Collections & Data Structures

Boaz Kantor
Introduction to Computer Science
IDC Herzliya

ABSTRACTION VS. IMPLEMENTATION

Abstraction vs. Implementation

- Data structures provide both abstraction and implementation.
  - Abstraction is the functionality
  - Implementation is the internal collection structure
- Abstraction may provide FIFO, LIFO or other mechanisms.
- Implementation may be an array, a linked list of various types, trees and graphs.

•Implementation

ARRAY

- Implemented by a Java array.
- Advantage:
  - Best performance for direct access.
- Disadvantage:
  - Only pre-determined size.

LINKED LIST

- Implemented as a collection of elements.
- Each element pointing to the next.
- A List class pointing to 'head'.
- Advantage:
  - Dynamic memory allocation.
- Disadvantage:
  - No direct access.
- Variations: doubly-linked list, keeping a 'tail' reference, etc.

© 2010 Brad Kahlbar, IDC Interactive

•Abstractions

QUEUE

- Provides FIFO abstraction.
- Can be implemented using a Linked List or a Java array.
- Provides this functionality:
  - Enqueue
  - Dequeue
  - IsEmpty
  - Peek
  - Clear

STACK

- Provides LIFO abstraction.
- Can be implemented using a Linked List or a Java array.
- Provides this functionality:
  - Push
  - Pop
  - IsEmpty
  - Peek (or Top)
  - Clear

© 2010 Brad Kahlbar, IDC Interactive

GENERICS

© 2010 Brad Kahlbar, IDC Interactive

What is generics?

- Generics is a fundamental mechanism in Java.
- It allows writing data types with an additional internal data type.
- The internal data type can be chosen by the user (other programmer).
- For example:
  - `ArrayList` is a list of references to objects of type Object.
  - `ArrayList<Character>` is a list of references to objects of type Character.
  - `ArrayList<Clock>` is a list of clocks.
- Writing generic collections is out of the scope of this course.

How to identify Generic classes

- Some classes come with angle brackets in their class definition API:

java.util
Class Stack<E>

java.lang.Object
  └ java.util.AbstractCollection<E>
    └ java.util.AbstractList<E>
      └ java.util.Vector<E>
        └ java.util.Stack<E>

All Implemented Interfaces:
Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

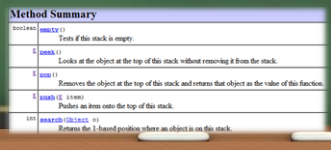
```
public class Stack<E>
extends Vector<E>
```

How to read generic API

- When we instantiate an object of a generic type, we have to provide a class name instead of that 'E':

```
Stack<Character> characterStack = new Stack<Character>();
```

- The API makes further use of 'E'. When we read 'E' in the API, we should replace it with the type we provided, in this case 'Character':



Method Summary

boolean	empty() Tests if this stack is empty.
E	peek() Looks at the object at the top of this stack without removing it from the stack.
E	pop() Removes the object at the top of this stack and returns that object as the value of this function.
E	push(E item) Pushes an item onto the top of this stack.
int	search(Object o) Returns the 1-based position where an object is on this stack.

Using a generic type

- We can now refer to 'E' as 'Character' whenever it's mentioned in the API, **without the need of casting**.

```
Character c = characterStack.pop();
```

pop

`public E pop()`

Removes the object at the top of this stack and returns that object as the value of this function.

Returns:
The object at the top of this stack (the last item of the `Vector` object).

Throws:
`EmptyStackException` - if this stack is empty.

© 2010 Boaz Kantor, IDC Herzliya

Reversing the words in each line of a text file using a Stack<E>

EXAMPLE

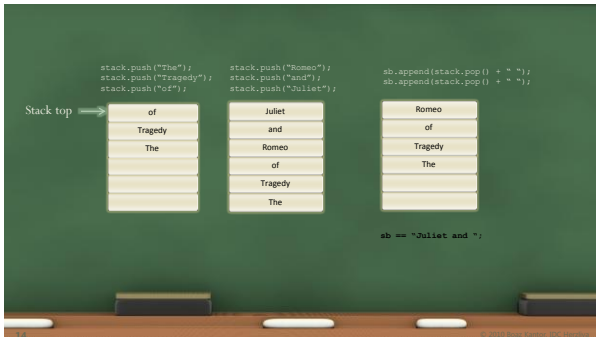
© 2010 Boaz Kantor, IDC Herzliya

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.util.Stack;

public class Example {

        public static void main(String[] args) {

                // local variables
                String filename = "Romeo.txt";
                StringBuilder sb = new StringBuilder();
                Stack<String> stack = new Stack<String>();
                Scanner lineScanner = null;

                // open the file for reading
                try {
                        lineScanner = new Scanner(new File(filename));
                } catch (FileNotFoundException e) {
                        System.out.println("Can't find the file " + filename);
                        System.exit(0);
                }
```


© 2010 Boaz Kantor, IDC Herzliya

```
                // iterate the file line by line
                while (lineScanner.hasNextLine()) {

                        // read word by word
                        String line = lineScanner.nextLine();
                        Scanner wordScanner = new Scanner(line);

                        // push all the words to a stack
                        while (wordScanner.hasNext()) {
                                stack.push(wordScanner.next());
                        }

                        // pop all the words and build the result line
                        while (stack.size() > 0) {
                                sb.append(stack.pop()).append(" ");
                        }

                        sb.append("\n");
                }

                // we're done reading the entire file, print the result
                System.out.println(sb);
        }
}
```


© 2010 Boaz Kantor IDC Herzliya



© 2010 Boaz Kantor IDC Herzliya