Collections

List(ArrayList)

**2. Search an Element**

Write a program to:

* Create an ArrayList of integers.
* Ask the user to enter a number.
* Check if the number exists in the list.

package Assignment8;

import java.util.ArrayList;

import java.util.Scanner;

public class searchelement {

public static void main(String[] args) {

ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(10);

numbers.add(20);

numbers.add(30);

numbers.add(40);

numbers.add(50);

Scanner sc = new Scanner(System.***in***);

System.***out***.print("Enter a number to search: ");

int num = sc.nextInt();

if (numbers.contains(num)) {

System.***out***.println(num + " is present in the list.");

} else {

System.***out***.println(num + " is not present in the list.");

}

sc.close();

}

}

Output:10 is present in the list.

**3. Remove Specific Element**

Write a program to:

* Create an ArrayList of Strings.
* Add 5 fruits.
* Remove a specific fruit by name.
* Display the updated list.

package Assignment8;

import java.util.ArrayList;

import java.util.Scanner;

public class removefruit {

public static void main(String[] args) {

ArrayList<String> fruits = new ArrayList<>();

fruits.add("Apple");

fruits.add("Banana");

fruits.add("Orange");

fruits.add("Mango");

fruits.add("Grapes");

Scanner sc = new Scanner(System.***in***);

System.***out***.print("Enter fruit name to remove: ");

String fruit = sc.nextLine();

if (fruits.remove(fruit)) {

System.***out***.println(fruit + " removed successfully.");

} else {

System.***out***.println(fruit + " not found in the list.");

}

System.***out***.println("Updated list: " + fruits);

sc.close();

}

}

Output: Enter fruit name to remove: Apple

Apple removed successfully.

Updated list: [Banana, Orange, Mango, Grapes]

**4. Sort Elements**

Write a program to:

* Create an ArrayList of integers.
* Add at least 7 random numbers.
* Sort the list in ascending order.
* Display the sorted list.

package Assignment8;

import java.util.ArrayList;

import java.util.Collections;

public class sortnumbers {

public static void main(String[] args) {

ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(45);

numbers.add(12);

numbers.add(78);

numbers.add(23);

numbers.add(9);

numbers.add(56);

numbers.add(31);

Collections.*sort*(numbers);

System.***out***.println("Sorted list (Ascending): " + numbers);

}

}

Output: Sorted list (Ascending): [9, 12, 23, 31, 45, 56, 78]

**5. Reverse the ArrayList**

Write a program to:

* Create an ArrayList of characters.
* Add 5 characters.
* Reverse the list using Collections.reverse() and display it.

package Assignment8;

import java.util.ArrayList;

import java.util.Collections;

public class reverselist {

public static void main(String[] args) {

ArrayList<Character> chars = new ArrayList<>();

chars.add('A');

chars.add('B');

chars.add('C');

chars.add('D');

chars.add('E');

Collections.*reverse*(chars);

System.***out***.println("Reversed list: " + chars);

}

}

Output: Reversed list: [E, D, C, B, A]

**6. Update an Element**

Write a program to:

* Create an ArrayList of subjects.
* Replace one of the subjects (e.g., “Math” to “Statistics”).
* Print the list before and after the update.

package Assignment8;

import java.util.ArrayList;

public class updateelement {

public static void main(String[] args) {

ArrayList<String> subjects = new ArrayList<>();

subjects.add("Math");

subjects.add("Science");

subjects.add("English");

subjects.add("History");

System.***out***.println("Before update: " + subjects);

int index = subjects.indexOf("Math");

if (index != -1) {

subjects.set(index, "Statistics");

}

System.***out***.println("After update: " + subjects);

}

}

Output: Before update: [Math, Science, English, History]

After update: [Statistics, Science, English, History]

**7. Remove All Elements**

Write a program to:

* Create an ArrayList of integers.
* Add multiple elements.
* Remove all elements using clear() method.
* Display the size of the list.

