

## TRABAJO PRÁCTICO Nº 3 – ASOCIACIÓN Y DEPENDENCIA

Unidad 4 – Asociación y Dependencia

PROGRAMACIÓN 2 - 2022 – 2do cuatrimestre

TECNICATURA UNIVERSITARIA EN DESARROLLO WEB

### EL TRABAJO PRÁCTICO Nº 3 TIENE POR OBJETIVO QUE EL ALUMNO

- Aplique y refuerce los conceptos referidos a Asociación y Dependencia de Clases y Objetos.
- Sea capaz de interpretar y traducir correctamente los diagramas de Clases en código Python respetando las relaciones entre dichas Clases y los Objetos creados a partir de ellas.

### CONDICIONES DE ENTREGA

- El Trabajo Práctico deberá ser:
  - Realizado en forma individual o en **grupos de NO más de 4 (cuatro) alumnos**.
  - Cargado en la sección del Campus Virtual correspondiente, en un archivo ZIP o RAR con las soluciones a cada ejercicio que requiera una solución en código Python, y en un archivo Word las respuestas a las preguntas que no requieran una solución en código Python. Las soluciones para los ejercicios que requieran de código Python debe estar contenidas cada una en un archivo .py distinto.
  - En caso de realizar el Trabajo Práctico en grupo, deberá indicarse el apellido y nombre de los integrantes del mismo. Todos los integrantes del grupo deben realizar la entrega en el campus y deberá agregarse al comprimido con las soluciones un archivo *integrantes.txt* con los nombres de los participantes.
  - Entregado antes de la fecha límite informada en el campus.

- El Trabajo Práctico será calificado como Aprobado o Desaprobado.
- Las soluciones del alumno/grupo deben ser de autoría propia. De encontrarse soluciones idénticas entre diferentes grupos, dichos trabajos prácticos serán clasificados como **DESAPROBADO**, lo cual será comunicado en la devolución.

## EJERCICIOS:

Al igual que el Trabajo Práctico Nro. 2, las siguientes consignas estarán relacionadas al universo de la película de Pixar / Disney, Monsters Inc, donde el alumno deberá modelar las distintas entidades de dicho universo.

A continuación, como punto de inicio, se requiere tomar del Trabajo Práctico Nro. 2 las clases implementadas para los siguientes diagramas:

Monstruo
<pre>&lt;&lt;Atributos de clase&gt;&gt; maxEnergia: int &lt;&lt;Atributos de instancia&gt;&gt; nombre: string especie: string energía: int</pre>
<pre>&lt;&lt;Constructores&gt;&gt; Monstruo(nom, esp: string) &lt;&lt;Comandos&gt;&gt; establecerNombre(nom: string) establecerEspecie(esp: string) establecerEnergia(ene: int) asustar(hum: Humano) divertir(hum: Humano) &lt;&lt;Consultas&gt;&gt; obtenerNombre(): string obtenerEspecie(): string obtenerEnergia(): int</pre>

Humano
<<Atributos de clase>> especie: string <<Atributos de instancia>> nombre: string estadoAsustado: boolean
<<Constructores>> Humano(nom: string) <<Comandos>> establecerNombre(nom: string) establecerEstadoAsustado(est: boolean) <<Consultas>> obtenerNombre(): string obtenerEstadoAsustado(): boolean

MonstersInc
<<Atributos de clase>> <<Atributos de instancia>> monstruos: Monstruo[] humanos: Humano[]
<<Constructores>> MonstersInc() <<Comandos>> agregarMonstruo(mon: Monstruo) agregarHumano(hum: Humano) eliminarMonstruo(mon: Monstruo) eliminarHumano(hum: Humano) <<Consultas>> obtenerMonstruos(): Monstruo[] obtenerHumanos(): Humano[]



monstruo.py



monstersinc.py



humano.py

1. Modificar la clase Monstruo para que se ajuste al siguiente diagrama:

Monstruo
<pre>&lt;&lt;Atributos de clase&gt;&gt; maxEnergia = 100 &lt;&lt;Atributos de instancia&gt;&gt; nombre: string especie: string energía: int <b>estadoDormido: boolean</b></pre>
<pre>&lt;&lt;Constructores&gt;&gt; Monstruo(nom, esp: string) &lt;&lt;Comandos&gt;&gt; establecerNombre(nom: string) establecerEspecie(esp: string) establecerEnergia(ene: int) <i>Requiere 0 &lt;= ene &lt;= 100</i> <b>establecerEstadoDormido(est: boolean)</b> asustar(hum: Humano) <i>Requiere hum Ligado, estadoDormido = False y energía &gt;= 10</i> <i>Requiere hum.estadoAsustado = False</i> divertir(hum: Humano) <i>Requiere hum Ligado, estadoDormido = False y energía &gt;= 20</i> <b>dormir()</b> <i>Requiere energia &lt;= maxEnergia</i> <b>despertar()</b> &lt;&lt;Consultas&gt;&gt; obtenerNombre(): string obtenerEspecie(): string obtenerEnergia(): int <b>obtenerEstadoDormido(): boolean</b></pre>

