MODULE-1

Java Fundamentals
Section 4: 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):
- Data types (Section 4.3)
- Creating classes/objects (Section 4.2)
- Instance variables/fields (Section 4.2)
- Constructors (Section 4.2)
- Methods (getters/accessors, setters/mutators) (Section 4.2)
- Overloading (Section 4.2)
- Main/tester classes (Section 4.2)
- toString() (Section 4.4)

TASKS:-

1. For the first part of the project you are required to think about what your inventory system will store.
a. Think of specific products that lend themselves to be stored in an inventory (for example, products in your
home, school, or workplace: they could be from the following categories; office supplies, music CDs, DVD
movies, or software). Write a list of at least 6 products that you want to store in your system, this project could
be used to store a wide range of products.
b. For each of the products that you identified, complete the following table:
Attribute Sample Data
Name of the product (the value that will
identify the product in your system).
Price (this value holds the price that each
item will be sold for).

Number of units in stock (this value will store
how many of each product item is currently
in stock).
Item number (used to uniquely identify the
product in your system).
This table gives you an understanding of the type of data that you will want to store for the attributes of each
product. It’s useful to do this so you have a clear understanding of the data that you will be working with!

ans)

a. List of Products

Here's a list of six products that can be stored in the inventory system:

1. **Wireless Mouse**
2. **External Hard Drive**
3. **Office Chair**
4. **Bluetooth Headphones**
5. **USB Flash Drive**
6. **Desk Lamp**

b. Table of Attributes and Data Types

To complete the table with attributes for each product, including sample data and data types, here’s how it looks:

Attribute	Product Name	Price	Number of Units in Stock	Item Number	Data Type
Name of the product	Wireless Mouse	25.99	50	2001	`string`
Price	25.99				`double`
Number of units in stock	50				`int`
Item number	2001				`int`
Name of the product	External Hard Drive	89.99	20	2002	`string`
Price	89.99				`double`
Number of units in stock	20				`int`
Item number	2002				`int`
Name of the product	Office Chair	120.50	15	2003	`string`
Price	120.50				`double`
Number of units in stock	15				`int`
Item number	2003				`int`
Name of the product	Bluetooth Headphones	45.75	35	2004	`string`
Price	45.75				`double`
Number of units in stock	35				`int`
Item number	2004				`int`
Name of the product	USB Flash Drive	15.50	60	2005	`string`
Price	15.50				`double`
Number of units in stock	60				`int`
Item number	2005				`int`
Name of the product	Desk Lamp	32.00	40	2006	`string`
Price	32.00				`double`
Number of units in stock	40				`int`
Item number	2006				`int`

2. The next step is to think about the correct data types that you will use to store the values in your system. To do this add
another column to your table that will identify the correct data type for each value that you have identified.

Attribute	Sample Data	Data Type
Name of the product	Wireless Mouse	`String`
Price	25.99	`double`
Number of units in stock	50	`int`
Item number	2001	`int`

Explanation of Data Types

- **Name of the product**:
  - **Data Type**: String
  - **Reason**: Product names are textual and can vary in length. The String type is used for sequences of characters.
- **Price**:
  - **Data Type**: double
  - **Reason**: Prices can have fractional values (e.g., 25.99). The double type is suitable for representing floating-point numbers with precision.
- **Number of units in stock**:
  - **Data Type**: int
  - **Reason**: The number of units is a whole number and does not require decimal places. The int type is used for integer values.
- **Item number**:
  - **Data Type**: int
  - **Reason**: Item numbers are unique identifiers for products and are typically whole numbers. The int type is appropriate for such values.

3. Create a Project Named inventory

In your Integrated Development Environment (IDE), create a new Java project named inventory.

4. Create the Product Class

Inside the inventory project, create a new Java class named Product.

5. Add Private Instance Fields

In the Product class, declare the private instance fields using the data types identified:

package inventory;

```
/**
 * Represents a product in the inventory system.
 */
public class Product {
    // Instance field declarations
    private int itemNumber;
    private String name;
    private int quantityInStock;
    private double price;

    // Constructors, getters, setters, and toString() will be added here
}
```

6. Add a Comment Above the Instance Field Declarations

The comment has already been added above the instance field declarations as specified in the code above.

7. Create Constructors

a. Default Constructor

Create a default constructor that initializes the fields to their default values:

```
/**
 * Default constructor initializes fields to default values.
 */
public Product() {
    this.itemNumber = 0;
    this.name = "";
    this.quantityInStock = 0;
    this.price = 0.0;
}
```

Explanation: The default constructor sets all instance variables to their default values: 0 for integers, 0.0 for doubles, and an empty string for the name.

b. Parameterized Constructor

Create a parameterized constructor that initializes fields with specific values:

java

```
/**
 * Parameterized constructor initializes fields with provided values.
 * @param number Item number of the product.
 * @param name Name of the product.
 * @param qty Quantity in stock.
 * @param price Price of the product.
 */
```

```
public Product(int number, String name, int qty, double price) {
   this.itemNumber = number;
   this.name = name;
   this.quantityInStock = qty;
   this.price = price;
}
```

8.Write getter/accessor and setter/mutator methods for each of the four instance variables. Write getter/accessor and setter/mutator methods for each of the four instance variables. Add comments above them to explain their purpose.

```
public class Product {
   // Instance variables
   private int itemNumber;
   private String name;
   private int quantityInStock;
   private double price;

   // Default constructor
   public Product() {
      this.itemNumber = 0;
      this.name = "Unknown";
      this.quantityInStock = 0;
      this.price = 0.0;
   }

   // Parameterized constructor
   public Product(int itemNumber, String name, int quantityInStock, double
price) {
      this.itemNumber = itemNumber;
      this.name = name;
      this.quantityInStock = quantityInStock;
      this.price = price;
   }

   // Getter for itemNumber
   /**
    * Gets the item number of the product.
    * @return the item number
```

```
 */
public int getItemNumber() {
    return itemNumber;
}

// Setter for itemNumber
/**
 * Sets the item number of the product.
 * @param itemNumber the item number to set
 */
public void setItemNumber(int itemNumber) {
    this.itemNumber = itemNumber;
}

// Getter for name
/**
 * Gets the name of the product.
 * @return the name of the product
 */
public String getName() {
    return name;
}

// Setter for name
/**
 * Sets the name of the product.
 * @param name the name to set
 */
public void setName(String name) {
    this.name = name;
}

// Getter for quantityInStock
/**
 * Gets the quantity in stock of the product.
 * @return the quantity in stock
 */
public int getQuantityInStock() {
    return quantityInStock;
}
```

```
    // Setter for quantityInStock
    /**
     * Sets the quantity in stock of the product.
     * @param quantityInStock the quantity in stock to set
     */
    public void setQuantityInStock(int quantityInStock) {
        this.quantityInStock = quantityInStock;
    }

    // Getter for price
    /**
     * Gets the price of the product.
     * @return the price of the product
     */
    public double getPrice() {
        return price;
    }

    // Setter for price
    /**
     * Sets the price of the product.
     * @param price the price to set
     */
    public void setPrice(double price) {
        this.price = price;
    }

    // Override toString method
    @Override
    public String toString() {
        return "Item Number : " + itemNumber + "\n" +
               "Name : " + name + "\n" +
               "Quantity in stock: " + quantityInStock + "\n" +
               "Price : " + price;
    }
}
```

Explanation of Methods

1. **Getter Methods**:

- **getItemNumber()**: Returns the item number of the product.
- **getName()**: Returns the name of the product.
- **getQuantityInStock()**: Returns the quantity of the product in stock.
- **getPrice()**: Returns the price of the product.

2. **Setter Methods**:
   - **setItemNumber(int itemNumber)**: Sets the item number of the product.
   - **setName(String name)**: Sets the name of the product.
   - **setQuantityInStock(int quantityInStock)**: Sets the quantity of the product in stock.
   - **setPrice(double price)**: Sets the price of the product.

9. Override the toString() method from the object class to show a description of each Product object that includes the instance field values in the following format:
Item Number : 1
Name : Greatest Hits
Quantity in stock: 25
Price : 9.99

```
public class Product {
   // Instance variables
   private int itemNumber;
   private String name;
   private int quantityInStock;
   private double price;

   // Default constructor
   public Product() {
      this.itemNumber = 0;
      this.name = "Unknown";
      this.quantityInStock = 0;
      this.price = 0.0;
   }

   // Parameterized constructor
   public Product(int itemNumber, String name, int quantityInStock,
double price) {
```

```
    this.itemNumber = itemNumber;
    this.name = name;
    this.quantityInStock = quantityInStock;
    this.price = price;
}

// Getter for itemNumber
public int getItemNumber() {
    return itemNumber;
}

// Setter for itemNumber
public void setItemNumber(int itemNumber) {
    this.itemNumber = itemNumber;
}

// Getter for name
public String getName() {
    return name;
}

// Setter for name
public void setName(String name) {
    this.name = name;
}

// Getter for quantityInStock
public int getQuantityInStock() {
    return quantityInStock;
}

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

// Getter for price
public double getPrice() {
    return price;
}
```

```
    // Setter for price
    public void setPrice(double price) {
        this.price = price;
    }

    // Override toString method
    @Override
    public String toString() {
        return String.format("Item Number : %d\n" +
                             "Name : %s\n" +
                             "Quantity in stock: %d\n" +
                             "Price : %.2f",
                             itemNumber, name, quantityInStock, price);
    }
}
```

