Implementing abstract data types
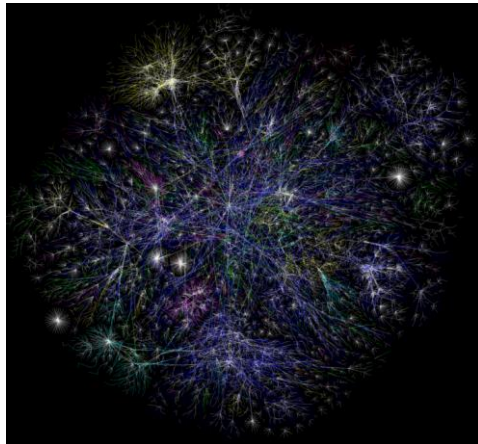

CSCI 135: Fundamentals of Computer Science • Keith Vertanen • Copyright © 2013

Overview

- Abstract Data Types (ADTs)
  - A collection of data and operations on that data
- Data structure
  - How we choose to implement an ADT
  - It is a *choice*, more than one way to skin a cat!
- Some possible choices:
  - Fixed array (last time)
  - Dynamically sized array (this time)
  - Linked data structure (this time)
    - Using object references to hook things together
    - Can create a wide-variety of structures:
      - Lists, Stacks, Queues, Graphs, Trees, …

2

FIFO Stack ADT

- Stack ADT
  - Support push/pop operations
  - Last time:
    - Fixed array data structure
    - Easy to implement
    - But may break if fixed size too small

http://www.flickr.com/photos/mac-ash/4534203626/

```
public class StackOfStringsArray
------------------------------------------------------------------------
         StackOfStringsArray(int max) // Construct a new stack with max size
    void push(String s)               // Add a new string to the queue
  String pop()                        // Remove the least recently added string
 boolean isEmpty()                    // Check if the queue is empty
  String toString()                   // Get string representation of stack
```

3

```
public class StackOfStringsArray
{
    private String [] items;  // Holds the items in the stack
    private int last;         // Location of the next available array position
    public StackOfStringsArray(int max)
    {
        items = new String[max];
        last = 0;
    }
    public void push(String s)
    {
        if (last == items.length)
            throw new RuntimeException("Stack is full!");
        items[last] = s;
        last++;
    }
    public String pop()
    {
        if (last == 0)
            throw new RuntimeException("Stack is empty!");
        last--;
        return items[last];
    }
    public boolean isEmpty()
    {
        return (last == 0);
    }
    public String toString()
    {
        String result = "";
        for (int i = 0; i < last; i++)
        {
            if (i > 0)
                result += " ";
            result += items[i];
        }
        return result;
    }
}
```

We'd like it if this never could happen. Users of our ADT should be able to push() until the cows come home.

We can't really prevent this from happening. User of the ADT should have checked isEmpty() first.

4

Fixed array Stack vs. Moby Dick

- Goal: Print backwards version of Moby Dick

```
Loomings

Call me Ishmael. Some years ago- never mind how long precisely-
having little or no money in my purse, and nothing particular to
interest me on shore, I thought I would sail about a little and see
the watery part of the world. It is a way I have of driving off the
```
mobydick.txt

```
% java ReverseWords < mobydick.txt

ago. years thousand five rolled it as on rolled sea the of shroud great the and
collapsed, all then sides; steep its against beat surf white sullen a gulf; yawn
ing yet the over screaming flew fowls small Now it. with herself helmeted and he
r, with along heaven of part living a dragged had she till hell to sink not woul
d Satan, like which, ship, his with down went Ahab, of flag the in folded form
aptive whole his and upwards, thrust beak imperial his and stacks, archa
```

