F1

Objetivo

- Diferenciar entre definición y creación de objetos.
- Creación de un programa principal que cumpla el objetivo solicitado.
- Aplicar de forma correcta los modificadores de visibilidad para no violar el encapsulamiento.
- Modelar su programa en UML.
- Documentar su programa utilizando Javadoc.

Introducción

La administración de la Fórmula 1 está solicitando ayuda de programadores para hacer un experimento en las ventas de tickets para su campeonato mundial en Italia. Debido a la inconformidad de los fans en los eventos anteriores por lo rápido que se vendieron los boletos se decidió implementar un nuevo algoritmo de compra aleatorio que sea justo y de las mismas posibilidades a todos.

Para poder comprar los boletos se debe hacer lo siguiente:

1. Solicitar una compra de un boleto. Para hacer esto se debe ingresar el nombre, dpi, cantidad de boletos a comprar y presupuesto máximo.
2. Cuando se hace la solicitud de compra se genera un número aleatorio de 1 a 28,000.
3. Ese número generado es el “ticket” y, para determinar si un ticket puede comprar boletos se debe generar 2 números aleatorios (a y b) adicionales dentro del mismo rango (1 a 15,000). Se debe validar que la suma del ticket y los otros 2 números sea impar. Si esto se cumple, el ticket es apto para comprar boletos.
4. Debido a que este proceso es más complejo solo se piensa utilizar como piloto para vender 60 entradas. distribuidas de forma igualitaria en 3 localidades diferentes:
    a. Localidad 1 - La más alejada del escenario pero la más barata con un precio de $300.
    b. Localidad 5 - En medio y con una mejor vista al escenario pero con un precio de $565.
    c. Localidad 10 - Hasta adelante, la mejor vista del concierto pero la más cara con un precio de $1495.
5. Cuando un ticket es apto para compra se debe seleccionar de forma aleatoria la localidad a asignarle dentro de las 3 disponibles.

6. Al tener la localidad definida se deben realizar 3 validaciones:
    a. Validar espacio: no es posible vender los tickets si la localidad no tiene espacio, es decir si ya se vendieron los 20 espacios disponibles.
    b. Validar disponibilidad de los boletos deseados: Se debe validar que exista espacio para la cantidad de boletos que el comprador quiere comprar. Si no es así se le vende la mayor cantidad posible sin exceder el límite deseado.
    c. Validar precio: si el precio de la sección es mayor al presupuesto no se puede comprar el boleto y se termina el proceso para ese comprador.
7. Si pasan las validaciones anteriores, se le vende al usuario la cantidad de boletos determinada.

Ejercicio 1 - Overloading y relaciones entre clases

Ejercicio

Análisis

Responda las siguientes preguntas:

1. ¿Qué propiedades y métodos tendrá cada clase?
2. ¿Qué tipo deben tener las propiedades y métodos de cada clase?
3. ¿Cuáles deben ser los modificadores de visibilidad de los miembros en cada clase?
4. ¿Qué parámetros serán requeridos por los métodos en sus clases?
5. ¿Cómo proveerá de valores iniciales a sus objetos? ¿Qué valores iniciales les asignará?

Diseño

Desarrolle un diagrama de clases que muestre los miembros de cada clase y que establezca una asociación entre ellas. Incluya en su diagrama al *driver program*. Use anotaciones (recuadros de texto) en su diagrama para aclarar cualquier información que considere necesaria.

Programa

Desarrolle un sistema que contenga un *driver program* con instancias de: localidad y comprador como mínimo. Así mismo debe mostrar un menú con las siguientes opciones:

1. Nuevo comprador: reemplaza la instancia actual del comprador activo.
2. Nueva solicitud de boletos: permite al comprador actual participar para comprar boletos.
3. Consultar disponibilidad total: se muestran cuántos boletos se han vendido en cada localidad y cuantos boletos hay disponibles para las 3 localidades.
4. Consultar disponibilidad individual: Se le pide al usuario que defina una localidad y se muestran solo los boletos disponibles para la localidad seleccionada.
5. Reporte de caja: Mostrar cuánto se ha generado de dinero dados los boletos vendidos en las 3 localidades.
6. Salir: termina la ejecución del programa.

Documente su código utilizando las reglas previstas por Javadoc
(https://es.wikipedia.org/wiki/Javadoc)

Universidad del Valle de Guatemala
CC2008 - Introducción a la Programación Orientada a Objetos
Semestre II, 2023

Ejercicio 1 - Overloading y relaciones entre clases

Material a entregar en Canvas

- Archivo PDF con el análisis y diseño (No incluya URL del diagrama sino en vez adjúntelo en el pdf o como un archivo externo).
- Url al repositorio **público** en GitHub conteniendo todos los archivos .java.

Evaluación

- **Análisis** (30 puntos)
  - **(5 puntos)** Se listan correctamente todos los requisitos funcionales del sistema a desarrollar, punto por punto.
  - **(5 puntos)** Se describe el propósito de cada una de las clases identificadas. La descripción es lógica y coherente con lo que hace la clase en el sistema que modela y resuelve el problema presentado.
  - **(10 puntos)** Se describe el propósito de cada atributo de cada clase. La descripción es suficiente para comprender la razón de la creación del atributo y su importancia como propiedad de las instancias (o clase).
  - **(10 puntos)** Se describe el propósito de la creación de cada método de cada clase. La descripción es suficiente para comprender qué acciones realizará cada método, qué información requerirá para funcionar y qué resultados producirá.
- **Diseño** (30 puntos)
  - **(8 puntos)** La representación de las clases coincide con el estándar del lenguaje UML.
  - **(12 puntos)** Están representados cada uno de los componentes de la clase de manera correcta y coinciden con lo descrito en el análisis.
  - **(10 puntos)** Las relaciones entre las clases están correctas.
- **Programa** (40 puntos)
  - **(10 puntos)** La cantidad de clases identificadas es suficiente para resolver el problema planteado.
  - **(20 puntos)** El problema se resolvió empleando programación orientada a objetos. Se almacenó la información necesaria creando y llenando objetos de las clases identificadas.
  - **(10 puntos)** El código fuente está suficientemente comentado utilizando Javadoc para comprender que se hace, incluyendo el encabezado con la información del creador, descripción del programa y crédito a toda fuente de información que haya aportado al desarrollo del programa. Preferiblemente, que también incluya fecha de creación y fecha de última modificación.