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

25 de Agosto de 2022
Actividad Formativa

Actividad Formativa 2

Programación Orientada a Objetos I

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AF2/`
- **Hora del *push*:** 16:20

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

Ante la vuelta a la presencialidad, el DCC ha decido realizar un DCCPalooza, un evento de varios días en donde habrá mucha música, baile y todo tipo de artistas de alta gama. Es por esto que tú, estudiante de programación avanzada, fuiste seleccionado para realizar y gestionar dicho evento.

Flujo del programa

El programa consiste en una simulación del DCCPalooza, partiendo en un **Menú de Inicio** que dará la opción de ingresar a la simulación. Al seleccionar la opción *Ingresar*, se instanciará un objeto de la clase **DCCPalooza**, la entidad principal del programa que está a cargo de controlar la simulación. Con lo anterior se cargan los datos de los artistas desde el archivo `artistas.csv`, que podrán ser artistas de distintos géneros musicales: **pop**, **rock**, **reggaeton** o **trap chileno**. Con dicha información se generan artistas y se escoge un grupo aleatorio que serán el line-up oficial del DCCPalooza.

Archivos

Archivo de datos

Para esta actividad se entregan los siguientes archivos de datos:

- `artistas.csv`: En este archivo se encuentra la información de todos los artistas del **DCCPalooza**. Cada línea contiene los datos de un artista separados por punto y coma, de la siguiente forma:

  `nombre;genero;dia_presentacion;hit_del_momento`

  Donde cada campo se puede interpretar de la siguiente forma:

`nombre`	corresponde al nombre del artista
`genero`	corresponde al género del artista (Pop, Rock, Trap Chileno y Reggaeton)
`dia_presentacion`	corresponde al día que le corresponde presentarse al artista
`hit_del_momento`	corresponde al nombre de la canción más escuchada del artista

  Puedes suponer que los datos del archivo `artistas.csv` están siempre correctos. No es necesario hacer verificaciones adicionales.

- `suministros.csv`: En este archivo se encuentra la información de todos los suministros del **DCCPalooza**. Cada línea contiene los datos de un suministro separados por punto y coma, de la siguiente forma:

  `nombre;valor_de_satisfaccion`

  Donde cada campo se puede interpretar de la siguiente forma:

`nombre`	corresponde al nombre del suministro
`valor_de_satisfaccion`	corresponde al valor de satifacción que otorga el suministro

Archivos de código

En el directorio de la actividad encontrarás los siguientes archivos con código:

- `cargar_datos.py`: No debes modificarlo Aquí encontrarás la función relacionada con el cargado de datos de los artistas, de nombre `cargar_artistas`.
- `artista.py`: Debes modificarlo Este archivo contiene la clase `Artista` con sus clases hijas correspondientes a `ArtistaPop`, `ArtistaRock`, `ArtistaTrapChileno` y `ArtistaReggaeton`.
- `dccpalooza.py`: Debes modificarlo Aquí encontrarás la clase `DCCPalooza`.
- `suministro.py`: No debes modificarlo Aquí encontrarás la clase `Suministro`.
- `main.py`: No debes modificarlo Este es el archivo principal. Aquí se encuentran las clases que controlan el flujo del programa. Debes correrlo para iniciar la simulación, y **te ayudará a probar tu código.**

- `parametros.py`: No debes modificarlo Este archivo contiene parámetros para la ejecución del programa. Se importa en `main.py`, `artista.py` y `dccpalooza.py` y a través de él puedes acceder a los valores de estos parámetros.

