```
::::::::::::::
StackInterface.java
::::::::::::::
/**
 * Purpose: Data Structure and Algorithms Lab 5
 * Status: Complete and thoroughly tested
 * Last update: 02/20/23
 * Submitted: 02/20/23
 * Comment: test suite and sample run attached
 * Comment: I declare that this is entirely my own work
 * @author: Antonio Rosado
 * @version: 2023.02.20
 */

public interface StackInterface<T>
{
    /**
     * Check if Stack is empty
     * @return Boolean true/false
     */
    public boolean isEmpty();
    // Precondition: None.
    // Postcondition: Returns true if the stack is empty;
    // otherwise returns false.

    /**
     * Removes or 'pops' item from the top of the stack
     * @throws StackException   thrown if stack is empty
     *
     * @return head == null
     */
    public T pop() throws StackException;
    // Precondition: None.
    // Postcondition: If the stack is not empty, the item
    // that was added most recently is removed from the
    // stack.
    // Exception: Throws StackException if the stack is
    // empty.

    /**
     * Removes all items on the stack
     */
    public void popAll();
    // Precondition: None.
    // PostCondition: Stack is empty.

    /**
     * Adds or 'pushes' an item to top of the stack
     * @param T newItem        new item to be pushed onto stack
     * @throws StackException   thrown if stack is empty
     */
    public void push(T newItem) throws StackException;
    // Precondition: newItem is the item to be added.
    // Postcondition: If insertion is successful, newItem
    // is on the top of the stack.
    // Exception: Some implementations may throw
    // StackException when newItem cannot be placed on
    // the stack.

    /**
     * Retrieves item from the top of the stack
     * @throws StackException   thrown if stack is empty
     */
    public T peek() throws StackException;
    // Precondition: None.
    // Postcondition: If the stack is not empty, the item
    // that was added most recently is returned. The
    // stack is unchanged.
    // Exception: Throws StackException if the stack is
    // empty.

    /**
     * Returns a string representation of stack
     */
    public String toString();
    // Return the String representation of the items in the
    // collection (top to bottom; single space delimited)
    // - no additional narratives
} // end StackInterface::::::::::::::
StackException.java
::::::::::::::
/**
 * Purpose: Data Structure and Algorithms Lab 5
 * Status: Complete and thoroughly tested
 * Last update: 02/20/23
 * Submitted: 02/20/23
 * Comment: test suite and sample run attached
 * Comment: I declare that this is entirely my own work
 * @author: Antonio Rosado
 * @version: 2023.02.20
 */

public class StackException
    extends java.lang.RuntimeException
{
    public StackException(String s)
    {
        super(s);
    } // end constructor
} // end StackException
::::::::::::::
StackRA.java
::::::::::::::
/**
 * Purpose: Data Structure and Algorithms Lab 5
 * Status: Complete and thoroughly tested
 * Last update: 02/20/23
 * Submitted: 02/20/23
 * Comment: test suite and sample run attached
 * Comment: I declare that this is entirely my own work
 * @author: Antonio Rosado
 * @version: 2023.02.20
 */

public class StackRA<T> implements StackInterface<T>
{
    private T[] items; // items in stack
    private int top; // pointer to top of stack

    /**
     * Constructor for objects of class StackRA
     *
     */
    @SuppressWarnings("unchecked")
```

```
public StackRA()
{
    items = (T[])new Object[3]; // initialize Array with generic casting
    top = -1;
}

/**
 * @override
 * Check if Stack is empty
 * @return top == -1;
 */
public boolean isEmpty()
{
    return top == -1;
}

/**
 * @override
 * Removes or 'pops' item from the top of the stack
 * @throws StackException   thrown if stack is empty
 * @return result
 */
public T pop()
{
    T result = null;
    if(top != -1) // if stack is not empty, proceed to pop
    {
        result = items[top]; // result = item at top
        items[top--] = null; // items at top -> down = null
    }
    return result;
}

/**
 * @override
 * Removes all items on the stack
 */
@SuppressWarnings("unchecked")
public void popAll()
{
    items = (T[])new Object[3];
    top = -1;
}

/**
 * @override
 * Adds or 'pushes' an item to top of the stack
 * @param T newItem        new item to be pushed onto stack
 * @throws StackException   thrown if stack is empty
 */
public void push(T newItem) throws StackException
{
    if(top == items.length -1)
    {
        resize();
    }
    items[++top] = newItem;
}

/**
 * @override
