# Stack

Ja-Hee Kim

# Agenda

## 01 ADT

Push, pop, peek, isEmpty, clear

## 02 Examples

Algebraic expression, program stack, undo/redo etc

## 03 Linked Stack

UML model & methods

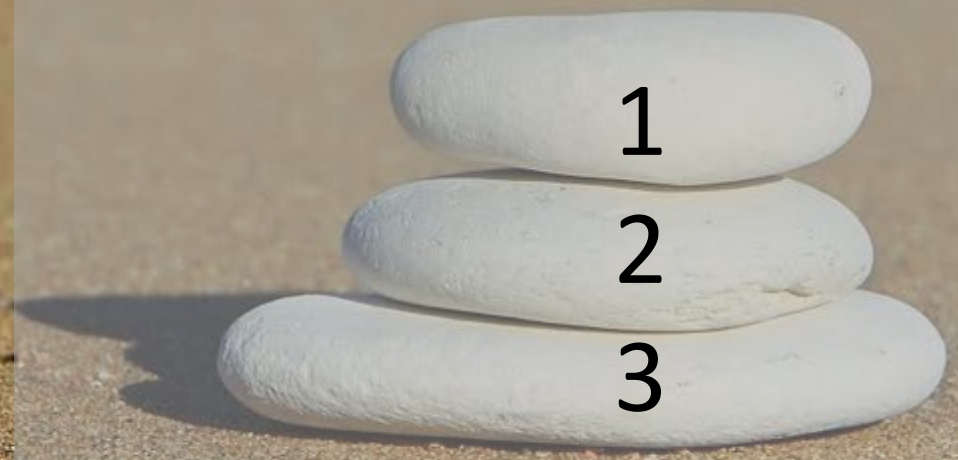
## 04 Array-based Stack

UML model & methods

# ADT of Stack

# Stack

- A stack is a data structure that removes items in the reverse order of which they were ins erted
- LIFO: Last In First Out
- A linked structure that inserts and deletes only at the head of the list is a stack



# ADT if a Stack

- push:
  =add
  - Task: add a given element to the top of the stack.
  - Input: newEntry is a new entry.
- pop:
  =remove, pull
  - Task: if the stack is not empty, delete and return its top element.
  - Output: returns the task's top entry.
- peek:
  getTop
  - Task: if the stack is not empty, return its top element.
  - Output: returns the task's top entry.
- isEmpty:
  - Task: detects whether the stack is empty.
  - Output: returns true if the stack is empty.
- clear:
  - Task: removes all entries from the stack

| Stack |
| --- |
| |
| +push(newEntry : T):void<br>+pop(): T<br>+peek(): T<br>+isEmpty(): Boolean<br>+clear(): void |

# An interface of the ADT Stack

```
public interface StackInterface<T>{
      /** Adds a new entry to the top of this stack.
      @param newEntry an object to be added to the stack */
      public void push(T newEntry);
      /** Removes and returns this stack's top entry.
      @return either the object at the top of the stack or, if the
      stack is empty before the operation, null */
      public T pop();
      /** Retrieves this stack's top entry.
      @return either the object at the top of the stack or null if
      the stack is empty */
      public T peek();
      /** Detects whether this stack is empty.
      @return true if the stack is empty */
      public boolean isEmpty();
      /** Removes all entries from this stack */
      public void clear();
} // end StackInterface
```

**Stack**

+push(newEntry : T):void
+pop(): T
+peek(): T
+isEmpty(): Boolean
+clear(): void

# Build-in library, Stack

- included in java.util package

- peek: If the stack is not empty, return its top element.
- pop: If the stack is not empty, delete and return its top element.
- push: Add a given element to the top of the stack.
- size: Return the number of elements in the stack.

# Usage of a Stack

- Demonstrating the stack methods: a stack of Toys

Red yoyo


black cookie man

Blue ninja

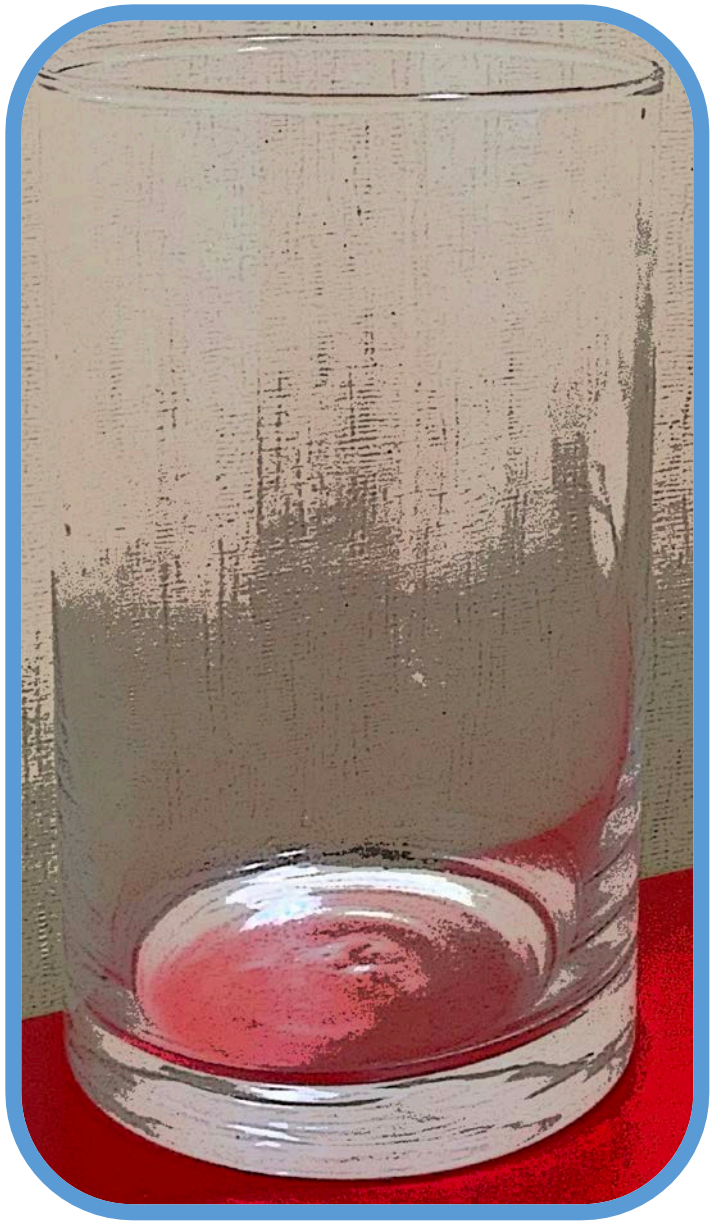
Yellow yoyo


- Demo
  1. Push


  2. Push

  3. Push

  4. Peek
  5. Size
  6. Pop
  7. Push



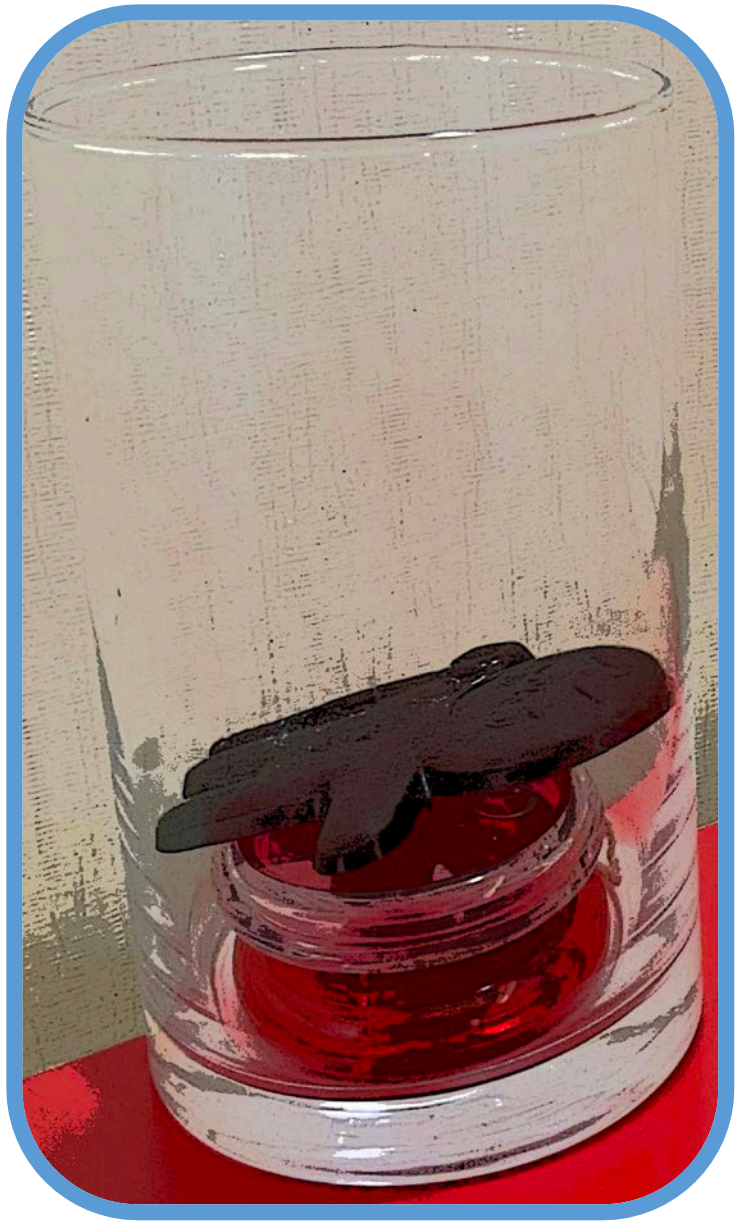
# Initializing

ToyStack.java

```
1 import java.util.*;
2 public class ToyStack {
3     public static void main(String[] args) {
4         Stack<String> toys = new Stack<>();
5     }
6 }
```


# Push


```java
import java.util.*;
public class ToyStack {
    public static void main(String[] args) {
        Stack<String> toys = new Stack<>();
        toys.push("Red YOYO");
    }
}
```



# Push

```
import java.util.*;
public class ToyStack {
    public static void main(String[] args) {
        Stack<String> toys = new Stack<>();
        toys.push("Red YOYO");
        toys.push("Black COOKIE MAN");
    }
}
```

# Push

```
import java.util.*;
public class ToyStack {
    public static void main(String[] args) {
        Stack<String> toys = new Stack<>();
        toys.push("Red YOYO");
        toys.push("Black COOKIE MAN");
        toys.push("Blue NINJA");
    }
}
```

