Bases de Datos II (2024) - Tarea 1

Diego Alvarez S -- 10/09/2024

Objetivo

Aplicar conocimientos de programación en Python, aplicando el paradigma de programación orientada a objetos.

Descripción

Implementar la estructura y lógica de un parque de diversiones. En esta estructura deben contemplarse principalmente atracciones y visitantes.

Requerimientos

1. Crear la clase `Visitante`:
   Esta clase representa a cada visitante del parque y debe tener los siguientes atributos:

   - `nombre`: Nombre del visitante.
   - `edad`: Edad del visitante.
   - `altura`: Altura del visitante en cm.
   - `dinero`: Cantidad de dinero disponible para gastar en el parque.
   - `tickets`: Lista de tickets comprados por el visitante.

   Métodos:

   - `comprar_ticket(self, atraccion)`: Agrega un ticket para una atracción específica a la lista de tickets del visitante, y deduce el costo del ticket de su dinero disponible.
   - `entregar_ticket(self, atraccion)`: Entrega el ticket correspondiente a la atracción si el visitante tiene uno, y lo elimina de la lista de tickets. En caso de no tener ticket, imprime un mensaje de error.
   - `hacer_cola(self, atraccion)`: El visitante se pone en la cola de una atracción específica. Sólo se puede estar en la cola de un juego a la vez, por lo que usar este método cuando el visitante ya está en una cola, hará que pierda su lugar.

2. Crear la clase `Atraccion`:
   Esta clase representa cada atracción del parque y debe tener los siguientes atributos:

   - `nombre`: Nombre de la atracción.
   - `capacidad`: Número máximo de personas por ronda.
   - `duración`: Duración de la atracción en minutos.
   - `estado`: Estado actual de la atracción (activo o fuera de servicio).
   - `cola`: Lista de visitantes esperando su turno para subirse a la atracción.

   Métodos:

   - `iniciar_ronda(self)`: Verifica si la atracción está activa y si hay suficientes visitantes en la cola. Luego, inicia la ronda haciendo entrar a los visitantes hasta alcanzar la capacidad máxima.
   - `comenzar_mantenimiento(self)`: Cambia el estado de la atracción a "fuera de servicio" para mantenimiento.
   - `finalizar_mantenimiento(self)`: Cambia el estado de la atracción a "activo" una vez finalizado el mantenimiento.

3. Crear la clase `Ticket`:
   Esta clase representa un ticket comprado por un visitante y debe tener los siguientes atributos:

   - `numero`: Número de identificación del ticket.
   - `atraccion`: Nombre de la atracción asociada al ticket.
   - `precio`: Precio pagado por el ticket.
   - `fecha_compra`: Fecha en la que se compró el ticket.

   Nota: Cada ticket solo puede utilizarse una vez y está vinculado a una atracción específica.

4. **Crear la clase `Parque`:**
   Esta clase representa el parque de diversiones y debe tener un nombre, una lista de atracciones y un sistema de gestión de entradas.

   Atributos:

   - `nombre`: Nombre del parque.
   - `juegos`: Lista de atracciones disponibles en el parque.

   Métodos:

   - `consultar_juegos_activos(self)`: Devuelve una lista de todas las atracciones que están activas (en estado "activo").
   - `cobrar_ticket(self, visitante, atraccion)`: Procesa el cobro de un ticket para una atracción específica. Deducir el precio del saldo del visitante, y agregar el ticket a la lista de tickets del visitante.

5. **Implementar clases para los siguientes tipos de atracciones:**

   - **`AtraccionInfantil`:**
     Solo permite el acceso a visitantes de 10 años o menos. Debe heredar de la clase `Atracción`. Se recomienda agregar un método `verificar_restricciones(self, visitante)` que valide la edad del visitante antes de permitirle acceder a la atracción.

   - **`MontañaRusa`:**
     Solo permite el acceso a visitantes que midan al menos 140 cm. Además de los atributos básicos de la clase `Atracción`, debe tener:

     - `velocidad_maxima`: Velocidad máxima de la montaña rusa.
     - `altura_maxima`: Altura máxima alcanzada.
     - `extension`: Longitud total del recorrido en metros.

     Debe incluir el método `verificar_restricciones(self, visitante)` para validar la altura del visitante antes de permitirle el acceso.

6. **Implementar resumen de ventas diario:** Añade el método `resumen_de_ventas(self, dia)` a la clase `Parque`. Este método debe generar un resumen de las ventas del día especificado, mostrando cuántos tickets se vendieron para cada atracción y la cantidad de dinero recaudada. Debe incluir un total de ingresos del día. Si es necesario, añade nuevos atributos a las clases ya existentes.

7. **Implementar la clase `VisitanteVIP`:**
   Los visitantes pueden comprar un pase diario que los convierte en visitantes VIP. Los VIP tienen los siguientes beneficios:

   - Acceso sin costo hasta un máximo de dos veces por cada atracción.

   - Prioridad en la cola para subir a las atracciones (pero se debe respetar que no más del 30% de los asistentes por ronda sean VIP).

   Implementa la clase `VisitanteVIP` con las características mencionadas. Además, modifica el método método `iniciar_ronda` de las atracciones, que debe implementarse una lógica para asegurar que no más del 40% de los participantes de una ronda sean VIP.

8. **Implementar script de prueba:**

   Implementar un script que simule la creación de un parque con sus atracciones y visitantes, de forma que se utilicen todas clases y métodos implementados.

Evaluación

- Cada uno de los requerimientos entregará 10 puntos al ser desarrollado por completo. En caso de ser desarrollado parcialmente, se evaluará según el grado de avance.
- El orden en el código, uso de buenas prácticas y legibilidad del código se evaluará con un máximo de 5 puntos. Recuerda que usar ruff puede ser útil para esto.
- El uso correcto de tipos se puntuará con un máximo de 5 puntos. Para verificar esto se utilizará la herramienta mypy de la siguiente manera:
  - Ejecución de `mypy` sin errores: 4 puntos.
  - Ejecución de `mypy --strict` sin errores: 5 puntos.
- El código deberá entregarse en un repositorio de GitHub, en el que se pueda ver el historial de commits. En caso de no querer dejar el repositorio público, se puede dar acceso al usuario dialvarezs. Se evaluará el último commit antes de la fecha de entrega. En la plataforma Pregrado Virtual debe indicarse la URL del repositorio.
- La estructura de el código es libre, siempre que respeten los nombres de las clases y métodos indicados.
- Si se detecta un alto nivel de similitud entre tareas entregadas, se interrogará a los involucrados sobre su dominio del contenido. En caso de confirmarse plagio, se evaluará con nota mínima.