 * Retrieves item from the top of the stack
 * @throws StackException   thrown if stack is empty
 * @return result           item at top of stack
 */
public T peek() throws StackException
{
    T result = null;
    if(top != -1)
    {
        result = items[top]; // retrieve item at top
    }
    return result;
}

/**
 * @override
 * Returns a string representation of stack
 * @return result       string representation of stack
 */
public String toString()
{
    String result = "";
    for(int index = 0; index <= top; index++)
    {
        result += items[index] + " ";
    }
    return result;
}

/**
 * resizes the array according to number of items
 */
@SuppressWarnings("unchecked")
public void resize()
{
    T[] temp = (T[])new Object[(int)(items.length * 1.5)];
    for(int index = 0; index < items.length; index++)
    {
        temp[index] = items[index];
    }
    items = temp;
}
}
::::::::::::::
Node.java
::::::::::::::
public class Node<T> {
    private T item;
    private Node<T> next;

    public Node(T newItem) {
        item = newItem;
        next = null;
    } // end constructor

    public Node(T newItem, Node<T> nextNode) {
        item = newItem;
        next = nextNode;
    } // end constructor

    public void setItem(T newItem) {
        item = newItem;
    } // end setItem
```

```
    public T getItem() {
        return item;
    } // end getItem

    public void setNext(Node<T> nextNode) {
        next = nextNode;
    } // end setNext

    public Node<T> getNext() {
        return next;
    } // end getNext

    public String toString()
    {
        return item + " ";
    } //end toString

} // end class Node:::::::::::::::
StackSLS.java
:::::::::::::::
/**
 * Purpose: Data Structure and Algorithms Lab 5
 * Status: Complete and thoroughly tested
 * Last update: 02/20/23
 * Submitted: 02/20/23
 * Comment: test suite and sample run attached
 * Comment: I declare that this is entirely my own work
 * @author: Antonio Rosado
 * @version: 2023.02.20
 */

public class StackSLS<T> implements StackInterface<T>
{
    private Node<T> top; // pointer to top of stack

    /**
     * Constructor for objects of class Stack
     */
    public StackSLS()
    {
        top = null;
    }

    /**
     * @override
     * Check if Stack is empty
     * @return top == -1;
     */
    public boolean isEmpty()
    {
        return top == null;
    }

    /**
     * @override
     * Removes or 'pops' item from the top of the stack
     * @throws StackException   thrown if stack is empty
     * @return result           result after pop
     */
    public T pop()
    {
        T result = null;
        if(top != null) // if stack is not empty, proceed to pop
        {
            result = top.getItem(); // result = item at top
            top = (Node<T>) top.getNext(); // items at top -> down = null
        }
        return result;
    }

    /**
     * @override
     * Removes all items on the stack
     */
    public void popAll()
    {
        top = null;
    }

    /**
     * @override
     * Adds or 'pushes' an item to top of the stack
     * @param T newItem        new item to be pushed onto stack
     * @throws StackException   thrown if stack is empty
     */
    public void push(T newItem) throws StackException
    {
        top = new Node<T>(newItem, top);
    }

    /**
     * @override
     * Retrieves item from the top of the stack
     * @throws StackException   thrown if stack is empty
     * @return result           item at top of stack
     */
    public T peek() throws StackException
    {
        T result = null;
        if(top != null)
        {
            result = top.getItem(); // retrieve item at top
        }
        return result;
    }

    /**
     * @override
     * Returns a string representation of stack
     * @return result       string representation of stack
     */
    public String toString()
    {
        StringBuilder sb = new StringBuilder();
        Node<T> curr = top;
        while(curr != null)
        {
            sb.append(curr.getItem().toString() + " ");
            curr = curr.getNext();
        }
        return sb.toString();
    }
}
```

```
::::::::::::::
StackRADriver.java
::::::::::::::
/**
 * Purpose: Data Structure and Algorithms Lab 5
 * Status: Complete and thoroughly tested
 * Last update: 02/20/23
 * Submitted: 02/20/23
 * Comment: test suite and sample run attached
 * Comment: I declare that this is entirely my own work
 * @author: Antonio Rosado
 * @version: 2023.02.20
 */
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class StackRADriver
{

    static BufferedReader stdin = new BufferedReader(new InputStreamReader(System.
