CSE214 Data Structures
Stacks

YoungMin Kwon

SUNY Korea
The State University of New York

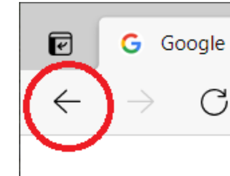
Stacks

- Stack
  - Stack is a collection of objects that are inserted and removed according to the Last-In, First-Out (LIFO) principle
- Main operations on stacks
  - Push: adding an element to a stack
  - Pop: removing the most recently added entry from a stack

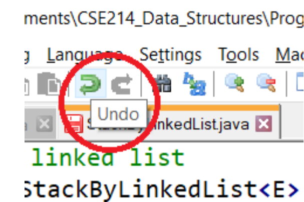
SUNY Korea

Examples of Stacks

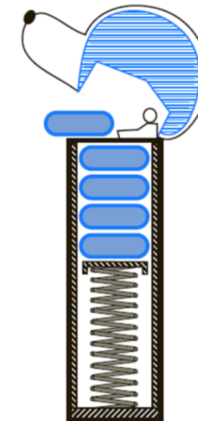
- Internet Web browsers
  - Address of recently visited sites



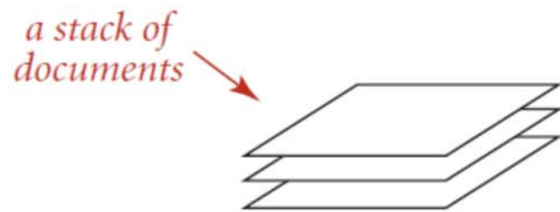
- Text editors
  - Undo mechanism

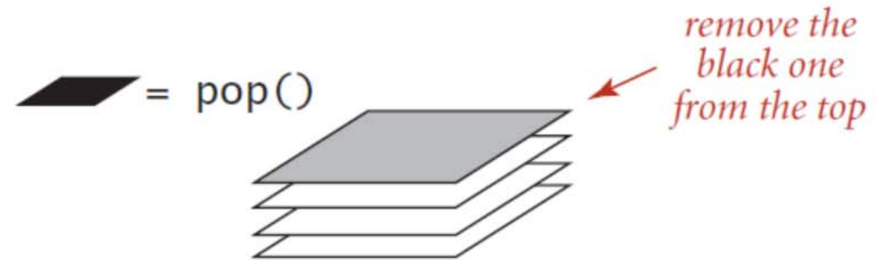


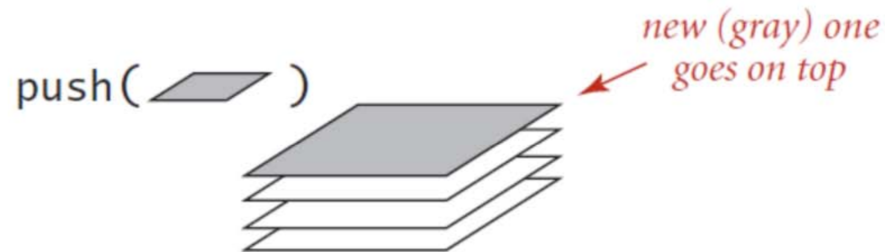
- PEZ® candy dispenser:

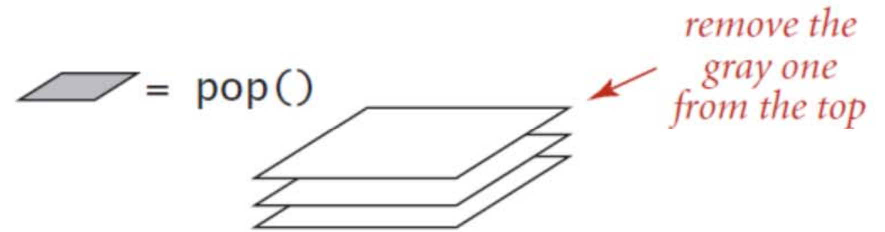


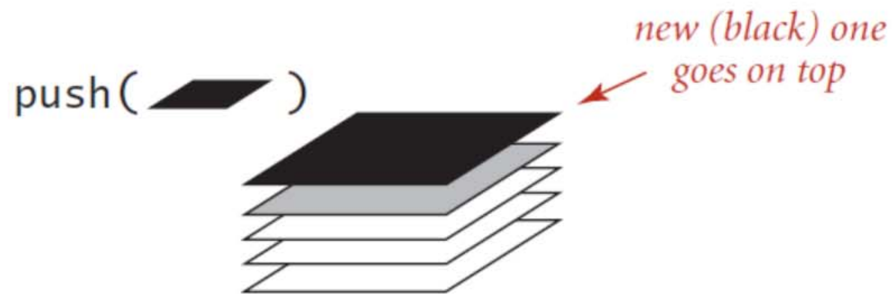
Stack Operations











SUNY Korea
The State University of New York

Stack Abstract Data Type

push(e): Adds element e to the top of the stack.

pop( ): Removes and returns the top element from the stack (or null if the stack is empty).

top( ): Returns the top element of the stack, without removing it (or null if the stack is empty).

size( ): Returns the number of elements in the stack.

isEmpty( ): Returns a boolean indicating whether the stack is empty.