- a. El atributo de instancia **estadoDormido** toma el valor inicial False.
- b. El comando **dormir** debe:
  - i. Cambiar el atributo de instancia **estadoDormido** a True.

- ii. Incrementar la energía del monstruo en 15 unidades. No debe ser posible superar el valor establecido en **maxEnergía**.
- c. El comando ***despertar*** debe cambiar el valor del atributo **estadoDormido** a False.
- d. Los comandos ***asustar*** y ***divertir***:
  - i. Solo deben tener efecto si **estadoDormido** tiene valor False.
  - ii. No deben permitir que la energía del monstruo se vea decrementada por debajo de 0.
- e. El comando ***asustar*** solo debe tener efecto si el atributo **estadoAsustado** de **hum** tiene valor False.

2. Implementar la clase Puerta como se especifica en el siguiente diagrama:

Puerta
<<Atributos de clase>> <<Atributos de instancia>> numero: int humano: Humano monstruo: Monstruo estadoActiva: boolean
<<Constructores>> Puerta(num: int, hum: Humano) <i>Requiere hum Ligado</i> <<Comandos>> establecerHumano(hum: Humano) <i>Requiere hum Ligado</i> establecerMonstruo(mon: Monstruo) <i>Requiere mon Ligado</i> establecerEstadoActiva(est: boolean) <<Consultas>> obtenerNumero(): int obtenerHumano(): Humano obtenerMonstruo(): Monstruo obtenerEstadoActiva(): boolean obtenerEstadoEnUso(): boolean <i>Requiere estadoActiva = True y monstruo Ligado</i>

- El valor inicial de **estadoActiva** debe ser False.
- Inicialmente, el atributo de instancia **monstruo** tiene valor nulo (debe inicializarse con valor None).
- El atributo de instancia **numero** no cuenta con un método para establecer su valor ya que este debe ser inmutable por ser el número identificador de cada puerta.
- La consulta **obtenerEstadoEnUso** debe devolver True cuando el valor atributo de instancia **estadoActiva** sea igual a True y **monstruo** tenga un valor distinto de None.

- e. Escribir el método **equals** de forma tal que compare los humanos asociados a ambas puertas, específicamente si los atributos **numero** y **estadoActivo** de ambas puertas coincide, y estas están ligadas a los mismos objetos de tipo Humano y de tipo Monstruo.
  - i. Luego de implementar este método, ¿su ejecución estará refiriendo a una comparación de estados internos (igualdad en profundidad) o de identidad (igualdad superficial)?



3. Modificar el comportamiento de la clase Monstruo de acuerdo al diagrama detallado a continuación:

Monstruo
<pre>&lt;&lt;Atributos de clase&gt;&gt; maxEnergia = 100 minEnergia = 15 &lt;&lt;Atributos de instancia&gt;&gt; nombre: string especie: string <b>tipo: string</b> <b>pareja: Monstruo</b> energía: int estadoDormido: boolean</pre>
<pre>&lt;&lt;Constructores&gt;&gt; Monstruo(nom, esp, tipo: string) <i>Requiere tipo = ('asustador', 'asistente')</i> &lt;&lt;Comandos&gt;&gt; establecerNombre(nom: string) establecerEspecie(esp: string) establecerPareja(mou: Monstruo) <i>Requiere mou ligado y mou.tipo &lt;&gt; tipo</i> establecerEnergia(ene: int) <i>Requiere 0 &lt;= ene &lt;= 100</i> establecerEstadoDormido(est: boolean) <b>activarPuerta(pue: Puerta, mou: Monstruo)</b> <i>Requiere tipo = 'asistente'</i> <i>Requiere pue y mou ligados</i> <i>Requiere pue.obtenerEstadoEnUso() = False</i> <b>asustar(pue: Puerta)</b> <i>Requiere pue ligado, estadoDormido = False y energía &gt;= 15</i> <i>Requiere pue.estadoActiva = True</i> <i>Requiere pue.humano y pue.monstruo ligado</i> <i>Requiere pue.monstruo.tipo = 'asustador'</i> <b>divertir(pue: Puerta)</b> <i>Requiere pue ligado, estadoDormido = False y energía &gt;= 15</i></pre>

```
dormir()  
Requiere energia <= maxEnergia  
despertar()  
<<Consultas>>  
obtenerNombre(): string  
obtenerEspecie(): string  
obtenerEnergia(): int  
obtenerEstadoDormido(): boolean
```