Parte I: Modelando entidades

Antes de poder comenzar con la simulación del DCCPalooza, es importante definir las entidades que formarán parte de ella. Estas son representadas por medio de las clases en `artista.py` y `dccpalooza.py`, las que deberás completar en base a los siguientes requerimientos:

- `class Artista`: Representa a un artista. Incluye los siguientes métodos: Debes modificarlo
  - `def __init__(self, nombre, genero, dia_presentacion, hit_del_momento)`: Este es el inicializador de la clase, y asigna los siguientes atributos: No debes modificarlo

`self.nombre`	Un `str` que representa el nombre del artista.
`self.genero`	Un `str` que representa el género musical del artista. Puede ser pop, rock, reggaeton o trap chileno.
`self.dia_presentacion`	Un `int` que representa el día en el que el artista tendrá su concierto. Tiene un valor de 1, 2 o 3.
`self.hit_del_momento`	Un `str` que representa la canción favorita del público de ese artista.
`self._afinidad_con_publico`	Un `int` que representa que tanta afinidad está teniendo el artista con el público. Es importante que este valor no puede bajar de 0 y tampoco puede superar 100. **Debes implementarlo como una property**
`self._afinidad_con_staff`	Un `int` que representa que tanta afinidad está teniendo el artista con el staff del concierto. Es importante que este valor no puede bajar de 0 y tampoco puede superar 100. **Debes implementarlo como una *property***

¡Ojo! No debes modificar el init pero sí debes setear como property la afinidad con el público y staff Debes modificarlo

  - `def animo(self)`: *Property* que calcula el ánimo del artista como una ponderación de sus atributos `self._afinidad_con_staff` y `self._afinidad_con_publico`. Se calcula de la siguiente manera: Debes modificarlo

$$animo = \lfloor \texttt{afinidad\_con\_publico} * 0.5 \rfloor + \lfloor \texttt{afinidad\_con\_staff} * 0.5 \rfloor$$

  - `def recibir_suministro(self, suministro)`: Esta función se llamará cada vez que quieras atender al artista. Recibe una instancia de `suministro` y deberás modificar el atributo `afinidad_con_staff` dependiendo del valor de satisfacción que tenga el suministro. Deberás sumar el valor de la satisfacción del suministro al atributo `afinidad_con_staff`. Este valor puede aumentar la afinidad o disminuirla. En cada caso, debes imprimir un mensaje indicando lo que pasó. Debes modificarlo
    Por ejemplo:

```
f"{self.nombre} recibió {suministro.nombre} en malas condiciones."
f"{self.nombre} recibió un {suministro.nombre} a tiempo!"
```

- `def cantar_hit(self)`: Esta función se llamará cada vez que quieras que el artista cante su `hit_del_momento`. Esto aumentara la `afinidad_con_publico` en `AFINIDAD_HIT`[1], y deberás imprimir el siguiente mensaje: **Debes modificarlo**

```
f"{self.nombre} está tocando su hit: {self.hit_del_momento}!"
```

- `def __str__(self)`: Este método se llamará cada vez que se imprima en pantalla una instancia del objeto. Puedes elegir cómo mostrar los valores, pero al menos deben contener `nombre`, `genero` y el `ánimo` del artista. **Debes modificarlo**
  Por ejemplo:

```
Nombre: Miley Cyrus
Genero: Pop
Animo: 38
```

- `class ArtistaPop`: Representa al artista de genero **Pop**. Deberás completarla de manera que herede de la clase `Artista`. **Debes modificarlo**
  - `def __init__(self)`: Debe llamar al constructor de la clase padre, y además definir dentro los siguientes atributos:
    - `self.accion = "Cambio de vestuario"`.
    - `self._afinidad_con_publico = AFINIDAD_PUBLICO_POP`
    - `self._afinidad_con_staff = AFINIDAD_STAFF_POP`
  - `def accion_especial(self)`: Este método deberá imprimir un mensaje específico para su género: `f"{self.nombre} hará un {self.accion}"` Este método también deberá aumentar la `afinidad_con_publico` en `AFINIDAD_ACCION_POP`.
  - `def animo(self)`: Además, deberás sobrescribir la *función animo* de la clase padre y esta, además de retornar el valor del property animo del padre, deberá imprimir un mensaje dependiendo de dicho valor. Si este es menor a 10 deberás imprimir:
    `f"ArtistaPop peligrando en el concierto. Animo: {valor_animo}"`.
- `class ArtistaRock`: Representa al artista de género **Rock**. Deberás completarla de manera que herede de la clase `Artista`. **Debes modificarlo**
  - `def __init__(self)`: Debe llamar al constructor de la clase padre, y además definir dentro los siguientes atributos:
    - `self.accion = "Solo de guitarra"`.
    - `self._afinidad_con_publico = AFINIDAD_PUBLICO_ROCK`
    - `self._afinidad_con_staff = AFINIDAD_STAFF_ROCK`
  - `def accion_especial(self)`: Este método deberá imprimir un mensaje específico para su género: `f"{self.nombre} hará un {self.accion}"` Este método también deberá aumentar la `afinidad_con_publico` en `AFINIDAD_ACCION_ROCK`.
  - `def animo(self)`: Además, deberás sobrescribir la *función animo* de la clase padre y esta, además de retornar el valor del property animo del padre, deberá imprimir un mensaje depen-

