**Batch: A2 Roll No.: 16010123032**

**Experiment / assignment / tutorial No.06**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE :Collection Framework** |

**AIM:** Create a class Employee which stores E-Name, E-Id and E-Salary of an Employee. Use class Vector to maintain an array of Employee with respect to the E-Salary. Provide the following functions

1) Create (): this function will accept the n Employee records in any order and will arrange them in the sorted order.

2) Insert (): to insert the given Employee record at appropriate index in the vector depending upon the E-Salary.

3) delete ByE-name( ): to accept the name of the Employee and delete the record having given name

4) deleteByE-Id ( ): to accept the Id of the Employee and delete the record having given E-Id.

Provide the following functions

1. boolean add(E e) : This method appends the specified element to the end of this Vector.
2. void addElement(E obj) This method adds the specified component to the end of this vector, increasing its size by one.
3. int lastIndexOf(Object o, int index) This method returns the index of the last occurrence of the specified element in this vector, searching backwards from index, or returns -1 if the element is not found.
4. void removeElementAt(int index)This method deletes the component at the specified index.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

**CO2:** Explore arrays, vectors, classes and objects in C++ and Java.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Ralph Bravaco , Shai Simoson , “Java Programing From the Group Up” Tata McGraw-Hill.

2.Grady Booch, Object Oriented Analysis and Design .

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

Vectors in Java are one of the most commonly used data structures. Similar to Arrays data structures which hold the data in a linear fashion. Vectors also store the data in a linear fashion, but unlike Arrays, they do not have a fixed size. Instead, their size can be increased on demand.

Vector class is a child class of AbstractList class and implements on List interface. To use Vectors, we first have to import Vector class from java.util package:

import java.util.Vector;

**Access Elements in Vector:**

We can access the data members simply by using the index of the element, just like we access the elements in Arrays.

Example- If we want to access the third element in a vector v, we simply refer to it as v[3].

**Vectors Constructors**

Listed below are the multiple variations of vector [constructors](https://www.edureka.co/blog/constructor-in-java/) available to use:

1. **Vector(int initialCapacity, int Increment)** – Constructs a vector with given initialCapacity and its Increment in size.
2. **Vector(int initialCapacity)*–***Constructs an empty vector with given initialCapacity. In this case, Increment is zero.
3. **Vector()** – Constructs a default vector of capacity 10.
4. **Vector(Collection c)*–***Constructs a vector with a given collection, the order of the elements is same as returned by the collection’s iterator.

There are also three protected parameters in vectors

* + **Int capacityIncrement()-** It automatically increases the capacity of the vector when the size becomes greater than capacity.
  + **Int elementCount()** – tell number of elements in the vector
  + **Object[] elementData()** – array in which elements of vector are stored

**Memory allocation of vectors:**

Vectors do not have a fixed size, instead, they have the ability to change their size dynamically. One might think that the vectors allocate indefinite long space to store objects. But this is not the case. Vectors can change their size based on two fields ‘capacity’ and ‘capacityIncrement’. Initially, a size equal to ‘capacity’ field is allocated when a vector is declared. We can insert the elements equal to the capacity. But as soon as the next element is inserted, it increases the size of the array by size ‘capacityIncrement’. Hence, it is able to change its size dynamically.

For a default constructor, the capacity is doubled whenever the capacity is full and a new element is to be inserted.

**Methods of Vectors :**

* Adding elements
* Removing elements
* Changing elements
* Iterating the vector

**Class Diagram:**

+-----------------------------------------------+

| Employee |

+-----------------------------------------------+

| - Ename: String |

| - Eid: int |

| - Esalary: double |

+-----------------------------------------------+

| + Employee(Ename: String, Eid: int, Esalary: double) |

| + Create(Employees: List<Employee>, n: int): void |

| + sort(Employees: List<Employee>): void |

| + insert(Employees: List<Employee>): void |

| + delete\_by\_Ename(Employees: List<Employee>): void |

| + delete\_by\_Eid(Employees: List<Employee>): void |

| + display(Employees: List<Employee>): void |

+-----------------------------------------------+

+-----------------------------------------------+

| Main |

+-----------------------------------------------+

| - Employees: List<Employee> |

| - n: int |

| - choice: int |

+-----------------------------------------------+

| + main(): void |

+-----------------------------------------------+

**Algorithm:**

**Step 1: Define Employee Class**

* Create a class Employee with attributes: Ename, Eid, and Esalary.
* Implement a constructor to initialize these attributes.

