Lecture 2: Abstract Data Types

CS2504/CS4092– Algorithms and Linear Data Structures

Dr. Kieran T. Herley

Department of Computer Science
University College Cork

2013/14

Summary

Data types and abstract data types. ADT Stack: definition, operation and example. Last-in first-out behaviour.

Data Types

- A data type is characterized by
  - a set of *values*
  - a set of *operations* that may be applied to those values
  - a representation of those values and an implementation of the operations
- Data type embodies both
  - abstract idea
  - concrete realization of that idea

Some Data Type Examples

Data Type	Values	Notes	Ops.
boolean	true, false	1 byte	|| && !
char	Unicode characters (e.g. 'U', '5', '$')	2 bytes	as for int
int	neg., zero and pos. integer numbers	32-bit twos-complement	$+ - * /$ $< == <=$
float	neg., zero, pos. floating-point nums	IEEE 32-bit floating point	as above
String	sequences of characters (e.g. "apple")		+ length charAt substring compareTo
ArrayList	grow-able array of objects		size get add
Card	playing card denominations	as discussed	getSuit, getValue

Data Type String

Data Type String embodies the idea of the series of characters

Values Any sequence of zero or more characters

Operations Data type supports operations to

- determine length of string
- probe individual characters
- extract substrings (contiguous subsequences. of characters)
- compare two strings
- (and many others)

Representation and Implementation internal to Java–no need to worry about this

Data Type String

Data Type String embodies the idea of the series of characters

Values Any sequence of zero or more characters

Operations Data type supports operations to

- determine length of string
- probe individual characters
- extract substrings (contiguous subsequences. of characters)
- compare two strings
- (and many others)

Representation and Implementation internal to Java–no need to worry about this

NB

NB

Can work with Strings without knowing precisely *how* they work (internally) as long as we know *what* they do

Data Type Card

Data Type Card embodies the idea of a playing card .

Values Any (legal) suit plus value (Ace to King) pairing

Operations Data type supports ops. to

- determine suit or value
- print the suit-value
- (plus others)

Representation and Implementation

- See previous lecture or code
- Can use Card with knowing internal code details

Abstract Data Types

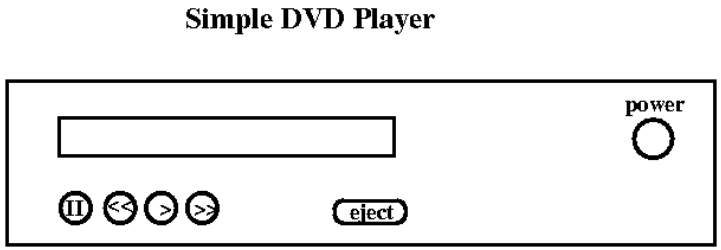
(Concrete) Data Type

$$\underbrace{\text{Values/Operations}}_{\text{abstract concept}} + \underbrace{\text{Representations/Implementation}}_{\text{concrete realization}}$$

Abstract Data Type

- Abstract concept embodied in type *i.e.* it's values and operations
- Data type divorced from specifics of representation/implementation
- Useful notion for software design:
  - Can use types just knowing *what* it does not *how*
  - Design focuses on *concepts* required for solution
  - Nitty-gritty issues concerning implementation of these concepts can be deferred or delegated or ignored

Abstraction



Definition

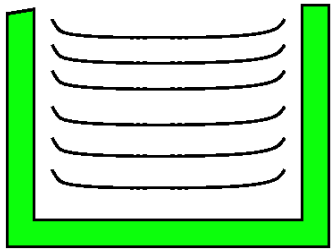
Abstraction Idealized model that provides a simplified conceptual model that cloaks a more complex underlying reality