package Assignment8;

import java.util.ArrayList;

public class removeallelements {

public static void main(String[] args) {

ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(10);

numbers.add(20);

numbers.add(30);

numbers.add(40);

System.***out***.println("Before clear: " + numbers);

numbers.clear();

System.***out***.println("After clear: " + numbers);

System.***out***.println("Size of list: " + numbers.size());

}

}

Output: Before clear: [10, 20, 30, 40]

After clear: []

Size of list: 0

**8. Iterate using Iterator**

Write a program to:

* Create an ArrayList of cities.
* Use Iterator to display each city.

package Assignment8;

import java.util.ArrayList;

import java.util.Iterator;

public class iterateusingiterartor {

public static void main(String[] args) {

ArrayList<String> cities = new ArrayList<>();

cities.add("Delhi");

cities.add("Mumbai");

cities.add("Chennai");

cities.add("Kolkata");

Iterator<String> itr = cities.iterator();

while (itr.hasNext()) {

System.***out***.println(itr.next());

}

}

}

Output: Delhi

Mumbai

Chennai

Kolkata

**9. Store Custom Objects**

Write a program to:

* Create a class Student with fields: id, name, and marks.
* Create an ArrayList of Student objects.
* Add at least 3 students.
* Display the details using a loop.

package Assignment8;

import java.util.ArrayList;

class Student {

int id;

String name;

double marks;

Student(int id, String name, double marks) {

this.id = id;

this.name = name;

this.marks = marks;

}

}

public class storecustomobj {

public static void main(String[] args) {

ArrayList<Student> students = new ArrayList<>();

students.add(new Student(1, "Ravi", 85.5));

students.add(new Student(2, "Priya", 90.0));

students.add(new Student(3, "Amit", 78.3));

for (Student s : students) {

System.***out***.println("ID: " + s.id + ", Name: " + s.name + ", Marks: " + s.marks);

}

}

}

Output: ID: 1, Name: Ravi, Marks: 85.5

ID: 2, Name: Priya, Marks: 90.0

ID: 3, Name: Amit, Marks: 78.3

**10. Copy One ArrayList to Another**

Write a program to:

* Create an ArrayList with some elements.
* Create a second ArrayList.
* Copy all elements from the first to the second using addAll() method.

package Assignment8;

import java.util.ArrayList;

public class copyarraylist {

public static void main(String[] args) {

ArrayList<String> list1 = new ArrayList<>();

list1.add("Apple");

list1.add("Banana");

list1.add("Cherry");

ArrayList<String> list2 = new ArrayList<>();

list2.addAll(list1);

System.***out***.println("First List: " + list1);

System.***out***.println("Second List (Copied): " + list2);

}

}

Output: First List: [Apple, Banana, Cherry]

Second List (Copied): [Apple, Banana, Cherry]

List(LinkedList)

**1. Create and Display a LinkedList**

Write a program to:

* Create a LinkedList of Strings.
* Add five colors to it.
* Display the list using a for-each loop.

package Assignment8;

import java.util.LinkedList;

public class linkedlistcolor {

public static void main(String[] args) {

LinkedList<String> colors = new LinkedList<>();

colors.add("Red");

colors.add("Green");

colors.add("Blue");

colors.add("Yellow");

colors.add("Purple");

System.***out***.println("Colors in the LinkedList:");

for (String color : colors) {

System.***out***.println(color);

}

}

}

Output: Colors in the LinkedList:

Red

Green

Blue

Yellow

Purple

**2. Add Elements at First and Last Position**

Write a program to:

* Create a LinkedList of integers.
* Add elements at the beginning and at the end.
* Display the updated list.
* package Assignment8;
* import java.util.LinkedList;
* public class addfirstlast {
* public static void main(String[] args) {
* LinkedList<Integer> numbers = new LinkedList<>();
* numbers.add(10);
* numbers.add(20);
* numbers.add(30);
* numbers.addFirst(5);
* numbers.addLast(40);
* System.***out***.println("Updated LinkedList: " + numbers);
* }
* }
* Output: Updated LinkedList: [5, 10, 20, 30, 40]

