

Estado	Finalizado
Comenzado	martes, 18 de noviembre de 2025, 18:18
Completado	martes, 18 de noviembre de 2025, 20:32
Duración	2 horas 13 minutos
Calificación	80 de 100



Pregunta 1

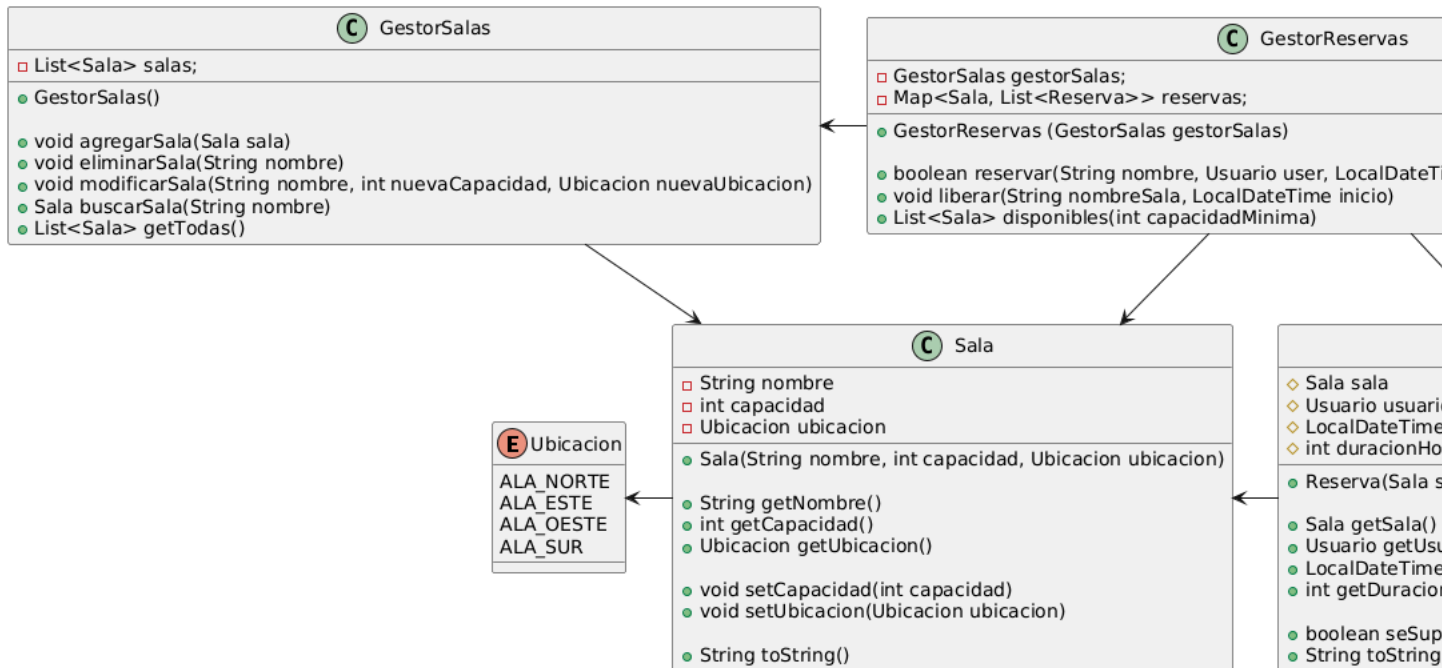
Sin contestar

Sin calificar

Se provee un sistema parcial para la gestión de reservas de salas de reunión. El sistema permite registrar salas con distintas capacidades y ubicaciones, asignar reservas a usuarios por varias horas, listar salas disponibles y cancelar reservas.

La tarea consiste en **completar e implementar las partes faltantes** del sistema según las consignas que se detallan a continuación. El objetivo es evaluar tu dominio de:

- Creación y uso de clases, objetos y colecciones.
- Implementación de constructores, getters, setters y `toString`.
- Uso de `enum`, herencia e interfaces.
- Manejo de excepciones.
- Comprensión del diseño orientado a objetos.

Descargar proyecto

Pregunta 2

Correcta

Se puntúa 20 sobre 20

Completar la clase **Sala**:

- Implementar el constructor.
- Definir los métodos `getNombre()`, `getCapacidad()` y `getUbicacion()`.
- Implementar los métodos `setCapacidad(...)` y `setUbicacion(...)`.
- Sobrescribir el método `toString()` para mostrar una descripción legible de la sala.

