{Nombre del problema a resolver}

# Análisis del Sistema (30 puntos)

## 1.1 Requisitos funcionales del sistema (5 pts)

Escribir los puntos exactos que debe cumplir tu sistema.

Ejemplo: “El usuario puede crear una nueva solicitud de boletos”, etc.

- El programa debe ser capaz de tener 4 tipos de arrays: 1 de salones pequeños (tamaño de 6 salones), 1 de salones medianos (tamaño de 4 salones), 1 de salones grandes (tamaño de 3 salones), y uno de eventos en espera (tamaño de 10 eventos).

- El usuario debe ser capaz de solicitar la reserva de un salón. Para ello, debe ingresar los datos que el sistema le pida. Una vez se ha ingresado la reserva, esta se almacena en la lista de eventos en espera

- El usuario debe ser capaz de ingresar un nuevo salón al sistema. Para ello, se le pedirá ingresar las características propias del salón. Una vez ha ingresado estas características, el salón se asigna a una lista en base al tamaño que tenga (pequeño, mediano o grande).

- El usuario debe ser capaz de asignar un evento a un salón. Al momento de seleccionar esta opción, el usuario elige que evento quiere asignar. Una vez seleccionado el evento, se despliega la lista de salones disponibles para asignarlo. El usuario elige a que salón quiere asignar el evento, y se verifica que se pueda realizar la asignación. En caso de ser posible, se muestra un mensaje congratulatorio, y se remueve el evento de la lista de espera. En caso de no ser posible, se explica por qué y el usuario debe elegir otro salón para asignar la reserva.

## 1.2 Clases necesarias y su propósito (5 pts)

|  |  |
| --- | --- |
| Clase | Propósito |
| Main | Es la clase de vista, en donde se realizan la mayoría de los mensajes al usuario. Además, desde aquí se manda información que el controlador recibe para hacer las operaciones necesarias. |
| Controlador | Esta clase se encarga de manejar el desarrollo del programa. Tiene objetos de Salon, Reserva y Reglas, y cuenta con diversos métodos que el programa Main usa. |
| Salón | Esta clase representa los salones con los que cuenta el negocio de eventos. Cada salón puede tener múltiples eventos asignados (aunque en diferente hora). |
| Reserva | Esta clase representa las solicitudes de reserva que se tienen en lista de espera. Cada reserva tiene características propias, y en base a ellas se verifica si se puede asignar a un salón o no. |
| Reglas | Esta clase representa todas las reglas que se deben cumplir para asignar una reserva a un salón. Está compuesta de funciones. |
| Lista de salon | Esta clase funciona para almacenar todos los salones que son creados por el usuario. Se encarga además de verificar que no se exceda la cantidad de cierto tipo de salón, y tiene diferentes funciones para tratar los salones en su interior. |
| Lista de espera | Esta clase funciona para almacenar todas las reservas que genera el sistema. Se encarga además de verificar que no se exceda la cantidad de reservas en la lista de espera, y tiene diferentes funciones para tratar las reservas en su interior. |

## 1.3 Atributos de cada clase (10 pts)

Clase: Salón

|  |  |  |  |
| --- | --- | --- | --- |
| Atributo | Tipo de dato | Visibilidad | Propósito |
| Nombre | String | Privada | Representa el nombre del salón, funciona como identificador. |
| Tamaño | String | Privada | Representa el tamaño del salón. Debe ser de tipo “pequeño”, “mediano” o “grande”. |
| Hora disponible | int | Privada | Representa la ultima hora disponible para asignar un evento a este salón. No lo asigna el usuario, empieza en 0. |
| Tiempo de reserva máximo | int | Privada | Representa el tiempo máximo que puede reservarse el salón. |
| Capacidad máxima | int | Privada | Representa la cantidad máxima de personas que puede mantener el salón. No lo ingresa el usuario, se decide por el tamaño del salón. |
| Costo de reserva (por hora) | double | Privada | Representa el costo por hora al usar el salón. |

Clase: Reserva

|  |  |  |  |
| --- | --- | --- | --- |
| Atributo | Tipo de dato | Visibilidad | Propósito |
| Nombre del evento | String | Privada | Es el nombre del evento, el cual se usa como identificador. |
| Cliente VIP | boolean | Privada | Es el tipo de cliente que hace la reserva. En caso de ser True, el cliente se considera un cliente VIP. En caso de ser False, el cliente se considera un cliente normal. |
| Hora de inicio | int | Privada | Es la hora a la que inicia el evento. Debe ser un número positivo. |
| Duración de evento | int | Privada | Es la duración del evento, en horas. Debe ser un número positivo. |
| Cantidad de personas que atenderán | int | Privada | Es la cantidad de personas que atenderán al evento. Debe ser un número positivo. |
| Pago inicial de salon | double | Privada | Es la cantidad de dinero que el cliente pagará inicialmente para usar el salón. |