[1] Los valores que sean escritos `DE_ESTA_MANERA` representan parámetros que serán encontrados en el archivo parametros.py.

diendo de dicho valor. Si este es menor a 5 deberás imprimir:
`f"ArtistaRock peligrando en el concierto. Animo: {valor_animo}"`.

- `class ArtistaTrapChileno`: Representa al artista de género **Trap Chileno**. Deberás completarla de manera que herede de la clase `Artista`. **Debes modificarlo**
  - `def __init__(self)`: Debe llamar al constructor de la clase padre, y además definir dentro los siguientes atributos:
    - `self.accion = "Malianteo"`.
    - `self._afinidad_con_publico = AFINIDAD_PUBLICO_TRAP_CHILENO`
    - `self._afinidad_con_staff = AFINIDAD_STAFF_TRAP_CHILENO`
  - `def accion_especial(self)`: Este método deberá imprimir un mensaje específico para su género: `f"{self.nombre} hará un {self.accion}"` Este método también deberá aumentar la `afinidad_con_publico` en `AFINIDAD_ACCION_TRAP_CHILENO`.
  - `def animo(self)`: Además, deberás sobrescribir la *función animo* de la clase padre y esta, además de retornar el valor del property animo del padre, deberá imprimir un mensaje dependiendo de dicho valor. Si este es menor a 20 deberás imprimir:
    `f"ArtistaTrapChileno peligrando en el concierto. Animo: {valor_animo}"`.
- `class ArtistaReggaeton`: Representa al artista de género **Reggaeton**. Deberás completarla de manera que herede de la clase `Artista`. **Debes modificarlo**
  - `def __init__(self)`: El constructor debe ser llamado del de la clase padre, y además definir dentro los siguientes atributos
    - `self.accion = "Perrear"`.
    - `self._afinidad_con_publico = AFINIDAD_PUBLICO_REGGAETON`
    - `self._afinidad_con_staff = AFINIDAD_STAFF_REGGAETON`
  - `def accion_especial(self)`: Este método deberá imprimir un mensaje específico para su género: `f"{self.nombre} hará un {self.accion}"` Este método también deberá aumentar la `afinidad_con_publico` en `AFINIDAD_ACCION_REGGAETON`.
  - `def animo(self)`: Además, deberás sobrescribir la *función animo* de la clase padre y esta, además de retornar el valor del property animo del padre, deberá imprimir un mensaje dependiendo de dicho valor. Si este es menor a 2 deberás imprimir:
    `f"ArtistaReggaeton peligrando en el concierto. Animo: {valor_animo}"`

Dentro del archivo `dccpalooza.py` se encuentra:

- `class DCCPalooza`: Representa la entidad que administra el correcto funcionamiento del DCCPalooza. Contiene los métodos:
  - `def __init__(self)`: Inicializador de la clase, contiene los siguientes atributos: **No debes modificarlo**

`self.artista_actual`	Un `str` que representa al artista que esta actualmente tocando en el concierto.
`self.__dia`	Un `int` que funciona como contador del progreso de la simulación. Debe verificar que se mantenga siempre positivo y con un incremento ascendente.
`self.line_up`	Un `list` que contiene instancias de las clases `ArtistaPop`, `ArtistaRock`, `ArtistaTrapChileno` y `ArtistaReggaeton` que tocaran cada día.
`self.cant_publico`	Un `int` que representa la cantidad de publico al final de cada día.
`self.artistas:`	Una `list` que contiene instancias de las clases `ArtistaPop`, `ArtistaRock`, `ArtistaTrapChileno` y `ArtistaReggaeton` de todo el concierto.
`self.prob_evento`	Una `float` que representa la probabilidad de que ocurra algun evento durante el concierto.
`self.suministros`	Una `list` que contiene instancias de la clase `Suministro`.

