Solución Taller Java La Pingüinera

Por: Jesús David Bonelo Cuellar

El Taller consiste en realizar un programa de una biblioteca la cual realiza préstamos de libros y novelas, de los cuales tiene un inventario. El programa debe ser construido usando Java, su interfaz debe ser de consola y debe persistir la información en una base de datos.

Identificar las entidades y sus atributos

- Usuario
    - Nombre
    - Email
    - Contraseña
    - Rol
- Libro
    - *Titulo*
    - *Autor*
    - *Copias*
    - *Copias prestadas*
    - *Copias disponibles (derivado)*
    - Área del conocimiento
    - Número de páginas
- Novela
    - *Titulo*
    - *Autor*
    - *Copias*
    - *Copias prestadas*
    - *Copias disponibles (derivado)*
    - Genero

    - Edad sugerida
- Prestamo
    - Usuario prestador
    - Fecha solicitud
    - Fecha de entrega
    - Estado

Al identificar las entidades se evidencia que `Libro` y `Novela` comparten un set de atributos, lo cual permite generalizar creando una nueva entidad `ItemBiblioteca` con disyunción total. [#Modelos en Java]

Se analizan los atributos

- A todas las entidades se les agregará un atributo clave `id`
- Los posibles roles de los usuarios son:
    - Lector
    - Empleado (en lugar de Asistente para mayor claridad)
    - Administrador

      Nota: los usuarios no registrados también podrán usar limitadas funciones del programa
- Las copias son las unidades que le pertenecen a la biblioteca
- Las copias prestadas son las unidades que se encuentran prestadas a los usuarios
- La fecha de solicitud del préstamo se toma en el momento que el usuario confirma los detalles de su pedido en la aplicación
- La fecha de entrega, es la entrega que el usuario indicó que regresará los elementos prestados al momento de realizar su pedido
- Los posibles estados de un préstamo son: (renombrados para mayor claridad)
    - Solicitado: cuando el usuario solicita el pedido en la aplicación
    - Prestado: cuando el empleado realiza la confirmación y entrega del préstamo
    - Finalizado: cuando el empleado confirma que el usuario devolvió los ítems

Se analizan las reglas del negocio

- Las copias prestadas no deberían superar las que le pertenecen a la biblioteca, el sistema lo notificará, pero lo permite para flexibilidad del trabajo de los encargados del inventario.
- Los lectores deben ver únicamente los ítems que tienen copias disponibles para prestar.
- La fecha de entrega tiene un límite de 15 días desde el solicitado. El sistema notifica y no permite realizar la solicitud si se excede este límite. (se incluye unit test)
- Al momento de la devolución si la fecha se excedió la establecida el sistema lo notificará, las acciones a tomar quedan a cargo del empleado.
- Las acciones de borrado en la aplicación deben ser *soft delete*, es decir, la información se marca como borrada y no se utiliza en la aplicación, sin embargo, continúa en la base de datos evitando pérdida de información accidental. Para facilitar esto se define que los identificadores serán generados en la base de datos de forma automática.

Según el modelo de negocio, interpreto que los usuarios tienen permisos incrementales, es decir los roles más altos heredan todos los permisos de los inferiores, ej: el rol administrador tiene definidos sus permisos específicos y además al ser el rol más alto tiene todos demás permisos.

Se identifican las acciones que tiene permiso un usuario según su Rol

Usuario (sin iniciar sesión)

- Registrar cuenta de usuario como lector
- Iniciar sesión
- Acceder a la listas de libros y novelas
- Listar autores de libros o novelas
- Buscar libros o novelas por autor

Lector (usuario logueado)

Todas las acciones de usuario más:

- Agregar libros y/o novelas a préstamo
- Solicitar prestamo
- ~~Devolver los ítems prestados~~ *esto lo confirma el empleado*

Empleado

Todas las acciones de lector más:

- CRUD libros y novelas
- Listar prestamos
- Buscar prestamo por correo de usuario
- Realizar préstamo
- Finalizar préstamo

Administrador

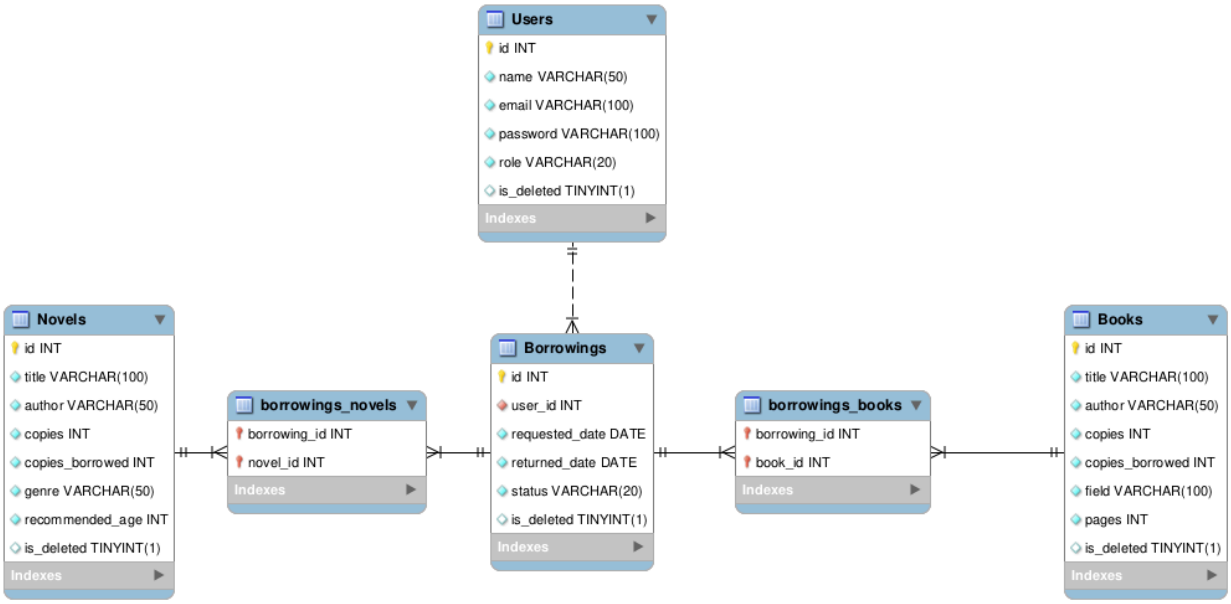
Todas las acciones de empleado más:

- Registrar cuentas de empleados
- CRUD de usuarios

Diseño de la base de datos

- Se define relación uno a muchos entre `Users` y `Borrowings`, con participación total (NOT NULL) de parte de el préstamo
- Relación muchos a muchos entre `Borrowings` y `Books` con participación parcial de ambos lados, es decir que los préstamos pueden no contener libros y los libros pueden no haber sido prestados.
- Relación muchos a muchos entre `Borrowings` y `Novels` con participación parcial de ambos lados, es decir que los préstamos pueden no contener novelas y las novelas pueden no haber sido prestadas.
- Al diagramar el modelo relacional, la disyunción al ser total indica que se crearán las tablas `Books` y `Novels` por separado con todos los atributos de `LibraryItem.` La herencia se aprovechará desde los modelos en el código java más adelante.

