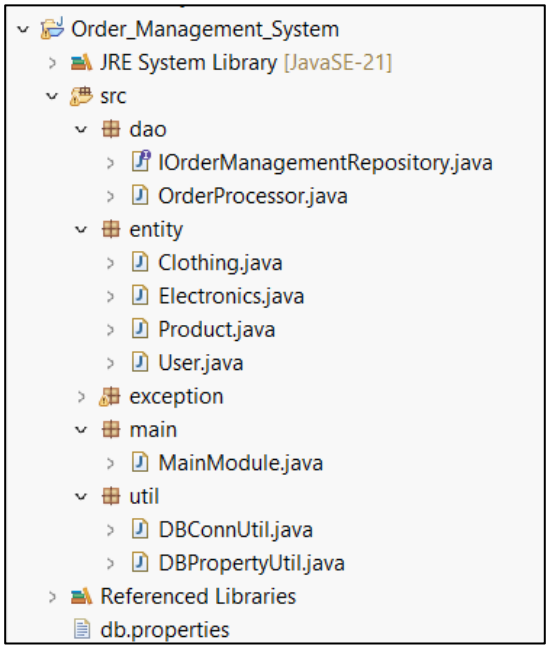
Coding Challenge - Order Management System

1. **The following Directory structure is to be followed in the application.**
    - **entity**
        - Create entity classes in this package. All entity class should not have any business logic.
    - **dao**
        - Create Service Provider interface to showcase functionalities.
        - Create the implementation class for the above interface with db interaction.
    - **exception**
        - Create user defined exceptions in this package and handle exceptions whenever needed.
    - **util**
        - Create a DBPropertyUtil class with a static function which takes property file name as parameter and returns connection string.
        - Create a DBConnUtil class which holds static method which takes connection string as parameter file and returns connection object (Use method defined in DBPropertyUtil class to get the connection String).
    - **main**
        - Create a class MainModule and demonstrate the functionalities in a menu driven application.



Problem Statement:

Create SQL Schema from the product and user class, use the class attributes for table column names.

1. Create a base class called Product with the following attributes:
   - productId (int)
   - productName (String)
   - description (String)
   - price (double)
   - quantityInStock (int)
   - type (String) [Electronics/Clothing]
2. Implement constructors, getters, and setters for the Product class.

```
package entity;

public class Product {
	private int productId;
	private String productName;
	private String description;
	private double price;
	private int quantityInStock;
	private String type;

	//default constructor
		public Product() {
			super();
			// TODO Auto-generated constructor stub
		}

		//parameterized constructor
		public Product(int productId, String productName, String description, double price, int
quantityInStock,
					String type) {
			super();
			this.productId = productId;
			this.productName = productName;
			this.description = description;
			this.price = price;
			this.quantityInStock = quantityInStock;
			this.type = type;
		}

		//getter & setter
		public int getProductId() {
			return productId;
		}

		public void setProductId(int productId) {
			this.productId = productId;
		}

		public String getProductName() {
			return productName;
		}
```

```
	public void setProductName(String productName) {
		this.productName = productName;
	}

	public String getDescription() {
		return description;
	}

	public void setDescription(String description) {
		this.description = description;
	}

	public double getPrice() {
		return price;
	}

	public void setPrice(double price) {
		this.price = price;
	}

	public int getQuantityInStock() {
		return quantityInStock;
	}

	public void setQuantityInStock(int quantityInStock) {
		this.quantityInStock = quantityInStock;
	}

	public String getType() {
		return type;
	}

	public void setType(String type) {
		this.type = type;
	}
	@Override
	public String toString() {
		return "Product [productId=" + productId + ", productName=" + productName + ",
description=" + description
				+ ", price=" + price + ", quantityInStock=" + quantityInStock + ", type=" +
type + "]";
	}

}
```

