Data Structures and Algorithms in Java

Algorithms and Data Structures: Basic Data Structures

Outline

1. Parametrized Type
2. Auto Boxing and Auto Unboxing
3. Linked List
4. Bag
5. Queue
6. Stack

Parametrized Type

Parametrized Type

Parametrized (aka generic) types enable us to implement collection ADTs that can store objects of any reference type

Parametrized Type

Parametrized (aka generic) types enable us to implement collection ADTs that can store objects of any reference type

Example

`LinkedStack<T> implements Stack<T>`

`LinkedStack()`	constructs an empty stack
`boolean isEmpty()`	returns `true` if this stack is empty, and `false` otherwise
`int size()`	returns the number of items in this stack
`void push(T item)`	adds `item` to the top of this stack
`T peek()`	returns the item at the top of this stack
`T pop()`	removes and returns the item at the top of this stack
`String toString()`	returns a string representation of this stack
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this stack in LIFO order

Parametrized Type

Parametrized Type

Example

```
LinkedStack<String> names = new LinkedStack<String>();
LinkedStack<Date> dates = new LinkedStack<Date>();
LinkedStack<Double> amounts = new LinkedStack<Double>();
names.push("Turing");
dates.push(new Date(6, 17, 1990));
amounts.push(644.08);
String name = names.pop();
Date date = dates.pop();
double amount = amounts.pop();
```

Auto Boxing and Auto Unboxing

Auto Boxing and Auto Unboxing

Java automatically converts a primitive to an object (auto boxing) and vice versa (auto unboxing)

Auto Boxing and Auto Unboxing

Java automatically converts a primitive to an object (auto boxing) and vice versa (auto unboxing)

Example

```
LinkedStack<Double> amounts = new LinkedStack<Double>();
amounts.push(644.08);        // auto boxing (double -> Double)
double amount = stack.pop(); // auto unboxing (Double -> double)
```

Auto Boxing and Auto Unboxing

Java automatically converts a primitive to an object (auto boxing) and vice versa (auto unboxing)

Example

```
LinkedStack<Double> amounts = new LinkedStack<Double>();
amounts.push(644.08);        // auto boxing (double -> Double)
double amount = stack.pop(); // auto unboxing (Double -> double)
```

Wrapper types

boolean	Boolean
byte	Byte
char	Character
short	Short
int	Integer
float	Float
long	Long
double	Double

Linked List

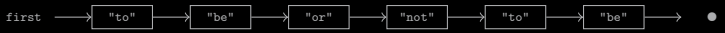
Linked List

A collection that is empty (ie, `null`) or is a reference to a node storing a generic item and a reference to the rest of the list

Linked List

A collection that is empty (ie, `null`) or is a reference to a node storing a generic item and a reference to the rest of the list

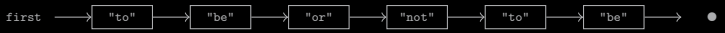
Example



Linked List

A collection that is empty (ie, `null`) or is a reference to a node storing a generic item and a reference to the rest of the list

Example



Linked list ADT

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

Linked List

Removing a node from the front

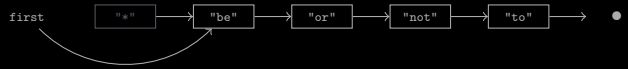
```
first = first.next;
```

Removing a node from the front

```
first = first.next;
```

Removing a node from the front

```
first = first.next;
```



Removing a node from the front

```
first = first.next;
```

Linked List

Linked List

Adding a node at the front

```
Node oldFirst = first;
first = new Node();
first.item = "to";
first.next = oldFirst;
```

Linked List

Adding a node at the front

```
Node oldFirst = first;
first = new Node();
first.item = "to";
first.next = oldFirst;
```

Linked List

Adding a node at the front

```
Node oldFirst = first;
first = new Node();
first.item = "to";
first.next = oldFirst;
```

Linked List

Adding a node at the front

```
Node oldFirst = first;
first = new Node();
first.item = "to";
first.next = oldFirst;
```