# Peek, size

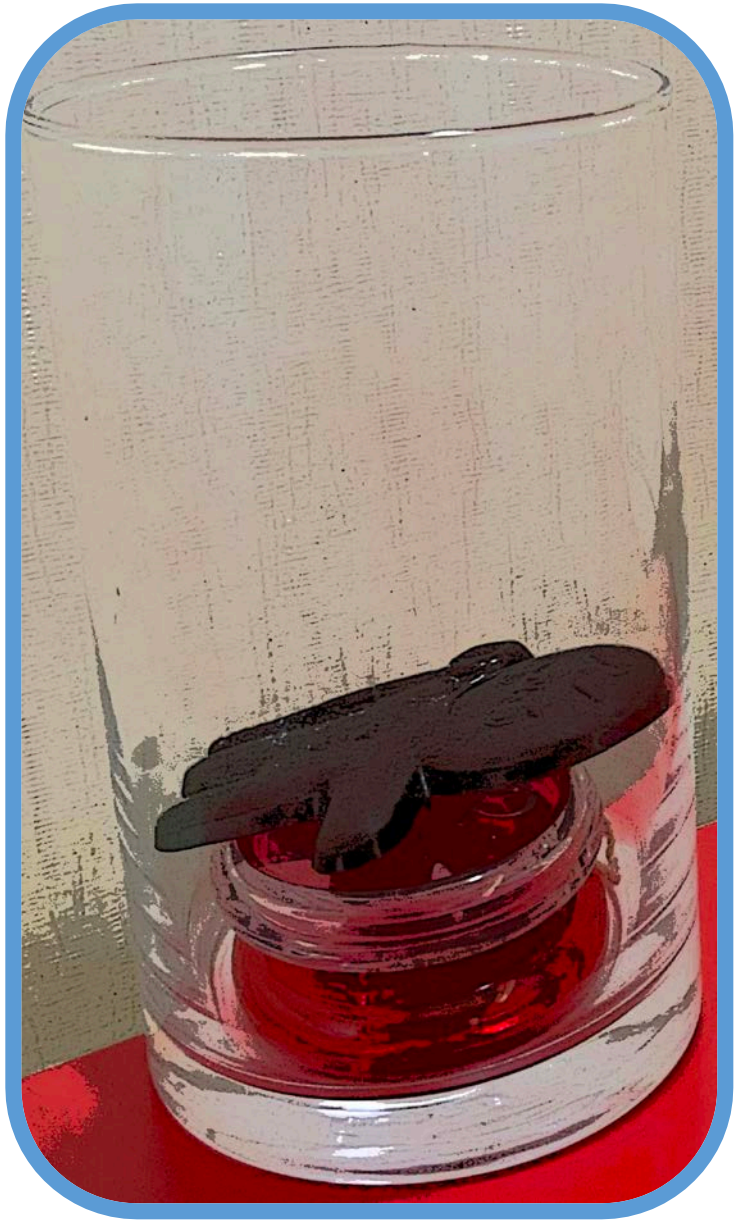
ToyStack.java

```
1  import java.util.*;
2  public class ToyStack {
3      public static void main(String[] args) {
4          Stack<String> toys = new Stack<>();
5          toys.push("Red YOYO");
6          toys.push("Black COOKIE MAN");
7          toys.push("Blue NINJA");
8          System.out.println("peek: "+ toys.peek());
9          System.out.println("size: "+ toys.size());
10     }
11 }
```

Problems Javadoc Declaration Console

<terminated> ToyStack [Java Application] C:₩Program Files₩Java₩jdk-14.0.2₩bin₩javaw.exe (2020. 9. 27. 오후 4:10:50 – 오후

```
peek: Blue NINJA
size: 3
```



# pop

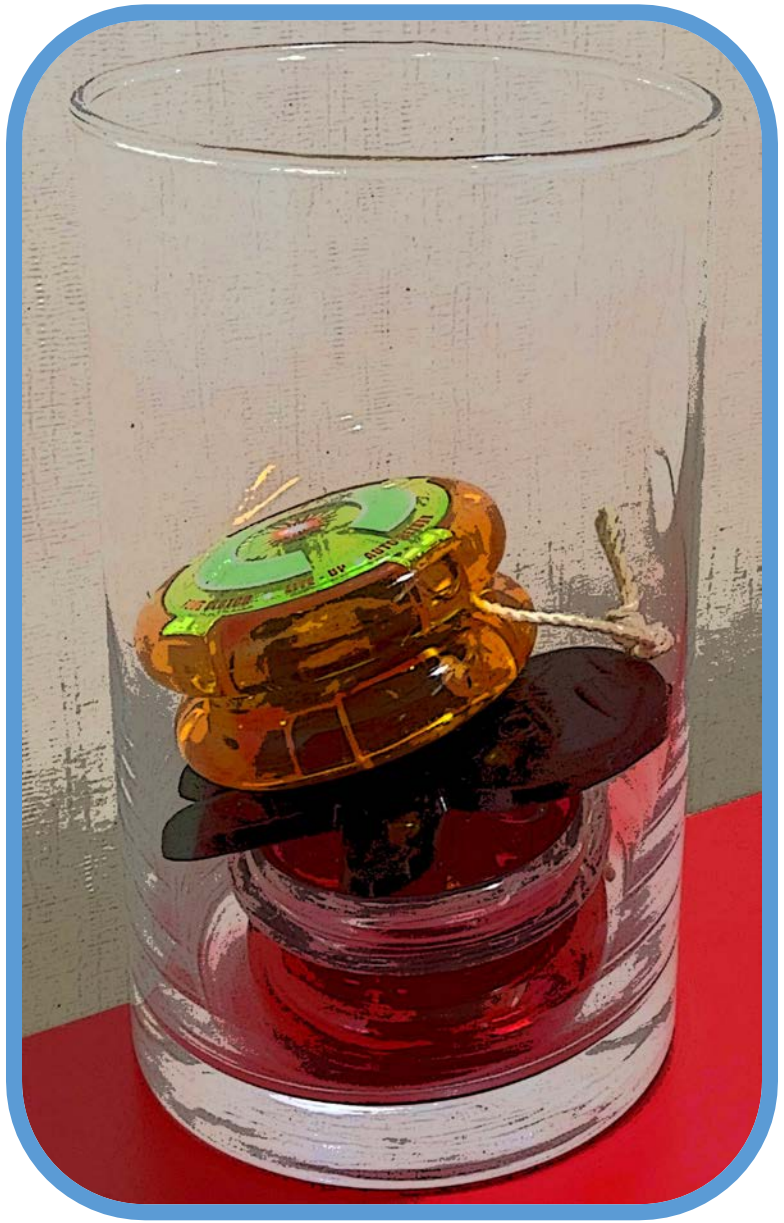
*ToyStack.java

```
1 import java.util.*;
2 public class ToyStack {
3     public static void main(String[] args) {
4         Stack<String> toys = new Stack<>();
5         toys.push("Red YOYO");
6         toys.push("Black COOKIE MAN");
7         toys.push("Blue NINJA");
8         System.out.println("peek: "+ toys.peek());
9         System.out.println("size: "+ toys.size());
10        System.out.println("pop: "+ toys.pop());
11    }
12 }
```

Problems Javadoc Declaration Console

<terminated> ToyStack [Java Application] C:₩Program Files₩Java₩jdk-14.0.2₩bin₩javaw.exe (2020. 9. 27. 오후 4:12:40

```
peek: Blue NINJA
size: 3
pop: Blue NINJA
```



# Push


ToyStack.java

```
1 import java.util.*;
2 public class ToyStack {
3     public static void main(String[] args) {
4         Stack<String> toys = new Stack<>();
5         toys.push("Red YOYO");
6         toys.push("Black COOKIE MAN");
7         toys.push("Blue NINJA");
8         System.out.println("peek: "+ toys.peek());
9         System.out.println("size: "+ toys.size());
10        System.out.println("pop: "+ toys.pop());
11        toys.push("Yellow YOYO");
12        System.out.println(toys.toString());
13    }
14 }
```

Problems Javadoc Declaration Console

<terminated> ToyStack [Java Application] C:₩Program Files₩Java₩jdk-14.0.2₩bin₩javaw.exe (2020. 9. 26. 오후 5:10:01 – 오

```
peek: Blue NINJA
size: 3
pop: Blue NINJA
[Red YOYO, Black COOKIE MAN, Yellow YOYO]
```

# Design Issue: pop or peek from an empty stack

- Return null
  - Methods such as peek and pop must b ehave reasonably when the stack is em pty.
  - In that case, the methods cannot retur n anything, so return null.

- Throws exception

**pop**

`public E pop()`

Removes the object at the top of this stack and returns that object as the value of this function.

**Returns:**

The object at the top of this stack (the last item of the `Vector` object).

**Throws:**

`EmptyStackException` - if this stack is empty.

**peek**

`public E peek()`

Looks at the object at the top of this stack without removing it from the stack.

**Returns:**

the object at the top of this stack (the last item of the `Vector` object).

