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 Assignment_Day8;

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

public class Problem1 {
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 scanner = new Scanner(System.in);
System.out.print("Enter a number to search in the list: ");
int searchNumber = scanner.nextInt();
if (numbers.contains(searchNumber)) {
System.out.println(searchNumber + " is present in the list.");
} else {
System.out.println(searchNumber + " is not present in the list.");
}
}
}
```

```
Enter a number to search in the list: 200
200 is not 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 Assignment_Day8;

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

public class Problem3 {
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");
System.out.println("Original Fruit List: " + fruits);
Scanner scanner = new Scanner(System.in);
System.out.print("Enter fruit name to remove: ");
String fruitToRemove = scanner.nextLine();
if (fruits.remove(fruitToRemove)) {
System.out.println(fruitToRemove + " removed successfully!");
} else {
System.out.println(fruitToRemove + " not found in the list.");
}
System.out.println("Updated Fruit List: " + fruits);
scanner.close();
}
}
```

```
Original Fruit List: [Apple, Banana, Orange, Mango, Grapes]
Enter fruit name to remove: Apple
Apple removed successfully!
Updated Fruit 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 Assignment_Day8;

import java.util.ArrayList;
import java.util.Collections;

public class Problem4 {
public static void main(String[] args) {

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

numbers.add(34);
numbers.add(12);
numbers.add(89);
numbers.add(5);
numbers.add(42);
numbers.add(17);
numbers.add(63);

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

Collections.sort(numbers);
System.out.println("Sorted List" + numbers);
}
}
```

```
Original List: [34, 12, 89, 5, 42, 17, 63]
Sorted List[5, 12, 17, 34, 42, 63, 89]
```

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.

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 Assignment_Day8;

import java.util.ArrayList;

public class Problem6 {
public static void main(String[] args) {
ArrayList<String> subjects = new ArrayList<>();
```

```
subjects.add("Math");
subjects.add("Science");
subjects.add("History");
subjects.add("English");
subjects.add("Art");
System.out.println("Original List: " + subjects);
System.out.println("Original Size: " + subjects.size());
int index = subjects.indexOf("Math");
if(index != -1) {
subjects.set(index, "Statistics");
System.out.println("\nAfter updating Math to Statistics:");
System.out.println("Updated List: " + subjects);
System.out.println("Size remains: " + subjects.size());
}
subjects.clear();
System.out.println("\nAfter clear():");
System.out.println("List content: " + subjects);
System.out.println("Size after clear: " + subjects.size());
}
}
```

```
Original List: [Math, Science, History, English, Art]
Original Size: 5

After updating Math to Statistics:
Updated List: [Statistics, Science, History, English, Art]
Size remains: 5

After clear():
List content: []
Size after clear: 0
```

8. Iterate using Iterator

Write a program to:

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

```
package Assignment_Day8;
```

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

public class Problem7 {
public static void main(String[] args) {

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

cities.add("New York");
cities.add("London");
cities.add("Tokyo");
cities.add("Paris");
cities.add("Sydney");
Iterator<String> iterator = cities.iterator();
System.out.println("Cities in the list:");
while(iterator.hasNext()) {
String city = iterator.next();
System.out.println(city);
}
}
}
```

```
Cities in the list:
New York
London
Tokyo
Paris
Sydney
```

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.

```java
package Assignment_Day8;
import java.util.ArrayList;

class Student {
private int id;
private String name;
private double marks;

public Student(int id, String name, double marks) {
this.id = id;
this.name = name;
this.marks = marks;
}



public int getId() { return id; }
public String getName() { return name; }
public double getMarks() { return marks; }

@Override
public String toString() {
return "Student [ID=" + id + ", Name=" + name + ", Marks=" + marks + "]";
}
}

public class Problem9 {
public static void main(String[] args) {
ArrayList<Student> students = new ArrayList<>();
students.add(new Student(101, "Sai", 87.5));
students.add(new Student(102, "Eswar", 92.0));
students.add(new Student(103, "Kumar", 78.3));

System.out.println("Student Details:");
for (Student student : students) {
System.out.println(student);
}
}
}
```

```
Student Details:
Student [ID=101, Name=Sai, Marks=87.5]
Student [ID=102, Name=Eswar, Marks=92.0]
Student [ID=103, Name=Kumar, Marks=78.3]
```

```java
package Assignment_Day8;

import java.util.ArrayList;

public class Problem10 {
public static void main(String[] args) {

ArrayList<String> originalList = new ArrayList<>();
originalList.add("Apple");
originalList.add("Banana");
originalList.add("Orange");
originalList.add("Mango");
System.out.println("Original ArrayList: " + originalList);

