John Rizkalla
CSC 236
Lab 4

Program #1

ArrayStackADT.java

```
public interface ArrayStackADT<T>{
	public void initializeStack();
	public boolean isEmptyStack();
	public boolean isFullStack();
	public T push(T newElement) throws StackOverflowException;
	public T peek() throws StackUnderflowException;
	public void pop() throws StackUnderflowException;
}
```

ArrayStackDataStrucClass.java

```
public class ArrayStackDataStrucClass<T> implements ArrayStackADT<T>{

	public int maxStackSize;
	public int stackTop;
	public T [] arr;

	public ArrayStackDataStrucClass(int maxStackSize2) {
		maxStackSize = 50;
		stackTop = 0;
		arr = (T[]) new Object[maxStackSize];
	}


	public ArrayStackDataStrucClass(int size, int top, T[] array) {
		this.maxStackSize = size;
		this.stackTop = top;
		this.arr = array;
	}

	public ArrayStackDataStrucClass(ArrayStackDataStrucClass<T> stack) {
		maxStackSize = stack.maxStackSize;
		stackTop = stack.stackTop;
		arr = stack.arr;
	}

	@Override
	public void initializeStack() {
```

```
		for(int i = 0; i < stackTop; i++) {
			arr[i] = null;
		}

		stackTop = 0;
}

@Override
public boolean isEmptyStack() {
		return (stackTop == 0);
}

@Override
public boolean isFullStack() {
		return (stackTop == maxStackSize);
}



@Override
public T push(T newElement) throws StackOverflowException{
		if(isFullStack()) {
			throw new StackOverflowException();
		}
			arr[stackTop] = (T) newElement;

			stackTop++;
			return arr[stackTop];

}

@Override
public T peek() throws StackUnderflowException{
		if(isEmptyStack()) {
			throw new StackUnderflowException();
		}
			return arr[stackTop -1];
}


@Override
public void pop(){
		if(isEmptyStack()) {
			throw new StackUnderflowException();
		}
			stackTop--;
```

```
                        arr[stackTop] = null;

        }

}
```

PrimeFactorizationDemoClass.java

```
public class PrimeFactorizationDemoClass {
        public static void main(String [] args) {
                ArrayStackDataStrucClass<Integer> factorStack = new
ArrayStackDataStrucClass<Integer>(50);

      int n[] = new int[]{3960, 1234, 222222, 13780};
      int i = 0;

      System.out.println("Factor Stack Elements:\n");
      while (i <= 3)
      {
         try
         {
            int d = 2;
            while (n[i] > 1)
            {
               if (n[i] % d == 0)
               {
                  factorStack.push(d);
                  n[i] = n[i] / d;
               } else
                  d++;
            }
         } catch (StackOverflowException e) {
            System.out.println(e.toString());
           }


         System.out.print("Prime Factors are: ");

         while (!factorStack.isEmptyStack())
         {
            System.out.print(factorStack.peek());
            factorStack.pop();
            if (!factorStack.isEmptyStack())
               System.out.print(" * ");
         }
         System.out.println();
```

```
            i++;
        }

            }

}
```

StackException.java

```
public class StackException extends RuntimeException{
        public StackException() {
                super ("Stack Exception");
        }

        public StackException(String msg) {
                super(msg);
        }

}
```

StackUnderflowException.java

```
public class StackUnderflowException extends RuntimeException{
        public StackUnderflowException() {
                super ("Stack Underflow");
        }

        public StackUnderflowException(String msg) {
                super(msg);
        }

}
```

StackOverflowException.java

```
public class StackOverflowException extends RuntimeException {
        public StackOverflowException() {
                super ("Stack Overflow");
        }

        public StackOverflowException(String msg) {
                super(msg);
        }

}
```

```
Factor Stack Elements:

Prime Factors are: 11 * 5 * 3 * 3 * 2 * 2 * 2
Prime Factors are: 617 * 2
Prime Factors are: 37 * 13 * 11 * 7 * 3 * 2
Prime Factors are: 53 * 13 * 5 * 2 * 2
```

Program #2

ListStackADT.java

```
public interface ListStackADT<T> {
	public boolean isEmpty();
	public void ifEmpty() throws EmptyStackException;
	public void push(T value);
	public T pop() throws EmptyStackException;
	public T peek() throws EmptyStackException;
	public String toString();

}
```

