```
package InventoryManagerandShoppingcart;


import java.io.IOException;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;


/**

 * The InventoryManager class represents a manager for the store's inventory of salable products.

 * @param <Inventory>

 */


public class InventoryManager<Inventory> {


   private List<SalableProducts> inventory;


   /**

    * Constructs an InventoryManager object and initializes the inventory.

    */

   public InventoryManager() {

      inventory = new ArrayList<>();

      initializeInventory();

   }


   /**
 * Initializes the inventory with some default products.
 */
private void initializeInventory() {
   Weapon weapon1 = new Weapon("Sword", "Golden Sword", 2506.50, 10, (int) 1.9);
   Weapon weapon2 = new Weapon("Mace", "Mace of Frost", 1004.20, 12, (int) 15.8);
   Armor armor1 = new Armor("Helmet", "Helmet of Wisdom", 16272.89, 3, 0);
   Armor armor2 = new Armor("Body Armor", "Armor of Deceit", 25672.12, 7, 0);
   Health health1 = new Health("Large Health", "Full Health", 12500.00, 4);
   Health health2 = new Health("Medium Health", "Half Health", 107500.00, 17);
   Health health3 = new Health("Small Health", "Quarter Health", 1500.00, 25);

   inventory.add(weapon1);
   inventory.add(weapon2);
   inventory.add(armor1);
   inventory.add(armor2);
   inventory.add(health1);
   inventory.add(health2);
   inventory.add(health3);
}

/**
 * Removes a product from the inventory at the specified index.
 *
 * @param index the index of the product to remove
 */
```

```
public void removeProduct(int index) {
    if (index >= 0 && index < inventory.size()) {
        SalableProducts product = inventory.get(index);
        int quantity = product.getQuantity();
        if (quantity > 0) {
            product.setQuantity(quantity - 1);
            System.out.println("Product removed: " + product.getName());
        } else {
            System.out.println("Product out of stock: " + product.getName());
        }
    } else {
        System.out.println("Invalid product index");
    }
}

/**
 * Adds a product to the inventory at the specified index.
 *
 * @param index the index of the product to add
 */
public void addProduct(int index) {
    if (index >= 0 && index < inventory.size()) {
        SalableProducts product = inventory.get(index);
        int quantity = product.getQuantity();
        product.setQuantity(quantity + 1);
        System.out.println("Product added: " + product.getName());
    } else {
        System.out.println("Invalid product index");
    }
}

/**
 * Retrieves the inventory of salable products.
 *
 * @return the list of products in the inventory
 */
public List<SalableProducts> getInventory() {
    return inventory;
}

/**
 * Adds a product to the inventory.
 *
 * @param product the product to add
 */
public void addProduct(SalableProducts product) {
    inventory.add(product);
}

/**
```

```
 * Reads the inventory from an external JSON file.
 *
 * @throws IOException if an error occurs during file reading
 */
public void readInventoryFromFile() throws IOException {
   try {
      InventoryManager inventoryData = FileService.readInventoryFromFile();
      inventory = inventoryData.getInventory();
      System.out.println("Inventory loaded from file: " + FileService.getInventoryFile());
   } catch (IOException e) {
      System.out.println("Failed to read inventory from file: " + FileService.getInventoryFile());
      throw e;
   }
}

/**
 * Writes the inventory to an external JSON file.
 *
 * @throws IOException if an error occurs during file writing
 */
public void writeInventoryToFile() throws IOException {
   try {
        Inventory inventoryData = new InventoryManager();
      FileService.writeInventoryToFile(inventoryData);
      System.out.println("Inventory saved to file: " + FileService.getInventoryFile());
```

```
        } catch (IOException e) {
            System.out.println("Failed to write inventory to file: " + FileService.getInventoryFile());
            throw e;
        }
    }
}
```

```
package InventoryManagerandShoppingcart;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.List;

public class FileService {
    private static final String INVENTORY_FILE = "inventory.json";
    private static final Gson gson = new GsonBuilder().setPrettyPrinting().create();

    public static InventoryManager readInventoryFromFile() throws IOException {
        try (BufferedReader reader = new BufferedReader(new FileReader(INVENTORY_FILE))) {
            return gson.fromJson(reader, InventoryManager.class);
        }
    }

    public static void writeInventoryToFile(InventoryManager inventory) throws IOException {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(INVENTORY_FILE))) {
            gson.toJson(inventory, writer);
        }
    }
}
```

```
package InventoryManagerandShoppingcart;
import java.io.IOException;
import java.util.Scanner;

/**
 * The StoreFrontApplication class represents an application for managing a store's inventory and shopping cart.
 */
public class StoreFrontApplication {

    private InventoryManager inventoryManager;
    private ShoppingCart shoppingCart;

    /**
     * Constructs a StoreFrontApplication object, initializing the inventory manager and shopping cart.
     */
    public StoreFrontApplication() {
        inventoryManager = new InventoryManager();

        shoppingCart = new ShoppingCart();

    }


    /**

     * Displays the inventory with product details.