Objetivos evaluados: creación de objetos, constructores, getters/setters, sobrescritura de métodos.**Por ejemplo:**

Prueba	Resultado
<pre>Sala s = new Sala("Uritorco", 6, Ubicacion.SUR); System.out.println (s.getNombre()); System.out.println (s.getUbicacion()); System.out.println (s.getCapacidad()); System.out.println (s);</pre>	<pre>Uritorco Sur 6 Sala Uritorco (capacidad: 6, ubicacion: Sur)</pre>

Respuesta: (sistema de penalización: 0, 0, 0, 5, 10, 15 %)

Reiniciar respuesta

```

1 public class Sala {
2     private String nombre;
3     private int capacidad;
4     private Ubicacion ubicacion;
5
6     public Sala(String nombre, int capacidad, Ubicacion ubicacion) {
7         this.nombre = nombre;
8         // Según el setter, la capacidad solo se asigna si es > 0
9         if (capacidad > 0) {
10             this.capacidad = capacidad;
11         } else {
12             this.capacidad = 0;
13         }
14         this.ubicacion = ubicacion;
15     }
16
17     public String getNombre() {
18         return nombre;
19     }
20
21     public int getCapacidad() {
22         return capacidad;
23     }
24
25     public Ubicacion getUbicacion() {
26         return ubicacion;
27     }
28
29     /**
30      * Setea la capacidad de la sala solo si la capacidad es mayor a 0
31      */
32     public void setCapacidad(int capacidad) {
33         if (capacidad > 0) {
34             this.capacidad = capacidad;
35         }
36     }
37
38     public void setUbicacion(Ubicacion ubicacion) {
39         this.ubicacion = ubicacion;
40     }
41
42     @Override
43     /**
44      * Retorna una representación de la Sala como String con el formato
45      * "Sala <nombre> (capacidad: <capacidad>, ubicacion: <ubicacion>)"
46      */
47     public String toString() {

```

```
48 |         return "Sala " + nombre + " (capacidad: " + capacidad + ", ubicacion: " + ubicacion + ")\n49 |     }\n50 |\n51 |
```

Todas las pruebas superadas. ✓

Correcta

Puntos para este envío: 20/20.

Pregunta 3

Correcta
Se puntúa 15 sobre 15

Completa los métodos indicados de la clase `Reserva` según la documentación

- Implementar el método `toString` con formato: "Reserva de <nombre_sala> por <usuario> desde <fecha_hora_inicio> por <duracion>h".
- Implementar el método `seSuperpone(...)` para verificar si una reserva se superpone con otra en función del horario.

El sistema utiliza la clase `LocalDateTime` de la biblioteca `java.time` para representar fechas y horas. Esta clase permite expresar momentos como "2025-06-15T10:00" y sumar horas usando el método `plusHours(int)`. También pueden compararse con operadores `>`, `<`, o utilizando el método `isBefore(LocalDateTime)`. No es necesario considerar zonas horarias ni diferencias geográficas. Todos los horarios se interpretan como locales.

Objetivos evaluados: uso de `LocalDateTime`, lógica condicional, comprensión de intervalos.

Por ejemplo:

Prueba	Resultado
<pre>Sala s = new Sala ("Uritorco",4,Ubicacion.NORTE); Usuario u = new Usuario ("Juan Perez", "jp@mail.com"); LocalDateTime ldt = LocalDateTime.parse("2025-06-16T10:00:00"); Reserva r = new Reserva(s,u,ldt,1); System.out.println (r); System.out.println ("9am, 1 hora se superpone: "+ r.seSuperpone(LocalDateTime.parse("2025-06-16T09:00:00"),1)); System.out.println ("9am, 2 horas se superpone: "+ r.seSuperpone(LocalDateTime.parse("2025-06-16T09:00:00"),2));</pre>	<pre>Reserva de Uritorco por Juan Perez <jp@mail.com> desde 2025-06-16T10:00 por 1h 9am, 1 hora se superpone: false 9am, 2 horas se superpone: true</pre>

Respuesta: (sistema de penalización: 0, 0, 0, 5, 10, 15 %)

Reiniciar respuesta

```
1 import java.time.LocalDateTime;
2
3 public class Reserva {
4     protected Sala sala;
5     protected Usuario usuario;
6     protected LocalDateTime inicio;
7     protected int duracionHoras;
8
9     public Reserva(Sala sala, Usuario usuario, LocalDateTime inicio, int duracionHoras) {
10         this.sala = sala;
11         this.usuario = usuario;
12         this.inicio = inicio;
13         this.duracionHoras = duracionHoras;
14     }
15
16     public Sala getSala() { return sala; }
17     public Usuario getUsuario() { return usuario; }
18     public LocalDateTime getInicio() { return inicio; }
19     public int getDuracionHoras() { return duracionHoras; }
20
21     /**
22      * indica si el horario de esta Reserva se superpone con otro horario y duración
23      * pasada como parámetro
24      */
25     public boolean seSuperpone(LocalDateTime otroInicio, int otrasHoras) {
26         LocalDateTime fin = this.inicio.plusHours(this.duracionHoras);
27         LocalDateTime otroFin = otroInicio.plusHours(otrasHoras);
28
29         // Se superponen si los intervalos se cruzan
30         return otroInicio.isBefore(fin) && this.inicio.isBefore(otroFin);
31     }
32
33     @Override
34     public String toString() {
35         return "Reserva de " + sala.getNombre() +
36             " por " + usuario +
37             " desde " + inicio +
38             " por " + duracionHoras + "h";
39     }
40 }
41
```

Todas las pruebas superadas. ✓

Correcta

Puntos para este envío: 15/15.