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

copiedList.addAll(originalList);
System.out.println("Copied ArrayList: " + copiedList);

System.out.println("Original list size: " + originalList.size());
System.out.println("Copied list size: " + copiedList.size());
}
}
```

```
Original ArrayList: [Apple, Banana, Orange, Mango]
Copied ArrayList: [Apple, Banana, Orange, Mango]
Original list size: 4
Copied list size: 4
```

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.

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.

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.

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.

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.

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.

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 Assignment_Day8;
import java.util.Collections;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.Scanner;

public class Problem11{
public static void main(String[] args) {
```

```
Scanner scanner = new Scanner(System.in);

LinkedList<Integer> numbers = new LinkedList<>();
numbers.add(10);
numbers.add(20);
numbers.add(30);
numbers.add(40);
numbers.add(50);
System.out.println("1. Original LinkedList: " + numbers);
numbers.addFirst(5);
numbers.addLast(60);
System.out.println("\n2. After adding first and last elements: " + numbers);
numbers.add(3, 25);
System.out.println("\n3. After inserting 25 at position 3: " + numbers);
System.out.println("\n4. Removal Operations:");
System.out.println("Original list: " + numbers);
numbers.removeFirst();
System.out.println("After removing first element: " + numbers);
numbers.removeLast();
System.out.println("After removing last element: " + numbers);
numbers.remove(Integer.valueOf(25));
System.out.println("After removing value 25: " + numbers);
System.out.print("\n5. Enter a number to search: ");
int searchNum = scanner.nextInt();
if (numbers.contains(searchNum)) {
System.out.println(searchNum + " found in the list at position " +
numbers.indexOf(searchNum));
} else {
System.out.println(searchNum + " not found in the list");
}
System.out.println("\n6. ListIterator Demonstration:");
ListIterator<Integer> iterator = numbers.listIterator();
System.out.print("Forward iteration: ");
while (iterator.hasNext()) {
System.out.print(iterator.next() + " ");
}
System.out.print("\nBackward iteration: ");
while (iterator.hasPrevious()) {
System.out.print(iterator.previous() + " ");
}
Collections.sort(numbers);
System.out.println("\n\n7. Sorted LinkedList: " + numbers);
scanner.close();
}
}
```

```
1. Original LinkedList: [10, 20, 30, 40, 50]

2. After adding first and last elements: [5, 10, 20, 30, 40, 50, 60]

3. After inserting 25 at position 3: [5, 10, 20, 25, 30, 40, 50, 60]

4. Removal Operations:
Original list: [5, 10, 20, 25, 30, 40, 50, 60]
After removing first element: [10, 20, 25, 30, 40, 50, 60]
After removing last element: [10, 20, 25, 30, 40, 50]
After removing value 25: [10, 20, 30, 40, 50]

5. Enter a number to search: 20
20 found in the list at position 1

6. ListIterator Demonstration:
Forward iteration: 10 20 30 40 50
Backward iteration: 50 40 30 20 10

7. Sorted LinkedList: [10, 20, 30, 40, 50]
```

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 Assignment_Day8;

import java.util.LinkedList;
import java.util.ArrayList;

public class Program12 {
public static void main(String[] args) {
LinkedList<String> linkedList = new LinkedList<>();
linkedList.add("Apple");
linkedList.add("Banana");
linkedList.add("Orange");
```

```
linkedList.add("Mango");
System.out.println("Original LinkedList:");
System.out.println(linkedList);
ArrayList<String> arrayList = new ArrayList<>(linkedList);
System.out.println("\nConverted ArrayList:");
System.out.println(arrayList);

System.out.println("LinkedList size: " + linkedList.size());
System.out.println("ArrayList size: " + arrayList.size());
}
}
```

```
Original LinkedList:
[Apple, Banana, Orange, Mango]

Converted ArrayList:
[Apple, Banana, Orange, Mango]
LinkedList size: 4
ArrayList size: 4
```

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 Assignment_Day8;

import java.util.LinkedList;

class Book {
private int id;
private String title;
private String author;

public Book(int id, String title, String author) {
this.id = id;
this.title = title;
```

```
this.author = author;
}

// Getters
public int getId() {
return id;
}

public String getTitle() {
return title;
}

public String getAuthor() {
return author;
}



@Override
public String toString() {
return "Book [ID=" + id + ", Title=" + title + ", Author=" + author + "]";
}
}

public class Problem13 {
public static void main(String[] args) {
LinkedList<Book> bookList = new LinkedList<>();

bookList.add(new Book(101, "The Great Gatsby", "F. Scott Fitzgerald"));
bookList.add(new Book(102, "To Kill a Mockingbird", "Harper Lee"));
bookList.add(new Book(103, "1984", "George Orwell"));

System.out.println("Books in the LinkedList:");
for (Book book : bookList) {
System.out.println(book);
}

}
}
```

```
Books in the LinkedList:
Book [ID=101, Title=The Great Gatsby, Author=F. Scott Fitzgerald]
Book [ID=102, Title=To Kill a Mockingbird, Author=Harper Lee]
Book [ID=103, Title=1984, Author=George Orwell]
```