Diagrama EER



- El campo `is_deleted` es de tipo booleano y se utiliza como bandera para la implementación del borrado lógico. 1 es TRUE y 0 FALSE

Definición de la base de datos en sql

(archivo de script sql `pingu_schema.sql` adjunto en el repositorio)

```
DROP DATABASE IF EXISTS pingu;
CREATE DATABASE IF NOT EXISTS pingu;
USE pingu;

CREATE TABLE IF NOT EXISTS Users (
      id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
      name VARCHAR(50) NOT NULL,
      email VARCHAR(100) NOT NULL,
      password VARCHAR(100) NOT NULL,
      role VARCHAR(20) NOT NULL DEFAULT 'READER',
      is_deleted BOOLEAN DEFAULT 0
)  ENGINE=INNODB;

-- default admin user
INSERT INTO Users VALUES (1, "John Doe", "administrador@pingu.com.co",
"contraseniasegura123456", "ADMINISTRATOR", 0);
```

```sql
CREATE TABLE IF NOT EXISTS Books (
      id INT PRIMARY KEY AUTO_INCREMENT,
      title VARCHAR(100) NOT NULL,
      author VARCHAR(50) NOT NULL,
      copies INT NOT NULL,
      copies_borrowed INT NOT NULL,

      field VARCHAR(100) NOT NULL,
      pages INT NOT NULL,
      is_deleted BOOLEAN DEFAULT 0
)  ENGINE=INNODB;

CREATE TABLE IF NOT EXISTS Novels (
      id INT PRIMARY KEY AUTO_INCREMENT,
      title VARCHAR(100) NOT NULL,
      author VARCHAR(50) NOT NULL,
      copies INT NOT NULL,
      copies_borrowed INT NOT NULL,

      genre VARCHAR(50) NOT NULL,
      recommended_age INT NOT NULL,
      is_deleted BOOLEAN DEFAULT 0
)  ENGINE=INNODB;

CREATE TABLE IF NOT EXISTS Borrowings (
      id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
      user_id INT NOT NULL,
      requested_date DATE NOT NULL,
      returned_date DATE NOT NULL,
      status VARCHAR(20) NOT NULL DEFAULT 'REQUESTED',
      is_deleted BOOLEAN DEFAULT 0,
      FOREIGN KEY (user_id)
      REFERENCES Users (id)
)  ENGINE=INNODB;

CREATE TABLE IF NOT EXISTS borrowings_books (
      borrowing_id INT NOT NULL,
      book_id INT NOT NULL,
      PRIMARY KEY (borrowing_id , book_id),
      FOREIGN KEY (borrowing_id)
      REFERENCES Borrowings (id),
      FOREIGN KEY (book_id)
```

```
      REFERENCES Books (id)
)  ENGINE=INNODB;

CREATE TABLE IF NOT EXISTS borrowings_novels (
      borrowing_id INT NOT NULL,
      novel_id INT NOT NULL,
      PRIMARY KEY (borrowing_id , novel_id),
      FOREIGN KEY (borrowing_id)
      REFERENCES Borrowings (id),
      FOREIGN KEY (novel_id)
      REFERENCES Novels (id)
)  ENGINE=INNODB;
```

Se aprovecha el script de definición de la base de datos para insertar el usuario administrador predeterminado.

Como datos de prueba para la base de datos, se adjunta en el repositorio el archivo `pingu_populate.sql` para realizar inserciones en las tablas.
Los datos de prueba están preparados con los siguientes casos para demostración:

- Libro sin unidades disponibles
- Préstamo por entregar con fecha vencida.

Código del programa en Java

Para desarrollar el programa decidí separarlo en capas de modelos, persistencia, servicios y ui, cada una encargada de sus funciones específicas.

Dependencias utilizadas:

- Conector de mysql para usar con JDBC
- Librería para leer variables privadas desde un archivo .env

```
dependencies {
      implementation "mysql:mysql-connector-java:8.0.33"
      implementation 'io.github.cdimascio:dotenv-java:3.0.0'
}
```

En la raíz del paquete principal se encuentran los archivos:

- Secrets para guardar variables constantes.

```
public class Secrets {
	public static final String DB_URL =
"jdbc:mysql://localhost:3307/pingu";
	public static final String DB_USER = "root", DB_PASSWORD =
"root1234";
}
```

(Por motivos de seguridad lo cambié por la implementación de un .env en éste commit, añadiendo explicación de su uso en el README.md del código)

- Conexión a la base de datos: contiene métodos estáticos para obtener y cerrar correctamente la conexión, evitando likear conexiones gracias a que se reutiliza una sola hasta que esta se cierre.

```
public class PinguDatabase {
	private static Connection connection;

	private static void startConnection() {
	    try {
	      connection = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASSWORD);
	    } catch (Exception e) {
	      System.out.println("Database connection error: " +
e.getMessage());
	    }
	}

	public static Connection getConnection() {
	    if (connection == null) {
		startConnection();
	    }
	    return connection;
	}

	public static void close() {
	    try {
		connection.close();
		connection = null;
	    } catch (SQLException e) {
		e.printStackTrace();
	    }
	}
}
```

- Archivo principal: aquí inicia la ejecución del programa, configura las variables de entorno, obtiene la conexión a la base de datos, inicia el menú pasandole la conexión y al salir del menú se cierra la conexión.

```
public class Main {
      public static void main(String[] args) {
       // setup variables from the .env file to System properties
        Environment.setup();

        Connection connection = PinguDatabase.getConnection();

        try {
            new MainMenu(connection).menu();
        } catch (Exception e) {
            logger.severe("General error, please contact the developer with
the following error " + "info: " + e.getMessage());
        }

        PinguDatabase.close();
      }
}
```

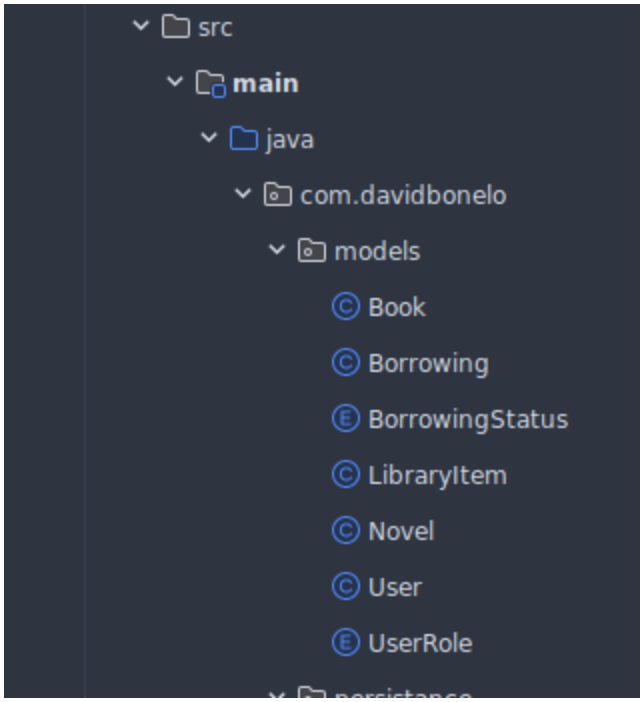
Se agregó el uso de Logger en este punto para tener un registro más específico de los errores que posiblemente se se escapen por no haber sido manejados en los puntos previos.

