo `personas.py`. Deberás definirlas correctamente en cuanto a sus atributos, métodos y relaciones. Recuerda que al final del enunciado hay un diagrama de clases que te puede ayudar a visualizar las relaciones.

Clase Persona

- `class Persona`: Es una clase **abstracta** que representa a las personas que visitan DCCoaster. Para crear una nueva clase de asistente, se deberá heredar de `Persona`, tal como lo hacen `Adulto` y `Nino`. Debe definirse como una clase **abstracta**. Debes modificarlo
  - `def __init__(self, nombre: str, edad: int, tiene_pase: bool, juegos: list):`

    No debes modificarlo

- `self.nombre`: En un `str` con el nombre de la persona.
- `self.edad`: Es un `int` con la edad de la persona.
- `self.tiene_pase`: Es un `bool` que es `True` en el caso que la persona tenga su pase y es `False` en el caso contrario.
- `self.juegos`: Es una `list` de `str`, contiendo el nombre de los juegos a los que se desea subir la persona en su visita al parque.
- `self.esperando`: Es un `bool` que es `True` cuando la persona está en la fila de un juego, y `False` en el caso contrario.
- `self.__salud`: Es un atributo privado, del tipo `int`, que representa la salud de la persona.
- `self.__felicidad`: Es un atributo privado, del tipo `int`, que indica el nivel de felicidad de la persona.

Es importante notar que el `__init__` de `Persona` tiene una llamada al método `definir_estados`, el cual tendrás que implementar según lo descrito más adelante.

- `def felicidad(self)`: Corresponde a la *property* de la felicidad de la persona. Este se encarga de verificar que `self.__felicidad` no pueda tomar valores menores a 0, en caso de que ocurra el atributo toma valor 0. No debes modificarlo
- `def salud(self)`: Corresponde a la *property* de la salud de la persona. Este se encarga de verificar que `self.__salud` no pueda tomar valores menores a 0, en caso de que ocurra el atributo toma valor 0. No debes modificarlo
- `def revision_juegos(self) -> bool:` Este método debe revisar si la persona está lista para ir al siguiente juego o no. Para esto, deberás revisar si la persona **no** está esperando y **aún le quedan juegos que quiera jugar**. En caso de que se cumplan estas dos condiciones, deberá **retornar** `True`, mientras que devolverá `False` en caso contrario. Debes modificarlo
- `def siguiente_juego(self) -> str:` este método debe **retornar** el nombre del siguiente juego al cual la persona quiere asistir. Para esto, debes devolver el **primer** elemento de la lista `self.juegos`, removiéndolo de esta misma Debes modificarlo
- `def actuar(self)`: Corresponde a un **método abstracto**, el cual debes dejar definido. Debes modificarlo
- `def definir_estados(self)`: Corresponde a un **método abstracto**, el cual debes dejar definido. Debes modificarlo

Clase Adulto

- `class Adulto` Esta clase hereda de `Persona` y representa a aquellas personas con una edad mayor a 13, consideradas adultas para el parque. Deberás completar la clase de tal forma que herede correctamente de `Persona`: Debes modificarlo
  - `def __init__(self, nombre: str, edad: int, tiene_pase: bool, juegos: list, dinero: int):` Para completar este método deberás enviar los atributos a la superclase. Además deberás definir el siguiente atributo: Debes modificarlo
    - `self.dinero`: Un `int` correspondiente al dinero que tiene el adulto para gastar.

    Es **muy importante** que tengas en cuenta que para **definir el estado** de un adulto, se necesita el valor de `self.dinero`.

- `def definir_estados(self)`: Este método se encarga de definir la salud y la felicidad inicial del `Adulto`. La salud se calcula como el número entero resultante de la multiplicación de la edad por un entero aleatorio entre 1 y 3. La felicidad se calcula como la multiplicación de la edad y el dinero. Este método **no retorna nada**. Debes modificarlo
- `def actuar(self)`: Se encarga de verificar el estado de salud y felicidad del Adulto, para desencadenar una acción especial en base a estos. No debes modificarlo

Clase Nino

- `class Nino`: Esta clase hereda de `Persona` y representa a aquellas personas con una edad menor o igual a 13. Deberás completar la clase de tal forma que herede correctamente de `Persona`: Debes modificarlo:
  - `def __init__(self, nombre: str, edad: int, tiene_pase: bool, juegos: list, padre: str)`: Para completar este método deberás enviar los atributos a la superclase. Además deberás definir el siguiente atributo: Debes modificarlo
    - `self.padre`: Un `str` correspondiente al nombre del padre.

