MINI – PROJECT
CRUD using collections

Product Management System Overview

Objective

To develop a comprehensive Java-based Product Management system that provides an intuitive and efficient way to manage product information. The system will enable users to perform CRUD (Create, Read, Update, Delete) operations on product data, ensuring that products are accurately managed and maintained.

Key Features

1. **Product Addition:**
    - Users can add new products to the system by providing details such as ID, name, and price.
2. **Product Retrieval:**
    - Users can retrieve and view details of a specific product or get a list of all products available in the system.
3. **Product Update:**
    - Users can update the information of existing products, including changing the name and price.
4. **Product Deletion:**
    - Users can delete products from the system based on product ID.
5. **Data Validation:**
    - The system will validate product details before adding or updating to ensure data integrity.
6. **Error Handling:**
    - Robust error handling for invalid operations, such as trying to add a product with a duplicate ID or update a non-existent product.
7. **In-Memory Data Storage:**
    - The system uses an in-memory repository to store product data, simulating a database without persistent storage.
8. **Service Layer:**

- A service layer manages interactions between the user interface and the data access layer, providing a clear API for CRUD operations.

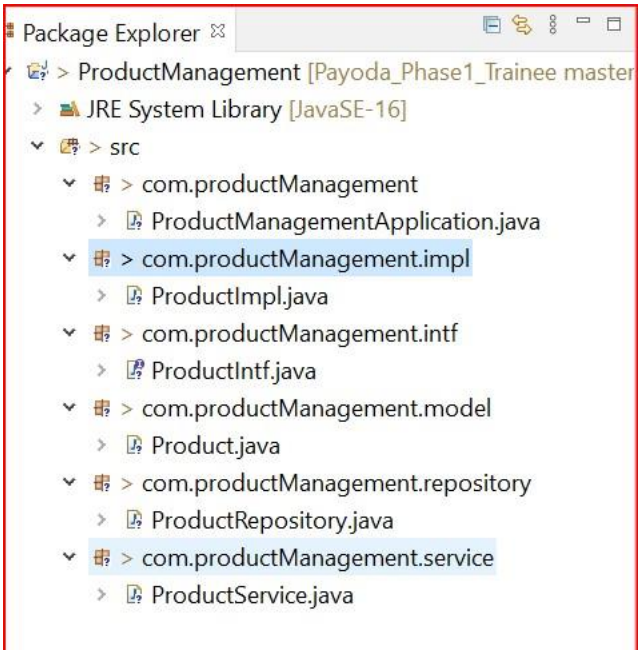
Expected User Interaction

1. **Add Product:**
    - **Action:** Users enter product details through a command-line interface or graphical user interface.
    - **Interaction:** System prompts for product ID, name, and price. After submission, the system confirms the addition.
2. **Retrieve Product:**
    - **Action:** Users request to view a specific product or all products.
    - **Interaction:** System displays the details of the requested product or lists all products.
3. **Update Product:**
    - **Action:** Users provide updated product details for an existing product.
    - **Interaction:** System retrieves the product by ID, applies updates, and confirms the changes.
4. **Delete Product:**
    - **Action:** Users request to delete a product by providing the product ID.
    - **Interaction:** System removes the product from the repository and confirms deletion.
5. **Handle Errors:**
    - **Action:** Users encounter errors during invalid operations.
    - **Interaction:** System displays error messages and provides instructions for corrective actions.

Technologies Used

1. **Java:**
    - **Description:** The primary programming language used to implement the application's functionality.
    - **Usage:** Core logic, CRUD operations, data modeling, and error handling.
2. **Collections Framework:**
    - **Description:** Java Collections Framework (List, Set, Map) used for in-memory data management.
    - **Usage:** Store and manage product data within the application.