ListStackDataStrucClass.java

```
public class ListStackDataStrucClass<T> implements ListStackADT<T> {

	public class StackNode<T> {
		public T value;
		public StackNode<T> next;


		public StackNode() {
			this.value = null;
			this.next =null;
		}

		public StackNode(T element, StackNode<T> point) {
			this.value = element;
			this.next = point;
		}

		public T getValue(){
			return value;
		}

		public void setValue(T element) {
```

```
            this.value = element;
        }

        public StackNode<T> getNext(){
            return next;
        }

        public void setNext(StackNode<T> point) {
            this.next = point;
        }

        public StackNode(StackNode<T> stack) {
            this.value = stack.getValue();
            this.next = stack.getNext();
        }



}

private StackNode<T> stack;
private StackNode<T> top;

public ListStackDataStrucClass() {
        this.top = null;
}

public ListStackDataStrucClass(StackNode<T> stackTop) {
        this.top = stackTop;
}



public StackNode<T> getTop() {
        return top;

}

public void setTop(StackNode<T> top) {
        this.top = top;

}

@Override
public boolean isEmpty() {
        return (top == null);
```

```
}

@Override
public void ifEmpty() throws EmptyStackException{
	if(isEmpty()) {
		throw new EmptyStackException();
	}

}

@Override
public void push(T value) {
	if(isEmpty()) {
		setTop(new StackNode<T>(value, null));
	} else {
		StackNode<T> newNode;
		newNode = new StackNode<T>(value, getTop());
		setTop(newNode);
	}
}

@Override
public T pop() throws EmptyStackException{
	if(top == null) {
		throw new EmptyStackException();
	} else {
		T element = getTop().getValue();
		setTop(getTop().getNext());
		return element;
	}

}

@Override
public T peek() throws EmptyStackException{
	if(top == null) {
		throw new EmptyStackException();
	} else {
		return getTop().getValue();
	}

}

@Override
public String toString() {
```

```
		StringBuffer sb = new StringBuffer();
		sb.append("[");

		if(!isEmpty()) {
			StackNode<T> temp = getTop();

			while(temp != null) {
				sb.append(temp.getValue() + " ");
			}
		}

		sb.append("]");
		return sb.toString();
	}

}
```

BaseConverter.java

```
import java.util.*;
import java.lang.*;

public class BaseConverter {
	public class BaseNumber {
		private long number;
		private long base;

		public BaseNumber() {
			number = 10;
			base = 0;
		}

		public BaseNumber(long num, long bas) {
			this.number = num;
			this.base = bas;
		}

		public long getNumber() {
			return number;
		}

		public long getBase() {
			return base;
		}

		public void setNumber(long num) {
```

```
                this.number = num;
        }

        public void setBase(long bas) {
                this.base = bas;
        }
    }

    private static final int MINIMUM_BASE = 2;
    private static final int MAXIMUN_BASE = 9;
    private static final int SIZE = 3;



    private BaseNumber [] baseNumber;

    public BaseConverter() {
        baseNumber = new BaseNumber[SIZE];

        inputPrompt();
    }

    public void inputPrompt() {
        Scanner input = new Scanner(System.in);

        System.out.println("Enter a positive integer in base 10 and a base number
between 2 and 9");

        long number;
        long base;

        for(int i = 0; i < SIZE; i++) {
            System.out.print("Number: ");
            number = input.nextLong();


            //System.out.println();


                System.out.print("Base: ");
                base = input.nextLong();


            //System.out.println();

            baseNumber[i] = new BaseNumber(number, base);
```

```
		}

	}

	public String convert(BaseNumber baseNumber) {
		ListStackDataStrucClass<Long> stack = new ListStackDataStrucClass<Long>();
		long number = baseNumber.getNumber();
		long base = baseNumber.getBase();
		String s = "";

		while(number != 0) {
			stack.push(number % base);
			s = s + (number % base) + "";
			number = number / base;
		}

		return s;
	}

	public static String reverseString(String rs) {
		StringBuilder sb = new StringBuilder(rs);
		sb.reverse();
		return sb.toString();
	}


	public String convertAll() {
		String str = "";

		for(int i = 0; i < SIZE; i++) {
			str = str + (baseNumber[i].getNumber() + "(Base 10) = " +
BaseConverter.reverseString(convert(baseNumber[i])) + "(Base " + (baseNumber[i].getBase()) +
")\n");
		}

		return str;
	}

	public String toString() {
		return convertAll();
	}

	public void processAndPrint() {
		System.out.println(this);
	}
```

```
}
```