    Es **muy importante** que tengas en cuenta que para **definir el estado** de un niño, se necesita el valor de `self.padre`.
  - `def definir_estados(self)`: Este método se encarga de definir la salud y la felicidad inicial del Nino. La salud se calcula como la edad multiplicada por un número entero aleatorio, en el rango [1, 5]. La felicidad se calcula como la multiplicación del largo del nombre del padre por 10. Este método **no retorna nada**. Debes modificarlo
  - `def actuar(self)`: Se encarga de verificar el estado de salud y felicidad del Adulto, para desencadenar una acción especial en base a estos. No debes modificarlo

Parte 2: Modelación de Atracciones

En esta parte, deberás completar las clases; `AtraccionFamiliar`, `AtraccionAdrenalinica`, `AtraccionAcuática` y `MontanaAcuatica`, las cuales se encuentran definidas en el archivo `atracciones.py`. Deberás definirlas correctamente en cuanto a sus atributos, métodos y relaciones. Recuerda que al final del enunciado hay un diagrama de clases que te puede ayudar a visualizar las relaciones.

Importante: Lee con atención el enunciado, ya que los métodos de las clases pueden verse similares, pero tienen pequeñas diferencias en los requerimientos a las que deberás prestar atención.

Clase Atraccion

- `class Atraccion`: Esta es la clase que define todas las atracciones que se encuentran dentro del **DCCoaster**. Representa a todas las atracciones, que deben heredar de esta clase. **Debes definirla como clase abstracta**. Debes modificarlo
  - `def __init__(self, nombre: str, capacidad: int)`: recibe el `nombre` de una `Atraccion` y su `capacidad`. Además, tiene el atributo `fila` correspondiente a una lista vacía. No debes modificarlo
    - `self.nombre`: Es un `str` con el nombre de la atracción
    - `self.capacidad_maxima`: Es un `int` con la cantidad de personas que pueden subir a una atracción.
    - `self.fila`: Es una lista de instancias de las clases `Nino` y/o `Adulto`.

- `def ingresar_persona(self, persona: Persona)`: Este método se encarga de ingresar a las personas a las respectivas atracciones. Recibe una instancia de la clase `Persona` y la añade a fila de la atracción. Luego de esto, se cambie el atributo `esperando` de la persona a `True` y se deberá imprimir un mensaje indicando que la `persona` (imprimiendo su nombre) ha entrado a la fila de la atracción. Por ejemplo: No debes modificarlo

```
"María José Hidalgo ha entrado a la fila de El Kraken"
```

- `def nueva_ronda(self)`: Simula una ronda de cada atracción. Verifica que la fila de la atracción no está vacía para así dejar ingresar a las personas de en su fila. No debes modificarlo
- `def iniciar_juego(self, personas: list)`: este método recibe una lista de objetos de clase `Persona` e itera sobre ellos, representando que jugaron en la `Atraccion`. Para cada persona, se cambiará su estado `esperando` a `False` y se llamará a `efecto_atraccion`. No debes modificarlo
- `def efecto_atracion(self, persona: Persona)`: corresponde a un método abstracto. Tendrás que ir implementándolo en la herencia con los efectos que hace una atracción a las personas que juegan en ella. No debes modificarlo

Clase AtraccionFamiliar

Tienes que implementar la totalidad de esta clase.

- `class AtraccionFamiliar`: Esta clase debe heredar `Atraccion`. Debes modificarlo
  - `def __init__(self, nombre: str, capacidad: int)`: `AtraccionFamiliar` tiene que mantener los mismos atributos que hereda de `Atraccion`, teniendo que recibir `nombre` y `capacidad` al instanciarse. Además de esto, deberás agregar los siguientes atributos: Debes modificarlo
    - `self.efecto_salud`: Es un `int` con el efecto en la salud de las personas al subirse a esta atracción. Su valor equivale a `SALUD_FAMILIA` ubicado en `parametros.py`
    - `self.efecto_felicidad`: Es un `int` con el efecto en la felicidad de las personas al subirse a esta atracción. Su valor equivale a `FELICIDAD_FAMILIA` ubicado en `parametros.py`
  - `def efecto_atraccion(self, persona: Persona)`: recibe a una persona. A la felicidad de la persona se le suma el efecto felicidad de la atracción y a la salud de la persona se le resta el efecto salud de la atracción. Debes modificarlo

Clase AtraccionAdrenalinica

Tienes que implementar la totalidad de esta clase.