SUNY Korea
The State University of New York

Stack Operation Example

Method	Return Value	Stack Contents
push(5)	–	(5)
push(3)	–	(5, 3)
size()	2	(5, 3)
pop()	3	(5)
isEmpty()	false	(5)
pop()	5	()
isEmpty()	true	()
pop()	null	()
push(7)	–	(7)
push(9)	–	(7, 9)
top()	9	(7, 9)
push(4)	–	(7, 9, 4)
size()	3	(7, 9, 4)
pop()	4	(7, 9)
push(6)	–	(7, 9, 6)
push(8)	–	(7, 9, 6, 8)
pop()	8	(7, 9, 6)

SUNY Korea

Interface Stack

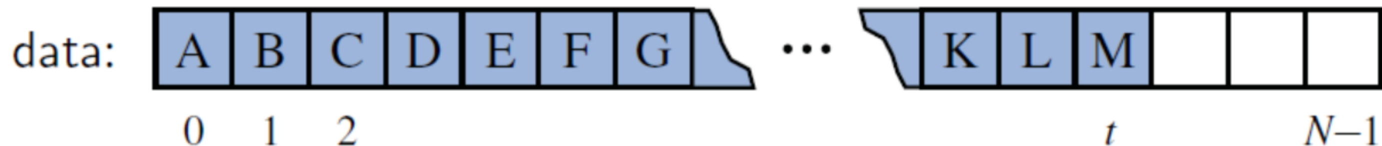
```java
public interface Stack<E> {
    public int size();
    public boolean isEmpty();
    public void push(E e);
    public E top();
    public E pop();
}
```

Our Stack ADT	Class java.util.Stack	
size()	size()	
isEmpty()	empty()	⇐
push(e)	push(e)	
pop()	pop()	
top()	peek()	⇐

SUNY Korea

Array-Based Stack Implementation

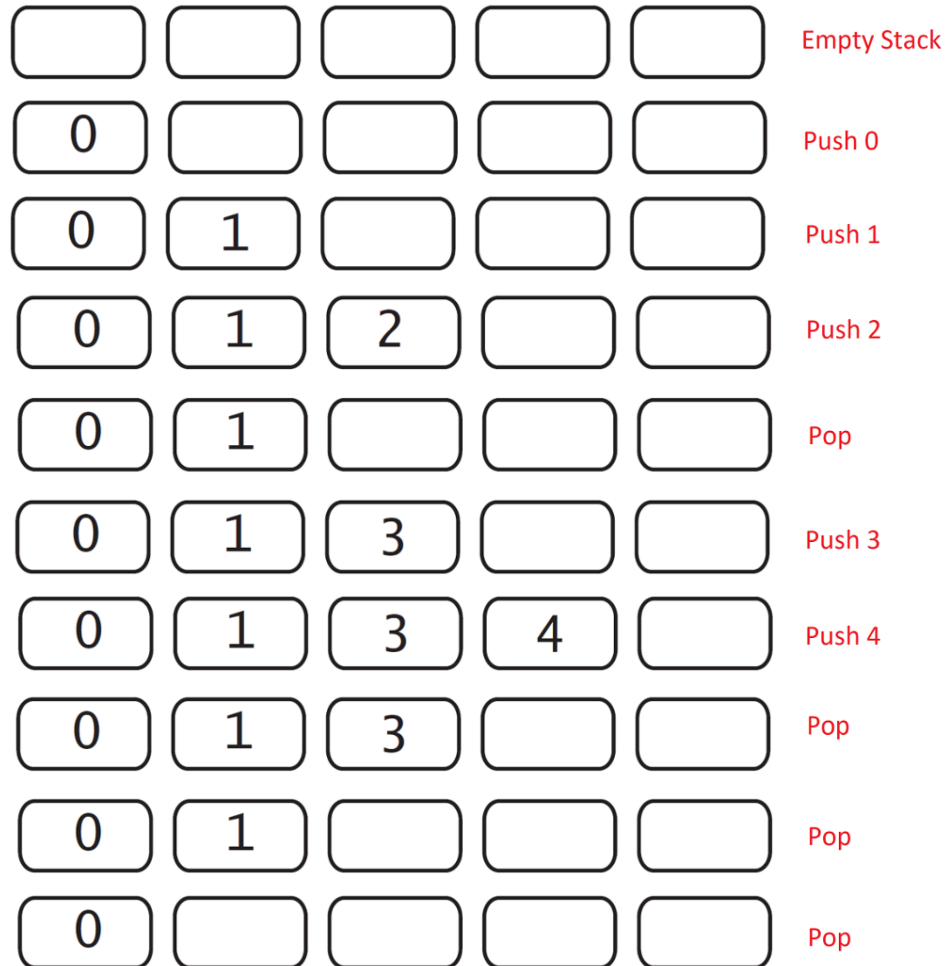
- Direction of stack growth
  - Bottom of the stack: `data[0]`
  - Top of the stack: `data[sp]` `(sp: stack pointer)`



  - Big-Oh for push and pop operations?
  - Big-Oh if the direction of stack growth is reversed?

SUNY Korea

Array-Based Stack Implementation



SUNY Korea

```java
@SuppressWarnings("unchecked")
public class StackByArray<E> implements Stack<E> {
    private static final int defCapacity = 1;
    private int capacity;
    private E[] arr;         //stack data
    private int sp;          //stack pointer

    public StackByArray() {
        this(defCapacity);
    }

    public StackByArray(int capacity) {
        this.capacity = capacity;
        arr = (E[])new Object[capacity];
        sp = 0;
    }

    //dynamic array
    private void resize(int cap) {
        E[] tmp = (E[])new Object[cap];
        for(int i = 0; i < sp; i++)
            tmp[i] = arr[i];
        arr = tmp;
        capacity = cap;
    }
```

