

Sergio Ruiz Montesinos



Proyecto 60% 2ºevaluacion reserva de aulas - 1ºDAW

Sergio Ruiz Montesinos

Profesores: José Luis López, Esteban Castañer, Mario Guzman

25/2/2026

Reserva de aulas

## Contenido

3. Resumen ejecutivo .....	3
4. Introducción .....	3
5. Análisis y diseño .....	4
• Descripción del problema a resolver.....	4
• Diseño del modelo de objetos: clases usuario, reserva, préstamo... .....	4
6. Desarrollo técnico .....	5
• Descripción de las clases principales y sus responsabilidades .....	5
• Métodos más relevantes .....	5
• Uso de colecciones (ArrayList).....	6
• Gestión de flujo del programa mediante bucles y estructuras de control .....	6
• Fragmentos de código .....	6
7. Gestión de excepciones .....	8
• Descripción de las excepciones personalizadas implementadas.....	8
• Uso de try-catch y control de errores en la entrada de datos .....	9
• Valoración de la robustez del programa.....	9
8. Pruebas y validación .....	10
• Casos de prueba realizados .....	10
• Resultados obtenidos .....	10
9. Conclusiones y mejoras.....	10
• Dificultades encontradas durante el desarrollo .....	10

• Soluciones adoptadas .....	11
• Propuestas de mejora .....	11
• Competencias profesionales desarrolladas .....	11
10. Bibliografía y webgrafía .....	12
Apéndices.....	13

### 3. Resumen ejecutivo

Desarrollar una aplicación de consola para la gestión de reservas de puestos de estudio en aulas o bibliotecas y del préstamo/devolución de equipos tecnológicos portátiles o tablets, asegurando un control eficiente de usuarios, recursos y sanciones por retrasos

### 4. Introducción

El proyecto consiste en desarrollar un sistema de gestión de reservas de puestos y prestamos de equipos tecnológicos para aulas y bibliotecas. Permitirá a alumnos y profesores reservar puestos, gestionar prestamos de portátiles y tablets, controlar retrasos y aplicar sanciones si es necesario.

Se implementará como aplicación de consola en java, usando programación orientada a objetos, con persistencia en MYSQL y exportación de datos a XML.

## 5. Análisis y diseño

- Descripción del problema a resolver

El proyecto busca automatizar y centralizar estas tareas mediante un sistema que permita reservar puestos, gestionar préstamos y devoluciones, controlar retrasos y generar informes, mejorando la eficiencia y la organización de los recursos educativos.

- Diseño del modelo de objetos: clases usuario, reserva, préstamo...

El sistema se basa en programación orientada a objetos, con clases que representan las entidades clave y sus relaciones

Usuario: representa alumnos y profesores almacenando información como nombre, idAlumno

Reserva: gestiona la asignación de un puesto a un usuario registrando fecha, hora y duración de la reserva.

Préstamo: controla la entrega y devolución de los equipos, calculando retrasos y aplicando sanciones cuando corresponde.

## 6. Desarrollo técnico

- Descripción de las clases principales y sus responsabilidades

El sistema se basa en programación orientada a objetos, organizando la información y funcionalidades mediante clases que representan las entidades clave

Usuario: representa a los alumnos y profesores que interactúan con el sistema.

Almacena datos personales, gestiona sus reservas y prestamos, consultar sanciones y mantener un historial de actividad

Reserva: representa la asignación de un puesto a un usuario para un horario específico. Crea, eliminar y modifica reservas, registra fechas, hora y duración

Préstamo: gestiona la entrega y devolución de equipos a los usuarios. Registra fecha de préstamo y devolución, calcular retrasos automáticamente,

- Métodos más relevantes

Cada clase del sistema cuenta con métodos específicos que permiten implementar la lógica de negocio y gestionar la información de manera eficiente

Usuario: modificarDatos

Préstamo: estaRetrasado, devolver

GestorPrestamos: registrarDevolucion, listarPrestamosActivos

gestorReservas: liberarPuesto, mostrarEstadoAula

- Uso de colecciones (ArrayList)

En mi proyecto lo utilizo para almacenar y gestionar dinámicamente los objetos principales, como usuarios, reservas, y préstamos.

