CS206 Data Structures

Stacks and Queues: Part I

Sung-eui Yoon (윤성의)
Department of Computer Science
KAIST

http://sglab.kaist.ac.kr/~sungeui

KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

Class Objectives

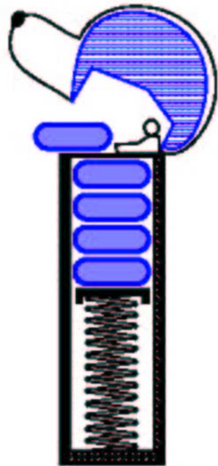
- ☐ Understand the concept of ADT, stack, and generic type
- ☐ Implement the stack using array

Abstract Data Types (ADTs)

- ☐ An abstract data type (ADT) is an abstraction of a data structure
- ☐ An ADT specifies:
  - Data stored
  - Operations on the data
  - Error conditions associated with operations
- ☐ Example: ADT modeling a simple stock trading system
  - The data stored are buy/sell orders
  - The operations supported are
    - order buy(stock, shares, price)
    - order sell(stock, shares, price)
    - void cancel(order)
  - Error conditions:
    - Buy/sell a nonexistent stock
    - Cancel a nonexistent order

The Stack ADT

- The Stack ADT stores arbitrary objects
- Insertions and deletions follow the last-in first-out scheme
- Think of a spring-loaded plate dispenser



- Main stack operations:
  - push(object): inserts an element
  - object pop(): removes and returns the last inserted element
- Auxiliary stack operations:
  - object top(): returns the last inserted element without removing it
  - integer size(): returns the number of elements stored
  - boolean isEmpty(): indicates whether no elements are stored

Stack Interface in Java

- Java interface corresponding to our Stack ADT
- Requires the definition of class EmptyStackException
- Different from the built-in Java class java.util.Stack

```java
public interface Stack<E>
{
    /**
     * Return the number of elements in the stack.
     * @return number of elements in the stack.
     */
    public int size();

    /**
     * Return whether the stack is empty.
     * @return true if the stack is empty, false otherwise.
     */
    public boolean isEmpty();

    /**
     * Inspect the element at the top of the stack.
     * @return top element in the stack.
     * @exception EmptyStackException if the stack is empty.
     */
    public E top() throws EmptyStackException;

    /**
     * Insert an element at the top of the stack.
     * @param element to be inserted.
     */
    public void push(E element);

    /**
     * Remove the top element from the stack.
     * @return element removed.
     * @exception EmptyStackException if the stack is empty.
     */
    public E pop() throws EmptyStackException;
}
```

Exceptions

- In the Stack ADT, operations pop and top cannot be performed if the stack is empty

- Attempting the execution of pop or top on an empty stack throws an EmptyStackException

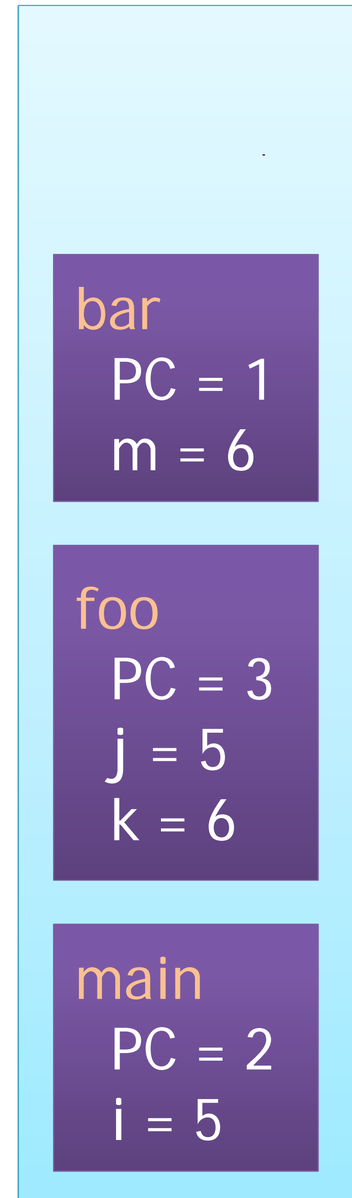
Applications of Stacks

- Direct applications
  - Page-visited history in a Web browser
  - Undo sequence in a text editor
  - Chain of method calls in the Java Virtual Machine
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

Method Stack in the JVM

- The Java Virtual Machine (JVM) keeps track of the chain of active methods with a stack
- When a method is called, the JVM pushes on the stack a frame containing
  - Local variables and return value
  - Program counter, keeping track of the statement being executed
- When a method ends, its frame is popped from the stack and control is passed to the method on top of the stack
- Allows for recursion

```
main() {
  int i = 5;
  foo(i);

}

foo(int j)
{
  int k;
  k = j+1;
  bar(k);
}

bar(int m)
{
  …
}
```