Paquete de utilidades: este contiene clases con métodos estáticos los cuales reutilizo en diferentes partes de la aplicación.

- Manejo de las interacciones del usuario leyendo desde la entrada estándar
- Validación de permisos de roles.
  (explicados más adelante en la sección de los menús)

Modelos

En esta capa se definen las clases base cada una con sus propiedades y métodos específicos.



Utilicé una clase abstracta `LibraryItem` como base, de la cual heredan sus propiedades y métodos las clases `Book` y `Novel`

```
public abstract class LibraryItem {
      private int id;
      private String title;
      private String author;
      private int copies;
      private int copiesBorrowed;
// ...
```

Utilizo los modificadores de acceso para realizar la debida encapsulación de las propiedades evitando que estén visibles en otras partes del código donde no deberían.

```
    public LibraryItem(String title, String author, int copies, int copiesBorrowed) {...}

    public LibraryItem(int id, String title, String author, int copies, int copiesBorrowed) {...}
```

Defino múltiples constructores, permitiendo el polimorfismo al crear instancias de Items sin id cuando se crea por input del usuario, e instancias con Id para su use general en la aplicación cuando se crean al leer desde la base de datos.

Creo getters y setters que permitan interactuar con las propiedades de las clases (se omiten por brevedad)

Creo un getter para el atributo derivado de las copias disponibles

```
public int getAvailableCopies() {
	return copies - copiesBorrowed;
}
```

Y un método privado para verificar la integridad lógica de la información sobre las copias, que según la regla de negocio muestra solo una advertencia al empleado que tomará la decisión si editarlo.

```
private void checkCopies() {
	if (copiesBorrowed > copies) {
		// Allow but show a warning
		System.out.println("Warning: more copies borrowed than copies
in existence");
	}
}
```

Este método lo utilizo en las demás partes de la clase que modifican la información sobre las copias.

```
public void setCopies(int copies) {
	this.copies = copies;
	checkCopies();
}
// ...
public void setCopiesBorrowed(int copiesBorrowed) {
	this.copiesBorrowed = copiesBorrowed;
	checkCopies();
}
```

Sobreescribo el toString

```
@Override
public String toString() {
	return "id=" + id + ", title='" + title + '\'' + ", author='" +
author + '\'' + ", copies" +
			"=" + copies + ", copiesBorrowed=" + copiesBorrowed;
}
```

De esta clase abstracta **heredan** los modelos de libro y novela cada uno definiendo sus propiedades específicas, reutilizan y sobreescriben los constructores y métodos de la clase padre.

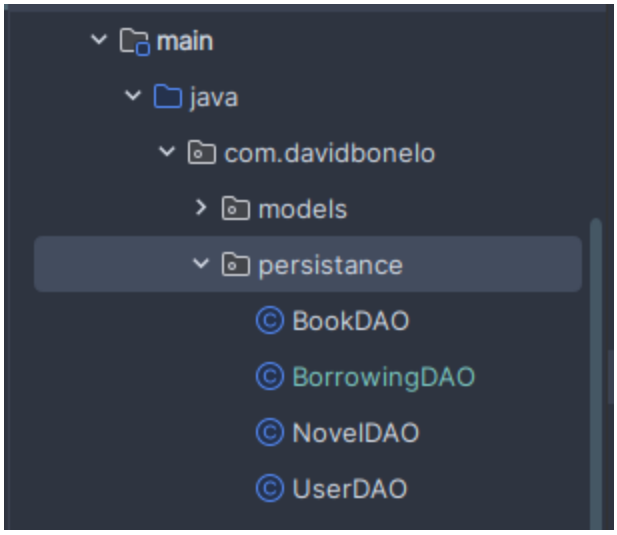
En esta capa también se encuentran los enumerables para los roles de usuario y estados de los préstamos, como no son muchos no considero necesario anidarlos dentro de otra carpeta “enums” en esta capa.

```
public enum UserRole {
	ADMINISTRATOR("ADMINISTRATOR"), EMPLOYEE("EMPLOYEE"),
READER("READER");

	private final String value;

	UserRole(String value) {
	this.value = value;
	}

	public String getValue() {
	return value;
	}
}
```

> Este procedimiento lo repito creando cada una de las clases en la capa de modelos

Persistencia (o repositorios)

En esta capa se encuentran las clases DAO (Data Access Object) que se encargan de la comunicación y las interacciones con la base de datos.



También en esta capa se encuentra una clase utilitaria para la lectura y escritura de archivos csv, lo cual considero como parte de la persistencia de datos.

Cada una de las clases DAO recibe en el constructor un objeto conexión para poder interactuar con la base de datos, y lo utiliza en en los métodos que definí para realizar todas las operaciones CRUD en cada una de las tablas.

En la construcción de las queries que requieren argumentos, utilizo `PreparedStatement` de jdbc, el cual trae métodos que permiten establecerle valores de forma segura, ya que estos metodos `.setXXXXX()` verifican que el dato que se está ingresando en la query no contenga caracteres reservados de sql, en cuyo caso los escaparía con `\` . Este método es más seguro que utilizar realizar concatenaciones o `String.format()` para construir las queries o serían vulnerables a inyección de sql.

```
public void createUser(User user, String password) throws SQLException {
      String sql = "INSERT INTO Users (name, email, password, role) VALUES
(?, ?, ?, ?)";
      try (PreparedStatement statement = connection.prepareStatement(sql))
{
            statement.setString(1, user.getName());
            statement.setString(2, user.getEmail());
            statement.setString(3, password);
            statement.setString(4, user.getRole().getValue());

            int rowsInserted = statement.executeUpdate();
            if (rowsInserted == 0) {
                  throw new SQLException("Creating user failed, no rows
affected");
```

```
            }
      }
}
```

Como se puede observar en los modelos, para el usuario no incluye la contraseña como medida preventiva de seguridad ya que se guarda como texto plano, En su lugar los métodos para crear y validar un usuario en la base de datos reciben como parámetro la contraseña.

```
public static User buildUserFromResult(ResultSet rs) throws SQLException {
      UserRole role = UserRole.valueOf(rs.getString("role"));
      return new User(rs.getInt("id"), rs.getString("name"),
rs.getString("email"), role);
}
```

Al programar los métodos de lectura de la base de datos me di cuenta que estaba repitiendo la lógica de construir los objetos modelo, entonces decidí extraerla en métodos reutilizables, los cuales fueron muy útiles puesto que los terminé utilizando varias veces.