5

```java
public class ReverseWords1
{
    public static void main(String [] args)
    {
        StackOfStringsArray stack;
        stack = new StackOfStringsArray(100000);

        while (!StdIn.isEmpty())
            stack.push(StdIn.readString());

        while (!stack.isEmpty())
            System.out.print(stack.pop() + " ");

        System.out.println();
    }
}
```

```java
public class StackOfStringsArray
{
    private String [] items;
    private int last;

    public StackOfStringsArray(int max)
    {
        items = new String[max];
        last = 0;
    }
    ...
}
```
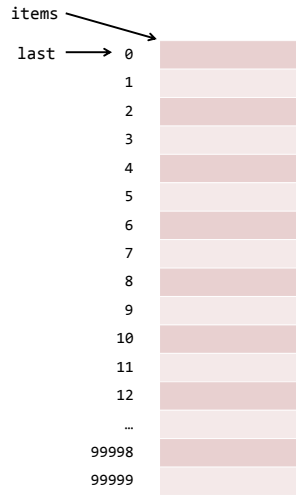
items ──→ null

last ──→ 0

6

```java
public class ReverseWords1
{
    public static void main(String [] args)
    {
        StackOfStringsArray stack;
        stack = new StackOfStringsArray(100000);

        while (!StdIn.isEmpty())
            stack.push(StdIn.readString());

        while (!stack.isEmpty())
            System.out.print(stack.pop() + " ");

        System.out.println();
    }
}
```
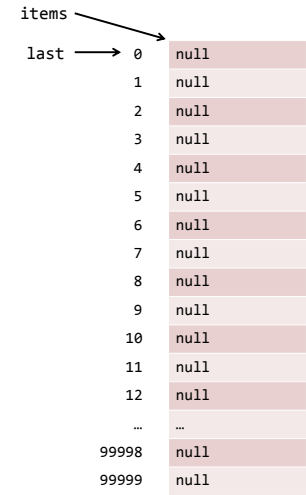
```java
public class StackOfStringsArray
{
    private String [] items;
    private int last;

    public StackOfStringsArray(int max)
    {
        items = new String[max];
        last = 0;
    }
    ...
}
```

items ──→

last ──→ 0
1
2
3
4
5
6
7
8
9
10
11
12
…
99998
99999

7

```java
public class ReverseWords1
{
    public static void main(String [] args)
    {
        StackOfStringsArray stack;
        stack = new StackOfStringsArray(100000);

        while (!StdIn.isEmpty())
            stack.push(StdIn.readString());

        while (!stack.isEmpty())
            System.out.print(stack.pop() + " ");

        System.out.println();
    }
}
```

```java
public class StackOfStringsArray
{
    private String [] items;
    private int last;

    public StackOfStringsArray(int max)
    {
        items = new String[max];
        last = 0;
    }
    ...
}
```
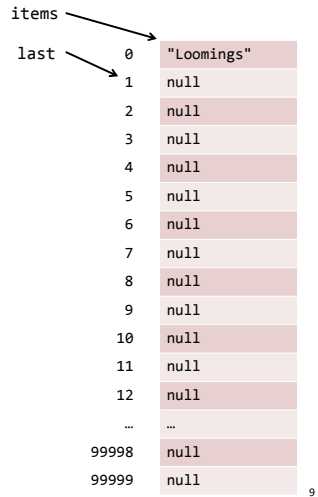
items ──→

last ──→ 0   null
1   null
2   null
3   null
4   null
5   null
6   null
7   null
8   null
9   null
10   null
11   null
12   null
…   …
99998 null
99999 null

8

Slide 9

```java
public class ReverseWords1
{
    public static void main(String [] args)
    {
        StackOfStringsArray stack;
        stack = new StackOfStringsArray(100000);

        while (!StdIn.isEmpty())
➡️          stack.push(StdIn.readString());

        while (!stack.isEmpty())
            System.out.print(stack.pop() + " ");

        System.out.println();
    }
}
```

```java
public class StackOfStringsArray
{
    private String [] items;
    private int last;

    public StackOfStringsArray(int max)
    {
        items = new String[max];
        last = 0;
    }
    ...
}
```
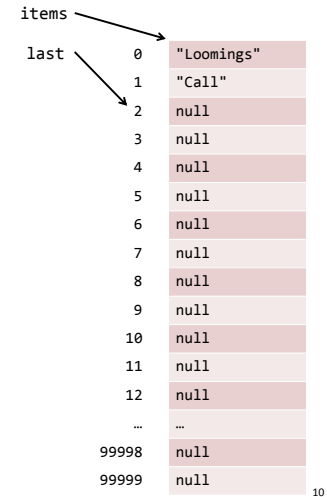
items →

last →

0	"Loomings"
1	null
2	null
3	null
4	null
5	null
6	null
7	null
8	null
9	null
10	null
11	null
12	null
…	…
99998	null
99999	null