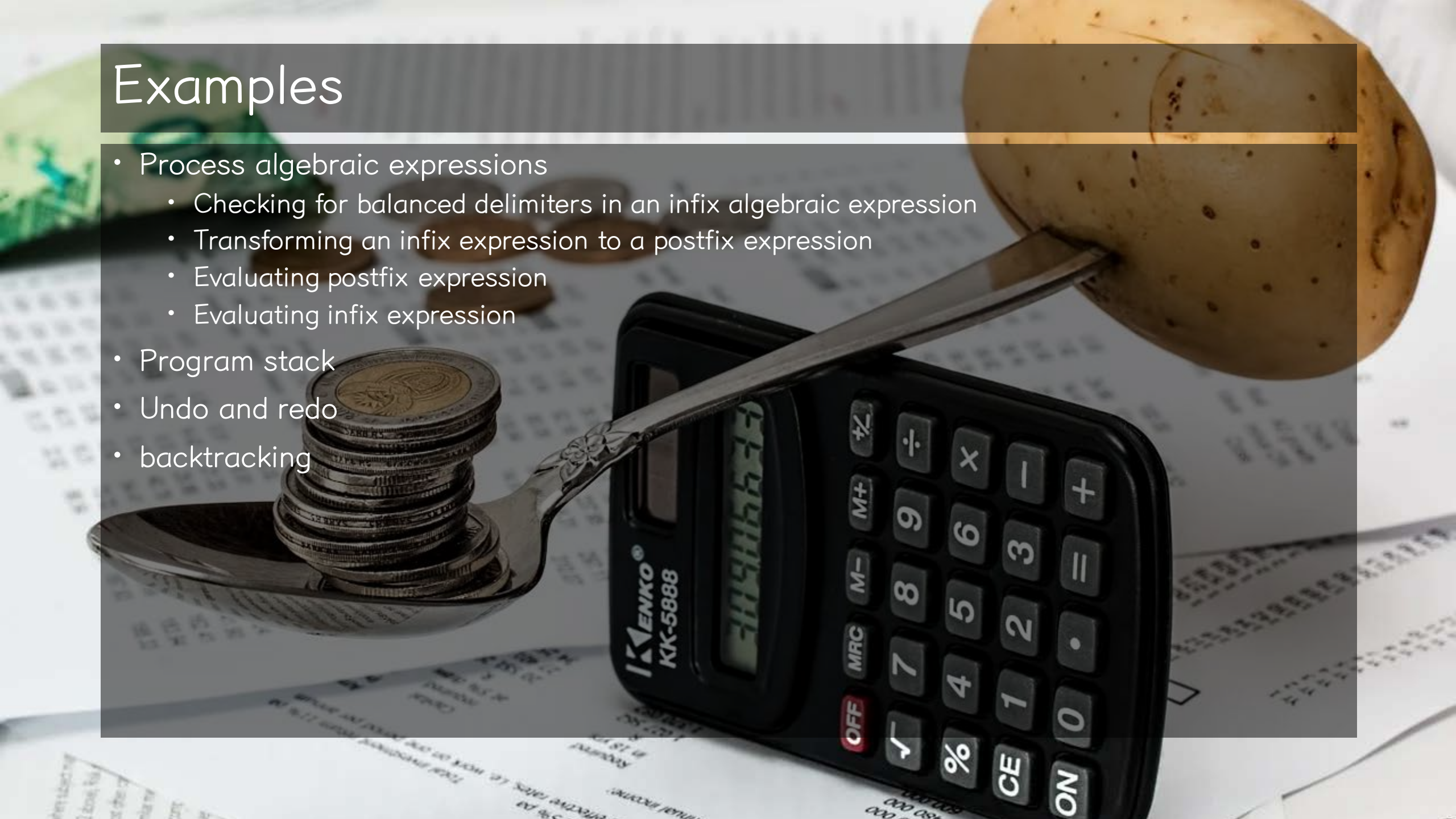
**Throws:**

`EmptyStackException` - if this stack is empty.

# Examples of Stack

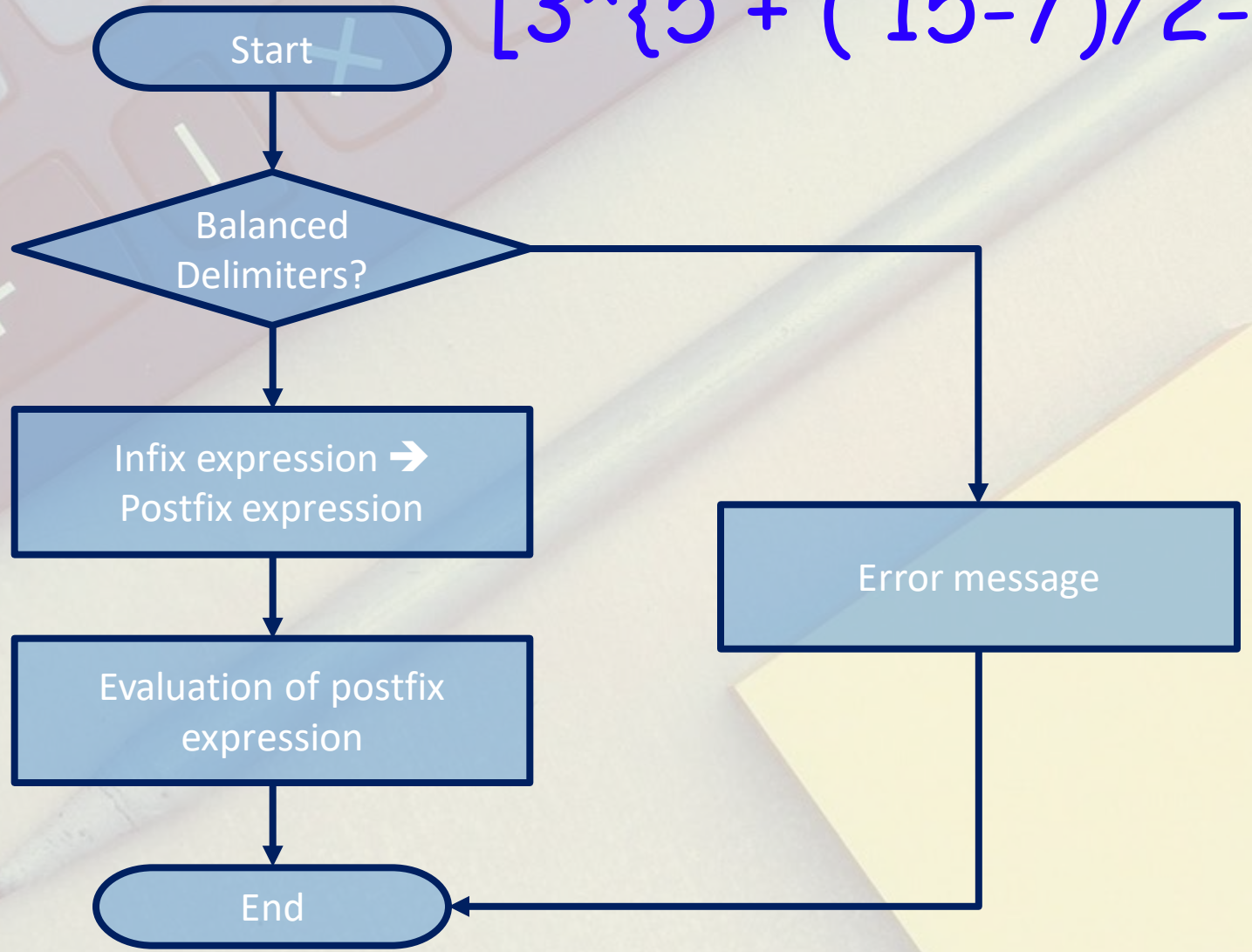
# Examples

- Process algebraic expressions
  - Checking for balanced delimiters in an infix algebraic expression
  - Transforming an infix expression to a postfix expression
  - Evaluating postfix expression
  - Evaluating infix expression
- Program stack
- Undo and redo
- backtracking



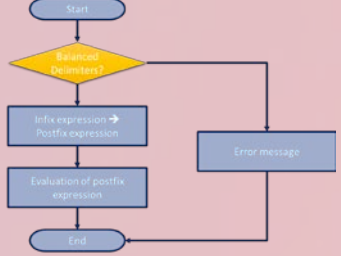
# Process algebraic expressions

[3*{5 + ( 15-7)/2-9} +1]^2



# Algebraic expression

- Infix notation
  - Example: a + b
  - A binary operator between its operands
- Prefix notation
  =Polish notation by Jan Łukasiewicz (1920s)
  - Example: + a b
  - A binary operator before its operands
- Postfix notation
  =reverse Polish notation RPN
  Example: a b +
  - A binary operator before its operands

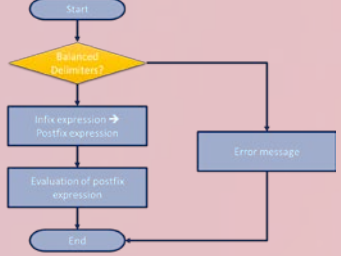



# Checking for balance delimiters in an infix notation

- The type of delimiters
  - [ {

- Balanced expression

[3*{5 + ( 15-7)/2-9} +1]^2

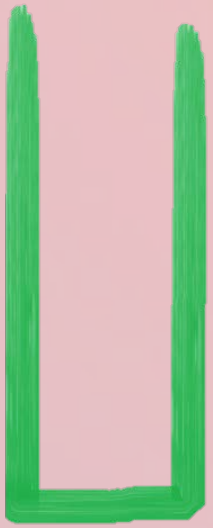



# Checking for balance delimiters in an infix notation

- The type of delimiters
  - [ {

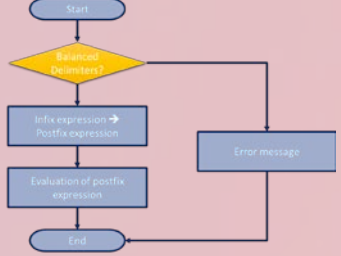
- intersecting pairs of delimiters.

[3*{5 + ( 15-7}/2-9) +1]^2



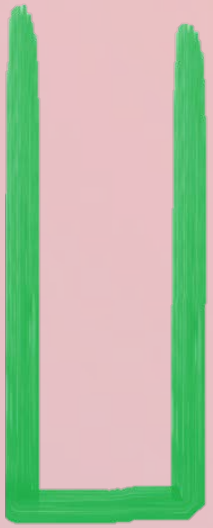
- Unbalance expression

[3*{5 + ( 15-7)/2-9}

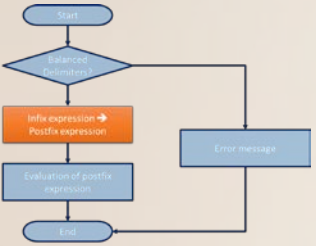



# Implementation

[3*{5 + ( 15-7)/2-9} +1]^2



```java
/**
 * Decide whether the parentheses, brackets,
 * and braces in a string occurs in left/right pairs.
 * @param ex a string to be checked
 * @return True if the delimiters are paired correctly
 */
public boolean checkBalance(String ex) {
    // list of considering open parentheses
    Stack<Character> parenStack = new Stack<Character>();
    int charCount = ex.length();
    boolean isBalanced = true; // the absence of delimiters is balanced
    char nextCharacter = ' '; // the next character in expression
    for(int i = 0 ; isBalanced && i < charCount ; i++) {
        nextCharacter = ex.charAt(i);
        switch(nextCharacter) {
        case '(' : case '[': case '{':
            // push nextCharacter onto stack
            parenStack.push(nextCharacter);
            break;
        case ')': case ']': case '}' :
            // when we meet the right pair but there is no left pair in the stack
            if(parenStack.isEmpty()) isBalanced = false;
            // if the stack is not empty, the value of isBalanced is the
            // result of comparison between delimiters
            // of the next character and the top of the stack
            else isBalanced = isPaired(parenStack.pop(), nextCharacter);
            break;
        }
    }
    //if the stack still has open delimiters after see all string,
    // the delimiters are not paired correctly.
    if(!parenStack.isEmpty()) isBalanced = false;
    return isBalanced;
}
/**
 * Decide whether open and close make a pair correctly.
 * @param open one of open parentheses, brackets, or braces
 * @param close one of close parentheses, brackets, or braces
 * @return True if the given characters, open and close, form
 * a pair of parentheses, brackets, or braces
 */
private boolean isPaired(char open, char close) {
    return (open == '(' && close == ')') ||
           (open == '[' && close == ']') ||
           (open == '{' && close == '}') ;
}
```