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 Assignment_Day8;

import java.util.LinkedList;

public class Problem14{
public static void main(String[] args) {
LinkedList<Integer> originalList = new LinkedList<>();
originalList.add(10);
originalList.add(20);
originalList.add(30);
originalList.add(40);
System.out.println("Original LinkedList: " + originalList);
LinkedList<Integer> clonedList = (LinkedList<Integer>) originalList.clone();
System.out.println("Cloned LinkedList: " + clonedList);
originalList.add(50);
clonedList.removeLast();
System.out.println("\nAfter modifications:");
System.out.println("Original LinkedList: " + originalList);
System.out.println("Cloned LinkedList: " + clonedList);
}
}
```

```
Original LinkedList: [10, 20, 30, 40]
Cloned LinkedList: [10, 20, 30, 40]

After modifications:
Original LinkedList: [10, 20, 30, 40, 50]
Cloned LinkedList: [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.
- **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 Assignment_Day8;

import java.util.Enumeration;
import java.util.Vector;

public class Problem15 {
public static void main(String[] args) {
Vector<Integer> intVector = new Vector<>();
intVector.add(10);
```

```
intVector.add(20);
intVector.add(30);
intVector.add(40);
intVector.add(50);
System.out.println("1. Original Integer Vector: " + intVector);
intVector.add(2, 25);
System.out.println(" After inserting 25 at position 3: " + intVector);

intVector.remove(1);
System.out.println(" After removing 2nd element: " + intVector);
System.out.println(" Elements using Enumeration:");
Enumeration<Integer> enumInt = intVector.elements();
while (enumInt.hasMoreElements()) {
System.out.println(" " + enumInt.nextElement());
}
Vector<String> strVector = new Vector<>();
strVector.add("Alice");
strVector.add("Bob");
strVector.add("Charlie");
strVector.add("David");
System.out.println("\n2. Original String Vector: " + strVector);
String searchName = "Bob";
System.out.println(" Does '" + searchName + "' exist? " +
strVector.contains(searchName));
strVector.set(2, "Eve");
System.out.println(" After replacing Charlie with Eve: " + strVector);
strVector.clear();
System.out.println(" After clearing: " + strVector);
Vector<Integer> copyVector = new Vector<>(intVector);
System.out.println("\n3. Copied Vector: " + copyVector);
System.out.println("4. Sum of elements: " + sumVector(intVector));
}
public static int sumVector(Vector<Integer> vec) {
int sum = 0;
for (int num : vec) {
sum += num;
}
return sum;
}
}
```

```
1. Original Integer Vector: [10, 20, 30, 40, 50]
   After inserting 25 at position 3: [10, 20, 25, 30, 40, 50]
   After removing 2nd element: [10, 25, 30, 40, 50]
   Elements using Enumeration:
   10
   25
   30
   40
   50

2. Original String Vector: [Alice, Bob, Charlie, David]
   Does 'Bob' exist? true
   After replacing Charlie with Eve: [Alice, Bob, Eve, David]
   After clearing: []

3. Copied Vector: [10, 25, 30, 40, 50]
4. Sum of elements: 155
```

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.
- **Reverse a string using Stack:**
- Input a string from the user.

- Use a stack to reverse and print the string.
- **Use Stack to check for balanced parentheses** in an expression.
- Input: (a+b) * (c-d)
- Output: Valid or Invalid expression
- **Convert a decimal number to binary using Stack.**

```
package Assignment_Day8;

import java.util.Stack;
import java.util.Scanner;

public class Problem16{
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);

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

numStack.push(10);
numStack.push(20);
numStack.push(30);
numStack.push(40);
numStack.push(50);
System.out.println("Stack after pushes: " + numStack);

System.out.println("Popped: " + numStack.pop());
System.out.println("Stack after pop: " + numStack);

System.out.println("Current top: " + numStack.peek());