Pregunta 4

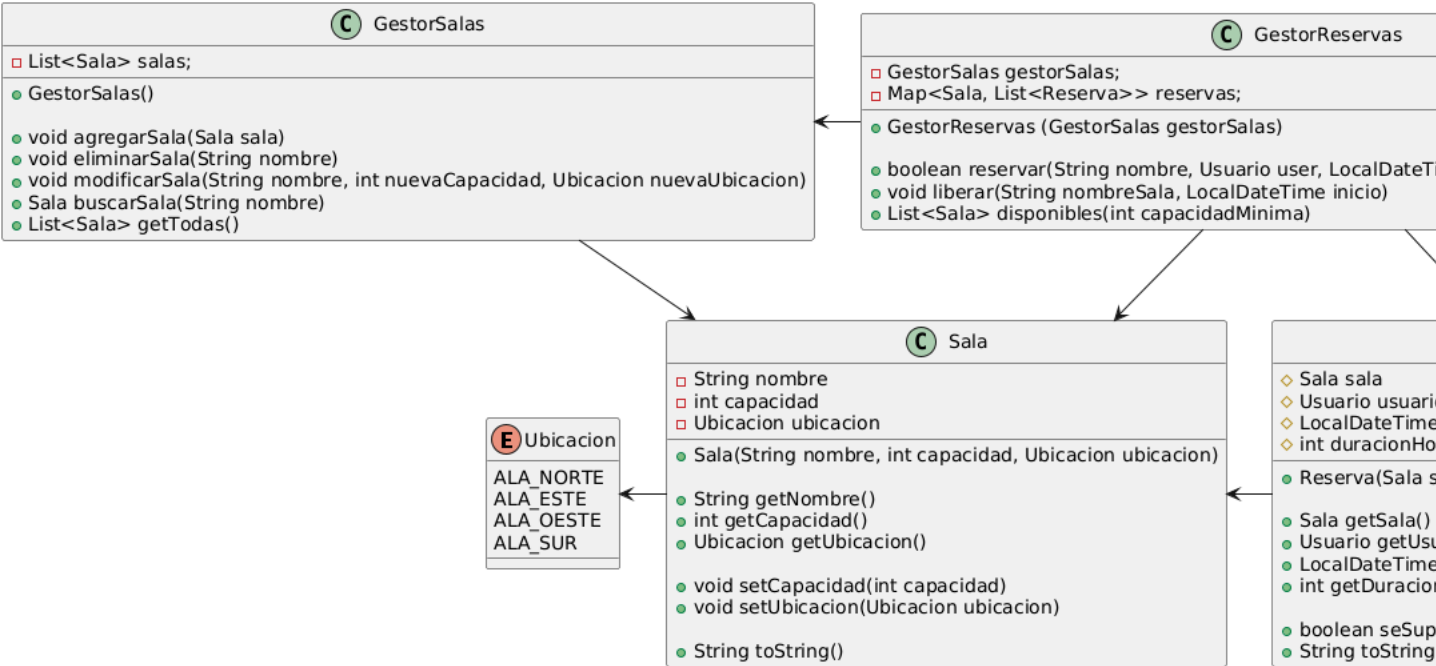
Incorrecta

Se puntúa 0 sobre 20

Implemente la Clase **ReservaRapida** tal que

- Extienda de **Reserva**
- Implemente la interfaz **Cancelable**.
- El método cancelar() muestre un mensaje simple por consola
- Reservar por defecto 1 hora (ReservaRapida es una Reserva con una duración de 1hr)

Objetivos evaluados: herencia, uso de **super**, interfaces, polimorfismo.



Por ejemplo:

Prueba	Resultado
<pre>Sala s = new Sala ("Uritorco",4,Ubicacion.NORTE); Usuario u = new Usuario ("Juan Perez", "jp@mail.com"); LocalDateTime ldt = LocalDateTime.parse("2025-06-16T10:00:00"); ReservaRapida r = new ReservaRapida(s,u,ldt); System.out.println ("Hereda de Reserva: " + (r instanceof Reserva)); System.out.println (r); r.cancelar();</pre>	<p>Hereda de Reserva: true Reserva de Uritorco por Juan Perez <jp@mail.com> desde 2025-06-16T10:00 por 1h Reserva rápida cancelada: Reserva de Uritorco por Juan Perez <jp@mail.com> desde 2025-06-16T10:00 por 1h</p>

Respuesta: (sistema de penalización: 0, 0, 0, 5, 10, 15 %)