     */

    public void displayInventory() {

        System.out.println("Inventory:");

        int index = 0;

        for (SalableProducts product : inventoryManager.getInventory()) {

            System.out.println(

                    "[" + index + "] " + product.getName() + " - " + product.getDescription() + " - Price: $"

                            + product.getPrice() + " - Quantity: " + product.getQuantity());

            index++;

        }

    }


    /**

     * Purchases a product from the inventory and adds it to the shopping cart.

     *

     * @param index the index of the product to purchase

     */

    public void purchaseProduct(int index) {

        if (index >= 0 && index < inventoryManager.getInventory().size()) {
            SalableProducts product = inventoryManager.getInventory().get(index);

            if (product.getQuantity() > 0) {

                inventoryManager.removeProduct(index);

                shoppingCart.addProduct(product);

            } else {

                System.out.println("Product is out of stock: " + product.getName());

            }

        } else {

            System.out.println("Invalid product index");

        }

    }


    /**

     * Cancels a purchase by removing a product from the shopping cart and adding it back to the
inventory.

     *

     * @param index the index of the product to cancel the purchase

     */

    public void cancelPurchase(int index) {

        if (index >= 0 && index < shoppingCart.getContents().size()) {

            SalableProducts product = shoppingCart.getContents().get(index);

            inventoryManager.addProduct(product);

            shoppingCart.removeProduct(product);

        } else {

            System.out.println("Invalid product index");

        }
```

```
}

/**
 * Displays the contents of the shopping cart.
 */
public void displayShoppingCart() {
   System.out.println("Shopping Cart:");
   int index = 0;
   for (SalableProducts product : shoppingCart.getContents()) {
      System.out.println(
            "[" + index + "] " + product.getName() + " - " + product.getDescription() + " - Price: $"
                  + product.getPrice() + " - Quantity: " + product.getQuantity());
      index++;
   }
}

/**
 * Empties the shopping cart.
 */
public void emptyShoppingCart() {
   shoppingCart.emptyCart();
}

/**
 * The main entry point of the StoreFrontApplication program.
 *
 * @param args command line arguments
 */
public static void main(String[] args) {
    StoreFrontApplication storeFront = new StoreFrontApplication();

    System.out.println("Welcome to the Store Front!");

    try {
        storeFront.inventoryManager.readInventoryFromFile();
    } catch (IOException e) {
        System.out.println("Failed to load inventory from file.");
    }

    storeFront.displayInventory();

    Scanner scanner = new Scanner(System.in);
    int action;

    do {
        System.out.println("\nActions:");
        System.out.println("1. View Inventory");
        System.out.println("2. Purchase a Product");
        System.out.println("3. Cancel a Purchase");
        System.out.println("4. View Shopping Cart");
System.out.println("5. Empty Shopping Cart");

System.out.println("6. Save Inventory to File");

System.out.println("0. Exit");

System.out.print("Enter the action number: ");

action = scanner.nextInt();


switch (action) {

  case 1:

    storeFront.displayInventory();

    break;

  case 2:

    System.out.print("Enter the index of the product to purchase: ");

    int purchaseIndex = scanner.nextInt();

    storeFront.purchaseProduct(purchaseIndex);

    break;

  case 3:

    System.out.print("Enter the index of the product to cancel the purchase: ");

    int cancelIndex = scanner.nextInt();

    storeFront.cancelPurchase(cancelIndex);

    break;

  case 4:

    storeFront.displayShoppingCart();

    break;

  case 5:

    storeFront.emptyShoppingCart();
```

```
                break;
            case 6:
                try {
                    storeFront.inventoryManager.writeInventoryToFile();
                } catch (IOException e) {
                    System.out.println("Failed to save inventory to file.");
                }
                break;
            case 0:
                break;
            default:
                System.out.println("Invalid action");
        }

    } while (action != 0);

    System.out.println("Thank you for using the Store Front Application!");
    scanner.close();
  }
}
```

```
package InventoryManagerandShoppingcart;


/**
 * The SalableProducts class represents a salable product that can be bought and
sold.
 * It provides common properties and behaviors for all salable products.
 * It implements the Comparable interface to enable comparison between products.
 */
```

```java
public abstract class SalableProducts implements Comparable<SalableProducts> {
    private String name;
    private String description;
    private double price;
    private int quantity;

    /**
     * Constructs a SalableProducts object with the specified name, description,
price, and quantity.
     *
     * @param name        the name of the salable product
     * @param description the description of the salable product
     * @param price       the price of the salable product
     * @param quantity    the quantity of the salable product
     */

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

    /**
     * Gets the name of the salable product.
     *
     * @return the name of the product
     */

    public String getName() {
        return name;
    }

    /**
     * Sets the name of the salable product.
     *
     * @param name the name of the product
     */

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

    /**
     * Gets the description of the salable product.
     *
     * @return the description of the product
     */

    public String getDescription() {
        return description;
    }
```

```java
/**
 * Sets the description of the salable product.