9

Slide 10

```java
public class ReverseWords1
{
    public static void main(String [] args)
    {
        StackOfStringsArray stack;
        stack = new StackOfStringsArray(100000);

        while (!StdIn.isEmpty())
➡️          stack.push(StdIn.readString());

        while (!stack.isEmpty())
            System.out.print(stack.pop() + " ");

        System.out.println();
    }
}
```

```java
public class StackOfStringsArray
{
    private String [] items;
    private int last;

    public StackOfStringsArray(int max)
    {
        items = new String[max];
        last = 0;
    }
    ...
}
```
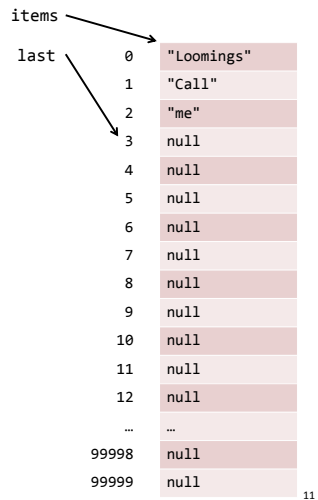
items →

last →

0	"Loomings"
1	"Call"
2	null
3	null
4	null
5	null
6	null
7	null
8	null
9	null
10	null
11	null
12	null
…	…
99998	null
99999	null

10

Slide 11

```java
public class ReverseWords1
{
    public static void main(String [] args)
    {
        StackOfStringsArray stack;
        stack = new StackOfStringsArray(100000);

        while (!StdIn.isEmpty())
➡️          stack.push(StdIn.readString());

        while (!stack.isEmpty())
            System.out.print(stack.pop() + " ");

        System.out.println();
    }
}
```

```java
public class StackOfStringsArray
{
    private String [] items;
    private int last;

    public StackOfStringsArray(int max)
    {
        items = new String[max];
        last = 0;
    }
    ...
}
```
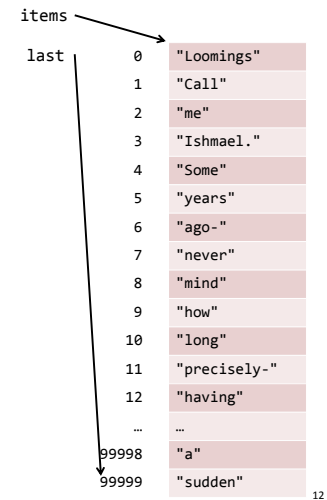
items →

last →

0	"Loomings"
1	"Call"
2	"me"
3	null
4	null
5	null
6	null
7	null
8	null
9	null
10	null
11	null
12	null
…	…
99998	null
99999	null

11

Slide 12

```java
public class ReverseWords1
{
    public static void main(String [] args)
    {
        StackOfStringsArray stack;
        stack = new StackOfStringsArray(100000);

        while (!StdIn.isEmpty())
➡️          stack.push(StdIn.readString());

        while (!stack.isEmpty())
            System.out.print(stack.pop() + " ");

        System.out.println();
    }
}
```

```java
public class StackOfStringsArray
{
    private String [] items;
    private int last;

    public StackOfStringsArray(int max)
    {
        items = new String[max];
        last = 0;
    }
    ...
}
```

items →

last →

0	"Loomings"
1	"Call"
2	"me"
3	"Ishmael."
4	"Some"
5	"years"
6	"ago-"
7	"never"
8	"mind"
9	"how"
10	"long"
11	"precisely-"
12	"having"
…	…
99998	"a"
99999	"sudden"

12

Slide 13

```java
public class ReverseWords1
{
    public static void main(String [] args)
    {
        StackOfStringsArray stack;
        stack = new StackOfStringsArray(100000);

        while (!StdIn.isEmpty())
            stack.push(StdIn.readString());

        whil

        Syst
    }
}
```

```java
public class
{
    private
    private int last;

    public StackOfStringsArray(int max)
    {
        items = new String[max];
        last = 0;
    }
    ...
}
```

items

last

0	"Loomings"
1	"Call"
2	"me"
3	"Ishmael."
...	
9	"how"
10	"long"
11	"precisely-"
...	
99999	"sudden"