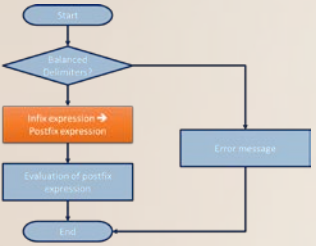



# Infix notation → postfix notation

- Examples of infix and postfix expressions

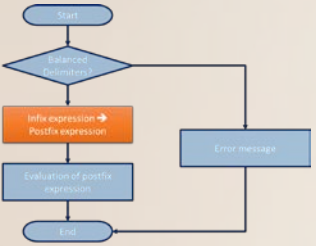
| infix | postfix |
|---|---|
| a+b | a b + |
| (a+b)*c | a b + c* |
| a+b*c | a b c * + |

- The evaluation of an infix expression is complex because of
  - the precedence of the operators
  - parentheses.
  - Solution:
    - Fully parenthesized infix notation: ((a+b)*c), (a+(b*c))
    - Infix expression ➔ postfix expression




# Conversion rule

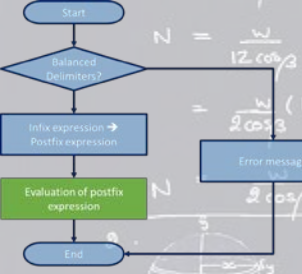
| Next character | Action |
|---|---|
| Operand such as number(1, 2, 3 …) or variables | Append each operand to the end of the output expression. |
| Operator ^ | Push ^ onto the stack. |
| Operator +, -, * , or / | Pop operators from the stack, appending them to the output expression, until the stack is empty or its top entry has a lower precedence than the new operator. Then push the new operator onto the stack. |
| Open parenthesis, bracket, brace | Push it onto the stack. |
| Close parenthesis, bracket, brace | Pop operators from the stack and append them to the output expression until an open parenthesis is popped. Discard both parentheses. |




# Implementation

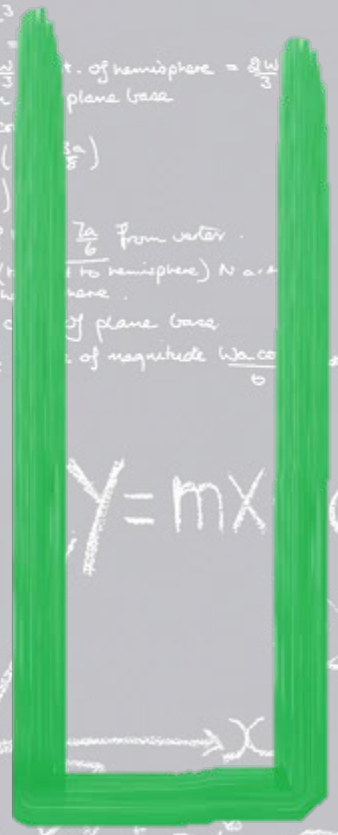
- [3*{5 + ( 15-7)/2-9} +1]^2
- Result:

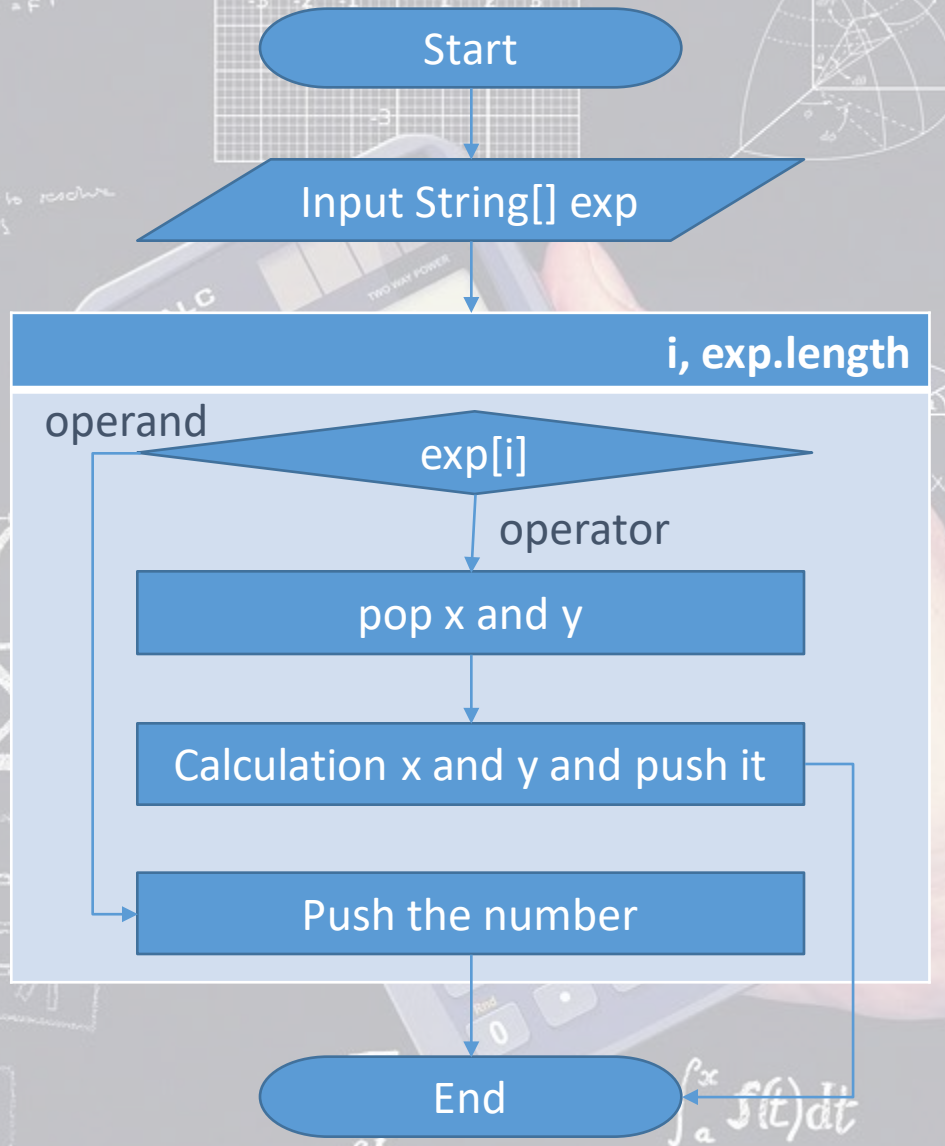
```
public String[] infix2postfix(String[] infix) {
    Stack<String> operatorStack = new Stack<>();    // a new empty stack
    System.out.println(Arrays.toString(infix));
    String[] postfix = new String[infix.length]; // a new empty string
    int size = 0;
    try {
        for(int i = 0 ; i < infix.length ; i++) {
            String next = infix[i]; // next non character left to parse
            switch(next) {
            case "^":
                operatorStack.push(next);
                break;
            case "+": case "-": case "*": case "/":
                while(!operatorStack.isEmpty() &&
                      precedence(next) <= precedence(operatorStack.peek()))
                        postfix[size++]=operatorStack.pop();
                operatorStack.push(next);
                break;
                case "(": case "{":case "[":
                    operatorStack.push(next);
                    break;
                case ")": case "}": case "]":
                    String topOperator = operatorStack.pop();
                    while("({[".indexOf(topOperator)<0) {
                        postfix[size++] = topOperator;
                        topOperator = operatorStack.pop();
                    }
                    break;
                default:
                    postfix[size++]=next;
                    break;
            }
        }
    }catch (Exception e) {
        System.err.println(e.getMessage());
        System.exit(-1);
    }
    while(!operatorStack.isEmpty())
        postfix[size++]= operatorStack.pop();
    String[] result = new String[size];
    for(int i = 0 ; i < size ; i++)
        result[i]=postfix[i];
    return result;
}
```




# Algorithm

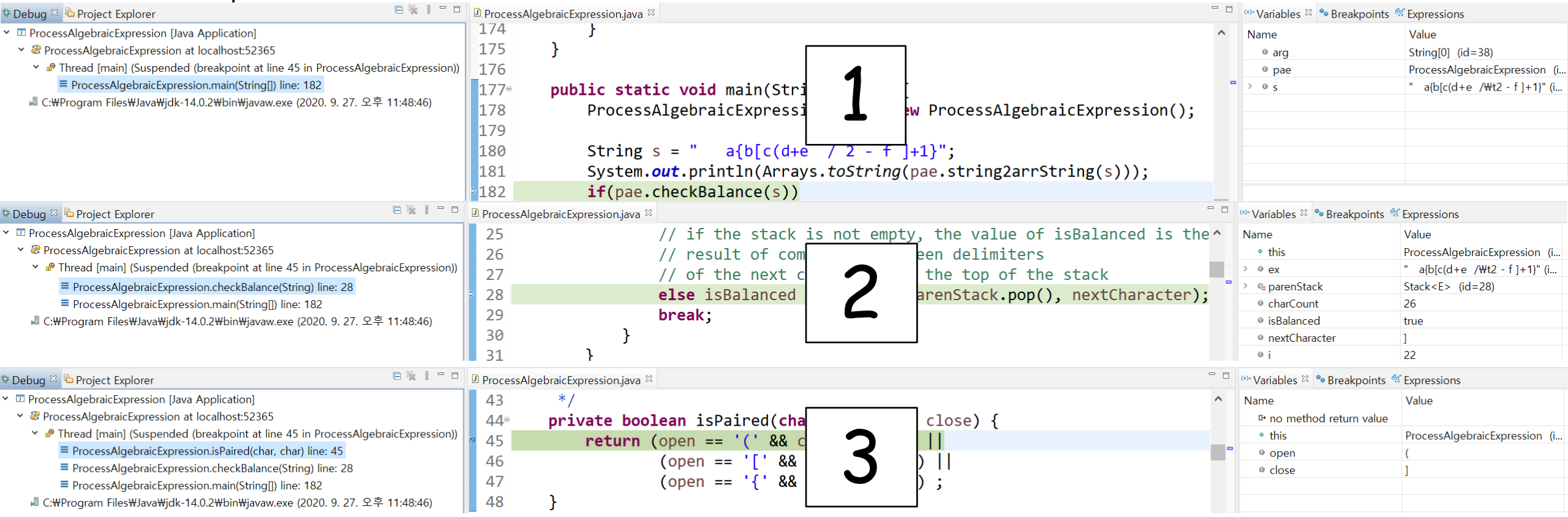
3 5 15 7 - 2 / + 9 - * 1 + 2 ^





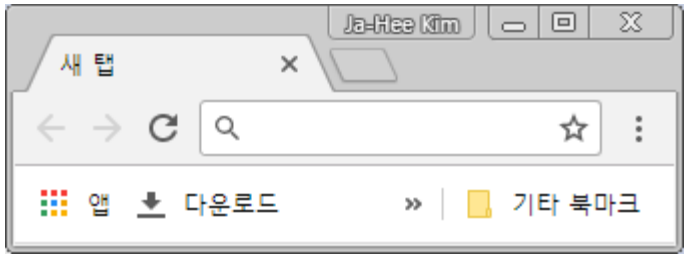
# Program stack

- Terminology
  - Program counter references the current instruction.
  - Activation record(frame) is an object tha run-time environment creates for the method.
  - Program stack is a stack onto which an activation record is pushed
- Example: main ➔ checkBalance ➔ isPaired

