Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2020-2

3 de septiembre de 2020
Actividad Sumativa

Actividad Sumativa 01

Programación Orientada a Objetos II

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AS01/`
- **Hora del *push*:** 16:50

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**.

Importante: Dada la extensión de la actividad, la nota máxima será **8.0**, es decir, que se puede obtener la nota máxima sin completar toda la actividad. De todas maneras te invitamos a intentarlo ¡Mucha suerte!

Introducción

El Quidditch es el deporte número uno del mundo mágico, seguido por millones de Programagos. Todos los años, los equipos profesionales reclutan jugadores jóvenes para unirse a sus líneas mediante pruebas de selección que miden sus habilidades en el juego.

Dentro de los próximos días serán las pruebas de selección del Deportivo Chudley Cannons, equipo que busca renovarse casi completamente para superar las 20 malas temporadas que ha tenido. Las pruebas se realizarán en el colegio Pythonwarts, y en ellas podrán competir sus estudiantes y otros jugadores no estudiantes que se presenten.

El director DumbleDrPinto te ha pedido usar tus destrezas para programar y tu conocimiento de OOP para simular las pruebas que determinarán quiénes ocuparán las vacantes del prestigioso equipo. Para predecir sus resultados deberás modelar a los jugadores que competirán y a los espectadores utilizando multiherencia y clases abstractas.

Flujo del programa

Importante: Dado que hay una gran cantidad de 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.

Todos los personajes en la simulación son **Programagos**, los cuales pueden ser **Estudiantes**, **Jugadores** o ambos a la vez. Los estudiantes, pueden ser **EstudianteGryffindor** o **EstudianteSlytherin** según la casa a la que pertenezcan; de acuerdo a la cual tendrán ciertas características y podrán dar o recibir apoyo a los jugadores de su casa. Al mismo tiempo, cada uno de los jugadores tiene una posición, que puede ser **Buscador**, **Golpeador** o **Cazador**. Tu labor es completar la herencia y los métodos de estas clases, ya que la simulación de las pruebas (explicada a continuación) ya viene implementada.

El programa comienza con la creación de las instancias de los estudiantes de la simulación. Utilizando la información que se encuentra en el archivo `estudiantes.csv`, se instancian los estudiantes que son jugadores y se guardan en el diccionario `PARTICIPANTES`, mientras que los estudiantes que no participan en las pruebas se guardan en el diccionario `ESPECTADORES`.

Posteriormente, se instancian los jugadores que son externos (que no son estudiantes) a Pythonwarts, cuya información se encuentra en el archivo `externos.csv` y se almacenan en el mismo diccionario `PARTICIPANTES`, dejando en este último a todos los Programagos que participarán en las pruebas de los Chudley Cannons.

Finalmente se da inicio a la prueba de selección. Primero, los estudiantes jugadores reciben aplausos y abucheos de los espectadores: es decir, los estudiantes no jugadores. Los aplausos de una casa alentarán a los jugadores de la misma casa, y los abucheos de esta misma casa desalentarán a los jugadores de la la casa contraria. Los aplausos y abucheos, junto con las características de la casa a la que pertenecen, generan modificaciones en las habilidades y afectan el rendimiento de los jugadores durante la competencia. Después, se simula la prueba para cada una de las posiciones, y aquellos que obtienen mejores resultados son seleccionados para llenar las vacante de los Chudley Cannons.

Archivos

Para esta actividad se te hará entrega de los siguientes archivos:

Código

- `main.py`: Este es el archivo principal del programa. Puedes ejecutarlo para probar el funcionamiento de tu programa completo. **Ya viene implementado, y NO debes modificarlo**.
- `cargar_datos.py`: Contiene las funciones `cargar_estudiantes` y `cargar_jugadores_externos`, que permiten procesar los archivos de datos. Este archivo ya está completo, **NO debes modificarlo**.