Borrado lógico

Para implementar el borrado lógico, lo realicé directamente en las queries, debido a que haberlo hecho con streams requeriría repetir mucho código.

```
public void softDeleteUser(int userId) throws SQLException {
      String sql = "UPDATE Users SET is_deleted=1 WHERE id = ?";
      try (PreparedStatement statement = connection.prepareStatement(sql))
{
            statement.setInt(1, userId);
            int deletedRows = statement.executeUpdate();
            if (deletedRows == 0) {
                  throw new SQLException("User with id " + userId + " not
found, can't delete");
            }
      }
}
```

```
public List<User> getAllUsers() throws SQLException {
      // Avoid selecting passwords
      String sql = "SELECT id, name, email, role FROM Users WHERE
```

```
is_deleted = 0";
      ArrayList<User> users = new ArrayList<>();
      try (PreparedStatement statement = connection.prepareStatement(sql);
ResultSet rs = statement.executeQuery()) {

            while (rs.next()) {
                  users.add(buildUserFromResult(rs));
            }
            return users;
      }
}
```

En los DAOs de los libros y novelas separé el reemplazo de los parametros de la query en un método aparte, para de esta forma permitir la implementación de inserciones optimizadas en un batch que utilizo en el servicio de importación desde csv.

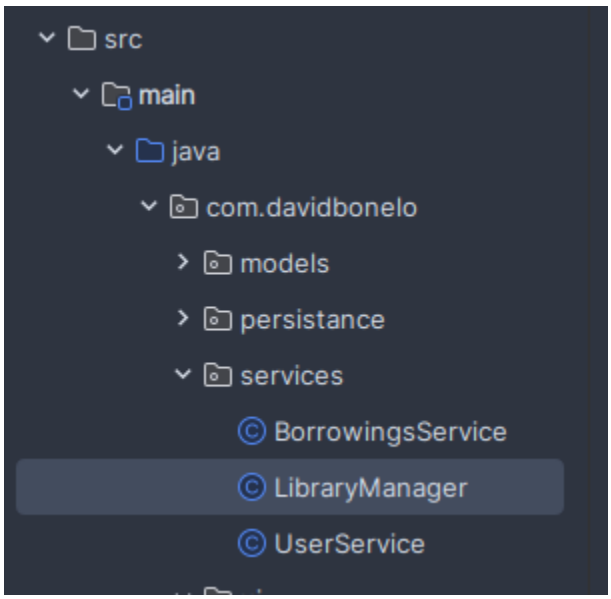
```
public void createBook(Book book) throws SQLException {
      String sql = "INSERT INTO Books ( id, title, author, copies,
copies_borrowed, field, " +
                  "pages ) VALUES ( ?, ?, ?, ?, ?, ?, ? )";
      try (PreparedStatement statement = connection.prepareStatement(sql))
{
            prepareCreate(statement, book);
            int rowsInserted = statement.executeUpdate();
            if (rowsInserted == 0) {
                  throw new SQLException("Creating book failed, no rows
affected.");
            }
      }
}

public void prepareCreate(PreparedStatement statement, Book book) throws
SQLException {
      if (containsId(book)) {
            statement.setInt(1, book.getId());
      } else {
            statement.setNull(1, MysqlType.NULL.getJdbcType());
      }
      statement.setString(2, book.getTitle());
      statement.setString(3, book.getAuthor());
      statement.setInt(4, book.getCopies());
```

```
        statement.setInt(5, book.getCopiesBorrowed());
        statement.setString(6, book.getField());
        statement.setInt(7, book.getPages());
}
```

(Al enviar un NULL en el id la base de datos está configurada para generar uno de forma automática)

Servicios

En esta capa se encuentran las clases encargadas de la lógica de negocio, haciendo los procedimientos y las llamadas necesarias a la base de datos.



Para la creación de instancias de los servicios se reciben en el constructor los DAOs que utilizan y la conexión si el servicio implementa transacciones a la base de datos.

Por ejemplo al crear la solicitud de un préstamo, se debe verificar la disponibilidad de los items, crearse el préstamo el cual requiere llenar varias tablas debido a la relación muchos a muchos, y por último actualizar el inventario de cada uno de los ítems prestados. Si alguna de estas acciones falla, se cancela la transacción y se devuelven las acciones realizadas a la base de datos hasta el punto inicial.

```
public void createBorrowing(Borrowing borrowing) throws SQLException {
      try {
            connection.setAutoCommit(false); // start transaction
            verifyItemsAvailability(itemsToBorrow);
```

```
            borrowing.setBorrowedItems(itemsToBorrow.stream().toList());

            borrowingDAO.createBorrowing(borrowing);
            updateItemsBorrowedCopies(borrowing.getBorrowedItems(), 1);

            connection.commit(); // commit transaction
            itemsToBorrow.clear(); // Clean this class set

      } catch (SQLException e) {
            connection.rollback();
            throw e;
      } finally {
            connection.setAutoCommit(true);
      }
}
```

La lógica de este método la separé en diferentes métodos que tienen una responsabilidad única.

```
/**
* @param factor the amount of copies to sum (or rest if negative)
*/
private void updateItemsBorrowedCopies(List<LibraryItem> items, int factor)
throws SQLException {
      for (LibraryItem li : items) {
            li.setCopiesBorrowed(li.getCopiesBorrowed() + factor);
            if (li instanceof Book book) {
                  bookDAO.updateBook(book);
            } else if (li instanceof Novel novel) {
                  novelDAO.updateNovel(novel);
            }
      }
}
```

Este método inicialmente lo tenía duplicado para sumar y restar del inventario, pero lo adapté para que sea reusable para aumentar o restar inventario de los ítems.