SUNY Korea

```
    //interface Stack
    public int size()         { return sp; }
    public boolean isEmpty() { return sp == 0; }

    public void push(E e) {
        if(sp == capacity)
            resize(capacity * 2);   //dynamic array
        arr[sp++] = e;
    }

    public E top() {
        if(isEmpty())
            throw new IndexOutOfBoundsException("empty stack");
        return arr[sp-1];
    }

    public E pop() {
        if(isEmpty())
            throw new IndexOutOfBoundsException("empty stack");
        E e = arr[--sp];
        arr[sp] = null; //to help garbage collection
        return e;
    }
}
```

SUNY Korea

Using Adapter Design Pattern

```java
//Stack using DynamicArrayList
public class StackByArrayAdapter<E> implements Stack<E> {
    private List<E> list; //adapter design pattern

    public StackByArrayAdapter() {
        list = new DynamicArrayList<E>();
    }

    //interface Stack
    public int size()        { return list.size(); }
    public boolean isEmpty() { return list.isEmpty(); }
    public void push(E e)    { list.add(list.size(), e); }
    public E top()           { return list.get(list.size()-1); }
    public E pop()           { return list.remove(list.size()-1); }
}
```

SUNY Korea

Big-Oh of Array-Based Stack

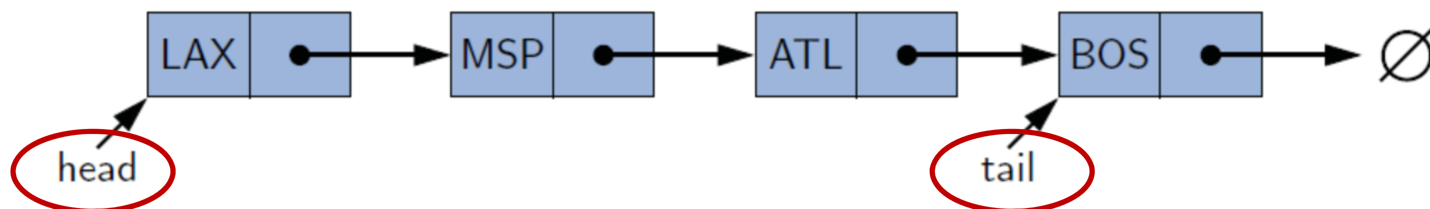
Method	Running Time
size	$O(1)$
isEmpty	$O(1)$
top	$O(1)$
push	$O(n)$
pop	$O(1)$

On average: O(1)

SUNY Korea

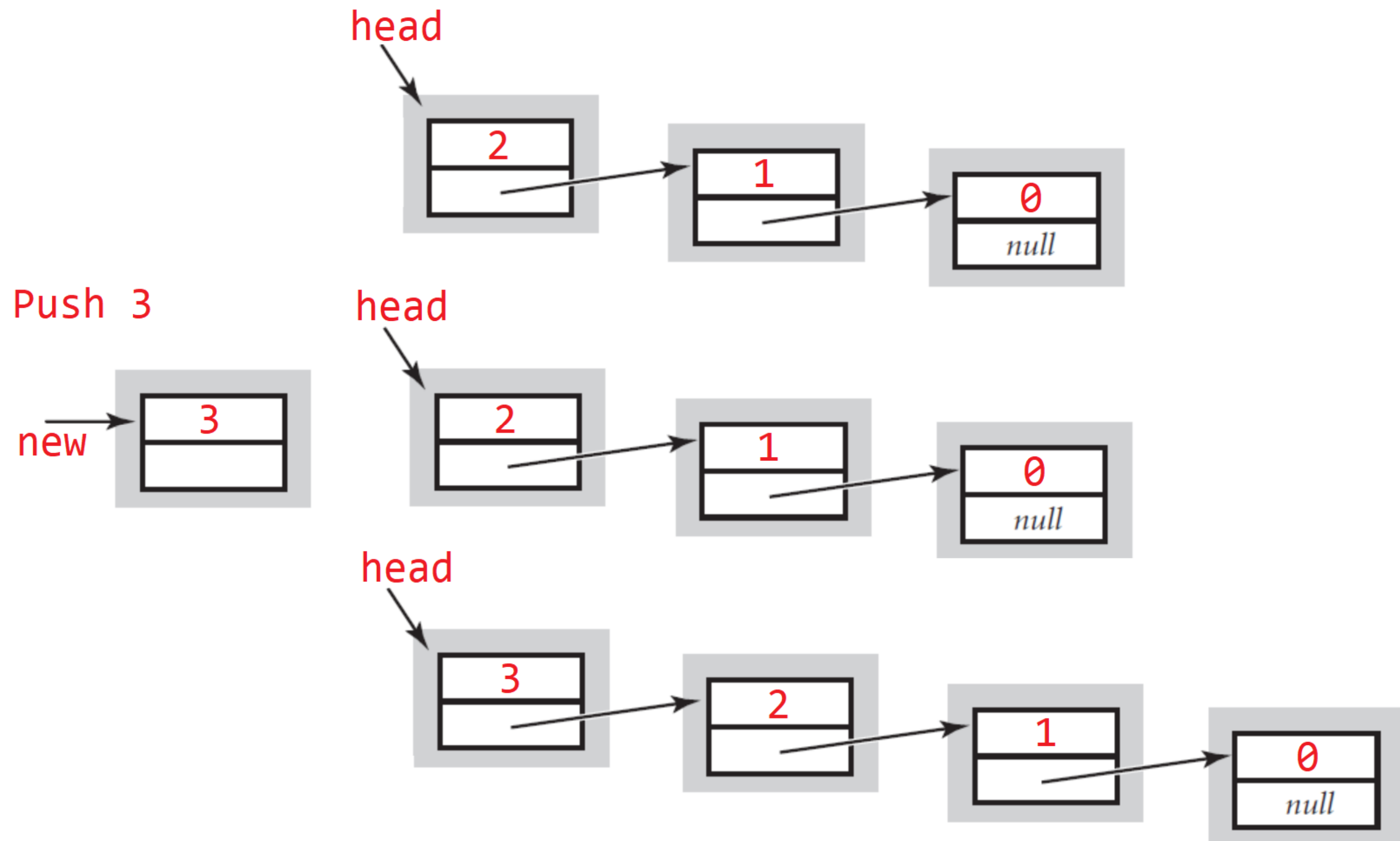
List-Based Stack Implementation

- Direction of stack growth
  - Add at the last element: need to traverse to the end.
    - O(1) to push (singly linked list), O(n) (without tail)
    - O(n) to pop (singly linked list)
  - Add at the first element: no need to move to the end of the list.
    - O(1) to push and pop

