Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2020-1

26 de marzo de 2020
Actividad Sumativa

Actividad Sumativa 01

OOP I

Antes de comenzar...

Para esta, y todas las actividades del semestre, como equipo docente esperamos que sigas el siguiente flujo de trabajo:

- Lee el enunciado completo, incluyendo las notas. Puedes revisar los archivos subidos a medida que lees, o al final, como te acomode.
- Antes de comenzar a programar, copia y pega todos los archivos de la carpeta AS01 del *Syllabus* y pégalos en la misma carpeta de **tu repositorio personal**.
- Haz `git add`, `git commit` y `git push` de los archivos copiados inmediatamente, para comprobar que el uso de Git esté funcionando correctamente. (***Tip:*** Si no recuerdas como utilizar estos comandos, puedes revisar el enunciado de la AC00)
- En caso de encontrar un error, contacta a un ayudante para resolver el problema lo antes posible, en caso de no hacerlo, tu actividad podría no entregarse correctamente.
- Comienza a trabajar en tu actividad en tu repositorio y recuerda subir cada vez que logres un avance significativo (con los mismos comandos de Git de antes).
- Todas las actividades y tareas tienen una fecha y hora de entrega, en la cual automáticamente se recolecta el último *commit* ***pusheado*** en tu repositorio. Esto no quiere decir que solo se consideran los cambios de ese último *commit*, si no que todos los avances **hasta** ese *commit*. Luego, es importante realices `git push` de todos tus avances, antes de la fecha y hora de entrega. En este caso, es a las 16:50 de hoy.

Introducción

Frente a los hechos ocurridos recientemente, los alumnos, alumnas, ayudantes y profesores de IIC2233 han decidido cumplir una cuarentena preventiva. Luego de un par de días de encierro, (y de verse la mitad de Netflix), los profesores comenzaron a cuestionar el estado de sus alumnos, alumnas y ayudantes. ¿Están lo suficientemente entretenidos? ¿O sus vidas han sido completamente consumidas por sus deberes estudiantiles? Para ello, los profesores recolectaron los distintos deberes y pasatiempos de cada uno de los alumnos, alumnas y ayudantes para estas semanas. Te han pedido que realices un programa capaz de sugerir actividades según los niveles de felicidad y estrés de cada persona.

Vamos a construir un programa orientado a objetos que modele a un conjunto de alumnos y ayudantes junto con sus respectivas actividades, que pueden ser *hobbies* o deberes. En base a esto, una clase controladora `DCCuarentena` sugerirá actividades a realizar para cada estudiante. Te aconsejamos seguir la secuencia entregada en el enunciado pues te guiará en el desarrollo de la actividad.

Archivos

- `main.py`: Este archivo es donde se cargan las instancias de los estudiantes y actividades y se instancia `DCCuarentena` para poder hacer la simulación. **No debes modificar este archivo.**
- `cargar.py`: Este archivo contiene las funciones necesarias para cargar los archivos debes **completar** este archivo.
- `estudiantes.py`: Este archivo contiene a las clases `Estudiante`, `Alumno` y `Ayudante`, las cuales deberás **completar** según lo pedido.
- `actividades.py`: Este archivo contiene a las clases `Actividad`, `Hobby` y `Deber`, las cuales deberás **completar** según lo pedido.
- `dccuarentena.py`: Este archivo contiene la clase `DCCuarentena`, la cual deberás **completar** según lo pedido.
- `estudiantes.csv`: Es la base de datos de los estudiantes. Cada línea de este archivo define a uno, y viene de la forma:

  `tipo;username;hobby_1,...,hobby_n;deber_1,...,deber_n`

  El separador entre tipo, `username`, `hobbies` y `deberes` es el carácter `";"`, mientras que los distintos `hobbies` y `deberes` se separan mediante `","` entre ellos. En cada línea, el `tipo` puede ser `"alumno"` o `"ayudante"`.
- `actividades.csv`: Es la base de datos de las actividades. Cada línea de este archivo define a una, y viene de la forma:

  `tipo;nombre;felicidad;estres`

  El separador entre `tipo`, `nombre`, `felicidad`, `estres` es el carácter `";"`. En cada línea, el `tipo` puede ser `"hobby"` o `"deber"`.

Parte I: Modelamiento de Clases

En esta primera parte debes implementar las clases que modelan el sistema.

