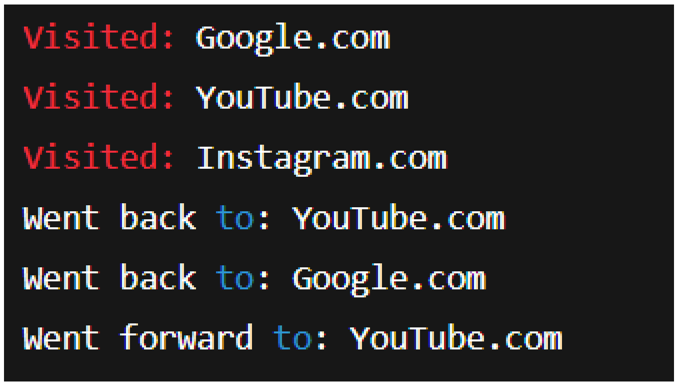
1. Browser History (Using Two Stacks)

```
import java.util.*;

public class BrowserHistory {
    Stack<String> backStack = new Stack<>();
    Stack<String> forwardStack = new Stack<>();

    public void visit(String url) {
        backStack.push(url);
        forwardStack.clear();
        System.out.println("Visited: " + url);
    }

    public void back() {
        if (backStack.size() > 1) {
            forwardStack.push(backStack.pop());
            System.out.println("Went back to: " + backStack.peek());
        } else {
            System.out.println("No previous page.");
        }
    }

    public void forward() {
        if (!forwardStack.isEmpty()) {
            String page = forwardStack.pop();
            backStack.push(page);
            System.out.println("Went forward to: " + page);
        } else {
            System.out.println("No forward page.");
        }
    }

    public static void main(String[] args) {
        BrowserHistory browser = new BrowserHistory();
        browser.visit("Google.com");
        browser.visit("YouTube.com");
        browser.visit("Instagram.com");
        browser.back();
        browser.back();
        browser.forward();
    }
}
```

OUTPUT

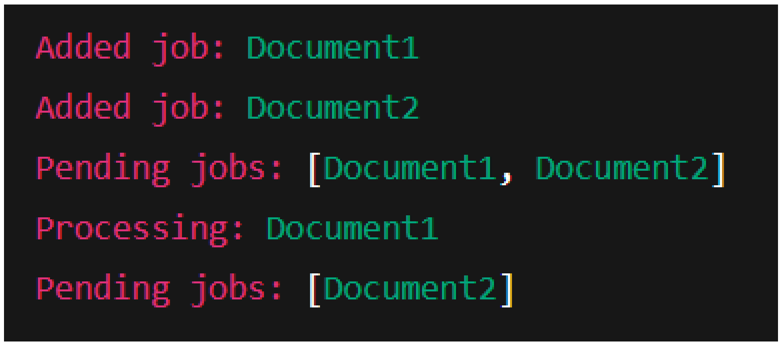


```
Visited: Google.com
Visited: YouTube.com
Visited: Instagram.com
Went back to: YouTube.com
Went back to: Google.com
Went forward to: YouTube.com
```

2. Print Queue (Using LinkedList as Queue)

```
import java.util.*;

public class PrintQueue {
    Queue<String> queue = new LinkedList<>();

    public void addJob(String job) {
        queue.add(job);
        System.out.println("Added job: " + job);
    }

    public void processJob() {
        if (!queue.isEmpty()) {
            System.out.println("Processing: " + queue.poll());
        } else {
            System.out.println("No jobs to process.");
        }
    }

    public void viewJobs() {
        System.out.println("Pending jobs: " + queue);
    }

    public static void main(String[] args) {
        PrintQueue printer = new PrintQueue();
        printer.addJob("Document1");
        printer.addJob("Document2");
        printer.viewJobs();
        printer.processJob();
        printer.viewJobs();
    }
}
```

OUTPUT

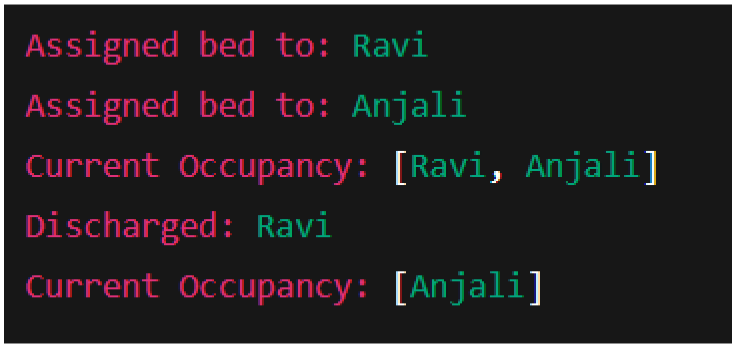


```
Added job: Document1
Added job: Document2
Pending jobs: [Document1, Document2]
Processing: Document1
Pending jobs: [Document2]
```