Linked List

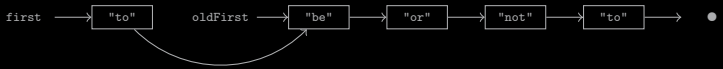
Adding a node at the front

```
Node oldFirst = first;
first = new Node();
first.item = "to";
first.next = oldFirst;
```

Linked List

Adding a node at the front

```
Node oldFirst = first;
first = new Node();
first.item = "to";
first.next = oldFirst;
```



Linked List

Adding a node at the front

```
Node oldFirst = first;
first = new Node();
first.item = "to";
first.next = oldFirst;
```

Linked List

Adding a node at the end

```
Node oldLast = last;
last = new Node();
last.item = "be";
oldLast.next = last;
```

Linked List

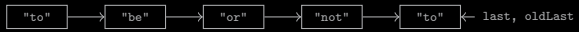
Adding a node at the end

```
Node oldLast = last;
last = new Node();
last.item = "be";
oldLast.next = last;
```

Linked List

Adding a node at the end

```
Node oldLast = last;
last = new Node();
last.item = "be";
oldLast.next = last;
```



Linked List

Adding a node at the end

```
Node oldLast = last;
last = new Node();
last.item = "be";
oldLast.next = last;
```

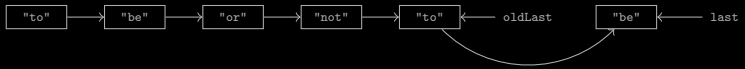
Linked List

Adding a node at the end

```
Node oldLast = last;
last = new Node();
last.item = "be";
oldLast.next = last;
```

Adding a node at the end

```
Node oldLast = last;
last = new Node();
last.item = "be";
oldLast.next = last;
```



Linked List

Adding a node at the end

```
Node oldLast = last;
last = new Node();
last.item = "be";
oldLast.next = last;
```

Linked List

Linked List

Traversing

```
for (Node x = first; x != null; x = x.next) {
    StdOut.print(x.item + " ");
}
```

or

```
Node x = first;
while (x != null) {
    StdOut.print(x.item + " ");
    x = x.next;
}
```

Linked List

Traversing

```
for (Node x = first; x != null; x = x.next) {
    StdOut.print(x.item + " ");
}
```

or

```
Node x = first;
while (x != null) {
    StdOut.print(x.item + " ");
    x = x.next;
}
```


```
>_
```

Linked List

Traversing

```
for (Node x = first; x != null; x = x.next) {
    StdOut.print(x.item + " ");
}
```

or

```
Node x = first;
while (x != null) {
    StdOut.print(x.item + " ");
    x = x.next;
}
```


```
>_
to be or not to be
```

Linked List

Operation	T(n)
Removing a node from the front	1
Adding a node at the front	1
Adding a node at the end	1
Traversing	n

Bag

Bag

An dynamic, iterable collection that can store objects of any (reference) type

Bag

An dynamic, iterable collection that can store objects of any (reference) type

dsa.Bag<T> extends *java.lang.Iterable<T>*	
`boolean isEmpty()`	returns `true` if this bag is empty, and `false` otherwise
`int size()`	returns the number of items in this bag
`void add(T item)`	adds `item` to this bag
`String toString()`	returns a string representation of this bag
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this bag

Stats.java

Standard input	a sequence of doubles
Standard output	their mean and standard deviation

Stats.java

Standard input	a sequence of doubles
Standard output	their mean and standard deviation

~/workspace/dsaj

```
$ _
```

Stats.java

Standard input	a sequence of doubles
Standard output	their mean and standard deviation

~/workspace/dsaj

```
$ java Stats
```

Stats.java

Standard input	a sequence of doubles
Standard output	their mean and standard deviation

~/workspace/dsaj

```
$ java Stats
-
```

Stats.java

Standard input	a sequence of doubles
Standard output	their mean and standard deviation

