Logistics

At Home

- Read Weiss Ch 15: ArrayList implementation
- Read Weiss Ch 16: Stacks and Queues implementation
- HW 1: Due Sunday by 11:59pm
- Questions on HW 1?

Goals Today

- ArrayList Implementation
- ArrayList Complexity
- Implementation of Stacks and Queues
Coded a version of ArrayList together quickly
Simple version: MyArrayList.java in code distrib
Also included java.util.ArrayList from Java 1.7 source
May also want to look at Weiss’s version in
textbook-source/weiss/util/ArrayList.java
Certain things proved impossible like
this.data = new T[10];
Limits of Types

Unfortunately, java type system has some limits.

new T[10] Not Allowed

```java
public class MyArrayList<T> {
    private T[] data;
    public MyArrayList(){
        this.data = new T[10]; // Grrrr
    }
    public T get(int i){
        this.rangeCheck(i);
        return this.data[i];
    }
}
```

Instead: Object[] + Caste

```java
public class MyArrayList<T> {
    private Object[] data[];
    public MyArrayList(){
        this.data = new Object[10];
    }
    public T get(int i){
        this.rangeCheck(i);
        return (T) this.data[i];
    }
}
```
Unsafe Operations

lila [w01-2-1-code]%% javac MyArrayList.java
Note: MyArrayList.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

lila [w01-2-1-code]%% javac -Xlint:unchecked MyArrayList.java
MyArrayList.java:77: warning: [unchecked] unchecked cast
found    : java.lang.Object
required: T
        return (T) this.data[i];

1 warning
Unsafe Operations

Offending code is

```java
// The internal store for
private Object [] data;

public T get(int i){
    this.rangeCheck(i);
    return (T) this.data[i];
}
```

It is unsafe (why?). But so is fire.
Tell the compiler to shut up

```java
@SuppressWarnings("unchecked")  // I know what I’m doing
public T get(int i){
    this.rangeCheck(i);
    return (T) this.data[i];
}
```
HW and Unsafe Operations

- Proper use of generics creates good compile-time type checking
- **Rarely** is casting required; ArrayList implementation is one such exception
- **HW1** is NOT such a case
  - Should not need to caste anything
  - Should not need to use @SuppressWarnings
  - Doing either may result in penalties
### ArrayList Complexities

#### ArrayList of size N

- Time/Space Complexities
- Worst-case or Average/Amortized

<table>
<thead>
<tr>
<th>Operation</th>
<th>Method</th>
<th>Worst Time</th>
<th>Average Time</th>
<th>Worst Space</th>
<th>Average Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size()</td>
<td>al.size()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get(i)</td>
<td>al.get(i)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set(i,x)</td>
<td>al.set(i,x)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add(x)</td>
<td>al.add(x)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert(i,x)</td>
<td>al.add(i,x)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove(i)</td>
<td>al.remove(i)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- What is the space complexity of an ArrayList with $N$ elements?
Expanding with Magic Numbers

- Size increase when expansion is required is interesting
- Can’t be constant: increase size by 1, or 2, or 10 will not give good complexity
- Standard Java ArrayList increases to $3/2 \times \text{oldSize} + 1$
- Chosen based on engineering experience rather than theory, can use bit shifts to compute it fast
- Default ArrayList size is 10
- **Magic Numbers**: 3/2 and 10, magic because there is no good reason for them
Average/Amortized Complexity

- Worst case complexity for `arrayList.add(x)` is $O(N)$ when expansion is required
- **But** expansion happens rarely if size increase by 150% during expansion
- Over many add operations, the average `add(x)` takes $O(1)$ time complexity
- **Amortized Analysis**: sort of like average case (definition is close enough for this class)
Stacks

A data structure, supports a few operations

- \( T \ s.\text{getTop}() \): return whatever is on top
- \( s.\text{push}(T \ x) \): put \( x \) on top
- \( \text{void } s.\text{pop}() \): remove whatever is on top
- \( \text{boolean } s.\text{isEmpty}() \): true when nothing is in it, false o/w

Stacks are a LIFO: Last In First Out

Questions

- Examples of stacks?
- How would you implement a stack using arrays?
Array Based Implementation

Just use ArrayList to make an AStack

class AStack<T>{
    private ArrayList<T> stuff;
    public AStack(); // Constructor
    public void push(T x); // Like add(x)
    public void pop(); // Like remove(size()-1)
    public T getTop(); // Like get(size()-1)
    public boolean isEmpty(); // Like size()==0
}

See: weiss/nonstandard/ArrayStack.java

Work It

- Stacks: more or less functionality than ArrayList?
- Worst and Amortized Complexity of stack operations?
- Can we do better?
Nodes

To get worst-case $O(1)$ push, need to change the underlying representation of the stack implementation.

- Simplest unit to support linked data structure
- ListNode in text
- Cons box in Lisp
- Tracks a piece of data and the next node in a sequence
- String them together by setting next

Node Class

class Node<T>{
    public T data;
    public Node<T> next;

    public Node(T d, Node<T> n ){
        this.data = d;
        this.next = n;
    }
}


Linked Nodes

Can string Nodes together by manipulating the next field

class Node<T>{
    public T data;
    public Node<T> next;

    public Node(T d, Node<T> n)
    {
        this.data = d;
        this.next = n;
    }
}

Node<Integer> n3 = new Node<T>(10,null);
Node<Integer> n2 = new Node<T>(22, n3);
Node<Integer> n1 = new Node<T>(5, n2);
Node<Integer> head = n1;

Node

Data (???)
Next (node)

Linked Nodes

5  22  10
  data  data  data
  next  next  next

(???)
Implement a Stack with linked Nodes

class LinkedStack<T>{
    // Fields, probably involving nodes
    public LinkedStack(); // Constructor
    public void push(T x); // Push an element
    public void pop(); // Pop an element
    public T getTop(); // Return top element
    public boolean isEmpty(); // True only when empty
}

Consider

- Which end of the stack needs to be tracked
- Is a size required
Implementations

Weiss Textbook Source

- package weiss.nonstandard.*
  - Stack interface, ArrayStack and ListStack implementations
  - ListNode and LinkedList classes
- package weiss.util.*
  - Reimplements java.util.* collections
  - Stack.java is based on arrays
- Included in today’s code pack

Java

- Deque interface - slight generalization of stack/queue
- ArrayDeque implements with arrays
- LinkedList implements with linked nodes
Next Week

- Linked lists proper
- Read Weiss Ch 16-17
- HW 1 due Sunday