- `estudiantes.py`: Contiene las clases `Programago`, `Estudiante`, `EstudianteGryffindor` y `EstudianteSlytherin`, las cuales **deberás completar** según lo pedido.
- `jugadores.py`: Contiene las clases `Jugador`, `Buscador`, `Golpeador` y `Cazador`, las cuales **deberás completar** según lo pedido.
- `jugadores_estudiantes.py`: Tiene las clases `BuscadorGryffindor`, `GolpeadorGryffindor`, `CazadorGryffindor`, `BuscadorSlytherin`, `GolpeadorSlytherin` y `CazadorSlytherin`, las cuales **deberás completar** según lo pedido.
- `competencia.py`: Contiene la clase `Competencia` que modela el enfrentamiento entre los jugadores. Este archivo ya está completo y **NO debes modificarlo**.
- `parametros.py`: Tiene constantes en `ESTE_FORMATO` que son usadas como parámetros en el resto del código. Este archivo ya está completo y **NO debes modificarlo**.

Datos

- `estudiantes.csv`: Contiene la base de datos de los estudiantes, tanto jugadores como espectadores. Cada línea define uno y viene de la forma:

  `nombre, saludo, casa, posicion, numero`

  donde `nombre` es nombre del `Estudiante`, `saludo` es su frase característica y `casa` indica si pertenece a Gryffindor o Slytherin. Si el `Estudiante` es además un `Jugador`, tendrá en `posicion`, la posición en la que juega, y en `numero` su número de camiseta. En caso de no ser un `Jugador`, los valores correspondientes a estas dos columnas estarán en blanco. Este archivo **NO debes modificarlo**.
- `externos.csv`: Contiene la base de datos de los jugadores externos, quienes no son estudiantes. Cada línea define uno y viene de la forma:

  `nombre, saludo, posicion, numero`

  donde `nombre` es nombre del `Jugador`, `saludo` es su frase característica, `posicion` indica la posición en la que juega, y `numero` es su número de camiseta. Este archivo **NO debes modificarlo**.

Parte 1: Modelación `Programago` 🧙

En esta parte deberás completar la clase `Programago`, que se encuentran en el archivo `estudiantes.py`. Debes definirla correctamente en cuanto a sus atributos y métodos, y en cuanto a sus relaciones. Al final del documento hay un diagrama de clase que te puede ayudar a visualizar las herencias.

- `class Programago`: Esta es la clase que tienen en común todas las entidades que deberás crear. Representa a todos los `Programagos`, incluyendo `Estudiantes` y `Jugadores`, que heredan de ella. Debes definir esta clase como **abstracta**.
  - `def __init__(self, nombre, saludo)`: Este metodo ya viene implementado. Recibe el `nombre` de un `Programago` y su `saludo`. Son asignados a los atributos `self.nombre` y `self.saludo` respectivamente. Estos son los únicos atributos de la clase.

Parte 2: Modelación Estudiantes

En esta parte deberás completar las clases que corresponden a estudiantes de Pythonwarts, que se encuentran en el archivo `estudiantes.py`.

Clase Estudiante 🎓

- `class Estudiante`: Esta es la clase que define a los estudiantes de Pythonwarts. Deberás completar esta clase, definiéndola como **abstracta** y agregando los **métodos abstractos** `abuchear` y `aplaudir`. Además de ser abstracta, esta clase hereda de **Programago**. Esta clase tiene los siguientes métodos:
  - `def __init__(self, nombre, saludo)`: Debes completar este método, instanciando `Programago`, que recibe `nombre` y `saludo`. Son asignados a los atributos `self.nombre` y `self.saludo` respectivamente. Incluyendo los mencionados, este método crea los siguientes atributos:
    - `self.nombre`: El nombre del `Estudiante`
    - `self.saludo`: El saludo del `Estudiante`

    Los siguientes atributos son valores comunes entre los estudiantes y son usados más adelante en la simulación. Estos 4 atributos ya están implementados en esta clase:
    - `self.valor`: `int` que representa esta característica.
    - `self.inteligencia`: `int` que representa esta característica.
    - `self.lealtad`: `int` que representa esta característica.
    - `self.ambicion`: `int` que representa esta característica.
  - `def abuchear(self)`: Deberás completar esto como un **método abstracto**. Puedes dejar un `pass` al crear el método, lo importante es que este definido correctamente como un **método abstracto**.
  - `def aplaudir(self)`: Deberás completar esto como un método abstracto. Puedes dejar un `pass` al crear el método, lo importante es que este definido correctamente como un **método abstracto**.