- Gestión de flujo del programa mediante bucles y estructuras de control

El flujo de ejecución del sistema se controla principalmente mediante bucles do-while y estructuras de control switch y condicionales if-else, lo que permite ofrecer un menú interactivo de consola para gestionar usuarios, reservas y prestamos

- Fragmentos de código

```

1  public abstract class usuario {
2      private int idusuario;    Field idUsuario can be final
3      private String nombre;
4      private String apellidos;
5      private String dni;     Field dni can be final
6      private boolean activo;
7
8      public usuario(int idUsuario, String nombre, String apellidos, String dni) {
9          this.idUsuario = idUsuario;
10         this.nombre = nombre;
11         this.apellidos = apellidos;
12         this.dni = dni;
13         this.activo = true;
14     }
15

```

```

1  public class profesor extends usuario {
2      private int idprofesor;    Field idProfesor can be final
3
4      public profesor(int idUsuario, String dni, String nombre, String apellidos, int idProfesor){
5          super(idUsuario, dni, nombre, apellidos);
6          this.idProfesor = idProfesor;
7      }
8      @Override
9      public int getMaxPrestamos(){
10         return 1;
11     }
12     @Override
13     public String toString(){
14         return super.toString() + ", idProfesor = " + idProfesor;
15     }
16
17 }
18

```

```
1 import java.util.ArrayList;
2 import java.util.List;
3 public class gestorReservas {
4     private List<reserva> reservas = new ArrayList<>();    Field reservas can be final
5     public void reservar (usuario u, aula a, int idPuesto, String franjaHoraria) {
6
7         puesto p = a.buscarPuesto(idPuesto);
8         if (p == null) {
9             System.out.println(x: "el puesto no existe");
10            return;
11        }
12
13        if (p.isOcupado()) [
14            System.out.println(x: "el puesto esta ocupado");
15            return;
16        ]
17
18        p.ocupar();
19        reservas.add(new reserva(u, a, p, franjaHoraria));
20        System.out.println(x: "la reserva se ha realizado con exito");
21    }
22
23    public void liberarPuesto(aula a, int idPuesto) {
24
25        puesto p = a.buscarPuesto(idPuesto);
26        if (p != null && p.isOcupado()) {
27            p.liberar();
28            System.out.println(x: "el puesto fue liberado correctamente");
29        } else {
30            System.out.println(x: "el puesto no esta ocupado");
31        }
32    }
33
34    public void mostrarEstadoAula(aula a) {
35        System.out.println("estado del aula" + a.getNombre());
36        for (puesto p : a.getPuestos()) {
37            System.out.println("puesto" + p.getIdPuesto() +
38                (p.isOcupado() ? "ocupado" : "libre"));
39        }
40    }
41 }
42 }
```

```
1 import java.time.LocalDate;
2 public class prestamo {
3     private usuario usuario;      Field usuario can be final
4     public equipo equipo;
5     private LocalDate fechaPrestamo;    Field fechaPrestamo can be final
6     public LocalDate fechaDevolucion;
7
8     public prestamo(usuario usuario, equipo equipo) {
9         this.usuario = usuario;
10        this.equipo = equipo;
11        this.fechaPrestamo = LocalDate.now();
12        this.fechaDevolucion = null;
13        equipo.prestar();
14    }
15
16    public boolean estaRetrasado() {
17        if (fechaDevolucion != null) return false;
18        return fechaPrestamo.plusDays(daysToAdd: 7).isBefore(LocalDate.now());
19    }
20
21    public void devolver() {
22        fechaDevolucion = LocalDate.now();
23        equipo.devolver();
24    }
25
26    @Override
27    public String toString() {
28        String estado = (fechaDevolucion == null) ? (estaRetrasado() ? "Retrasado" : "en curso ") : "Devuelto";
29        return "Prestamo: " + usuario.getIdUsuario() + " - " + equipo + " - Estado: " + estado;
30    }
31
32 }
33 }
```

## 7. Gestión de excepciones

- Descripción de las excepciones personalizadas implementadas

En el sistema, la gestión de excepciones permite controlar errores que pueden ocurrir durante la ejecución del programa.

Intentar registrar un usuario con datos inválidos, reservar un puesto不存在 o ya ocupado, registrar un préstamo de un equipo que no está disponible, consultar usuarios, reservas o préstamos inexistentes

- Uso de try-catch y control de errores en la entrada de datos

Se realiza mediante menús de consola. Es importante controlar posibles errores de entrada, como introducir letras en lugar de números o seleccionar opciones inexistentes

- Valoración de la robustez del programa

En el menú de consola y al registrar datos de usuarios, reservas o prestamos, se controla que la información introducida sea valida

Se evita que se ingresen letras donde se espera un número, se comprueba que los datos obligatorios como DNI, nombre no estén vacíos.

Se verifica que los usuarios estén activos, los puestos libres y los equipos disponibles.