Project structure:



Code:

com.productManagement:

ProductManagementApplication.java file:

```
package com.productManagement;


// Importing necessary classes.
import java.util.ArrayList;
import java.util.Scanner;


import com.productManagement.model.Product;
import com.productManagement.service.ProductService;
```

```
/**
* Main application class for managing products.
* Provides an interactive console interface for adding, deleting, updating, and retrieving products.
* Handles user input and interacts with ProductService to perform various product management operations.
*
* @author sanjeevkumar.v
*/
public class ProductManagementApplication {

  /**
   * The entry point of the application.
   * Initializes the ProductService and handles user interactions through a console menu.
   *
   * @param args command-line arguments (not used)
   */
  public static void main(String[] args) {
    ProductService productService = new ProductService();
    System.out.println("----------WELCOME TO ADMIN DASHBOARD----------");

    try (Scanner scanner = new Scanner(System.in)) {
      char userChoice;
      do {
        System.out.print("a. Add Products\nb. Delete Products\nc. Get Products\nd. Update Products\ne. Exit\nEnter your choice: ");
        userChoice = scanner.next().charAt(0);
```

```
        scanner.nextLine(); // Consume newline left-over

        switch (userChoice) {
          case 'a':
            // Add Product
            System.out.println("Enter Product Details [ProductName : Description : Price :
SupplierId : Goods : Goods Description]: ");
            String addProductInfo = scanner.nextLine();
            String addResult = productService.createProduct(addProductInfo);
            if (addResult != null) {
              System.out.println(addResult);
            } else {
              System.out.println("Product not added.");
            }
            break;

          case 'b':
            // Delete Product
            System.out.println("Enter ProductId: ");
            String deleteProductId = scanner.nextLine();
            if (productService.deleteProduct(deleteProductId)) {
              System.out.println("Product successfully deleted.");
            } else {
              System.out.println("Product not deleted or does not exist.");
            }
            break;
```

```
            case 'c':
                // Get Products
                ArrayList<Product> productList = (ArrayList<Product>)
productService.getAllProduct();
                System.out.println();
                if (!productList.isEmpty()) {
                    System.out.printf("%-30s%-30s%-30s%-10s%-50s%-15s%-20s", "ProductId",
"ProductName", "Description", "Unit Price", "Supplier", "Goods", "Goods Description");
                    System.out.println();
                    System.out.println("----------------------------------------------------------------------
----------------------");
                    for (Product product : productList) {
                        System.out.println(product);
                    }
                    System.out.println();
                } else {
                    System.out.println("No products available.");
                }
                break;

            case 'd':
                // Update Product
                System.out.println("Enter Product Details [ProductId : ProductName : Description :
Price : SupplierId : Goods : Goods Description]: ");
                String updateProductInfo = scanner.nextLine();
                String updateResult = productService.updateProduct(updateProductInfo);
```

```
                System.out.println(updateResult != null ? updateResult : "Product details not
updated.");
                break;

            case 'e':
                // Exit
                System.out.println(".....Thank you.....\nClosing Application.");
                break;

            default:
                System.out.println("Invalid choice. Please select a correct option.");
                break;
          }
        } while (userChoice != 'e');
      } catch (Exception e) {
        System.out.println();
        System.out.println("An error occurred. Please try again.");
        System.out.println("Error details: " + e.getMessage());
        System.out.println();
        // Log the exception and consider restarting or exiting the application as appropriate
      }
    }
}
```

com.productManagement.impl:

