Java Fundamentals
Section 7 Part 1: Creating an Inventory Project
Project

Overview

This project will progress with you throughout Sections 4, 5, 6, and 7 of the course. After each section there will be more to add until it
builds into a complete Java application to maintain Inventory. For each part, build upon the last part so that both the old and new
requirements are met. Include all parts in a package called inventory.
Create an inventory program that can be used for a range of different products (cds, dvds, software, etc.).
Topic(s):
• Modifying programs
 Creating Static methods (Section 7.3)
 using parameters in a method (Section 7.1)
 Return a value from a method (Section 7.1)
• Adding methods(behaviours) to an existing class (Section 7.2)
• Implementing a user interface (Sections 5.1, 5.2, 6.2)
1. Open the inventory program that was updated in Section 6: Creating an inventory Project
2. You are now going to modify your code so that the main class will not do any processing but simply call static methods when
required.
a) Create a static method in the ProductTester class after the end of the main method called displayInventory. This
method will not return any values and will accept the products array as a parameter. Remember when you pass
an array as a parameter you use the class name as the data type, a set of empty square brackets and then the
array name (ClassName[] arrayName)
b) Copy the code that displays the array from the main method into the new displayInventory method.
c) Where you removed the display code from main replace it with a method call to the displayInventory method.
Remember to include the correct argument list to match the parameter list in the method you are calling.
d) Run and test your code
e) Create a static method in the ProductTester class after the end of the main method called addToInventory. This
method will not return any values and will accept the products array and the scanner as parameters.
f) Copy the code that adds the values to the array from the main method into the new addToInventory method.
g) To resolve the errors that you have in your code move the local variables required (tempNumber, tempName,
tempQty, tempPrice) from the main method into the top of the addInventory method.

h) Add a method call in main to the addToInventory() method where you removed the for loop from.
i) Run and test your code
j) Create a method in ProductTester that will return an integer value named getNumProducts() that accepts the
scanner as a parameter. Move all the code that gets the maximum number of products from the user into this
method, put a method call in main to your new method. You will store the returned value in your maxSize variable
so you will need to declare 2 of these, one in your main method and one in the getNumProducts() method. You
can remove the initial value of -1 from the declaration in main.
k) Run and test your code

3. Create two new methods in the Product class, one that will allow the user to add to the number of units in stock
(addToInventory), and one that will allow the user to deduct from the number of units in stock (deductFromInventory). Both
methods should accept a parameter (quantity) that holds the number of items to add/deduct. Place these under your
constructors.
4. Modify the ProductTester class so that the user can view, modify or discontinue the products through a user interface that is
based on a menu system.
a) Display a menu system that will display options and return the menu choice entered by the user.
i. The method should be called getMenuOption, return an integer value and take a Scanner object as
a parameter. Write the code under main.
ii. The menu should look like the following:
1. View Inventory
2. Add Stock
3. Deduct Stock
4. Discontinue Product
0. Exit
Please enter a menu option:
iii. Only numbers between 0 and 4 should be accepted, any other input should force a re-prompt to the
user. Remember when adding a try catch statement you have to initialise the variable to something
that will fail the while condition!
b) Create a method that will display the index value of the array and name of each product allowing the user to
select the product that they want to update (add/deduct).
i. The method should be called getProductNumber, return an integer value and take the products array
and a Scanner object as parameters. It should have a single local variable named

productChoice of
type integer that is initialized to -1. Write the code under main.
ii. A traditional FOR loop should be used to display the index value and the product name. Use the
length of the array to terminate the loop. The name for each product can be accessed through its
appropriate getter method.
iii. The user should only be allowed to enter values between 0 and 1 less than the length of the array.
All input should have appropriate error handling.
c) Create a method that will add stock values to each identified product.
i. The method should be called addInventory, have no return value and take the products array and a
Scanner object as parameters. It should have two local variables named productChoice that does
not need to be initialized and another named updateValue that should should be initialized to -1.
Both local variables should be able to store integer values. Write the code under main.
ii. Add a method call to the getProductNumber method passing the correct parameters and saving the
result in the productChoice variable.
iii. The user should be prompted with the message “How many products do you want to add?” and only
be allowed to enter positive values of 0 and above. All input should have both appropriate error
handling and error messages.
iv. Once a valid update value has been added then the selected product stock levels should be updated
through the addToInventory method that you created earlier. The productChoice variable is used to
identify the index value of the product in the array and the updateValue is the amount of stock to be
added.
d) Create a method that will deduct stock values to each identified product.
i. Follow the same procedure as you did to add stock but name your method deductInventory. The
restrictions on his input is that the value must be 0 or greater and cannot be greater than the current
quantity of stock for that product. All input should have both appropriate error handling and error
messages. Use the deductFromInventory method to make the change to the product object in the
array.