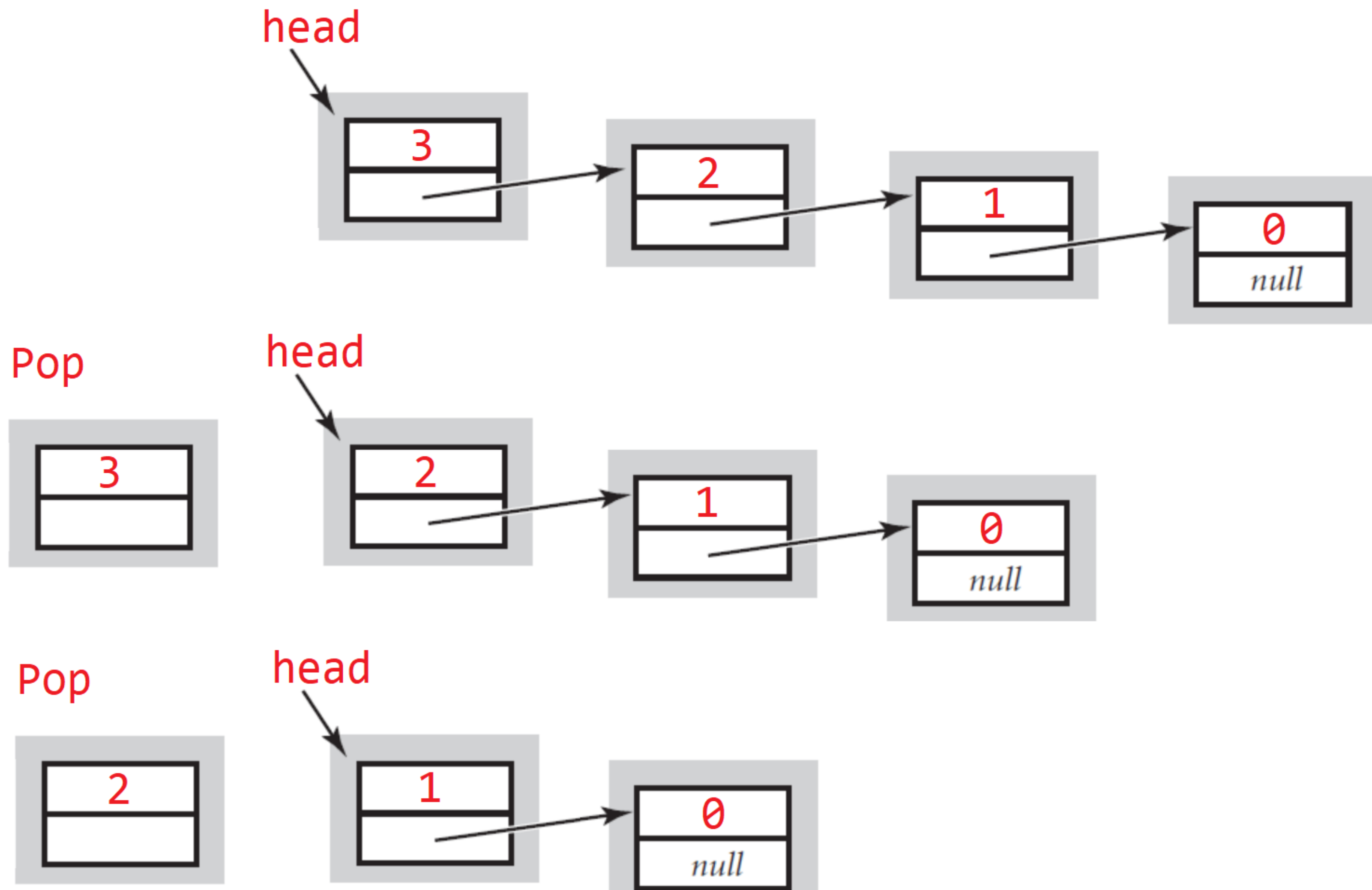


SUNY Korea

Push in List-Based Stack



Pop in List-Based Stack



SUNY Korea

List-Based Stack Implementation

- Adapter design pattern
  - Contain an instance of an existing class
  - Implement methods using the contained instance's methods

SUNY Korea

```java
//Stack using linked list
public class StackByLinkedList<E> implements Stack<E> {
    private List<E> list; //adapter design pattern

    public StackByLinkedList() {
        list = new CircularlyDblLinkedList<E>();
        //list = new DynamicArrayList<E>();
    }

    //interface Stack
    public int size()         { return list.size(); }
    public boolean isEmpty() { return list.isEmpty(); }
    public void push(E e)     { list.add(0, e); }
    public E top()            { return list.get(0); }
    public E pop()            { return list.remove(0); }
}
```