ProductImpl.java file:

```
package com.productManagement.impl;


//Importing all necessary classes for collections
import java.util.List;


import com.productManagement.intf.ProductIntf;
import com.productManagement.model.Product;
import com.productManagement.repository.ProductRepository;


/**
* Implementation of the ProductIntf interface for managing products.
* This class interacts with the ProductRepository to perform CRUD operations on Product objects.
*
* @author sanjeevkumar.v
*/
public class ProductImpl implements ProductIntf {


   private ProductRepository repository = new ProductRepository();


   /**
    * Creates a new product and stores it in the repository.
    *
    * @param product the Product object to be created
    * @return true if the product was successfully added, false otherwise
```

```
 */
@Override
public boolean createProduct(Product product) {
    try {
        return repository.addProduct(product);
    } catch (Exception e) {
        System.err.println("Error creating product: " + e.getMessage());
        return false;
    }
}

/**
 * Retrieves a product by its ID from the repository.
 *
 * @param productId the ID of the product to retrieve
 * @return the Product object with the specified ID, or null if not found
 */
@Override
public Product getProduct(String productId) {
    try {
        return repository.getProduct(productId);
    } catch (Exception e) {
        System.err.println("Error retrieving product: " + e.getMessage());
        return null;
    }
}
```

```
/**
 * Updates the details of an existing product in the repository.
 *
 * @param product the Product object with updated details
 * @return true if the product was successfully updated, false otherwise
 */
@Override
public boolean updateProduct(Product product) {
    try {
        return repository.updateProduct(product);
    } catch (Exception e) {
        System.err.println("Error updating product: " + e.getMessage());
        return false;
    }
}

/**
 * Deletes a product from the repository by its ID.
 *
 * @param productId the ID of the product to delete
 * @return true if the product was successfully deleted, false otherwise
 */
@Override
public boolean deleteProduct(String productId) {
    try {
```

```
            return repository.deleteProduct(productId);
        } catch (Exception e) {
            System.err.println("Error deleting product: " + e.getMessage());
            return false;
        }
    }

    /**
     * Retrieves all products from the repository.
     *
     * @return a List of all Product objects in the repository
     */
    @Override
    public List<Product> getAllProduct() {
        try {
            return repository.getAllProduct();
        } catch (Exception e) {
            System.err.println("Error retrieving products: " + e.getMessage());
            return List.of(); // Return an empty list in case of an error
        }
    }
}
```

com.productManagement.intf:

ProductIntf.java file:

```
package com.productManagement.intf;

//Importing all necessary classes for collections
import java.util.List;

import com.productManagement.model.Product;

/**
* Interface for managing products in the product management system.
* Defines CRUD operations that can be performed on Product objects.
* Implementing classes should provide concrete implementations for these methods.
*
* @author sanjeevkumar.v
*/
public interface ProductIntf {

   /**
    * Creates a new product and stores it in the repository.
    *
    * @param product the Product object to be created
    * @return true if the product was successfully created, false otherwise
    */
   boolean createProduct(Product product);

   /**
    * Retrieves a product by its ID.
```

```
 *
 * @param productId the ID of the product to retrieve
 * @return the Product object with the specified ID, or null if not found
 */
Product getProduct(String productId);

/**
 * Updates the details of an existing product.
 *
 * @param product the Product object with updated details
 * @return true if the product was successfully updated, false otherwise
 */
boolean updateProduct(Product product);

/**
 * Deletes a product by its ID.
 *
 * @param productId the ID of the product to delete
 * @return true if the product was successfully deleted, false otherwise
 */
boolean deleteProduct(String productId);

/**
 * Retrieves all products from the repository.
 *
 * @return a List of all Product objects
```

```
    */
    List<Product> getAllProduct();
}
```

com.productManagement.model:

Product.java file:

```
package com.productManagement.model;


/**
* Represents a product in the product management system.
* Contains details about the product including its ID, name, description, price, supplier information, and goods details.
* Provides methods to access and modify product attributes.
*
* @author sanjeevkumar.v
*/
public class Product {

    private String productId;
    private String productName;
    private String description;
    private double unitPrice;
    private String supplierInfo;
    private String goods;
    private String goodsDescription;
```

```
    /**
     * Constructs a Product object with the specified details.
     *
     * @param productId       the unique identifier of the product
     * @param productName     the name of the product
     * @param description     a description of the product
     * @param unitPrice       the price per unit of the product
     * @param supplierInfo    information about the supplier of the product
     * @param goods           type of goods
     * @param goodsDescription description of the goods
     */
    public Product(String productId, String productName, String description, double unitPrice,
String supplierInfo,
            String goods, String goodsDescription) {
        this.productId = productId;
        this.productName = productName;
        this.description = description;
        this.unitPrice = unitPrice;
        this.supplierInfo = supplierInfo;
        this.goods = goods;
        this.goodsDescription = goodsDescription;
    }

    // Getters and setters
```

```
public String getProductId() {
   return productId;
}

public void setProductId(String 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 getUnitPrice() {
   return unitPrice;
```

```
}

public void setUnitPrice(double unitPrice) {
   this.unitPrice = unitPrice;
}

public String getSupplierInfo() {
   return supplierInfo;
}

public void setSupplierInfo(String supplierInfo) {
   this.supplierInfo = supplierInfo;
}

public String getGoods() {
   return goods;
}

public void setGoods(String goods) {
   this.goods = goods;
}

public String getGoodsDescription() {
   return goodsDescription;
}
```

```
    public void setGoodsDescription(String goodsDescription) {
        this.goodsDescription = goodsDescription;
    }

    /**
     * Returns a string representation of the Product object.
     * The string contains all product details formatted in a readable way.
     *
     * @return a formatted string representing the product details
     */
    @Override
    public String toString() {
        return String.format("%-30s%-30s%-30s%-10.2f%-50s%-15s%-20s",
                productId, productName, description, unitPrice, supplierInfo, goods, goodsDescription);
    }
}
```