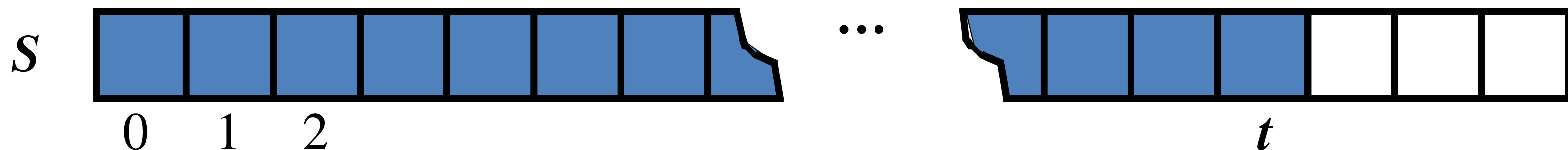


KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

Array-based Stack

- A simple way of implementing the Stack ADT uses an array
- We add elements from left to right
- A variable keeps track of the index of the top element

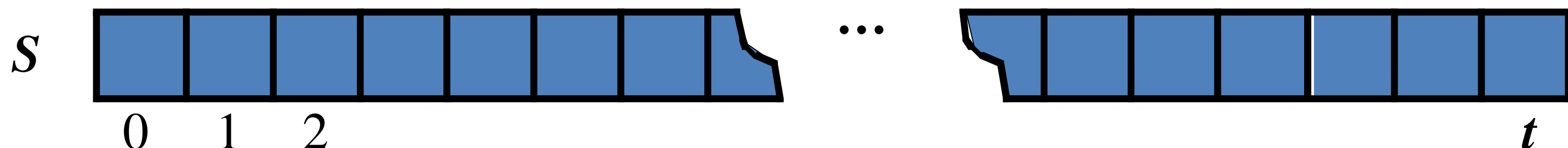
```
Algorithm size()
  return t + 1

Algorithm pop()
  if isEmpty() then
    throw EmptyStackException
  else
    t ← t − 1
    return S[t + 1]
```



Array-based Stack (cont.)

- The array storing the stack elements may become full
- A push operation will then throw a FullStackException
  - Limitation of the array-based implementation
  - Not intrinsic to the Stack ADT

Algorithm $push(o)$
  if $t = S.length - 1$ **then**
    throw $FullStackException$
  else
    $t \leftarrow t + 1$
    $S[t] \leftarrow o$



Performance and Limitations

☐ Performance

- Let n be the number of elements in the stack
- The space used is O(n)
- Each operation runs in time O(1)

☐ Limitations

- The maximum size of the stack must be defined a priori and cannot be changed
- Trying to push a new element into a full stack causes an implementation-specific exception

Array-based Stack in Java

```
/**
 * Implementation of the stack ADT using a fixed-length array.  An
 * exception is thrown if a push operation is attempted when the size
 * of the stack is equal to the length of the array.  This class
 * includes the main methods of the built-in class java.util.Stack.
 */
public class ArrayStack<E> implements Stack<E>
{
    protected int capacity; // The actual capacity of the stack array
    public static final int CAPACITY = 1000; // default array capacity
    protected E S[]; // Generic array used to implement the stack
    protected int top = -1; // index for the top of the stack

    public ArrayStack()
    {
        this(CAPACITY); // default capacity
    }

    public ArrayStack(int cap)
    {
        capacity = cap;
        S = (E[]) new Object[capacity]; // compiler may give warning, but this is ok
    }

    public int size()
    {
        return (top + 1);
    }

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

Array-based Stack in Java (cont.)

```
public void push(E element) throws FullStackException
{
    if (size() == capacity)
        throw new FullStackException("Stack is full.");
    S[++top] = element;
}

public E top() throws EmptyStackException
{
    if (isEmpty())
        throw new EmptyStackException("Stack is empty.");
    return S[top];
}

public E pop() throws EmptyStackException
{
    E element;
    if (isEmpty())
        throw new EmptyStackException("Stack is empty.");
    element = S[top];
    S[top--] = null; // dereference S[top] for garbage collection.
    return element;
}
public String toString()
{
    String s;
    s = "[";
    if (size() > 0)
        s += S[0];
    if (size() > 1)
        for (int i = 1; i <= size() - 1; i++)
        {
            s += ", " + S[i];
        }
    return s + "]";
}