THE EXPLODING WHALE.com

NOVEMBER 12, 1970

"The blast blasted blubber beyond all believable bounds." – Paul Linnman, TV news reporter

```
% java ReverseWords1 < mobydick.txt
Exception in thread "main" java.lang.RuntimeException: Stack is full!
        at StackOfStringsArray.push(StackOfStringsArray.java:17)
        at ReverseWords1.main(ReverseWords1.java:15)
```

13

Slide 14

```java
public class ReverseWords2
{
    public static void main(String [] args)
    {
        Stats stats = new Stats();

        StackOfStringsArray stack = new StackOfStringsArray(Integer.parseInt(args[0]));
        while (!StdIn.isEmpty())
        {
            stack.push(StdIn.readString());
        }

        System.out.println(stats);
    }
}
```

```
% wc -w *.txt
  209341 mobydick.txt
 3794316 wiki_200k.txt

% ls -lh *.txt
-rwx------+ 1 Administrators None 1.2M Sep 30 09:13 mobydick.txt
-rwx------+ 1 Administrators None  22M Nov 20  2010 wiki_200k.txt
```

```
% java ReverseWords2 209341 < mobydick.txt
elapsed (s)           : 0.383
heap memory used (KB) : 16074
```

```
% java ReverseWords2 3794316 < wiki_200k.txt
elapsed (s)           : 3.674
heap memory used (KB) : 244227
```

14

Slide 15

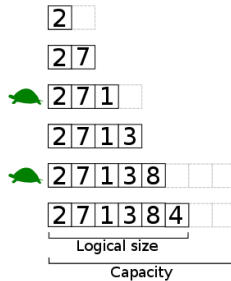# Dynamic arrays

- ## Dynamically sized array
  - Use a fixed array to store data
  - If you run out of space:
    - Creating a new bigger array
    - Copy existing data to new array
    - Java garbage collector will free up old array
  - How much to increase by?
    - Fixed number (e.g. 1)
      - Very memory efficient, but probably not very fast...
    - Double the current size
      - Increases frequent at first, but eventually lasts a long time

2

2	7

🐢 | 2 | 7 | 1 |

| 2 | 7 | 1 | 3 |

🐢 | 2 | 7 | 1 | 3 | 8 |

| 2 | 7 | 1 | 3 | 8 | 4 |

Logical size

Capacity

15

Slide 16

```java
public class StackOfStringsArrayDynamic
{
    private static final int INIT_SIZE = 16;  // Initial array size
    private String [] items;  // Holds the items in the stack
    private int last;         // Location of the next available array position

    public StackOfStringsArrayDynamic()
    {
        items = new String[INIT_SIZE];
        last = 0;
    }

    public void push(String s)
    {
        if (last == items.length)
        {
            String [] bigger = new String[items.length + 1];
            for (int i = 0; i < items.length; i++)
                bigger[i] = items[i];
            items = bigger;
        }
        items[last] = s;
        last++;
    }
}
```

Make one bigger, copy the data into the new array and then update the instance variable to point to the new array.

StackOfStringArrayDynamic

```
% java ReverseWords3 < mobydick.txt
elapsed (s)           : 41.938
heap memory used (KB) : 15997
```

```
% java ReverseWords3 < wiki_200k.txt
elapsed (s)           : 24921.821
heap memory used (KB) : 3071061
```

StackOfStringArray

```
% java ReverseWords2 209341 < mobydick.txt
elapsed (s)           : 0.383
heap memory used (KB) : 16074
```

```
% java ReverseWords2 3794316 < wiki_200k.txt
elapsed (s)           : 3.674
heap memory used (KB) : 244227
```

16

Slide 17

```java
public class StackOfStringsArrayDouble
{
    private static final int INIT_SIZE = 16;  // Initial array size
    private String [] items;  // Holds the items in the stack
    private int last;         // Location of the next available array position

    public StackOfStringsArrayDouble()
    {
        items = new String[INIT_SIZE];
        last = 0;
    }

    public void push(String s)
    {
        if (last == items.length)
        {
            String [] bigger = new String[items.length * 2];
            for (int i = 0; i < items.length; i++)
                bigger[i] = items[i];
            items = bigger;
        }
        items[last] = s;
        last++;
    }
```

Double the size and copy into the bigger array whenever we run out of space.