com.productManagement.repository:

ProductRepository.java file:

```
package com.productManagement.repository;

//Importing all necessary classes for collections
import java.util.*;

import com.productManagement.model.Product;
```

```
/**
* Repository class for managing product data.
* Provides methods to perform CRUD operations on products stored in an in-memory map.
*
* @author sanjeevkumar.v
*/
public class ProductRepository {

  // Map to store products with productId as the key
  private Map<String, Product> productMap = new HashMap<>();

  /**
   * Adds a new product to the repository.
   *
   * @param product the Product object to be added
   * @return true if the product was successfully added, false otherwise
   */
  public boolean addProduct(Product product) {
    if (product == null || product.getProductId() == null) {
      throw new IllegalArgumentException("Product or Product ID cannot be null");
    }
    productMap.put(product.getProductId(), product);
    return productMap.containsKey(product.getProductId());
  }
```

```
/**
 * Retrieves a product by its ID.
 *
 * @param productId the ID of the product to retrieve
 * @return the Product object with the specified ID, or null if not found
 */
public Product getProduct(String productId) {
    if (productId == null) {
        throw new IllegalArgumentException("Product ID cannot be null");
    }
    return productMap.get(productId);
}

/**
 * Updates the details of an existing product.
 *
 * @param product the Product object with updated details
 * @return true if the product was successfully updated, false otherwise
 */
public boolean updateProduct(Product product) {
    if (product == null || product.getProductId() == null) {
        throw new IllegalArgumentException("Product or Product ID cannot be null");
    }
    return productMap.replace(product.getProductId(), product) != null;
}
```

```
    /**
     * Deletes a product from the repository by its ID.
     *
     * @param productId the ID of the product to delete
     * @return true if the product was successfully deleted, false otherwise
     */
    public boolean deleteProduct(String productId) {
        if (productId == null) {
            throw new IllegalArgumentException("Product ID cannot be null");
        }
        return productMap.remove(productId) != null;
    }

    /**
     * Retrieves all products from the repository.
     *
     * @return a List of all Product objects in the repository
     */
    public List<Product> getAllProduct() {
        return new ArrayList<>(productMap.values());
    }
}
```

com.productManagement.service:

ProductService.java file:

```
package com.productManagement.service;

//Importing all necessary classes for collections
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;

import com.productManagement.impl.ProductImpl;
import com.productManagement.intf.ProductIntf;
import com.productManagement.model.Product;

/**
* Service class for managing products.
* Provides business logic for creating, retrieving, updating, and deleting products.
* Uses ProductIntf implementation to interact with product data.
*
* @author sanjeevkumar.v
*/
public class ProductService {

   private ProductIntf productIntf = new ProductImpl();

   /**
    * Creates a new product with the given product information.
    *
    * @param productInfo a string containing product details separated by colons
```

```
    * @return a message containing the product ID if the product was created successfully, or null otherwise
    */
   public String createProduct(String productInfo) {
      try {
         String[] productDetails = productInfo.split(":");

         if (productDetails.length != 6) {
            throw new IllegalArgumentException("Invalid product information format");
         }

         String productId = "PROD" + generateUniqueId();
         Product product = new Product(productId, productDetails[0], productDetails[1],
               Double.parseDouble(productDetails[2]), productDetails[3], productDetails[4],
               productDetails[5]);

         if (productIntf.createProduct(product)) {
            return "ProductId: " + productId;
         }
      } catch (NumberFormatException e) {
         System.err.println("Invalid number format in product information: " + e.getMessage());
      } catch (Exception e) {
         System.err.println("Error creating product: " + e.getMessage());
      }
      return null;
   }
```