3. Hospital Bed Management (Using LinkedList)

```
import java.util.*;

public class HospitalBeds {
    LinkedList<String> beds = new LinkedList<>();

    public void assignBed(String patient) {
        beds.add(patient);
        System.out.println("Assigned bed to: " + patient);
    }

    public void discharge(String patient) {
        if (beds.remove(patient)) {
            System.out.println("Discharged: " + patient);
        } else {
            System.out.println("Patient not found.");
        }
    }

    public void displayBeds() {
        System.out.println("Current Occupancy: " + beds);
    }

    public static void main(String[] args) {
        HospitalBeds hospital = new HospitalBeds();
        hospital.assignBed("Ravi");
        hospital.assignBed("Anjali");
        hospital.displayBeds();
        hospital.discharge("Ravi");
        hospital.displayBeds();
    }
}
```

OUTPUT

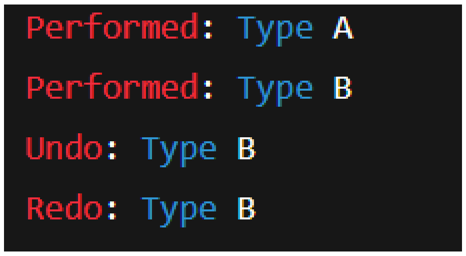


```
Assigned bed to: Ravi
Assigned bed to: Anjali
Current Occupancy: [Ravi, Anjali]
Discharged: Ravi
Current Occupancy: [Anjali]
```

4. Undo-Redo Function (Using Two Stacks)

```
import java.util.*;

public class UndoRedo {
    Stack<String> undoStack = new Stack<>();
    Stack<String> redoStack = new Stack<>();

    public void performAction(String action) {
        undoStack.push(action);
        redoStack.clear();
        System.out.println("Performed: " + action);
    }

    public void undo() {
        if (!undoStack.isEmpty()) {
            String action = undoStack.pop();
            redoStack.push(action);
            System.out.println("Undo: " + action);
        } else {
            System.out.println("Nothing to undo.");
        }
    }

    public void redo() {
        if (!redoStack.isEmpty()) {
            String action = redoStack.pop();
            undoStack.push(action);
            System.out.println("Redo: " + action);
        } else {
            System.out.println("Nothing to redo.");
        }
    }

    public static void main(String[] args) {
        UndoRedo editor = new UndoRedo();
        editor.performAction("Type A");
        editor.performAction("Type B");
        editor.undo();
        editor.redo();
    }
}
```

OUTPUT

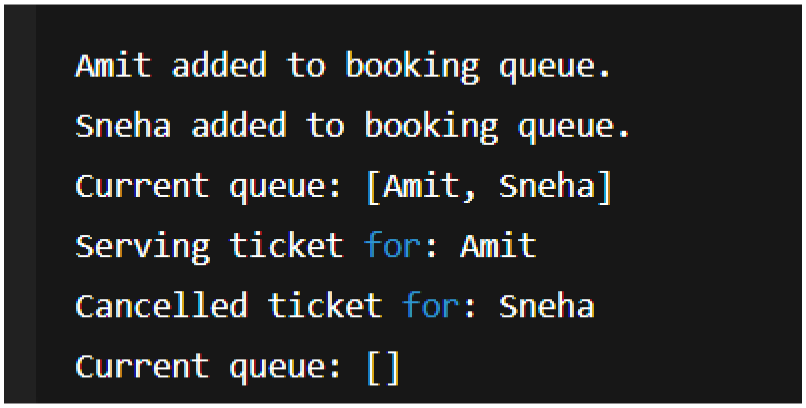


```
Performed: Type A
Performed: Type B
Undo: Type B
Redo: Type B
```