DVD Player

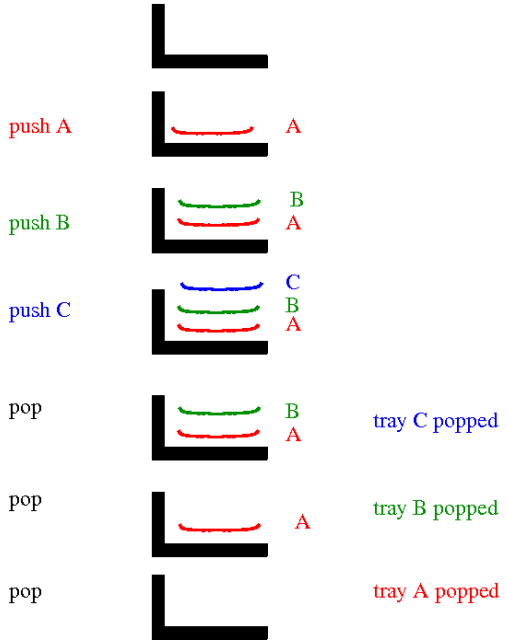
- A box that plays movies; supports following behaviour:
  - play
  - stop/pause
  - fast forward/back
- Description focusses on *what* player does not *how* it does it
  - Ingores complex technological underpinning
  - But sufficient to allow us to use it

A Useful (Abstract) Concept: The Stack

Container



Cafeteria tray holder, trays accessible only at top



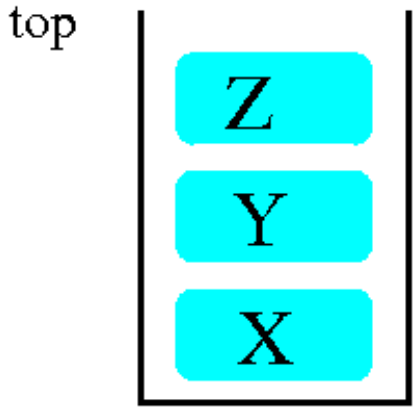
Operations

- Add a tray to the top (**push**)
- Remove tray from top (**pop**)

Observation Last-in First-out(LIFO) behaviour.

ADT Stack Concept

Stack A Stack is an abstraction of this tray-dispenser idea pared down to its essential features for CS use.



"Containerness" A stack embodies the concept of a container– an object intended to hold/contain other objects (integers, strings, *etc.*).

Operations Stack accessible only at its top and supports operations

- add item (push)
- remove item (pop)

Stack Applications in CS

Browser History Lists Web browsers generally maintain a stack-like "history" of recently visited pages. The URL of each newly visited page is "pushed"; clicking the back button causes a "pop".

Java Virtual Machine The JVM relies heavily on a stack to manage method calls and returns. Each method call causes a push, each return a pop. Space for local variables typically allocated on stack. Facilitates recursion.

Compilers Many compiler parsing algorithms are stack-based

ADT Stack

A stack is an abstract data type on which the following operations are defined. The notation EltType refers to the type of the items. [1]

push(o): Insert item o at top of stack. *Input: EltType*; *Output: None*.

pop(): Remove and return top item on stack; illegal if stack is empty. *Input: None*; *Output: EltType*.

size(): Return no. of items in the stack. *Input: None*; *Output: int*.

isEmpty(): Return boolean indicating if stack is empty. *Input: None*; *Output: boolean*.

top(): Return, but do not remove, top stack item; illegal if stack is empty. *Input: None*; *Output: EltType*.

[1] Note abstract concept specified in manner unspecific to type of items contained within it.

ADT Stack

ADT Specification

pop(): Remove and return top item on stack; illegal if stack is empty. *Input: None*; *Output: EltType*.

Spec. of each operation describes (i) name of, (ii) description of, (iii) arguments taken by and (iv) results retured by operation.

Notes

- For now think of stack concept as a "family" of related data types with same operations that differ on types of things held
- StackOfInt "holds" ints, StackOfString "holds" Strings *etc.*
- Above "EltType" is a placeholder for whatever type of object populatres the container *e.g.* "int" for StackIfInt *etc.*

Illustration

Operation	Stack	Output
push(A)	A	

Illustration

Operation	Stack	Output
push(A)	A	
push(B)	A B	

Illustration

Operation	Stack	Output
push(A)	A	
push(B)	A B	
size()	A B	2

Illustration

Operation	Stack	Output
push(A)	A	
push(B)	A B	
size()	A B	2
pop()	A	B

Illustration

Operation	Stack	Output
push(A)	A	
push(B)	A B	
size()	A B	2
pop()	A	B
push(C)	A C	