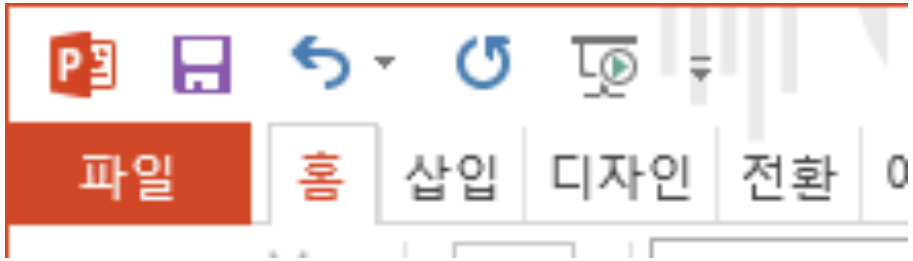


# Undo and redo

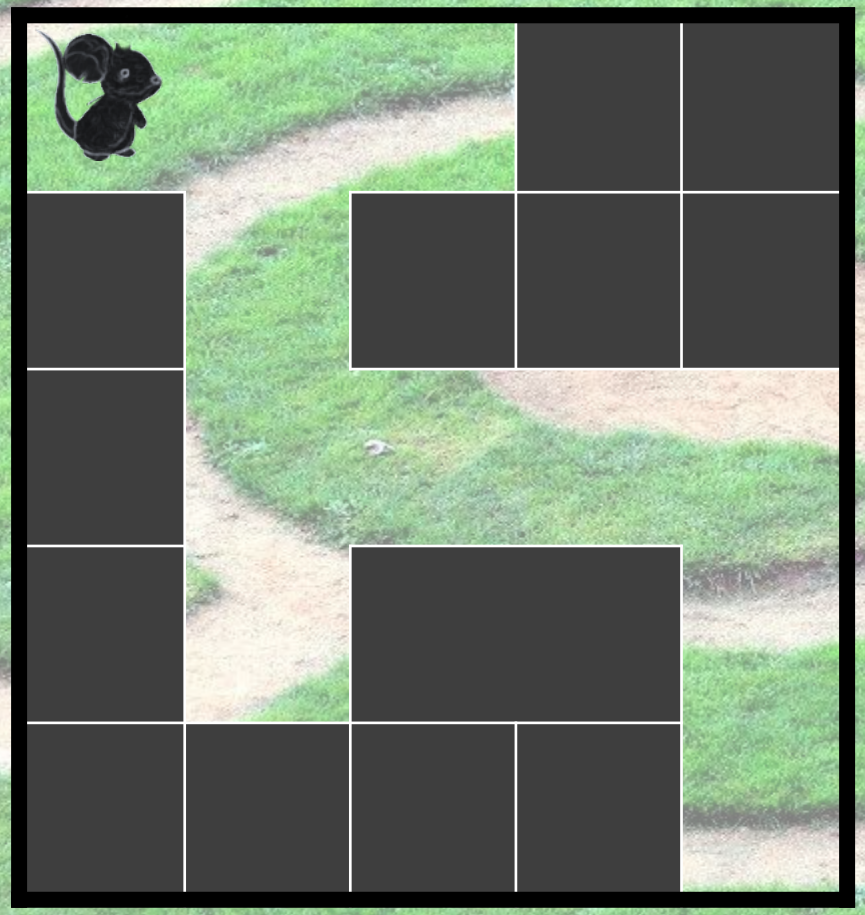
- Backward & Forward button: site address

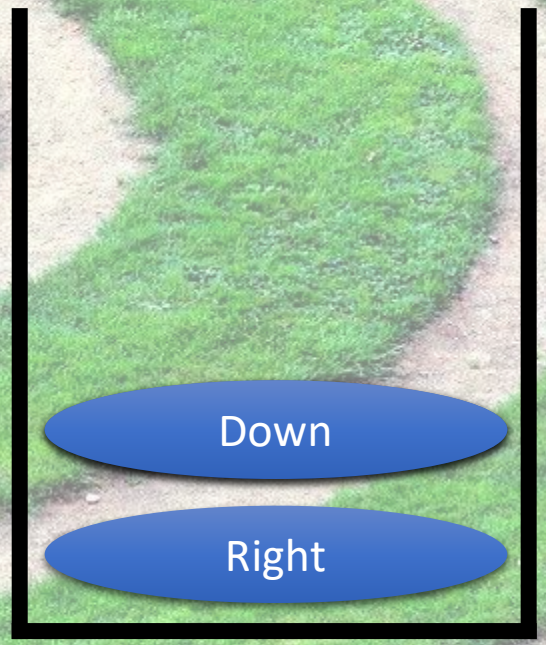




- Undo and redo: action



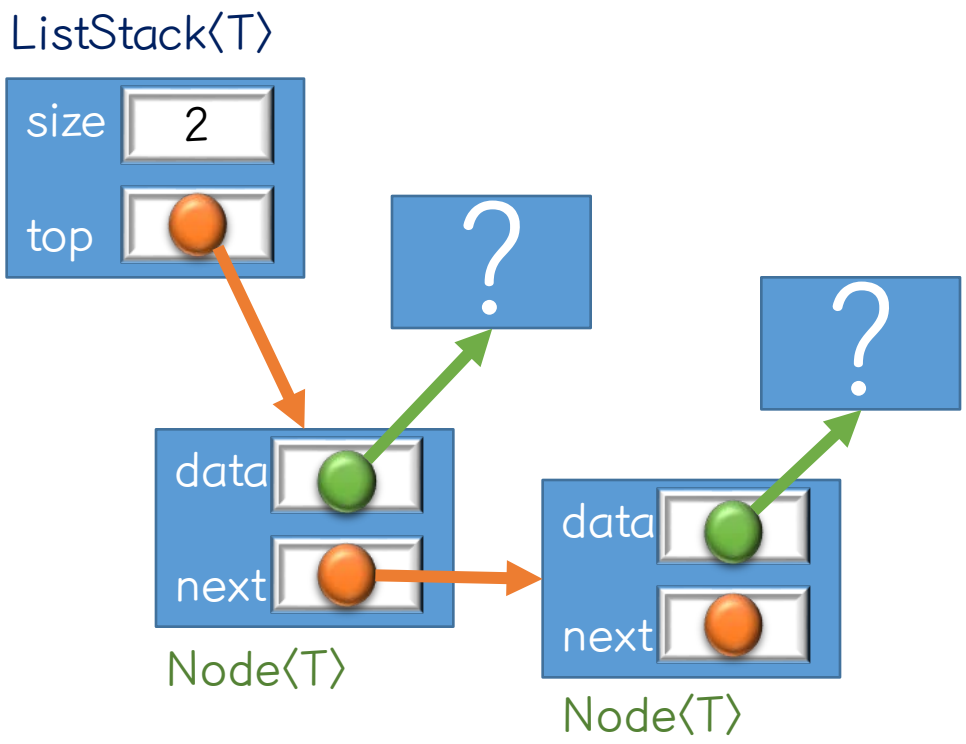
# Backtracking





# Linked Stack

# Outline of the class




the size of the stack

top points the last pushed node

**ListStack<T>**

- size: int
- top: Node<T>

+ push(T newItem)
+ peek(): T
+ pop(): T
+ isEmpty(): Boolean
+clear(): void
+size():int
+toString(): String

**Node<T>**

- data: T
- next: Node<T>

+ Node<T>()
+ Node<T>(T data, Node<T> next)

# Method push

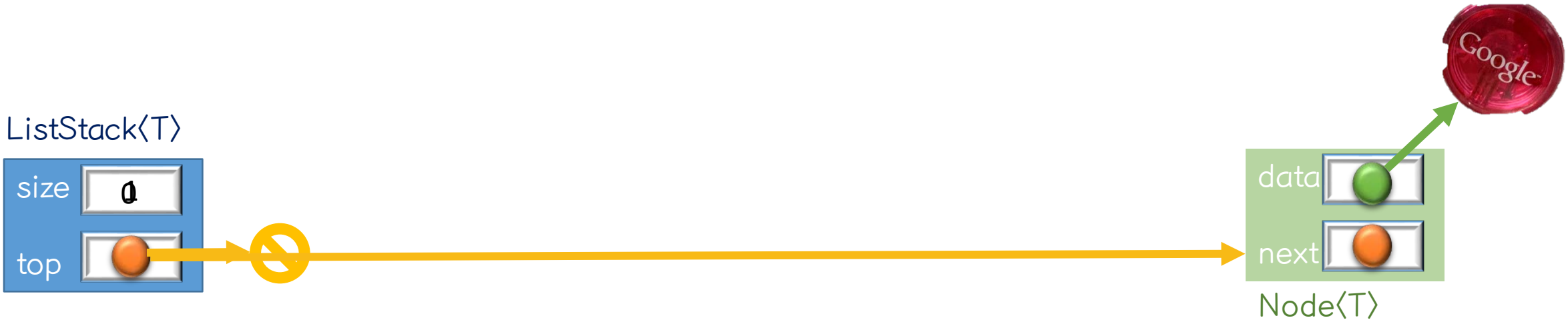
- Client program

```
toys.push("Red YOYO");
```

- In class LinkedStack

```
public void push(T newEntry) {
     top = new Node(newEntry, top);
     size++;
}
```

O(1)



# Method push

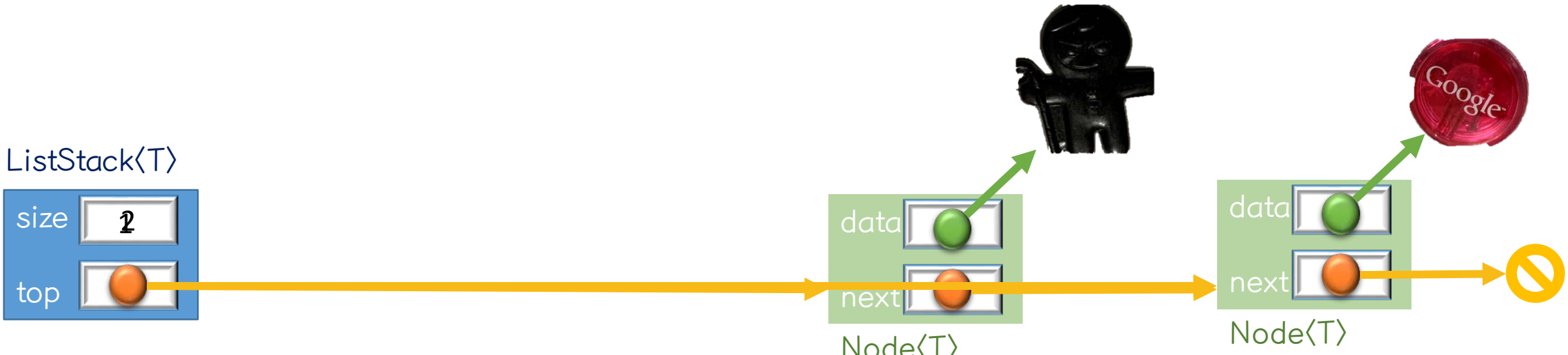
- Client program

```
toys.push("Red YOYO");
toys.push("Black COOKIE MAN");
```