3. Create a subclass Electronics that inherits from Product. Add attributes specific to electronics products, such as:
    - brand (String)
    - warrantyPeriod (int)

```
package entity;

public class Electronics extends Product {
```

```
        //instance variable
        private String brand;
    private int warrantyPeriod;

    //default constructor
        public Electronics() {
                super();
        }
        //parameterized constructor
        public Electronics(int productId, String productName, String description, double price, int quantityInStock,
                        String type, String brand, int warrantyPeriod) {
                super(productId, productName, description, price, quantityInStock, type);
                this.brand = brand;
       this.warrantyPeriod = warrantyPeriod;
        }

        //getter and setter
        public String getBrand() {
                return brand;
        }
        public void setBrand(String brand) {
                this.brand = brand;
        }
        public int getWarrantyPeriod() {
                return warrantyPeriod;
        }
        public void setWarrantyPeriod(int warrantyPeriod) {
                this.warrantyPeriod = warrantyPeriod;
        }
      @Override
        public String toString() {
                return "Electronics [brand=" + brand + ", warrantyPeriod=" + warrantyPeriod + "]";
        }
}
```

4. Create a subclass Clothing that also inherits from Product. Add attributes specific to clothing products, such as:

- size (String)
- color (String)

```
package entity;

public class Clothing  extends Product{
        //instance variable
        private String size;
    private String color;

    //default constructor
        public Clothing() {
                super();
        }

        //parameterized constructor
        public Clothing(int productId, String productName, String description, double price, int quantityInStock,
```

```
                                String type, String size, String color) {
                        super(productId, productName, description, price, quantityInStock, type);
                        this.size = size;
        this.color = color;
            }

            //getter and setter
            public String getSize() {
                        return size;
            }

            public void setSize(String size) {
                        this.size = size;
            }

            public String getColor() {
                        return color;
            }

            public void setColor(String color) {
                        this.color = color;
            }

          @Override
            public String toString() {
                        return "Clothing [size=" + size + ", color=" + color + "]";
            }


}
```

5. Create a User class with attributes:

- userId (int)
- username (String)
- password (String)
- role (String) // "Admin" or "User"

```
package entity;

public class User {
            //instance variable
            private int userId;
    private String username;
    private String password;
    private String role;

    //default constructor
            public User() {
                        super();

            }

            //parameterized constructor
            public User(int userId, String username, String password, String role) {
```

```
                super();
                this.userId = userId;
                this.username = username;
                this.password = password;
                this.role = role;
        }

        //getter and setter
        public int getUserId() {
                return userId;
        }

        public void setUserId(int userId) {
                this.userId = userId;
        }

        public String getUsername() {
                return username;
        }

        public void setUsername(String username) {
                this.username = username;
        }

        public String getPassword() {
                return password;
        }

        public void setPassword(String password) {
                this.password = password;
        }

        public String getRole() {
                return role;
        }

        public void setRole(String role) {
                this.role = role;
        }
}
```

6. Define an interface/abstract class named IOrderManagementRepository with methods for:
   • **createOrder** (User user, list of products): check the user as already present in database to create order or create user (store in database) and create order.
   • **cancelOrder** (int userId, int orderId): check the userid and orderId already present in database and cancel the order. if any userId or orderId not present in database throw exception corresponding UserNotFound or OrderNotFound exception
   • **createProduct** (User user, Product product): check the admin user as already present in database and create product and store in database.
   • **createUser** (User user): create user and store in database for further development
   • **getAllProducts():** return all product list from the database.

• **getOrderByUser**(User user): return all product ordered by specific user from database.

```
package dao;
import entity.Product;
import entity.User;
import java.util.List;

public interface IOrderManagementRepository {
            //createUser
            void createUser(User user) throws  Exception;
            //createProduct
   void createProduct(User user, Product product) throws Exception;
   //createOrder
   void createOrder(User user, List<Product> productList) throws Exception;
   //cancelOrder
   void cancelOrder(int userId, int orderId) throws Exception;
   //getAllProducts
   List<Product> getAllProducts() throws Exception;
   //getOrderByUser
   List<Product> getOrderByUser(User user) throws Exception;
}
```