**Step 2: Implement Create Function**

* **Input**: A vector Employees to hold employee objects, and an integer n for the number of employees.
* **Process**:
  + Loop n times to gather employee information (name, ID, and salary).
  + For each employee, create an Employee object and add it to the Employees vector.
* **Output**: Vector of employees populated with n employee records.

**Step 3: Implement sort Function**

* **Input**: Vector Employees and integer n.
* **Process**:
  + Sort employees by Esalary in ascending order using a bubble sort algorithm.
  + Compare salaries of adjacent employees and swap if needed to maintain order.
* **Output**: Vector Employees sorted by salary.

**Step 4: Implement insert Function**

* **Input**: Vector Employees.
* **Process**:
  + Gather details for a new employee (name, ID, salary).
  + Create a new Employee object and add it to the end of Employees.
* **Output**: Vector Employees updated with the new employee.

**Step 5: Implement delete\_by\_Ename Function**

* **Input**: Vector Employees, integer n.
* **Process**:
  + Take the employee's name to delete as input.
  + Iterate through the vector and check each Employee's Ename.
  + If a match is found, remove that employee from the vector and print a success message.
  + If no match is found, print an error message.
* **Output**: Vector Employees with the employee removed (if found).

**Step 6: Implement delete\_by\_Eid Function**

* **Input**: Vector Employees, integer n.
* **Process**:
  + Take the employee's ID to delete as input.
  + Iterate through the vector and check each Employee's Eid.
  + If a match is found, remove that employee from the vector and print a success message.
  + If no match is found, print an error message.
* **Output**: Vector Employees with the employee removed (if found).

**Step 7: Implement display Function**

* **Input**: Vector Employees.
* **Process**:
  + Loop through each Employee in Employees.
  + Print the employee's Ename, Eid, and Esalary.
* **Output**: Display each employee’s details.

