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 (§4.2)

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

public interface Stack {
    public int size();
    public boolean isEmpty();
    public Object top() throws EmptyStackException;
    public void push(Object o);
    public Object pop() throws EmptyStackException;
}
Exceptions

- Attempting the execution of an operation of ADT may sometimes cause an error condition, called an exception.
- Exceptions are said to be “thrown” by an operation that cannot be executed.

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) {
    ...
}
```

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]
```

```
S
0 1 2 ... t
```
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(a)`

```
if t = S.length - 1 then
    throw FullStackException
else
    t ← t + 1
    S[t] ← a
```

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

```java
public class ArrayStack implements Stack {
    // holds the stack elements
    private Object S[];
    // index to top element
    private int top = -1;
    // constructor
    public ArrayStack(int capacity) {
        S = new Object[capacity];
    }
    public Object pop() throws EmptyStackException {
        if (isEmpty())
            throw new EmptyStackException;
        Object temp = S[top];
        S[top] = null;
        top = top - 1;
        return temp;
    }
}
```

Parentheses Matching

Each “(”, “{”, or “[” must be paired with a matching “)”, “}”, or “]”

- correct: ( ( ( ) ( [ ] ) ) )
- correct: ( ( ) ( ( ) ) [ ] )
- incorrect: ) ( ( ) ( [ ] ) )
- incorrect: ( [ ] )