System.out.println("Is stack empty " + numStack.empty());
System.out.print("\nEnter a string to reverse: ");
String input = scanner.nextLine();
Stack<Character> charStack = new Stack<>();
for (char c : input.toCharArray()) {
charStack.push(c);
```

```
}
StringBuilder reversed = new StringBuilder();
while (!charStack.empty()) {
reversed.append(charStack.pop());
}
System.out.println("Reversed string: " + reversed);

System.out.print("\nEnter an expression to check parentheses: ");
String expr = scanner.nextLine();
System.out.println("Parentheses balanced " + isBalanced(expr));
scanner.close();
}
private static boolean isBalanced(String expr) {
Stack<Character> stack = new Stack<>();
for (char c : expr.toCharArray()) {
if (c == '(') {
stack.push(c);
} else if (c == ')') {
if (stack.empty()) return false;
stack.pop();
}
}
return stack.empty();
}
}
```

```
Stack after pushes: [10, 20, 30, 40, 50]
Popped: 50
Stack after pop: [10, 20, 30, 40]
Current top: 40
Is stack empty false

Enter a string to reverse: abcd
Reversed string: dcba

Enter an expression to check parentheses: (a*b)+c
Parentheses balanced true
```

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.
2. **Perform operations:**
    - Remove an element.
    - Check if a city exists.
    - Clear the entire HashSet.
3. **Write a method** that takes a HashSet<Integer> and returns the maximum element.

```
package Assignment_Day8;

import java.util.HashSet;
import java.util.Iterator;
import java.util.Scanner;

public class Problem17{
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
HashSet<String> cities = new HashSet<>();

cities.add("New York");
cities.add("London");
cities.add("Tokyo");
cities.add("Paris");
cities.add("Delhi");
System.out.println("Original HashSet: " + cities);

boolean added = cities.add("London");
System.out.println("'London' added " + added);
System.out.println("HashSet after adding: " + cities);
System.out.println("\nCities using Iterator:");
Iterator<String> iterator = cities.iterator();
while (iterator.hasNext()) {
```

```
System.out.println(iterator.next());
}
cities.remove("Delhi");
System.out.println("\nAfter removing 'Delhi': " + cities);

System.out.print("\nEnter city to check: ");
String city = scanner.nextLine();
System.out.println("Contains '" + city + "'? " + cities.contains(city));
cities.clear();
System.out.println("\nAfter clear(): " + cities);
System.out.println("Is empty " + cities.isEmpty());

HashSet<Integer> numbers = new HashSet<>();
numbers.add(10);
numbers.add(5);
numbers.add(20);
numbers.add(15);
System.out.println("\nInteger HashSet: " + numbers);
System.out.println("Max value: " + findMax(numbers));
scanner.close();
}
private static int findMax(HashSet<Integer> set) {
int max = Integer.MIN_VALUE;
for (int num : set) {
if (num > max) max = num;
}
return max;
}
}
```

```
Original HashSet: [New York, Delhi, Tokyo, London, Paris]
'London' added false
HashSet after adding: [New York, Delhi, Tokyo, London, Paris]

Cities using Iterator:
New York
Delhi
Tokyo
London
Paris

After removing 'Delhi': [New York, Tokyo, London, Paris]

Enter city to check: Tokyo
Contains 'Tokyo'? true

After clear(): []
Is empty  true

Integer HashSet: [20, 5, 10, 15]
Max value: 20
```

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.
2. **Task Manager:**
    - Queue of tasks (String values).
    - Add tasks, peek at the next task, and poll completed tasks.
3. **Write a method:**

- That takes a queue of integers and returns a list of even numbers.

```
package Assignment_Day8;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

public class Problem19{
public static void main(String[] args) {
Queue<String> customerQueue = new LinkedList<>();
customerQueue.add("Customer 1");
customerQueue.add("Customer 2");
customerQueue.add("Customer 3");
customerQueue.add("Customer 4");
customerQueue.add("Customer 5");
System.out.println("Bank Queue:");
while (!customerQueue.isEmpty()) {
String customer = customerQueue.poll();
System.out.println("Serving: " + customer);
System.out.println("Remaining queue: " + customerQueue);
}

Queue<String> tasks = new LinkedList<>();
tasks.add("Print reports");
tasks.add("Email clients");
tasks.add("Update database");
System.out.println("\nNext task: " + tasks.peek());
System.out.println("Completed: " + tasks.poll());
System.out.println("Remaining tasks: " + tasks);

Queue<Integer> numberQueue = new LinkedList<>();
numberQueue.add(1);
numberQueue.add(2);
numberQueue.add(3);
numberQueue.add(4);
numberQueue.add(5);
System.out.println("\nEven numbers: " + getEvens(numberQueue));
}
```