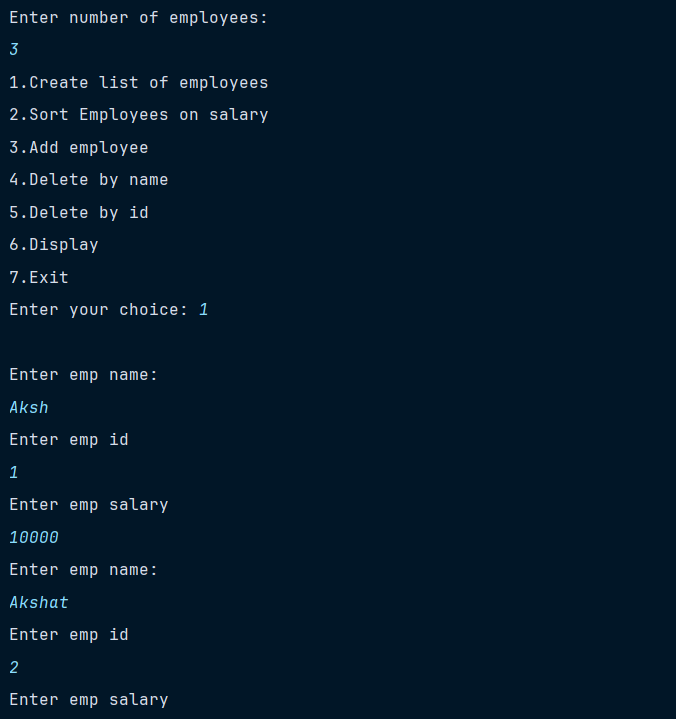
**Step 8: Main Function**

* Initialize a vector Employees.
* Input the number of employees n.
* Display a menu with options to:
  1. Create a list of employees.
  2. Sort employees by salary.
  3. Add a new employee.
  4. Delete an employee by name.
  5. Delete an employee by ID.
  6. Display all employees.
  7. Exit.
* Use a loop to repeatedly display the menu and execute the corresponding function based on the user’s choice.
* **Terminate** the program when the user chooses to exit (choice 7).

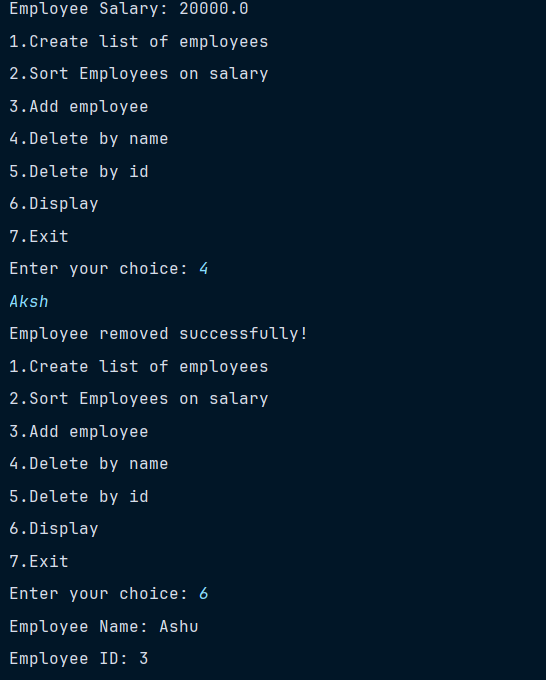
**Implementation details:**

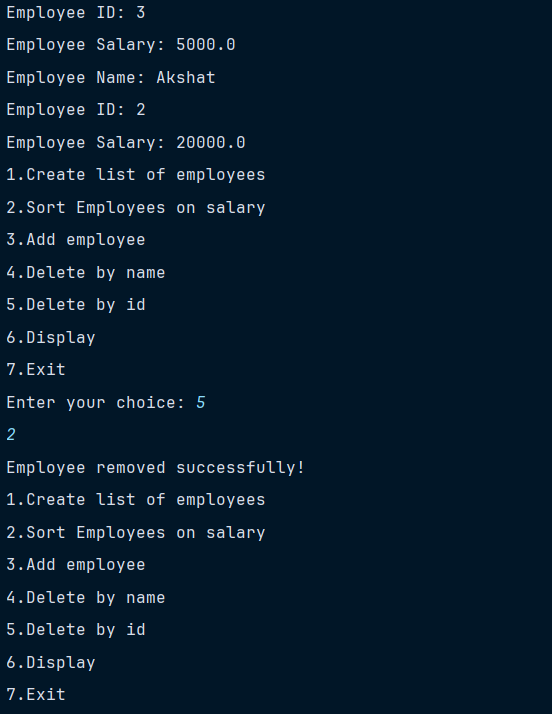
import java.util.\*;  
class Employee  
{  
 String Ename;  
 int Eid;  
 double Esalary;  
  
 Employee(String Ename,int Eid,double Esalary)  
 {  
 this.Ename = Ename;  
 this.Eid = Eid;  
 this.Esalary = Esalary;  
 }  
 static void Create(Vector<Employee>Employees,int n)  
 {  
 Scanner sc = new Scanner(System.in);  
  
 for(int i=0;i<n;i++)  
 {  
 sc.nextLine();  
 System.out.println("Enter emp name: ");  
 String Ename = sc.nextLine();  
 System.out.println("Enter emp id");  
 int Eid = sc.nextInt();  
 sc.nextLine();  
 System.out.println("Enter emp salary");  
 double Esalary = sc.nextDouble();  
  
