Tarea DI01 – Breve Informe

El proyecto consta de varios paquetes, siguiendo la norma Modelo, Vista, Controlador.

* **com.cypherestudios.booking.app :** Contiene la clase main del proyecto.
* **com.cypherestudios.booking.controller :** Aquí se almacenan los controladores de las vistas de los formularios.
* **com.cypherestudios.booking.dao :** En este paquete se encuentran todas las clases relacionadas con la gestión de los datos (se almacenan en un ArrayList).
* **com.cypherestudios.booking.exceptions :** Incluye la clase de excepciones personalizada para manejar errores en la app.
* **com.cypherestudios.booking.model :** Contiene los modelos de los objetos utilizados (Booking, Banquet, Workshop , Meeting y HostingRoom).
* **com.cypherestudios.booking.resources :** Incluye las imágenes utilizadas en la app.
* **com.cypherestudios.booking.view :** Se encuentra la vista formulario que interactúa con el usuario.

Ya que el proyecto utiliza muchos archivos, paso a dar una breve descripción a los más importantes.

Para ampliar información, cada clase tiene sus comentarios y descripciones.

Tengo definidas varios modelos, para la gestión de las reservas:

* **Booking:** Clase abstracta que representa una reserva de evento genérica. Contiene atributos como el nombre del cliente, fecha de reserva, número de asistentes y tipo de cocina.

Es la clase base para los tipos específicos de eventos (Congreso, Banquete, Jornada).

* **Banquet**: Representa una reserva de tipo "Banquete". Extiende la clase Booking y define el tipo de evento como "Banquete".
* **Workshop**: Resumen: Representa una reserva de tipo "Jornada". Extiende la clase Booking y define el tipo de evento como "Jornada".
* **HostingRoom:** Representa las habitaciones de hotel necesarias para una reserva de tipo "Congreso". Contiene atributos para el número de días y habitaciones. Tiene un método estático evaluateRoomsData() para validar los datos de reserva.

Utilizada por la clase Meeting para gestionar la reserva de habitaciones si es necesario.

* **Meeting:** Modela una reserva de tipo "Congreso". Extiende la clase Booking. Además, tiene atributos específicos como el número de jornadas, necesidad de alojamiento (HostingRoom) y métodos para gestionar la reserva de habitaciones.

Hereda de la clase Booking y utiliza la clase HostingRoom para gestionar las habitaciones de hotel.

* Para almacenar y gestionar las reservas, existe la clase **BookingsArrayList** que implementa una lista de reservas utilizando un ArrayList de la clase Booking. Proporciona métodos para agregar, obtener y contar reservas.

Clase CtrlInit

Esta clase es el controlador base que gestiona la lógica y la interacción del usuario para la aplicación de reserva (*CypherBookingSystem*).

**Package e Imports:**

* La clase está en el paquete com.cypherstudios.booking.controller.
* Importa las clases BookingDAO, BookingsArrayList, y AppInit.

**Atributos:**

* protected BookingsArrayList publicBookingList: Representa la lista de reservas. Siendo un atributo protegido las clases que heredan de CtrlInit pueden acceder a este atributo.
* protected final BookingDAO op: Una instancia de la clase BookingDAO, siendo ésta la que gestiona los datos relacionados con las reservas.
* protected final AppInit appInit: Una instancia de la clase AppInit, correspondiente a la ventana principal de la aplicación.

**Constructor:**

* Configura listeners para los botones en la interfaz de usuario (appInit). En particular, los botones para reservas (btnBooking), listar reservas (btnBookingList), y salir de la aplicación (btnExit).

**Métodos:**

* public void appRun(): Este método arranca la aplicación haciendo visible el JFrame (appInit), establece su título y lo posiciona en el centro de la pantalla.
* @Override public void actionPerformed(ActionEvent e): Este método se ejecuta cuando ocurre un evento de acción (por ejemplo, clic en un botón). Realiza acciones específicas según el botón que se presiona.

**Acciones en el Método actionPerformed:**

* Cierra la aplicación si se hace clic en el botón "Exit" (btnExit).
* Inicia el sistema de reservas si se hace clic en el botón "Booking" (btnBooking). Crea una instancia de CtrlBooking y lanza un JDialog para realizar reservas.
* Abre el panel donde se listan las reservas guardadas si se hace clic en el botón "Booking List" (btnBookingList). Crea una instancia de CtrlBookingList y muestra un JDialog con la lista de reservas.

En resumen, esta clase CtrlInit es el punto de inicio para la aplicación, gestionando la interfaz principal y las acciones asociadas a los botones. Las acciones específicas se delegan a controladores adicionales (CtrlBooking y CtrlBookingList) para mantener la modularidad y la separación de implementaciones en el diseño de la aplicación.

Clase CtrlBooking

Esta clase controller maneja la lógica y la interacción del usuario para la reserva.

**Package y Imports:**

* La clase se encuentra en el paquete com.cypherstudios.booking.controller.
* Importa las clases BookingsArrayList (clase que maneja la lista de reservas), BookingDialog (formulario de diálogo para realizar reservas) y algunas las clases relacionadas con eventos y GUI necesarias

**Extiende e Implementa:**

* Extiende la clase CtrlInit.
* Implementa la interfaz ActionListener, lo que significa que esta clase puede manejar eventos de acción.