- `def exito_del_concierto(self)`: Es una property que se llama cada vez que pasa un día, para verificar si se cumplen las condiciones de término de la simulación. No debes modificarlo
- `def funcionando(self)`: Es una property que verifica si el concierto está funcionando. Retorna `True` si es que aún no terminan los días y hay gente suficiente, y `False` en caso contrario No debes modificarlo
- `def imprimir_estado(self)`: Imprime en consola información importante del estado del DCCPalooza. Debe imprimir algo como: No debes modificarlo

```
Día: 1
Cantidad de personas: 2398
Artistas:
```

- `def ingresar_artista(self, artista)`: Este método agrega un objeto de cualquiera de las subclases de `Artista` a la lista `self.artistas`, e imprime alguna de sus características en pantalla. No debes modificarlo
- `def despedir_artista(self, artista)`: Contrario al método anterior, este remueve una instancia de `Artista` desde `self.artistas`, e imprime un mensaje en consola informándolo. No debes modificarlo
- `def nuevo_dia(self)`: Este método simula el paso de un día. Primero debe verificar la condición de término llamando a `self.exito_del_concierto()`; si está siendo exitoso, debes aumentar el atributo `self.__dia` en 1, y en caso de que el día sea menor o igual 3 (ya habiendo sumado la unidad) se imprime que comienza un nuevo día. Debes modificarlo
- `def ejecutar_evento(self)`: Este método debe encargarse verificar la ocurrencia de algún evento y ejecutar su efecto en dicho caso. Si se cumple la probabilidad contenida en `self.prob_evento` deberás escoger alguno de los siguientes eventos de forma ALEATORIA: Debes modificarlo
  - **Lluvia**: Deberás disminuir la afinidad con el público del `artista` actual en una cantidad `AFINIDAD_LLUVIA` e imprimir un mensaje que lo indique
  - **Terremoto**: Deberás disminuir la cantidad de público `DCCPalooza` en una cantidad `PUBLICO_EVENTO` e imprimir un mensaje que lo indique

- **Ola de calor**: Deberás disminuir la afinidad con el público del `artista` actual en una cantidad `AFINIDAD_OLA_CALOR` e imprimir un mensaje que lo indique y deberás disminuir la cantidad de público de `DCCPalooza` en una cantidad `PUBLICO_OLA_CALOR` e imprimir un mensaje que lo indique

Simulación

Una vez definidas las entidades, puedes ejecutar el archivo `main.py`. En este se ejecuta todo el código que desarrollaste, y te permitirá dar vida al DCCPalooza mediante una simulación. No hay nada más que desarrollar aquí y puedes usarlo para probar que tu código funciona correctamente.

Notas

- Recuerda que la ubicación de tu entrega es en tu **repositorio personal**. Verifica que no estés trabajando en el **Syllabus**.
- Se recomienda completar la actividad en el orden del enunciado.
- Para las *properties* pueden crear nuevos métodos o modificar los existentes si creen que es necesario.
- 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.

Objetivos de la Actividad

Los objetivos de la actividad están ordenados según su prioridad en la actividad. Se recomiendo resolver la actividad respetando ese orden.

1. artista.py
   - Implementar correctamente la clase Artista y sus *properties*.
   - Implementar la herencia y métodos de las clases ArtistaPop, ArtistaRock, ArtistaTrapChileno, Artista Reggaeton.
2. dccpalooza.py
   - Implementar correctamente el método `nuevo_dia` de la clase `DCCPalooza`.
   - Implementar correctamente el método `ejecutar_evento` de la clase `DCCPalooza`.