5. Ticket Booking System (Using Queue with LinkedList)

```
import java.util.*;

public class TicketBooking {
    Queue<String> queue = new LinkedList<>();

    public void bookTicket(String name) {
        queue.add(name);
        System.out.println(name + " added to booking queue.");
    }

    public void serveTicket() {
        if (!queue.isEmpty()) {
            System.out.println("Serving ticket for: " + queue.poll());
        } else {
            System.out.println("No one in queue.");
        }
    }

    public void cancelTicket(String name) {
        if (queue.remove(name)) {
            System.out.println("Cancelled ticket for: " + name);
        } else {
            System.out.println(name + " not found in queue.");
        }
    }

    public void viewQueue() {
        System.out.println("Current queue: " + queue);
    }

    public static void main(String[] args) {
        TicketBooking system = new TicketBooking();
        system.bookTicket("Amit");
        system.bookTicket("Sneha");
        system.viewQueue();
        system.serveTicket();
        system.cancelTicket("Sneha");
        system.viewQueue();
    }
}
```

OUTPUT

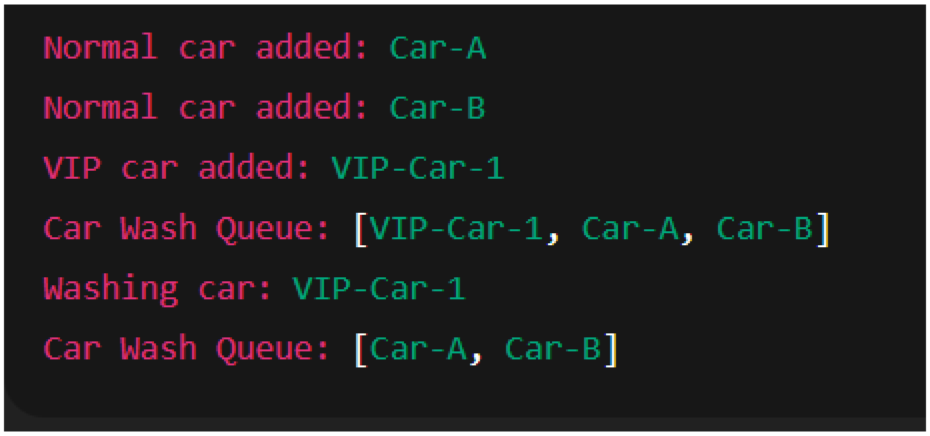


```
Amit added to booking queue.
Sneha added to booking queue.
Current queue: [Amit, Sneha]
Serving ticket for: Amit
Cancelled ticket for: Sneha
Current queue: []
```

6. Car Wash Service Queue (Using LinkedList)

```
import java.util.LinkedList;

public class CarWashService {
    LinkedList<String> queue = new LinkedList<>();

    public void addNormalCar(String car) {
        queue.addLast(car);
        System.out.println("Normal car added: " + car);
    }

    public void addVIPCar(String car) {
        queue.addFirst(car);
        System.out.println("VIP car added: " + car);
    }

    public void washCar() {
        if (!queue.isEmpty()) {
            System.out.println("Washing car: " + queue.removeFirst());
        } else {
            System.out.println("No cars in queue.");
        }
    }

    public void viewQueue() {
        System.out.println("Car Wash Queue: " + queue);
    }

    public static void main(String[] args) {
        CarWashService wash = new CarWashService();
        wash.addNormalCar("Car-A");
        wash.addNormalCar("Car-B");
        wash.addVIPCar("VIP-Car-1");
        wash.viewQueue();
        wash.washCar();
        wash.viewQueue();
    }
}
```

OUTPUT



```
Normal car added: Car-A
Normal car added: Car-B
VIP car added: VIP-Car-1
Car Wash Queue: [VIP-Car-1, Car-A, Car-B]
Washing car: VIP-Car-1
Car Wash Queue: [Car-A, Car-B]
```

7. Library Book Stack (Using Stack)

```
import java.util.Stack;

public class LibraryStack {
    Stack<String> bookStack = new Stack<>();

    public void addBook(String book) {
        bookStack.push(book);
        System.out.println("Added book: " + book);
    }

    public void removeBook() {
        if (!bookStack.isEmpty()) {
            System.out.println("Removed book: " + bookStack.pop());
        } else {
            System.out.println("No books to remove.");
        }
    }

    public void peekBook() {
        if (!bookStack.isEmpty()) {
            System.out.println("Top book: " + bookStack.peek());
        } else {
            System.out.println("Stack is empty.");
        }
    }

    public static void main(String[] args) {
        LibraryStack library = new LibraryStack();
        library.addBook("Java Basics");
        library.addBook("Data Structures");
        library.peekBook();
        library.removeBook();
        library.peekBook();
    }
}
```

OUTPUT

```
Added book: Java Basics
Added book: Data Structures
Top book: Data Structures
Removed book: Data Structures
Top book: Java Basics
```