Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2022-1

24 de Marzo 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*:** 16:40

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

Dos años han pasado en los cuales una infinidad de estudiantes han reportado que les ha faltado la música y el baile. Es por esto, que los tiempos de silencio y aburrimiento han acabado y el evento más esperado del año ha vuelto.
Se te ha encargado realizar y gestionar el DCConcierto, una instancia de varios días en los cuales se podrá disfrutar de música, baile y artistas de primer nivel, los cuales se ceñirán a tus indicaciones ya que saben que no hay mejor DCCordinador que tú.

Flujo del programa

El programa consiste en una simulación del DCConcierto, 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 **DCConcierto**, la entidad principal del programa que está a cargo de controlar la simulación. Con lo anterior se cargan los datos de los artistas desde una base de datos almacenada en el archivo `artistas.csv`, que podrán ser artistas de distintos géneros musicales **pop**, **rock**, **reggaeton** o **rap**. Con dicha información se generan artistas y se escoge un grupo aleatorio que serán el line-up oficial del DCConcierto.

Archivos

Archivo de datos

- `artistas.csv`: donde los valores se interpretan de esta manera:

`Nombre`	corresponde al nombre del artista
`Género`	corresponde al género del artista (Pop, Rock, Rap y Reggaeton)
`Día`	corresponde al día que le corresponde presentarse al artista
`Hit del Momento`	corresponde al nombre de la canción más escuchada del artista
`Afinidad Público` `Afinidad Staff`	valores entre 0 y 100 que indican los niveles de afinidad con el público y staff del concierto, donde 100 es el máximo (buena relación) mientras que 0 es el mínimo (mala relación).

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

Archivos de código

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

- `cargar_datos.py`: Aquí encontrarás la función relacionada con el cargado de datos de los artistas, de nombre `cargar_artistas`. **Debes modificarlo**
- `artista.py`: Este archivo contiene la clase `Artista` con sus clases hijas correspondientes a `ArtistaPop`, `ArtistaRock`, `ArtistaRap` y `ArtistaReggaeton`. **Debes modificarlo**
- `dcconcierto.py`: Aquí encontrarás la clase `DCConcierto`. **Debes modificarlo**
- `suministro.py`: Aquí encontrarás la clase `Suministro`. **No debes modificarlo**
- `main.py`: 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.** **No debes modificarlo**
- `parametros.py`: Este archivo contiene parámetros para la ejecución del programa. Se importa en `main.py`, `artista.py` y `dcconcierto.py` y a través de él puedes acceder a los valores de estos parámetros. **No debes modificarlo**

Parte I: Modelando entidades

Antes de poder comenzar con la simulación del DCConcierto, es importante definir las entidades que formarán parte de ella. Estas son representadas por medio de las clases en `artista.py` y `dcconcierto.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 genero musical del artista, puede ser pop, rock, reggaeton, o rap.
`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 esta teniendo el artista con el público, es importante que no pueda ser negativo y tampoco más grande que 100. **Debe implementarse como una property**
`self._afinidad_con_staff`	Un `int` que representa que tanta afinidad esta teniendo el artista con el staff del concierto, es importante que no pueda ser negativo y tampoco más grande que 100. **Debe implementarse como una *property***

  - `def animo(self)`: *Property* que calcula el animo del artista, como una ponderación de sus atributos `self._afinidad_con_staff` y `self._afinidad_con_publico`. 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. Deberas 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:

```
print(f"{self.nombre} recibió {suministro.nombre} en malas condiciones.")
print(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**

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

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

- `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 `mood` 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 acción_especial(self)`: este método deberá imprimir un mensaje específico para su genero, `print(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 heredar la *property* `animo` de la clase padre y esta deberá imprimir un mensaje dependiendo del valor del animo. Si este es menor a 10 deberás imprimir: `print(f"ArtistaPop peligrando en el concierto. Animo: {self.animo}")`
- `class ArtistaRock`: Representa al artista de genero **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 acción_especial(self)`: este método deberá imprimir un mensaje específico para su género, `print(f"{self.nombre} hará un {self.accion}")` Este método también deberá aumentar la `afinidad_con_publico` en `AFINIDAD_ACCION_ROCK`.
  - `def animo(self)`: Ademas, deberás heredar la *property* `animo` de la clase padre y esta deberá imprimir un mensaje dependiendo del valor del ánimo. Si este es menor a 5 deberás imprimir: `print(f"ArtistaRock peligrando en el concierto. Animo: {self.animo}")`
- `class ArtistaRap`: Representa al artista de genero **Rap**. 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 = "Doble tempo"`.
    - `self._afinidad_con_publico = AFINIDAD_PUBLICO_RAP`

    - `self._afinidad_con_staff = AFINIDAD_STAFF_RAP`
  - `def acción_especial(self)`: este método deberá imprimir un mensaje específico para su genero, `print(f"{self.nombre} hará un {self.accion}")` Este método también deberá aumentar la `afinidad_con_publico` en `AFINIDAD_ACCION_RAP`.
  - `def animo(self)`: Además, deberás heredar la *property* `animo` de la clase padre y esta deberá imprimir un mensaje dependiendo del valor del animo. Si este es menor a 20 deberás imprimir: `print(f"ArtistaRap peligrando en el concierto. Animo: {self.animo}")`
- `class ArtistaReggaeton`: Representa al artista de genero **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 acción_especial(self)`: este método deberá imprimir un mensaje específico para su genero, `print(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 heredar la función `mood` de la clase padre y esta deberá imprimir un mensaje dependiendo del valor del animo. Si este es menor a 2 deberas imprimir: `print(f"ArtistaReggaeton peligrando en el concierto. Animo: {self.animo}")`

Dentro del archivo `dcconcierto.py` se encuentra:

- `class DCConcierto`: Representa la entidad que administra el correcto funcionamiento del concierto. 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. **Debe implementarse como una property**.
`self.line_up`	Un `list` que contiene instancias de las clases `ArtistaPop`, `ArtistaRock`, `ArtistaRap` 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`, `ArtistaRap` 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 DCConcierto. 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 las condiciones de término llamando a `self.exito_del_concierto()`; si está siendo exitoso 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 `DCConcierto` 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 `DCConcierto` 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 DCConcierto 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

- artista.py
  - Implementar correctamente la clase Artista y sus *properties*.
  - Implementar la herencia y métodos de las clases ArtistaPop, ArtistaRock, ArtistaRap, Artista Reggaeton.
- dcconcierto.py
  - Implementar correctamente el método `nuevo_dia` de la clase `DCConcierto`.
  - Implementar correctamente el método `ejecutar_evento` de la clase `DCConcieto`.