PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

PROGRAMACIÓN 3
2da. práctica (tipo b)
(Segundo Semestre 2024)

Indicaciones Generales:

- Tiempo estimado: 1h 50 minutos
- Se les recuerda que, de acuerdo al reglamento disciplinario de nuestra institución, constituye una falta grave copiar del trabajo realizado por otro estudiante o cometer plagio para el desarrollo de esta práctica.
- Está permitido el uso de apuntes de clase, diapositivas, ejercicios de clase y código fuente. (Debe descargarlos antes de iniciar con la solución del enunciado)
- No está permitido el uso de entornos de desarrollo integrado (IDEs).
  Debe utilizar un editor de texto: Notepad++, Sublime, etc. (No es posible utilizar VISUAL STUDIO CODE).
- Está permitido el uso de Internet (únicamente para consultar páginas oficiales de Microsoft y Oracle). No obstante, está prohibida toda forma de comunicación con otros estudiantes o terceros.

PARTE PRÁCTICA (20 puntos)

PUEDE UTILIZAR MATERIAL DE CONSULTA.
Se considerará en la calificación el uso de buenas prácticas de programación (aquellas vistas en clase).

PREGUNTA 1 (20 puntos)

TELEPASSES S.A. es una compañía nueva que desea ingresar al rubro de gestión de eventos musicales en el mercado peruano. Para ello, lo ha contratado a Ud. para la programación de este módulo. Se le solicita realizar la programación en JAVA o C# que de soporte a toda la lógica de negocio y a la impresión del reporte.

Las empresas encargadas de organizar **eventos** son conocidas como "***productoras***". Una productora tiene asociados varios eventos, los cuales pueden ser de dos tipos: "**obras teatrales**" y "**conciertos**". En el caso de las productoras, se desea registrar únicamente el nombre de la misma. En el caso de los eventos, indistintamente del tipo, se requiere registrar el nombre, el costo de realización y el tipo de público para el cual está dirigido el evento (el cual debe ser una letra de tipo de dato **char**): 'A' si el evento es para adultos, 'J' si el evento es para jóvenes, 'N' si el evento es para niños y 'T' si el evento es para todo público. Asimismo, los eventos deben tener un identificador único que sea correlativo y que debe ser generado por el programa de software de forma automática.

En un evento pueden participar uno o varios **artistas**. Con respecto a los artistas, se requiere registrar el nombre y la nacionalidad. La estructura de clases que represente a un artista debe considerar que un artista puede estar conformado por otros artistas. Este es el caso por ejemplo de la banda de rock LIBIDO, que puede ser considerada como el artista de un evento. Sin embargo, LIBIDO a su vez está conformado por otros artistas tales como: "SALIM VERA", "MANOLO HIDALGO", "ANTONIO JAUREGUI" y "JEFFRY FISCHMAN", que también podrían participar como artistas en otros eventos. Es decir, tanto LIBIDO, así como "SALIM VERA", "MANOLO HIDALGO", "ANTONIO JAUREGUI" y "JEFFRY FISCHMAN" pueden ser representados como artistas.

Un evento se realiza en un **local**, y la empresa está trabajando únicamente con 4 tipos específicos de locales: TEATRO, AUDITORIO, ANFITEATRO o ESTADIO. Sobre el local, se requiere registrar el nombre, la dirección, la capacidad del local y el tipo de local.

Un evento se realiza en una o varias funciones. Para cada **función** de un evento, se requiere especificar la fecha y la hora de la misma.

Con respecto a los eventos que son de tipo obras teatrales se requiere además de registrar los datos generales de un evento, que se registre el número de actos que tiene la obra y si la obra se requiere o no escenografía. Con respecto a los eventos que son de tipo concierto, se requiere registrar además de los datos generales de un evento, el tipo de concierto (el cual puede ser "ACUSTICO", "DE_SALA", "RECITAL" o "SINFONICO") y si existe o no permiso para que los asistentes puedan grabar.

Las clases que representan a los locales, las funciones de los eventos, los artistas y los eventos obligatoriamente deben implementar una función que retorne datos. Esta función se llamará `devolverDatos()`. Los locales deberán devolver el texto "LOCAL: " seguido del nombre, el texto " – CAPACIDAD:", la capacidad del local y un salto de línea. Las funciones de los eventos deberán devolver el texto "FECHA:", seguido de la fecha de la función en formato "dd-MM-yyyy", la hora y un salto de línea. Los artistas deberán devolver el texto "ARTISTA: " seguido del nombre del mismo. Los eventos deberán devolver datos dependiendo del tipo de evento. Todos los eventos (sin importar el tipo) deberán devolver una cabecera de reporte que contiene el texto "EVENTO Nro. ", seguido del identificador, seguido de un separador de datos ("----"), los datos de los artistas asociados al evento, los datos del local del evento, los datos de las funciones del evento. Posterior a ello, si se trata de un evento de tipo concierto, se agregará el texto "CONCIERTO – TIPO:" seguido del tipo de concierto, el texto "PERMISO GRABACION: ", el indicador de si se permite o no grabación

para los asistentes y un separador de datos ("----") como cierre. En caso se trate de un evento de tipo obra teatral, se agregará el texto "OBRA TEATRAL – NUM. ACTOS: ", seguido del número de actos, el texto "REQ. ESCENOGRAFIA: ", el indicador de si requiere o no escenografía y un separador de datos ("----") como cierre.