Reiniciar respuesta

```
1 //TODO
2 //Implemente la Clase ReservaRapida tal que extienda de Reserva e implemente la interfaz
3 //Una ReservaRapida es una Reserva con una duración de 1hr
4 //Al cancelar una reserva, solo se imprime un mensaje indicando "Reserva rápida cancelada"
5 //Ej. "Reserva rápida cancelada: Reserva de Uritorco por Juan Perez <jp@mail.com> desde
6 import java.time.LocalDateTime;
7 import java.lang.String;
8
9 public class ReservaRapida extends Reserva implements Cancelable {
10     //Atributos
11     //Atributos de la interfaz Cancelable
12     //Métodos de la interfaz Cancelable
13     //Métodos de la clase ReservaRapida
14     //Métodos de la clase Reserva
15     //Métodos de la clase Usuario
16     //Métodos de la clase LocalDateTime
17     //Métodos de la clase String
18     //Métodos de la clase ArrayList
19     //Métodos de la clase Map
20     //Métodos de la clase HashSet
21     //Métodos de la clase LinkedList
22     //Métodos de la clase PriorityQueue
23     //Métodos de la clase Stack
24     //Métodos de la clase Vector
25     //Métodos de la clase Enumeration
26     //Métodos de la clase Iterator
27     //Métodos de la clase Comparable
28     //Métodos de la clase Comparator
29     //Métodos de la clase Runnable
30     //Métodos de la clase Thread
31     //Métodos de la clase Runnable
32     //Métodos de la clase Thread
33     //Métodos de la clase Runnable
34     //Métodos de la clase Thread
35     //Métodos de la clase Runnable
36     //Métodos de la clase Thread
37     //Métodos de la clase Runnable
38     //Métodos de la clase Thread
39     //Métodos de la clase Runnable
40     //Métodos de la clase Thread
41     //Métodos de la clase Runnable
42     //Métodos de la clase Thread
43     //Métodos de la clase Runnable
44     //Métodos de la clase Thread
45     //Métodos de la clase Runnable
46     //Métodos de la clase Thread
47     //Métodos de la clase Runnable
48     //Métodos de la clase Thread
49     //Métodos de la clase Runnable
50     //Métodos de la clase Thread
51     //Métodos de la clase Runnable
52     //Métodos de la clase Thread
53     //Métodos de la clase Runnable
54     //Métodos de la clase Thread
55     //Métodos de la clase Runnable
56     //Métodos de la clase Thread
57     //Métodos de la clase Runnable
58     //Métodos de la clase Thread
59     //Métodos de la clase Runnable
60     //Métodos de la clase Thread
61     //Métodos de la clase Runnable
62     //Métodos de la clase Thread
63     //Métodos de la clase Runnable
64     //Métodos de la clase Thread
65     //Métodos de la clase Runnable
66     //Métodos de la clase Thread
67     //Métodos de la clase Runnable
68     //Métodos de la clase Thread
69     //Métodos de la clase Runnable
70     //Métodos de la clase Thread
71     //Métodos de la clase Runnable
72     //Métodos de la clase Thread
73     //Métodos de la clase Runnable
74     //Métodos de la clase Thread
75     //Métodos de la clase Runnable
76     //Métodos de la clase Thread
77     //Métodos de la clase Runnable
78     //Métodos de la clase Thread
79     //Métodos de la clase Runnable
80     //Métodos de la clase Thread
81     //Métodos de la clase Runnable
82     //Métodos de la clase Thread
83     //Métodos de la clase Runnable
84     //Métodos de la clase Thread
85     //Métodos de la clase Runnable
86     //Métodos de la clase Thread
87     //Métodos de la clase Runnable
88     //Métodos de la clase Thread
89     //Métodos de la clase Runnable
90     //Métodos de la clase Thread
91     //Métodos de la clase Runnable
92     //Métodos de la clase Thread
93     //Métodos de la clase Runnable
94     //Métodos de la clase Thread
95     //Métodos de la clase Runnable
96     //Métodos de la clase Thread
97     //Métodos de la clase Runnable
98     //Métodos de la clase Thread
99     //Métodos de la clase Runnable
100    //Métodos de la clase Thread
```

```
10     int duracionHoras = 1; // Aca declaro el valor constante (de 1 hora)
11
12     // Constructor: duración fija de 1 hora
13     public ReservaRapida(Sala sala, Usuario usuario, LocalDateTime inicio, int duracionHoras) {
14         super(sala, usuario, inicio, duracionHoras); // Duración siempre es 1 hora
15     }
16
17     // Implementación del método cancelar de la interfaz Cancelable
18
19     public void cancelar() {
20         System.out.println("Reserva rápida cancelada: " + this.toString());
21     }
22 }
23
24
```

Error(es) de sintaxis

__Tester__.java:144: error: constructor ReservaRapida in class ReservaRapida cannot be applied to given types;
ReservaRapida r = new ReservaRapida(s,u,ldt);
 ^
required: Sala,Usuario,LocalDateTime,int
found: Sala,Usuario,LocalDateTime
reason: actual and formal argument lists differ in length
1 error

Incorrecta

Puntos para este envío: 0/20.

Pregunta 5

Correcta

Se puntúa 20 sobre 20

Completar los métodos de **GestorSalas** para:

- Agregar salas (agregarSala).
- Eliminar salas por nombre (eliminarSala).
- Modificar capacidad y ubicación (modificarSala).
- Buscar una sala por nombre (buscarSala).
- Obtener la lista completa (getTodas).

Objetivos evaluados: uso de colecciones (List), búsqueda, eliminación y modificación de elementos.