Los ArrayList se recorren con bucles y métodos de búsqueda para evitar errores de índices

El menú principal con bucle do-while y switch permite corregir errores sin cerrar el programa

## 8. Pruebas y validación

- Casos de prueba realizados

Probar el registro de alumnos y profesores con datos válidos, el sistema muestra un mensaje de error al intentar registrar un usuario con datos incompletos o vacíos. Se probaron entradas invalidas en el menú y en las operaciones texto en lugar de números, números fuera de rango...

- Resultados obtenidos

Tras el desarrollo y las pruebas realizadas, el sistema cumple correctamente con los objetivos planteados al inicio del proyecto. Se ha conseguido implementar un programa funcional que permite registrar usuarios, gestionar reservas de puestos y controlar el préstamo y devolución de equipos. El programa también responde adecuadamente ante situaciones incorrectas, como intentar reservar un puesto ocupado o prestar un equipo no disponible, mostrando mensajes claros sin cerrarse inesperadamente.

## 9. Conclusiones y mejoras

- Dificultades encontradas durante el desarrollo

Organizar correctamente las clases y sus relaciones, asegurando que cada una tuviera su responsabilidad sin mezclar funciones, también el control de errores en la entrada de datos, especialmente cuando el usuario introduce valores incorrectos en el menú o en los formularios de registro.

- Soluciones adoptadas

Se reorganizo el diseño del programa aplicando correctamente la programación orientada a objetos, separando las responsabilidades de cada clase, en cuanto a los errores de entrada de datos se implementaron validaciones y estructuras de control como if-else y try-catch.

- Propuestas de mejora

Optimizar el código y generar mejores informes y estadísticas, seguridad y control de acceso, interfaz de usuario

- Competencias profesionales desarrolladas

Durante el desarrollo de proyecto se han reforzado competencias en programación orientada a objetos, diseño de clases y organización del código. También se ha mejorado la capacidad de resolución de problemas, el manejo de colecciones, estructuras de control y gestión de errores. Además, se han adquirido conocimientos en el uso de bases de datos y generación de archivos XML

## 10. Bibliografía y webgrafía

TecnoDigital (2025) Guía completa sobre getters y setters en Java

<https://informatedigital.com/getters-y-setters-en-java/>

datacamp.com. (2026) super palabra clave en java.

[https://www.datacamp.com/es/doc/java/super?dc\\_referrer=https%3A%2F%2Fes.search.yahoo.com%2F](https://www.datacamp.com/es/doc/java/super?dc_referrer=https%3A%2F%2Fes.search.yahoo.com%2F)

Maldonado, R. (2026) @override en java: ¿Qué es y por qué usarlo siempre?

<https://keepcoding.io/blog/que-es-override-en-java/>

## Apéndices

Figura 1

```
public abstract class usuario {
    private int idUsuario;      Field idUsuario can be final
    private String nombre;
    private String apellidos;
    private String dni;        Field dni can be final
    private boolean activo;

    public usuario(int idUsuario, String nombre, String apellidos, String dni) {
        this.idUsuario = idUsuario;
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.dni = dni;
        this.activo = true;
    }

    public int getIdUsuario(){
        return idUsuario;
    }

    public String getNombre() {
        return nombre;
    }
    public String getApellidos() {
        return apellidos;
    }

    public String getDni() {
        return dni;
    }

    public boolean isActive() {
        return activo;
    }
    public void darBaja() {
        this.activo = false;
    }
    public void modificarDatos(String nombre, String apellidos) {
        this.nombre = nombre;
        this.apellidos = apellidos;
    }

    public abstract int getMaxPrestamos();
}
```

Figura 2

```
1 ✓ public class alumno extends usuario {
2     private String nia;      Field nia can be final
3
4 ✓     public alumno(int idUsuario, String dni, String nombre, String apellidos, String nia) {
5         super(idUsuario, nombre, apellidos, dni);
6         this.nia = nia;
7     }
8
9     @Override
10    public int getMaxPrestamos() {
11        return 1;
12    }
13
14    @Override
15    public String toString() {
16        return super.toString() + ", nia = " + nia;
17    }
18 }
19
```