**3. Insert Element at Specific Position**

Write a program to:

* Create a LinkedList of names.
* Insert a name at index 2.
* Display the list before and after insertion.

package Assignment8;

import java.util.LinkedList;

public class insertatindex {

public static void main(String[] args) {

LinkedList<String> names = new LinkedList<>();

names.add("Alice");

names.add("Bob");

names.add("Charlie");

System.***out***.println("Before Insertion: " + names);

names.add(2, "David");

System.***out***.println("After Insertion: " + names);

}

}

Output: Before Insertion: [Alice, Bob, Charlie]

After Insertion: [Alice, Bob, David, Charlie]

**4. Remove Elements**

Write a program to:

* Create a LinkedList of animal names.
* Remove the first and last elements.
* Remove a specific element by value.
* Display the list after each removal.

package Assignment8;

import java.util.LinkedList;

public class removeele {

public static void main(String[] args) {

LinkedList<String> animals = new LinkedList<>();

animals.add("Dog");

animals.add("Cat");

animals.add("Elephant");

animals.add("Tiger");

animals.add("Lion");

System.***out***.println("Original List: " + animals);

animals.removeFirst();

System.***out***.println("After removing first: " + animals);

animals.removeLast();

System.***out***.println("After removing last: " + animals);

animals.remove("Elephant");

System.***out***.println("After removing 'Elephant': " + animals);

}

}

Output: Original List: [Dog, Cat, Elephant, Tiger, Lion]

After removing first: [Cat, Elephant, Tiger, Lion]

After removing last: [Cat, Elephant, Tiger]

After removing 'Elephant': [Cat, Tiger]

**5. Search for an Element**

Write a program to:

* Create a LinkedList of Strings.
* Ask the user for a string to search.
* Display if the string is found or not.

package Assignment8;

import java.util.LinkedList;

import java.util.Scanner;

public class searchelement1 {

public static void main(String[] args) {

LinkedList<String> list = new LinkedList<>();

list.add("Apple");

list.add("Banana");

list.add("Mango");

list.add("Orange");

list.add("Grapes");

Scanner sc = new Scanner(System.***in***);

System.***out***.print("Enter a fruit to search: ");

String fruit = sc.nextLine();

if (list.contains(fruit)) {

System.***out***.println(fruit + " is found in the list.");

} else {

System.***out***.println(fruit + " is not found in the list.");

}

sc.close();

}

}

Output: Enter a fruit to search: Apple

Apple is found in the list.

**6. Iterate using ListIterator**

Write a program to:

* Create a LinkedList of cities.
* Use ListIterator to display the list in both forward and reverse directions.

package Assignment8;

import java.util.LinkedList;

import java.util.ListIterator;

public class listiterator1 {

public static void main(String[] args) {

LinkedList<String> cities = new LinkedList<>();

cities.add("New York");

cities.add("London");

cities.add("Paris");

cities.add("Tokyo");

cities.add("Sydney");

ListIterator<String> it = cities.listIterator();

System.***out***.println("Forward direction:");

while (it.hasNext()) {

System.***out***.println(it.next());

}

System.***out***.println("Reverse direction:");

while (it.hasPrevious()) {

System.***out***.println(it.previous());

}

}

}

Output: Forward direction:

New York

London

Paris

Tokyo

Sydney

Reverse direction:

Sydney

Tokyo

Paris

London

New York

**7. Sort a LinkedList**

Write a program to:

* Create a LinkedList of integers.
* Add unsorted numbers.
* Sort the list using Collections.sort().
* Display the sorted list.

package Assignment8;

import java.util.Collections;

import java.util.LinkedList;

public class sortlinkedlist {

public static void main(String[] args) {

LinkedList<Integer> numbers = new LinkedList<>();

numbers.add(25);

numbers.add(5);

numbers.add(15);

numbers.add(40);

numbers.add(10);

Collections.*sort*(numbers);

