**Lab 2**

|  |  |  |  |
| --- | --- | --- | --- |
| Student Name | | Student CSUSM ID | Contribution percentage |
| 1 | Lauren Gonzalez | gonza823 | 50 |
| 2 | Sirena Murphree | murph135 | 50 |

**Grading Rubrics (for instructor only):**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criteria | 1. Beginning | 2. Developing | 3. Proficient | 4. Exemplary |
| Modeling | 0-14 | 15-19 | 20-24 | 25-30 |
|  |  |  |  |
| Program: functionality  *correctness* | 0-9 | 10-14 | 15-19 | 20 |
|  |  |  |  |
| Program: functionality  *Behavior Testing* | 0-9 | 10-14 | 15-19 | 20 |
|  |  |  |  |
| Program: quality ->  *Readability* | 0-2 | 3-5 | 6-9 | 10 |
|  |  |  |  |
| Program: quality ->  *Modularity* | 0-2 | 3-5 | 6-9 | 10 |
|  |  |  |  |
| Program: quality ->  *Simplicity* | 0-2 | 3-5 | 6-9 | 10 |
|  |  |  |  |
| Total Grade (100) |  | | | |

**Problems:**

The ABC Company typically uses an object of the SortingUtility class to sort products. A product has at least three attributes: ID, name and price. All are accessible through their corresponding get() method but the ID is fixed once set.

The SortingUtility class implements two private sorting algorithms, bubbleSort and quickSort, each of which takes the list of products and returns an ordered list of products. The SortingUtility class also has a public method List<Product> sort(List<Product> items, int sortingApproach), which simply calls the specified sorting approach (i.e., bubbleSort or quickSort) to return a list of sorted products to its client.

The SortingUtility currently does not log the list of sorted products before returning it to the client. Now the ABC Company would like to have an improved sorting service that can log (for this lab, simply printing to the display console) the list of sorted products before returning it to the client. To implement this improved service you cannot change **the existing** SortingUtility **class for compatible reason**. Moreover, the returned products from bubbleSort should be logged (printed) with ID followed by name and price, whereas the returned products from the quicksort should be logged (printed) with name first followed by ID and price.

(30 pts) What design pattern can be used? Document your pattern-based design in UML class diagram, ensure attributes, methods, visibility, arguments and relationships are correctly included.

(70 pts) Implement your pattern-based design in Java. Implement two test scenarios: one using quicksort to sort a list of products such as books, bags, and buttons, another using bubblesort to sort the same list of products.

**Solution:**

* First, remember to zip the src folder of your project and submit the zip file to the ungraded assignment named “Lab2CodeSubmission”. One submission from each team.
* Paste a screenshot of a run of your program here.
* Also paste all you source code here.
* Save this report in PDF, then **each student** needs to submit the pdf report to the graded assignment named “Lab2ReportSubmission”.

Using Proxy with Super Helper Class

Diagram

Description automatically generated

Screen Shots

A picture containing text, newspaper, screenshot, receipt

Description automatically generatedA picture containing text, newspaper, receipt, screenshot

Description automatically generated

Driver.java

package src;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.Scanner;

public class Driver {

public static void main(String[] args) {

List<Product> items = new ArrayList<Product>();

items.add(new Product( 5715451, "Vintage Canvas Shoulder Laptop Bag", 49.99));

items.add(new Product( 7989021, "Bentgo Bag - Insulated Lunch Box Bag", 9.99));

items.add(new Product( 9379410, "Polyhedral 7-Die Scarab Dice Set", 9.41));

items.add(new Product( 9249825, "Deluxe Reusable Grocery Bag w/ Bottle Sleeves", 25.95));

items.add(new Product( 4729011, "Coleman Tent Kit", 12.97));

items.add(new Product( 7400234, "Flashlight Holster", 7.85));

items.add(new Product( 3958646, "Premium Bookends for Shelves", 17.73));

items.add(new Product( 8050622, "Dungeon & Dragons: Dungeon Master's Guide 4E", 35.10));

items.add(new Product( 5169007, "Lies My Teacher Told Me: History Textbook", 18.08));

items.add(new Product( 8277821, "Vintage Travel Casual Daypack", 26.74));

items.add(new Product( 6245816, "Batman Vol. 5: Zero Year - Dark City", 11.13));

items.add(new Product( 1165822, "Lamentations of the Flame Princess", 17.08));