10. Create a Java main class called ProductTester

```
public class ProductTester {
    public static void main(String[] args) {
        // Create and initialize six Product objects

        // Two products using default constructor
        Product product1 = new Product();
        Product product2 = new Product();

        // Four products using parameterized constructor
        Product product3 = new Product(1, "Greatest Hits", 25, 9.99);
        Product product4 = new Product(2, "Summer Special", 50, 12.49);
        Product product5 = new Product(3, "Winter Collection", 15, 19.99);
        Product product6 = new Product(4, "Spring Trends", 30, 14.99);

        // Display the details of each product
        System.out.println("Product 1 Details:");
        System.out.println(product1);
        System.out.println();

        System.out.println("Product 2 Details:");
        System.out.println(product2);
```

```
        System.out.println();

        System.out.println("Product 3 Details:");
        System.out.println(product3);
        System.out.println();

        System.out.println("Product 4 Details:");
        System.out.println(product4);
        System.out.println();

        System.out.println("Product 5 Details:");
        System.out.println(product5);
        System.out.println();

        System.out.println("Product 6 Details:");
        System.out.println(product6);
    }
}
```

11. Create and initialize six Product objects based on the list that you created in task 1.
a. Two of the Products should be created using the default constructor.
b. The other four should be created by providing values for the arguments that match the parameters of the constructor

```
public class ProductTester {
    public static void main(String[] args) {
        // Create and initialize six Product objects

        // a. Two Products using the default constructor
        Product product1 = new Product(); // Default values
        Product product2 = new Product(); // Default values

        // b. Four Products using the parameterized constructor
        Product product3 = new Product(1, "Greatest Hits", 25, 9.99);
        Product product4 = new Product(2, "Summer Special", 50, 12.49);
        Product product5 = new Product(3, "Winter Collection", 15, 19.99);
        Product product6 = new Product(4, "Spring Trends", 30, 14.99);

        // Display the details of each product
```

```
        System.out.println("Product 1 Details:");
        System.out.println(product1);
        System.out.println();

        System.out.println("Product 2 Details:");
        System.out.println(product2);
        System.out.println();

        System.out.println("Product 3 Details:");
        System.out.println(product3);
        System.out.println();

        System.out.println("Product 4 Details:");
        System.out.println(product4);
        System.out.println();

        System.out.println("Product 5 Details:");
        System.out.println(product5);
        System.out.println();

        System.out.println("Product 6 Details:");
        System.out.println(product6);
    }
}
```

12. Using the ProductTester class, display the details of each product to the console

```
public class ProductTester {
    public static void main(String[] args) {
        // Create and initialize six Product objects

        // Two Products using the default constructor
        Product product1 = new Product(); // Default values
        Product product2 = new Product(); // Default values

        // Four Products using the parameterized constructor
        Product product3 = new Product(1, "Greatest Hits", 25, 9.99);
        Product product4 = new Product(2, "Summer Special", 50, 12.49);
        Product product5 = new Product(3, "Winter Collection", 15, 19.99);
        Product product6 = new Product(4, "Spring Trends", 30, 14.99);
```

```
        // Display the details of each product
        System.out.println("Product 1 Details:");
        System.out.println(product1); // Calls product1.toString()
        System.out.println();

        System.out.println("Product 2 Details:");
        System.out.println(product2); // Calls product2.toString()
        System.out.println();

        System.out.println("Product 3 Details:");
        System.out.println(product3); // Calls product3.toString()
        System.out.println();

        System.out.println("Product 4 Details:");
        System.out.println(product4); // Calls product4.toString()
        System.out.println();

        System.out.println("Product 5 Details:");
        System.out.println(product5); // Calls product5.toString()
        System.out.println();

        System.out.println("Product 6 Details:");
        System.out.println(product6); // Calls product6.toString()
    }
}
```

13. Save your project.

1. **Open Your Project**:
    - Open IntelliJ IDEA and load your project.
2. **Verify Project Structure**:
    - Ensure that your source files (Product.java and ProductTester.java) are located in the src folder.
3. **Compile**:
    - IntelliJ IDEA usually compiles your code automatically. You can manually compile by going to Build -> Build Project or by pressing Ctrl + F9 (Windows/Linux) or Cmd + F9 (Mac).
4. **Run**:

- Right-click on the ProductTester class file in the Project view and select Run 'ProductTester.main()'.

5. **Save**:
   - IntelliJ IDEA saves your files automatically, but you can manually save by pressing Ctrl + S (Windows/Linux) or Cmd + S (Mac).
6. **Backup**:
   - Use IntelliJ's version control integration (like Git) to commit your changes and keep backups.

Using Eclipse

1. **Open Your Project**:
   - Open Eclipse and load your project.
2. **Verify Project Structure**:
   - Ensure that your source files (Product.java and ProductTester.java) are in the src folder.
3. **Compile**:
   - Eclipse compiles your code automatically. You can manually compile by selecting Project -> Build Project if auto-build is disabled.
4. **Run**:
   - Right-click on the ProductTester class file in the Package Explorer and select Run As -> Java Application.
5. **Save**:
   - Eclipse saves your files automatically, but you can manually save by pressing Ctrl + S (Windows/Linux) or Cmd + S (Mac).
6. **Backup**:
   - Use Eclipse's version control integration (like Git) to commit your changes and create backups.

Final Note

Ensure that all your files are saved and that the compiled classes are up to date. If you're using version control (like Git), commit your changes with a descriptive message to keep a history of your work. For a more robust backup, consider using cloud storage services or version control systems to keep track of your project's history and changes.

Java Fundamentals
Section 5: 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):
• Scanner/Keyboard Input (Section 5.1)
• Boolean fields and ternary operators (Section 5.1)
Instructions:
1. Open the inventory program that was created in Section 4: Creating an inventory Project.
o open the inventory program that you created in Section 4, follow these steps:

Step 1: Open Your IDE

Launch the Java Integrated Development Environment (IDE) you used to create the project in Section 4. Common options include IntelliJ IDEA, Eclipse, or NetBeans.

Step 2: Locate the Project

1. **In IntelliJ IDEA:**
    - Go to **File > Open** and navigate to the directory where your project is saved.
    - Select the project folder and click **OK** to open it.
2. **In Eclipse:**
    - Go to **File > Open Projects from File System...**.
    - Browse to the location of your project and select the project folder.
    - Click **Finish** to open the project.
3. **In NetBeans:**
    - Go to **File > Open Project**.
    - Browse to the project directory and select the project.
    - Click **Open Project**.

Step 3: Verify Your Project Structure

Once the project is open, verify that the structure is intact:

- The **src** folder should contain your Java source files.
- There should be a package called **inventory**, inside which your Java classes are organized.

Step 4: Open the Main Class

- Open the main class (likely named **Inventory** or similar) to start working on the next section.

If you're encountering issues or need help with specific steps, feel free to ask!

2. Modify the ProductTester class.
a. Add a Scanner called in to the beginning of your main method.
b. Create local variables that will store values for the for each of the attributes of the Product class. Name the variables
tempNumber, tempName, tempQty and tempPrice.
c. Ask the user to input values for the for each of the attributes of the Product class. Ask for the name, quantity, price
and item number, store the values in temporary local variables that you have just created.
d. Use the values that were entered by the user to create the p1 object. This means that you will be using the
constructor that takes 4 parameters instead of the default constructor

1. Add a Scanner to Your main Method

First, you need to add a **Scanner** object to take user input.

2. Create Local Variables for Product Attributes

Create temporary local variables named **tempNumber**, **tempName**, **tempQty**, and **tempPrice** to store the product attributes.

3. Ask the User for Input

Use the **Scanner** to prompt the user to input values for each of these attributes.

4. Create the p1 Object Using the Constructor

Here's what the modified **ProductTester** class might look like:

java

```
import java.util.Scanner;

public class ProductTester {
   public static void main(String[] args) {
      // a. Add a Scanner called in to the beginning of your main method.
      Scanner in = new Scanner(System.in);

      // b. Create local variables to store the product attributes.
      int tempNumber;
      String tempName;
      int tempQty;
      double tempPrice;

      // c. Ask the user to input values for each of the attributes.
      System.out.print("Enter the product item number: ");
      tempNumber = in.nextInt();

      in.nextLine();  // Consume newline left-over

      System.out.print("Enter the product name: ");
      tempName = in.nextLine();

      System.out.print("Enter the product quantity: ");
      tempQty = in.nextInt();

      System.out.print("Enter the product price: ");
      tempPrice = in.nextDouble();

      // d. Use the values entered by the user to create the p1 object.
      Product p1 = new Product(tempNumber, tempName, tempQty, tempPrice);
```

```
        // Display the product information to verify.
        System.out.println("Product created: " + p1);

        // Close the scanner
        in.close();
    }
}
```

3. You are going to get the user to provide you with values for p2.
a. Use the same local variables as before to get input from the user to create the p2 object. Copy and paste
the code after the line that makes the p1 object.
b. Run the program and identify where an error has occurred.
c. The program doesn't appear to ask you for a value for name. This is because the last value entered was a
numeric value and it has left some special characters in the input buffer. To clear the input buffer, add the
following statement before you ask for any values for p2:
in.nextLine();
This takes in any values stored in the buffer and discards them leaving an empty buffer.
d. Run the program now, it should be error free and display all the values including the user entered ones in
the console.
e. Close the Scanner object when you are finished with it.