Clases `EstudianteGryffindor` 🦁 y `EstudianteSlytherin` 🐍

Estas clases heredan de `Estudiante` y representa a aquellos estudiantes de la casa Gryffindor y Slytherin respectivamente. Deberás completar las clases tal que hereden correctamente de `Estudiante`. Además deberás sumar al atributo característico de cada casa un `int` aleatorio [1] entre valores mínimos y máximos para cada casa, que se encuentran en el diccionario `BONOS_CASA` en el archivo `parametros.py`, recuerda revisar este archivo. En el caso de `EstudianteGryffindor` deberás bonificar el atributo `valor` y para `EstudianteSlytherin` deberás bonificar el atributo `ambicion`.

Esta clase tiene los siguientes métodos:

- `def __init__(self, nombre, saludo, *args)`: Deberás completar este método. Se envian los argumentos recibidos a la **superclase** y agrega la bonificación al atributo `valor` o `ambicion` según corresponda.
- `def aplaudir(self)`: Este método ya viene implementado. Calcula una probabilidad dada por el valor `PROBABILIDAD_APLAUDIR_GRYFFINDOR` o `PROBABILIDAD_APLAUDIR_SLYTHERIN`. En caso de cumplirse, el método imprime un mensaje alentador y retorna `True`. En caso contrario retorna `False`.
- `def abuchear(self)`: Este método ya viene implementado. Calcula una probabilidad dada por el valor `PROBABILIDAD_ABUCHEAR_GRYFFINDOR` o `PROBABILIDAD_ABUCHEAR_SLYTHERIN`. En caso de

[1] Te recomendamos utilizar la función `randint` de la librería `random`

cumplirse, el método imprime un mensaje dirigido al equipo rival y retorna `True`. En caso contrario retorna `False`.

Para probar tus avances en este módulo, puedes correr directamente el archivo `estudiantes.py`. El código dentro de `if __name__ == "__main__"` instancia las clases `EstudianteGryffindor` y `EstudianteSlytherin`, y corrobora que las herencias estén correctamente realizadas. Si tus clases están definidas correctamente, el *output* debería tener un formato similar a este:

```
"Soy Pruebardo y estoy probando la clase EstudianteGryffindor"
"EstudianteGryffindor hereda correctamente de Estudiante!"
"EstudianteSlytherin hereda correctamente de Programago!"
```

Parte 3: Modelación de los Jugadores

Para esta parte deberás completar las clases que se encuentran en el archivo `jugadores.py`: `Jugador`, `Buscador`, `Golpeador` y `Cazador`. Debes definir correctamente las clases y métodos **abstractos**, y utilizar herencia cuando sea necesario.

Clase `Jugador` 

- `class Jugador`: Esta es una clase abstracta que hereda de `Programago` y representa a aquellos `Programagos` que juegan Quidditch. Deberás completarla tal que herede correctamente y definirla como una clase `abstracta`. Además deberás definir los **métodos abstractos** `competir` y `asignar_cualidades` , el **método concreto** `celebrar` y completar el método `__init__`.
  - `def __init__(self, nombre, saludo, numero_polera)`: Debes completar este método, enviando los argumentos a la **Superclase**.. Este método tiene los siguientes atributos (ya implementados):
    - `self.nombre`: Un `str` que representa el nombre del `Jugador`.
    - `self.saludo`: Un `str` que representa el saludo del `Jugador`.
    - `self.numero_polera`: Un `int` que representa el numero de polera del `Jugador`.
    - `self.nerviosismo`: `float` que indica que tan nervioso está el jugador. Su valor se asigna cuando se llama al método `asignar_cualidades(self)` en las subclases de `Jugador`.
    - `self.velocidad`: `float` que indica lo rápido que es el jugador. Su valor se asigna cuando se llama al método `asignar_cualidades(self)` en las subclases de `Jugador`
    - `self.equilibrio`: `float` que indica que tanto equilibrio tiene el jugador. Su valor se asigna cuando se llama al método `asignar_cualidades(self)` en las subclases de `Jugador`
  - `def asignar_cualidades(self)`: Deberás crear esto como un método abstracto. Puedes dejar un `pass` al crear el método, lo importante es que este definido correctamente como un **método abstracto**.
  - `def competir(self)`: Deberás crear esto como un método abstracto. Puedes dejar un `pass` al crear el método, lo importante es que este definido correctamente como un **método abstracto**.
  - `def celebrar(self`: Debes completar este método tal que, al llamarlo, imprima el nombre del `Jugador` y un mensaje de celebración. Por ejemplo:

```
        "Dobby: Lo logré!"
```