 Employees.add(new Employee(Ename,Eid,Esalary));  
 }  
  
 }  
 static void sort(Vector<Employee>Employees,int n)  
 {  
 *//Sorting on the basis of salary  
 //Bubble sort* for(int i=0;i<n;i++)  
 {  
 for(int j=i+1;j<n;j++)  
 {  
 if(Employees.get(j).Esalary < Employees.get(i).Esalary)  
 {  
 Employee temp = Employees.get(i);  
 Employees.set(i, Employees.get(j));  
 Employees.set(j,temp);  
 }  
 }  
 }  
 }  
 static void insert(Vector<Employee>Employees)  
 {  
 Scanner sc = new Scanner(System.in);  
 System.out.println("Enter name: ");  
 String Ename = sc.nextLine();  
 System.out.println("Enter ID: ");  
 int Eid = sc.nextInt();  
 sc.nextLine();  
 System.out.println("Enter salary: ");  
 double Esalary = sc.nextDouble();  
 Employees.addElement(new Employee(Ename,Eid,Esalary));  
 }  
 static void delete\_by\_Ename(Vector<Employee>Employees,int n)  
 {  
 Scanner sc = new Scanner(System.in);  
 String temp\_name = sc.nextLine();  
 boolean found = false;  
 for(int i=0;i<n;i++)  
 {  
 if(Employees.get(i).Ename.equalsIgnoreCase(temp\_name))  
 {  
 found = true;  
 Employees.removeElementAt(i);  
 break;  
 }  
 }  
 if(found) System.out.println("Employee removed successfully!");  
  
 else System.out.println("Employee name not found!");  
  
 }  
 static void delete\_by\_Eid(Vector<Employee>Employees,int n)  
 {  
 Scanner sc = new Scanner(System.in);  
 int temp\_id = sc.nextInt();  
 boolean found = false;  
 for(int i=0;i<n;i++)  
 {  
 if(Employees.get(i).Eid == temp\_id)  
 {  
 found = true;  
 Employees.removeElementAt(i);  
 break;  
 }  
 }  
 if(found) System.out.println("Employee removed successfully!");  
  
 else System.out.println("Employee name not found!");  
  
 }  
 static void display(Vector<Employee>Employees)  
 {  
 for(Employee emp : Employees)  
 {  
 System.out.println("Employee Name: "+emp.Ename);  
 System.out.println("Employee ID: "+emp.Eid);  
 System.out.println("Employee Salary: "+emp.Esalary);  
 }  
 }  
  
}  
public class Main  
{  
 public static void main(String[] args)  
 {  
 Scanner sc = new Scanner(System.in);  
 int n;  
 int choice=-1;  
 Vector<Employee> Employees = new Vector<>();  
 System.out.println("Enter number of employees: ");  
 n = sc.nextInt();  
 while(choice!=7)  
 {  
 System.out.println("1.Create list of employees");  
 System.out.println("2.Sort Employees on salary");  
 System.out.println("3.Add employee");  
 System.out.println("4.Delete by name");  
 System.out.println("5.Delete by id");  
 System.out.println("6.Display");  
 System.out.println("7.Exit");  
 System.out.print("Enter your choice: ");  
 choice = sc.nextInt();  
 switch(choice)  
 {  
 case 1:  
  