- `class AtraccionAdrenalinica`: Esta clase debe heredar `Atraccion`. Debes modificarlo
  - `def __init__(self, nombre: str, capacidad: int, salud_necesaria: int)`: al instanciar `AtraccionAdrenalinica`, se deberá recibir los atributos `nombre` y `capacidad`, y entregárselos a la superclase, además de recibir un tercer parámetro `salud_necesaria`, correspondiente a un `int` que deberás guardar en un atributo de nombre `self.salud_necesaria`. Los atributos que deberás agregar son los siguientes:
    - `self.salud_necesaria`: Es un `int` con la salud necesaria para poder ingresar a esta atracción. El valor de este atributo viene a partir del tercer argumento entregado al instanciar la clase

- `self.efecto_salud`: Es un `int` con el efecto en la salud de las personas al subirse a esta atracción. Su valor equivale a `SALUD_ADRENALINA` ubicado en `parametros.py`
- `self.efecto_felicidad`: Es un `int` con el efecto en la felicidad de las personas al subirse a esta atracción. Su valor equivale a `FELICIDAD_ADRENALINA` ubicado en `parametros.py`

- `def efecto_atraccion(self, persona: Persona)`: Recibe una persona y existen dos posibles efectos. **Debes modificarlo**
  - Si la persona **no tiene la salud necesaria**, correspondiente al atributo `self.salud_necesaria`, se imprime un mensaje indicando que lo bajaron del juego. Luego a la salud de la persona se le suma la mitad del efecto salud de la atracción y a la felicidad de la persona se le resta la mitad del efecto felicidad de la atracción.
  - Si la persona cumple con el requisito de salud, se suma el efecto felicidad de la atracción a la felicidad de la persona y se resta el efecto salud de la atracción a la salud de la persona.

Clase AtraccionAcuatica

Tienes que implementar la totalidad de esta clase.

- `class AtraccionAcuatica`: Esta clase hereda de `AtraccionFamiliar`.
  - `def __init__(self, nombre: str, capacidad: int)`: esta clase deberá mantener el método `__init__` heredado de `AtraccionFamiliar`. Además, deberás cambiar el valor del efecto felicidad de esta clase por el valor `FELICIDAD_ACUATICA` definido en `parametros.py`
  - `def ingresar_persona(self, persona: Persona)`: Revisa que al ingresar a esta atracción, la instancia de `Persona` tenga su pase de movilidad. En caso de tenerlo, llama al método `ingresar_persona` de la superclase, de lo contrario, no la deja entrar. **Debes modificarlo**

Clase MontanaAcuatica

- `class MontanaAcuatica`: Esta clase hereda de la clase `AtraccionAdrenalinica` y `AtraccionAcuatica`. Deberás completar la clase tal que herede correctamente de sus superclases. Lo importante en esta clase, es que se debe mantener `efecto_felicidad` y `efecto_salud` de la clase `AtraccionAdrenalinica`, pero el método `ingresar_personas` de la clase `AtraccionAcuatica`.
  - `def __init__(self, nombre: str, capacidad: int, salud_necesaria: int, dificultad: int)`: Para completar este método deberás enviar los atributos a la superclase, y además agregar el siguiente atributo: **Debes modificarlo**
    - `self.dificultad`: es un `int` y corresponde a la dificultad del juego
  - `def iniciar_juego(self, personas: list)`: Este método recibe una lista de objetos de la clase `Persona` y sobreescribe el método de su clase padre.

    Debes reimplementar este método tal que recorra la lista personas y por cada elemento de la lista imprima un mensaje avisando que esta persona ha jugado en esta atracción. Luego, debes cambiar el valor del atributo **esperando** de la persona por `False`. Si la salud es menor o igual a la salud necesaria de la atracción multiplicada por el nivel de dificultad, la persona de caerá al agua y perderá su pase, por lo que el atributo respectivo a su pase cambiará a `False`.

    Finalmente, independiente si cumplió o no la condición antes descrita, se le debe aplicar el efecto de la atracción a la persona (utilizando el método que corresponda). **Debes modificarlo**

Parte 3: Parque

En esta parte, deberás completar la clase `Parque`, ubicada en el archivo `parque.py`. En específico, tendrás que implementar la distribución de las personas entre las distintas atracciones.