BaseConverterDemo.java

```
public class BaseConverterDemo {

	public static void main(String[] args) {
		BaseConverter baseConvert = new BaseConverter();

		baseConvert.processAndPrint();

	}

}
```

StackException.java

```
public class StackException extends RuntimeException{
	public StackException() {
		super ("Stack Exception");
	}

	public StackException(String msg) {
		super(msg);
	}

}
```

EmptyStackException.java

```
public class EmptyStackException extends StackException{
	public EmptyStackException() {
		super ("Empty Stack");
	}

	public EmptyStackException(String msg) {
		super (msg);
	}

}
```

FullStackException.java

```
public class FullStackException extends StackException{
	public FullStackException() {
		super ("Full Stack");
```

```
	}

	public FullStackException(String msg) {
		super (msg);
	}

}
```

```
<terminated> BaseConverterDemo [Java Application] /Applications/Eclipse.app/Contents/Eclipse/plugins/org.eclipse.ju
Enter a positive integer in base 10 and a base number between 2 and 9
Number: 72
Base: 4
Number: 53
Base: 2
Number: 3553
Base: 8
72(Base 10) = 1020(Base 4)
53(Base 10) = 110101(Base 2)
3553(Base 10) = 6741(Base 8)
```

Program #3

LinkedStackADT.java

```
public interface LinkedStackADT<T> {
	public String toString();
	public boolean isEmptyStack();
	public void push(T newItem);
	public T peek() throws StackUnderflowException;
	public T pop() throws StackUnderflowException;

}
```

LinkedStackDS.java

```
public class LinkedStackDS<T> {

	private class StackNode<T>{
		private T value;
		private StackNode<T> next;

		public StackNode() {
			this.value = null;
			this.next = null;
```

```
        }

        public StackNode(T element, StackNode<T> point) {
                this.value = element;
                this.next = point;
        }

        public String toString() {
                return (String) stackTop.value;
        }
}

private StackNode<T> stackTop;

public LinkedStackDS() {
        stackTop = null;
}

public LinkedStackDS(StackNode<T> top) {
        this.stackTop = top;
}

public LinkedStackDS(LinkedStackDS<T> stack) {
        this.stackTop = stack.stackTop;
}

public boolean isEmptyStack() {
        return(stackTop == null);
}

public void push(T newItem) {
        StackNode<T> newNode = new StackNode<T>(newItem, stackTop);

        stackTop = newNode;
}

public T peek() throws StackUnderflowException{
        if(isEmptyStack()) {
                throw new StackUnderflowException();
        }

        return stackTop.value;
}

public T pop() throws StackUnderflowException{
        if(isEmptyStack()) {
```

```
                throw new StackUnderflowException();
            } else {
                T element = stackTop.value;
                stackTop = stackTop.next;

                return element;
            }
        }
}
```

StackException.java

```
public class StackException extends RuntimeException{
        public StackException() {
                super ("Stack Exception");
        }

        public StackException(String msg) {
                super(msg);
        }

}
```

StackUnderflowException.java

```
public class StackUnderflowException extends StackException{
        public StackUnderflowException() {
                super ("Stack Underflow");
        }

        public StackUnderflowException(String msg) {
                super(msg);
        }

}
```