Figura 3

```
import java.util.ArrayList;
import java.util.List;

public class gestorUsuario {
    private List<usuario> usuarios = new ArrayList<>();
    private int contadorId = 1;

    public gestorUsuario() {
        usuarios = new ArrayList<>();
    }

    public void registrarAlumno(String dni, String nombre, String apellidos, String nia) {
        usuarios.add(new alumno(contadorId++, dni, nombre, apellidos, nia));
        System.out.println("El alumno ha sido registrado");

        XMLUsuarios.guardarUsuario(usuarios.get(usuarios.size() - 1));
    }

    public void registrarProfesor(String dni, String nombre, String apellidos, int idProfesor) {
        usuarios.add(new profesor(contadorId++, dni, nombre, apellidos, idProfesor));
        System.out.println("El profesor ha sido registrado");
    }

    XMLUsuarios.guardarUsuario(usuarios.get(usuarios.size() - 1));

}

public void listarUsuarios() {
    if (usuarios.isEmpty()) {
        System.out.println("No hay ningún usuario registrado");
    } else {
        usuarios.forEach(System.out::println);
    }
}

public void darBajaUsuario(int idUsuario) {
    usuario u = buscarUsuario(idUsuario);

    if (u != null) {
        u.darBaja();
        System.out.println("el usuario a sido dado de baja");
    } else {
        System.out.println("no se ha encontrado el usuario");
    }
}
```

Figura 4

```

        }
    }

    public void modificarUsuario(int idUsuario, String nuevoNombre, String nuevosApellidos) {
        usuario u = buscarUsuario(idUsuario);
        if (u != null) {
            u.modificarDatos(nuevoNombre, nuevosApellidos);
            System.out.println("usuario modificado");
        } else {
            System.out.println("usuario no encontrado");
        }
    }

    public usuario buscarUsuario(int id) {
        for (usuario u : usuarios) {
            if (u.getIdUsuario() == id) {
                return u;
            }
        }
        return null;
    }
}

```

Figura 5

```

1  public class profesor extends usuario {
2      private int idProfesor;      Field idProfesor can be final
3
4      public profesor(int idUsuario, String dni, String nombre, String apellidos, int idProfesor){
5          super(idUsuario, nombre, apellidos, dni);
6          this.idProfesor = idProfesor;
7      }
8      @Override
9      public int getMaxPrestamos(){
10         return 1;
11     }
12     @Override
13     public String toString(){
14         return super.toString() + ", idProfesor = " + idProfesor;
15     }
16
17 }
18

```

Figura 6

```

1 import java.util.ArrayList;
2 import java.util.List;
3
4 public class aula {
5
6     private String nombre;      Field nombre can be final
7     private List<puesto> puestos = new ArrayList<>();    Field puesto
8
9     public aula(String nombre, int cantidadPuestos) {
10         this.nombre = nombre;
11
12         for (int i = 1; i <= cantidadPuestos; i++) {
13             puestos.add(new puesto(i));
14         }
15     }
16
17     public String getNombre() {
18         return nombre;
19     }
20
21     public List<puesto> getPuestos() {
22         return puestos;
23     }
24
25     public puesto buscarPuesto(int id) {
26         for (puesto p : puestos) {
27             if (p.getIdPuesto() == id) {
28                 return p;
29             }
30         }
31         return null;
32     }
33 }
34

```

Figura 7

```

1 import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.ResultSet;
4 import java.sql.Statement;
5 public class App {
Run | Debug | Run main | Debug main
6     public static void main(String[] args) {
7         String host = "192.168.150.128"; // ip de la máquina virtual
8         int port = 3306; // puerto para acceder a mariadb, por defecto 3306
9         String database = "reservaDeAulas"; //nombre de tu base de datos
10        String user = "sergio"; // usuario creado en el paso 5.2
11        String password = "1234"; // password creado en el paso 5.2
12        String url = "jdbc:mariadb://" + host + ":" + port + "/" + database;
13        try (Connection conn = DriverManager.getConnection(url, user, password);
14             Statement stmt = conn.createStatement();
15             ResultSet rs = stmt.executeQuery(sql: "SELECT 1 AS resultado")) {
16            rs.next();
17            System.out.println("Conexión exitosa. Resultado: " + rs.getInt(columnLabel: "resultado"));
18        } catch (Exception e) {
19            System.err.println(x: "Error al conectar con MariaDB:");
20            e.printStackTrace();    Print Stack Trace
21        }
22    }
23 }
24

```

Figura 8

```

1 public class puesto {
2     private int idPuesto;      Field idPuesto can be final
3     private boolean ocupado;
4
5     public puesto (int idPuesto) {
6         this.idPuesto = idPuesto;
7         this.ocupado = false;
8     }
9
10    public int getIdPuesto() {
11        return idPuesto;
12    }
13
14    public boolean isOcupado() {
15        return ocupado;
16    }
17
18    public void ocupar() {
19        this.ocupado = true;
20    }
21
22    public void liberar() {
23        ocupado = false;
24    }
25
26    @Override
27    public String toString() {
28        return "Puesto, " + idPuesto + ", Ocupado: " + ocupado;
29    }
30
31 }
```