Clase Parque

- `class Parque`: modela el comportamiento del parque de atracciones durante un día, en pasos de una hora. Esta clase tiene una relación de agregación tanto con `Persona` como con `Atracción`, ya que esta recibe como parámetros múltiples objetos de ambas clases.
  - `def __init__(self, personas: list, atracciones: list)`: Este método recibe un diccionario con todas las atracciones y una lista con todas las personas que asistirán al parque en el día. No debes modificarlo
    - `self.personas`: Es una lista de objetos `Nino` y `Adulto`. Estas son todas las personas que asistirán al parque.
    - `self.atracciones`: Es un diccionario de objetos `Atraccion`, con formato `{"NombreAtraccion": Atraccion}` . Contiene todas las atracciones del parque.
  - `def distribuir_personas(self):` Este método debe recorrer la lista de personas que visitan el parque y por cada verificar si puede ingresar a su siguiente atracción (recuerda que hay un método de `Persona` para verificar esta condición y otro para obtener su siguiente atracción). Si puede subirse a la siguiente atracción de su lista, debes permitir que ingrese a la fila haciendo uso del método implementado para esta acción. Debes modificarlo
  - `def simular_hora(self, hora: int)`: Recibe la hora actual, distribuye a todas las personas a una atracción de su preferencia y luego corre una nueva ronda en todas las atracciones disponibles, aunque estén vacías. No debes modificarlo
  - `def revisar_termino(self) -> bool`: Revisa si aún quedan personas con atracciones a las que aún no se han subido. Si hay al menos una persona con atracciones pendientes, retorna `True`, de lo contrario, retorna `False`. No debes modificarlo
  - `def iniciar_dia(self)`: Para cada hora, llama a los métodos `simular_hora()` y `revisar_termino()`, en ese orden. Termina de ejecutar una vez que no quedan personas con actividades pendientes en el parque. No debes modificarlo

Bonus

Debido al futuro crecimiento del parque de atracciones, deberás modelar dos nuevos tipos de atracciones. En el archivo `bonus.py`, encontrarás las clases definidas de una manera similar a las presentadas en la segunda parte de la actividad.

Si deseas hacer el bonus **debes descomentar la línea 10 del archivo `main.py`**. Adicionalmente, puedes comentar la línea 9 del mismo archivo, pero no es realmente necesario.

Clase AtraccionTerrorifica

Tienes que implementar la totalidad de esta clase.

- `class AtraccionTerrorifica`: Esta clase hereda de `AtraccionAdrenalinica`. **Debes modificarlo**
  - `def __init__(self, nombre: str, capacidad: int, salud_necesaria: int)`: Esta clase tiene que mantener los mismos atributos que hereda de `AtraccionAdrenalinica`, teniendo que entregarle estos a la superclase al instanciarse. Además de esto, deberás modificar los siguientes atributos: **Debes modificarlo**
    - `self.efecto_salud` por `SALUD_TERROR` ubicado en `parametros.py`
    - `self.efecto_felicidad` por `FELICIDAD_TERROR` ubicado en `parametros.py`
  - `def iniciar_juego(self, personas: list)`: Este método recibe una lista de objetos de la clase `Persona` y sobreescribe el método de su clase padre.

    Para cada persona en la lista, revisa si la salud es menor o igual al doble de la salud necesaria de la atracción. De ser así, se imprime un mensaje con el nombre de la persona diciendo que necesitará capacitación antes de jugar, para luego llamar al método `definir_estados` de esa persona. Una vez recorrida toda la lista, se llama al método `inicar_juego` de la superclase. **Debes modificarlo**

Clase CasaEmbrujada

Tienes que implementar la totalidad de esta clase.

- `class CasaEmbrujada`: Esta clase hereda de la clase `AtraccionFamiliar` y `AtraccionTerrorifica`. Deberás completar la clase tal que herede correctamente de sus superclases. **Debes modificarlo**

  El orden en la que se realiza la multiherencia tiene que ser tal que se mantenga el `efecto_salud` y `efecto_felicidad` de `AtraccionTerrorifica`.
  - `def iniciar_juego(self, personas: list)`: Este método sobreescribe el método de su superclase, y recibe una lista de objetos de la clase `Persona`. Tienes que reimplementar este método tal que llame al método definido en la clase `AtraccionFamiliar`. **Debes modificarlo**

Requerimientos

- **(2.25 pts) Modelación de Personas**
  - (0.75 pts) Modelar correctamente la clase Persona
  - (0.75 pts) Modelar correctamente la clase Adulto
  - (0.75 pts) Modelar correctamente la clase Niño