 *
 * @param description the description of the product
 */

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

/**
 * Gets the price of the salable product.
 *
 * @return the price of the product
 */

public double getPrice() {
    return price;
}

/**
 * Sets the price of the salable product.
 *
 * @param price the price of the product
 */

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

/**
 * Gets the quantity of the salable product.
 *
 * @return the quantity of the product
 */

public int getQuantity() {
    return quantity;
}

/**
 * Sets the quantity of the salable product.
 *
 * @param quantity the quantity of the product
 */

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

/**
* Compares this salable product with another product based on their names.
*
```

```
    * @param otherProduct the other product to compare
    * @return a negative integer, zero, or a positive integer as this product is less 
than,
    *         equal to, or greater than the other product
    */

    @Override
    public int compareTo(SalableProducts otherProduct) {
        return this.name.compareToIgnoreCase(otherProduct.name);
    }
}
```

```
package inventoryManagerandShoppingCart;



import java.util.ArrayList;

import java.util.Collections;

import java.util.List;



/**

 * The InventoryManager class represents a manager for the store's inventory of salable products.

 */



public class InventoryManager {



    private List<SalableProducts> inventory;



    /**

     * Constructs an InventoryManager object and initializes the inventory.

     */
```

```
public InventoryManager() {

    inventory = new ArrayList<>();

    initializeInventory();

}


/**

 * Initializes the inventory with some default products.

 */


private void initializeInventory() {

    Weapon weapon1 = new Weapon("Sword", "Golden Sword", 2506.50, 10, (int) 1.9);

    Weapon weapon2 = new Weapon("Mace", "Mace of Frost", 1004.20, 12, (int) 15.8);

    Armor armor1 = new Armor("Helmet", "Helmet of Wisdom", 16272.89, 3, 0);

    Armor armor2 = new Armor("Body Armor", "Armor of Deceit", 25672.12, 7, 0);

    Health health1 = new Health("Large Health", "Full Health", 12500.00, 4);

    Health health2 = new Health("Medium Health", "Half Health", 107500.00, 17);

    Health health3 = new Health("Small Health", "Quarter Health", 1500.00, 25);


    inventory.add(weapon1);

    inventory.add(weapon2);

    inventory.add(armor1);

    inventory.add(armor2);

    inventory.add(health1);

    inventory.add(health2);

    inventory.add(health3);
}

/**
 * Removes a product from the inventory at the specified index.
 *
 * @param index the index of the product to remove
 */

public void removeProduct(int index) {
    if (index >= 0 && index < inventory.size()) {
        SalableProducts product = inventory.get(index);
        int quantity = product.getQuantity();
        if (quantity > 0) {
            product.setQuantity(quantity - 1);
            System.out.println("Product removed: " + product.getName());
        } else {
            System.out.println("Product out of stock: " + product.getName());
        }
    } else {
        System.out.println("Invalid product index");
    }
}

/**
 * Adds a product to the inventory at the specified index.
 *
 * @param index the index of the product to add
 */

public void addProduct(int index) {
   if (index >= 0 && index < inventory.size()) {
      SalableProducts product = inventory.get(index);
      int quantity = product.getQuantity();
      product.setQuantity(quantity + 1);
      System.out.println("Product added: " + product.getName());
   } else {
      System.out.println("Invalid product index");
   }
}

/**
 * Retrieves the inventory of salable products.
 *
 * @return the list of products in the inventory
 */

public List<SalableProducts> getInventory() {
   return inventory;
}
```

```
    /**
     * Adds a product to the inventory.
     *
     * @param product the product to add
     */

    public void addProduct(SalableProducts product) {
        inventory.add(product);
    }
}
```

```
package InventoryManagerandShoppingcart;


import java.util.ArrayList;
import java.util.List;


/**
 * The ShoppingCart class represents a shopping cart that can hold salable products.
 */


public class ShoppingCart {


    private List<SalableProducts> contents;


    /**
     * Constructs an empty shopping cart.
 */

public ShoppingCart() {
   contents = new ArrayList<>();
}

/**
 * Adds a product to the shopping cart.
 *
 * @param product the product to be added
 */

public void addProduct(SalableProducts product) {
   contents.add(product);
   System.out.println("Product added to the shopping cart: " + product.getName());
}

/**
 * Removes a product from the shopping cart.
 *
 * @param product the product to be removed
 */

public void removeProduct(SalableProducts product) {
   boolean removed = contents.remove(product);
```

```
    if (removed) {
        System.out.println("Product removed from the shopping cart: " + product.getName());
    } else {
        System.out.println("Product not found in the shopping cart: " + product.getName());
    }
}

/**
 * Constructs an empty shopping cart.
 */



public void emptyCart() {
    contents.clear();
    System.out.println("The shopping cart is emptied.");
}

/**
 * Retrieves the contents of the shopping cart.
 *
 * @return the list of products in the shopping cart
 */

public List<SalableProducts> getContents() {
    return contents;
    }
  }
```