UTNMDP
Regional Mar del Plata

2023

Proyecto final “Cinema Arcano”

PROGRAMACIÓN Y LABORATORIO III

CALDERON DOLORES, MARTINEZ AUGUSTO, PINEDA DIEGO, TORESANI RAFAEL

Enlace de GitHub al proyecto:

https://github.com/AugustoMartinez/sistema-cine-net

Manual de usuario de “Cinema Arcano”:

Cinema Arcano es un proyecto que permite gestionar la compra y venta de entradas en un cine. El sistema cuenta con una función de inicio de sesión, donde los usuarios pueden registrarse o iniciar sesión si ya tienen una cuenta. Una vez dentro, se muestra un menú específico para cada tipo de usuario (cliente, gerente y administrador), donde solo se muestra el menú correspondiente al tipo de usuario ingresado.

Dentro del **menú de usuario**, se encuentran diferentes funcionalidades disponibles. Una de ellas es la generación de reservas, donde se muestra una lista de películas disponibles y las funciones asociadas a la película elegida, incluyendo información sobre la sala. El usuario puede seleccionar la ubicación de los asientos. Una vez seleccionados estos elementos, se muestra una interfaz con todos los detalles de la función, incluyendo el precio basado en el tipo de película y la cantidad de asientos seleccionados. Una vez confirmada la reserva, se imprime un ticket y se genera un archivo PDF para poder mostrar el ticket en el cine.

El usuario también puede acceder a la sección "Mis reservas" para ver sus reservas vigentes, mostradas en forma de lista. Además, tiene la opción de editar sus datos personales y darse de baja.

El **menú del administrador** se divide en tres secciones, que permiten el ABM (alta, baja, modificación) de películas, gerentes y salas. Cada sección presenta una lista desplegable con los elementos correspondientes. También ofrece un botón para crear nuevas funciones.

En la sección de agregar películas, se ingresan las características básicas como título, duración, género, etc. Además, se especifica el tipo de proyección (2D, 3D, 2D Atmos, 3D Atmos) mediante una casilla de verificación. La fecha de estreno se maneja con un componente JCalendar, y también se carga la imagen de la portada de la película. Es importante destacar que el administrador es el único responsable de crear gerentes y solo él puede dar de alta o baja a cualquier gerente. Por último, en la sección de salas, se crea una sala a partir de una matriz genérica de filas y columnas de asientos. Se seleccionan las butacas y su ubicación en la sala, y también se puede modificar o dar de baja la sala existente.

El botón "Crear función" permite seleccionar una película de una lista desplegable, que muestra el nombre de la película junto con su tipo (por ejemplo, Akira 3D). Luego, según la película elegida, se muestran las salas disponibles que se pueden seleccionar en función del tipo de proyección (Atmos o no). A continuación, se elige la fecha y el horario para crear la función. Solo se

mostrarán los horarios disponibles para ese día en particular, teniendo en cuenta la disponibilidad de la sala seleccionada. En caso de no haber más horarios disponibles para ese día, se mostrará un mensaje informativo.

Por último, el **menú del gerente** presenta las mismas secciones que el menú del administrador, con la excepción de la sección "Crear gerente".

Problemas presentados durante el proyecto:

Durante el desarrollo del proyecto final en Java, nos encontramos con varios desafíos.
Uno de ellos trataba acerca del trabajo con instancias de un mismo objeto. Esta situación generó problemas que afectaron el funcionamiento del programa, ya que las modificaciones realizadas en una instancia afectaban involuntariamente a las demás. Para solucionar este inconveniente, recurrimos al método “clone()”.
El problema principal radica en el hecho de que, al tener múltiples instancias de un mismo objeto, cualquier modificación realizada en una instancia se reflejaba en todas las demás. Esto condujo a resultados inesperados y dificultades para mantener la consistencia de los datos. Además, al trabajar con referencias a objetos, se presentaron casos en los que se perdían las referencias originales o se producía interferencia entre las instancias.