 Employee.*Create*(Employees,n);  
 break;  
  
 case 2:  
 Employee.*sort*(Employees,n);  
 break;  
  
 case 3:  
 Employee.*insert*(Employees);  
 break;  
  
 case 4:  
 Employee.*delete\_by\_Ename*(Employees,n);  
 break;  
  
 case 5:  
 Employee.*delete\_by\_Eid*(Employees,n);  
 break;  
  
 case 6:  
 Employee.*display*(Employees);  
 break;  
  
 case 7:  
 System.out.println("Exiting the program...");  
 }  
 }  
  
 }  
}

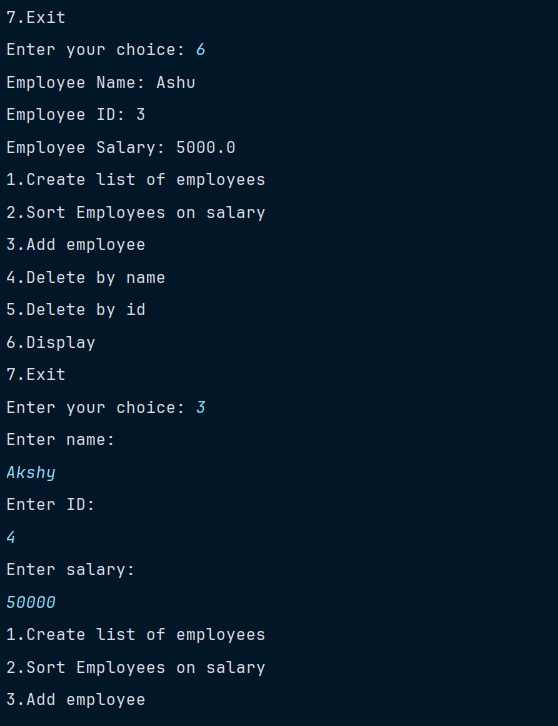
**Output:**

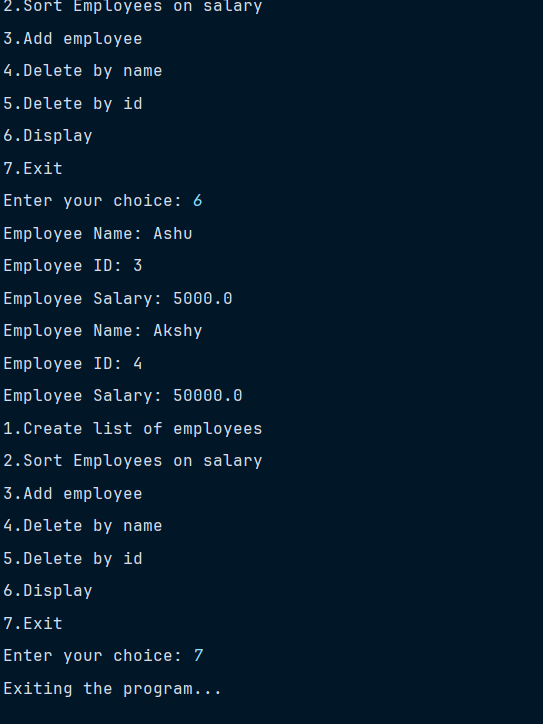












**Conclusion:**

**We learnt about Vectors in java; how they are used to create array of objects. We also learnt about Vector constructors and their working in real time.**

**Date:\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Post Lab Descriptive Questions**

1. **Write a note on the collection framework.**

The Java Collection Framework is a unified architecture that represents and manipulates a group of objects. It provides various classes and interfaces to manage collections of objects in a structured and efficient way. Introduced in Java 2 (JDK 1.2), the Collection Framework simplifies the process of working with data structures by providing pre-implemented and reusable data structures such as lists, sets, and maps.