StackOverflowException.java

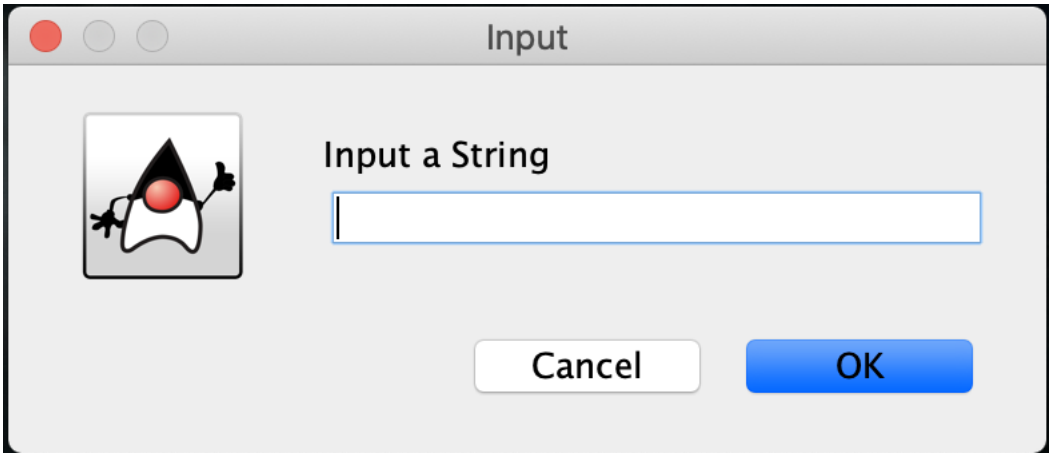
```
public class StackOverflowException extends StackException{
        public StackOverflowException() {
                super ("Stack Overflow");
        }

        public StackOverflowException(String msg) {
                super(msg);
        }
```