Clase: Reglas

|  |  |  |  |
| --- | --- | --- | --- |
| Atributo | Tipo de dato | Visibilidad | Propósito |
| Ninguno |  |  |  |

Clase: Lista de salones

|  |  |  |  |
| --- | --- | --- | --- |
| Atributo | Tipo de dato | Visibilidad | Propósito |
| Tamaño | String | Privada | Es el tamaño de los salones que se almacenan en esta lista. Existen tres tamaños de salón: pequeño, mediano y grande. |
| Salones | Salon[] | Privada | Es la lista en la que se almacenan los objetos de tipo Salon. |
| Posición para nuevo salón | int | Privada | Este número sirve para saber en qué posición debe colocarse un Salon que se ha añadido a la lista. |

Clase: Lista de espera

|  |  |  |  |
| --- | --- | --- | --- |
| Atributo | Tipo de dato | Visibilidad | Propósito |
| Reservas | Reserva[] | Privada | Es la lista en que se almacenan los objetos de tipo Reserva. |
| Disponibles | boolean[] | Privada | Esta lista funciona para saber en que posiciones de la lista de reservas puede almacenarse una nueva reserva. Empieza con valor de True en todos, y cada valor puede cambiar a lo largo del programa. True representa espacio desocupado, False representa espacio ocupado. |

Clase: Controlador

|  |  |  |  |
| --- | --- | --- | --- |
| Atributo | Tipo de dato | Visibilidad | Propósito |
| Lista de salones pequeños | Lista de Salones | Privada | Esta lista contiene todos los salones pequeños con los que cuenta el establecimiento. |
| Lista de salones medianos | Lista de Salones | Privada | Esta lista contiene todos los salones medianos con los que cuenta el establecimiento. |
| Lista de salones grandes | Lista de Salones | Privada | Esta lista contiene todos los salones grandes con los que cuenta el establecimiento. |
| Lista de salones actual | Lista de Salones | Privada | Sirve como referencia para saber con qué lista de salones se trabaja en una función, y no tener que recorrer las tres listas de salones en cada ocasión. |
| Lista de espera | Lista de Espera | Privada | Esta lista contiene todas las reservas que aún no se han asignado a salones. |

### 1.4 Métodos de cada clase (10 pts)

Clase: Salón

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | Parámetros:  Tipo de dato | Resultado:  Tipo de dato | Visibilidad | Propósito |
| Actualizar disponibilidad | Nueva hora: int, tiempo de reserva: int | void | publica | Este método actualiza la ultima hora en que puede asignarse un salón. |
| Asignar preset | Tamaño: String | void | Privada | Este método se usa junto con el constructor. Su propósito es asignar al salón su capacidad máxima de personas, en base al tamaño que tiene |

Clase: Reserva

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | Parámetros:  Tipo de dato | Resultado:  Tipo de dato | Visibilidad | Propósito |
| Ninguno |  |  |  |  |

Clase: Regla

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | Parámetros:  Tipo de datos | Resultado:  Tipo de dato | Visibilidad | Propósito |
| Verificar VIP | Tamaño de salón: String  Reserva VIP: boolean | boolean | Pública | Este método verifica si la reserva puede asignarse a un salon grande. En caso de que el tamaño del salon sea grande, y la reserva no está hecha por una persona VIP, se regresa False. De lo contrario, se regresa True. |
| Verificar capacidad | Tamaño de salón: VIP  Cantidad de asistentes: int | boolean | Pública | Este método verifica si se tiene la cantidad de asistentes necesaria para reservar el salón. Esto depende del tamaño del salón: 1/2 para salones pequeños, 2/3 para salones medianos, y 4/3 para salones grandes. Si no se cubre la cuota, se regresa False. De lo contrario, se regresa True. |
| Verificar hora | Hora disponible: int  Hora de inicio: int | boolean | Pública | Este método verifica si el salón está disponible a la hora que se necesita. En caso de no estarlo (hora posible > hora de inicio), se regresa False. De lo contrario, se regresa True. |
| Verificar tiempo de reserva | Tiempo máximo de reserva: int  Duración de evento: int | boolean | Pública | Este método verifica si las horas que durará el evento es adecuada al salón deseado. Cada salón cuenta con un máximo de duración de evento, por lo que si se excede el número de horas que dura el evento, se regresa False. De lo contrario, se regresa True. |
| Verificar presupuesto | Costo de salón: double  Duracion\_evento: int  Pago inicial: double | boolean | Pública | Este método verifica que el pago inicial de la reserva sea suficiente para reservar el salón. Es necesario pagar al menos la mitad del pago total para poder reservar el salón (pago total = horas de evento \* costo de salón). Si no puede pagarse, se regresa False. De lo contrario, se regresa True. |