Illustration

Operation	Stack	Output
push(A)	A	
push(B)	A B	
size()	A B	2
pop()	A	B
push(C)	A C	
pop()	A	C

Illustration

Operation	Stack	Output
push(A)	A	
push(B)	A B	
size()	A B	2
pop()	A	B
push(C)	A C	
pop()	A	C
pop()		A

Illustration

Operation	Stack	Output
push(A)	A	
push(B)	A B	
size()	A B	2
pop()	A	B
push(C)	A C	
pop()	A	C
pop()		A
isEmpty()		true

Illustration

Operation	Stack	Output
push(A)	A	
push(B)	A B	
size()	A B	2
pop()	A	B
push(C)	A C	
pop()	A	C
pop()		A
isEmpty()		true
push(D)	D	

Illustration

Operation	Stack	Output
push(A)	A	
push(B)	A B	
size()	A B	2
pop()	A	B
push(C)	A C	
pop()	A	C
pop()		A
isEmpty()		true
push(D)	D	
isEmpty()	D	false

Illustration

Operation	Stack	Output
push(A)	A	
push(B)	A B	
size()	A B	2
pop()	A	B
push(C)	A C	
pop()	A	C
pop()		A
isEmpty()		true
push(D)	D	
isEmpty()	D	false
push(E)	D E	

Illustration

Operation	Stack	Output
push(A)	A	
push(B)	A B	
size()	A B	2
pop()	A	B
push(C)	A C	
pop()	A	C
pop()		A
isEmpty()		true
push(D)	D	
isEmpty()	D	false
push(E)	D E	
push(F)	D E F	

Illustration

Operation	Stack	Output
push(A)	A	
push(B)	A B	
size()	A B	2
pop()	A	B
push(C)	A C	
pop()	A	C
pop()		A
isEmpty()		true
push(D)	D	
isEmpty()	D	false
push(E)	D E	
push(F)	D E F	
pop()	D E	F

Illustration

Operation	Stack	Output
push(A)	A	
push(B)	A B	
size()	A B	2
pop()	A	B
push(C)	A C	
pop()	A	C
pop()		A
isEmpty()		true
push(D)	D	
isEmpty()	D	false
push(E)	D E	
push(F)	D E F	
pop()	D E	F
size()	D E	2

Note: Top of stack on right.

Stacks in Java

- Class StackOfInt.java provides Java implementation

Stacks in Java

- Class StackOfInt.java provides Java implementation
- Usage:

Declaring stack object

```
StackOfInt s;
```

Stacks in Java

- Class StackOfInt.java provides Java implementation
- Usage:

Declaring stack object

```
StackOfInt s;
```

Creating stack object

```
s = new StackOfInt();
```

Stacks in Java

- Class StackOfInt.java provides Java implementation
- Usage:

Declaring stack object

```
StackOfInt s;
```

Creating stack object

```
s = new StackOfInt();
```

Using stack object

```
s.push(17);
int num = s.pop();
```

Stacks in Java

- Class StackOfInt.java provides Java implementation
- Usage:

  Declaring stack object

```
StackOfInt s;
```

  Creating stack object

```
s = new StackOfInt();
```

  Using stack object

```
s.push(17);
int num = s.pop();
```

- Notes:
  - StackOfInt will be superceded shortly by more general class
  - Java's java.util.Stack similar to but slightly more complex to use

Stacks and Java

```
public class StackExercise
{  public static void main(String args [])
   {  StackOfInt stk = new StackOfInt(); int num;
      for (num = 1; num <= 6; num++)
      {  System.out. println ("Pushing␣"+num);
         stk.push(num);
      }
      while (!stk.isEmpty())
      {  num = stk.pop();
         System.out. println ("Popped␣"+num);
      }
   }
}
```

Stacks and Java

```
public class StackExercise
{  public static void main(String args [])
   {  StackOfInt stk = new StackOfInt(); int num;
      for (num = 1; num <= 6; num++)
      {  System.out. println ("Pushing␣"+num);
         stk.push(num);
      }
      while (!stk.isEmpty())
      {  num = stk.pop();
         System.out. println ("Popped␣"+num);
      }
   }
}
```

Exercise: What does his do?