Stacks and Queues
(Chapter 5)

COMP53
Oct 15, 2007
Note: Building GUls with Swing

• We will begin looking at Graphical User Interface implementations using Swing over the next few weeks.
  – this information is not in the textbook
  – there are many tutorials available on the web
  – a good place to start is:
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

• ADTs are similar to pseudo code algorithms
  – language independent
  – used for specifying reusable, generic concepts
Stacks

• A stack stores objects in the order they are added
• Objects can only be removed from a stack in reverse of the order they were added
  – LIFO = last in – first out
• Real world analogies:
  – stack of plates
  – PEZ dispenser
• Despite their simplicity, stacks are incredibly useful in computing
The Stack ADT

- A **Stack** stores arbitrary objects
- Stack operations:
  - **push**(object): inserts an element
  - **pop()**: removes the last inserted element
  - **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
Queues

• A queue stores objects in the order they are added
• Objects can only be removed from a stack in the order they were added
  – FIFO = first in – first out
• Real world analogies:
  – check-out lines
  – single lane roads (no passing!)
• Like stacks, queue have a variety of uses in computing
The Queue ADT

- A **Queue** stores arbitrary objects
- **Queue operations:**
  - `enqueue(object)`: inserts an element at the end of the queue
  - `dequeue()`: removes the element at the front of the queue
  - `object front()`: returns the element at the front (without removing it)
  - `integer size()`: returns the number of elements stored
  - `boolean isEmpty()`: indicates whether no elements are stored
Variations

• The textbook suggests that Stack.pop() and Queue.dequeue() should also return the element that is being removed from the structure.

• Example implements in lecture will assume that these functions simply modify the structure without returning a value.
Exception Handling

- Stacks and queues may encounter the same problems
  - attempt to access an element in an empty stack/queue
    - stack: pop, top – queue: dequeue, front
  - attempt to add an element to a full stack/queue
    - stack: push – queue: enqueue
    - (implementation dependent, generally not considered for ADTs)
- When these problems occur: exceptions are throw
Implementation

• Java has built-in stacks and queues
  – we’ll look at these later
• Both stacks and queue can be constructed by adapting various other data structures
  – arrays and singly linked lists are common
Stack Interface in Java

```java
public interface Stack {
    public int size();
    public boolean isEmpty();
    public Object top() throws EmptyStackException;
    public void push(Object o);
    public void pop() throws EmptyStackException;
}
```
public interface Queue {
    public int size();
    public boolean isEmpty();
    public Object front() throws EmptyQueueException;
    public void enqueue(Object o);
    public void dequeue() throws EmptyQueueException;
}
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
  – stack based calculators (RPN)

• Indirect applications
  – Auxiliary data structure for algorithms
  – Component of other data structures
Applications of Queues

• Direct applications
  – Waiting lists, bureaucracy
  – Access to shared resources (example: printers)
  – Communication paths (example: network packets)

• Indirect applications
  – Auxiliary data structure for algorithms
  – Component of other data structures
Advanced ADT Specification

• We can also specify the behavior of an ADT declaratively (non-procedurally)
  – this can be important for proving properties of algorithms and programs
  – you might see this in books/courses on programming language theory
  – following examples are not required knowledge for this course
Stack ADT with Axioms

type stack (element) imports boolean

operations:
createstk: stack
push: stack \times element \rightarrow stack
pop: stack \rightarrow stack
top: stack \rightarrow element
emptystk: stack \rightarrow boolean

variables: s: stack; x: element

axioms:
emptystk (createsstk) = true
emptystk (push (s, x)) = false
top (createsstk) = error
top (push (s, x)) = x
pop (createsstk) = error
pop (push (s, x)) = s

From “Programming Languages: Principles and Practice, 2nd ed.”
By Kenneth C. Louden
Queue ADT with Axioms

```plaintext
type queue(element) imports boolean
operations:
    createq: queue
    enqueue: queue × element → queue
    dequeue: queue → queue
    frontq: queue → element
    emptyq: queue → boolean
variables: q: queue; x: element
axioms:
    emptyq(createq) = true
    emptyq(enqueue(q,x)) = false
    frontq(createq) = error
    frontq(enqueue(q,x)) = if emptyq(q) then x else frontq(q)
    dequeue(createq) = error
    dequeue(enqueue(q,x)) = if emptyq(q) then q else
    enqueue(dequeue(q),x)
```

From “Programming Languages: Principles and Practice, 2nd ed.”
By Kenneth C. Louden