System.***out***.println("Sorted list: " + numbers);

}

}

Output: Sorted list: [5, 10, 15, 25, 40]

**8. Convert LinkedList to ArrayList**

Write a program to:

* Create a LinkedList of Strings.
* Convert it into an ArrayList.
* Display both the LinkedList and ArrayList.

package Assignment8;

import java.util.ArrayList;

import java.util.LinkedList;

public class linkedlisttoarraylist {

public static void main(String[] args) {

LinkedList<String> list = new LinkedList<>();

list.add("Apple");

list.add("Banana");

list.add("Cherry");

ArrayList<String> arrayList = new ArrayList<>(list);

System.***out***.println("LinkedList: " + list);

System.***out***.println("ArrayList: " + arrayList);

}

}

Output: LinkedList: [Apple, Banana, Cherry]

ArrayList: [Apple, Banana, Cherry]

**9. Store Custom Objects in LinkedList**

Write a program to:

* Create a class Book with fields: id, title, and author.
* Create a LinkedList of Book objects.
* Add 3 books and display their details using a loop.

package Assignment8;

import java.util.LinkedList;

class Book {

int id;

String title;

String author;

Book(int id, String title, String author) {

this.id = id;

this.title = title;

this.author = author;

}

}

public class booklinkedlist {

public static void main(String[] args) {

LinkedList<Book> books = new LinkedList<>();

books.add(new Book(1, "Java Basics", "James"));

books.add(new Book(2, "Python Guide", "Guido"));

books.add(new Book(3, "C++ Primer", "Bjarne"));

for (Book b : books) {

System.***out***.println(b.id + " " + b.title + " " + b.author);

}

}

}

Output: 1 Java Basics James

2 Python Guide Guido

3 C++ Primer Bjarne

**10. Clone a LinkedList**

Write a program to:

* Create a LinkedList of numbers.
* Clone it using the clone() method.
* Display both original and cloned lists.
* package Assignment8;
* import java.util.LinkedList;
* public class clonelinkedlist {
* public static void main(String[] args) {
* LinkedList<Integer> numbers = new LinkedList<>();
* numbers.add(10);
* numbers.add(20);
* numbers.add(30);
* LinkedList<Integer> clonedList = (LinkedList<Integer>) numbers.clone();
* System.***out***.println("Original List: " + numbers);
* System.***out***.println("Cloned List: " + clonedList);
* }
* }
* Output: Original List: [10, 20, 30]
* Cloned List: [10, 20, 30]

Vector

* **Create a Vector of integers** and perform the following operations:
* Add 5 integers to the Vector.
* Insert an element at the 3rd position.
* Remove the 2nd element.
* Display the elements using Enumeration.

package Assignment8;

import java.util.Vector;

import java.util.Enumeration;

public class vectorexamp {

public static void main(String[] args) {

Vector<Integer> numbers = new Vector<>();

numbers.add(10);

numbers.add(20);

numbers.add(30);

numbers.add(40);

numbers.add(50);

numbers.add(2, 25);

numbers.remove(1);

Enumeration<Integer> e = numbers.elements();

while (e.hasMoreElements()) {

System.***out***.println(e.nextElement());

}

}

}

Output: 10

25

30

40

50

* **Create a Vector of Strings** and:
* Add at least 4 names.
* Check if a specific name exists in the vector.
* Replace one name with another.
* Clear all elements from the vector.
* **Write a program** to:
* Copy all elements from one Vector to another Vector.
* Compare both vectors for equality.
* **Write a method** that takes a Vector<Integer> and returns the **sum of all elements**.

package Assignment8;

import java.util.Vector;

public class vectortasks {

public static void main(String[] args) {

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

names.add("Alice");

names.add("Bob");

names.add("Charlie");

names.add("David");

System.***out***.println("Contains Bob? " + names.contains("Bob"));

names.set(1, "Brian");

System.***out***.println("After replacement: " + names);

names.clear();

System.***out***.println("After clear: " + names);