Las acciones de filtrado las implementé directamente en esta capa utilizando streams y collectors en lugar de definir métodos con queries sql en la capa de persistencia, esto debido a que el propósito del taller es de practicar java.

```
public List<Borrowing> getBorrowingsByEmail(User employee, String email) {
	return getAllBorrowings(employee).stream().filter(b ->
b.getBorrower().getEmail().equals(email)).toList();
}
```

```
public List<Borrowing> getAllBorrowings(User user) {
	try {
		List<Borrowing> borrowings = borrowingDAO.getAllBorrowings();
		// Show all borrowings only if authorized
		if (validPermission(user, UserRole.EMPLOYEE)) {
			return borrowings;
		} else {
			// Filter only the ones where user is owner
			return borrowings.stream().filter(b ->
b.getBorrower().getId() == user.getId()).toList();
		}
	} catch (SQLException e) {
		e.printStackTrace();
	}
	return Collections.emptyList();
}
```

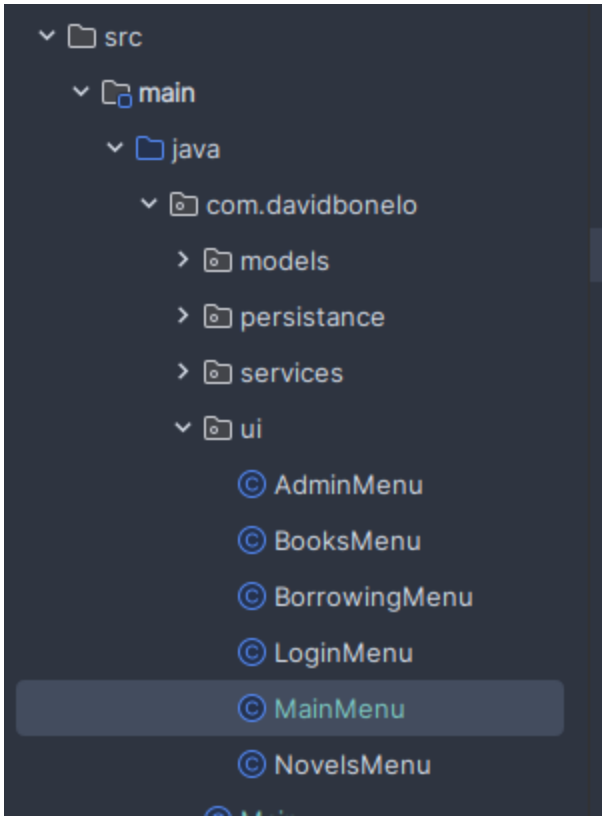
En este método implemento el listado de todos los préstamos, verificando si el usuario tiene el rol de empleado quien podrá ver todos los préstamos, y si es usuario filtra mostrando solo los que le pertenecen a él.

```
public Set<String> getAuthorsList(List<? extends LibraryItem> items) {
	return
items.stream().map(LibraryItem::getAuthor).collect(Collectors.toSet());
}
```

Este método aprovecha el polimorfismo de los libros y novelas como ítems ya que solo trabaja con el valor del autor que heredan de la clase padre, por lo tanto sirve para generar la lista de autores de libros o novelas.

UI

Esta es la capa final que se encarga de las interacciones con el usuario a través de menús dinámicos, mostrando opciones al usuario por la salida de consola estándar y leyendo las opciones del usuario por la entrada estándar.



Este fue una de las partes más retantes de este taller debido a que decidí implementar un menú dinámico de forma que muestre las opciones disponibles basándose en el estado de inicio de sesión y el rol del usuario.

Para resolverlo, en las utilidades programé varios métodos para manejar las interacciones con el usuario como `askNumber` `askText` y `askDate` los cuales imprimen un mensaje, verifican que la entrada sea correcta y en caso de no serlo, lo informa y pregunta nuevamente hasta tener un dato válido. Ej:

```
public static LocalDate askDate(String prompt) {
	ResourceBundle messages = ResourceBundle.getBundle("ask");
	prompt += " (YYYY-MM-DD)\n> ";
	System.out.print(prompt);
	String input = scanner.nextLine().trim();
	LocalDate date = null;
	// Invalid input handling
	while (input.isEmpty() || date == null) {
		try {
			date = LocalDate.parse(input);
		} catch (Exception ignored) {
```

```
System.out.println(messages.getString("ask.invalid.date"));
                    System.out.print(prompt);
                    input = scanner.nextLine().trim();
                }
            }
            return date;
}
```

Al construir la lógica de mostrar las opciones de forma dinámica en cada uno de los menús, vi la oportunidad de abstraerla en una clase aparte y reutilizarla

```
public class MenuChoices {
      private final String menuName;
      private final String visitorChoices;
      private final String readerChoices;
      private final String employeeChoices;
      private final String adminChoices;
// ...
      public int showMenu(User user) {
      String menuMessage = buildMenuMessage(user);
      return askNumber(menuMessage);
      }
// ver el resto de la lógica de construcción del mensaje en el archivo del
repo
}
```

Se construye con el nombre del menú, para mostrar el menú construye el mensaje con cada una de las opciones disponibles según el rol del usuario que se le pase.

En cada clase de menú defino las opciones por cada rol y se llama `MenuChoices` para mostrar el menú y recibir la entrada

```
private int showMenu() {
       String visitorChoices = messages.getString("main.choices.visitor");
       String readerChoices = messages.getString("main.choices.reader");
       String adminChoices = messages.getString("main.choices.admin");
       return new MenuChoices(messages.getString("main.keyword"),
visitorChoices, readerChoices, "", adminChoices).showMenu(user);
}
```

(Las strings están definidas en un archivo .properties aparte)

```
public void menu() {
	while (true) {
		int menuChoice = showMenu();
		switch (menuChoice) {
			case 1 -> listBooks();
			case 2 -> listAuthors();
			case 3 -> searchByAuthor();
			case 4 -> addToBorrowing();
			case 5 -> registerBook();
			case 6 -> updateBook();
			case 7 -> deleteBook();
			case 0 -> {
			return;
			}
			default ->
System.out.println(messages.getString("unknownOption"));
		}
	}
}
```

Cada clase de menú cuenta con un método para lanzar el menú en bucle infinito, realizar las acciones correspondientes según la selección del usuario y la opción para salir del menú.

En cada **case** se llama a un método en la clase del menú o si la opción es de navegación a un submenú crea la clase pasandole los servicios que requiera.

```
    case 4 -> new BorrowingMenu(borrowingsService, user).menu();
    case 5 -> new AdminMenu(userService, dataService, user).menu();
```

Los menús que requieren un mínimo rol para acceder a ellos tienen una validación apenas se intenta acceder al menú

```
// Menú de préstamos
public void menu() {
	while (true) {
		if (!validMenuAccess(user, UserRole.READER)) {
			return;
		}
		int menuChoice = showMenu();
		switch (menuChoice) {
// ...
```

```
// Menú de admin
public void menu() {
```

```
      while (true) {
            if (!validMenuAccess(user, UserRole.ADMINISTRATOR)) {
                  return;
            }

            int menuChoice = showMenu();
            switch (menuChoice) {
// ...
```

Cuando alguna de las opciones del menú son específicas en un rol superior al requerido para acceder al menú en el que se encuentra, se valida el permiso directamente en el método. Por ej: el menú de Libros no requiere rol para listar los libros, pero sí para borrarlos.

```
private void deleteBook() {
      if (!validMenuAccess(user, UserRole.EMPLOYEE)) {
            return;
      }
      int bookId = askNumber(messages.getString("books.req.deleteId"));
      libraryManager.deleteBook(bookId);
}
```