Para abordar este problema, optamos por utilizar el método “clone()” de Java. Utilizamos este método ya que nos permitió crear copias independientes del objeto original, evitando así los problemas asociados con las modificaciones involuntarias y las referencias compartidas. De esta manera, cada instancia se convirtió en una entidad separada con su propio estado y referencias, evitando así las interferencias no deseadas.

A continuación incorporamos cómo utilizamos “clone()” en nuestro código:

```
public class Sala implements Cloneable {

 @Override
 public Object clone() throws CloneNotSupportedException {
     Sala salaCopia = (Sala) super.clone();
     salaCopia.butacas = new Butaca[this.filas][this.columnas];

     for (int i = 0; i < this.filas; i++) {
         for (int j = 0; j < this.columnas; j++) {
             salaCopia.butacas[i][j] = (Butaca) this.butacas[i][j].clone();
         }
     }

     return salaCopia;
 }
funcion.setSala((Sala)sala.clone());
```

En este ejemplo, implementamos la interfaz “Cloneable” y sobrescribimos el método “clone()”. Luego, utilizamos “super.clone()” para crear una copia del objeto original y asegurarnos de que todas las propiedades se copien correctamente.

De esta forma mantenemos la integridad de los datos al trabajar con copias independientes, evitando modificaciones no deseadas en las instancias originales y además, la implementación de “clone()” facilitó el manejo de los objetos y nos permitió mantener un mayor control sobre su comportamiento.

En el proyecto nos encontramos con otro problema al querer generar correctamente la fecha y hora de las reservas. El desafío residía en que la hora estaba representada como un Enum y el día como un Date, ya que los datos provenían de un componente JCalendar.

Sin embargo, pudimos resolver este problema gracias a una función que implementamos la cual compartimos a continuación:

```
public String retornaStringListaReservas() {
String txt = "";
LocalDate fechaActual = LocalDate.now();
LocalTime horaActual = LocalTime.now();

for (int i = 0; i < listaReserva.size(); i++) {
    Date fechaReserva = listaReserva.get(i).getFuncion().getDia();
    String horita = listaReserva.get(i).getFuncion().getHorario().getHorario();

    LocalDate fechaReservaLocalDate = fechaReserva.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();

    int hora = Integer.parseInt(horita.substring(0, 2));
    int minutos = Integer.parseInt(horita.substring(3, 5));

    LocalTime horaReserva = LocalTime.of(hora, minutos);

    if (fechaReservaLocalDate.isAfter(fechaActual) || (fechaReservaLocalDate.isEqual(fechaActual) && horaReserva.isAfter(horaActual))) {
        txt += listaReserva.get(i).toString();
        if (i < listaReserva.size() - 1) {
            txt += "\n\n";
        }
    }
}
   return txt;
}
```

Esta función realiza la comparación de fechas y horas de las reservas con la fecha y hora actual. Primero, se obtiene la fecha actual utilizando LocalDate.now(), y la hora actual utilizando LocalTime.now(). Luego, itera sobre la lista de reservas y obtiene la fecha y hora de cada reserva.

Para solucionar el problema de representación de la fecha, convertimos la fecha de reserva de tipo Date a LocalDate utilizando el método toInstant().atZone(ZoneId.systemDefault()).toLocalDate().

Además, extrajimos la hora y los minutos de la hora de reserva utilizando operaciones de manipulación de cadenas, y creamos un objeto LocalTime con estos valores.
Finalmente, realizamos la comparación de fechas y horas utilizando los métodos isAfter() y isEqual() de LocalDate y LocalTime, respectivamente.

De esta manera, logramos obtener la representación en forma de cadena de caracteres de las reservas que tenían una fecha y hora posterior a la fecha y hora actual.

Enlace público a GitHub:

- https://github.com/dolocalderon
- https://github.com/RafaToresani
- https://github.com/AugustoMartinez
- https://github.com/DiegoPineda