Por ejemplo:

Prueba	Resultado
<pre>GestorSalas gestorSalas = new GestorSalas(); Sala sala1 = new Sala("Verde", 10, Ubicacion.NORTE); Sala sala2 = new Sala("Azul", 20, Ubicacion.SUR); gestorSalas.agregarSala(sala1); gestorSalas.agregarSala(sala2); System.out.println (gestorSalas.getTodas());</pre>	[Sala Verde (capacidad: 10, ubicacion: Norte), Sala Azul (capacidad: 20, ubicacion: Sur)]

Respuesta: (sistema de penalización: 0, 0, 0, 5, 10, 15 %)

Reiniciar respuesta

```

1  import java.util.*;
2
3  public class GestorSalas {
4      private List<Sala> salas = new ArrayList<>();
5
6      /**
7       * Agrega una Sala a la lista de salas del gestor
8       */
9      public void agregarSala(Sala sala) {
10         //TODO Implementar
11         if (sala != null){
12             salas.add(sala);
13         }
14     }
15
16     /**
17      * Elimina de la lista la Sala cuyo nombre coincide
18      * con el nombre pasado como argumento.
19      */
20     public void eliminarSala(String nombre) {
21         //TODO Implementar
22         salas.removeIf(s-> s.getNombre().equalsIgnoreCase(nombre));
23     }
24
25     /**
26      * Actualiza la capacidad y ubicación de la Sala cuyo
27      * nombre coincide con el nombre pasado como argumento.
28      */
29     public void modificarSala(String nombre, int nuevaCapacidad, Ubicacion nuevaUbicacion) {
30         //TODO Implementar
31         Sala sala = buscarSala(nombre);
32         if (sala != null) {
33             sala.setCapacidad(nuevaCapacidad);
34             sala.setUbicacion(nuevaUbicacion);
35         }
36     }
37
38     /**
39      * Devuelve la Sala cuyo nombre coincide con el nombre
40      * pasado como argumento
41      */
42     public Sala buscarSala(String nombre) {
43         //TODO Implementar
44
45         for (Sala sala : salas){
46             if (sala.getNombre().equalsIgnoreCase(nombre)){

```

```

47         return sala;
48     }
49 }
50
51 return null;
52

```

	Prueba	Esperado	Conseguido	
✓	<pre> System.out.println("Test agregarSala / getTodas"); GestorSalas gestorSalas = new GestorSalas(); Sala sala1 = new Sala("Verde", 10, Ubicacion.NORTE); Sala sala2 = new Sala("Azul", 20, Ubicacion.SUR); gestorSalas.agregarSala(sala1); gestorSalas.agregarSala(sala2); System.out.println (gestorSalas.getTodas()); </pre>	<pre> Test agregarSala / getTodas [Sala Verde (capacidad: 10, ubicacion: Norte), Sala Azul (capacidad: 20, ubicacion: Sur)] </pre>	<pre> Test agregarSala / getTodas [Sala Verde (capacidad: 10, ubicacion: Norte), Sala Azul (capacidad: 20, ubicacion: Sur)] </pre>	✓
✓	<pre> System.out.println("Test eliminarSala"); GestorSalas gestorSalas = new GestorSalas(); Sala sala1 = new Sala("Verde", 10, Ubicacion.NORTE); Sala sala2 = new Sala("Azul", 20, Ubicacion.SUR); gestorSalas.agregarSala(sala1); gestorSalas.agregarSala(sala2); gestorSalas.eliminarSala ("Verde"); System.out.println (gestorSalas.getTodas()); </pre>	<pre> Test eliminarSala [Sala Azul (capacidad: 20, ubicacion: Sur)] </pre>	<pre> Test eliminarSala [Sala Azul (capacidad: 20, ubicacion: Sur)] </pre>	✓
✓	<pre> System.out.println("Test modificarSala"); GestorSalas gestorSalas = new GestorSalas(); Sala sala1 = new Sala("Verde", 10, Ubicacion.NORTE); gestorSalas.agregarSala(sala1); gestorSalas.modificarSala("Verde", 5, Ubicacion.ESTE); System.out.println (gestorSalas.getTodas()); </pre>	<pre> Test modificarSala [Sala Verde (capacidad: 5, ubicacion: Este)] </pre>	<pre> Test modificarSala [Sala Verde (capacidad: 5, ubicacion: Este)] </pre>	✓
✓	<pre> System.out.println("Test buscarSala"); GestorSalas gestorSalas = new GestorSalas(); Sala sala1 = new Sala("Verde", 10, Ubicacion.NORTE); Sala sala2 = new Sala("Azul", 20, Ubicacion.SUR); gestorSalas.agregarSala(sala1); gestorSalas.agregarSala(sala2); Sala s = gestorSalas.buscarSala ("Azul"); System.out.println (s); s = gestorSalas.buscarSala ("Roja"); System.out.println (s); </pre>	<pre> Test buscarSala Sala Azul (capacidad: 20, ubicacion: Sur) null </pre>	<pre> Test buscarSala Sala Azul (capacidad: 20, ubicacion: Sur) null </pre>	✓

Todas las pruebas superadas. ✓

Correcta

Puntos para este envío: 20/20.