Las acciones que utilizan los servicios que pueden lanzar excepciones realizan el manejo informandole al usuario si la acción fue exitosa, o sino le muestra el error:

```
private void exportInventoryData() {
      System.out.println(messages.getString("admin.info.exporting"));
      String fileName = askText(messages.getString("admin.req.exportName"))
+ ".csv";
      try {
            dataService.exportInventory(fileName);
            System.out.println(messages.getString("admin.res.exportOk") +
fileName + ".csv");
      } catch (SQLException | IOException e) {
            System.out.println(messages.getString("main.res.exportBad") +
e.getMessage());
      }
}
```

Evidencias del funcionamiento

- Menú principal como usuario no logeado
- Inicio de sesión
- Menú principal como usuario admin
- Cierre de sesión
- Salida del programa

```
Welcome to La Pingüinera library!!
Main menu: 1. Login | 2. Books | 3. Novels | 0. Exit
> 1
Login menu: 1. Login with email | 2. Register | 0. Back
> 1
Email:
> administrador@pingu.com.co
Password:
> contraseniasegura123456
Successful login as: User{id=1, name='John Doe', email='administrador@pingu.com.co', role=ADMINISTRATOR}
Main menu: 2. Books | 3. Novels | 4. Borrowings | 5. Admin menu | 8. Idioma Español | 9. Log out | 0. Exit
> 9
Main menu: 1. Login | 2. Books | 3. Novels | 0. Exit
> 0

Process finished with exit code 0
```

Opción de mejora: ocultar los caracteres al ingresar la contraseña

- Menú principal como lector (igual para empleado)

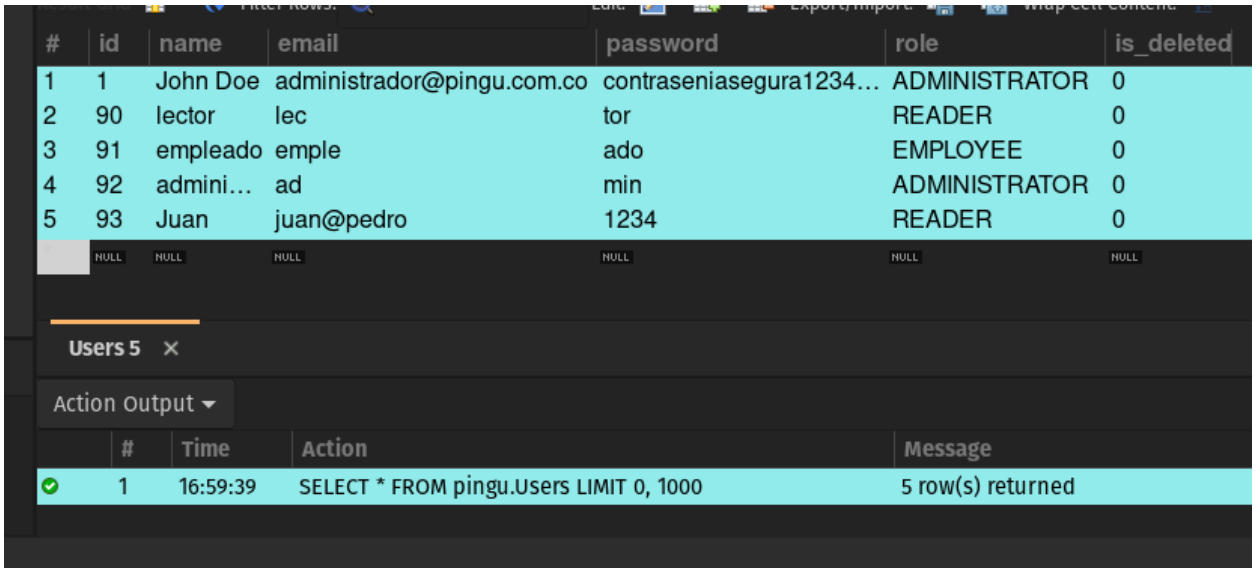
```
Successful login as: User{id=90, name='lector', email='lec', role=READER}
Main menu: 2. Books | 3. Novels | 4. Borrowings | 8. Idioma Español | 9. Log out | 0. Exit
>
```

- Registro de usuario (automáticamente asigna rol de lector)

```
Main menu: 1. Login | 2. Books | 3. Novels | 0. Exit
> 1
Login menu: 1. Login with email | 2. Register | 0. Back
> 2
Name:
> Juan
Email:
> juan@pedro
Password:
> 1234
Login menu: 1. Login with email | 2. Register | 0. Back
> 1
Email:
> juan@pedro
Password:
> 1234
Successful login as: User{id=93, name='Juan', email='juan@pedro', role=READER}
Main menu: 2. Books | 3. Novels | 4. Borrowings | 8. Idioma Español | 9. Log out | 0. Exit
>
```

Opción de mejora: validar formato de correo con una regex

- Usuario creado en la base de datos



#	id	name	email	password	role	is_deleted
1	1	John Doe	administrador@pingu.com.co	contraseniasegura1234…	ADMINISTRATOR	0
2	90	lector	lec	tor	READER	0
3	91	empleado	emple	ado	EMPLOYEE	0
4	92	admini…	ad	min	ADMINISTRATOR	0
5	93	Juan	juan@pedro	1234	READER	0
	NULL	NULL	NULL	NULL	NULL	NULL

Users 5 ×

Action Output

#	Time	Action	Message
1	16:59:39	SELECT * FROM pingu.Users LIMIT 0, 1000	5 row(s) returned

- Menú de libros como usuario no logueado

```
Main menu: 1. Login | 2. Books | 3. Novels | 0. Exit
> 2
Books menu: 1. List Books | 2. List authors | 3. Search by author | 0. Back
>
```

- Menú de libros como usuario lector (agrega opción de seleccionar libro para préstamo)

```
Successful login as: User{id=93, name='Juan', email='juan@pedro', role=READER}
Main menu: 2. Books | 3. Novels | 4. Borrowings | 8. Idioma Español | 9. Log out | 0. Exit
> 2
Books menu: 1. List Books | 2. List authors | 3. Search by author |4. Borrow a Book | 0. Back
>
```

- Menú de libros como usuario empleado (agrega opciones crud) - Igual que el de admin

```
Books menu: 1. List Books | 2. List authors | 3. Search by author |4. Borrow a Book |
5. Register Book | 6. Update Book | 7. Delete Book | 0. Back
>
```