Vector<String> v1 = new Vector<>();

v1.add("A");

v1.add("B");

Vector<String> v2 = new Vector<>();

v2.addAll(v1);

System.***out***.println("v1 equals v2? " + v1.equals(v2));

Vector<Integer> nums = new Vector<>();

nums.add(10);

nums.add(20);

nums.add(30);

System.***out***.println("Sum = " + *sumVector*(nums));

}

public static int sumVector(Vector<Integer> v) {

int sum = 0;

for (int n : v) sum += n;

return sum;

}

}

Output: Contains Bob? true

After replacement: [Alice, Brian, Charlie, David]

After clear: []

v1 equals v2? true

Sum = 60

**Stack**

* Understand how to use the Stack class for LIFO (Last In, First Out) operations.
* **Create a Stack of integers** and:
* Push 5 elements.
* Pop the top element.
* Peek the current top.
* Check if the stack is empty.

package Assignment8;

import java.util.Stack;

public class stackemp {

public static void main(String[] args) {

Stack<Integer> stack = new Stack<>();

stack.push(10);

stack.push(20);

stack.push(30);

stack.push(40);

stack.push(50);

System.***out***.println("Stack after pushes: " + stack);

stack.pop();

System.***out***.println("After pop: " + stack);

System.***out***.println("Top element (peek): " + stack.peek());

System.***out***.println("Is stack empty? " + stack.isEmpty());

}

}

Output: Stack after pushes: [10, 20, 30, 40, 50]

After pop: [10, 20, 30, 40]

Top element (peek): 40

Is stack empty? false

* **Reverse a string using Stack**:
* Input a string from the user.
* Use a stack to reverse and print the string.

package Assignment8;

import java.util.\*;

public class revstr {

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

System.***out***.print("Enter a string: ");

String input = sc.nextLine();

Stack<Character> stack = new Stack<>();

for (char ch : input.toCharArray()) {

stack.push(ch);

}

String reversed = "";

while (!stack.isEmpty()) {

reversed += stack.pop();

}

System.***out***.println("Reversed string: " + reversed);

}

}

Output: Enter a string: apple

Reversed string: elppa

* **Use Stack to check for balanced parentheses** in an expression.
* Input: (a+b) \* (c-d)
* Output: Valid or Invalid expression

package Assignment8;

import java.util.\*;

public class balanced\_paren {

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

System.***out***.print("Enter expression: ");

String expr = sc.nextLine();

Stack<Character> stack = new Stack<>();

boolean isValid = true;

for (char ch : expr.toCharArray()) {

if (ch == '(') {

stack.push(ch);

} else if (ch == ')') {

if (stack.isEmpty()) {

isValid = false;

break;

}

stack.pop();

}

}

if (!stack.isEmpty()) isValid = false;

System.***out***.println(isValid ? "Valid" : "Invalid");

}

}

Output: Enter expression: (a+b) \* (c-d)

Valid

* **Convert a decimal number to binary using Stack**.

package Assignment8;

import java.util.\*;

public class dec\_tobin {

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

System.***out***.print("Enter decimal number: ");

int num = sc.nextInt();

Stack<Integer> stack = new Stack<>();

while (num > 0) {

stack.push(num % 2);

num /= 2;

}

System.***out***.print("Binary: ");

while (!stack.isEmpty()) {

System.***out***.print(stack.pop());

}

}

}

Output: Enter decimal number: 2

Binary: 10

HashSet

1. **Create a HashSet of Strings**:
   * Add 5 different city names.
   * Try adding a duplicate city and observe the output.
   * Iterate using an Iterator and print each city.

package Assignment8;

import java.util.\*;

public class hashset\_emp {

public static void main(String[] args) {

HashSet<String> cities = new HashSet<>();

cities.add("Delhi");

cities.add("Mumbai");

cities.add("Chennai");

cities.add("Kolkata");

cities.add("Bangalore");

cities.add("Mumbai");

Iterator<String> itr = cities.iterator();

while (itr.hasNext()) {

System.***out***.println(itr.next());