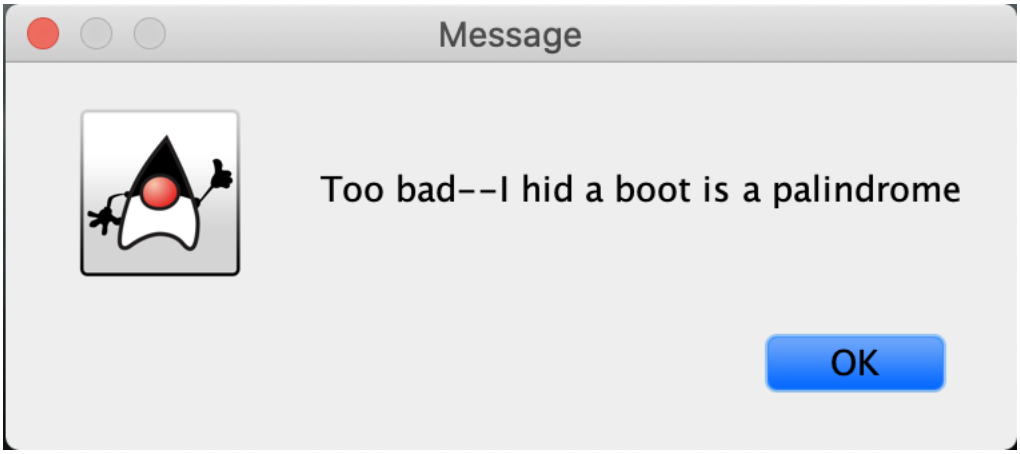
}

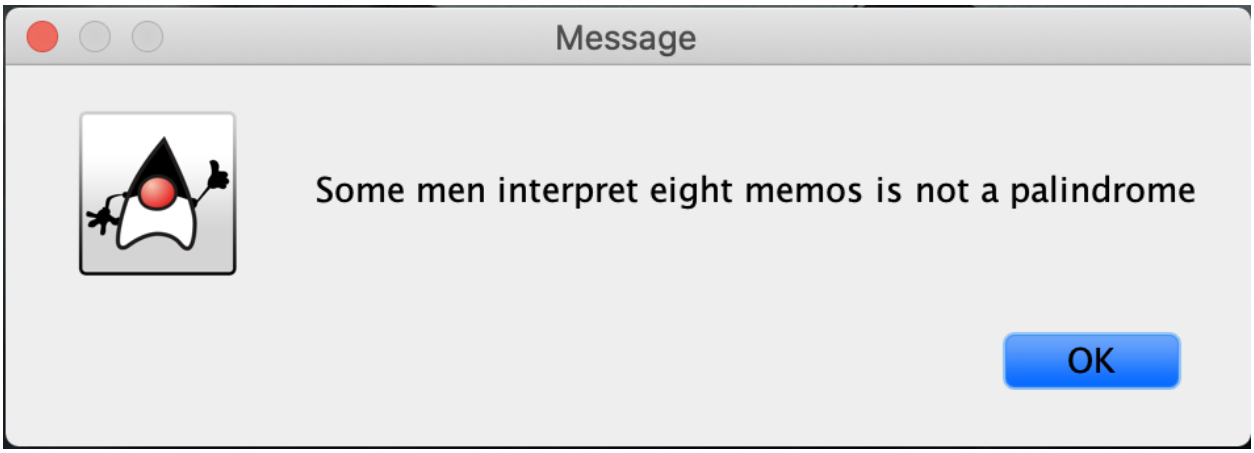
PalindromeDemo.java

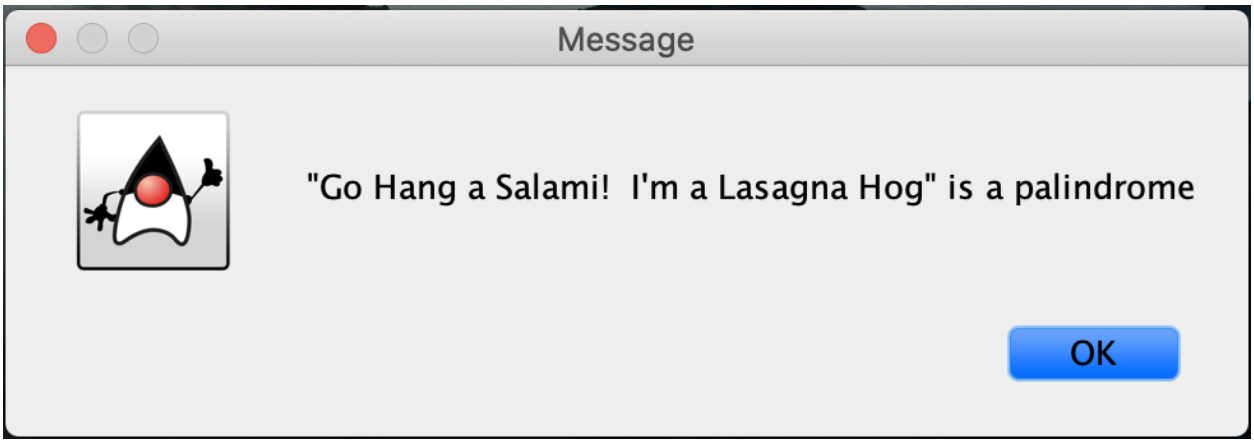
```
import javax.swing.*;

public class PalindromeDemo {

	public static void main(String[] args) {
		LinkedStackDS<Character> palindromeStack = new
LinkedStackDS<Character>();

		String inputStr;
		String replaceStr;
		String reverseStr = "";
		int test;

		do {
			inputStr = JOptionPane.showInputDialog("Input a String");

			replaceStr = inputStr.replaceAll("[^a-zA-Z]", "");
			replaceStr = replaceStr.replaceAll(",", "");
			replaceStr = replaceStr.replaceAll("!", "");
			replaceStr = replaceStr.replaceAll("'", "");

			for(int i = 0; i < replaceStr.length(); i++) {
				char c = replaceStr.charAt(i);
				palindromeStack.push(replaceStr.charAt(i));
			}

			while(!palindromeStack.isEmptyStack()) {
				Character reverseChar = palindromeStack.peek();
				reverseStr += reverseChar;
				palindromeStack.pop();
			}

			if(replaceStr.compareToIgnoreCase(reverseStr) == 0) {
				JOptionPane.showMessageDialog(null, inputStr + " is a
palindrome");
			} else {
				JOptionPane.showMessageDialog(null, inputStr + " is not a
palindrome");
			}

			test = JOptionPane.showConfirmDialog(null, "Test another palindrome?",
```