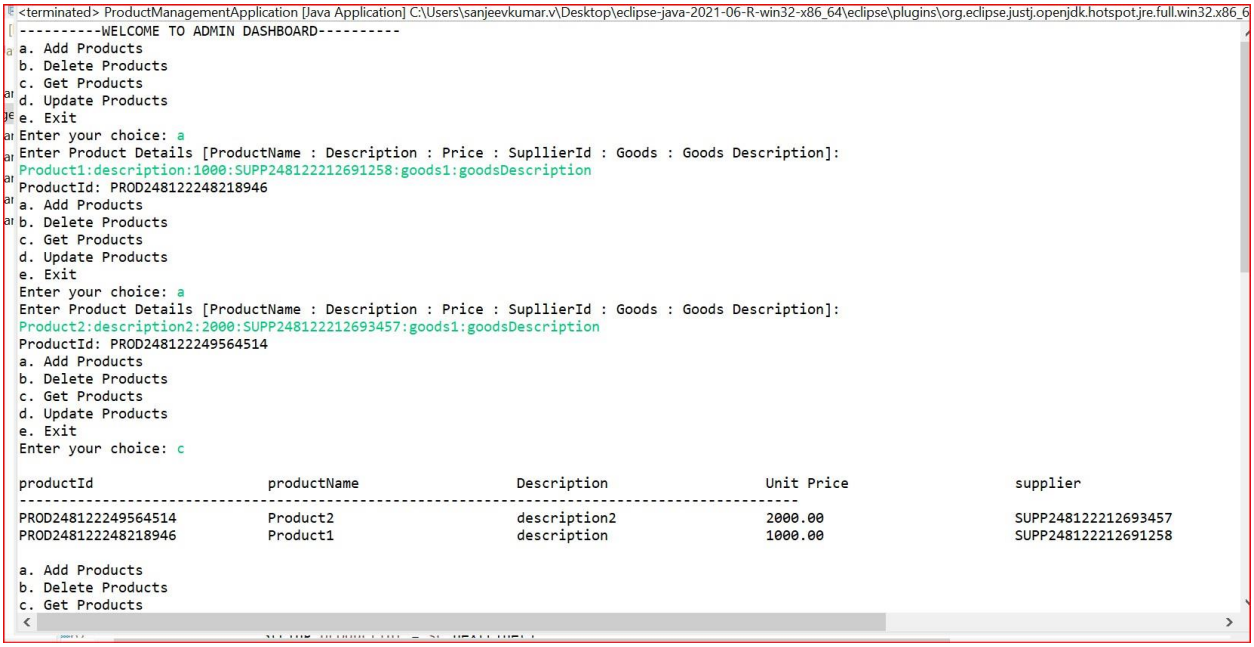
```
/**
 * Retrieves a product by its ID.
 *
 * @param productId the ID of the product to retrieve
 * @return the Product object if found, or null otherwise
 */
public Product getProduct(String productId) {
   if (productId == null || productId.trim().isEmpty()) {
      throw new IllegalArgumentException("Product ID cannot be null or empty");
   }
   return productIntf.getProduct(productId);
}

/**
 * Updates an existing product with the given product information.
 *
 * @param productInfo a string containing product details separated by colons
 * @return a message indicating whether the update was successful or not
 */
public String updateProduct(String productInfo) {
   try {
      String[] productDetails = productInfo.split(":");

      if (productDetails.length != 7) {
         throw new IllegalArgumentException("Invalid product information format");
```