- a. El atributo de clase **minEnergia** indica la energía mínima que se requiere para que puedan ser consumidos los servicios **asustar** y **divertir** de un monstruo. El mismo tiene un valor fijo de 15 unidades. Además, ambos comandos pasan a consumir la misma cantidad de energía, 10 unidades, al ser ejecutados.
- b. El atributo de instancia **tipo** solo puede tomar los valores “asustador” y “asistente”.
- c. El atributo de instancia **pareja** debe tomar como valor un objeto de tipo Monstruo cuyo atributo **tipo** no coincida con el de él. Es decir que para un monstruo cuyo atributo de instancia **tipo** tiene valor “asustador”, el valor del atributo de instancia **tipo** de su **pareja** debe ser “asistente” y viceversa.
- d. El comando **activarPuerta** solo tiene efecto si el monstruo es de tipo “asistente”. Un monstruo de tipo “asustador” no puede activar una puerta. Al invocarse dicho servicio se debe:
  - i. Marcar la puerta como activa.
  - ii. Asociar la puerta con el monstruo pareja.
  - iii. Previamente verificar que no está en uso por otro monstruo.
- e. Los comandos **asustar** y **divertir** de un monstruo deben:
  - i. Verificar que la puerta se encuentra activa y asociada a dicho monstruo.
  - ii. **asustar** solo puede ser ejecutado por un objeto de tipo Monstruo cuyo valor de su atributo de instancia **tipo** sea igual a “asustador”.
  - iii. Al invocarse los servicios en cuestión, las condiciones y requisitos expuestos en puntos anteriores deben aplicarse sobre el humano asociado a la puerta.

4. Modificar la clase MonstersInc de acuerdo a lo especificado en el siguiente diagrama:

MonstersInc
<<Atributos de clase>> <<Atributos de instancia>> <b>monstruos: Monstruo[]</b> <b>puertas: Puerta[]</b>
<<Constructores>> MonstersInc() <<Comandos>> agregarMonstruos(mon: Monstruo) <b>agregarPuerta(pue: Puerta)</b> eliminarMonstruos(mon: Monstruo) <b>eliminarPuerta(pue: Puerta)</b> <<Consultas>> obtenerMonstruos(): Monstruo[] obtenerPuertas(): Puerta[]

- a. Monstruos deja de ser una lista de objetos de tipo Monstruo para pasar a ser una lista de diccionarios de la siguiente forma:
- ```
{“asustador”: Monstruo, “asistente”: Monstruo}
```
- Donde tanto “asustador” y “asistente” están enlazados por el atributo **pareja**, y estos cuentan con la restricción de ser de tipo “asustador” y “asistente”, respectivamente.
- b. El comando **eliminarPuerta** debe eliminar de la lista **puertas** aquella puerta para la cual la consulta **equals** devuelva True respecto del parámetro formal **pue**.

5. Construya un programa, utilizando la clase proveedora `MonstersInc` que, a partir de la entrada del usuario permita evaluar todos los puntos expuestos a lo largo del TP:
- a. Agregar y eliminar monstruos, humanos y puertas.
  - b. Reproducir escenarios donde un monstruo asuste un humano.
  - c. Reproducir escenarios donde un monstruo divierta un humano.
  - d. Reproducir escenarios donde un monstruo no pueda asustar y/o divertir a un humano:
    - i. Humano con estado asustado.
    - ii. Monstruo con energía por debajo de la mínima.
    - iii. Puerta no activa.
    - iv. Puerta ya en uso.
  - e. Hacer que un monstruo recupere su energía durmiendo.

Para la construcción de dicho programa crear una clase de nombre `TesterMonstersInc` que actúe como cliente de la clase proveedora `MonstersInc`, cuyo único servicio sea de nombre ***main***, sin parámetros, que ejecute los puntos descriptos anteriormente. A continuación, un ejemplo de cómo dicho programa puede ser construido:

```
class TesterMonstersInc:
    def main():
        # Solución de los puntos 5.a., 5.b, 5.c, ...

if __name__ == '__main__':
    testerMonstersInc = TesterMonstersInc()
    testerMonstersInc.main()
```