Clases `Buscador` 🔎, `Golpeador` 🏏 y `Cazador` 🏹

Estas clases heredan de `Jugador`. Deberás completarlas tal que hereden correctamente y completar el método `competir`, tal que sobreescriba el metodo abstracto `competir` de la clase `Jugador`.

Estas clases tienen los siguientes métodos:

- `def __init__(self, nombre, saludo, numero_polera)`: Deberás completar este método realizando la herencia de la clase `Jugador`. Además, se llama al método `asignar_cualidades`, lo cual ya está implementado.
- `def competir(self)`: Deberás completar este método. Retorna el valor (como `float`) con el que el jugador se va a enfrentar a otro.

  El valor se calcula como la suma de `velocidad` y `equilibrio`, multiplicados por sus respectivos ponderadores y restando `neviosismo` multiplicado por su ponderador. Existe un ponderador para cada cualidad y para cada posición y estos se encuentran en el archivo `parametros.py`

  Por ejemplo, para el golpeador, debe ser:

```
self.velocidad * PONDERADOR_VELOCIDAD_GOLPEADOR
+ self.equilibrio * PONDERADOR_EQUILIRBIO_GOLPEADOR
- self.nerviosismo * PONDERADOR_NERVIOSISMO_GOLPEADOR
```

- `def asignar_cualidades(self)`: Este método ya viene implementado. Sobreescribe el método abstracto y determina aleatoriamente (pero dentro de un rango) el valor de las cualidades del jugador.

Para probar tus avances en este módulo, puedes correr directamente el archivo `jugadores.py`. El código dentro de `if __name__ == "__main__"` instancia las clases `Buscador`, `Golpeador` y `Cazador`, y corrobora que las herencias estén correctamente realizadas. Si tus clases están definidas correctamente, el *output* debería tener un formato similar a este:

```
"Soy Pruebinelda y estoy probando la clase Buscador, mi numero de polera es 42"
"Soy Pruebardo y estoy probando la clase Golpeador, mi numero de polera es Pi"
"Golpeador hereda correctamente de Jugador!"
"Cazador hereda correctamente de Jugador!"
```

Parte 4: Modelación de las clases Jugadores-Estudiantes

Esta parte incluyes las clases `CazadorGryffindor`, `GolpeadorGryffindor`, `BuscadorGryffindor`, `CazadorSlytherin`, `GolpeadorSlytherin` y `BuscadorSlytherin`. Estas clases se encuentran en el archivo `jugadores_estudiantes.py`, y representan a estudiantes que además son jugadores. A continuación se describe el método que tienen en común:

- `def __init__(self)`: Deberás completar este método para que se aplique la herencia correspondiente.

Además, todas las clases tienen el siguiente método que las diferencia, por lo que debe ser implementado de manera distinta en cada clase:

- `def celebrar(self)`: Estos métodos vienen implementados. Imprimen el nombre del jugador-estudiante junto a su celebración, que es característica para cada tipo de jugador-estudiante.
- `def competir(self)`: Deberás completar este método para cada una de las clases. Esta función retorna un número dependiendo del tipo de `Jugador` y se detalla a continuación:

- **CazadorGryffindor** y **CazadorSlytherin**
  Retorna un número definido en el método competir de la clase `Cazador`, más un bono que se calcula de la siguiente manera:

```
    bono = self.valor * 0.2
```