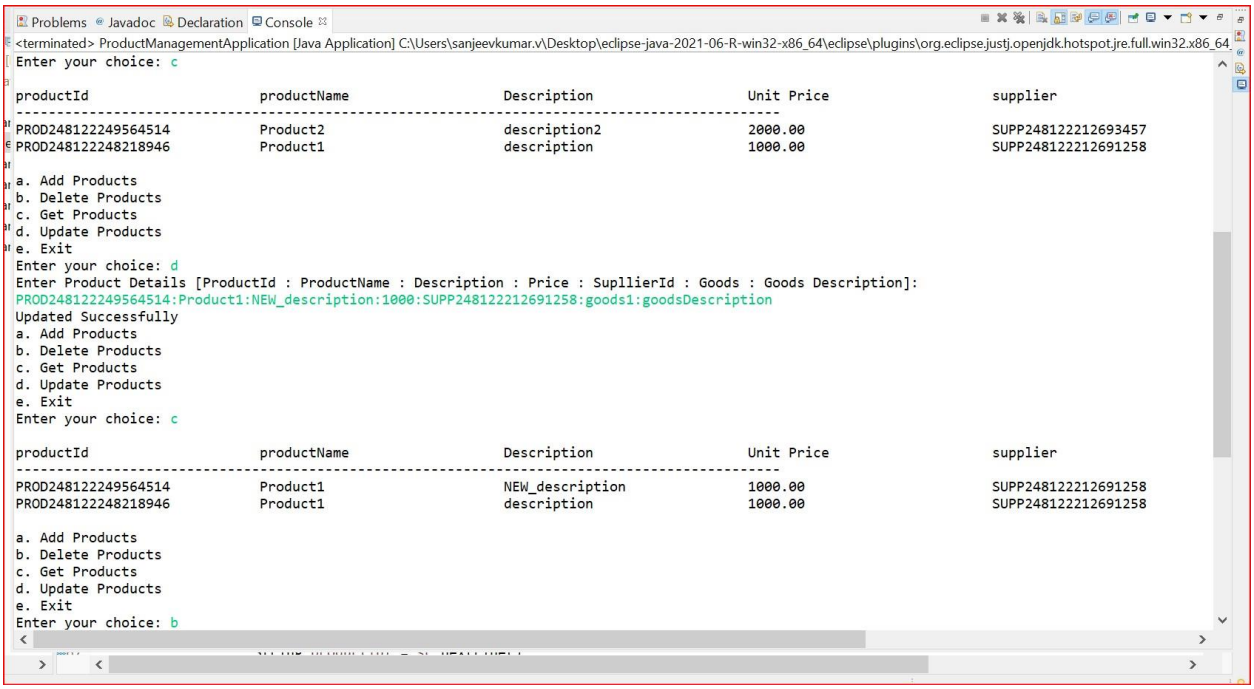
```
        }

        Product product = new Product(productDetails[0], productDetails[1], productDetails[2],
                Double.parseDouble(productDetails[3]), productDetails[4], productDetails[5],
                productDetails[6]);

        return productIntf.updateProduct(product) ? "Updated Successfully" : "Not updated.";
    } catch (NumberFormatException e) {
        System.err.println("Invalid number format in product information: " + e.getMessage());
    } catch (Exception e) {
        System.err.println("Error updating product: " + e.getMessage());
    }
    return "Update failed.";
}

/**
 * Deletes a product by its ID.
 *
 * @param productId the ID of the product to delete
 * @return true if the product was successfully deleted, false otherwise
 */
public boolean deleteProduct(String productId) {
    if (productId == null || productId.trim().isEmpty()) {
        throw new IllegalArgumentException("Product ID cannot be null or empty");
    }
    return productIntf.deleteProduct(productId);
```

```
}

/**
 * Retrieves all products.
 *
 * @return a List of all Product objects
 */
public List<Product> getAllProduct() {
   return productIntf.getAllProduct();
}

/**
 * Generates a unique identifier for a product.
 *
 * @return a unique product ID
 */
public String generateUniqueId() {
   return generateSCMId();
}

/**
 * Generates a unique SCM ID based on the current timestamp and a random suffix.
 *
 * @return a unique SCM ID
 */
private String generateSCMId() {
```