Figura 9

```

1 public class reserva {
2
3     private usuario usuario;      Field usuario can be final
4     private aula aula;      Field aula can be final
5     private puesto puesto;      Field puesto can be final
6     private String franjaHoraria;      Field franjaHoraria can be final
7
8     public reserva(usuario usuario, aula aula, puesto puesto, String franjaHoraria) {
9         this.usuario = usuario;
10        this.aula = aula;
11        this.puesto = puesto;
12        this.franjaHoraria = franjaHoraria;
13    }
14
15    @Override
16    public String toString() {
17        return "Reserva de " + usuario +
18            "usuario=" + usuario +
19            ", aula=" + aula.getNombre() +
20            ", puesto=" + puesto.getIdPuesto() +
21            ", horario=" + franjaHoraria;
22    }
23
24 }
```

Figura 10

```
1 import java.util.ArrayList;
2 import java.util.List;
3 public class gestorReservas {
4     private List<Reserva> reservas = new ArrayList<>(); Field reservas can be final
5     public void reservar (Usuario u, Aula a, int idPuesto, String franjaHoraria) {
6
7         Puesto p = a.buscarPuesto(idPuesto);
8         if (p == null) {
9             System.out.println(x: "el puesto no existe");
10            return;
11        }
12
13        if (p.isOcupado()) {
14            System.out.println(x: "el puesto esta ocupado");
15            return;
16        }
17
18        p.ocupar();
19        reservas.add(new Reserva(u, a, p, franjaHoraria));
20        System.out.println(x: "la reserva se ha realizado con exito");
21    }
22
23    public void liberarPuesto(Aula a, int idPuesto) {
24
25        Puesto p = a.buscarPuesto(idPuesto);
26        if (p != null && p.isOcupado()) {
27            p.liberar();
28            System.out.println(x: "el puesto fue liberado correctamente");
29        } else {
30            System.out.println(x: "el puesto no esta ocupado");
31        }
32    }
33
34    public void mostrarEstadoAula(Aula a) {
35        System.out.println("estado del aula" + a.getNombre());
36        for (Puesto p : a.getPuestos()) {
37            System.out.println("puesto" + p.getIdPuesto() +
38                (p.isOcupado() ? "ocupado" : "libre"));
39        }
40    }
41 }
```

Figura 11

```
1 import java.time.LocalDate;
2 public class prestamo {
3     private usuario usuario;    Field usuario can be final
4     public equipo equipo;
5     private LocalDate fechaPrestamo;    Field fechaPrestamo can be final
6     public LocalDate fechaDevolucion;
7
8     public prestamo(usuario usuario, equipo equipo) {
9         this.usuario = usuario;
10        this.equipo = equipo;
11        this.fechaPrestamo = LocalDate.now();
12        this.fechaDevolucion = null;
13        equipo.prestar();
14    }
15
16    public boolean estaRetrasado() {
17        if (fechaDevolucion != null) return false;
18        return fechaPrestamo.plusDays(daysToAdd: 7).isBefore(LocalDate.now());
19    }
20
21    public void devolver() {
22        fechaDevolucion = LocalDate.now();
23        equipo.devolver();
24    }
25
26    @Override
27    public String toString() {
28        String estado = (fechaDevolucion == null) ? (estaRetrasado()) ? "Retrasado" : "en curso " : "Devuelto";
29        return "Prestamo: " + usuario.getIdUsuario() + " - " + equipo + " - Estado: " + estado;
30    }
31
32 }
```

Figura 12

```

1 import java.util.ArrayList;
2 import java.util.List;
3 public class gestorPrestamos {
4     private List<prestamo> prestamos = new ArrayList<>(); Field prestamos can be final
5     public void registrarPrestamo(usuario u, equipo e) {
6         if (!e.isDisponible()){
7             System.out.println(x: "Equipo no disponible");
8             return;
9         }
10        prestamos.add(new prestamo(u, e));
11        System.out.println("Préstamo registrado: " + u.getIdUsuario() + " -> " + e.getTipo() + " " + e.getIdEquipo());
12    }
13
14    public void registrarDevolucion(equipo e) {
15        for (prestamo p : prestamos) {
16            if (p.equipo == e && p.fechaDevolucion == null) {
17                p.devolver();
18                if (p.estáRetrasado()) {
19                    System.out.println(x: "Devolución hecha con retraso.");
20                }else{
21                    System.out.println(x: "Devolución hecha a tiempo.");
22                }
23                return;
24            }
25        }
26        System.out.println(x: "No se ha encontrado ningún préstamo activo para este equipo.");
27    }
28
29    public void listarPrestamosActivos() {
30        System.out.println(x: "Prestamos activos:");
31        for (prestamo p : prestamos) {
32            if (p.fechaDevolución == null){
33                System.out.println(p);
34            }
35        }
36    }
37 }
38