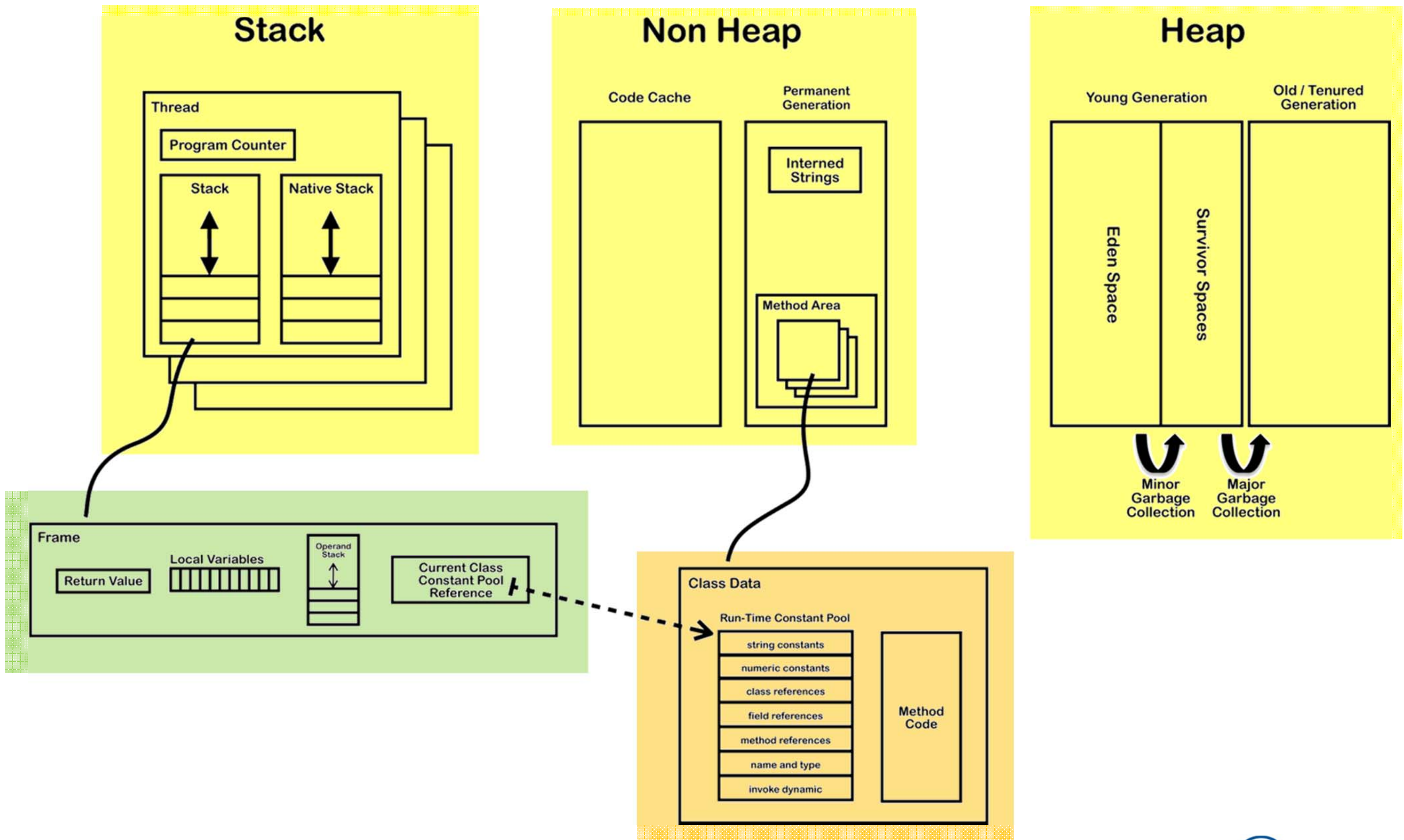
SUNY Korea

Function Calls and Stack

- When a function is invoked
  - A stack frame is pushed on to the stack of the thread
- A stack frame comprises
  - Actual parameters
  - Local variables
  - Temporary variables
  - Return address

SUNY Korea

JVM Architecture



SUNY Korea

Stack Frame

```java
public static String reverse(String str) {
    String substr = null;
    if(str.length() == 0)
        return str;
    else {
        substr = str.substring(1);
        return reverse(substr) +
               str.charAt(0);
    }
}

public static void main(String[] args) {
    String revstr = reverse("ABC");
    System.out.println("rev: " + revstr);
}
```

```
reverse
    str: ""
    substr: null
    return address
```

```
reverse
    str: "C"
    substr: ""
    revserse("") + "C"
    return address
```

```
reverse
    str: "BC"
    substr: "C"
    revserse("C") + "B"
    return address
```

```
reverse
    str: "ABC"
    substr: "BC"
    revserse("BC") + "A"
    return address
```

```
main
    args = {…}
    revstr = null
    "rev: " + revstr
    return address
```

SUNY Korea

Removing Recursion

- Use a stack to remove recursive calls

```
public static String revIter2(String str) {
    Stack<Character> stack = new StackByArray<>();

    for(int i = 0; i < str.length(); i++)
        stack.push(str.charAt(i));

    String rev = "";
    while(!stack.isEmpty())
        rev = rev + stack.pop();

    return rev;
}
```

SUNY Korea

Removing Recursion Example 2

- Reversing a singly linked list

```
public static void main(String[] args) {
    IterableSinglyLinkedList<Integer> list =
                                new IterableSinglyLinkedList<Integer>();
    list.addLast(2);
    list.addLast(3);
    list.addLast(4);
    list.addFirst(1);

    IterableSinglyLinkedList<Integer> rev = reverse1(list.iterator());

    int i = 4;
    for(int j : rev)
        onFalseThrow(j == i--);
}
```

SUNY Korea

Reversing a singly linked list

- Recursive version

```
//recursive version
public static <E>
IterableSinglyLinkedList<E> reverse1(Iterator<E> i) {

    if(!i.hasNext())
        return new IterableSinglyLinkedList<E>();

    else {
        E e = i.next();
        IterableSinglyLinkedList<E> ret = reverse1(i);
        ret.addLast(e);
        return ret;
    }
}
```

  - Big-O?