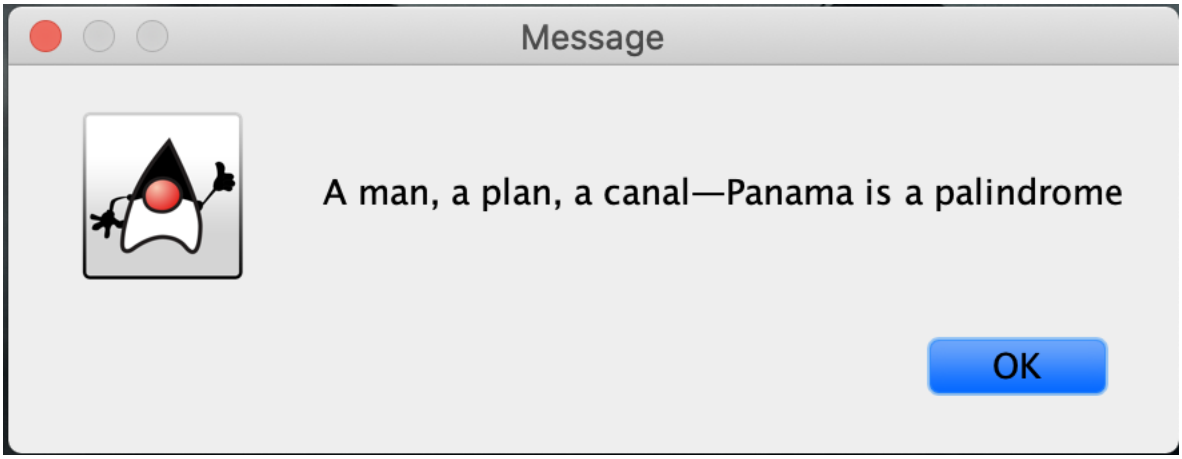
```
                                        "Try another palindrome",
JOptionPane.YES_NO_OPTION);

                        if (test == 1) {
                                JOptionPane.showMessageDialog(null, "Goodbye", "Program
Ends.",
                                                JOptionPane.INFORMATION_MESSAGE);
                        }


                }
                while(test == 0);

        }

}
```

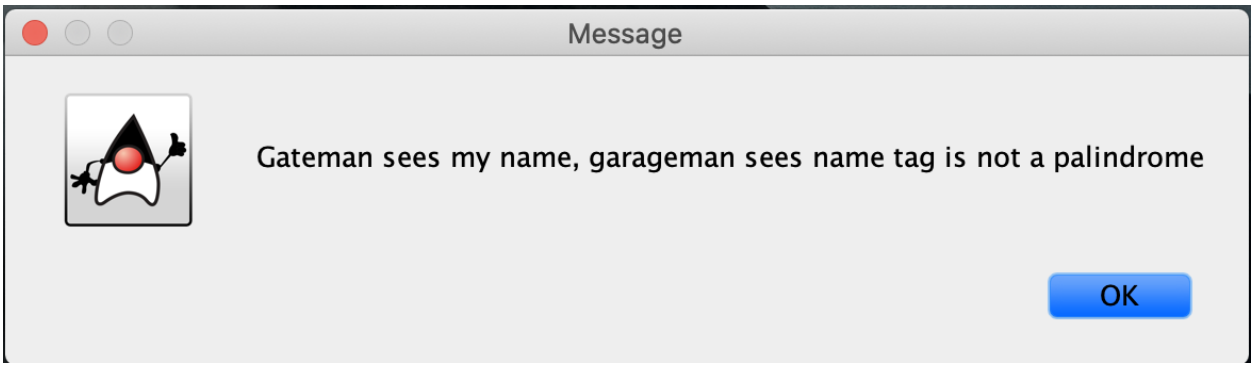


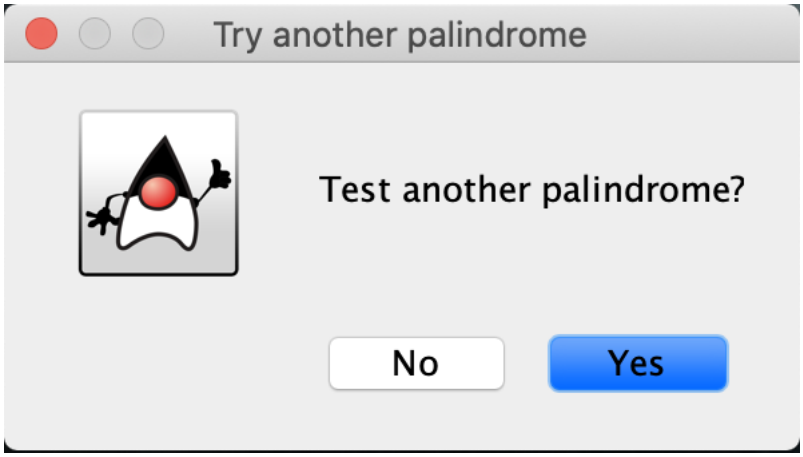



Message
Some men interpret eight memos is not a palindrome
OK


Message
"Go Hang a Salami! I'm a Lasagna Hog" is a palindrome
OK


Message
A man, a plan, a canal—Panama is a palindrome
OK


Message
Gateman sees my name, garageman sees name tag is not a palindrome
OK


Try another palindrome
Test another palindrome?
No
Yes


Program Ends.
Goodbye
OK