```

Figura 13

```

1 public abstract class equipo {
2     protected int idEquipo;
3     protected String tipo;
4     protected boolean disponible;
5
6     public equipo(int idEquipo, String tipo) {
7         this.idEquipo = idEquipo;
8         this.tipo = tipo;
9         this.disponible = true;
10    }
11
12    public int getIdEquipo(){
13        return idEquipo;
14    }
15
16    public String getTipo(){
17        return tipo;
18    }
19
20    public boolean isDisponible() {
21        return disponible;
22    }
23
24    public void prestar() {
25        disponible = false;
26    }
27
28    public void devolver() {
29        disponible = true;
30    }
31
32    @Override
33    public String toString() {
34        return tipo + "ID:" + idEquipo + ", Disponible:" + disponible;
35    }
36 }

```

Figura 14

```

1  public class tablet extends equipo{
2      public tablet(int idEquipo) {
3          super(idEquipo, tipo: "Tablet");
4      }
5  }
6

```

Figura 15

```

1  public class portatil extends equipo {
2      public portatil(int idEquipo) {
3          super(idEquipo, tipo: "Portátil");
4      }
5  }
6
7

```

Figura 16

```

1 import java.util.ArrayList;
2 import java.util.List;
3 import java.util.Scanner;
4 public class main {
    Run | Debug | Run main | Debug main
5     public static void main(String[] args) {      This method has a constructor name
6         Scanner sc = new Scanner (System.in);      Convert to try-with-resources
7         gestorUsuario gestorUsuarios = new gestorUsuario();
8         gestorPrestamos gestorPrestamos = new gestorPrestamos();
9         gestorReservas gestorReservas = new gestorReservas();
10
11         aula aula1 = new aula(nombre: "Aula 1", cantidadPuestos: 10);
12
13         List<equipo> equipos = new ArrayList<>();
14         equipos.add(new portatil(idEquipo: 1));
15         equipos.add(new portatil(idEquipo: 2));
16         equipos.add(new tablet(idEquipo: 1));
17         equipos.add(new tablet(idEquipo: 2));
18         int opcion;
19         do {
20             System.out.println(x: "1. Registrar alumno");
21             System.out.println(x: "2. Registrar profesor");
22             System.out.println(x: "3. Listar usuarios");
23             System.out.println(x: "4. Registrar préstamo");
24             System.out.println(x: "5. Registrar devolución");
25             System.out.println(x: "6. Listar préstamos activos");
26             System.out.println(x: "7. Reservar puesto");
27             System.out.println(x: "8. Liberar puesto");
28             System.out.println(x: "9. Mostrar estado del aula");
29             System.out.println(x: "0. Salir");
30             System.out.print(s: "Seleccione una opción: ");
31             opcion = sc.nextInt();
32             sc.nextLine();

```

Figura 17

```
switch (opcion) {    Convert switch to rule switch

    case 1:
        System.out.print(s: "DNI: ");
        String dniAlumno = sc.nextLine();
        System.out.print(s: "Nombre: ");
        String nombreAlumno = sc.nextLine();
        System.out.print(s: "Apellidos: ");
        String apellidosAlumno = sc.nextLine();
        System.out.print(s: "NIA: ");
        String nia = sc.nextLine();
        gestorUsuarios.registrarAlumno(dniAlumno, nombreAlumno, apellidosAlumno, nia);
        break;

    case 2:
        System.out.print(s: "DNI: ");
        String dniProfesor = sc.nextLine();
        System.out.print(s: "Nombre: ");
        String nombreProfesor = sc.nextLine();
        System.out.print(s: "Apellidos: ");
        String apellidosProfesor = sc.nextLine();
        System.out.print(s: "ID Profesor: ");
        int idProfesor = sc.nextInt();
        gestorUsuarios.registrarProfesor(dniProfesor, nombreProfesor, apellidosProfesor, idProfesor);
        break;

    case 3:
        gestorUsuarios.listarUsuarios();
        break;

    case 4:
        System.out.print(s: "Id usuario:");
        int idUsuario = sc.nextInt();
        usuario u = gestorUsuarios.buscarUsuario(idUsuario);
        if (u != null) {
            System.out.print(s: "id equipo:");
            int idPuesto = sc.nextInt();
            sc.nextLine();
            System.out.print(s: "Franja horaria:");
            String franja = sc.nextLine();
            gestorReservas.reservar(u, aula1, idPuesto, franja);
        }
}
```