ISortingUtility sul = SuperSortingUtility.getSortingUtility();

Scanner scanner = new Scanner(System.in);

int menuSelection;

do {

printMenu();

menuSelection = scanner.nextInt();

switch(menuSelection) {

case 1: case 2: items = sul.sort(items, menuSelection);

break;

case 3: shuffleList((List)items);

case 4: printList(items);

break;

default: break;

}

}while(menuSelection != 0);

}

/\*\*

\* print driver menu

\*/

public static void printMenu() {

System.out.println("\n-----------------------------------------");

System.out.printf("%-19s [%d] %-20s\n","Menu:", 0, "Exit");

System.out.printf("[%d] %-15s [%d] %-20s\n", 1, "Sort BubbleSort", 3, "Shuffle & Print List");

System.out.printf("[%d] %-15s [%d] %-20s\n", 2, "Sort QuickSort", 4, "Print List");

System.out.println("-----------------------------------------");

System.out.print("Selection: ");

}

/\*\*

\* print product list using System.out.println(Product::toString())

\* @param items

\*/

public static void printList(List<Product> items) {

System.out.println("\*\* Default print using System.out.println(Prodct::toString())");

for(Object item: items) {

System.out.println(item.toString());

}

}

/\*\*

\* Shuffle List

\* @param items

\*/

public static void shuffleList(List<Object> items) {

System.out.println("\n\*\* Shuffle List");

int limmit = items.size();

for(int i = 0; i < limmit; i++) {

int ii = (int)Math.abs((i+Math.random()\*(limmit-1-i)));

Collections.swap(items, i, ii);

}

}

}

Product.java

package src;

public class Product {

/\*\*

\* general product ID

\*/

private int productID;

/\*\*

\* general product price s

\*/

private String productName;

/\*\*

\*

\*/

private float productPrice;

/\*\*

\* Overloaded constructor

\* @param productID the product ID

\* @param productName the product name

\* @param productPrice the product price

\*/

public Product(int productID, String productName, double productPrice) {

this(productID, productName, (float) productPrice);

}

public Product(int productID, String productName, float productPrice) {

// TODO Auto-generated constructor stub

this.productID = productID;

this.productName = productName;

this.productPrice = productPrice;

}

/\*\*

\* @return the productID

\*/

public int getProductID(){

return productID;

}

/\*\*

\* @return the product name

\*/

public String getProductName() {

return productName;

}

/\*\*

\* @return the product price

\*/

public float getProductPrice() {

return productPrice;

}

/\*\*

\* @param name the name to set

\*/

public void setProductName(String name) {

this.productName = name;

}

/\*\*

\* @param price the price to set

\*/

public void setProductPrice(float price) {

this.productPrice = price;

}

/\*\*

\* @return String that describes the product

\*/

public String toString() {

return String.valueOf(productID) + " - " + productName + " - $" + String.valueOf(productPrice);

}

}

SuperSortingUtility.java

package src;

public class SuperSortingUtility {

public static ISortingUtility getSortingUtility(){

return new SortingUtilityLog(new SortingUtility());

}

}

ISortingUtility.java

package src;

import java.util.List;

public interface ISortingUtility {

public List<Product> sort(List<Product> items, int sortingApproach);

}

SortingUtility.java

package src;

import java.util.List;

import java.util.Collections;

public class SortingUtility implements ISortingUtility {

/\*\*

\* Sort a list of Products

\* @param items list of products to be sorted

\* @param sortingApproach the sort method to be used 1 = bubble sort, 2 = quick sort

\*/

public List<Product> sort(List<Product> items, int sortingApproach){

switch(sortingApproach) {

case 1: items = bubbleSort(items);

break;

case 2: items = quickSort(items);

break;

default: break;

}

return items;