7. Implement the IOrderManagementRepository interface/abstractclass in a class called OrderProcessor. This class will be responsible for managing orders.

```
package dao;
import entity.*;
import util.DBConnUtil;

import java.sql.*;
import java.util.*;

public class OrderProcessor implements IOrderManagementRepository {

            private static Connection connection;

   public OrderProcessor() throws SQLException {
      connection = DBConnUtil.getDbConnection();
   }

   public void createUser(User user) throws Exception {
      String sql = "INSERT INTO users (userId, username, password, role) VALUES (?, ?, ?, ?)";
      try (PreparedStatement stmt = connection.prepareStatement(sql)) {
         stmt.setInt(1, user.getUserId());
         stmt.setString(2, user.getUsername());
         stmt.setString(3, user.getPassword());
         stmt.setString(4, user.getRole());
         stmt.executeUpdate();
         System.out.println("User created successfully!");
      } catch (SQLException e) {
         System.out.println("Error creating user: " + e.getMessage());
```

```
            throw new Exception("Failed to create user", e);
        }
    }

    @Override
    public List<Product> getAllProducts() throws Exception {
        List<Product> products = new ArrayList<>();

        String sql = "SELECT * FROM products";
        try (PreparedStatement stmt = connection.prepareStatement(sql);
             ResultSet rs = stmt.executeQuery()) {

            while (rs.next()) {
                String type = rs.getString("type");

                if (type.equalsIgnoreCase("Electronics")) {
                    Electronics e = new Electronics(
                        rs.getInt("productId"),
                        rs.getString("productName"),
                        rs.getString("description"),
                        rs.getDouble("price"),
                        rs.getInt("quantityInStock"),
                        type,
                        rs.getString("brand"),
                        rs.getInt("warrantyPeriod")
                    );
                    products.add(e);
                } else if (type.equalsIgnoreCase("Clothing")) {
                    Clothing c = new Clothing(
                        rs.getInt("productId"),
                        rs.getString("productName"),
                        rs.getString("description"),
                        rs.getDouble("price"),
                        rs.getInt("quantityInStock"),
                        type,
                        rs.getString("size"),
                        rs.getString("color")
                    );
                    products.add(c);
                }
            }
        } catch (SQLException e) {
            System.out.println("Error retrieving products: " + e.getMessage());
            throw new Exception("Failed to retrieve products", e);
        }

        return products;
    }

    @Override
    public void createProduct(User user, Product product) throws Exception {
        if (!user.getRole().equalsIgnoreCase("Admin")) {
            throw new Exception("Only admin can create a product.");
        }

        String query = "INSERT INTO products (productId, productName, description, price, quantityInStock, type, brand,
warrantyPeriod, size, color) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
```

```
        try (PreparedStatement stmt = connection.prepareStatement(query)) {
            stmt.setInt(1, product.getProductId());
            stmt.setString(2, product.getProductName());
            stmt.setString(3, product.getDescription());
            stmt.setDouble(4, product.getPrice());
            stmt.setInt(5, product.getQuantityInStock());
            stmt.setString(6, product.getType());

            // Set subclass-specific fields
            if (product instanceof Electronics e) {
                stmt.setString(7, e.getBrand());
                stmt.setInt(8, e.getWarrantyPeriod());
                stmt.setNull(9, Types.VARCHAR);
                stmt.setNull(10, Types.VARCHAR);
            } else if (product instanceof Clothing c) {
                stmt.setNull(7, Types.VARCHAR);
                stmt.setNull(8, Types.INTEGER);
                stmt.setString(9, c.getSize());
                stmt.setString(10, c.getColor());
            } else {
                // For safety: if product is not Electronics or Clothing
                stmt.setNull(7, Types.VARCHAR);
                stmt.setNull(8, Types.INTEGER);
                stmt.setNull(9, Types.VARCHAR);
                stmt.setNull(10, Types.VARCHAR);
            }

            stmt.executeUpdate();
            System.out.println("Product created successfully!");
        } catch (SQLException e) {
            System.out.println("Error creating product: " + e.getMessage());
            throw new Exception("Failed to create product", e);
        }
    }

                    @Override
                    public void createOrder(User user, List<Product> productList) throws Exception {
                        try {
                            connection.setAutoCommit(false);

                            // Check if user exists
                            PreparedStatement userCheck = connection.prepareStatement("SELECT * FROM users WHERE
userId = ?");
                            userCheck.setInt(1, user.getUserId());
                            ResultSet rs = userCheck.executeQuery();

                            if (!rs.next()) {