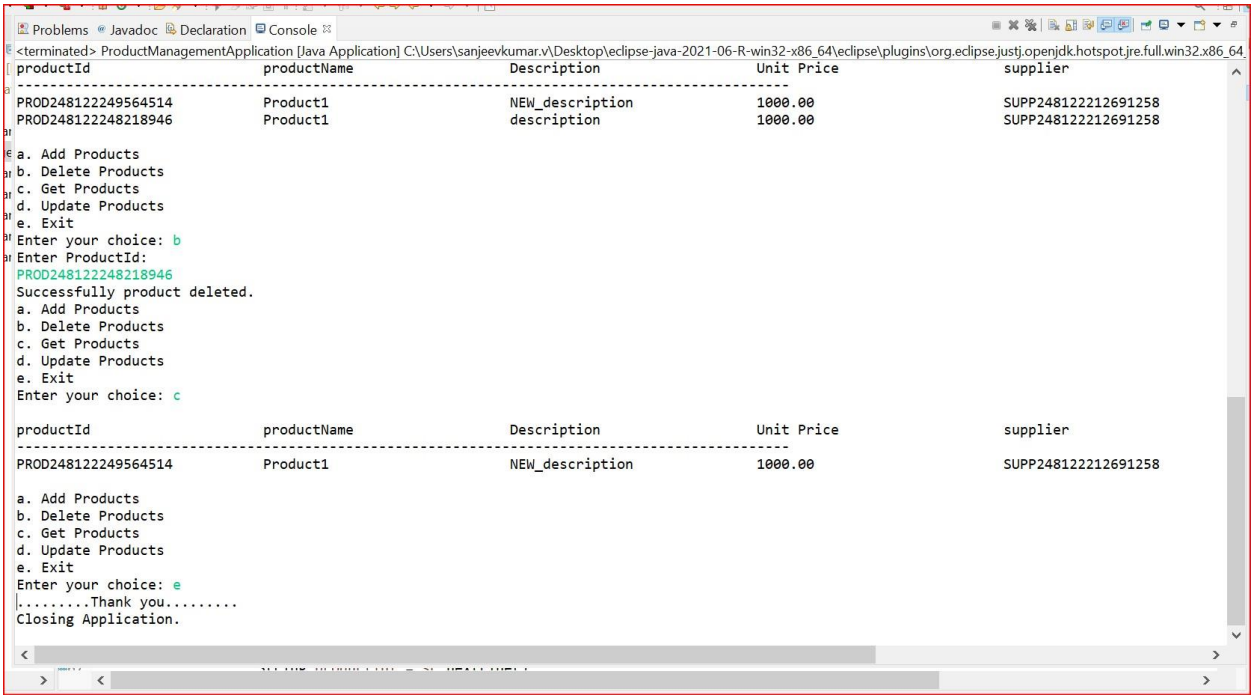
```
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyMMddHHmmss");
    String timestamp = dateFormat.format(new Date());
    int randomSuffix = (int) (Math.random() * 1000); // Add a random suffix for uniqueness
    return timestamp + String.format("%03d", randomSuffix);
  }
}
```