- In class LinkedStack

```
public void push(T newEntry) {
     top = new Node(newEntry, top);
     size++;
}
```

O(1)



# Method push

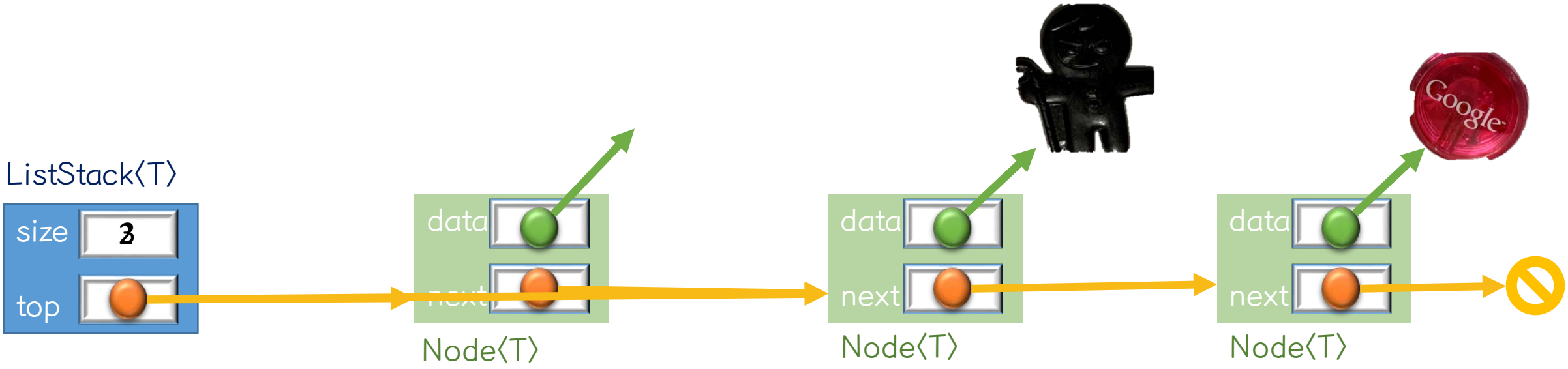
- Client program

```
toys.push("Red YOYO");
toys.push("Black COOKIE MAN");
toys.push("Blue NINJA");
```

- In class LinkedStack

```
public void push(T newEntry) {
     top = new Node(newEntry, top);
     size++;
}
```

O(1)



# Method peek and size

- Client program

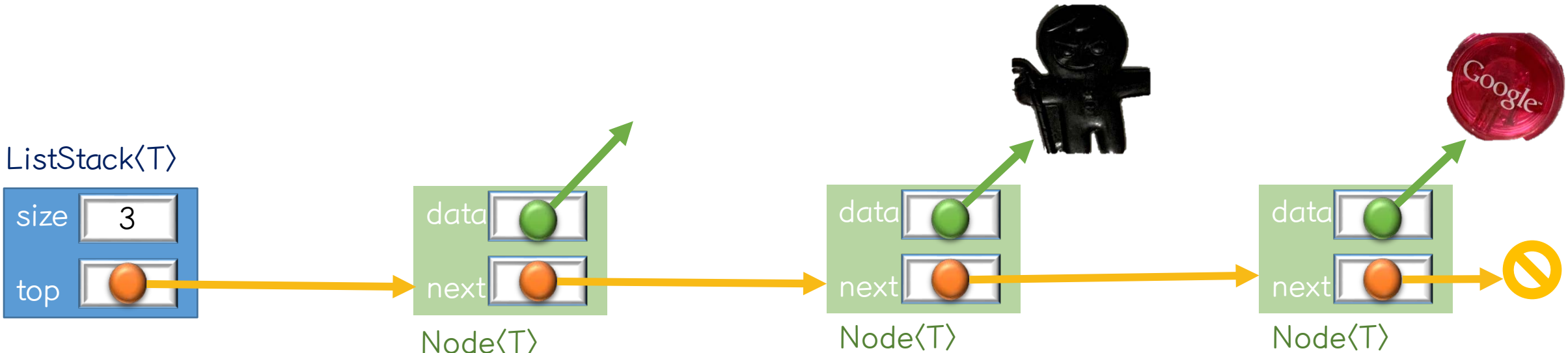
```
System.out.println("peek: "+ toys.peek());
System.out.println("size: "+ toys.size());
```

- In class stack

```
public T peek() {
    T result = null;
    if(top!=null) result = top.data;
    return result;
}
public int size() {
        return size;
}
```

O(1)

O(1)



# Method pop

- Client program

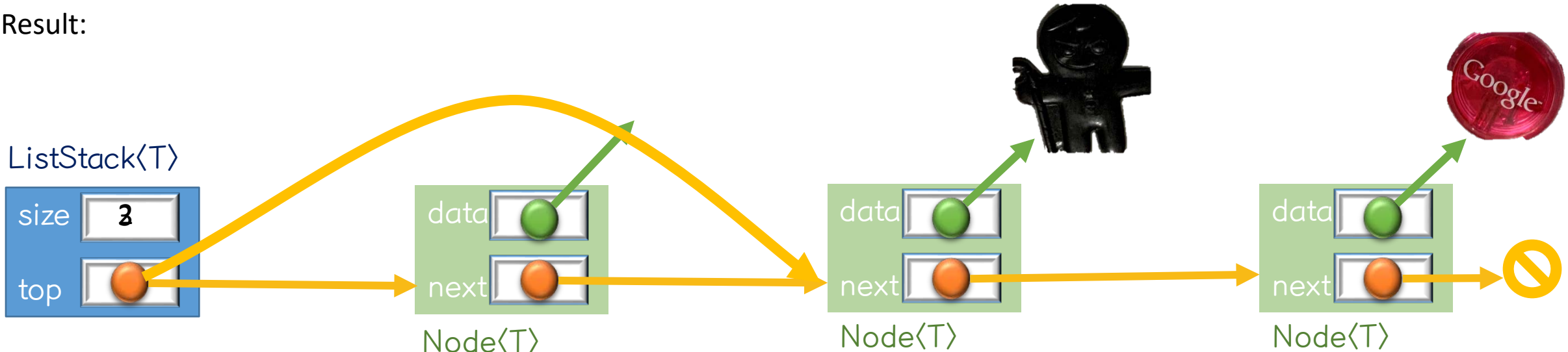
```
System.out.println("pop: "+ toys.pop());
```

- In class stack

```
public T pop() {
    T result = peek();
    if(top!=null) {
        top = top.next;
        size--;
    }
    return result;
}
```

O(1)

Result:



# Method isEmpty

- Client program

```
System.out.println("isEmpty: "+
                        toys.isEmpty());
```

- In class stack

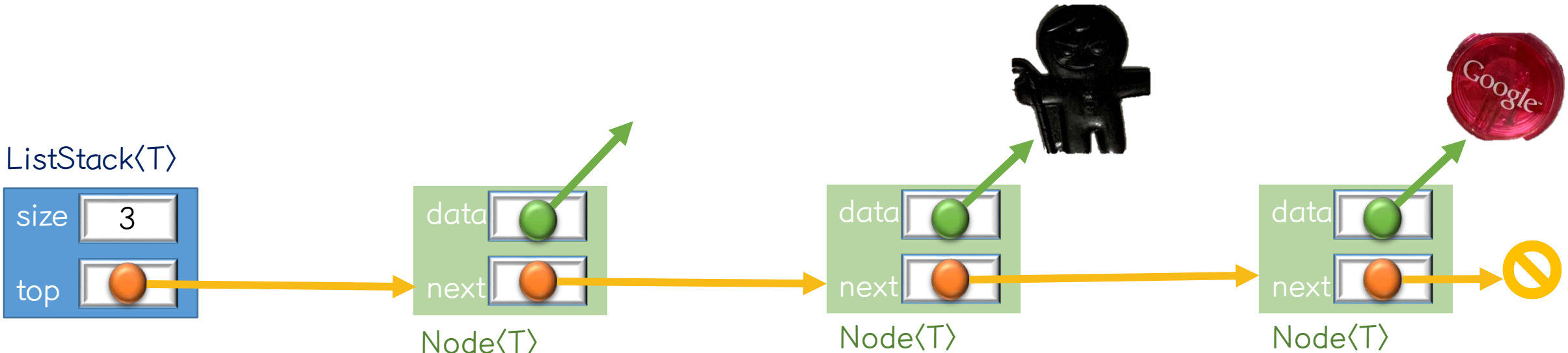
```
public boolean isEmpty() {
        return size==0;
}
```

O(1)

or

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

O(1)