                                // User doesn't exist, create them
                                createUser(user);
                            }

                            // Insert into orders table
                            String orderQuery = "INSERT INTO orders (userId) VALUES (?)";
                            PreparedStatement orderStmt = connection.prepareStatement(orderQuery,
Statement.RETURN_GENERATED_KEYS);
                            orderStmt.setInt(1, user.getUserId());
```

```
                orderStmt.executeUpdate();

                ResultSet generatedKeys = orderStmt.getGeneratedKeys();
                int orderId = -1;
                if (generatedKeys.next()) {
                    orderId = generatedKeys.getInt(1);
                } else {
                    throw new Exception("Order ID generation failed.");
                }

                // Insert into order_products table
                String orderProductQuery = "INSERT INTO order_products (orderId, productId) VALUES (?, ?)";
                PreparedStatement orderProductStmt = connection.prepareStatement(orderProductQuery);

                for (Product product : productList) {
                    orderProductStmt.setInt(1, orderId);
                    orderProductStmt.setInt(2, product.getProductId());
                    orderProductStmt.executeUpdate();
                }

                connection.commit();
                System.out.println("Order created successfully with ID: " + orderId);
            } catch (SQLException e) {
                connection.rollback();
                System.out.println("Order creation failed: " + e.getMessage());
                throw new Exception("Failed to create order", e);
            } finally {
                connection.setAutoCommit(true);
            }




    }


            @Override
            public void cancelOrder(int userId, int orderId) throws Exception {
                try {
                    connection.setAutoCommit(false);

                    // Check if order exists and belongs to the user
                    PreparedStatement checkStmt = connection.prepareStatement("SELECT * FROM orders WHERE
orderId = ? AND userId = ?");
                    checkStmt.setInt(1, orderId);
                    checkStmt.setInt(2, userId);
                    ResultSet rs = checkStmt.executeQuery();

                    if (!rs.next()) {
                        throw new Exception("Order ID or User ID not found!");
                    }

                    // Delete from order_products first
                    PreparedStatement deleteOP = connection.prepareStatement("DELETE FROM order_products
WHERE orderId = ?");
                    deleteOP.setInt(1, orderId);
                    deleteOP.executeUpdate();
```

```
                // Then delete from orders
                PreparedStatement deleteOrder = connection.prepareStatement("DELETE FROM orders WHERE
orderId = ?");
                deleteOrder.setInt(1, orderId);
                deleteOrder.executeUpdate();

                connection.commit();
                System.out.println("Order cancelled successfully!");
            } catch (SQLException e) {
                connection.rollback();
                throw new Exception("Failed to cancel order: " + e.getMessage(), e);
            } finally {
                connection.setAutoCommit(true);
            }
        }



        @Override
        public List<Product> getOrderByUser(User user) throws Exception {
            List<Product> products = new ArrayList<>();

            String query = """
                SELECT p.* FROM products p
                JOIN order_products op ON p.productId = op.productId
                JOIN orders o ON op.orderId = o.orderId
                WHERE o.userId = ?
            """;

            try (PreparedStatement stmt = connection.prepareStatement(query)) {
                stmt.setInt(1, user.getUserId());
                ResultSet rs = stmt.executeQuery();

                while (rs.next()) {
                    String type = rs.getString("type");

                    if (type.equalsIgnoreCase("Electronics")) {
                        Electronics e = new Electronics(
                            rs.getInt("productId"),
                            rs.getString("productName"),
                            rs.getString("description"),
                            rs.getDouble("price"),
                            rs.getInt("quantityInStock"),
                            type,
                            rs.getString("brand"),
                            rs.getInt("warrantyPeriod")
                        );
                        products.add(e);
                    } else if (type.equalsIgnoreCase("Clothing")) {
                        Clothing c = new Clothing(
                            rs.getInt("productId"),
                            rs.getString("productName"),
                            rs.getString("description"),
                            rs.getDouble("price"),
                            rs.getInt("quantityInStock"),
```

```
                                type,
                                rs.getString("size"),
                                rs.getString("color")
                            );
                            products.add(c);
                        }
                    }
                }
                return products;
        }

}
```