Pregunta 6

Correcta

Se puntúa 25 sobre 25

La clase **GestorReservas** administra las reservas de Salas. Posee un **GestorSalas** que gestiona las Salas, y un **Mapa** que relaciona cada **Sala** con una Lista de Reservas.

Hay que implementar los siguientes métodos.

- Agregar una reserva a una Sala (reservar).
- Libera una reserva de una Sala (liberar).
- Lista todas las Salas disponibles (disponibles).

Nota: puede utilizar los métodos **seSuperpone** de la clase Reserva para determinar si un horario y duración se superpone con la reserva.

Objetivos evaluados: uso de colecciones (List, Map), búsqueda, eliminación y modificación de elementos.

Por ejemplo:

Prueba	Resultado
<pre>// Setup del TEST: Crea gestores, salas, usuarios, y hora GestorSalas gestorSalas = new GestorSalas(); GestorReservas gestorReservas = new GestorReservas(gestorSalas); Sala sala1 = new Sala("Verde", 10, Ubicacion.ALA_CENTRO); gestorSalas.agregarSala(sala1); Usuario ana = new Usuario("Ana", "ana@mail.com"); LocalDateTime ahora = LocalDateTime.of(2025, 6, 15, 10, 0); // Test Reservar System.out.println("Reservando Salas..."); try { gestorReservas.reservar("Verde", ana, ahora, 2); } catch (Exception e) { System.out.println("Error al reservar: " + e.getMessage()); } // Verifica las reservas System.out.println("\n\nReservas"); for (Reserva r : gestorReservas.getReservasPorSala("Verde")){ System.out.println(r); } // Test disponibles System.out.println("\n\nSalas disponibles para una reunion de 1hr a las "+ahora); for (Sala s : gestorReservas.disponibles(ahora, 1)) { System.out.println(s); } System.out.println("\n\nSalas disponibles para una reunion de 1hr a las "+ahora.plusHours(2)); for (Sala s : gestorReservas.disponibles(ahora.plusHours(2), 1)) { System.out.println(s); } // Test liberar Sala System.out.println("\n\nLiberando sala Verde a las " + ahora); gestorReservas.liberar("Verde", ahora); // Verifica reservas System.out.println("\nReservas sala Verde:"); for (Reserva r : gestorReservas.getReservasPorSala("Verde")) { System.out.println(r); }</pre>	<p>Reservando Salas...</p> <p>Reservas Reserva de Verde por Ana <ana@mail.com> desde 2025-06-15T10:00 por 2h</p> <p>Salas disponibles para una reunion de 1hr a las 2025-06-15T10:00</p> <p>Salas disponibles para una reunion de 1hr a las 2025-06-15T12:00 Sala Verde (capacidad: 10, ubicacion: Ala_centro)</p> <p>Liberando sala Verde a las 2025-06-15T10:00</p> <p>Reservas sala Verde:</p>

Respuesta: (sistema de penalización: 0, 0, 0, 5, 10, 15 %)

Reiniciar respuesta

```
1 import java.util.*;
2 import java.time.LocalDateTime;
3
4 public class GestorReservas {
5
6     private GestorSalas gestorSalas;
```

```

7   protected Map<Sala, List<Reserva>> reservas;
8
9   public GestorReservas(GestorSalas gestorSalas) {
10      this.gestorSalas = gestorSalas;
11      this.reservas = new HashMap<>();
12   }
13
14   public List<Reserva> getReservasPorSala(String nombreSala) {
15      Sala sala = gestorSalas.buscarSala(nombreSala);
16      if (sala != null && reservas.containsKey(sala)) {
17          return reservas.get(sala);
18      }
19      return new ArrayList<>();
20   }
21
22   /**
23    * Reserva una sala si existe y no hay superposición de reservas
24    */
25   public boolean reservar(String nombreSala, Usuario usuario, LocalDateTime inicio, int duracionHoras) {
26
27      Sala sala = gestorSalas.buscarSala(nombreSala);
28      if (sala == null) {
29          throw new Exception("La sala no existe.");
30      }
31
32      List<Reserva> lista = reservas.getOrDefault(sala, new ArrayList<>());
33
34      // Verificar superposición
35      for (Reserva r : lista) {
36          if (r.seSuperpone(inicio, duracionHoras)) {
37              throw new Exception("La sala ya está reservada en ese horario.");
38          }
39      }
40
41      // Crear nueva reserva (requiere Sala, Usuario, LocalDateTime, int)
42      Reserva nueva = new Reserva(sala, usuario, inicio, duracionHoras);
43
44      lista.add(nueva);
45      reservas.put(sala, lista);
46
47      return true;
48   }
49
50   /**
51    * Elimina la reserva que comience exactamente a esa hora
52   */

```

	Prueba	Esperado	Conseguido	
✓	<pre> // Setup del TEST: Crea gestores, salas, usuarios, y hora GestorSalas gestorSalas = new GestorSalas(); GestorReservas gestorReservas = new GestorReservas(gestorSalas); Sala sala1 = new Sala("Verde", 10, Ubicacion.ALA_CENTRO); gestorSalas.agregarSala(sala1); Usuario ana = new Usuario("Ana", "ana@mail.com"); LocalDateTime ahora = LocalDateTime.of(2025, 6, 15, 10, 0); // Test Reservar sala inexistente System.out.println("Reservando Sala inexistente..."); try { gestorReservas.reservar("Azul", ana, ahora, 2); } catch (Exception e) { System.out.println("Error al reservar: "); } </pre>	<pre> Reservando Sala inexistente... Error al reservar: </pre>	<pre> Reservando Sala inexistente... Error al reservar: </pre>	✓