# Method clear

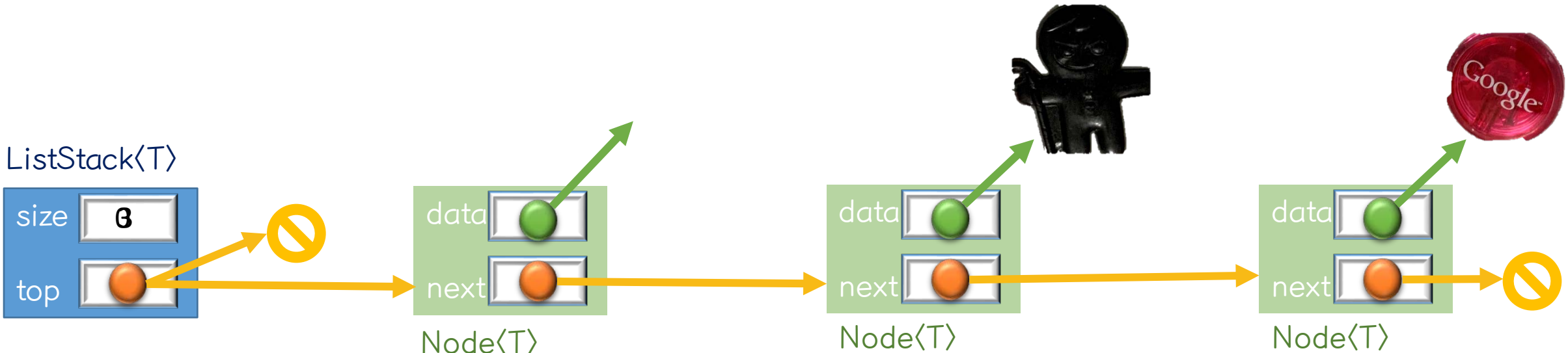
- Client program

```
toys.clear()
```

- In class stack

```
public void clear() {
    top = null;
    size = 0;
}
```

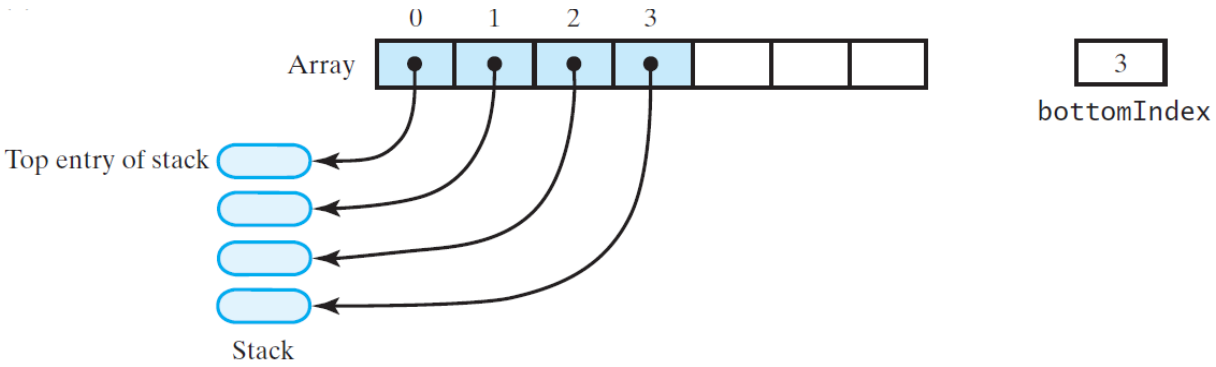
O(1)



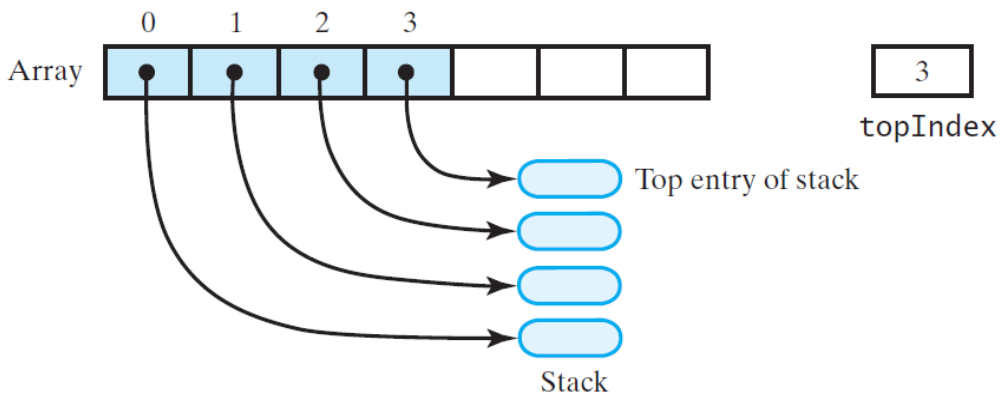
# Array Stack

# Design Issue: which the array's first element references

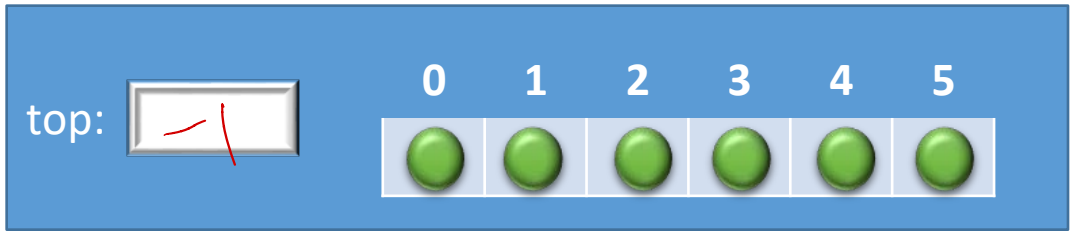
- The stack's top entry




- The stack's bottom entry




# Outline of an ArrayStack




the index for the last data

**ArrayStack<T>**

- top: int
- item: T[]
- integrityOK: boolean
- DEFAULT_CAPACITY: int
- MAX_CAPACITY: int

the array for the data

+ push(T newItem)
+ pop(): T
+ peek(): T
+ isEmpty(): Boolean
+clear(): void
+size():int
+toString(): String
- isFull(): Boolean

# Method push

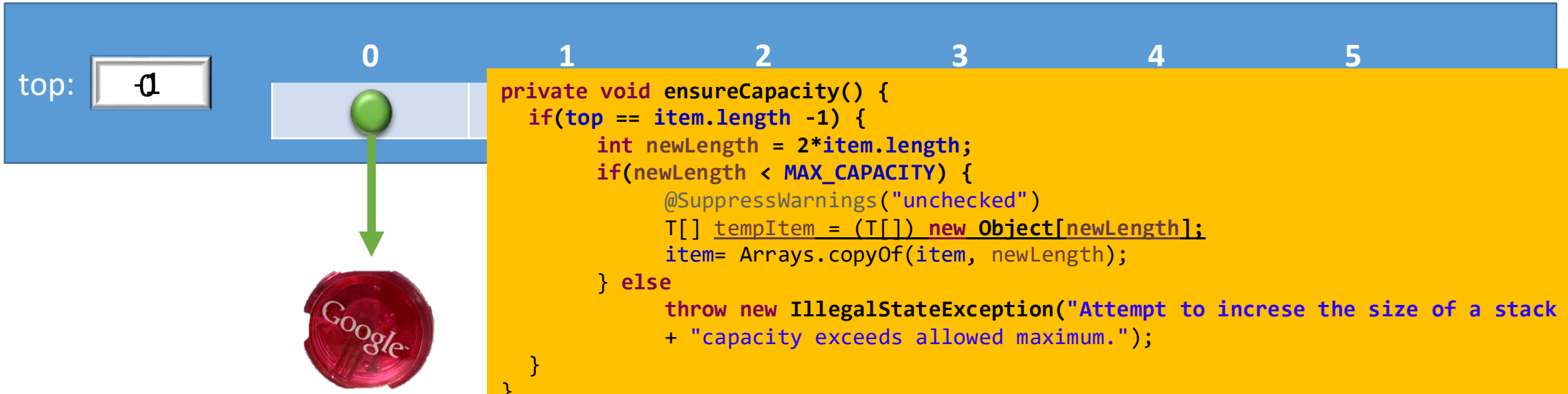
- Client program

```
toys.push("Red YOYO");
```

- In class stack

```
public void push(T newEntry) {
    checkIntegrity();
    ensureCapacity();
    item[++top]=newEntry;
}
```

Usually O(1), if a stack is full, O(n)




```
private void ensureCapacity() {
  if(top == item.length -1) {
       int newLength = 2*item.length;
       if(newLength < MAX_CAPACITY) {
            @SuppressWarnings("unchecked")
            T[] tempItem = (T[]) new Object[newLength];
            item= Arrays.copyOf(item, newLength);
       } else
            throw new IllegalStateException("Attempt to increse the size of a stack whose "
            + "capacity exceeds allowed maximum.");
  }
}
```

# Method push

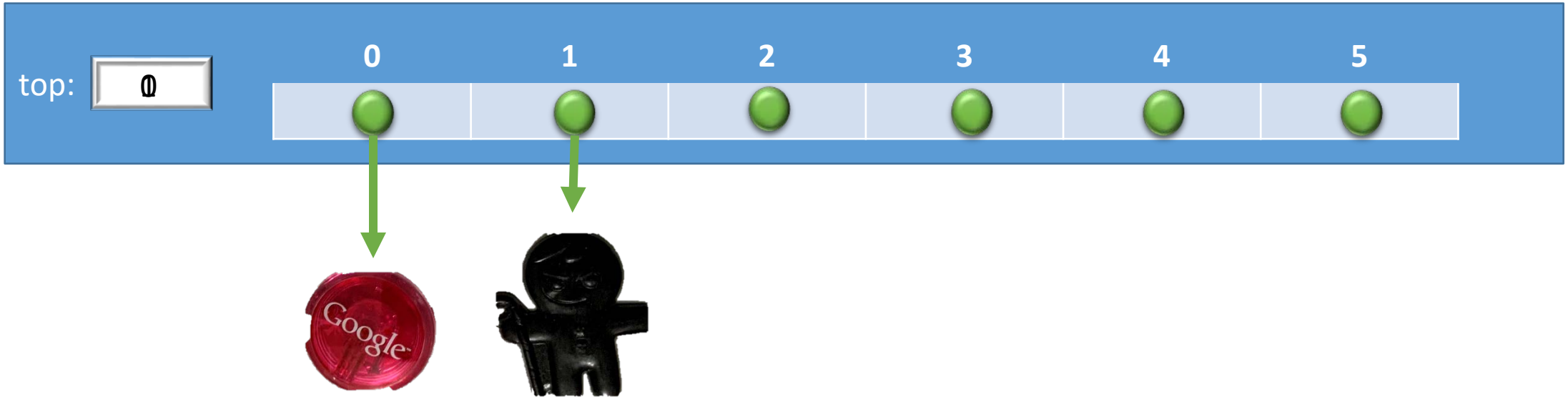
- Client program

```
toys.push("Red YOYO");
toys.push("Black COOKIE MAN");
```

- In class stack

```
public void push(T newEntry) {
    checkIntegrity();
    ensureCapacity();
    item[++top]=newEntry;
}
```

Usually O(1), if a stack is full, O(n)