}

}

}

Output: Delhi

Chennai

Kolkata

Mumbai

Bangalore

1. **Perform operations**:
   * Remove an element.
   * Check if a city exists.
   * Clear the entire HashSet.

package Assignment8;

import java.util.\*;

public class hashsetops {

public static void main(String[] args) {

HashSet<String> cities = new HashSet<>();

cities.add("Delhi");

cities.add("Mumbai");

cities.add("Chennai");

cities.add("Kolkata");

cities.add("Bangalore");

cities.remove("Chennai");

System.***out***.println("Contains Mumbai? " + cities.contains("Mumbai"));

cities.clear();

System.***out***.println("HashSet after clear: " + cities);

}

}

Output: Contains Mumbai? true

HashSet after clear: []

1. **Write a method** that takes a HashSet<Integer> and returns the maximum element.

package Assignment8;

import java.util.\*;

public class Hashsetmax {

public static int getMax(HashSet<Integer> set) {

return Collections.*max*(set);

}

public static void main(String[] args) {

HashSet<Integer> numbers = new HashSet<>();

numbers.add(10);

numbers.add(25);

numbers.add(5);

numbers.add(40);

numbers.add(15);

System.***out***.println("Max: " + *getMax*(numbers));

}

}

Output: Max: 40

**LinkedHashSet**

**1.Create a LinkedHashSet of Integers**:

* + Add numbers: 10, 5, 20, 15, 5.
  + Print the elements and observe the order.

package Assignment8;

import java.util.\*;

public class linkedhashset {

public static void main(String[] args) {

LinkedHashSet<Integer> numbers = new LinkedHashSet<>();

numbers.add(10);

numbers.add(5);

numbers.add(20);

numbers.add(15);

numbers.add(5); // duplicate

System.***out***.println("LinkedHashSet elements: " + numbers);

}

}

Output: LinkedHashSet elements: [10, 5, 20, 15]

1. **Create a LinkedHashSet of custom objects (e.g., Student with id and name)**:
   * Override hashCode() and equals() properly.
   * Add at least 3 Student objects.
   * Try adding a duplicate student and check if it gets added.

package Assignment8;

import java.util.\*;

class Student{

int id;

String name;

Student(int id,String name){

this.id=id;

this.name=name;

}

public int hashCode(){

return Objects.*hash*(id,name);

}

public boolean equals(Object obj){

if(this==obj)return true;

if(!(obj instanceof Student))return false;

Student s=(Student)obj;

return id==s.id&&name.equals(s.name);

}

public String toString(){

return id+"-"+name;

}

}

public class lhs\_custom{

public static void main(String[] args){

LinkedHashSet<Student> students=new LinkedHashSet<>();

students.add(new Student(1,"Ravi"));

students.add(new Student(2,"Priya"));

students.add(new Student(3,"Amit"));

students.add(new Student(1,"Ravi"));

System.***out***.println("Students in LinkedHashSet:"+students);

}

}

Output: Students in LinkedHashSet:[1-Ravi, 2-Priya, 3-Amit]

1. **Write a program** to:
   * Merge two LinkedHashSets and print the result.

package Assignment8;

import java.util.\*;

public class merge\_lhs{

public static void main(String[] args){

LinkedHashSet<Integer> set1=new LinkedHashSet<>();

set1.add(1);

set1.add(2);

set1.add(3);

LinkedHashSet<Integer> set2=new LinkedHashSet<>();

set2.add(3);

set2.add(4);

set2.add(5);

set1.addAll(set2);

System.***out***.println("Merged LinkedHashSet:"+set1);

}

}

Output: Merged LinkedHashSet:[1, 2, 3, 4, 5]

**TreeSet**

**1. Create a TreeSet of Strings**:

* + Add 5 country names in random order.
  + Print the sorted list of countries using TreeSet.