Se ha realizado un análisis preliminar y se han detectado algunas clases de manera genérica:

- **InfoProvider**: clase de tipo interface que define la obligación para los locales, artistas, funciones de eventos y eventos de devolver datos a través del método: "`String devolverDatos();`". En el caso de los eventos, este método hace uso del método que devuelve la cabecera del reporte.
- **IDataProvider**: clase de tipo interface que define la obligación a todos los eventos de devolver una cabecera de reporte con los principales datos del evento a través del método: "`String devolverCabecera()`".
- **Productora**: clase que define los atributos y comportamiento de una productora. Además, define el método "`String consultarDatosEvento(int indice)`" que devuelve los datos del evento de un índice en particular asociado a la productora y el método "`String consultarObrasTeatrales()`" que devuelve los datos de todos los eventos de tipo obras teatrales asociados a la productora.

Con el objetivo de implementar todo lo indicado en el enunciado, se ha elaborado el método `main` tal como se presenta en la Figura 01, el cual genera la salida que se presenta en la Figura 02. Utilice los distintos métodos solicitados optimizando el código fuente para generar la impresión del reporte.

```
C:\Users\PC_Lima\OneDrive\LP2\2024-2\Laboratorio 02\Solucion>java Principal
EVENTO Nro. 1
------------------------------------------------------------
NOMBRE: LIBIDO - LA REUNION - TIPO PUBLICO: T
ARTISTA 1: ARTISTA: LIBIDO
LOCAL: TEATRO NOS - CAPACIDAD:600
FUNCION 1: FECHA:14-09-2024 20:00
FUNCION 2: FECHA:15-09-2024 16:30
CONCIERTO - TIPO:ACUSTICO
PERMISO GRABACION: SI
------------------------------------------------------------
EVENTO Nro. 2
------------------------------------------------------------
NOMBRE: YO CINNA - TIPO PUBLICO: A
ARTISTA 1: ARTISTA: SALVADOR DEL SOLAR
LOCAL: TEATRO CANOUT - CAPACIDAD:1000
FUNCION 1: FECHA:31-10-2024 17:30
FUNCION 2: FECHA:31-10-2024 20:30
OBRA TEATRAL - NUM. ACTOS: 5
REQ. ESCENOGRAFIA: SI
------------------------------------------------------------

C:\Users\PC_Lima\OneDrive\LP2\2024-2\Laboratorio 02\Solucion>
```

Cabecera del reporte del evento

Cabecera del reporte del evento

Fig. 02. Reporte generado por el programa solicitado

Es indispensable que se encuentren programados todos los atributos de las clases y sus relaciones que permiten la impresión del reporte. **Con respecto a los constructores, *getters* y *setters*, puede programar solo aquellos que son requeridos para la salida del reporte**.

Se ha compartido la clase Principal.java y Principal.cs en PAIDEIA. Una vez finalizada la evaluación, suba todo su código fuente a PAIDEIA en un archivo **.zip** que tenga como nombre su código PUCP.

Aspectos a considerar para evitar descuento de puntos:

• Colocar su nombre completo y código PUCP únicamente en la clase Principal.
• Nombrar correctamente a las clases, atributos y métodos.
• Las clases deben programarse de forma ordenada y estar estructuradas según lo visto en clase.
• Utilice variables tipificadas (no se permitirá el uso de var y object).
• No realice importaciones innecesarias de clases (no utilice * para las importaciones).
• Utilice los principios de POO: abstracción (en clases y métodos donde sea requerido), polimorfismo y encapsulamiento.
• Utilice correctamente ámbitos, clases / métodos abstractos(as), clases de tipo interface, enumerados donde sea requerido.
• Utilice la etiqueta "@Override" donde corresponda.
• Respetar el orden en la estructura de una clase.
• Emplear un archivo por clase.