Here’s a detailed implementation guide to extend your Inventory Project, incorporating static methods, user interface improvements, and additional functionalities for managing inventory.

Step-by-Step Implementation Guide

1. Initial Setup
Make sure you have the basic `Product` and `ProductTester` classes created from the previous sections. Now, we'll modify these classes according to the new instructions.

2. Product Class
Add new methods to the `Product` class to handle inventory updates.

```java
package inventory;

public class Product {
    private String name;
    private int quantity;
    private double price;
    private int itemNumber;

    public Product(String name, int quantity, double price, int itemNumber) {
        this.name = name;
        this.quantity = quantity;
        this.price = price;
        this.itemNumber = itemNumber;
    }

    public String getName() {
        return name;
    }

    public int getQuantity() {
        return quantity;
    }

    public double getPrice() {
        return price;
    }

    public int getItemNumber() {
        return itemNumber;
    }

    public void addToInventory(int quantity) {
        this.quantity += quantity;
    }

    public void deductFromInventory(int quantity) {
```

```
        if (this.quantity >= quantity) {
            this.quantity -= quantity;
        } else {
            System.out.println("Not enough stock to deduct.");
        }
    }

    @Override
    public String toString() {
        return "Product [Name=" + name + ", Quantity=" + quantity + ", Price=" + price
+ ", Item Number=" + itemNumber + "]";
    }
}
```