- **GolpeadorGryffindor** y **GolpeadorSlytherin**
  Retorna un número definido en el método competir de la clase `Golpeador`, más un bono que se calcula de la siguiente manera:

```
    bono = self.ambicion * 0.2
```

- **BuscadorGryffindor** y **BuscadorSlytherin**
  Retorna un número definido en el método competir de la clase `Buscador`, más un bono que se calcula de la siguiente manera:

```
    bono = self.inteligencia * 0.2
```

Para probar tus avances en este módulo, puedes correr directamente el archivo `jugadores_estudiantes.py`. El código dentro de `if __name__ == "__main__"` instancia las clases implementadas en esta parte, y corrobora que las herencias estén correctamente realizadas. Si tus clases están definidas correctamente, el *output* debería tener un formato similar a este:

```
"Soy Pruebina y estoy probando la clase BuscadorGryffindor, mi numero de polera es 42"
"Soy Pruebin y estoy probando la clase GolpeadorGryffindor, mi numero de polera es Pi"
"BuscadorGryffindor hereda correctamente de EstudianteGryffindor!"
"GolpeadorGryffindor hereda correctamente de EstudianteGryffindor!"
```

Además, se imprimirá un mensaje por cada herencia correcta de de las clases padre. Por ejemplo, si BuscadorGryffindor está bien implementado, se imprimirán los siguientes mensajes (no necesariamente seguidos):

```
    BuscadorGryffindor hereda correctamente de EstudianteGryffindor!
    BuscadorGryffindor hereda correctamente de Buscador!
```

Output Esperado

Una vez completada la actividad, al correr el archivo `main.py` el *output* debería tener un formato similar a este:

```
    Magomeflores: buuuuu!! Que mal Gryffindor!!!
    Santelisium: *clap clap*
    ...


    ¡Felicitamos a le/les golpeadores seleccionados!
    Felicidades jugador numero 13
    Ulan Bator: Soy el mejor, malditos!


    ¡Felicitamos a le/les cazadores seleccionados!
```

```
    Felicidades jugador numero 17
    Lily416 Potter: Lo logré solito!


    ¡Felicitamos a le/les buscadores seleccionados!
    Felicidades jugador numero 6
    Patolf el gris: No son nada para mi
```

Notas

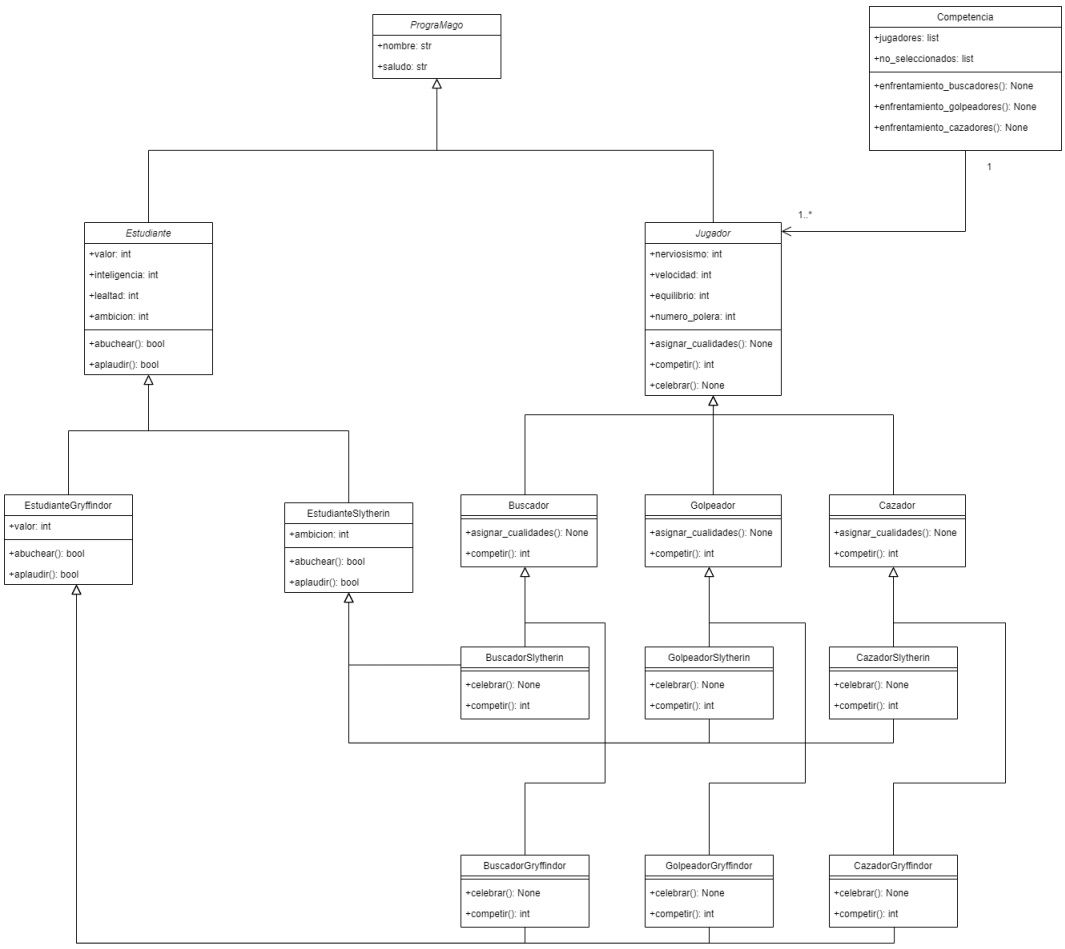
- La recolección de la actividad se hará en la rama principal (`master`) de tu repositorio.