package Assignment8;

import java.util.\*;

public class treesetstr{

public static void main(String[] args){

TreeSet<String> countries=new TreeSet<>();

countries.add("India");

countries.add("USA");

countries.add("Japan");

countries.add("Canada");

countries.add("Brazil");

System.***out***.println(countries);

}

}

Output: [Brazil, Canada, India, Japan, USA]

1. **Create a TreeSet of Integers**:
   * Add some numbers and print the first and last elements.
   * Find the elements lower than and higher than a given number using lower() and higher() methods.

package Assignment8;

import java.util.\*;

public class treesetint{

public static void main(String[] args){

TreeSet<Integer> numbers=new TreeSet<>();

numbers.add(10);

numbers.add(5);

numbers.add(20);

numbers.add(15);

System.***out***.println("First:"+numbers.first());

System.***out***.println("Last:"+numbers.last());

System.***out***.println("Lower than 15:"+numbers.lower(15));

System.***out***.println("Higher than 15:"+numbers.higher(15));

}

}

Output: First:5

Last:20

Lower than 15:10

Higher than 15:20

1. **Create a TreeSet with a custom comparator**:
   * Sort strings in **reverse alphabetical order** using Comparator.

package Assignment8;

import java.util.\*;

public class treeset\_rev{

public static void main(String[] args){

TreeSet<String> set=new TreeSet<>(Comparator.*reverseOrder*());

set.add("Apple");

set.add("Banana");

set.add("Mango");

set.add("Orange");

System.***out***.println(set);

}

}

Output: [Orange, Mango, Banana, Apple]

Queue

1. **Bank Queue Simulation**:
   * Create a queue of customer names using Queue<String>.
   * Add 5 customers to the queue.
   * Serve (remove) customers one by one and print the queue after each removal.

package Assignment8;

import java.util.\*;

public class bankqueue{

public static void main(String[] args){

Queue<String> queue=new LinkedList<>();

queue.add("Alice");

queue.add("Bob");

queue.add("Charlie");

queue.add("David");

queue.add("Eva");

while(!queue.isEmpty()){

System.***out***.println("Serving:"+queue.poll());

System.***out***.println(queue);

}

}

}

Output: Serving:Alice

[Bob, Charlie, David, Eva]

Serving:Bob

[Charlie, David, Eva]

Serving:Charlie

[David, Eva]

Serving:David

[Eva]

Serving:Eva

[]

1. **Task Manager**:
   * Queue of tasks (String values).
   * Add tasks, peek at the next task, and poll completed tasks.

package Assignment8;

import java.util.\*;

public class taskmanager{

public static void main(String[] args){

Queue<String> tasks=new LinkedList<>();

tasks.add("Task1");

tasks.add("Task2");

tasks.add("Task3");

System.***out***.println("Next:"+tasks.peek());

tasks.poll();

System.***out***.println(tasks);

}

}

Output: Next:Task1

[Task2, Task3]

1. **Write a method**:
   * That takes a queue of integers and returns a list of even numbers.

package Assignment8;

import java.util.\*;

public class evennumqueue{

public static List<Integer> getEvenNumbers(Queue<Integer> queue){

List<Integer> evens=new ArrayList<>();

for(int num:queue){

if(num%2==0)evens.add(num);

}

return evens;

}

public static void main(String[] args){

Queue<Integer> nums=new LinkedList<>();

nums.add(1);

nums.add(2);

nums.add(3);

nums.add(4);

nums.add(6);

System.***out***.println(*getEvenNumbers*(nums));

}

}

Output: [2, 4, 6]

**PriorityQueue**

1. **Hospital Emergency Queue**:
   * Create a class Patient with fields: name and severityLevel (int).
   * Use PriorityQueue<Patient> with a comparator to serve the most critical patients first (highest severityLevel).

package Assignment8;

import java.util.\*;

class Patient{

String name;

int severityLevel;

Patient(String name,int severityLevel){

this.name=name;

this.severityLevel=severityLevel;

}

public String toString(){

return name+"("+severityLevel+")";