Key Components of the Collection Framework

1. Interfaces: The Collection Framework defines several core interfaces that represent different types of collections. These interfaces are the foundation of the framework:
   * Collection: The root interface from which most collection interfaces derive.
   * List: Represents an ordered collection (sequence) that allows duplicate elements. Examples: ArrayList, LinkedList.
   * Set: Represents a collection that does not allow duplicate elements. Examples: HashSet, TreeSet.
   * Queue: Represents a collection used to hold multiple elements prior to processing, typically in a FIFO (First-In-First-Out) order. Examples: PriorityQueue, LinkedList.
   * Map: Represents a collection of key-value pairs, where keys are unique. Examples: HashMap, TreeMap.
2. Classes: The Collection Framework provides various concrete classes that implement the collection interfaces. Some of the commonly used classes are:
   * ArrayList: A resizable array implementation of the List interface.
   * LinkedList: A doubly-linked list implementation of both List and Deque interfaces.
   * HashSet: An implementation of the Set interface that uses a hash table for storage.
   * TreeSet: An implementation of the Set interface that stores elements in a sorted tree structure.
   * HashMap: A hash table-based implementation of the Map interface.
   * TreeMap: A red-black tree-based implementation of the Map interface.
3. Algorithms: The framework provides several utility methods in the Collections class, which perform various operations on collections such as sorting, searching, and shuffling.
4. Iterator: An object that enables traversing the elements of a collection. The Iterator interface provides methods like hasNext(), next(), and remove() to iterate through elements.
5. **Explain any 10 methods of Vector class in detail with the help of example**

add(E e)

* Description: Appends the specified element to the end of this Vector.

Example:

Vector<String> vector = new Vector<>();

vector.add("Apple");

vector.add("Banana");

System.out.println(vector); // Output: [Apple, Banana]

add(int index, E element)

* Description: Inserts the specified element at the specified position in this Vector.

Example:

Vector<String> vector = new Vector<>();

vector.add("Apple");

vector.add("Banana");

vector.add(1, "Orange");

System.out.println(vector); // Output: [Apple, Orange, Banana]

get(int index)

* Description: Returns the element at the specified position in this Vector.

Example:

Vector<String> vector = new Vector<>();

vector.add("Apple");

vector.add("Banana");

String fruit = vector.get(1);

System.out.println(fruit); // Output: Banana

remove(int index)

* Description: Removes the element at the specified position in this Vector.

Example:

Vector<String> vector = new Vector<>();

vector.add("Apple");

vector.add("Banana");

vector.remove(0);

System.out.println(vector); // Output: [Banana]

remove(Object o)

* Description: Removes the first occurrence of the specified element in this Vector.

Example:

Vector<String> vector = new Vector<>();

vector.add("Apple");

vector.add("Banana");

vector.add("Apple");

vector.remove("Apple");

System.out.println(vector); // Output: [Banana, Apple]

size()

* Description: Returns the number of components in this Vector.

Example:

Vector<String> vector = new Vector<>();

vector.add("Apple");

vector.add("Banana");

int size = vector.size();

System.out.println(size); // Output: 2

contains(Object o)

* Description: Returns true if this Vector contains the specified element.

Example:

Vector<String> vector = new Vector<>();

vector.add("Apple");

vector.add("Banana");

boolean hasApple = vector.contains("Apple");

System.out.println(hasApple); // Output: true

clear()

* Description: Removes all of the elements from this Vector.

Example:

Vector<String> vector = new Vector<>();

vector.add("Apple");

vector.add("Banana");

vector.clear();

System.out.println(vector); // Output: []

indexOf(Object o)

* Description: Returns the index of the first occurrence of the specified element in this Vector, or -1 if this Vector does not contain the element.

Example:

Vector<String> vector = new Vector<>();

vector.add("Apple");

vector.add("Banana");

int index = vector.indexOf("Banana");

System.out.println(index); // Output: 1

isEmpty()

* Description: Returns true if this Vector contains no elements.

Example:

Vector<String> vector = new Vector<>();

boolean empty = vector.isEmpty();

System.out.println(empty); // Output: true

vector.add("Apple");

empty = vector.isEmpty();