SUNY Korea

- Removing recursion using a stack

```
//iterative version using stack
public static <E>
IterableSinglyLinkedList<E> reverse2(Iterator<E> i) {
    //make an iterable from iterator using lambda
    Iterable<E> col = () -> i;
    Stack<E> stack = new StackByArray<>();
    IterableSinglyLinkedList<E> ret = new IterableSinglyLinkedList<E>();

    //push elements to stack
    for(E e : col)
        stack.push(e);

    //pop elements and add to the back of the list
    while(!stack.isEmpty())
        ret.addLast(stack.pop());

    return ret;
}
```

  - Big-O?

SUNY Korea

- Removing recursion using 2 stacks

```
//iterative version using 2 stacks (for fun)
public static <E>
IterableSinglyLinkedList<E> reverse3(Iterator<E> i) {
    Stack<E> stack  = new StackByArray<>();
    Stack<E> stack2 = new StackByArray<>();
    IterableSinglyLinkedList<E> ret = new IterableSinglyLinkedList<E>();

    //push elements to stack
    for(E e : (Iterable<E>)() -> i)
        stack.push(e);

    //pop and push to the second stack
    while(!stack.isEmpty())
        stack2.push(stack.pop());

    //pop and add to the front
    while(!stack2.isEmpty())
        ret.addFirst(stack2.pop());

    return ret;
}
```

  - Big-O?

SUNY Korea

- Iterative version

```
//iterative version without a stack
public static <E>
IterableSinglyLinkedList<E> reverse4(Iterator<E> i) {
    IterableSinglyLinkedList<E> ret = new IterableSinglyLinkedList<E>();

    for(E e : (Iterable<E>)() -> i)
        ret.addFirst(e);

    return ret;
}
```

  - Big-O?

SUNY Korea

Matching Parentheses

- Pair of grouping symbols
  - Parentheses (and)
  - Braces { and }
  - Brackets [and]
- Example of expressions
  - Correct: `[ (5 + x) - (y + z) ]`
  - Incorrect: `[ (1 + x} - {y - z] )`

SUNY Korea

Matching Parentheses

- Algorithm
  - On seeing an opening parenthesis, push it onto a stack
  - One seeing a closing parenthesis
    - Check if the stack is not empty
    - Pop from the stack
    - Check if the matching opening parenthesis is popped
  - At the end of the input string
    - Check if the stack is empty

SUNY Korea

Matching Parentheses Example

					[(5 + x) - (y + z)]
[					(5 + x) - (y + z)]
[	(				5 + x) - (y + z)]
[	(				) - (y + z)]
[					(y + z)]
[	(				y + z)]
[	(				)]
[					]

SUNY Korea
The State University of New York

```java
public static boolean isMatched(String expr) {
    Stack<Integer> stack = new StackByArray<Integer>();

    for(char c: expr.toCharArray()) {

        int oi = "({[".indexOf(c); //indexes of opening parenthesis
        if(oi != -1)
            stack.push(oi);

        int ci = ")}]".indexOf(c); //indexes of closing parenthesis
        if(ci != -1) {
            if(stack.isEmpty())    //less opening parenthesis
                return false;

            if(ci != stack.pop())  //open parenthesis doesn't match
                return false;
        }
    }
    return stack.isEmpty();        //more opening parenthesis
}
```

SUNY Korea

Evaluating Arithmetic Expressions

- Example: `1 + 2 * 3 / (4 – 5)`
  - `1 + 2` cannot be evaluated immediately
  - `2 * 3` can only be evaluated after checking that the next operator is `/`
  - `/` cannot be evaluated immediately
  - After evaluating `4 – 5`, `6 / -1` can be evaluated
  - Now, `1 + -6` can be evaluated

SUNY Korea

Evaluating Arithmetic Expressions

- Arithmetic expressions can be evaluated
  - Using an operator stack and an operand stack
  - On seeing a number, push it onto the operand stack
  - On seeing an operator:
    while the top of the operator stack has a higher than or an equal to precedence of the operator
    - pop the operator and two operands;
    - apply the operator;
    - push the result to the operand stack

    Push the operator to the operator stack

SUNY Korea
The State University of New York

Evaluating Arithmetic Expressions

```
Operator Stack   Operand Stack   Expr
[#],             []              1 + 2 * 3 / (4 – 5)$
[#],             [1]               + 2 * 3 / (4 – 5)$
[# +],           [1]                 2 * 3 / (4 – 5)$
[# +],           [1 2]                 * 3 / (4 – 5)$
[# + *],         [1 2]                   3 / (4 – 5)$
[# + *],         [1 2 3]                   / (4 – 5)$
[# +],           [1 6]                     / (4 – 5)$
[# + /],         [1 6]                       (4 – 5)$
[# + / (],       [1 6]                        4 – 5)$
[# + / (],       [1 6 4]                        – 5)$
[# + / ( -],     [1 6 4]                          5)$
[# + / ( -],     [1 6 4 5]                         )$
[# + / (],       [1 6 -1]                          )$
[# + /],         [1 6 -1]                           $
[# +],           [1 -6]                             $
[#],             [-5]                               $
[],              [-5]                               $
```