Mira el archivo `estudiantes.py`. En él se encuentra la clase `Estudiante`, que modela un estudiante, y las clases `Alumno` y `Ayudante` que heredan de ella. Debes completar estas clases de acuerdo a la siguiente

descripción:

- `class Estudiante`: El constructor de esta clase recibe tres argumentos: un nombre de usuario `username`, una lista de nombres de `hobbies` y una lista de nombres de `deberes`. Cada estudiante posee además dos tuplas, `rango_felicidad` y `rango_estres`, de dos elementos, que establecen los niveles mínimo y máximo de felicidad y estrés del estudiante. También, poseen los siguientes métodos:
  - `def realizar_actividad(self, actividad)`: Recibe como argumento una instancia de `Actividad`. Este método **imprime la actividad** que se está realizando.
  - `def felicidad(self)`: *Getter* del atributo `__felicidad` del objeto `Estudiante`.
  - `def felicidad(self, nueva_felicidad)`: *Setter* del atributo `__felicidad`. Debe verificar que `nueva_felicidad` no se salga del `rango_de_felicidad` permitido del estudiante. Si está dentro de los límites (incluyendo ambos extremos), actualiza `__felicidad`, de lo contrario el nuevo valor es el límite máximo o mínimo permitido.
  - `def estres(self)`: *Getter* del atributo `__estres` del estudiante.
  - `def estres(self, nuevo_estres)`: *Setter* de `__estres`. Debe verificar que `nuevo_estres` no se salga del `rango_de_estres` permitido . Si está dentro de los límites (incluyendo ambos extremos), actualiza `__estres`, de lo contrario el nuevo valor es el límite máximo o mínimo permitido.

Tanto alumnos como ayudantes poseen distintas tolerancias a la felicidad y al estrés, y también tienen distintas actividades que les parecen interesantes. Mientras un alumno prefiere estudiar los contenidos de IIC2233, un ayudante opta por corregir tareas de programación.

- `class Alumno`: Recibe como argumentos `username`, una lista de `hobbies` y una lista de `deberes`, los cuales deben ser pasados por el constructor de la clase padre. Tiene un nivel de `felicidad` inicial de 75 y un nivel de `estres` inicial de 25. Su `rango_de_felicidad` permitido va desde 0 hasta 200. Un `Alumno` obtiene una mayor `felicidad` cuando realiza *hobbies*. Por otro lado, poseen menor tolerancia al `estres`, por lo que su `rango_de_estres` va desde 0 hasta 100.
  - `def realizar_actividad(self, actividad)`: Recibe como argumento una instancia de `Actividad`. Debe **imprimir la actividad** que se está realizando. Suma el valor de `felicidad` de `actividad` multiplicado por 1,5 al nivel de `felicidad` del alumno y suma el valor de `estres` de `actividad` al nivel de `estres` del alumno.
- `class Ayudante`: Recibe como argumentos `username`, una lista de `hobbies` y una lista de `deberes`, los cuales deben ser pasados por el constructor de la clase padre. Tiene un nivel de `felicidad` inicial de 25 y un nivel de `estres` inicial de 75. Dados los muchos deberes que tienen este semestre, se estresan mucho más que un `Alumno`. Los `Ayudantes` se conforman con poca `felicidad` :'(, y ésta tiene un `rango_de_felicidad` permitido entre 0 y 100. Como están acostumbrados a hacer muchas cosas, poseen un `rango_de_estres` permitido entre 0 y 200.
  - `def realizar_actividad(self, actividad)`: Recibe como argumento una instancia de `Actividad`. Debe **imprimir la actividad** que se está realizando. Suma el valor de `felicidad` de `actividad` al nivel de `felicidad` del ayudante y suma **el doble** del valor de `estres` al nivel de `estres` del ayudante.

A continuación modelaremos las actividades mediante la clase `Actividad`. Las actividades disponibles pueden ser *hobbies* o deberes. Debes trabajar en el archivo `actividades.py` y completar las siguientes clases.

- `class Actividad`: tiene como argumentos los valores para `nombre`, `felicidad`, y `estres`.