8. Create DBUtil class and add the following method.

- DBPropertyUtil

```
package util;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class DBPropertyUtil {

        //this method takes the filename which contains db connection like
                        //user name, pwd, port number, protocol and db name as an argument
                        //and returns a connection
                        public static String getConnectionString(String fileName)throws IOException {
                                String connStr=null;
                                Properties props=new Properties();
                                FileInputStream fis=new FileInputStream(fileName);
                                props.load(fis);

                                String user = props.getProperty("user");
                                String password = props.getProperty("password");
                    String port = props.getProperty("port");
                    String database = props.getProperty("database");
                    String protocol = props.getProperty("protocol");
                    String system = props.getProperty("system");

                    connStr=protocol+"//"+system+":"+port+"/"+database+"?user="+user+"&password="+password;
                                return connStr;
                        }

                }
```

- **DBConnUtil**

```
package util;

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnUtil {

	private static final String fileName = "db.properties";

	  public static Connection getDbConnection() {
	     Connection con = null;
	     String connString=null;
	     try {
	        connString = DBPropertyUtil.getConnectionString(fileName); // Get URL from properties
	     } catch (IOException e) {
	        System.out.println("Connection string could not be retrieved.");
	        e.printStackTrace();
	     }
	     if (connString != null) {
	          try {
	           con = DriverManager.getConnection(connString); // Get actual Connection object
	          }
	          catch (SQLException e) {
	        System.out.println("Database connection failed.");
	        e.printStackTrace();
	     }
	     }
	     return con;
	  }

}
```

9. Create OrderManagement main class and perform following operation: • main method to simulate the loan management system. Allow the user to interact with the system by entering choice from menu such as:

- **createUser**

```
Welcome to the Order Management System

--- Menu ---
1. Create User
2. Create Product (Admin only)
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Orders By User
7. Exit
Enter choice: 1
Enter User ID: 2
Enter Username: Sai
Enter Password: 2727
Enter Role (Admin/User): admin
User created successfully!
yeah!! User created successfully.
Do you want to continue (yes/no)? yes
```

➢ createProduct

```
Enter choice: 2
Enter User ID: 2
Enter Username: Sai
Enter Password: 2727
Enter Role: admin
Enter Product ID: 4
Enter Product Name: Smart Watch
Enter Description: With Fitness Check
Enter Price: 2000
Enter Quantity: 10
Enter Type (Electronics/Clothing): Electronics
Enter Brand: Boat
Enter Warranty Period (months): 12
Product created successfully!
oiii Product created successfully.
Do you want to continue (yes/no)? Session ended. Thank you!
```

➢ OrderCreated

```
Welcome to the Order Management System

--- Menu ---
1. Create User
2. Create Product (Admin only)
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Orders By User
7. Exit
Enter choice: 3
Enter User ID: 1
Enter Username: Smrthi
Enter Password: 143143
Enter Role: User
How many products to order? 1
Enter Product ID: 1
Order created successfully with ID: 1
Order placed successfully.
```

➢ cancelOrder

```
Welcome to the Order Management System

--- Menu ---
1. Create User
2. Create Product (Admin only)
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Orders By User
7. Exit
Enter choice: 4
Enter User ID: 1
Enter Order ID: 1
Order cancelled successfully!
```

➢ getAllProducts

```
Welcome to the Order Management System

--- Menu ---
1. Create User
2. Create Product (Admin only)
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Orders By User
7. Exit
Enter choice: 5
--- All Products ---
Electronics [brand=Dell, warrantyPeriod=24]
Clothing [size=M, color=Blue]
Electronics [brand=Samsung, warrantyPeriod=12]
Electronics [brand=Boat, warrantyPeriod=12]
Do you want to continue (yes/no)?
```

- **getOrderbyUser**