SUNY Korea

```
public class Expression {
    private static enum Action { Push, Pop }
    private static final Action[][] ACT;

    static {
        // ACT: an action table. based on the input token and the top symbol
        //      of the stack, what action needs to be taken.
        // Push: push the current input token to the stack
        // Pop:  pop the operator from the stack and perform the operation
        // Undefined elements (null) mean an error

        ACT = new Action[128][128]; //[stack top][input token]

        //stack top is '#'
        for(char c : "(+-*/".toCharArray())     ACT['#'][c] = Action.Push;
        for(char c : "$".toCharArray())         ACT['#'][c] = Action.Pop;

        //stack top is '('
        for(char c : "(+-*/".toCharArray())     ACT['('][c] = Action.Push;
        for(char c : ")".toCharArray())         ACT['('][c] = Action.Pop;
```

SUNY Korea

```
    //stack top is '+'
    for(char c : "(*/".toCharArray())        ACT['+'][c] = Action.Push;
    for(char c : "+-)$".toCharArray())       ACT['+'][c] = Action.Pop;

    //stack top is '-'
    for(char c : "(*/".toCharArray())        ACT['-'][c] = Action.Push;
    for(char c : "+-)$".toCharArray())       ACT['-'][c] = Action.Pop;

    //stack top is '*'
    for(char c : "(".toCharArray())          ACT['*'][c] = Action.Push;
    for(char c : "+-*/)$".toCharArray())     ACT['*'][c] = Action.Pop;

    //stack top is '/'
    for(char c : "(".toCharArray())          ACT['/'][c] = Action.Push;
    for(char c : "+-*/)$".toCharArray())     ACT['/'][c] = Action.Pop;
}
```

SUNY Korea

```java
public static double evalExpr(String expr) {
    Scanner scan = new Scanner(expr);

    //number stack and operator stack
    Stack<Double> num    = new StackByArray<Double>();
    Stack<Character> opr = new StackByArray<Character>();
    //Data Abstraction: StackByArray or StackByLinkedList
    // work equally well

    opr.push('#');

    for(String tok : scan) {
        char cur = tok.charAt(0);
        double n1, n2;  //operands

        if(Scanner.isDigit(cur)) {
            num.push(Double.parseDouble(tok));
        }
…
```

SUNY Korea

```java
//if(Scanner.isDigit(cur))…
else {
    Action act;
    loop:
    while((act = ACT[opr.top()][cur]) == Action.Pop) {
        char op = opr.pop();
        switch(op) {
        case '+': n2 = num.pop(); n1 = num.pop(); num.push(n1 + n2); break;
        case '-': n2 = num.pop(); n1 = num.pop(); num.push(n1 - n2); break;
        case '*': n2 = num.pop(); n1 = num.pop(); num.push(n1 * n2); break;
        case '/': n2 = num.pop(); n1 = num.pop(); num.push(n1 / n2); break;
        case '#':
        case '(': break loop; //cur cancels op
        default:
            throw new UnsupportedOperationException("Error: " + tok);
        }
    }

    if(act == Action.Push)
        opr.push(cur);
    else if(act == null)
        throw new IllegalStateException("Syntax error: " + tok);
}
…
```

SUNY Korea

```
    //for(String tok : scan) {
    //…
    } //end for

    //make sure that opr stack is empty and
    //num stack has the result
    if(opr.size() != 0 || num.size() != 1)
        throw new IllegalStateException("Syntax error");

    return num.pop();
}
```

SUNY Korea
The State University of New York

```
import java.util.Iterator;

public class Scanner implements Iterable<String> {
    private char[] buff;

    public static boolean isWhiteSpace(char c) {
        return c == ' ' || c == '\t' || c == '\n' || c == '\r';
    }
    public static boolean isDigit(char c) {
        return '0' <= c && c <= '9';
    }
    public static boolean isAlpha(char c ) {
        return 'a' <= c && c <= 'z' || 'A' <= c && c <= 'Z';
    }
    public Scanner(String str) {
        buff = (str + '$').toCharArray();
    }
    public Iterator<String> iterator() {
        return new TokenIterator();
    }
```

SUNY Korea

```java
public class TokenIterator implements Iterator<String> {
    int pos;
    public TokenIterator()   { pos = 0; }
    public boolean hasNext() { return pos < buff.length; }
    public String next() {
        while(pos < buff.length && isWhiteSpace(buff[pos])) //skip white spaces
            pos++;
        if(pos == buff.length)          //end of string
            return "";
        if(isDigit(buff[pos])) {        //number
            int begin = pos;
            while(isDigit(buff[pos]))
                pos++;
            return new String(buff, begin, pos-begin);
        }
        else if(isAlpha(buff[pos])) {   //identifier
            int begin = pos;
            while(isAlpha(buff[pos]) || isDigit(buff[pos]))
                pos++;
            return new String(buff, begin, pos-begin);
        }
        else {                          //punctuation marks
            pos++;
            return new String(buff, pos-1, 1);
        }
    }
}
```

SUNY Korea

```java
public class App {
    public static void main(String[] args) {
        System.out.print("Enter an expression: ");
        java.util.Scanner sc = new java.util.Scanner(System.in);
        String expr = sc.nextLine();
        sc.close();

        System.out.println("result : " + Expression.evalExpr(expr));
    }
}
```

SUNY Korea

Postfix Expressions

- Infix expressions
  - Operators are in between their operands
  - E.g. 1 + 2 * 3
- Postfix expressions
  - Operators are after their operands
  - E.g. 1 2 3 * +