**Atributos:**

* private final BookingDialog bookingDialogWindow: Instancia de BookingDialog que representa el cuadro de diálogo para realizar reservas.

**Constructor:**

* Se encarga de configurar los listeners para las diferentes acciones que pueda realizar el usuario.

**Métodos:**

* public BookingsArrayList runBooking(BookingsArrayList publicBookingList): Este método lanza el cuadro de diálogo de reserva. Toma una lista de reservas como parámetro, establece la lista de reservas en la instancia de la clase y hace visible el cuadro de diálogo.
* @Override public void actionPerformed(ActionEvent e): Este método se ejecuta cuando ocurre un evento de acción (por ejemplo, clic en un botón o elemento de menú). Realiza acciones según el origen del evento.

**Acciones en el Método actionPerformed:**

* Cierra la aplicación si se hace clic en el elemento de menú "Exit".
* Cierra el cuadro de diálogo y vuelve al JFrame principal si se hace clic en el botón "Return" o en el elemento de menú "Return".
* Guarda la reserva si se hace clic en el botón "Save Booking" o en el elemento de menú "Save Booking".
* Abre una nueva ventana (CtrlBookingList) si se hace clic en el elemento de menú "Booking List".

En general, esta clase está diseñada para manejar la interacción del usuario en el proceso de reserva, gestionando eventos y realizando las acciones correspondientes.

Clase CtrlBookingList

Esta clase controlador (CtrlBookingList) gestiona la visualización y la interacción del usuario con la lista de reservas en el cuadro de diálogo (BookingList).

**Package e Imports:**

* La clase está en el paquete com.cypherstudios.booking.controller.
* Importa las clases BookingsArrayList, BookingList, Date y algunas clases del paquete com.cypherstudios.booking.model (como Workshop, Banquet, Meeting).

**Atributos:**

* private final BookingList bookingListWindows: Instancia de BookingList, que es un JDialog para mostrar la lista de reservas.

**Constructor:**

* Configura escuchadores para los elementos del JDialog (bookingListWindows)

**Métodos:**

* public void runListWindow(BookingsArrayList publicBookingList): Este método lanza el JDialog de la lista de reservas. Toma una lista de reservas como parámetro, establece la lista de reservas en la instancia de la clase y hace visible el JDialog.
* @Override public void actionPerformed(ActionEvent e): Este método se ejecuta cuando ocurre un evento de acción (por ejemplo, clic en un botón o elemento de menú). Realiza acciones según el origen del evento.
* private void addExamples(): Este método añade datos de ejemplo a la lista de reservas. En este caso, crea objetos de tipo Workshop, Banquet, y Meeting y los agrega a la lista.

**Acciones en el Método actionPerformed:**

* Cierra la aplicación si se hace clic en el elemento de menú "Exit".
* Cierra el JDialog y vuelve al JFrame principal si se hace clic en el botón "Return" (btnReturn) o en el elemento de menú "Return".
* Al hacer clic en el botón "Booking List" (btnBookingList), llama al método addExamples() para agregar datos de ejemplo a la lista de reservas y luego utiliza la instancia de BookingDAO (op) para mostrar la lista en la tabla de la interfaz gráfica.

En resumen, esta clase CtrlBookingList es el controlador que maneja la lógica y la interacción del usuario para mostrar y gestionar la lista de reservas en un JDialog (BookingList).

Clase BookingDAO

Clase responsable de gestionar las operaciones relacionadas con la persistencia y manipulación de las reservas.

**Package e Imports:**

* La clase está en el paquete com.cypherstudios.booking.dao.
* Importa diversas clases del paquete com.cypherstudios.booking y clases relacionadas con excepciones (BookingExceptions), modelos (Booking, Meeting, Banquet, Workshop), vistas (BookingDialog), y componentes GUI (JTable, DefaultTableModel, JOptionPane, JComboBox).

**Atributos:**

* No hay atributos de clase, solo un constructor vacío.

**Métodos:**

**Operaciones referentes al almacenamiento de Reservas:**

* public BookingsArrayList saveBooking(BookingDialog bookingWindow, BookingsArrayList publicBookingList): Este método guarda la reserva en un ArrayList de reservas (publicBookingList) y maneja posibles excepciones relacionadas con las operaciones de reserva.
* private Booking actionBtnSaveBooking(BookingDialog bookingWindow) throws BookingExceptions: Método auxiliar que recoge los datos del formulario para crear una reserva. Distingue entre varios tipos de eventos (Congreso, Banquete, Jornada) y crea el objeto correspondiente.
* private Booking buildStandardBooking(BookingDialog bookingWindow, Booking reservation): Construye el objeto Booking estándar (común a Workshop y Banquet) con datos comunes como nombre del cliente, fecha de reserva, asistentes, tipo de cocina, etc.