in));
    public static void main (String[] args) throws IOException
    {
        StackRA<Object> stackRA = new StackRA<Object>();
        boolean exit = false;
        while (!exit) {
            System.out.println("Select from the following menu: \n"
                               + "\t 0. Exit. \n"
                               + "\t 1. Push item onto the stack. \n"
                               + "\t 2. Pop item from the stack. \n"
                               + "\t 3. Display the top item of the stack. \n"
                               + "\t 4. Display items in the stack. \n"
                               + "\t 5. Clear the stack. \n");

            System.out.print("Make your menu selection now: ");
            int input = Integer.parseInt(stdin.readLine().trim());
            System.out.println(input);
            // possible cases for initial input
            switch (input) {
            case 0:
                System.out.println("Exiting program... good bye");
                exit = true;
                break;

            case 1:
                System.out.println("You are now pushing an item onto the stack.");
                System.out.print("\t Enter item: ");
                Object item = stdin.readLine().trim();
                System.out.println(item);
                stackRA.push(item);
                System.out.println("Item " + item + " was successfully pushed onto
 the stack. \n");
                break;

            case 2:
                if(!(stackRA.isEmpty()))
                {
                    System.out.println("You are now popping an item from the stack
...");
                    System.out.print("Item " + stackRA.peek() + " was successfully
 popped off the stack. \n");
                    stackRA.pop();
                }

                else
                {
                    System.out.println("Stack is empty! \n");
                }

                break;

            case 3:
                if(!(stackRA.isEmpty()))
                {
                    System.out.println("The stack contains: " + stackRA.toString()
);
                    System.out.print("Item " + stackRA.peek() + " is on the top of
 the stack. \n");
                }

                else
                {
                    System.out.println("Stack is empty! \n");
                }
                break;

            case 4:
                if(!(stackRA.isEmpty()))
                {
                    System.out.println("The stack contains: " + stackRA.toString()
);
                }

                else
                {
                    System.out.println("Stack is empty! \n");
                }
                break;

            case 5:
                if(!(stackRA.isEmpty()))
                {
                    System.out.println("Emptying stack... ");
                    System.out.println("Items " + stackRA.toString() + " have been
 removed from the stack. \n");
                    stackRA.popAll();
                }

                else
                {
                    System.out.println("Stack is already empty! \n");
                }
                break;
            }
        }
    }
}
::::::::::::::
StackSLSDriver.java
::::::::::::::
/**
 * Purpose: Data Structure and Algorithms Lab 5
 * Status: Complete and thoroughly tested
```

```
 * Last update: 02/20/23
 * Submitted: 02/20/23
 * Comment: test suite and sample run attached
 * Comment: I declare that this is entirely my own work
 * @author: Antonio Rosado
 * @version: 2023.02.20
 */
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class StackSLSDriver
{

    static BufferedReader stdin = new BufferedReader(new InputStreamReader(System.
in));
    public static void main (String[] args) throws IOException
    {
        StackSLS<Object> stackSLS = new StackSLS<Object>();
        boolean exit = false;
        while (!exit) {
            System.out.println("Select from the following menu: \n"
                               + "\t 0. Exit. \n"
                               + "\t 1. Push item onto the stack. \n"
                               + "\t 2. Pop item from the stack. \n"
                               + "\t 3. Display the top item of the stack. \n"
                               + "\t 4. Display items in the stack. \n"
                               + "\t 5. Clear the stack. \n");

            System.out.print("Make your menu selection now: ");
            int input = Integer.parseInt(stdin.readLine().trim());
            System.out.println(input);
            // possible cases for initial input
            switch (input) {
            case 0:
                System.out.println("Exiting program... good bye");
                exit = true;
                break;

            case 1:
                System.out.println("You are now pushing an item onto the stack.");
                System.out.print("\t Enter item: ");
                Object item = stdin.readLine().trim();
                System.out.println(item);
                stackSLS.push(item);
                System.out.println("Item " + item + " was successfully pushed onto
 the stack. \n");
                break;

            case 2:
                if(!(stackSLS.isEmpty()))
                {
                    System.out.println("You are now popping an item from the stack
...");
                    System.out.print("Item " + stackSLS.peek() + " was successfull
y popped off the stack. \n");
                    stackSLS.pop();
                }

                else
                {
                    System.out.println("Stack is empty! \n");
                }

                break;

            case 3:
                if(!(stackSLS.isEmpty()))
                {
                    System.out.println("The stack contains: " + stackSLS.toString(
));
                    System.out.print("Item " + stackSLS.peek() + " is on the top o
f the stack. \n");
                }

                else
                {
                    System.out.println("Stack is empty! \n");
                }
                break;

            case 4:
                if(!(stackSLS.isEmpty()))
                {
                    System.out.println("The stack contains: " + stackSLS.toString(
));
                }

                else
                {
                    System.out.println("Stack is empty! \n");
                }
                break;

            case 5:
                if(!(stackSLS.isEmpty()))
                {
                    System.out.println("Emptying stack... ");
                    System.out.println("Items " + stackSLS.toString() + " have bee
n removed from the stack. \n");
                    stackSLS.popAll();
                }

                else
                {
                    System.out.println("Stack is already empty! \n");
                }
                break;
            }
        }
    }
}
```