StackOfStringArrayDouble	StackOfStringArray
% java ReverseWords4 < mobydick.txt elapsed (s) : 0.391 heap memory used (KB) : 17431	% java ReverseWords2 209341 < mobydick.txt elapsed (s) : 0.383 heap memory used (KB) : 16074
% java ReverseWords4 < wiki_200k.txt elapsed (s) : 3.614 heap memory used (KB) : 254760	% java ReverseWords2 3794316 < wiki_200k.txt elapsed (s) : 3.674 heap memory used (KB) : 244227

17

Slide 18

```java
import java.util.ArrayList;

public class StackOfStringsArrayList
{
    private ArrayList<String> items = new ArrayList<String>();

    public void push(String s)
    {
        items.add(s);
    }
    public String pop()
    {
        if (items.size() == 0)
            throw new RuntimeException("Stack is empty!");
        return items.remove(items.size() - 1);
    }
    ...
}
```

Use the Java built-in ArrayList class. Turns out it operates similar to our array doubling implementation.

From the javadoc:
"As elements are added to an ArrayList, its capacity grows automatically. The details of the growth policy are not specified beyond the fact that adding an element has constant amortized time cost."

StackOfStringArrayList	StackOfStringArrayDouble
% java ReverseWords5 < mobydick.txt elapsed (s) : 0.364 heap memory used (KB) : 18419	% java ReverseWords4 < mobydick.txt elapsed (s) : 0.391 heap memory used (KB) : 17431
% java ReverseWords5 < wiki_200k.txt elapsed (s) : 3.67 heap memory used (KB) : 270505	% java ReverseWords4 < wiki_200k.txt elapsed (s) : 3.614 heap memory used (KB) : 254760

18

Slide 19

Sequential vs. Linked

- **Sequential data structures**
  - Put one object next to another
    - A block of consecutive memory in the computer
  - Java: array of objects
    - Arbitrary access, "get me the ith object"
    - Fixed size
- **Linked data structures**
  - Each object has link to another (or perhaps several)
  - Java: link is a reference to another object
    - Dynamic size
    - Flexible and widely used way of organizing data
    - More challenging to code and debug

19

Slide 20

Sequential vs. Linked

Memory address	Value
C0	"The"
C1	"cat"
C2	"sat"
C3	-
C4	-
C5	-
C6	-
C7	-
C8	-
C9	-

array

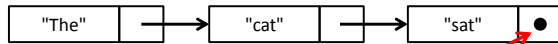| Memory address	Value
C0	"cat"
C1	C8
C2	-
C3	-
C4	"The"
C5	C0
C6	-
C7	-
C8	"sat"
C9	null

linked list

20

Linked list

- Linked list
  - Simplest linked data structure
  - A recursive data structure
  - Each node contains:
    - An item (some data)
    - A pointer to next node in the list

```
private class Node
{
    private String item;
    private Node next;
}
```

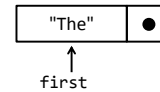Three Node objects hooked together to form a linked list


```
"The" ──→ "cat" ──→ "sat" ●
```

Special pointer value null terminates the list. We denote with a dot.

Building a linked list

```
Node first  = new Node();
first.item = "The";
```

Memory address	Value
C0	-
C1	-
C2	-
C3	-
C4 (first →)	"The"
C5	null
C6	-
C7	-
C8	-
C9	-

```
"The" ●
  ↑
first
```

Building a linked list

```
Node first  = new Node();
first.item = "The";

Node second = new Node();
second.item = "cat";
```

Memory address	Value
C0 (second →)	"cat"
C1	null
C2	-
C3	-
C4 (first →)	"The"
C5	null
C6	-
C7	-
C8	-
C9	-

```
"The" ●        "cat" ●
  ↑              ↑
first          second
```

Building a linked list

```
Node first  = new Node();
first.item = "The";

Node second = new Node();
second.item = "cat";

Node third  = new Node();
third.item = "sat";
```