1. Copy and Paste the Code to Get Input for p2
Use the same local variables (**tempNumber**, **tempName**, **tempQty**, and **tempPrice**) to get input for **p2**. Copy the input code you used for **p1** and paste it after the creation of the **p1** object.
2. Add the in.nextLine(); to Clear the Buffer
Add the **in.nextLine();** statement before asking for the values for **p2** to clear the input buffer.
3. Close the Scanner Object at the End
Make sure to close the **Scanner** object when you're done.
Updated ProductTester Class:

```
import java.util.Scanner;

public class ProductTester {
    public static void main(String[] args) {
        // a. Add a Scanner called in to the beginning of your main method.
        Scanner in = new Scanner(System.in);

        // b. Create local variables to store the product attributes.
        int tempNumber;
        String tempName;
        int tempQty;
        double tempPrice;

        // c. Ask the user to input values for each of the attributes for p1.
        System.out.print("Enter the product item number for p1: ");
        tempNumber = in.nextInt();

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

```
        System.out.print("Enter the product name for p1: ");
        tempName = in.nextLine();

        System.out.print("Enter the product quantity for p1: ");
        tempQty = in.nextInt();

        System.out.print("Enter the product price for p1: ");
        tempPrice = in.nextDouble();

        // Create the p1 object using the values entered by the user.
        Product p1 = new Product(tempNumber, tempName, tempQty, tempPrice);

        // Display the product information to verify.
        System.out.println("Product p1 created: " + p1);

        // b. Clear the input buffer before asking for p2 values.
        in.nextLine();

        // Ask the user to input values for each of the attributes for p2.
        System.out.print("Enter the product item number for p2: ");
        tempNumber = in.nextInt();

        in.nextLine();  // Consume newline left-over

        System.out.print("Enter the product name for p2: ");
        tempName = in.nextLine();

        System.out.print("Enter the product quantity for p2: ");
        tempQty = in.nextInt();
h
        System.out.print("Enter the product price for p2: ");
        tempPrice = in.nextDouble();

        // Create the p2 object using the values entered by the user.
        Product p2 = new Product(tempNumber, tempName, tempQty, tempPrice);

        // Display the product information to verify.
        System.out.println("Product p2 created: " + p2);

        // e. Close the Scanner object when you are finished with it.
        in.close();
    }
}
```

4.You want to be able to mark your products as active or discontinued. If a product is discontinued it means that the
remaining stock will be the last of it, and no more orders are to be made.

a. Add a Boolean instance field to your Product class called active that has a default value of true.

b. Create getter/setter methods for this new field.

c. Add the value of this new field to the toString() method so that the output matches the following:

Item Number : 1

Name : Greatest Hits
Quantity in stock: 25
Price : 9.99
Product Status : true.

To implement this new functionality in your **Product** class, follow these steps:

1. Add a Boolean Instance Field

Add a Boolean instance field called **active** to your **Product** class. Set its default value to **true**.

2. Create Getter/Setter Methods

Create getter and setter methods for the **active** field to allow other classes to access and modify this value.

3. Modify the toString() Method

Update the **toString()** method to include the value of the **active** field in the output.

```
public class Product {
    private int itemNumber;
    private String name;
    private int quantity;
    private double price;
    private boolean active = true;  // a. Add a Boolean instance field with a default value of true.

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

    // b. Create getter/setter methods for the active field.
    public boolean isActive() {
        return active;
    }

    public void setActive(boolean active) {
        this.active = active;
    }

    // c. Add the value of this new field to the toString() method.
    @Override
    public String toString() {
        return "Item Number: " + itemNumber +
                "\nName: " + name +
                "\nQuantity in stock: " + quantity +
                "\nPrice: $" + price +
```

```
            "\nProduct Status: " + active;
    }
}
```

5. When you run the code, you get back a printed value for active as either true or false. This is not user friendly and
would be better if the output stated either Active (true) or Discontinued (false). Add a ternary operator in your
toString() method to achieve this

To make the output more user-friendly by displaying "Active" or "Discontinued" instead of **true** or **false**, you can use a ternary operator in the **toString()** method. Here's how you can modify the method:

```
@Override
public String toString() {
    return "Item Number: " + itemNumber +
        "\nName: " + name +
        "\nQuantity in stock: " + quantity +
        "\nPrice: $" + price +
        "\nProduct Status: " + (active ? "Active" : "Discontinued");
}
```

6. Call the setter from the driver class and set the active value to false for the p6 object before you display the values
to screen. Run and test your code.

o set the **active** value to **false** for the **p6** object in your driver class (which might be the **ProductTester** class), you would typically call the setter method for the **active** attribute. Here's how you can do it:

1. **Ensure You Have a Setter Method**: First, confirm that your **Product** class has a setter method for the **active** attribute. It might look something like this:

```
public class ProductTester {
    public static void main(String[] args) {
        // Assuming p6 is already created, for example:
        Product p6 = new Product("DVD", "The Matrix", 19.99, true);

        // Set the active value to false using the setter method
        p6.setActive(false);

        // Display the values to the screen (assuming a toString method in Product)
        System.out.println(p6.toString());

        // If you do not have a toString method, you can manually print the attributes
        System.out.println("Product Name: " + p6.getName());
        System.out.println("Price: $" + p6.getPrice());
        System.out.println("Active: " + p6.isActive());
    }
}
```

7. Create a method in the Product class that will return the inventory value for each item. Use the product price
multiplied by the quantity of stock to calculate the inventory value. Do not use any local variables in this method
simply return the value in a single line of code

Here's how you can create a method in the **Product** class that returns the inventory value using a single line of code:

1. Add the Method to the Product Class:

Assuming you have **price** and **quantity** as attributes in your **Product** class, the method could look like this:

```
public class Product {
   private String name;
   private String type;
   private double price;
   private int quantity;
   private boolean active;

   // Constructor, getters, and other methods...

   // Method to calculate and return the inventory value
   public double getInventoryValue() {
      return price * quantity;
   }
}
```

8. Update the toString() method in the Product class to include a method call to the getInventoryValue() method that you
have just created so that the output is as follows:

Item Number : 1
Name : Greatest Hits
Quantity in stock: 25
Price : 9.99
Stock Value : 249.75
Product Status : true

To update the **toString()** method in the **Product** class so that it includes the inventory value, you can modify the method as follows:

1. Update the toString() Method:

In your **Product** class, the **toString()** method should be updated to include a call to the **getInventoryValue()** method:

```
@Override
public String toString() {
   return "Item Number: " + 1 + "\n" +  // Assuming you have an item number
         "Name: " + name + "\n" +
         "Quantity in stock: " + quantity + "\n" +
         "Price: $" + price + "\n" +
         "Stock Value: $" + getInventoryValue() + "\n" +
         "Product Status: " + active;
}
```

Java Fundamentals
Section 6: 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):
• Using loops (Sections 5.2 and 6.1)
• Handling exceptions (Section 6.2)
• Using if statements (Section 5.1)
• Arrays of objects (Section 6.1)

Instructions:
1. Open the inventory program that was updated in Section 5: Creating an inventory Project.

1. **Open IntelliJ IDEA**:
    - Launch IntelliJ IDEA from your desktop or start menu.
2. **Open Your Project**:
    - Go to File -> Open... (or Open Project if it's your first time).
    - Navigate to the directory where your project is located.
    - Select the project folder (it should contain a file like pom.xml for Maven projects or build.gradle for Gradle projects, or just the source files if it's a plain Java project).
    - Click Open or OK.
3. **Verify Project Structure**:
    - Ensure your project structure is correct in the Project view. You should see a package named inventory containing your Java files (Product.java, ProductTester.java, etc.).
4. **Explore Files**:
    - You can browse and edit your Java files in the Project view or by opening them directly in the editor.

Using Eclipse

1. **Open Eclipse**:
    - Launch Eclipse from your desktop or start menu.
2. **Open Your Project**:
    - Go to File -> Open Projects from File System... or Import... -> General -> Existing Projects into Workspace.
    - Navigate to the root directory of your project.
    - If using Import..., select the project and click Finish. If using Open Projects from File System..., ensure the Directory field points to your project folder and click Finish.
3. **Verify Project Structure**:
    - Check the Package Explorer or Project Explorer view to ensure your project is loaded correctly. You should see the inventory package with your source files.
4. **Explore Files**:

- You can open and edit files by double-clicking on them in the Package Explorer.

2.Ask the user to enter the number of products they wish to add. Accept a positive integer for the number of products and
handle the value of zero.
a. Create a variable named maxSize that can store integers.
b. Create a prompt at the beginning of your main method that will instruct the user to enter the required value for the
number of products they wish to store:
Enter the number of products you would like to add
Enter 0 (zero) if you do not wish to add products
c. Use a do while loop so that the program will not continue until a valid positive value is entered. If a value less than
zero is entered an error message stating “Incorrect Value entered” should be displayed before the user is re-prompted
to enter a new value. You should not leave the loop until a value of zero or greater is entered

 Declare a Variable:

- Create an integer variable named maxSize to store the number of products.

 Prompt the User:

- Provide a clear message to the user explaining how to enter the number of products, including how to enter 0 if they do not wish to add any products.

 Use a do-while Loop:

- Ensure that the loop continues prompting the user until a valid input (0 or a positive integer) is entered.
- Display an error message if the input is less than zero and prompt the user to re-enter the value.