Las clases `Hobby` y `Deber` deben **heredar** de `Actividad`. También debes poder mostrar la información de cada una, específicando el tipo, su valor de felicidad y de estrés mediante `__str)_`.
Ejemplo: `"Hobby - FIFA - Felicidad: 10 - Estres: -10"`

- `class Hobby`: recibe como argumentos `nombre`, `felicidad`, `estres`, estos deben ser pasados al constructor de la clase padre.
- `class Deber`: recibe como argumentos `nombre`, `felicidad`, `estres`, estos deben ser pasados al constructor de la clase padre.

Parte II: Poblar el sistema

Ahora trabajaremos con el archivo `cargar.py`. Una vez que las clases están modeladas debemos poblar el sistema. Debes completar las siguientes funciones:

- `def cargar_estudiantes(ruta_archivo_estudiantes)`: Este método recibe la ruta del archivo de estudiantes, los procesa y guarda sus características como variables, esto está **implementado**. Sólo debes completar instanciando los objetos, según sean de la clase `Alumno` o `Ayudante`, teniendo cuidado en qué atributos recibe cada uno. Una vez que el estudiante está instanciado, debes guardar cada estudiante en el diccionario `estudiantes`, de manera que su *key* sea su `username` y su *value* sea la **instancia recién creada**.
- `def cargar_actividades(ruta_archivo_actividades)`: Este método recibe la ruta del archivo de actividades, y guarda cada dato en variables. Debes completarlo creando las instancias según sea el caso de `Hobby` o de `Deber`. Luego, debes almacenar esta instancia en el diccionario `actividades`, de manera que su *key* sea el `nombre` de la actividad y el *value* sea la **instancia correspondiente**.

Para poder revisar tu trabajo hasta este punto, puedes correr el módulo `cargar.py`, este imprimirá el formato de tus diccionarios y las características de tus clases. Atención, esto no garantiza que no haya otros errores en la implementación.

Parte III: DCCuarentena

Para la última parte debemos trabajar en el archivo `dccuarentena.py`. La clase `DCCuarentena` controla nuestro programa; contiene tanto a los estudiantes como a sus actividades y decide qué actividades entregar. Esta parte consiste en completar dos métodos:

- `class DCCuarentena`: Recibe como argumento un diccionario de `Estudiantes`, con llave el *username* de cada estudiante y valor su respectiva instancia, y un diccionario de `Actividades`, con llave el nombre de la actividad y valor su respectiva instancia. También posee un atributo llamado `usuario_actual`, que indica qué estudiante está usando el sistema, y comienza siendo `None`.
  - `def revisar_identidad(self)`: Verifica que el estudiante ingrese con usuario existente. Si existe entonces guarda la **instancia** del usuario que ingresó en `self.usuario_actual`. **Este método ya está implementado**.
  - `def sugerir_actividad(self)`: Este método sugiere una actividad de acuerdo al estado de `usuario_actual`:
    - Si tiene `hobbies` por hacer, y su felicidad es menor a 50 o su estrés es mayor a 50, se recomendará el primer **hobby** de su lista.
    - En otro caso, recomendará el primer **deber** de su lista.

- Si no quedan **deberes** por hacer, recomendará el primer **hobby** de su lista. Una vez que no queden actividades por hacer, debe imprimir un mensaje similar a: `"¡No te puedo sugerir más actividades! Es hora de descansar :)"`.

Las actividades sugeridas **deben ser realizadas y eliminadas** de la lista correspondiente.

Requerimientos

- (3.00 pts) Clases `Estudiante`, `Alumno` y `Ayudante`.
  - (0.5 pts) Completa el método `realizar_actividad` correctamente.
  - (0.25 pts) Completa la *property* `felicidad` correctamente.
  - (0.25 pts) Completa la *property* `estres` correctamente.
  - (1.0 pt) Completa la clase *Alumno* correctamente.
  - (1.0 pt) Completa la clase *Ayudante* correctamente.
- (1.00 pt) Clases `Actividad`, `Hobby` y `Deber`.
  - (0.5 pts) Completa la clase `Actividad` correctamente.
  - (0.25 pts) Completa la clase `Hobby` correctamente.
  - (0.25 pts) Completa la clase `Deber` correctamente.
- (1.00 pt) Módulo `cargar.py`.
  - (0.5 pts) Completa el método `cargar_estudiantes` correctamente.
  - (0.5 pts) Completa el método `cargar_actividades` correctamente.
- (1.00 pt) Completa el método `sugerir_actividad` de clase `DCCuarentena` correctamente.

Entrega

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