**Operaciones para el acceso y listado de Reservas:**

* public void tableBookingList(JTable jtBookingList, BookingsArrayList publicBookingList): Este método llena una tabla (JTable) con información sobre las reservas existentes.

**Métodos Auxiliares:**

* private void cleanForm(BookingDialog bookingWindow): Limpia los campos del formulario después de realizar una reserva.
* private void fillColumns(DefaultTableModel bookingTable, int x, Booking getBooking) throws BookingExceptions: Rellena las columnas de la tabla con datos específicos de la reserva.
* Varias funciones que realizan comprobaciones y evaluaciones de datos (dataEvaluate, validReservation, existBooking, dataEvaluateOption, valueBookingType).

En general, esta clase parece ser responsable de gestionar la persistencia de las reservas, realizar operaciones de guardado y listado, así como realizar diversas validaciones y manipulaciones de datos. Además, utiliza clases relacionadas con la interfaz gráfica de usuario (GUI) para interactuar con el usuario.

Tarea para DI01.

Enunciado.

La empresa BK continua con el desarrollo de las interfaces para la aplicación de gestión hotelera y solicitan tu ayuda para completar las interfaces que necesitan. Entre otras, la empresa tiene la necesidad de cubrir la reserva de salones disponibles para convenciones, eventos, congresos, etc. La práctica que debes realizar consiste en crear una interfaz que permita gestionar la **reserva** de uno de estos salones, llamado "Salón Habana". Los requisitos que debe cumplir esta interfaz son:

1. Se debe escribir el nombre y teléfono de contacto (Jlabel)de la persona que hace la reserva.
2. Hay que cumplimentar:
   * Fecha del evento. Utiliza un spinner (JSpinner - SpinnerDateModel)al que deberás modificar el modelo.
   * Tipo, se puede escoger entre **banquete**, **jornada** o **congreso**. Componente a tu criterio. (JComboBox)
   * Número de personas que asistirán. Componente a tu criterio. (JSpinner – SpinnerNumberModel)
   * Tipo de cocina que se precisará, a elegir entre bufé, carta, pedir cita con el chef o no precisa. Componente a tu criterio. (JComboBox)
3. Si el evento es un **congreso** se debe preguntar el número de jornadas del congreso y si se requerirán habitaciones para los asistentes al evento. Esto puedes hacerlo añadiendo los componentes necesarios con la propiedad enabled a falso.
4. Las reservas deben quedar guardadas, aunque sea simplemente en un array y por tanto deberemos habilitar un botón o una opción del menú, o ambas cosas, para poder ver dichas reservas.
   * Nota Victor: ¿podría guardar el array en un xml? Ufff…. No te lies

Tu tarea consistirá en generar una aplicación que cuente con un diálogo nuevo en el que crearás la interfaz con las restricciones que se indican más arriba. Además de añadir los componentes que necesites para darle funcionalidad deberás cumplir con los siguientes requisitos no funcionales:

* Debes **cambiar el nombre** y añadir un **ToolTipText** a los componentes.
* Modifica el formato de texto de las etiquetas y añade separadores y paneles con título para aportar vistosidad a la interfaz.
* Utiliza el modo de diseño libre para colocar los componentes en el diálogo. Cuida que la composición quede armónica.
* La interfaz generada irá conectada a la interfaz principal de la aplicación mediante el menú principal y mediante un botón que debes añadirle que tendrás que programar correctamente para que abran el diálogo que has creado.
* Debes crear la interfaz modal.
* Cuando se selecciona como tipo de evento banquete o jornada los componentes para número de días y habitaciones deben estar desactivados, sin embargo, si es congreso se deben activar.
* Deberán poder verse las reservas realizadas hasta el momento.

**SpinnerListModel**

The SpinnerListModel is a model whose values are defined by an array of objects or a List object. The Month spinner in the SpinnerDemo example uses this model, initialized with an array derived from the value returned by the getMonths method of the java.text.DateFormatSymbols class. See SpinnerDemo.java for details.

**SpinnerNumberModel**

The SpinnerNumberModel supports sequences of numbers which can be expressed as double objects, int objects, or Number objects. You can specify the minimum and maximum allowable values, as well as the step size — the amount of each increment or decrement. The Year spinner uses this model, created with the following code:

SpinnerModel model =

new SpinnerNumberModel(currentYear, //initial value

currentYear - 100, //min

currentYear + 100, //max

1); //step

**spinner** (SpinnerDateModel)

The SpinnerDateModel supports sequences of Date objects. You can specify the minimum and maximum dates, as well as the field (such as Calendar.YEAR) to increment or decrement. Note, however, that some types of look and feel ignore the specified field, and instead change the field that appears selected. The Another Date spinner uses this model, created with the following code:

Date initDate = calendar.getTime();

calendar.add(Calendar.YEAR, -100);

Date earliestDate = calendar.getTime();

calendar.add(Calendar.YEAR, 200);

Date latestDate = calendar.getTime();

model = new SpinnerDateModel(initDate,

earliestDate,

latestDate,

Calendar.YEAR);

<https://docs.oracle.com/javase/tutorial/uiswing/components/spinner.html>