```
package inventory;


import java.util.InputMismatchException;

import java.util.Scanner;


public class InventoryApplication {

    public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);
        int maxSize = -1; // Initialize to an invalid value

        // Prompt the user for the number of products
        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");

        // Using a do-while loop to ensure valid input
        do {
            try {
                System.out.print("Number of products: ");
                maxSize = scanner.nextInt();

                // Validate input
                if (maxSize < 0) {
                    System.out.println("Incorrect value entered. Please enter a non-negative integer.");
                }
            } catch (InputMismatchException e) {
                System.out.println("Invalid input. Please enter a numeric value.");
                scanner.next(); // Clear the invalid input
            }
        } while (maxSize < 0); // Continue looping until a non-negative value is entered

        // Now you can use maxSize to proceed with adding products
        if (maxSize == 0) {
```

```
            System.out.println("No products will be added.");
        } else {
            System.out.println("You will be adding " + maxSize + " products.");
            // Proceed with further logic to add products
        }

        // Close the scanner
        scanner.close();
    }
}
```

You are now going to add some error handling to deal with run-time errors in your code. Currently your program deals with
numbers entered outside the given range but cannot handle incorrect data type entries.
a. Add a try block that surrounds all of the code inside the do while loop.
b. Add a catch statement above the while that will take an Exception e parameter. The program should use a console
output statement to display the value of e to screen.
c. As you now assign a value for maxSize inside a try statement there is the possibility that maxSize will not have ben
assigned a value when you get to the while clause. To ensure this does not happen assign an initial value of -1 to
maxSize when it is declared.
HINT: Always assign a value that will fail the loop so that your code is forced to assign a correct value before it

 2
continues.
d. Run and test your code by entering a character instead of a number.
e. Add a line of code in your catch statement that will clear out the input buffer so that the prompt will be displayed, and
the system will wait for user input.
f. Take a note of the specific type of Exception produced when you enter a character and create a catch statement just
for that exception. This error should display an Incorrect data type entered! message to the console and

should also clear the input buffer.
g. Run and test your code by entering a vriety of different input values

```
package inventory;


import java.util.InputMismatchException;

import java.util.Scanner;


public class InventoryApplication {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int maxSize = -1; // Initialize to an invalid value


        // Prompt the user for the number of products

        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");


        // Using a do-while loop to ensure valid input

        do {

            try {

                System.out.print("Number of products: ");

                maxSize = scanner.nextInt(); // Read user input


                // Validate input

                if (maxSize < 0) {

                    System.out.println("Incorrect value entered. Please enter a non-negative
integer.");
```

```
            }
        } catch (InputMismatchException e) {
            System.out.println("Incorrect data type entered! Please enter a numeric value.");
            scanner.next(); // Clear the invalid input
        } catch (Exception e) {
            System.out.println("An error occurred: " + e.getMessage());
            scanner.next(); // Clear the invalid input
        }
    } while (maxSize < 0); // Continue looping until a non-negative value is entered

    // Now you can use maxSize to proceed with adding products
    if (maxSize == 0) {
        System.out.println("No products will be added.");
    } else {
        System.out.println("You will be adding " + maxSize + " products.");
        // Proceed with further logic to add products
    }

    // Close the scanner
    scanner.close();
  }
}
```

4.Modify the ProductTester class to handle multiple products using a single dimensional array if a value greater than zero is
entered.
a. Create an if statement that will display the message “No products required!” to the console if the value of maxSize is
zero.

b. Add an Else statement to deal with any value other than zero. Create a single one-dimension array named products
based on the Product class that will have the number of elements specified by the user in the maxSize variable

```
package inventory;


import java.util.Scanner;


public class ProductTester {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int maxSize = -1; // Initialize to an invalid value


        // Prompt the user for the number of products

        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");


        // Using a do-while loop to ensure valid input

        do {

            try {

                System.out.print("Number of products: ");

                maxSize = scanner.nextInt(); // Read user input


                // Validate input

                if (maxSize < 0) {

                    System.out.println("Incorrect value entered. Please enter a non-negative integer.");
```

```
            }
        } catch (InputMismatchException e) {
            System.out.println("Incorrect data type entered! Please enter a numeric value.");
            scanner.next(); // Clear the invalid input
        } catch (Exception e) {
            System.out.println("An error occurred: " + e.getMessage());
            scanner.next(); // Clear the invalid input
        }
    } while (maxSize < 0); // Continue looping until a non-negative value is entered

    // Handle based on the value of maxSize
    if (maxSize == 0) {
        System.out.println("No products required!");
    } else {
        // Create a Product array with size maxSize
        Product[] products = new Product[maxSize];
        System.out.println("You will be adding " + maxSize + " products.");

        // Initialize the Product array
        for (int i = 0; i < products.length; i++) {
            System.out.println("Enter details for product " + (i + 1) + ":");

            // Get product details from the user
            System.out.print("Enter item number: ");
            int itemNumber = scanner.nextInt();
```

```
        scanner.nextLine(); // Consume newline

        System.out.print("Enter product name: ");
        String name = scanner.nextLine();

        System.out.print("Enter quantity in stock: ");
        int quantity = scanner.nextInt();

        System.out.print("Enter price: ");
        double price = scanner.nextDouble();
        scanner.nextLine(); // Consume newline

        // Create a new Product object and add it to the array
        products[i] = new Product(itemNumber, name, quantity, price);
    }

    // Display the details of all products
    System.out.println("Products added:");
    for (Product product : products) {
        System.out.println(product);
        System.out.println();
    }
}

// Close the scanner
```

```
        scanner.close();
    }
}
```

5.You are now going to populate the array, getting the values from the user for each field in a product object.
a. Inside the else statement under where you created the array write a for loop that will iterate through the array from zero
to 1 less than maxSize.
b. As the last input you received from the user was numeric you will need to add a statement that clears the input buffer
as the first line in your for loop.
c. Copy the code that you used to get input from the user for all a products fields into the for loop. This includes the
name, quantity, price and item number.
d. Add a new product object into the array using the index value for the position and the constructer that takes 4
parameters

```
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 = -1; // Initialize to an invalid value


        // Prompt the user for the number of products

        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");
```

```
        // Using a do-while loop to ensure valid input
        do {
            try {
                System.out.print("Number of products: ");
                maxSize = scanner.nextInt(); // Read user input

                // Validate input
                if (maxSize < 0) {
                    System.out.println("Incorrect value entered. Please enter a non-negative integer.");
                }
            } catch (InputMismatchException e) {
                System.out.println("Incorrect data type entered! Please enter a numeric value.");
                scanner.next(); // Clear the invalid input
            } catch (Exception e) {
                System.out.println("An error occurred: " + e.getMessage());
                scanner.next(); // Clear the invalid input
            }
        } while (maxSize < 0); // Continue looping until a non-negative value is entered

        // Handle based on the value of maxSize
        if (maxSize == 0) {
            System.out.println("No products required!");
        } else {
            // Create a Product array with size maxSize
            Product[] products = new Product[maxSize];
```

```
System.out.println("You will be adding " + maxSize + " products.");

// Populate the Product array
for (int i = 0; i < products.length; i++) {
    // Clear input buffer if necessary
    scanner.nextLine(); // Consume any leftover newline characters

    System.out.println("Enter details for product " + (i + 1) + ":");

    System.out.print("Enter item number: ");
    int itemNumber = scanner.nextInt();

    System.out.print("Enter product name: ");
    scanner.nextLine(); // Consume the newline left by nextInt()
    String name = scanner.nextLine();

    System.out.print("Enter quantity in stock: ");
    int quantity = scanner.nextInt();

    System.out.print("Enter price: ");
    double price = scanner.nextDouble();

    // Create a new Product object and add it to the array
    products[i] = new Product(itemNumber, name, quantity, price);
}
```

```
        // Display the details of all products
        System.out.println("Products added:");
        for (Product product : products) {
            System.out.println(product);
            System.out.println();
        }
    }

    // Close the scanner
    scanner.close();
  }
}
```

6.Use a for each loop to display the information for each individual product in the products array.

```
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 = -1; // Initialize to an invalid value

    // Prompt the user for the number of products
```

```
        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");

        // Using a do-while loop to ensure valid input
        do {
            try {
                System.out.print("Number of products: ");
                maxSize = scanner.nextInt(); // Read user input

                // Validate input
                if (maxSize < 0) {
                    System.out.println("Incorrect value entered. Please enter a non-negative integer.");
                }
            } catch (InputMismatchException e) {
                System.out.println("Incorrect data type entered! Please enter a numeric value.");
                scanner.next(); // Clear the invalid input
            } catch (Exception e) {
                System.out.println("An error occurred: " + e.getMessage());
                scanner.next(); // Clear the invalid input
            }
        } while (maxSize < 0); // Continue looping until a non-negative value is entered

        // Handle based on the value of maxSize
        if (maxSize == 0) {
            System.out.println("No products required!");
```

```
} else {
    // Create a Product array with size maxSize
    Product[] products = new Product[maxSize];
    System.out.println("You will be adding " + maxSize + " products.");

    // Populate the Product array
    for (int i = 0; i < products.length; i++) {
        // Clear input buffer if necessary
        scanner.nextLine(); // Consume any leftover newline characters

        System.out.println("Enter details for product " + (i + 1) + ":");

        System.out.print("Enter item number: ");
        int itemNumber = scanner.nextInt();

        System.out.print("Enter product name: ");
        scanner.nextLine(); // Consume the newline left by nextInt()
        String name = scanner.nextLine();

        System.out.print("Enter quantity in stock: ");
        int quantity = scanner.nextInt();

        System.out.print("Enter price: ");
        double price = scanner.nextDouble();
```

```
            // Create a new Product object and add it to the array
            products[i] = new Product(itemNumber, name, quantity, price);
        }

        // Display the details of all products using a for-each loop
        System.out.println("Products added:");
        for (Product product : products) {
            System.out.println(product);
            System.out.println();
        }
    }

    // Close the scanner
    scanner.close();
  }
}
```

7.Remove any unnecessary code that’s not used in this exercise

```
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 = -1; // Initialize to an invalid value

        // Prompt the user for the number of products
        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");

        // Using a do-while loop to ensure valid input
        do {
            try {
                System.out.print("Number of products: ");
                maxSize = scanner.nextInt(); // Read user input

                // Validate input
                if (maxSize < 0) {
                    System.out.println("Incorrect value entered. Please enter a non-negative integer.");
                }
            } catch (InputMismatchException e) {
                System.out.println("Incorrect data type entered! Please enter a numeric value.");
                scanner.next(); // Clear the invalid input
            }
        } while (maxSize < 0); // Continue looping until a non-negative value is entered

        // Handle based on the value of maxSize
        if (maxSize == 0) {
            System.out.println("No products required!");
```

```
} else {
    // Create a Product array with size maxSize
    Product[] products = new Product[maxSize];
    System.out.println("You will be adding " + maxSize + " products.");

    // Populate the Product array
    for (int i = 0; i < products.length; i++) {
        // Clear input buffer if necessary
        scanner.nextLine(); // Consume any leftover newline characters