Figura 18

```

} else{
    System.out.println(x: "Usuario no encontrado");
}
break;

case 5:
    System.out.print(s: "id del puesto a liberar:");
    int idP = sc.nextInt();
    gestorReservas.librearPuesto(aula1, idP);
    break;

case 6:
    gestorReservas.mostrarEstadoAula(aula1);
    break;

case 7:
    System.out.print(s: "id usuario:");
    idUsuario = sc.nextInt();
    u = gestorUsuarios.buscarUsuario(idUsuario);
    if (u == null) {
        System.out.println(x: "usuario no encontrado");
        break;
    }

    System.out.println(x: "equipos disponibles:");
    for (equipo e : equipos){
        if(e.isDisponible()){
            System.out.println(e.getIdEquipo() + " - " + e.getTipo());
        }
    }

    System.out.print(s: "id equipo:");
    int idEquipo = sc.nextInt();

    equipo equipoSeleccionado = null;
    for (equipo e : equipos){
        if(e.getIdEquipo() == idEquipo) {
            equipoSeleccionado = e;
        }
    }
    if (equipoSeleccionado != null) {
        gestorPrestamos.registrarPrestamo(u, equipoSeleccionado);
    }
}

```

Figura 19

```

} else {
    System.out.println(x: "equipo no encontrado");
}
break;

case 8:
    System.out.print(s: " Id equipo a devolver:");
    idEquipo = sc.nextInt();

    for (equipo e : equipos) {
        if (e.getIdEquipo() == idEquipo) {
            gestorPrestamos.registrarDevolucion(e);
        }
    }
    break;

case 9:
    gestorPrestamos.listarPrestamosActivos();
    break;

case 0:
    System.out.println(x: "saliendo...");
    break;

default:
    System.out.println(x: "opcion no valida");
}

} while (opcion != 0);
sc.close();
}
}

```

Figura 20

```

1 import java.io.File;
2 import javax.xml.parsers.DocumentBuilder;
3 import javax.xml.parsers.DocumentBuilderFactory;
4 import javax.xml.transform.*;
5 import javax.xml.transform.dom.DOMSource;
6 import javax.xml.transform.stream.StreamResult;
7 import org.w3c.dom.*;
8
9 public class XMLUsuarios {
10
11     private static final String FICHERO = "usuarios.xml";
12
13     public static void guardarUsuario(usuario u) {
14
15         try {
16
17             DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
18             DocumentBuilder builder = factory.newDocumentBuilder();
19             Document doc;
20
21             File file = new File(FICHERO);
22
23             if (file.exists()) {
24                 doc = builder.parse(file);
25                 doc.getDocumentElement().normalize();
26             } else {
27                 doc = builder.newDocument();
28                 Element root = doc.createElement(tagName: "usuarios");
29                 doc.appendChild(root);
30             }
31
32             Element root = doc.getDocumentElement();
33
34             Element usuarioXML = doc.createElement(tagName: "usuario");
35
36             Element id = doc.createElement(tagName: "id");
37             id.setTextContent(String.valueOf(u.getIdUsuario()));
38             usuarioXML.appendChild(id);
39
40             Element tipo = doc.createElement(tagName: "tipo");
41             tipo.setTextContent(u.getClass().getSimpleName());
42             usuarioXML.appendChild(tipo);
43
44         }
45     }
46
47 }

```

Figura 21

```

Element nombre = doc.createElement(tagName: "nombre");
nombre.setTextContent(u.getNombre());
usuarioXML.appendChild(nombre);

Element apellidos = doc.createElement(tagName: "apellidos");
apellidos.setTextContent(u.getApellidos());
usuarioXML.appendChild(apellidos);

Element dni = doc.createElement(tagName: "dni");
dni.setTextContent(u.getDni());
usuarioXML.appendChild(dni);

Element activo = doc.createElement(tagName: "activo");
activo.setTextContent(String.valueOf(u.isActivo()));
usuarioXML.appendChild(activo);

root.appendChild(usuarioXML);

Transformer transformer = TransformerFactory.newInstance().newTransformer();
transformer.setOutputProperty(OutputKeys.INDENT, value: "yes");
transformer.transform(new DOMSource(doc), new StreamResult(file));

System.out.println(x: "Usuario guardado en XML correctamente.");
}

} catch (Exception e) { Can be replaced with multicatch or several catch clauses catch
System.out.println(x: "ERROR XML:");
e.printStackTrace(); Print Stack Trace
}
}
}