Memory address	Value
C0 (second →)	"cat"
C1	null
C2	-
C3	-
C4 (first →)	"The"
C5	null
C6	-
C7	-
C8 (third →)	"sat"
C9	null

```
"The" ●        "cat" ●        "sat" ●
  ↑              ↑              ↑
first          second         third
```
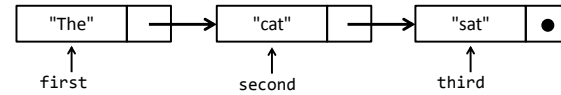
Building a linked list

```
Node first  = new Node();
first.item  = "The";

Node second = new Node();
second.item = "cat";

Node third  = new Node();
third.item  = "sat";

first.next  = second;
```

Memory address	Value
C0	"cat"
C1	null
C2	-
C3	-
C4	"The"
C5	C0
C6	-
C7	-
C8	"sat"
C9	null

second → C0
first → C4
third → C8

"The" → "cat" → "sat" ●
first   second   third

Building a linked list

```
Node first  = new Node();
first.item  = "The";

Node second = new Node();
second.item = "cat";

Node third  = new Node();
third.item  = "sat";

first.next  = second;
second.next = third;
```

Memory address	Value
C0	"cat"
C1	C8
C2	-
C3	-
C4	"The"
C5	C0
C6	-
C7	-
C8	"sat"
C9	null

second → C0
first → C4
third → C8

"The" → "cat" → "sat" ●
first   second   third

Traversing a list

- Iterate over all elements in a linked list
  - Assume list is null terminated
  - Print all the strings in the list

```
Node current = first;
while (current != null)
{
    System.out.println(current.item);
    current = current.next;
}
```

```
for (Node current = first; current != null; current = current.next)
    System.out.println(current.item);
```
shorthand version

"The" → "cat" → "sat" ●
first

Traversing a list

```
Node current = first;
while (current != null)
{
    System.out.println(current.item);
    current = current.next;
}
```
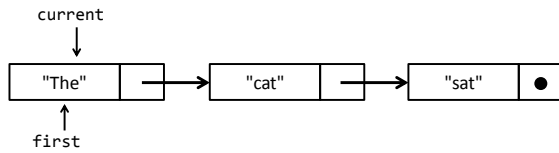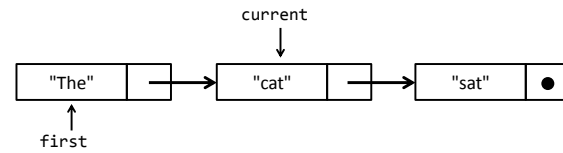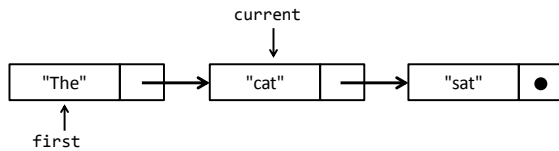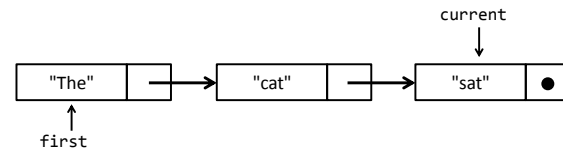
current

"The" → "cat" → "sat" ●
first

Traversing a list

```
Node current = first;
while (current != null)
{
➡  System.out.println(current.item);
    current = current.next;
}
```
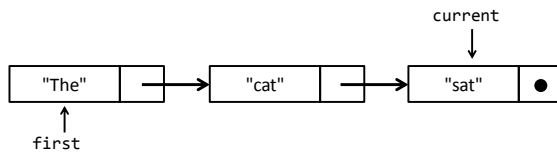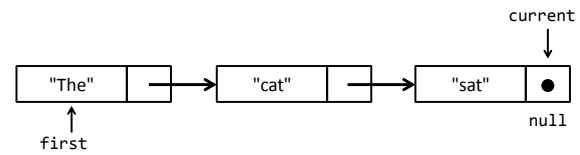
The

current

"The" → "cat" → "sat" ●

first

Traversing a list

```
Node current = first;
while (current != null)
{
    System.out.println(current.item);
➡  current = current.next;
}
```

The

current

"The" → "cat" → "sat" ●

first

Traversing a list

```
Node current = first;
while (current != null)
{
➡  System.out.println(current.item);
    current = current.next;
}
```