- Listado de libros como visitante/lector (filtra los no disponibles)
- Listado de autores de libros
- Búsqueda de libros por autor

```
Main menu: 1. Login | 2. Books | 3. Novels | 0. Exit
> 2
Books menu: 1. List Books | 2. List authors | 3. Search by author | 0. Back
> 1
Book{id=1, title='Cien años de soledad', author='Gabriel García Márquez', copies=6, copiesBorrowed=1, field='Ficción', pages=432}
Book{id=3, title='Crónica de una muerte anunciada', author='Gabriel García Márquez', copies=2, copiesBorrowed=0, field='Ficción', pages=128}
Book{id=4, title='La sombra del viento', author='Carlos Ruiz Zafón', copies=4, copiesBorrowed=0, field='Ficción', pages=576}
Book{id=5, title='Cien años de soledad', author='Gabriel García Márquez', copies=10, copiesBorrowed=0, field='Ficción', pages=434}
Book{id=6, title='Rayuela', author='Julio Cortázar', copies=5, copiesBorrowed=2, field='Ficción', pages=700}
Book{id=7, title='asdf', author='asdff', copies=1, copiesBorrowed=0, field='asdfasd', pages=90}
Books menu: 1. List Books | 2. List authors | 3. Search by author | 0. Back
> 2
Carlos Ruiz Zafón
Gabriel García Márquez
Julio Cortázar
asdff
Books menu: 1. List Books | 2. List authors | 3. Search by author | 0. Back
> 3
Type the name of the author you want to search:
> Gabriel García Márquez
Book{id=1, title='Cien años de soledad', author='Gabriel García Márquez', copies=6, copiesBorrowed=1, field='Ficción', pages=432}
Book{id=3, title='Crónica de una muerte anunciada', author='Gabriel García Márquez', copies=2, copiesBorrowed=0, field='Ficción', pages=128}
Book{id=5, title='Cien años de soledad', author='Gabriel García Márquez', copies=10, copiesBorrowed=0, field='Ficción', pages=434}
Books menu: 1. List Books | 2. List authors | 3. Search by author | 0. Back
>
```

- Listado de libros como empleado/admin (puede ver libros no disponibles para préstamo)

```
Successful login as: User{id=91, name='empleado', email='emple', role=EMPLOYEE}
Main menu: 2. Books | 3. Novels | 4. Borrowings | 8. Idioma Español | 9. Log out | 0. Exit
> 2
Books menu: 1. List Books | 2. List authors | 3. Search by author |4. Borrow a Book |
5. Register Book | 6. Update Book | 7. Delete Book | 0. Back
> 1
Book{id=1, title='Cien años de soledad', author='Gabriel García Márquez', copies=6, copiesBorrowed=1, field='Ficción', pages=432}
Book{id=2, title='El amor en los tiempos del cólera', author='Gabriel García Márquez', copies=1, copiesBorrowed=1, field='Ficción', pages=368}
Book{id=3, title='Crónica de una muerte anunciada', author='Gabriel García Márquez', copies=2, copiesBorrowed=0, field='Ficción', pages=128}
Book{id=4, title='La sombra del viento', author='Carlos Ruiz Zafón', copies=4, copiesBorrowed=0, field='Ficción', pages=576}
Book{id=5, title='Cien años de soledad', author='Gabriel García Márquez', copies=10, copiesBorrowed=0, field='Ficción', pages=434}
Book{id=6, title='Rayuela', author='Julio Cortázar', copies=5, copiesBorrowed=2, field='Ficción', pages=700}
Book{id=7, title='asdf', author='asdff', copies=1, copiesBorrowed=0, field='asdfasd', pages=90}
Books menu: 1. List Books | 2. List authors | 3. Search by author |4. Borrow a Book |
5. Register Book | 6. Update Book | 7. Delete Book | 0. Back
>
```

- Seleccionar **múltiples** libros para préstamo (informa que debe completar el pedido en el otro menú)
- Listar libros seleccionados para préstamo

```
Books menu: 1. List Books | 2. List authors | 3. Search by author |4. Borrow a Book | 0. Back
> 4
Type the id of the book you want to borrow:
> 3
Book added, go to the Borrowings menu to complete the request
Books menu: 1. List Books | 2. List authors | 3. Search by author |4. Borrow a Book | 0. Back
> 4
Type the id of the book you want to borrow:
> 5
Book added, go to the Borrowings menu to complete the request
Books menu: 1. List Books | 2. List authors | 3. Search by author |4. Borrow a Book | 0. Back
> 0
Main menu: 2. Books | 3. Novels | 4. Borrowings | 8. Idioma Español | 9. Log out | 0. Exit
> 4
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request | 0. Back
> 1
Selected items pending to confirm request: 3
Book{id=3, title='Crónica de una muerte anunciada', author='Gabriel García Márquez', copies=2, copiesBorrowed=0, field='Ficción', pages=128}
Book{id=6, title='Rayuela', author='Julio Cortázar', copies=5, copiesBorrowed=1, field='Ficción', pages=700}
Book{id=5, title='Cien años de soledad', author='Gabriel García Márquez', copies=10, copiesBorrowed=0, field='Ficción', pages=434}
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request | 0. Back
>
```

- Realizar solicitud de préstamo
- Manejo de fecha inválida
- Intento de hacer la solicitud con más de 15 días para entrega rechaza la solicitud

```
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request | 0. Back
> 4
Requesting selected items
Type the due date before all the items must be returned: (YYYY-MM-DD)
> no ingreso una fecha
Invalid input. Please enter a date in the format YYYY-MM-DD
Type the due date before all the items must be returned: (YYYY-MM-DD)
> 2024-12-12
Couldn't request the borrowing, returnDate must be less than 15 days since requestedDate
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request | 0. Back
> 4
Requesting selected items
Type the due date before all the items must be returned: (YYYY-MM-DD)
> 2024-04-25
Successful request, find an employee to borrow the items from
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request | 0. Back
>
```

- Pedido en la base de datos

#	id	user_id	requested_date	returned_date	status	is_deleted	name	email	role
1	1	2	2024-04-03	2024-04-17	BORROWED	0	Juan P…	juan@example.com	ADMINISTRATOR
2	2	3	2024-04-05	2024-04-19	REQUESTED	0	María …	maria@example.com	READER
3	3	4	2024-04-07	2024-04-21	REQUESTED	0	Pedro …	pedro@example.com	READER
4	4	5	2024-04-09	2024-04-23	FINALIZED	0	Luisa …	luisa@example.com	EMPLOYEE
5	5	6	2024-04-01	2024-04-11	BORROWED	0	Andrés …	andres@example.com	READER
6	6	90	2024-04-16	2024-04-25	REQUESTED	0	lector	lec	READER

Borrowings 1 × Result 7 ×

Action Output

#	Time	Action	Message
1	17:32:17	SELECT b.*, u.name, u.email, u.role FROM Borrowings b LEFT JOIN Users u ON b.user_id = u.id LIMIT 0, 1000	6 row(s) returned