# Method push

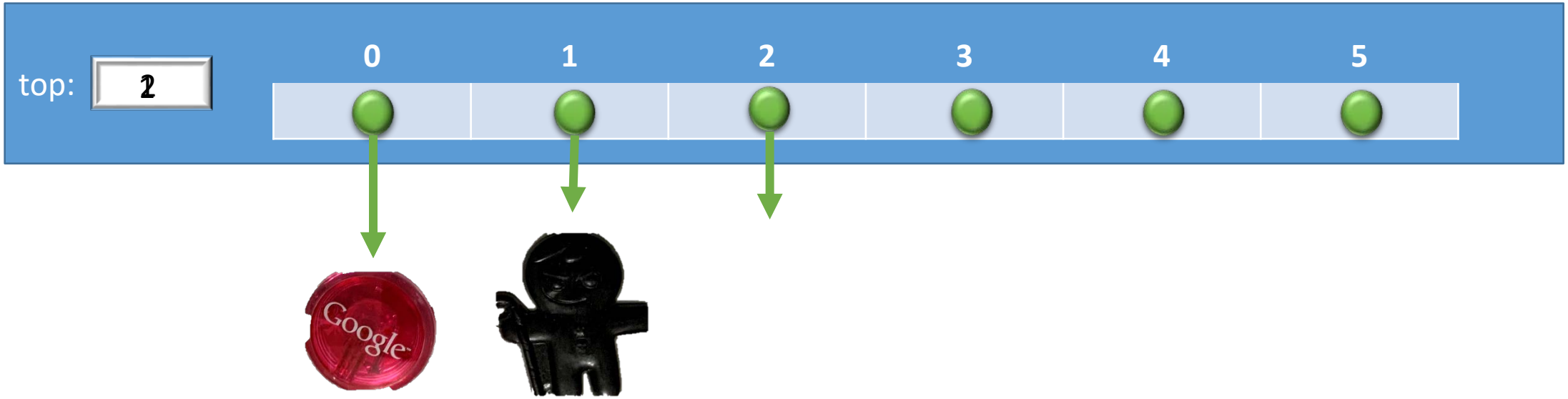
- Client program

```
toys.push("Red YOYO");
toys.push("Black COOKIE MAN");
toys.push("Blue NINJA");
```

- In class stack

```
public void push(T newEntry) {
    checkIntegrity();
    ensureCapacity();
    item[++top]=newEntry;
}
```

Usually O(1), if a stack is full, O(n)



# Method peek and size

- Client program

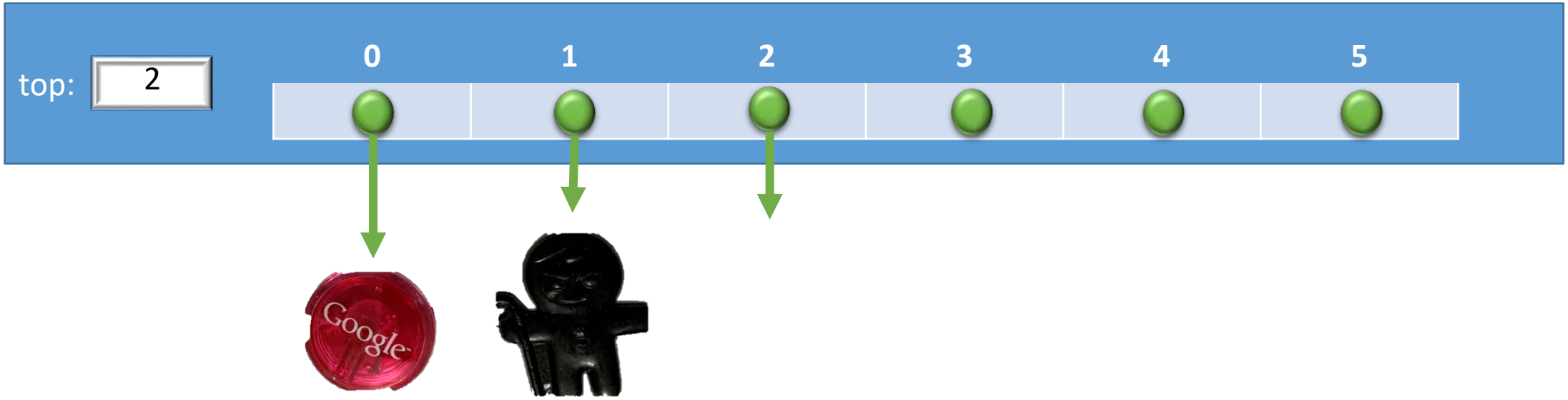
```
System.out.println("peek: "+ toys.peek());
System.out.println("size: "+ toys.size());
```

- In class stack

```
public T peek() {
     if(isEmpty()) return null;
     else return item[top];
}
public int size() {
          return top+1;
}
```

O(1)

O(1)



# Method pop

- Client program

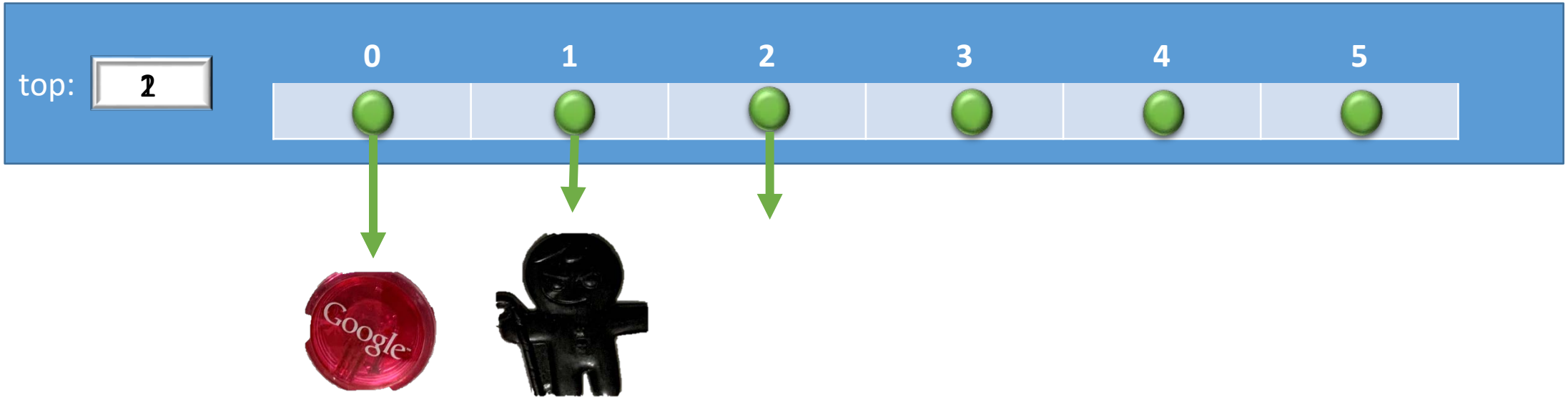
```
System.out.println("pop: "+ toys.pop());
```

Result:

- In class stack

```
public T pop() {
     if(isEmpty()) return null;
     else {
          T result = item[top];
          item[top] = null;
          top--;
          return result;
     }
}
```

O(1)



# Method isEmpty

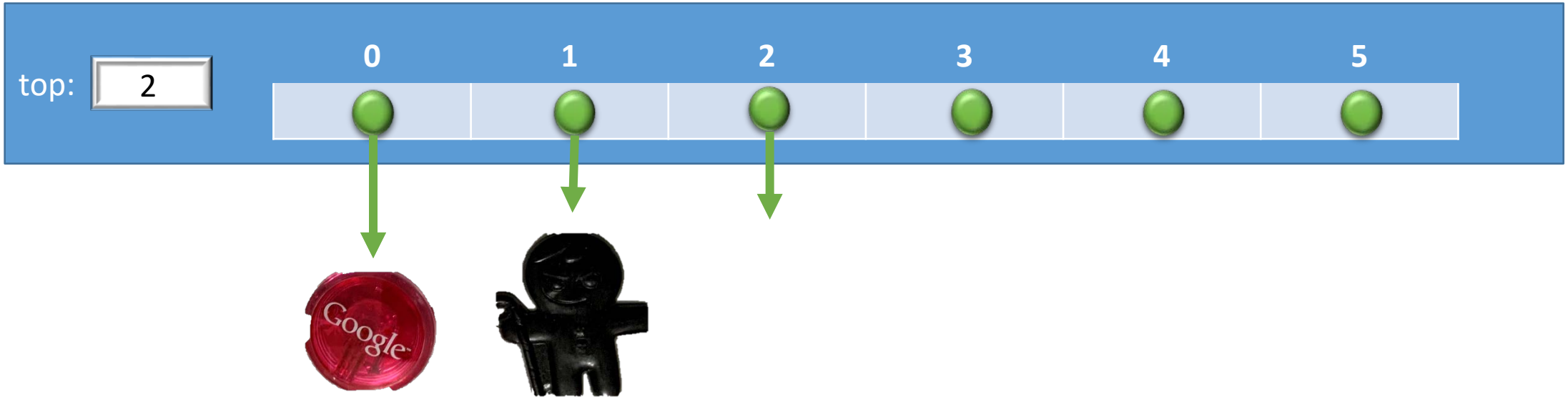
- Client program

```
System.out.println("isEmpty: "+
                        toys.isEmpty());
```

- In class stack

```
public boolean isEmpty() {
        return top<0;
}
```

O(1)



# Method clear

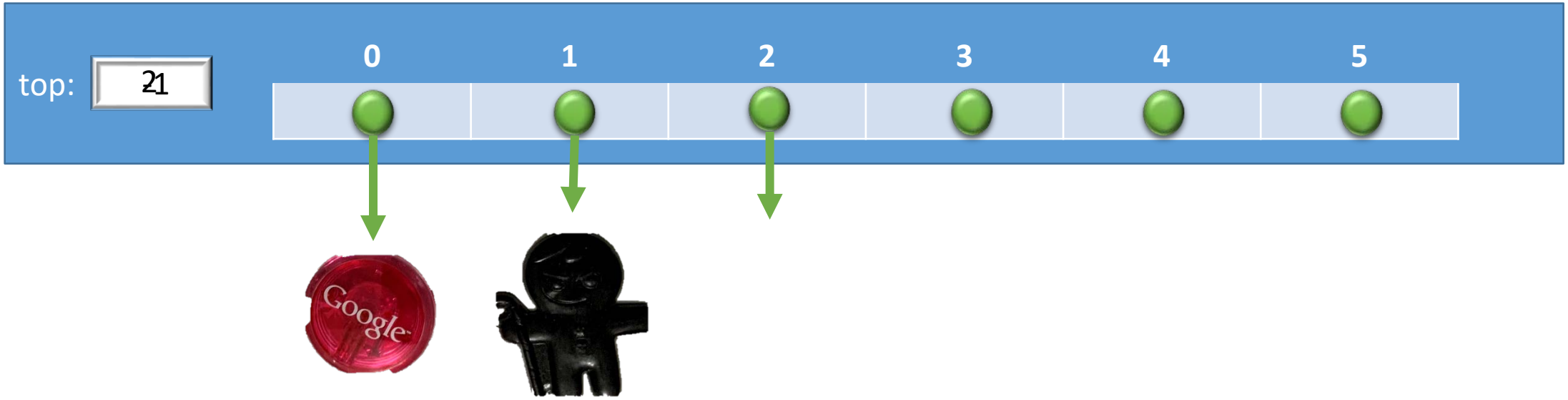
- Client program

```
toys.clear()
```

- In class stack

```
public void clear() {
    @SuppressWarnings("unchecked")
    T[] tempItem = (T[]) new Object[item.length];
    item = tempItem;
    top = -1;
}
```

O(1)



# Thank you