        System.out.println("Enter details for product " + (i + 1) + ":");

        System.out.print("Enter item number: ");
        int itemNumber = scanner.nextInt();

        System.out.print("Enter product name: ");
        scanner.nextLine(); // Consume the newline left by nextInt()
        String name = scanner.nextLine();

        System.out.print("Enter quantity in stock: ");
        int quantity = scanner.nextInt();

        System.out.print("Enter price: ");
        double price = scanner.nextDouble();
```

```
            // Create a new Product object and add it to the array
            products[i] = new Product(itemNumber, name, quantity, price);
        }

        // Display the details of all products using a for-each loop
        System.out.println("Products added:");
        for (Product product : products) {
            System.out.println(product);
            System.out.println();
        }
    }

    // Close the scanner
    scanner.close();
  }
}
```

8.Save the Project

1. In Eclipse

1. **Save Individual Files**:
   - Right-click on the file you want to save in the Project Explorer.
   - Select **"Save"** or **"Save All"** to save changes.
2. **Save the Entire Project**:
   - Click on **File** in the menu bar.
   - Select **"Save All"** or press Ctrl + Shift + S to save all open files and the entire project.
3. **Export the Project** (Optional, for backup or sharing):
   - Right-click on your project in the Project Explorer.
   - Select **"Export"**.
   - Choose **"General"** > **"Archive File"**.
   - Follow the prompts to create a ZIP file of your project.

2. In IntelliJ IDEA

1. **Save Individual Files**:
    - Click **File** in the menu bar.
    - Select **"Save All"** or press Ctrl + S to save the currently open files.
2. **Save the Entire Project**:
    - IntelliJ IDEA auto-saves changes, but you can manually ensure everything is saved by clicking **File** > **"Save All"**.
3. **Export the Project** (Optional, for backup or sharing):
    - Click **File** in the menu bar.
    - Select **"Export to Zip File"** or **"Project Structure"** and follow the prompts to create an archive of your project.

3. In NetBeans

1. **Save Individual Files**:
    - Click **File** in the menu bar.
    - Select **"Save"** or **"Save All"**.
2. **Save the Entire Project**:
    - Click **File** in the menu bar.
    - Select **"Save All"** or press Ctrl + Shift + S.
3. **Export the Project** (Optional, for backup or sharing):
    - Right-click on your project in the Projects tab.
    - Select **"Export"**.
    - Choose **"To ZIP file"** or similar options to create an archive of your project.

4. Using Command Line

1. **Navigate to Project Directory**:
    - Open a terminal or command prompt.
    - Use cd to navigate to your project directory.
2. **Save Files**:
    - Ensure all files are saved in your text editor before closing.
3. **Create a ZIP Archive** (Optional, for backup or sharing):
    - Use a command like zip -r project.zip project-directory/ to create a ZIP file of your project directory.

Final Steps

- **Verify**: Open your project and check that all files and changes are present.
- **Backup**: Consider creating a backup of your project using the export options provided by your IDE.

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.

Step 1: Review Existing Classes

Ensure you have the following basic classes and methods:

Product Class:

```
package inventory;
public class Product {
   private int id;
   private String name;
   private int quantity;
   private double price;

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

   // Getters and Setters
   public int getId() { return id; }
   public void setId(int id) { this.id = id; }

   public String getName() { return name; }
   public void setName(String name) { this.name = name; }

   public int getQuantity() { return quantity; }
   public void setQuantity(int quantity) { this.quantity = quantity; }

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

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

```
}
```

Inventory Class:

```
package inventory;

import java.util.ArrayList;
import java.util.Iterator;

public class Inventory {
    private ArrayList<Product> products;

    public Inventory() {
        products = new ArrayList<>();
    }

    public void addProduct(Product product) {
        products.add(product);
    }

    public void removeProduct(int id) {
        Iterator<Product> iterator = products.iterator();
        while (iterator.hasNext()) {
            Product product = iterator.next();
            if (product.getId() == id) {
                iterator.remove();
                return;
            }
        }
    }

    public Product findProductById(int id) {
        for (Product product : products) {
            if (product.getId() == id) {
                return product;
            }
        }
        return null;
    }

    public ArrayList<Product> findProductsByName(String name) {
        ArrayList<Product> result = new ArrayList<>();
        for (Product product : products) {
            if (product.getName().equalsIgnoreCase(name)) {
                result.add(product);
            }
        }
        return result;
    }

    public void displayInventory() {
        for (Product product : products) {
            System.out.println(product);
```

```
    }
  }

  public ArrayList<Product> getProducts() {
    return products;
  }
}
```

Step 2: Implement Static Methods (Section 7.3)

Add a utility class for static methods related to inventory management.

InventoryUtils Class

```
package inventory;

public class InventoryUtils {
  public static double calculateTotalValue(Inventory inventory) {
    double total = 0;
    for (Product product : inventory.getProducts()) {
      total += product.getQuantity() * product.getPrice();
    }
    return total;
  }

  public static Product findCheapestProduct(Inventory inventory) {
    if (inventory.getProducts().isEmpty()) {
      return null;
    }
    Product cheapest = inventory.getProducts().get(0);
    for (Product product : inventory.getProducts()) {
      if (product.getPrice() < cheapest.getPrice()) {
        cheapest = product;
      }
    }
    return cheapest;
  }
}
```

Step 3: Create a User Interface (Section 6.2)

Enhance the console-based user interface to support additional features.

InventoryApp Class

```
package inventory;

import java.util.ArrayList;
import java.util.Scanner;

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

    while (true) {
      System.out.println("1. Add Product");
      System.out.println("2. Remove Product");
```

```
System.out.println("3. Find Product by ID");
System.out.println("4. Find Products by Name");
System.out.println("5. Display Inventory");
System.out.println("6. Calculate Total Value");
System.out.println("7. Find Cheapest Product");
System.out.println("8. Exit");

int choice = scanner.nextInt();
scanner.nextLine(); // Consume newline

switch (choice) {
    case 1:
        System.out.print("Enter ID: ");
        int id = scanner.nextInt();
        scanner.nextLine();
        System.out.print("Enter Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Quantity: ");
        int quantity = scanner.nextInt();
        System.out.print("Enter Price: ");
        double price = scanner.nextDouble();
        inventory.addProduct(new Product(id, name, quantity, price));
        break;

    case 2:
        System.out.print("Enter ID to remove: ");
        id = scanner.nextInt();
        inventory.removeProduct(id);
        break;

    case 3:
        System.out.print("Enter ID to find: ");
        id = scanner.nextInt();
        Product product = inventory.findProductById(id);
        if (product != null) {
            System.out.println(product);
        } else {
            System.out.println("Product not found.");
        }
        break;

    case 4:
        System.out.print("Enter Name to search: ");
        name = scanner.nextLine();
        ArrayList<Product> productsByName = inventory.findProductsByName(name);
        if (productsByName.isEmpty()) {
            System.out.println("No products found with that name.");
        } else {
            for (Product p : productsByName) {
                System.out.println(p);
            }
```

```
                }
                break;

            case 5:
                inventory.displayInventory();
                break;

            case 6:
                double totalValue = InventoryUtils.calculateTotalValue(inventory);
                System.out.println("Total Inventory Value: " + totalValue);
                break;

            case 7:
                Product cheapestProduct = InventoryUtils.findCheapestProduct(inventory);
                if (cheapestProduct != null) {
                    System.out.println("Cheapest Product: " + cheapestProduct);
                } else {
                    System.out.println("No products in inventory.");
                }
                break;

            case 8:
                System.out.println("Exiting...");
                scanner.close();
                return;

            default:
                System.out.println("Invalid choice. Try again.");
        }
    }
}
```

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. Thismethod will not return any values and will accept the products array as a parameter. Remember when you passan 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

a) Create the displayInventory Method
In your ProductTester class, create a static method called displayInventory that accepts an array of Product as a parameter. This method will print the products in the inventory.

```
package inventory;

public class ProductTester {

   public static void main(String[] args) {
      // Your main method code
   }
   // New static method
   public static void displayInventory(Product[] products) {
      for (Product product : products) {
         if (product != null) {
            System.out.println(product);
         }
      }
   }
}
```

b) Move Display Code to displayInventory
Move the code that prints the products from the main method into the displayInventory method.
Assuming the main method has code like this to display the products:

```
// Example code to display inventory
for (Product product : products) {
   if (product != null) {
      System.out.println(product);
   }
}
```

This code will now be part of the displayInventory method.
c) Call displayInventory from main
Replace the code in main that displays the products with a call to the new displayInventory method.

```
// Call the displayInventory method
displayInventory(products);
```

2. Refactor addToInventory Method

e) Create the addToInventory Method

Add a static method addToInventory to handle adding products to the inventory.

```
public static void addToInventory(Product[] products, Scanner scanner) {
    // Your method implementation will go here
}
```

f) Move Product Addition Code

Move the code that adds products to the addToInventory method. The code might look like this:

```
// Example code to add products
for (int i = 0; i < products.length; i++) {
    System.out.print("Enter ID: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // Consume newline
    System.out.print("Enter Name: ");
    String name = scanner.nextLine();
    System.out.print("Enter Quantity: ");
    int quantity = scanner.nextInt();
    System.out.print("Enter Price: ");
    double price = scanner.nextDouble();
    scanner.nextLine(); // Consume newline

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

g) Move Local Variables

Move the local variables used in the addToInventory method (e.g., tempNumber, tempName, tempQty, tempPrice) to the method's parameter list or as local variables within the method.

h) Call addToInventory from main

Replace the loop in the main method that adds products with a call to addToInventory.

```
// Call the addToInventory method
addToInventory(products, scanner);
```

3. Refactor getNumProducts Method

j) Create the getNumProducts Method

Add a static method getNumProducts to handle getting the number of products from the user.