• El programa debe compilar correctamente, se descontarán puntos por errores de compilación.
• Utilice los métodos solicitados para la impresión del reporte (**optimice el código**).

```
import java.util.ArrayList;
import java.time.LocalTime;
import java.text.SimpleDateFormat;
class Principal{
	public static void main(String[] args) throws Exception{
		//Creamos un objeto para manejo de fechas
		SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
		//Creamos una productora
		Productora prod1 = new Productora("RUIDOS RITMICOS PRODUCCIONES");
		//Creamos un artista (una banda de rock)
		Artista artGrupo = new Artista("LIBIDO","PERUANA");
		//Creamos a los integrantes de la banda (quienes también son artistas)
		Artista artInt1 = new Artista("SALIM VERA","PERUANA");
		Artista artInt2 = new Artista("MANOLO HIDALGO","PERUANA");
		Artista artInt3 = new Artista("ANTONIO JAUREGUI","PERUANA");
		Artista artInt4 = new Artista("JEFFRY FISCHMAN","PERUANA");
		//Asociamos los integrantes a la banda
		artGrupo.agregarArtista(artInt1);
		artGrupo.agregarArtista(artInt2);
		artGrupo.agregarArtista(artInt3);
		artGrupo.agregarArtista(artInt4);
		//Creamos un artista de obras teatrales
		Artista actor = new Artista("SALVADOR DEL SOLAR","PERUANA");
		//Creamos locales
		Local local1 = new Local("TEATRO NOS","Av. Camino Real 1037 - San
Isidro",600,TipoLocal.TEATRO);
		Local local2 = new Local("TEATRO CANOUT", "Av. Petit Thouars 4550 - Miraflores",
1000,TipoLocal.TEATRO);
		//Creamos un evento de tipo concierto rock
		Evento evento1 = new Concierto("LIBIDO - LA
REUNION",1500000.00,'T',TipoConcierto.ACUSTICO,true);
		//Asociamos los datos del evento
		evento1.agregarArtista(artGrupo);
		evento1.setLocal(local1);
		evento1.agregarFuncion(new Funcion(sdf.parse("14-09-
2024"),LocalTime.of(20,00,00)));
		evento1.agregarFuncion(new Funcion(sdf.parse("15-09-
2024"),LocalTime.of(16,30,00)));
		//Creamos un evento de tipo obra teatral
		Evento evento2 = new ObraTeatral("YO CINNA",5500.00,'A',5,true);
		//Asociamos los datos del evento
		evento2.agregarArtista(actor);
		evento2.setLocal(local2);
		evento2.agregarFuncion(new Funcion(sdf.parse("31-10-
2024"),LocalTime.of(17,30,00)));
		evento2.agregarFuncion(new Funcion(sdf.parse("31-10-
2024"),LocalTime.of(20,30,00)));
		//Asociamos los eventos a la productora
		prod1.agregarEvento(evento1);
		prod1.agregarEvento(evento2);
		//Imprimimos datos de los eventos
		System.out.print(prod1.consultarDatosEvento(0));
		System.out.print(prod1.consultarObrasTeatrales());
	}
}
```

Fig. 01. Código fuente de la clase Principal

Utilice el operador **instanceof** (JAVA) o **is** (C#) para comprobar si un objeto es una instancia de un tipo específico (una clase o interfaz). Por ejemplo:

```
if (elemento instanceof ObraTeatral){ ... }
```

Para crear un **getter** de un atributo de tipo ArrayList utilice la siguiente sintaxis:

```
Class Ejemplo{
   ...
   private ArrayList<Clase> elementos;
   ...
   public ArrayList<Clase> getElementos(){
      return new ArrayList<Clase>(elementos)
   }
   ...
}
```

Para crear un método agregarElemento para un atributo de tipo ArrayList utilice la siguiente sintaxis:

```
Class Ejemplo{
   ...
   private ArrayList<Clase> elementos;
   ...
   public void agregarElemento(Clase elemento){
      elementos.add(elemento);
   }
   ...
}
```

Considere también que una variable ArrayList contiene un método **`size()`** que devuelve la cantidad de elementos que tiene la lista. En el caso de una variable List contiene una propiedad **`Count`**.

Rúbrica de calificación:

(0.5puntos) Correcta programación de las clases de tipo interface.
(0.5puntos) Correcta programación de las clases de tipo enum.
(6.0 puntos) Correcta programación de todas las clases con sus atributos, relaciones, constructos, getters y setters (respetando encapsulamiento, abstracción (métodos y clases), herencia, interfaces y enumerados).
(7.0 puntos) Correcta programación del método `devolverDatos()` en las clases donde es requerido.
(2.0 puntos) Correcta programación del método `devolverCabecera()` en la clase donde se requiere.
(1.0 puntos) Correcta programación del método `consultarDatosEvento(int indice)` en la clase donde se requiere.
(2.0 puntos) Correcta programación del método `consultarObrasTeatrales()` en la clase donde se requiere.
(1.0 puntos) Correcto funcionamiento del programa e impresión del reporte.

28 de agosto del 2024