SUNY Korea

Postfix Expressions

- Evaluating postfix expressions
  - No need to consider the operator precedence
  - On seeing a number push it to a stack
  - On seeing an operator
    - pop two numbers from the stack
    - apply the operator
    - push the result to the stack

SUNY Korea

Evaluating Postfix Expressions

Operand Stack	Postfix expr [infix: 1 + 2 * 3 / (4 – 5)]
[]	1 2 3 * 4 5 – / + $
[1]	2 3 * 4 5 – / + $
[1 2]	3 * 4 5 – / + $
[1 2 3]	* 4 5 – / + $
[1 6]	4 5 – / + $
[1 6 4]	5 – / + $
[1 6 4 5]	– / + $
[1 6 -1]	/ + $
[1 -6]	+ $
[-5]	$

SUNY Korea

Converting Infix Expr to Postfix Expr

- Using an operator stack an infix expr can be converted to a postfix expr
  - On seeing a number, print the number
  - On seeing an operator
    - while the top of the operator stack has a higher than or an equal to precedence of the current operator
      - pop the operator from the stack
      - print it
    - push the operator to the operator stack

SUNY Korea

Infix Expr to Postfix Expr

Operator Stack	Output	Expr
[#],		1 + 2 * 3 / (4 – 5)$
[#],	1	+ 2 * 3 / (4 – 5)$
[# +],	1	2 * 3 / (4 – 5)$
[# +],	1 2	* 3 / (4 – 5)$
[# + *],	1 2	3 / (4 – 5)$
[# + *],	1 2 3	/ (4 – 5)$
[# +],	1 2 3 *	/ (4 – 5)$
[# + /],	1 2 3 *	(4 – 5)$
[# + / (],	1 2 3 *	4 – 5)$
[# + / (],	1 2 3 * 4	– 5)$
[# + / (-],	1 2 3 * 4	5)$
[# + / (-],	1 2 3 * 4 5	)$
[# + / (],	1 2 3 * 4 5 -	)$
[# + /],	1 2 3 * 4 5 -	$
[# +],	1 2 3 * 4 5 - /	$
[#],	1 2 3 * 4 5 - / +	$
[],	1 2 3 * 4 5 - / +	$

SUNY Korea

Appendix: Anonymous Classes

- Instantiating an abstract class or an interface without a concrete class

```java
public class AnonymousClass1 {
    public interface Adder {
        public  int add(int a, int b);  //method without a body
        default int sub(int a, int b) { //default implementation
            return add(a, -b);
        }
    }

    public static void main(String[] args) {
        Adder adder = new Adder() {    // Anonymous class
            public int add(int a, int b) { return a + b; }
        };
        System.out.println("adder.add(3, 2): " + adder.add(3, 2));
        System.out.println("adder.sub(3, 2): " + adder.sub(3, 2));
    }
}
```

SUNY Korea

Appendix: Lambda

- Lambda expression
  - Provides a body for a Single Abstract Method
  - default method: an interface method with a body

```
public class AnonymousClass2 {
    public interface Adder {
        public  int add(int a, int b);  //Single Abstract Method
        default int sub(int a, int b) { //default implementation
            return add(a, -b);
        }
    }

    public static void main(String[] args) {
        Adder adder = (a, b) -> a + b; //Lambda expression
        System.out.println("adder.add(3, 2): " + adder.add(3, 2));
        System.out.println("adder.sub(3, 2): " + adder.sub(3, 2));
    }
}
```

SUNY Korea

Appendix: Currying

- Currying
  - Multi-parameter function ⇒ Nested single-parameter functions

```
public class AnonymousClass3 {
    public interface Fun<P, R> {
        R apply(P a);                        //Single Abstract Method
    }

    public static void main(String[] args) {
        //Curried add
        Fun<Integer, Fun<Integer, Integer>> add = a -> b -> a + b;
        System.out.println("add(2)(3): " + add.apply(2).apply(3));

        Fun<Integer, Integer> inc = add.apply(1);  //inc = b -> 1 + b
        System.out.println("inc(3): " + inc.apply(3));
    }
}
```

SUNY Korea

Programming Assignment 4

- In this assignment, we will add unary minus operator (~) and implement two functions related to postfix expressions
  - ~ operator has higher priority than * and /
    - 1 + ~ 2 * 3 - ~ ~ 4 = -9
    - Postfix of 1 + ~ 2 * 3 - ~ ~ 4 is 1 2 ~ 3 * + 4 ~ ~ -
  - Update ACT table and evalExpr to handle unary minus operator
  - `infixToPostfix`: takes an infix expression and converts it to its postfix form
  - `evalPostfixExpr`: takes a postfix expression and evaluates it
- Due date: 4/14

SUNY Korea

```java
public class Expression {
…
    ACT = new Action[128][128]; //[stack top][input token]
    //TODO: add table entries for unary minus '~' operator
…
    public static double evalExpr(String expr) {
        //TODO: handle unary minus operator
…
    }
    public static String infixToPostfix(String expr) {
        //TODO: implement this method
    }

    public static double evalPostfixExpr(String expr) {
        //TODO: implement this method
    }
}
```

SUNY Korea

```java
public class App {
    public static void main(String[] args) {
        System.out.print("Enter an expression: ");
        java.util.Scanner sc = new java.util.Scanner(System.in);
        String expr = sc.nextLine();
        sc.close();

        System.out.println("result : " + Expression.evalExpr(expr));
        String post = Expression.infixToPostfix(expr);
        System.out.println("postfix: " + post);
        System.out.println("result : " + Expression.evalPostfixExpr(post));

    }
}
```

SUNY Korea
The State University of New York