//  Prints status information about a recent operation and the stack.
public void status(String op, Object element)
{
    System.out.print("------> " + op); // print this operation
    System.out.println(", returns " + element); // what was returned
    System.out.print("result: size = " + size() + ", isEmpty = "
            + isEmpty());
    System.out.println(", stack: " + this); // contents of the stack
}
```

Testing Array-based Stack Using Main

```
public static void main(String[] args)
    {
        Object o;
        ArrayStack<Integer> A = new ArrayStack<Integer>();
        A.status("new ArrayStack<Integer> A", null);
        A.push(7);
        A.status("A.push(7)", null);
        o = A.pop();
        A.status("A.pop()", o);
        A.push(9);
        A.status("A.push(9)", null);
        o = A.pop();
        A.status("A.pop()", o);
        ArrayStack<String> B = new ArrayStack<String>();
        B.status("new ArrayStack<String> B", null);
        B.push("Bob");
        B.status("B.push(\"Bob\")", null);
        B.push("Alice");
        B.status("B.push(\"Alice\")", null);
        o = B.pop();
        B.status("B.pop()", o);
        B.push("Eve");
        B.status("B.push(\"Eve\")", null);
    }
```

Testing Array-based Stack Using JUnit

```
public class ArrayStackTest extends TestCase
{
    public void testArryStack()
    {
        Object o;
        ArrayStack<Integer> A = new ArrayStack<Integer>();
        assertTrue(A.isEmpty());
        assertTrue(A.size() == 0);
        A.push(7);
        assertTrue(!A.isEmpty());
        assertTrue(A.size() == 1);
        assertTrue(A.top() == 7);
        o = A.pop();
        assertTrue(A.isEmpty());
        assertTrue(A.size() == 0);
        assertTrue(o.equals(7));
        A.push(9);
        assertTrue(!A.isEmpty());
        assertTrue(A.top() == 9);
        o = A.pop();
        assertTrue(o.equals(9));

        ArrayStack<String> B = new ArrayStack<String>();
        B.push("Bob");
        assertTrue(B.top().equals("Bob"));
        B.push("Alice");
        assertTrue(B.top().equals("Alice"));
        assertTrue(B.size() == 2);
        o = B.pop();
        assertTrue(o.equals("Alice"));
        B.push("Eve");
        assertTrue(B.top().equals("Eve"));
        assertTrue(B.size() == 2);
    }
}
```

Generic Types

- A Generic type (Java 5.0) is a type not defined at compilation time, but becomes fully specified at run time
  - A generic type takes formal type parameters
  - We use actual type parameters to make it concrete

```
public class Pair<K, V>
{
    K key;
    V value;

    public void set(K k, V v)
    {         key = k; value = v;     }

    public K getKey()
    {         return key;     }

    public V getValue()
    {         return value;     }

    public String toString()
    {         return "[" + getKey() + ", " + getValue() + "]";     }

    public static void main(String[] args)
    {
        Pair<String, Integer> pair1 = new Pair<String, Integer>();
        pair1.set(new String("height"), new Integer(36));
        System.out.println(pair1);
        Pair<Student, Double> pair2 = new Pair<Student, Double>();
        pair2.set(new Student("A5976", "Sue", 19), new Double(9.5));
        System.out.println(pair2);
    }
}
```

More Examples of Generic Types

- ☐ Use `extends` to limit the actual type parameter

```
public class PersonPairDirectoryGeneric<P extends Person>
{
    // ... instance variables would go here ...
    public PersonPairDirectoryGeneric()
    { /* default constructor goes here */
    }
    public void insert(P person, P other)
    { /* insert code goes here */
    }
    public P findOther(P person)
    {
        return null;
    } // stub for find
    public void remove(P person, P other)
    { /* remove code goes here */
    }
}
```

- ☐ Define generic versions of methods

```
 public static <K extends Comparable,V,L,W> int
         comparePairs(Pair<K,V> p, Pair<L,W> q) {
    return p.getKey().compareTo(q.getKey());  // p's key implements compareTo
  }
```

KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

Class Objectives were:

- ☐ Understand the concept of ADT, stack, and generic type
- ☐ Implement the stack using array

PA2

- Find a path from a maze
  - Use the stack that is based on the linked list

Next Time

- Queues

- HW:
  - Go over the next lecture slides before the class
  - Just 10 min ~ 20 min should be okay

Any Questions?

- ☐ Come up with one question on what we have discussed in the class and submit at the end of the class
  - 1 for typical questions
  - 2 for questions with thoughts or that surprised me
- ☐ Write questions at least 4 times
  - Write a question about one out of four classes
- ☐ You can type at KLMS