}

/\*\*

\* Sort Product list using bubble sort algorithm

\* @param items list of products to be sorted

\* @return the sorted list

\*/

private List<Product> bubbleSort(List<Product> items){

int max = items.size();

for(int i = 0; i < max-1; i++) {

for(int ii = i+1; ii < max ; ii++) {

if(items.get(i).getProductID() > items.get(ii).getProductID()) {

Collections.swap(items, i, ii);

}

}

}

return items;

}

/\*\*

\* Sort Product list using quick sort algorithm

\* @param items list of products to be sorted

\* @return the sorted list

\*/

private List<Product> quickSort(List<Product> items){

quickSort(items, 0, items.size()-1);

return items;

}

/\*\*

\* sort list recursively

\* @param items the list of items

\* @param first starting index

\* @param last ending index

\*/

private void quickSort(List<Product> items, int first, int last) {

int i = first;

int ii = last;

int pivotID = items.get((first + last)/2).getProductID();

do {

while(items.get(i).getProductID() < pivotID)

i++;

while(items.get(ii).getProductID() > pivotID)

ii--;

if(i <= ii) {

Collections.swap(items, i, ii);

i++;

ii--;

}

}while(i <= ii);

if(first < ii)

quickSort(items, first, ii);

if(i < last)

quickSort(items, i, last);

}

}

SortingUtilityLog.java

package src;

import java.lang.reflect.InvocationTargetException;

import java.lang.reflect.Method;

import java.util.List;

public class SortingUtilityLog implements ISortingUtility {

/\*\*

\* the original class that this is based on

\*/

private ISortingUtility sortingUtil\_original;

/\*\*

\* Constructor

\* @param su

\*/

public SortingUtilityLog(ISortingUtility su) {

this.sortingUtil\_original = su;

}

/\*\*

\* Get the specified sort method

\* @param method

\* @return Method, the found method

\* @throws NoSuchMethodException

\* @throws InvocationTargetException

\*/

private Method reflectionSortMethod(String method) throws NoSuchMethodException, InvocationTargetException{

Method sortMethod = sortingUtil\_original.getClass().getDeclaredMethod(method, List.class);

sortMethod.setAccessible(true);

return sortMethod;

}

/\*\*

\* Sort a list of Products

\* @param items list of products to be sorted

\* @param sortingApproach the sort method to be used 1 = bubble sort, 2 = quick sort

\*/

public List<Product> sort(List<Product> items, int sortingApproach){

try {

switch(sortingApproach) {

case 1: items = (List<Product>) (this.reflectionSortMethod("bubbleSort")).invoke(sortingUtil\_original, items);

bubbleSortPrint(items);

break;

case 2: items = (List<Product>) (this.reflectionSortMethod("quickSort")).invoke(sortingUtil\_original, items);

quickSortPrint(items);

break;

default: break;

}

}catch (NoSuchMethodException e) {

System.out.println("No Such Method");

e.printStackTrace();

} catch (InvocationTargetException e) {

e.printStackTrace();

} catch(IllegalAccessException e) {}

return items;

}

/\*\*

\* Print the formatted list

\* @param items

\*/

private void bubbleSortPrint(List<Product> items) {

System.out.println("\nBubble Sort Called");

System.out.printf("%-11s %-50s %-7s\n", "Product ID", "Product Name", "Price");

System.out.printf("%-11s %-50s %-7s\n", "----------", "------------", "-----");

for(Product item: items) {

System.out.printf("%-11d %-50s $%6.2f\n", item.getProductID(), item.getProductName(), item.getProductPrice());

}

}

/\*\*

\* Print the formatted list

\* @param items

\*/

private void quickSortPrint(List<Product> items) {

System.out.println("\nQuick Sort Called");

System.out.printf("%-50s %-11s %-7s\n", "Product Name", "Product ID", "Price");

System.out.printf("%-50s %-11s %-7s\n", "------------", "----------", "-----");

for(Product item: items) {

System.out.printf("%-50s %-11d $%6.2f\n", item.getProductName(), item.getProductID(), item.getProductPrice());

}

}

}