}

}

public class hosp\_queue{

public static void main(String[] args){

PriorityQueue<Patient> pq=new PriorityQueue<>((a,b)->b.severityLevel-a.severityLevel);

pq.add(new Patient("Alice",5));

pq.add(new Patient("Bob",2));

pq.add(new Patient("Charlie",4));

while(!pq.isEmpty()){

System.***out***.println("Serving:"+pq.poll());

}

}

}

Output: Serving:Alice(5)

Serving:Charlie(4)

Serving:Bob(2)

1. **Print Jobs Priority**:
   * Add different print jobs (String) with priority levels.
   * Use PriorityQueue to simulate serving high-priority jobs before others.

package Assignment8;

import java.util.\*;

class PrintJob{

String jobName;

int priority;

PrintJob(String jobName,int priority){

this.jobName=jobName;

this.priority=priority;

}

public String toString(){

return jobName+"("+priority+")";

}

}

public class priorityqueue{

public static void main(String[] args){

PriorityQueue<PrintJob> pq=new PriorityQueue<>((a,b)->b.priority-a.priority);

pq.add(new PrintJob("Job1",2));

pq.add(new PrintJob("Job2",5));

pq.add(new PrintJob("Job3",3));

while(!pq.isEmpty()){

System.***out***.println("Printing:"+pq.poll());

}

}

}

Output: Printing:Job2(5)

Printing:Job3(3)

Printing:Job1(2)

1. **Write a method**:
   * To merge two PriorityQueue<Integer> and return a sorted merged queue.

package Assignment8;

import java.util.\*;

public class merge\_pq{

public static PriorityQueue<Integer> mergeQueues(PriorityQueue<Integer> q1,PriorityQueue<Integer> q2){

PriorityQueue<Integer> merged=new PriorityQueue<>(q1);

merged.addAll(q2);

return merged;

}

public static void main(String[] args){

PriorityQueue<Integer> q1=new PriorityQueue<>(Arrays.*asList*(1,3,5));

PriorityQueue<Integer> q2=new PriorityQueue<>(Arrays.*asList*(2,4,6));

System.***out***.println(*mergeQueues*(q1,q2));

}

}

Output: [1, 2, 5, 3, 4, 6]

**Deque**

1. **Palindrome Checker**:
   * Input a string and check if it is a palindrome using a Deque<Character>.

package Assignment8;

import java.util.\*;

public class palindrome\_check{

public static void main(String[] args){

String str="madam";

Deque<Character> dq=new ArrayDeque<>();

for(char c:str.toCharArray())dq.add(c);

boolean pal=true;

while(dq.size()>1){

if(dq.removeFirst()!=dq.removeLast()){pal=false;break;}

}

System.***out***.println(pal?"Palindrome":"Not Palindrome");

}

}

Output: Palindrome

1. **Double-ended Order System**:
   * Add items from front and rear.
   * Remove items from both ends.
   * Display contents of the deque after each operation.

package Assignment8;

import java.util.\*;

public class doubleendeddorder{

public static void main(String[] args){

Deque<String> dq=new ArrayDeque<>();

dq.addFirst("A");

dq.addLast("B");

System.***out***.println(dq);

dq.addFirst("C");

System.***out***.println(dq);

dq.removeFirst();

System.***out***.println(dq);

dq.removeLast();

System.***out***.println(dq);

}

}

Output: [A, B]

[C, A, B]

[A, B]

[A]

1. **Browser History Simulation**:
   * Implement browser back and forward navigation using two deques.

package Assignment8;

import java.util.\*;

public class browser\_history{

public static void main(String[] args){

Deque<String> back=new ArrayDeque<>();

Deque<String> forward=new ArrayDeque<>();

String current="Home";

back.push(current);

current="Page1";

back.push(current);

current="Page2";

back.push(current);

forward.push(back.pop());

current=back.peek();

System.***out***.println("Back to:"+current);

back.push(forward.pop());

System.***out***.println("Forward to:"+back.peek());

}

}

Output: Back to:Page1

Forward to:Page2