- **(3.00 pts) Modelación de Atracciones**
  - (0.75 pts) Modelar correctamente la clase AtraccionFamiliar
  - (0.75 pts) Modelar correctamente la clase AtraccionAdrenalinica
  - (0.75 pts) Modelar correctamente la clase AtraccionAcuatica
  - (0.75 pts) Modelar correctamente la clase MontanaAcuatica
- **(0.75 pts) Implementar correctamente la distribución de personas en Parque**
- **(0.50 pts) Bonus**

Diagrama de clase

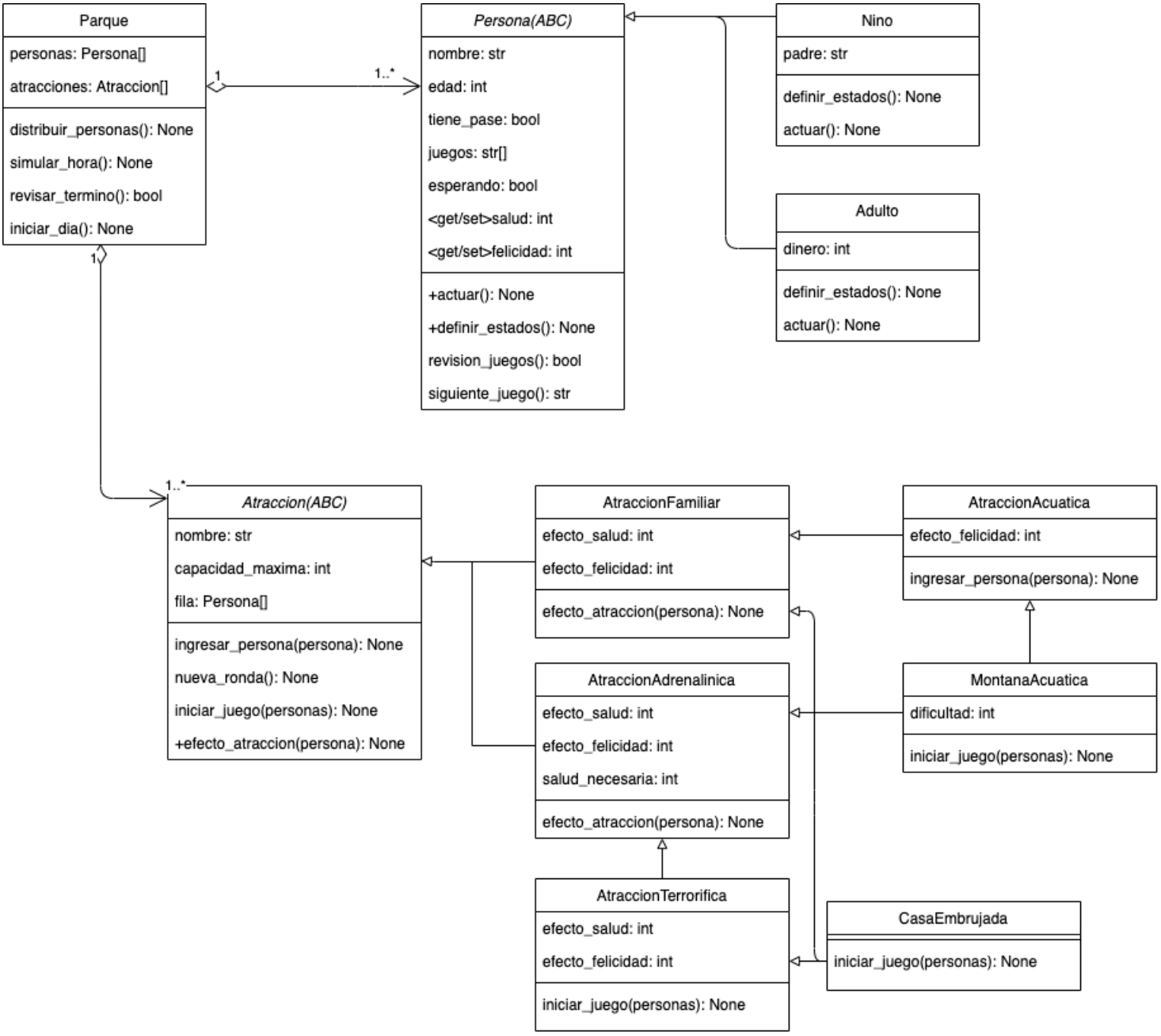

Figura 2: Diagrama de clases