```
public static int getNumProducts(Scanner scanner) {
    System.out.print("Enter the maximum number of products: ");
    int maxSize = scanner.nextInt();
    scanner.nextLine(); // Consume newline
    return maxSize;
}
```

k) Update main Method

Replace the code in the main method that gets the number of products with a call to getNumProducts. Use the returned value to initialize products.

```
// Call the getNumProducts method and store the result
int maxSize = getNumProducts(scanner);
Product[] products = new Product[maxSize];
```

Final ProductTester Class

Here's the updated ProductTester class with all changes applied:

java

Copy code

```
package inventory;

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);
        displayInventory(products);

        scanner.close();
    }

    public static void displayInventory(Product[] products) {
        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++) {
            System.out.print("Enter ID: ");
            int id = scanner.nextInt();
            scanner.nextLine(); // Consume newline
            System.out.print("Enter Name: ");
            String name = scanner.nextLine();
            System.out.print("Enter Quantity: ");
            int quantity = scanner.nextInt();
            System.out.print("Enter Price: ");
            double price = scanner.nextDouble();
            scanner.nextLine(); // Consume newline

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

    public static int getNumProducts(Scanner scanner) {
        System.out.print("Enter the maximum number of products: ");
        int maxSize = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        return maxSize;
    }
}
```

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.

```
package inventory;

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

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

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public int getQuantity() { return quantity; }
    public void setQuantity(int quantity) { this.quantity = quantity; }

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

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

    // Method to add to inventory
    public void addToInventory(int quantity) {
        if (quantity > 0) {
            this.quantity += quantity;
        } else {
            System.out.println("Cannot add a negative quantity.");
        }
    }

    // Method to deduct from inventory
    public void deductFromInventory(int quantity) {
        if (quantity > 0) {
            if (this.quantity >= quantity) {
                this.quantity -= quantity;
```

```
            } else {
                System.out.println("Not enough stock to deduct.");
            }
        } else {
            System.out.println("Cannot deduct a negative quantity.");
        }
    }
}
```

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.

1. Implement the Menu System
a) Create the getMenuOption Method
This method will display the menu options and return the user's choice. It will validate input to ensure only numbers between 0 and 4 are accepted.

```
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 menuOption;
      do {
         menuOption = getMenuOption(scanner);
         switch (menuOption) {
            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 option. Please try again.");
         }
      } while (menuOption != 0);

      scanner.close();
```

```
    }

    // Method to display menu and get user choice
    public static int getMenuOption(Scanner scanner) {
        int option = -1;
        while (option < 0 || option > 4) {
            try {
                System.out.println("Menu:");
                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: ");
                option = scanner.nextInt();
                scanner.nextLine(); // Consume newline
                if (option < 0 || option > 4) {
                    System.out.println("Invalid option. Please enter a number between 0 and 4.");
                }
            } catch (InputMismatchException e) {
                System.out.println("Invalid input. Please enter a number between 0 and 4.");
                scanner.next(); // Clear the invalid input
            }
        }
        return option;
    }
```

2. Create Method for Selecting Product

b) Create the getProductNumber Method

This method allows users to select a product by displaying the product names and their index in the array.

```
    // Method to get the product number from the user
    public static int getProductNumber(Product[] products, Scanner scanner) {
        int productChoice = -1;
        while (productChoice < 0 || productChoice >= products.length) {
            try {
                System.out.println("Select a product to update:");
                for (int i = 0; i < products.length; i++) {
                    if (products[i] != null) {
                        System.out.println(i + ": " + products[i].getName());
                    }
                }
                System.out.print("Enter the product number: ");
                productChoice = scanner.nextInt();
                scanner.nextLine(); // Consume newline
                if (productChoice < 0 || productChoice >= products.length || products[productChoice] ==
null) {
                    System.out.println("Invalid product number. Please select a valid product.");
                }
            } catch (InputMismatchException e) {
                System.out.println("Invalid input. Please enter a valid product number.");
                scanner.next(); // Clear the invalid input
```

```
        }
      }
      return productChoice;
    }
```

3. Create Method for Adding Inventory

c) Create the addInventory Method

This method will add stock to a selected product. It prompts the user for the amount to add and updates the product.

java

Copy code

```
  // Method to add inventory
  public static void addInventory(Product[] products, Scanner scanner) {
    int productChoice = getProductNumber(products, scanner);
    int updateValue = -1;
    while (updateValue < 0) {
      try {
        System.out.print("How many products do you want to add? ");
        updateValue = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        if (updateValue < 0) {
          System.out.println("Quantity must be 0 or greater.");
        } else {
          products[productChoice].addToInventory(updateValue);
          System.out.println("Stock updated successfully.");
        }
      } catch (InputMismatchException e) {
        System.out.println("Invalid input. Please enter a positive number.");
        scanner.next(); // Clear the invalid input
      }
    }
  }
```

4. Create Method for Deducting Inventory

d) Create the deductInventory Method

This method will deduct stock from a selected product. It ensures that the deduction amount is valid and does not exceed the available stock.

```
  // Method to deduct inventory
  public static void deductInventory(Product[] products, Scanner scanner) {
    int productChoice = getProductNumber(products, scanner);
    int updateValue = -1;
    while (updateValue < 0) {
      try {
        System.out.print("How many products do you want to deduct? ");
        updateValue = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        if (updateValue < 0) {
          System.out.println("Quantity must be 0 or greater.");
        } else if (updateValue > products[productChoice].getQuantity()) {
          System.out.println("Cannot deduct more than the available stock.");
        } else {
          products[productChoice].deductFromInventory(updateValue);
          System.out.println("Stock updated successfully.");
```

```
            }
        } catch (InputMismatchException e) {
            System.out.println("Invalid input. Please enter a positive number.");
            scanner.next(); // Clear the invalid input
        }
    }
}
```

5. Create Method for Discontinuing Products

e) Create the discontinueProduct Method

This method will allow the user to remove a product from the inventory.

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

Final ProductTester Class

Here's the complete ProductTester class with all the methods added:

```
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 menuOption;
        do {
            menuOption = getMenuOption(scanner);
            switch (menuOption) {
                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 option. Please try again.");
        }
    } while (menuOption != 0);

    scanner.close();
}

public static int getMenuOption(Scanner scanner) {
    int option = -1;
    while (option < 0 || option > 4) {
        try {
            System.out.println("Menu:");
            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: ");
            option = scanner.nextInt();
            scanner.nextLine(); // Consume newline
            if (option < 0 || option > 4) {
                System.out.println("Invalid option. Please enter a number between 0 and 4.");
            }
        } catch (InputMismatchException e) {
            System.out.println("Invalid input. Please enter a number between 0 and 4.");
            scanner.next(); // Clear the invalid input
        }
    }
    return option;
}

public static int getNumProducts(Scanner scanner) {
    System.out.print("Enter the maximum number of products: ");
    int maxSize = scanner.nextInt();
    scanner.nextLine(); // Consume newline
    return maxSize;
}

public static void displayInventory(Product[] products) {
    for (int i = 0; i < products.length; i++) {
        if (products[i] != null) {
            System.out.println(i + ": " + products[i]);
        }
    }
}

public static void addToInventory(Product[] products, Scanner scanner) {
    for (int i = 0; i < products.length; i++) {
        System.out.print("Enter ID: ");
        int id = scanner.nextInt();
```

```
        scanner.nextLine(); // Consume newline
        System.out.print("Enter Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Quantity: ");
        int quantity = scanner.nextInt();
        System.out.print("Enter Price: ");
        double price = scanner.nextDouble();
        scanner.nextLine(); // Consume newline

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

    public static int
```

e) The final menu option to implement is the ability to mark stock as discontinued.
i. The method should be called discontinueInventory, have no return value and take the products array
and a Scanner object as parameters. It should have a single local variable named productChoice that stores an integer value and does not need to be initialized. 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. Now use the setActive method to set the value of active to false for the chosen product object.
f) You now need to create a method that will bring it all together.
i. The method should be called executeMenuChoice, have no return value and take the menu choice, products array and a Scanner object as parameters. It does not require any local variables. Write the code under main.
ii. Use a switch statement to execute the methods that you have created in this exercise. For each case statement use an output statement that will display one of the following heading before executing the appropriate method:
View Product List
Add Stock
Deduct Stock
Discontinue Stock
g) The final stage of this exercise is to update the main method to make use of the new functionality. Update your
code so that your main method matches the following code:
h) Run and test your code

e) Add the discontinueInventory Method

This method will allow users to mark a product as discontinued by setting its active status to false. First, ensure that the Product class has an active field and a setActive method. If not, add them:

Updated Product Class with active Field:

```
package inventory;

public class Product {
    private int id;
    private String name;
    private int quantity;
    private double price;
    private boolean active; // New field
```

```
    // Constructor
    public Product(int id, String name, int quantity, double price) {
        this.id = id;
        this.name = name;
        this.quantity = quantity;
        this.price = price;
        this.active = true; // By default, the product is active
    }

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public int getQuantity() { return quantity; }
    public void setQuantity(int quantity) { this.quantity = quantity; }

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

    public boolean isActive() { return active; }
    public void setActive(boolean active) { this.active = active; }

    @Override
    public String toString() {
        return "Product [ID=" + id + ", Name=" + name + ", Quantity=" + quantity + ", Price=" + price + ",
Active=" + active + "]";
    }

    public void addToInventory(int quantity) {
        if (quantity > 0) {
            this.quantity += quantity;
        } else {
            System.out.println("Cannot add a negative quantity.");
        }
    }

    public void deductFromInventory(int quantity) {
        if (quantity > 0) {
            if (this.quantity >= quantity) {
                this.quantity -= quantity;
            } else {
                System.out.println("Not enough stock to deduct.");
            }
        } else {
            System.out.println("Cannot deduct a negative quantity.");
        }
    }
}
```

discontinueInventory Method Implementation

```
// Method to mark a product as discontinued
public static void discontinueInventory(Product[] products, Scanner scanner) {
    int productChoice = getProductNumber(products, scanner);
    if (productChoice >= 0 && productChoice < products.length && products[productChoice] != null)
{
        products[productChoice].setActive(false);
        System.out.println("Product discontinued successfully.");
    } else {
        System.out.println("Invalid product selection.");
    }
}
```

2. Implement the executeMenuChoice Method

f) Create the executeMenuChoice Method

This method will handle executing the appropriate functionality based on the user's menu choice.