Clase: Lista de salones

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | Parámetros:  Tipo de datos | Resultado:  Tipo de dato | Visibilidad | Propósito |
| Verificar capacidad máxima | Ninguno | boolean | Pública | Este método verifica si aún puede añadirse un nuevo salón a la lista de salones. Para ello, compara si la posición de nuevo salón es igual al tamaño del array de salones. En caso de ser verdadero, regresa False. De lo contrario, regresa True. |
| Verificar hay salón en la lista de salones | Ninguno | boolean | Pública | Este método verifica si hay al menos 1 salón asignado en la lista de salones. Para ello, compara si la posición de nuevo salón es igual a 0. En caso de ser verdadero, regresa False. De lo contrario, regresa True. |
| Añadir salón | Nombre: String  Tamaño: String  Costo de reserva: double  Tiempo máximo de reserva: int | boolean | Pública | Este método añade un nuevo salón a la lista de salones. Para ello, verifica primero si puede añadirse un nuevo salón a la lista. En caso de no poderse, regresa False. De lo contrario, se crea un objeto nuevo de Salon con los parámetros ingresados, se asigna a la lista de salones en la posición para ingresar salones, se actualiza dicha posición y se regresa True. |
| Buscar salón | Nombre de salón a buscar: String | Salon | Privada | Este método busca si el nombre del salón ingresado le corresponde a alguno de los salones de la lista. Para ello, recorre la lista de salones, comparando el nombre de cada uno con el nombre ingresado. Si no se encuentra, se regresa Null. De lo contrario, se regresa el Salon que coincida con el nombre. |
| Salon se encuentra en lista | Nombre de salón a buscar: String | boolean | Pública | Este método llama al método de Buscar salón. Si este regresa Null, se regresa False. De lo contrario, se regresa True. |
| Mostrar salones | ninguno | String | Pública | Este método da el nombre de todos los salones que se encuentran en la lista de salones. Para ello, empieza un String con el tamaño de salones de esta lista. Seguido de ello, recorre la lista de salones hasta la posición de nuevo salón, añadiendo el nombre de cada salón al String de mensaje. Al final, regresa el String de mensaje. |

Clase: Lista de espera

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | Parámetros:  Tipo de datos | Resultado:  Tipo de dato | Visibilidad | Propósito |
| Verificar espacio para reserva | Ninguno | boolean | Pública | Este método verifica si aún puede añadirse una nueva reserva a la lista de espera. Para ello, recorre la lista de disponibles. Si encuentra que en una posición hay True, regresa True. Por el contrario, si logra recorrer toda la lista, regresa False. |
| Verificar reservas en lista de espera | Ninguno | boolean | Pública | Este método verifica si hay alguna reserva en la lista de espera. Para ello, recorre la lista de disponibles. Si encuentra que en una posición hay False, regresa True. Por el contrario, si logra recorrer toda la lista, regresa False. |
| Añadir reserva | Nombre: String  Cliente VIP: boolean  Hora de inicio: int  Duración de evento: int  Cantidad de asistentes: int  Pago inicial: double | boolean | Pública | Este método ingresa una nueva reserva a la lista de espera. Para ello, verifica si se puede ingresar una nueva reserva. En caso de que no, se regresa False. De lo contrario, se busca la primera posición de la lista de disponibles que tenga True. Cambia esta posición a False, e ingresa un nuevo objeto de Reserva en la misma posición, en la lista de reservas. El nuevo objeto de Reserva se crea con los parámetros ingresados. Después de esto, regresa True. |
| Buscar reserva | Nombre de reserva a buscar: String | Reserva | Privada | Este método busca si una reserva se encuentra en la lista de reservas. Para ello, recorre la lista de disponibles. En cada paso, si el valor disponible es False, comprueba la posición equivalente en la lista de reservas. Si el nombre coincide con el ingresado, se regresa el objeto de esa reserva. De lo contrario, se sigue buscando. Si se recorre toda la lista, se regresa Null. |
| Reserva existe en lista | Nombre de reserva a buscar: String | boolean | Publica | Este método llama al método de Buscar reserva. En caso de obtener Null, regresa False. De lo contrario, regresa True. |
| Quitar reserva | Nombre de reserva a quitar: String | void | Publica | Este método remueve una reserva de la lista de espera. Para ello, se recorre la lista de disponible, y se comprueba si son False. En caso de serlo, se comprueba si su equivalente en la lista de reservas tiene el mismo nombre que el de la reserva a quitar. En caso de ser verdadero, a esa posición se le asigna null, y en la lista de disponibles se coloca True. |
| Mostrar reservas | ninguno | String | Pública | Este método da el nombre de todas las reservas en la lista de espera. Para ello, se recorre la lista de disponibles, y se comprueba si cada posición es False. En caso de serlo, el nombre de la reserva equivalente en la lista de reservas se añade a un String. Al finalizar, se regresa el String. |