```
private static List<Integer> getEvens(Queue<Integer> queue) {
List<Integer> evens = new ArrayList<>();
for (Integer num : queue) {
if (num % 2 == 0) evens.add(num);
}
return evens;
}
}
```

```
Bank Queue:
Serving: Customer 1
Remaining queue: [Customer 2, Customer 3, Customer 4, Customer 5]
Serving: Customer 2
Remaining queue: [Customer 3, Customer 4, Customer 5]
Serving: Customer 3
Remaining queue: [Customer 4, Customer 5]
Serving: Customer 4
Remaining queue: [Customer 5]
Serving: Customer 5
Remaining queue: []

Next task: Print reports
Completed: Print reports
Remaining tasks: [Email clients, Update database]

Even numbers: [2, 4]
```

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).

2. **Print Jobs Priority:**
    - Add different print jobs (String) with priority levels.
    - Use PriorityQueue to simulate serving high-priority jobs before others.
3. **Write a method:**
    - To merge two PriorityQueue<Integer> and return a sorted merged queue.

```
package Assignment_Day8;

import java.util.PriorityQueue;
import java.util.Comparator;

public class Problem20 {
public static void main(String[] args) {
PriorityQueue<Patient> emergencyQueue = new PriorityQueue<>(
Comparator.comparingInt(Patient::getSeverity).reversed()
);
emergencyQueue.add(new Patient("Jay", 3));
emergencyQueue.add(new Patient("Sarath", 5));
emergencyQueue.add(new Patient("Mani", 1));
System.out.println("Hospital Queue (Most critical first):");
while (!emergencyQueue.isEmpty()) {
System.out.println("Treating: " + emergencyQueue.poll());
}

PriorityQueue<PrintJob> printQueue = new PriorityQueue<>(
Comparator.comparingInt(PrintJob::getPriority)
);
printQueue.add(new PrintJob("Document", 2));
printQueue.add(new PrintJob("Urgent Report", 1));
printQueue.add(new PrintJob("Presentation", 3));
System.out.println("\nPrint Jobs (Priority order):");
printQueue.forEach(System.out::println);
}
```

```
}

class Patient {
String name;
int severity;
public Patient(String name, int severity) {
this.name = name;
this.severity = severity;
}
public int getSeverity() { return severity; }
@Override
public String toString() {
return name + " (Severity: " + severity + ")";
}
}

class PrintJob {
String name;
int priority;
public PrintJob(String name, int priority) {
this.name = name;
this.priority = priority;
}
public int getPriority() { return priority; }
@Override
public String toString() {
return name + " (Priority: " + priority + ")";
}
}
```

```
Hospital Queue (Most critical first):
Treating: Sarath (Severity: 5)
Treating: Jay (Severity: 3)
Treating: Mani (Severity: 1)

Print Jobs (Priority order):
Urgent Report (Priority: 1)
Document (Priority: 2)
Presentation (Priority: 3)
```

Deque

1. **Palindrome Checker:**
    - Input a string and check if it is a palindrome using a Deque<Character>.
2. **Double-ended Order System:**
    - Add items from front and rear.
    - Remove items from both ends.
    - Display contents of the deque after each operation.

```
package Assignment_Day8;

import java.util.ArrayDeque;
import java.util.Deque;
import java.util.Scanner;

public class Problem21 {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter a string to check for palindrome: ");
String input = scanner.nextLine();
System.out.println("Is palindrome? " + isPalindrome(input));
Deque<String> deque = new ArrayDeque<>();
deque.addFirst("Front 1");
deque.addLast("Rear 1");
deque.addFirst("Front 2");
deque.addLast("Rear 2");
System.out.println("\nDeque after additions: " + deque);
System.out.println("Removed from front: " + deque.removeFirst());
System.out.println("Removed from rear: " + deque.removeLast());
System.out.println("Deque after removals: " + deque);

}
private static boolean isPalindrome(String str) {
Deque<Character> deque = new ArrayDeque<>();
for (char c : str.toCharArray()) {
deque.addLast(c);
```

```
}
while (deque.size() > 1) {
if (!deque.removeFirst().equals(deque.removeLast())) {
return false;
}
}
return true;
}
}
```

```
Enter a string to check for palindrome: ababa
Is palindrome? true

Deque after additions: [Front 2, Front 1, Rear 1, Rear 2]
Removed from front: Front 2
Removed from rear: Rear 2
Deque after removals: [Front 1, Rear 1]
```