```
// Method to execute menu choice
public static void executeMenuChoice(int menuChoice, Product[] products, Scanner scanner) {
    switch (menuChoice) {
        case 1:
            System.out.println("View Product List");
            displayInventory(products);
            break;
        case 2:
            System.out.println("Add Stock");
            addInventory(products, scanner);
            break;
        case 3:
            System.out.println("Deduct Stock");
            deductInventory(products, scanner);
            break;
        case 4:
            System.out.println("Discontinue Stock");
            discontinueInventory(products, scanner);
            break;
        default:
            System.out.println("Invalid option.");
            break;
    }
}
```

3. Update the main Method

g) Update the main Method

Replace the existing menu handling logic in main with calls to the executeMenuChoice method.

```
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 menuOption;
    do {
```

```
        menuOption = getMenuOption(scanner);
        executeMenuChoice(menuOption, products, scanner);
    } while (menuOption != 0);

    scanner.close();
}
```

4. Summary

Here's a complete ProductTester class incorporating all the functionality:

```
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 menuOption;
        do {
            menuOption = getMenuOption(scanner);
            executeMenuChoice(menuOption, products, scanner);
        } while (menuOption != 0);

        scanner.close();
    }

    public static int getMenuOption(Scanner scanner) {
        int option = -1;
        while (option < 0 || option > 4) {
            try {
                System.out.println("Menu:");
                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: ");
                option = scanner.nextInt();
                scanner.nextLine(); // Consume newline
                if (option < 0 || option > 4) {
                    System.out.println("Invalid option. Please enter a number between 0 and 4.");
                }
            } catch (InputMismatchException e) {
                System.out.println("Invalid input. Please enter a number between 0 and 4.");
                scanner.next(); // Clear the invalid input
            }
```

```
        }
        return option;
    }

    public static int getNumProducts(Scanner scanner) {
        System.out.print("Enter the maximum number of products: ");
        int maxSize = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        return maxSize;
    }

    public static void displayInventory(Product[] products) {
        for (int i = 0; i < products.length; i++) {
            if (products[i] != null) {
                System.out.println(i + ": " + products[i]);
            }
        }
    }

    public static void addToInventory(Product[] products, Scanner scanner) {
        for (int i = 0; i < products.length; i++) {
            System.out.print("Enter ID: ");
            int id = scanner.nextInt();
            scanner.nextLine(); // Consume newline
            System.out.print("Enter Name: ");
            String name = scanner.nextLine();
            System.out.print("Enter Quantity: ");
            int quantity = scanner.nextInt();
            System.out.print("Enter Price: ");
            double price = scanner.nextDouble();
            scanner.nextLine(); // Consume newline

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

    public static int getProductNumber(Product[] products, Scanner scanner) {
        int productChoice = -1;
        while (productChoice < 0 || productChoice >= products.length) {
            try {
                System.out.println("Select a product to update:");
                for (int i = 0; i < products.length; i++) {
                    if (products[i] != null) {
                        System.out.println(i + ": " + products[i].getName());
                    }
                }
                System.out.print("Enter the product number: ");
                productChoice = scanner.nextInt();
                scanner.nextLine(); // Consume newline
                if (productChoice < 0 || productChoice >= products.length || products[productChoice] ==
null) {
```

```
                System.out.println("Invalid product number. Please select a valid product.");
            }
        } catch (InputMismatchException e) {
            System.out.println("Invalid input. Please enter a valid product number.");
            scanner.next(); // Clear the invalid input
        }
    }
    return productChoice;
}

public static void addInventory(Product[] products, Scanner scanner) {
    int productChoice = getProductNumber(products, scanner);
    int updateValue = -1;
    while (updateValue < 0) {
        try {
            System.out.print("How many products do you want to add? ");
            updateValue = scanner.nextInt();
            scanner.nextLine(); // Consume newline
            if (updateValue < 0) {
                System.out.println("Quantity must be 0 or greater.");
            } else {
                products[productChoice].addToInventory(updateValue);
                System.out.println("Stock updated successfully.");
            }
        } catch (InputMismatchException e) {
            System.out.println("Invalid input. Please enter a positive number.");
            scanner.next(); // Clear the invalid input
        }
    }
}

public static void deductInventory(Product[] products, Scanner scanner) {
    int productChoice = getProductNumber(products, scanner);
    int updateValue = -1;
    while (updateValue < 0) {
        try {
            System.out.print("How many products do you want to deduct? ");
            updateValue = scanner.nextInt();
            scanner.nextLine(); // Consume newline
            if (updateValue < 0) {
                System.out.println("Quantity must be 0 or greater.");
            } else if (updateValue > products[productChoice].getQuantity()) {
                System.out.println("Cannot deduct more than the available stock
```

Java Fundamentals
Section 7 Part 2: 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):

• Working with subclasses (Section 7.4)

♣ using extends

♣ using super()

♣ Overriding methods from a superclass

• Updating the user interface (Section 4.2)

DVD	
Attribute	Sample Data
Name of the product.	Daredevil
Price.	8.99
Quantity of units in stock.	50
Item number.	1
Length (minutes)	99
Age Rating	15
Film Studio	20th Century Fox

ii. Complete a sample Data table for a list of CDs of your choice.

CD	
Attribute	Sample Data
Name of the product.	Dreams we never lost
Price.	7.99
Quantity of units in stock.	50
Item number.	2
Artist	Tidelines
Number of songs	14
label	Tide Lines Music

Attribute	Sample Data
Name of the product	Daredevil
Price	8.99
Quantity of units in stock	50
Item number	1
Length (minutes)	99
Age Rating	15
Film Studio	20th Century Fox

ii. Sample Data Table for CDs

Attribute	Sample Data
Name of the product	Dreams We Never Lost
Price	7.99
Quantity of units in stock	50
Item number	2
Artist	Tide Lines
Number of songs	14
Label	Tide Lines Music

2.You are going to implement inheritance by creating subclasses of the Product class.
a. Create a subclass of the Product class called DVD that has additional instance fields to store the movie length in
minutes, the age rating and the Film Studio that released the movie.
b. Create a single constructor that accepts values for every instance field for both the DVD and Product classes. Use
the super() call to the constructor in Product passing the required parameters.
c. Create getters and setters for the DVD instance fields.
d. Follow exactly the same process to create a subclass of Product named CD. Create the instance fields, constructor
and getter and setters.

```
public class DVD extends Product {
    private int length; // Movie length in minutes
    private int ageRating; // Age rating
    private String filmStudio; // Film studio

    // Constructor
    public DVD(String name, double price, int quantity, int itemNumber, int length, int ageRating,
String filmStudio) {
        super(name, price, quantity, itemNumber); // Call to the Product constructor
        this.length = length;
        this.ageRating = ageRating;
        this.filmStudio = filmStudio;
    }
```

```
    // Getters and Setters
    public int getLength() {
        return length;
    }

    public void setLength(int length) {
        this.length = length;
    }

    public int getAgeRating() {
        return ageRating;
    }

    public void setAgeRating(int ageRating) {
        this.ageRating = ageRating;
    }

    public String getFilmStudio() {
        return filmStudio;
    }

    public void setFilmStudio(String filmStudio) {
        this.filmStudio = filmStudio;
    }
}
```

3.In the DVD subclass, override the method to calculate the value of the inventory of a DVD with the same name as that method
previously created in the product class. The DVD subclass method should also add a 5% restocking fee to the value of the
inventory of that product. You will need to get the values for price and quantity in stock from the superclass.

3a. Override the getInventoryValue() Method in the DVD Subclass:

1. **Access Superclass Fields**: Use the **getPrice()** and **getQuantity()** methods from the **Product** class to access the **price** and **quantity** values.
2. **Add the 5% Restocking Fee**: Calculate the inventory value with the additional 5% restocking fee.

```
public class DVD extends Product {
    private int length; // Movie length in minutes
    private int ageRating; // Age rating
    private String filmStudio; // Film studio

    // Constructor
    public DVD(String name, double price, int quantity, int itemNumber, int length, int ageRating,
String filmStudio) {
        super(name, price, quantity, itemNumber); // Call to the Product constructor
        this.length = length;
        this.ageRating = ageRating;
        this.filmStudio = filmStudio;
    }
```

```
    // Override the method to calculate inventory value with a 5% restocking fee
    @Override
    public double getInventoryValue() {
        double baseValue = super.getPrice() * super.getQuantity(); // Base inventory value
        return baseValue * 1.05; // Add 5% restocking fee
    }

    // Getters and Setters
    public int getLength() {
        return length;
    }

    public void setLength(int length) {
        this.length = length;
    }

    public int getAgeRating() {
        return ageRating;
    }

    public void setAgeRating(int ageRating) {
        this.ageRating = ageRating;
    }

    public String getFilmStudio() {
        return filmStudio;

    public void setFilmStudio(String filmStudio) {
        this.filmStudio = filmStudio;
    }
}
```

4.You are now going to define the output for both the DVD and CD objects.

a. Override the toString() method from the Product class so that all information about new subclass objects (DVDs) can be printed to the output console. Use the getter methods for the values instead of referencing the fields directly, this enforces using the local version of the methods if they exist instead of the methods at the superclass. Your output should look like the following:

Item Number : 1
Name : Daredevil
Movie Length : 99
Age Rating : 15
Film Studio : 20th Century Fox
Quantity in stock: 50
Price : 8.99
Stock Value : 471.975
Product Status : Active

b. Do the same in the CD class so that you produce the following output:

Item Number : 2

Name : Dreams we never lost
Artist : Tidelines
Songs on Album : 14
Record label : Tide Lines Music
Quantity in stock: 50
Price : 7.99
Stock Value : 399.5
Product Status : Active.