3. ProductTester Class
Modify the `ProductTester` class to implement static methods, user interface improvements, and additional functionalities.

```java
package inventory;

import java.util.InputMismatchException;
import java.util.Scanner;

public class ProductTester {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int maxSize = getNumProducts(scanner);
        Product[] products = new Product[maxSize];

        addToInventory(products, scanner);

        int choice;
        do {
            choice = getMenuOption(scanner);
            switch (choice) {
                case 1:
                    displayInventory(products);
                    break;
                case 2:
                    addInventory(products, scanner);
                    break;
                case 3:
                    deductInventory(products, scanner);
                    break;
                case 4:
                    discontinueProduct(products, scanner);
                    break;
```

```
            case 0:
                System.out.println("Exiting...");
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
                break;
        }
    } while (choice != 0);

    scanner.close();
}

// Static methods
public static void displayInventory(Product[] products) {
    System.out.println("Product details:");
    for (Product product : products) {
        if (product != null) {
            System.out.println(product);
        }
    }
}

public static void addToInventory(Product[] products, Scanner scanner) {
    for (int i = 0; i < products.length; i++) {
        scanner.nextLine(); // clear the input buffer

        System.out.println("Enter details for product " + (i + 1) + ":");
        System.out.print("Name: ");
        String name = scanner.nextLine();
        System.out.print("Quantity: ");
        int quantity = scanner.nextInt();
        System.out.print("Price: ");
        double price = scanner.nextDouble();
        System.out.print("Item Number: ");
        int itemNumber = scanner.nextInt();

        products[i] = new Product(name, quantity, price, itemNumber);
    }
}

public static int getNumProducts(Scanner scanner) {
    int maxSize = -1;
    do {
        try {
            System.out.println("Enter the number of products you would like to add:");
            System.out.println("Enter 0 (zero) if you do not wish to add products");
            maxSize = scanner.nextInt();

            if (maxSize < 0) {
```

```
                System.out.println("Incorrect value entered. Please enter a positive
number or zero.");
            }
        } catch (InputMismatchException e) {
            System.out.println("Incorrect data type entered! Please enter a valid
number.");
            scanner.next(); // clear the input buffer
        }
    } while (maxSize < 0);

    return maxSize;
  }

  public static int getMenuOption(Scanner scanner) {
    int choice = -1;
    do {
      try {
        System.out.println("1. View Inventory");
        System.out.println("2. Add Stock");
        System.out.println("3. Deduct Stock");
        System.out.println("4. Discontinue Product");
        System.out.println("0. Exit");
        System.out.print("Please enter a menu option: ");
        choice = scanner.nextInt();

        if (choice < 0 || choice > 4) {
          System.out.println("Invalid choice. Please enter a number between 0 and
4.");
        }
      } catch (InputMismatchException e) {
        System.out.println("Incorrect data type entered! Please enter a valid
number.");
        scanner.next(); // clear the input buffer
      }
    } while (choice < 0 || choice > 4);

    return choice;
  }

  public static int getProductNumber(Product[] products, Scanner scanner) {
    int productChoice = -1;
    do {
      try {
        for (int i = 0; i < products.length; i++) {
          if (products[i] != null) {
            System.out.println(i + ": " + products[i].getName());
          }
        }
        System.out.print("Please enter the product number: ");
        productChoice = scanner.nextInt();
```

```
            if (productChoice < 0 || productChoice >= products.length ||
products[productChoice] == null) {
                System.out.println("Invalid product number. Please try again.");
                productChoice = -1;
            }
        } catch (InputMismatchException e) {
            System.out.println("Incorrect data type entered! Please enter a valid
number.");
            scanner.next(); // clear the input buffer
        }
    } while (productChoice == -1);

    return productChoice;
  }

  public static void addInventory(Product[] products, Scanner scanner) {
    int productChoice = getProductNumber(products, scanner);
    int updateValue = -1;

    do {
      try {
        System.out.print("How many products do you want to add? ");
        updateValue = scanner.nextInt();

        if (updateValue < 0) {
          System.out.println("Invalid value. Please enter a positive number.");
          updateValue = -1;
        }
      } catch (InputMismatchException e) {
        System.out.println("Incorrect data type entered! Please enter a valid
number.");
        scanner.next(); // clear the input buffer
      }
    } while (updateValue == -1);

    products[productChoice].addToInventory(updateValue);
    System.out.println("Stock added successfully.");
  }

  public static void deductInventory(Product[] products, Scanner scanner) {
    int productChoice = getProductNumber(products, scanner);
    int updateValue = -1;

    do {
      try {
        System.out.print("How many products do you want to deduct? ");
        updateValue = scanner.nextInt();
```

```
            if (updateValue < 0 || updateValue > products[productChoice].getQuantity())
{
                System.out.println("Invalid value. Please enter a number between 0 and "
+ products[productChoice].getQuantity() + ".");
                updateValue = -1;
            }
        } catch (InputMismatchException e) {
            System.out.println("Incorrect data type entered! Please enter a valid
number.");
            scanner.next(); // clear the input buffer
        }
    } while (updateValue == -1);

    products[productChoice].deductFromInventory(updateValue);
    System.out.println("Stock deducted successfully.");
  }

  public static void discontinueProduct(Product[] products, Scanner scanner) {
    int productChoice = getProductNumber(products, scanner);
    products[productChoice] = null;
    System.out.println("Product discontinued successfully.");
  }
}
```

Explanation of Steps
1. **Static Methods**: Implement static methods `displayInventory`, `addToInventory`, `getNumProducts`, `getMenuOption`, `getProductNumber`, `addInventory`, `deductInventory`, and `discontinueProduct` to handle various functionalities.
2. **Product Class**: Add methods `addToInventory` and `deductFromInventory` to update the inventory.
3. **Menu System**: Implement a menu system to allow users to interact with the inventory through options like viewing inventory, adding stock, deducting stock, and discontinuing a product.
4. **Error Handling**: Ensure proper error handling for invalid inputs.

Running and Testing
- Compile and run the `ProductTester` class.
- Test the program by interacting with the menu options and validating the inventory management functionalities.

This completes the implementation of an inventory management application with enhanced functionalities and user interface improvements.