Clase: Controlador

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | Parámetros:  Tipo de datos | Resultado:  Tipo de dato | Visibilidad | Propósito |
| Anadir salón | Nombre: String  Tamaño: String  Costo de reserva: double  Tiempo máximo de reserva: int | boolean | Pública | Este método añade un nuevo salón a su lista correspondiente. Para ello, se selecciona una lista de salones en base al tamaño ingresado. Si esta lista no puede tener más salones, se regresa False. De lo contrario, se llama al método de Añadir salón de la lista de salones correspondiente. En esta llamada se le envían todos los parámetros ingresados. Después de esto, se regresa True. |
| Anadir reserva | Nombre: String  Cliente VIP: boolean  Hora de inicio: int  Duración de evento: int  Cantidad de asistentes: int  Pago inicial: double | boolean | Pública | Este método añade una nueva reserva a la lista de espera. Para ello, se verifica primero si la lista de espera no está en su capacidad máxima. En caso de estarlo, se regresa False. De lo contrario, se llama a la función de Añadir reserva de la lista de espera. En esta llamada se le pasan todos los parámetros ingresados. Después de esto, se regresa True. |
| Verificar reserva puede asignarse | Nombre de reserva a asignar: String  Nombre de salon deseado: String | int | Privado | Se revisa si una reserva puede asignarse a un salón. Para ello, se aplican diferentes ifs que usan características de la reserva y del salón deseado. En caso de no cumplirse, cada if regresa un numero diferente (un código). Si todos los ifs se cumplen, se regresa 0. |
| Asignar reserva | Nombre de reserva a asignar: String  Nombre de salón deseado: String | int | Pública | Este método asigna una reserva de la lista de espera al salón deseado. Para ello, se busca si la reserva y el salón ingresados se encuentran en el sistema. En caso de no estarlo, se regresa -2 y -1, respectivamente. De lo contrario, se llama a la función de verificar si reserva puede asignarse, usando los parámetros ingresados. Si esta función regresa 0, se actualizan los datos del salón correspondiente, y se remueve la reserva de la lista de espera. Al finalizar, se regresa 0. Si, por el contrario, la función regresa otro número, no se realiza ninguna otra acción y se regresa el mismo número. |

### 2. Diseño: Diagrama de Clases (30 puntos)

* Asegúrate de mostrar atributos y métodos con visibilidad (+, -).
* Indica relaciones entre clases (asociación, agregación, etc.).
* Incluye el driver program (Main).

<https://lucid.app/lucidchart/b2f9cbea-99f6-4b1a-a100-71eff16c504c/edit?viewport_loc=-207%2C-967%2C2558%2C1408%2C0_0&invitationId=inv_7b6be7cd-6054-4706-b327-c8b6ec71e2eb>

### 3. Programa (40 puntos)

En cada archivo `.java`, asegurarse de incluir:

* Las clases necesarias.
* Uso adecuado de objetos.

Menú que debe implementar el driver program:

* 1. Nuevo comprador
  2. Nueva solicitud de boletos
  3. …
  4. …
  5. Salir

GitHub: colocar aquí la URL:

<https://github.com/Oscarro11/Manager-de-salones->

# Checklist antes de entregar

Está claro el análisis?

El diagrama tiene los elementos UML correctamente?

Subiste tu código a GitHub con todo lo necesario?