Requerimientos

- (0.5 pts) Modelación de `Programago`
  - (0.5 pts) Completar correctamente `Programago`
- (2.5) Modelación de `Estudiante`
  - (1 pts) Completar correctamente `Estudiante`
  - (0.75) Completar correctamente `EstudianteGryffindor`
  - (0.75) Completar correctamente `EstudianteSlytherin`
- (2.5 pts) Modelación de jugadores
  - (1 pts) Completar correctamente `Jugador`
  - (0.5 pts) Completar correctamente `Buscador`
  - (0.5 pts) Completar correctamente `Golpeador`
  - (0.5 pts) Completar correctamente `Cazador`
- (1.5 pts) Modelación de jugadores-estudiantes
  - (0.25 pts) Completar correctamente `BuscadorGryffindor`
  - (0.25 pts) Completar correctamente `GolpeadorGryffindor`
  - (0.25 pts) Completar correctamente `CazadorGryffindor`
  - (0.25 pts) Completar correctamente `BuscadorSlytherin`
  - (0.25 pts) Completar correctamente `GolpeadorSlytherin`
  - (0.25 pts) Completar correctamente `CazadorSlytherin`

Anexo

A continuación, a modo de guía, se adjunta un diagrama de clases del programa:


PrograMago
+nombre: str
+saludo: str
Competencia
+jugadores: list
+no_seleccionados: list
+enfrentamiento_buscadores(): None
+enfrentamiento_golpeadores(): None
+enfrentamiento_cazadores(): None
1
1..*
Estudiante
+valor: int
+inteligencia: int
+lealtad: int
+ambicion: int
+abuchear(): bool
+aplaudir(): bool
Jugador
+nerviosismo: int
+velocidad: int
+equilibrio: int
+numero_polera: int
+asignar_cualidades(): None
+competir(): int
+celebrar(): None
EstudianteGryffindor
+valor: int
+abuchear(): bool
+aplaudir(): bool
EstudianteSlytherin
+ambicion: int
+abuchear(): bool
+aplaudir(): bool
Buscador
+asignar_cualidades(): None
+competir(): int
Golpeador
+asignar_cualidades(): None
+competir(): int
Cazador
+asignar_cualidades(): None
+competir(): int
BuscadorSlytherin
+celebrar(): None
+competir(): int
GolpeadorSlytherin
+celebrar(): None
+competir(): int
CazadorSlytherin
+celebrar(): None
+competir(): int
BuscadorGryffindor
+celebrar(): None
+competir(): int
GolpeadorGryffindor
+celebrar(): None
+competir(): int
CazadorGryffindor
+celebrar(): None
+competir(): int