Output:





```
Problems  Javadoc  Declaration  Console
<terminated> ProductManagementApplication [Java Application] C:\Users\sanjeevkumar.v\Desktop\eclipse-java-2021-06-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64
Enter your choice: c

productId                   productName             Description             Unit Price              supplier
-------------------------------------------------------------------------------------------------
PROD248122249564514         Product2                description2            2000.00                 SUPP248122212693457
PROD248122248218946         Product1                description             1000.00                 SUPP248122212691258

a. Add Products
b. Delete Products
c. Get Products
d. Update Products
e. Exit
Enter your choice: d
Enter Product Details [ProductId : ProductName : Description : Price : SupllierId : Goods : Goods Description]:
PROD248122249564514:Product1:NEW_description:1000:SUPP248122212691258:goods1:goodsDescription
Updated Successfully
a. Add Products
b. Delete Products
c. Get Products
d. Update Products
e. Exit
Enter your choice: c

productId                   productName             Description             Unit Price              supplier
-------------------------------------------------------------------------------------------------
PROD248122249564514         Product1                NEW_description         1000.00                 SUPP248122212691258
PROD248122248218946         Product1                description             1000.00                 SUPP248122212691258

a. Add Products
b. Delete Products
c. Get Products
d. Update Products
e. Exit
Enter your choice: b
```



```
Problems  Javadoc  Declaration  Console
<terminated> ProductManagementApplication [Java Application] C:\Users\sanjeevkumar.v\Desktop\eclipse-java-2021-06-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64
productId                   productName             Description             Unit Price              supplier
-------------------------------------------------------------------------------------------------
PROD248122249564514         Product1                NEW_description         1000.00                 SUPP248122212691258
PROD248122248218946         Product1                description             1000.00                 SUPP248122212691258

a. Add Products
b. Delete Products
c. Get Products
d. Update Products
e. Exit
Enter your choice: b
Enter ProductId:
PROD248122248218946
Successfully product deleted.
a. Add Products
b. Delete Products
c. Get Products
d. Update Products
e. Exit
Enter your choice: c

productId                   productName             Description             Unit Price              supplier
-------------------------------------------------------------------------------------------------
PROD248122249564514         Product1                NEW_description         1000.00                 SUPP248122212691258

a. Add Products
b. Delete Products
c. Get Products
d. Update Products
e. Exit
Enter your choice: e
.........Thank you.........
Closing Application.
```

Use Case Specifications

1. Use Case: Add Product

Use Case ID: UC01

Name: Add Product

Description: Allows a user to add a new product to the system.

Actors: Product Manager, Administrator

Preconditions: The user must be authenticated and have the necessary permissions.

Postconditions: The new product is added to the system and stored in the repository.

Main Flow:

1. The user provides product details: ID, name, and price.
2. The system validates the product details.
3. The system creates a new `Product` object.
4. The system adds the `Product` object to the repository.
5. The system confirms that the product has been successfully added.

Alternative Flow:

- **Generate Product ID:** If the product data is generate (e.g., missing fields or Generate ID), the system displays an message and requests correct information.

Exceptions:

- **Repository Error:** If the repository fails to store the product, an error message is displayed.

2. Use Case: Update Product

Use Case ID: UC02

Name: Update Product

Description: Allows a user to update the details of an existing product.

Actors: Product Manager, Administrator

Preconditions: The product must exist in the system.

Postconditions: The product details are updated in the repository.

Main Flow:

1. The user provides the updated product details (ID, name, price).
2. The system retrieves the `Product` object from the repository using the ID.
3. The system validates the updated product details.
4. The system updates the existing `Product` object with new details.
5. The system saves the updated `Product` object in the repository.
6. The system confirms that the product has been successfully updated.

Alternative Flow:

- **Product Not Found:** If the product with the given ID does not exist, the system displays an error message.

Exceptions:

- **Repository Error:** If the repository fails to update the product, an error message is displayed.

3. Use Case: Delete Product

Use Case ID: UC03

Name: Delete Product

Description: Allows a user to delete a product from the system.

Actors: Product Manager, Administrator

Preconditions: The product must exist in the system.

Postconditions: The product is removed from the repository.

Main Flow:

1. The user provides the ID of the product to be deleted.

2. The system retrieves the `Product` object from the repository using the ID.
3. The system removes the `Product` object from the repository.
4. The system confirms that the product has been successfully deleted.

Alternative Flow:

- **Product Not Found:** If the product with the given ID does not exist, the system displays an error message.

Exceptions:

- **Repository Error:** If the repository fails to delete the product, an error message is displayed.

5. Use Case: Get All Products

Use Case ID: UC04

Name: Get All Products

Description: Allows a user to retrieve a list of all products in the system.

Actors: Product Manager, Administrator

Preconditions: The system must have products stored in the repository.

Postconditions: A list of all products is displayed to the user.

Main Flow:

1. The user requests the list of all products.
2. The system retrieves the list of all `Product` objects from the repository.
3. The system displays the list of products to the user.

Exceptions:

- **Repository Error:** If the repository fails to retrieve the list of products, an error message is displayed.