```
@Override
public String toString() {
   return "Item Number: " + getItemNumber() + "\n" +
         "Name: " + getName() + "\n" +
         "Movie Length: " + getLength() + " minutes\n" +
         "Age Rating: " + getAgeRating() + "\n" +
         "Film Studio: " + getFilmStudio() + "\n" +
         "Quantity in stock: " + getQuantity() + "\n" +
         "Price: $" + getPrice() + "\n" +
         "Stock Value: $" + getInventoryValue() + "\n" +
         "Product Status: " + (isActive() ? "Active" : "Inactive");
}
@Override
public String toString() {
   return "Item Number: " + getItemNumber() + "\n" +
         "Name: " + getName() + "\n" +
         "Artist: " + getArtist() + "\n" +
         "Songs on Album: " + getNumberOfSongs() + "\n" +
         "Record label: " + getLabel() + "\n" +
         "Quantity in stock: " + getQuantity() + "\n" +
         "Price: $" + getPrice() + "\n" +
         "Stock Value: $" + getInventoryValue() + "\n" +
         "Product Status: " + (isActive() ? "Active" : "Inactive");
}
```

5. Modify the ProductTester class so that you populate the products array with either CD or DVD objects.

a. Copy the addToInventory method and paste it directly under the original. Rename the method to addCDToInventory.

b. Add additional temporary variables to include the instance fields that you added to the CD class.

c. Update the prompts to ask the user for the information in the following order:

Please enter the CD name:

3

Please enter the artist name:

Please enter the record label name:

Please enter the number of songs:

Please enter the quantity of stock for this product:

Please enter the price for this product:

Please enter the item number:

You may have to clear the input buffer before you ask for any values.

d. The last line in the method creates a product object in the products array, you will need to update

this.
i. Currently the position in the array is identified with the variable i. As this method is no longer in the for loop
it does not recognise i as a variable. To resolve this add i as a parameter at the end of the method signature.
ii. You will now use polymorphism to create a CD object instead of a generic product object. Update the code
to identify CD as the object type after the = new statement.
iii. You get an error when you do this as the argument list does not match the parameter list. Update the
parameters so that they match the parameter list for the CD constructor.
e. Follow the same process to create a DVD object. The prompts should be displayed in the following order:
Please enter the DVD name:
Please enter the film studio name:
Please enter the age rating:
Please enter the length in minutes:
Please enter the quantity of stock for this product:
Please enter the price for this product:
Please enter the item number:

Copy and Modify the addToInventory Method to Create addCDToInventory Method:

1. **Copy the addToInventory Method**: Locate the **addToInventory** method in your **ProductTester** class and copy it.

2. **Rename the Method**: Paste the copied method directly under the original and rename it to **addCDToInventory**.

```
public static void addCDToInventory(CD[] products, int i) {
   Scanner input = new Scanner(System.in);

   // Prompt and capture CD specific details
   System.out.print("Please enter the CD name: ");
   String name = input.nextLine();

   System.out.print("Please enter the artist name: ");
   String artist = input.nextLine();

   System.out.print("Please enter the record label name: ");
   String label = input.nextLine();

   System.out.print("Please enter the number of songs: ");
   int numberOfSongs = input.nextInt();

   // Clear the input buffer if necessary
   input.nextLine();

   // Prompt and capture common product details
   System.out.print("Please enter the quantity of stock for this product: ");
   int quantity = input.nextInt();

   System.out.print("Please enter the price for this product: ");
   double price = input.nextDouble();
```

```
    System.out.print("Please enter the item number: ");
    int itemNumber = input.nextInt();

    // Create a CD object and store it in the products array
    products[i] = new CD(name, price, quantity, itemNumber, artist, numberOfSongs, label);
}
public static void addDVDToInventory(DVD[] products, int i) {
    Scanner input = new Scanner(System.in);

    // Prompt and capture DVD specific details
    System.out.print("Please enter the DVD name: ");
    String name = input.nextLine();

    System.out.print("Please enter the film studio name: ");
    String filmStudio = input.nextLine();

    System.out.print("Please enter the age rating: ");
    int ageRating = input.nextInt();

    System.out.print("Please enter the length in minutes: ");
    int length = input.nextInt();

    // Clear the input buffer if necessary
    input.nextLine();

    // Prompt and capture common product details
    System.out.print("Please enter the quantity of stock for this product: ");
    int quantity = input.nextInt();

    System.out.print("Please enter the price for this product: ");
    double price = input.nextDouble();

    System.out.print("Please enter the item number: ");
    int itemNumber = input.nextInt();

    // Create a DVD object and store it in the products array
    products[i] = new DVD(name, price, quantity, itemNumber, length, ageRating, filmStudio);
}
```

6. Use a for each loop to display the information for each individual product in the products array.

```
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 = -1; // Initialize to an invalid value

        // Prompt the user for the number of products
```

```
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");

// Using a do-while loop to ensure valid input
do {
    try {
        System.out.print("Number of products: ");
        maxSize = scanner.nextInt(); // Read user input

        // Validate input
        if (maxSize < 0) {
            System.out.println("Incorrect value entered. Please enter a non-negative integer.");
        }
    } catch (InputMismatchException e) {
        System.out.println("Incorrect data type entered! Please enter a numeric value.");
        scanner.next(); // Clear the invalid input
    }
} while (maxSize < 0); // Continue looping until a non-negative value is entered

// Handle based on the value of maxSize
if (maxSize == 0) {
    System.out.println("No products required!");
} else {
    // Create a Product array with size maxSize
    Product[] products = new Product[maxSize];
    System.out.println("You will be adding " + maxSize + " products.");

    // Populate the Product array
    for (int i = 0; i < products.length; i++) {
        // Clear input buffer if necessary
        scanner.nextLine(); // Consume any leftover newline characters

        System.out.println("Enter details for product " + (i + 1) + ":");

        System.out.print("Enter item number: ");
        int itemNumber = scanner.nextInt();

        System.out.print("Enter product name: ");
        scanner.nextLine(); // Consume the newline left by nextInt()
        String name = scanner.nextLine();

        System.out.print("Enter quantity in stock: ");
        int quantity = scanner.nextInt();

        System.out.print("Enter price: ");
        double price = scanner.nextDouble();

        // Create a new Product object and add it to the array
        products[i] = new Product(itemNumber, name, quantity, price);
    }
```

```
        // Display the details of all products using a for-each loop
        System.out.println("Products added:");
        for (Product product : products) {
            System.out.println(product);
            System.out.println();
        }
    }

    // Close the scanner
    scanner.close();
  }
}
```

8. Do not allow the user to add stock to a discontinued product line.

Update the addToInventory method in the Product class to stop the adding of stock to a discontinued line

```
package inventory;

public class Product {
  // Instance variables
  private int itemNumber;
  private String name;
  private int quantity;
  private double price;
  private boolean discontinued; // New field to indicate if the product is discontinued

  // Default constructor
  public Product() {
    this.itemNumber = 0;
    this.name = "";
    this.quantity = 0;
    this.price = 0.0;
    this.discontinued = false; // Default value is not discontinued
  }

  // Constructor with parameters
  public Product(int itemNumber, String name, int quantity, double price, boolean discontinued) {
    this.itemNumber = itemNumber;
    this.name = name;
    this.quantity = quantity;
    this.price = price;
    this.discontinued = discontinued;
  }

  // Getter and setter methods
  public int getItemNumber() {
    return itemNumber;
  }

  public void setItemNumber(int itemNumber) {
    this.itemNumber = itemNumber;
  }
```

```
public String getName() {
   return name;
}

public void setName(String name) {
   this.name = name;
}

public int getQuantity() {
   return quantity;
}

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

public double getPrice() {
   return price;
}

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

public boolean isDiscontinued() {
   return discontinued;
}

public void setDiscontinued(boolean discontinued) {
   this.discontinued = discontinued;
}

// Method to add stock
public void addToInventory(int quantityToAdd) {
   if (!discontinued) { // Check if the product is not discontinued
      if (quantityToAdd > 0) {
         this.quantity += quantityToAdd;
         System.out.println("Added " + quantityToAdd + " units to inventory.");
      } else {
         System.out.println("Quantity to add must be greater than zero.");
      }
   } else {
      System.out.println("Cannot add stock to a discontinued product.");
   }
}

@Override
public String toString() {
   return "Item Number : " + itemNumber +
         "\nName : " + name +
```

```
            "\nQuantity in stock: " + quantity +
            "\nPrice : " + price +
            "\nDiscontinued : " + (discontinued ? "Yes" : "No");
    }
}
```

9. Run and test your code

```
Enter the number of products you would like to add:
Enter 0 (zero) if you do not wish to add products
Number of products: 2
Enter details for product 1:
Enter item number: 101
Enter product name: Widget
Enter quantity in stock: 50
Enter price: 19.99
Is the product discontinued (true/false): false

Enter details for product 2:
Enter item number: 102
Enter product name: Gizmo
Enter quantity in stock: 30
Enter price: 29.99
Is the product discontinued (true/false): false

Products added:
Item Number : 101
Name : Widget
Quantity in stock: 50
Price : 19.99
Discontinued : No

Item Number : 102
Name : Gizmo
Quantity in stock: 30
Price : 29.99
Discontinued : No
```