System.out.println(empty); // Output: false

1. **What is an Arraylist? How does it differ from the array?**

| Feature | Array | ArrayList |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| Size | Fixed-size, defined at creation. | Dynamic, resizable automatically. |

|  |  |  |
| --- | --- | --- |
| Performance | Faster for accessing elements. | Slower due to dynamic resizing. |

|  |  |  |
| --- | --- | --- |
| Type Safety | Can hold both primitives and objects. | Can only hold objects (primitives require wrapper classes). |

|  |  |  |
| --- | --- | --- |
| Memory Usage | More memory efficient (no overhead). | Less memory efficient (due to resizing and methods overhead). |

|  |  |  |
| --- | --- | --- |
| Flexibility | Less flexible, no built-in methods for common operations. | More flexible, with many built-in methods (add, remove, etc.). |

|  |  |  |
| --- | --- | --- |
| Generics Support | Does not support generics. | Supports generics for type safety. |

|  |  |  |
| --- | --- | --- |
| Multidimensional | Supports multidimensional arrays (e.g., int[][]). | Supports lists of lists, but not true multidimensional structures. |

|  |  |  |
| --- | --- | --- |
| Element Access | Index-based access. | Index-based access. |

|  |  |  |
| --- | --- | --- |
| Resizing | Cannot be resized once created. | Automatically resizes as needed. |

|  |  |  |
| --- | --- | --- |
| Ease of Use | Requires more manual handling for operations like adding/removing elements. | Provides built-in methods for easier element management. |

1. **Implement a menu driven program for the following:**

**Accepts a shopping list (name, price and quantity)from the command line and stores them in a vector.**

**To delete specific item (given by user) in the vector**

**Add item at the end of the vector**

**Add item at specific location**

**Print the contents of vector using enumeration interface.**

import java.util.Enumeration;

import java.util.Scanner;

import java.util.Vector;

class ShoppingItem {

private String name;

private double price;

private int quantity;

public ShoppingItem(String name, double price, int quantity) {

this.name = name;

this.price = price;

this.quantity = quantity;

}

public String getName() {

return name;

}

@Override

public String toString() {

return "Item: " + name + ", Price: " + price + ", Quantity: " + quantity;

}

}

class ShoppingListManager {

private Vector<ShoppingItem> shoppingList;

public ShoppingListManager() {

shoppingList = new Vector<>();

}

public void addItem(ShoppingItem item) {

shoppingList.add(item);

}

public void addItemAt(int index, ShoppingItem item) {

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

shoppingList.add(index, item);

} else {

System.out.println("Invalid index. Item not added.");

}

}

public void deleteItem(String name) {

shoppingList.removeIf(item -> item.getName().equalsIgnoreCase(name));

}

public void printList() {

Enumeration<ShoppingItem> enumeration = shoppingList.elements();

while (enumeration.hasMoreElements()) {

System.out.println(enumeration.nextElement());

}

}

}

public class ShoppingListApp {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

ShoppingListManager manager = new ShoppingListManager();

while (true) {

System.out.println("\nShopping List Menu");

System.out.println("1. Add Item");

System.out.println("2. Delete Item");

System.out.println("3. Add Item at Specific Location");

System.out.println("4. Print Shopping List");

System.out.println("5. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

scanner.nextLine();

switch (choice) {

case 1:

System.out.print("Enter Item Name: ");

String name = scanner.nextLine();

System.out.print("Enter Item Price: ");

double price = scanner.nextDouble();

System.out.print("Enter Item Quantity: ");

int quantity = scanner.nextInt();

manager.addItem(new ShoppingItem(name, price, quantity));

break;

case 2:

System.out.print("Enter Item Name to Delete: ");

name = scanner.nextLine();

manager.deleteItem(name);

break;

case 3:

System.out.print("Enter Item Name: ");

name = scanner.nextLine();

System.out.print("Enter Item Price: ");

price = scanner.nextDouble();

System.out.print("Enter Item Quantity: ");

quantity = scanner.nextInt();

System.out.print("Enter Position to Insert (0-based index): ");

int index = scanner.nextInt();

manager.addItemAt(index, new ShoppingItem(name, price, quantity));

break;

case 4:

System.out.println("Shopping List Contents:");

manager.printList();

break;

case 5:

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

scanner.close();

System.exit(0);

default:

System.out.println("Invalid choice. Please try again.");

break;

}

}

}

}