```

Figura 22

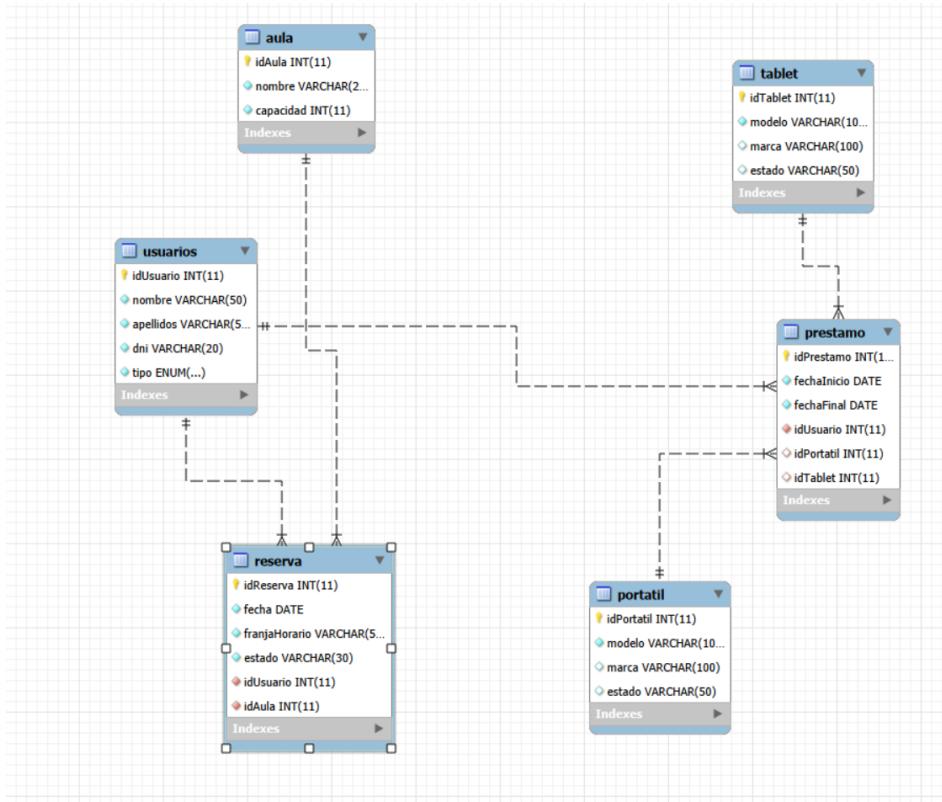


Figura 23

Tables_in_reservaDeAulas	
▶	aula
▶	portatil
▶	prestamo
▶	reserva
▶	tablet
▶	usuarios

Figura 24

The screenshot shows a terminal window titled 'sergio@debian: ~'. The window contains the following text:

```

sergio@debian:~$ sudo mariadb
[sudo] contraseña para sergio:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 31
Server version: 11.8.3-MariaDB-0+deb13u1 from Debian -- Please help get to 10k stars at https://github.com/MariaDB/Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> USE reservaDeAulas
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [reservaDeAulas]> █

```

Figura 25

```

Database changed
MariaDB [reservaDeAulas]> SHOW TABLES;
+-----+
| Tables_in_reservaDeAulas |
+-----+
| aula
| portatil
| prestamo
| reserva
| tablet
| usuarios
+-----+
6 rows in set (0,001 sec)

MariaDB [reservaDeAulas]> █

```

Figura 26

```
1 import java.util.ArrayList;
2 import java.util.List;
3 import java.util.Scanner;
4 public class main {
5     Run | Debug | Run main | Debug main
6     public static void main(String[] args) {    This method has a constructor name
7         Scanner sc = new Scanner (System.in);    Convert to try-with-resources
8         gestorUsuario gestorUsuarios = new gestorUsuario();
9         gestorPrestamos gestorPrestamos = new gestorPrestamos();
10        gestorReservas gestorReservas = new gestorReservas();
11
12        aula aula1 = new aula(nombre: "Aula 1", cantidadPuestos: 10);
13
14        List<equipo> equipos = new ArrayList<>();
15        equipos.add(new portatil(idEquipo: 1));
16        equipos.add(new portatil(idEquipo: 2));
17        equipos.add(new tablet(idEquipo: 1));
18        equipos.add(new tablet(idEquipo: 2));
19        int opcion;
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
311
312
313
313
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
```

Figura 27