- Se actualiza el inventario de los libros solicitados

id	title	author	copies	copies_borrowe	field	pages	is_deleted
1	Cien años de soledad	Gabriel García Márquez	6	1	Ficción	432	0
2	El amor en los tiempos del cólera	Gabriel García Márquez	1	1	Ficción	368	0
3	Crónica de una muerte anunciada	Gabriel García Márquez	2	1	Ficción	128	0
4	La sombra del viento	Carlos Ruiz Zafón	4	0	Ficción	576	0
5	Cien años de soledad	Gabriel García Márquez	10	1	Ficción	434	0
6	Rayuela	Julio Cortázar	5	2	Ficción	700	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Se limpia la lista de elementos seleccionados, dejándola preparada para una nueva solicitud

```
Selected items pending to confirm request: 0
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request | 0. Back
>
```

- Lector lista todos los pedidos (solo ve los suyos)

```
> 2024-04-25
Successful request, find an employee to borrow the items from
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request | 0. Back
> 2
Borrowings list:
Borrowing{ id=6, borrowerId=90, borrower=lector, requestedDate=2024-04-16, returnDate=2024-04-25, status=REQUESTED}
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request | 0. Back
```

- Mostrar detalles de pedido

```
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request | 0. Back
> 3
Type the id of the borrowing you want to see its details
> 6
Borrowing details:
Borrowing{ id=6, borrowerId=90, borrower=lector, requestedDate=2024-04-16, returnDate=2024-04-25, status=REQUESTED}
Items list:
Book{id=3, title='Crónica de una muerte anunciada', author='Gabriel García Márquez', copies=2, copiesBorrowed=1, field='Ficción', pages=128}
Book{id=5, title='Cien años de soledad', author='Gabriel García Márquez', copies=10, copiesBorrowed=1, field='Ficción', pages=434}
Book{id=6, title='Rayuela', author='Julio Cortázar', copies=5, copiesBorrowed=2, field='Ficción', pages=700}

Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request | 0. Back
>
```

- Mostrar detalles de pedido que no le pertenece al lector

```
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request | 0. Back
> 3
Type the id of the borrowing you want to see its details
> 5
Couldn't get borrowing details,After end of result set
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request | 0. Back
>
```

- Empleado lista todos los pedidos de todos los usuarios

```
Successful login as: User{id=91, name='empleado', email='emple', role=EMPLOYEE}
Main menu: 2. Books | 3. Novels | 4. Borrowings | 8. Idioma Español | 9. Log out | 0. Exit
> 4
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request |
5. Search by email | 6. Confirm borrowing | 7. Finalize borrowing | 8. Delete borrowing | 0. Back
> 2
Borrowings list:
Borrowing{ id=1, borrowerId=2, borrower=Juan Pérez, requestedDate=2024-04-03, returnDate=2024-04-17, status=BORROWED}
Borrowing{ id=2, borrowerId=3, borrower=María Gómez, requestedDate=2024-04-05, returnDate=2024-04-19, status=REQUESTED}
Borrowing{ id=3, borrowerId=4, borrower=Pedro Ramírez, requestedDate=2024-04-07, returnDate=2024-04-21, status=REQUESTED}
Borrowing{ id=4, borrowerId=5, borrower=Luisa Martínez, requestedDate=2024-04-09, returnDate=2024-04-23, status=FINALIZED}
Borrowing{ id=5, borrowerId=6, borrower=Andrés López, requestedDate=2024-04-01, returnDate=2024-04-11, status=BORROWED}
Borrowing{ id=6, borrowerId=90, borrower=lector, requestedDate=2024-04-16, returnDate=2024-04-25, status=REQUESTED}
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request |
5. Search by email | 6. Confirm borrowing | 7. Finalize borrowing | 8. Delete borrowing | 0. Back
>
```

- Empleado accede a detalles de pedidos de otros usuarios

```
Successful login as: User{id=91, name='empleado', email='emple', role=EMPLOYEE}
Main menu: 2. Books | 3. Novels | 4. Borrowings | 8. Idioma Español | 9. Log out | 0. Exit
> 4
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request |
5. Search by email | 6. Confirm borrowing | 7. Finalize borrowing | 8. Delete borrowing | 0. Back
> 3
Type the id of the borrowing you want to see its details
> 1
Borrowing details:
Borrowing{ id=1, borrowerId=2, borrower=Juan Pérez, requestedDate=2024-04-03, returnDate=2024-04-17, status=BORROWED}
Items list:
Book{id=1, title='Cien años de soledad', author='Gabriel García Márquez', copies=6, copiesBorrowed=1, field='Ficción', pages=432}
Book{id=2, title='El amor en los tiempos del cólera', author='Gabriel García Márquez', copies=1, copiesBorrowed=1, field='Ficción', pages=368}

Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request |
5. Search by email | 6. Confirm borrowing | 7. Finalize borrowing | 8. Delete borrowing | 0. Back
>
```

- Búsqueda de préstamo por correo

```
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request |
5. Search by email | 6. Confirm borrowing | 7. Finalize borrowing | 8. Delete borrowing | 0. Back
> 5
Type the email of the user you want to list its borrowings
> lec
List of borrowings made by:
Borrowing{ id=6, borrowerId=90, borrower=lector, requestedDate=2024-04-16, returnDate=2024-04-25, status=REQUESTED}
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request |
5. Search by email | 6. Confirm borrowing | 7. Finalize borrowing | 8. Delete borrowing | 0. Back
>
```

- Empleado marca como confirmado (prestado) el estado del préstamo

```
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request |
5. Search by email | 6. Confirm borrowing | 7. Finalize borrowing | 8. Delete borrowing | 0. Back
> 6
Type the id of the borrowing you want to confirm as delivered
> 6
Borrowing confirmed successfully
Borrowings menu: 1. List selected items | 2. List all borrowings | 3. Show borrowing details | 4. Confirm request |
5. Search by email | 6. Confirm borrowing | 7. Finalize borrowing | 8. Delete borrowing | 0. Back
> 3
Type the id of the borrowing you want to see its details
> 6
Borrowing details:
Borrowing{ id=6, borrowerId=90, borrower=lector, requestedDate=2024-04-16, returnDate=2024-04-25, status=BORROWED}
```

Funcionalidad extra en desarrollo: Internacionalización

```
Welcome to La Pingüinera library!!
Main menu: 2. Books | 3. Novels | 4. Borrowings | 5. Admin menu | 8. Idioma Español | 9. Log out | 0. Exit
> 8
Menú principal:  2. Libros | 3. Novelas | 4. Prestamos | 5. Menu admin | 8. English Language | 9. Log out | 0. Exit
>
```

Aprendido durante este taller

- Enums con values en Java
- Manejo de relaciones M:N en la capa de persistencia
- Sistema de roles y permisos
- Menús de consola con opciones dinámicas
- Internacionalización i18n de strings con resource bundles.
- Cargar System Properties desde un archivo .env
- Leer y escribir archivos csv
- Aplicar recomendaciones de SonarQube

Posibles mejoras identificadas

- Implementar funcionalidad de login con manejo de sesión, seguridad real y funcionalidad de cambio de contraseña
- Reestructurar el modelo de la base de datos para reducir relaciones