The
cat

current

"The" → "cat" → "sat" ●

first

Traversing a list

```
Node current = first;
while (current != null)
{
    System.out.println(current.item);
➡  current = current.next;
}
```

The
cat

current

"The" → "cat" → "sat" ●

first

Traversing a list

```
Node current = first;
while (current != null)
{
➡   System.out.println(current.item);
    current = current.next;
}
```
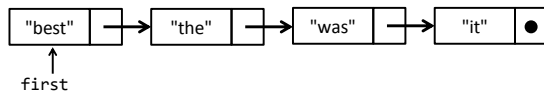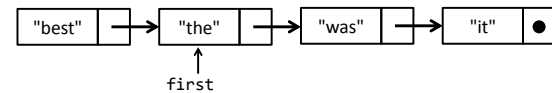
```
The
cat
sat
```

current
↓

| "The" | | → | "cat" | | → | "sat" | ● |

↑
first

Traversing a list

```
Node current = first;
while (current != null)
{
    System.out.println(current.item);
➡   current = current.next;
}
```

```
The
cat
sat
```

current
↓

| "The" | | → | "cat" | | → | "sat" | ● |

↑   null
first

Stack ADT: Linked List
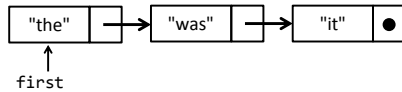
- Stack pop
  - Get the first thing in the linked list
  - Move the first pointer to next item
  - Java garbage collector will take care of orphaned Node

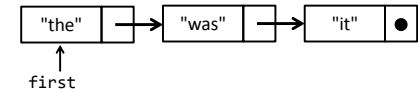| "best" | | → | "the" | | → | "was" | | → | "it" | ● |

↑
first

Stack ADT: Linked List

- Stack pop
  - Get the first thing in the linked list
  - Move the first pointer to next item
  - Java garbage collector will take care of orphaned Node

| "best" | | → | "the" | | → | "was" | | → | "it" | ● |

↑
first

Stack ADT: Linked List

- Stack pop
  - Get the first thing in the linked list
  - Move the first pointer to next item
  - Java garbage collector will take care of orphaned Node

```
"the" → "was" → "it" ●
  ↑
first
```
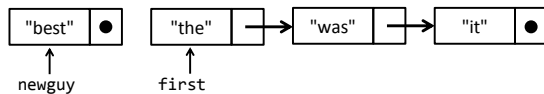
Stack ADT: Linked List

- Stack push
  - Create a new Node to hold the data
  - Hook the new Node up to the previous first item
  - Update first to point to new Node

```
"the" → "was" → "it" ●
  ↑
first
```
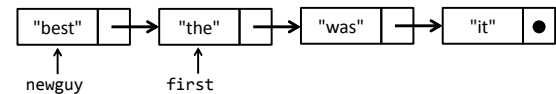
Stack ADT: Linked List

- Stack push
  - Create a new Node to hold the data
  - Hook the new Node up to the previous first item
  - Update first to point to new Node

```
"best" ●    "the" → "was" → "it" ●
  ↑           ↑
newguy       first
```

Stack ADT: Linked List
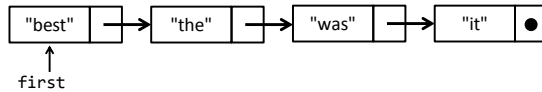
- Stack push
  - Create a new Node to hold the data
  - Hook the new Node up to the previous first item
  - Update first to point to new Node

```
"best" → "the" → "was" → "it" ●
  ↑         ↑
newguy     first
```

Stack ADT: Linked List

- Stack push
  - Create a new Node to hold the data
  - Hook the new Node up to the previous first item
  - Update first to point to new Node

| "best" | → | "the" | → | "was" | → | "it" | ● |

first

Summary

- Stack ADT
  - Explored different possible data structures:
    - Fixed array
    - Dynamic array that grows by one
      - Memory efficient, but slow
    - Dynamic array that doubles in size
      - Can require up to 2x memory, but fast (usually)
    - Using a Java ArrayList (similar to doubling array)
    - Linked list
- Linked structures
  - Common construct in computer science
  - Can represent a wide-variety of useful structures
    - Lists, trees, graphs, …