	Prueba	Esperado	Conseguido	
✓	<pre>// Setup del TEST: Crea gestores, salas, usuarios, y hora GestorSalas gestorSalas = new GestorSalas(); GestorReservas gestorReservas = new GestorReservas(gestorSalas); Sala sala1 = new Sala("Verde", 10, Ubicacion.ALA_CENTRO); gestorSalas.agregarSala(sala1); Usuario ana = new Usuario("Ana", "ana@mail.com"); LocalDateTime ahora = LocalDateTime.of(2025, 6, 15, 10, 0); // Test Reservar System.out.println("Reservando Salas..."); try { gestorReservas.reservar("Verde", ana, ahora, 2); gestorReservas.reservar("Verde", ana, ahora, 1); } catch (Exception e) { System.out.println("Error al reservar: "); } // Verifica las reservas for (Reserva r : gestorReservas.getReservasPorSala("Verde")) { System.out.println(r); }</pre>	<pre>Reservando Salas... Error al reservar: Reserva de Verde por Ana <ana@mail.com> desde 2025-06- 15T10:00 por 2h</pre>	<pre>Reservando Salas... Error al reservar: Reserva de Verde por Ana <ana@mail.com> desde 2025-06- 15T10:00 por 2h</pre>	✓
✓	<pre>// Setup del TEST: Crea gestores, salas, usuarios, y hora GestorSalas gestorSalas = new GestorSalas(); GestorReservas gestorReservas = new GestorReservas(gestorSalas); Sala sala1 = new Sala("Verde", 10, Ubicacion.ALA_CENTRO); gestorSalas.agregarSala(sala1); Usuario ana = new Usuario("Ana", "ana@mail.com"); LocalDateTime ahora = LocalDateTime.of(2025, 6, 15, 10, 0); List<Reserva> l = new ArrayList<Reserva>(); l.add(new Reserva(sala1,ana,ahora,1)); gestorReservas.reservas = new HashMap<Sala,List<Reserva>>(); gestorReservas.reservas.put (sala1,l); // Verifica las reservas System.out.println("Reservas"); for (Reserva r : gestorReservas.getReservasPorSala("Verde")) { System.out.println(r); } // Test liberar Sala System.out.println("\n\nLiberando sala Verde a las " + ahora); gestorReservas.liberar("Verde", ahora); // Verifica reservas System.out.println("\nReservas"); for (Reserva r : gestorReservas.getReservasPorSala("Verde")) { System.out.println(r); }</pre>	<pre>Reservas Reserva de Verde por Ana <ana@mail.com> desde 2025-06- 15T10:00 por 1h Liberando sala Verde a las 2025- 06-15T10:00 Reservas</pre>	<pre>Reservas Reserva de Verde por Ana <ana@mail.com> desde 2025-06- 15T10:00 por 1h Liberando sala Verde a las 2025- 06-15T10:00 Reservas</pre>	✓

	Prueba	Esperado	Conseguido	
✓	<pre>// Setup del TEST: Crea gestores, salas, usuarios, y hora GestorSalas gestorSalas = new GestorSalas(); GestorReservas gestorReservas = new GestorReservas(gestorSalas); Sala sala1 = new Sala("Verde", 10, Ubicacion.ALA_CENTRO); Sala sala2 = new Sala("Azul", 20, Ubicacion.ALA_SUR); gestorSalas.agregarSala(sala1); gestorSalas.agregarSala(sala2); Usuario ana = new Usuario("Ana", "ana@mail.com"); Usuario luis = new Usuario("Luis", "luis@mail.com"); LocalDateTime ahora = LocalDateTime.of(2025, 6, 15, 10, 0); List<Reserva> l = new ArrayList<Reserva>(); l.add(new Reserva(sala1,ana,ahora,1)); l.add(new Reserva(sala1,ana,ahora.plusHours(3),1)); List<Reserva> l2 = new ArrayList<Reserva>(); l2.add(new Reserva(sala2,luis,ahora.plusHours(2),1)); gestorReservas.reservas = new HashMap<Sala,List<Reserva>>(); gestorReservas.reservas.put (sala1,l); gestorReservas.reservas.put (sala2,l2); System.out.println("Salas disponibles para reunion de 1hr a las "+ahora); for (Sala s : gestorReservas.disponibles(ahora, 1)) { System.out.println(s); } System.out.println("\nSalas disponibles para reunion de 3hr a las "+ahora); for (Sala s : gestorReservas.disponibles(ahora, 3)) { System.out.println(s); } System.out.println("\nSalas disponibles para reunion de 1hr a las "+ahora.plusHours(4)); for (Sala s : gestorReservas.disponibles(ahora.plusHours(4), 1)) { System.out.println(s); } }</pre>	<p>Salas disponibles para reunion de 1hr a las 2025-06-15T10:00 Sala Azul (capacidad: 20, ubicacion: Ala_sur)</p> <p>Salas disponibles para reunion de 3hr a las 2025-06-15T10:00</p> <p>Salas disponibles para reunion de 1hr a las 2025-06-15T14:00 Sala Verde (capacidad: 10, ubicacion: Ala_centro) Sala Azul (capacidad: 20, ubicacion: Ala_sur)</p>	<p>Salas disponibles para reunion de 1hr a las 2025-06-15T10:00 Sala Azul (capacidad: 20, ubicacion: Ala_sur)</p> <p>Salas disponibles para reunion de 3hr a las 2025-06-15T10:00</p> <p>Salas disponibles para reunion de 1hr a las 2025-06-15T14:00 Sala Verde (capacidad: 10, ubicacion: Ala_centro) Sala Azul (capacidad: 20, ubicacion: Ala_sur)</p>	✓

	Prueba	Esperado	Conseguido	
✓	<pre>// Setup del TEST: Crea gestores, salas, usuarios, y hora GestorSalas gestorSalas = new GestorSalas(); GestorReservas gestorReservas = new GestorReservas(gestorSalas); Sala sala1 = new Sala("Verde", 10, Ubicacion.ALA_CENTRO); Sala sala2 = new Sala("Azul", 20, Ubicacion.ALA_SUR); gestorSalas.agregarSala(sala1); gestorSalas.agregarSala(sala2); Usuario ana = new Usuario("Ana", "ana@mail.com"); Usuario luis = new Usuario("Luis", "luis@mail.com"); LocalDateTime ahora = LocalDateTime.of(2025, 6, 15, 10, 0); // Test Reservar System.out.println("Reservando Salas..."); try { gestorReservas.reservar("Verde", ana, ahora, 2); gestorReservas.reservar("Verde", ana, ahora.plusHours(3), 1); gestorReservas.reservar("Azul", luis, ahora.plusHours(1), 1); } catch (Exception e) { System.out.println("Error al reservar: " + e.getMessage()); } // Verifica las reservas System.out.println("\n\nReservas"); for (Reserva r : gestorReservas.getReservasPorSala("Verde")) { System.out.println(r); } for (Reserva r : gestorReservas.getReservasPorSala("Azul")) { System.out.println(r); } // Test disponibles System.out.println("\n\nSalas disponibles para una reunion de 1hr a las "+ahora); for (Sala s : gestorReservas.disponibles(ahora, 1)) { System.out.println(s); } // Test liberar Sala System.out.println("\n\nLiberando sala Verde a las " + ahora); gestorReservas.liberar("Verde", ahora); // Verifica reservas System.out.println("\n\nReservas sala Verde:"); for (Reserva r : gestorReservas.getReservasPorSala("Verde")) { System.out.println(r); } // Verifica disponibles System.out.println("\n\nSalas disponibles tras liberar para reunion de 1hr a las "+ahora); for (Sala s : gestorReservas.disponibles(ahora, 1)) { System.out.println(s); } }</pre>	<pre>Reservando Salas... Reservas Reserva de Verde por Ana <ana@mail.com> desde 2025-06- 15T10:00 por 2h Reserva de Verde por Ana <ana@mail.com> desde 2025-06- 15T13:00 por 1h Reserva de Azul por Luis <luis@mail.com> desde 2025-06- 15T11:00 por 1h Salas disponibles para una reunion de 1hr a las 2025-06- 15T10:00 Sala Azul (capacidad: 20, ubicacion: Ala_sur) Liberando sala Verde a las 2025- 06-15T10:00 Reservas sala Verde: Reserva de Verde por Ana <ana@mail.com> desde 2025-06- 15T13:00 por 1h Salas disponibles tras liberar para reunion de 1hr a las 2025- 06-15T10:00 Sala Verde (capacidad: 10, ubicacion: Ala_centro) Sala Azul (capacidad: 20, ubicacion: Ala_sur)</pre>	<pre>Reservando Salas... Reservas Reserva de Verde por Ana <ana@mail.com> desde 2025-06- 15T10:00 por 2h Reserva de Verde por Ana <ana@mail.com> desde 2025-06- 15T13:00 por 1h Reserva de Azul por Luis <luis@mail.com> desde 2025-06- 15T11:00 por 1h Salas disponibles para una reunion de 1hr a las 2025-06- 15T10:00 Sala Azul (capacidad: 20, ubicacion: Ala_sur) Liberando sala Verde a las 2025- 06-15T10:00 Reservas sala Verde: Reserva de Verde por Ana <ana@mail.com> desde 2025-06- 15T13:00 por 1h Salas disponibles tras liberar para reunion de 1hr a las 2025- 06-15T10:00 Sala Verde (capacidad: 10, ubicacion: Ala_centro) Sala Azul (capacidad: 20, ubicacion: Ala_sur)</pre>	✓

Todas las pruebas superadas. ✓

Correcta

Puntos para este envío: 25/25.