~/workspace/dsaj

```
$ java Stats
1 3 5 7 9
```

Stats.java

Standard input	a sequence of doubles
Standard output	their mean and standard deviation

~/workspace/dsaj

```
$ java Stats
1 3 5 7 9
-
```

Bag · Example (Basic Statistics)

Stats.java

Standard input	a sequence of doubles
Standard output	their mean and standard deviation

~/workspace/dsaj

```
$ java Stats
1 3 5 7 9
<ctrl-d>
```

Bag · Example (Basic Statistics)

Stats.java

Standard input	a sequence of doubles
Standard output	their mean and standard deviation

>_ ~/workspace/dsaj

```
$ java Stats
1 3 5 7 9
<ctrl-d>
Mean:    5.00
Std dev: 3.16
$ _
```

Bag · Example (Basic Statistics)

</> Stats.java

```java
import dsa.LinkedBag;
import stdlib.StdIn;
import stdlib.StdOut;

public class Stats {
    public static void main(String[] args) {
        LinkedBag<Double> bag = new LinkedBag<Double>();
        while (!StdIn.isEmpty()) {
            bag.add(StdIn.readDouble());
        }
        int n = bag.size();
        double sum = 0.0;
        for (double x : bag) {
            sum += x;
        }
        double mean = sum / n;
        sum = 0.0;
        for (double x : bag) {
            sum += (x - mean) * (x - mean);
        }
        double stddev = Math.sqrt(sum / (n - 1));
        StdOut.printf("Mean:    %.2f\n", mean);
        StdOut.printf("Std dev: %.2f\n", stddev);
    }
}
```

Bag · Linked Bag

`dsa.LinkedBag<T>` implements `dsa.Bag<T>`

`LinkedBag()`	constructs an empty bag
`boolean isEmpty()`	returns `true` if this bag is empty, and `false` otherwise
`int size()`	returns the number of items in this bag
`void add(T item)`	adds `item` to this bag
`String toString()`	returns a string representation of this bag
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this bag

dsa.LinkedBag<T> implements dsa.Bag<T>	
`LinkedBag()`	constructs an empty bag
`boolean isEmpty()`	returns `true` if this bag is empty, and `false` otherwise
`int size()`	returns the number of items in this bag
`void add(T item)`	adds `item` to this bag
`String toString()`	returns a string representation of this bag
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this bag

Instance variables

- Reference to the front: `Node first`

- Number of items: `int n`

Bag · Linked Bag

</> LinkedBag.java 1/2

```java
package dsa;

import java.util.Iterator;
import stdlib.StdIn;
import stdlib.StdOut;

public class LinkedBag<T> implements Bag<T> {
    private Node first;
    private int n;

    public LinkedBag() {
        this.first = null;
        this.n = 0;
    }

    public boolean isEmpty() {
        return this.size() == 0;
    }

    public int size() {
        return this.n;
    }

    public void add(T item) {
        Node oldFirst = this.first;
        this.first = new Node();
        this.first.item = item;
        this.first.next = oldFirst;
        this.n++;
    }

    public String toString() {
        String s = "";
        for (T item : this) {
            s += item + ", ";
```

LinkedBag.java 2/2

```
36          }
37          return this.isEmpty() ? s + "[]" : "[" + s.substring(0, s.length() - 2) + "]";
38      }
39
40      public Iterator<T> iterator() {
41          return new LinkedListIterator();
42      }
43
44      private class Node {
45          private T item;
46          private Node next;
47      }
48
49      private class LinkedListIterator implements Iterator<T> {
50          private Node current;
51
52          public LinkedListIterator() {
53              this.current = first;
54          }
55
56          public boolean hasNext() {
57              return this.current != null;
58          }
59
60          public T next() {
61              T item = this.current.item;
62              this.current = this.current.next;
63              return item;
64          }
65      }
66
67      public static void main(String[] args) {
68          // Unit tests the data type
69      }
70  }
```

Operation	T(n)
`LinkedBag()`	1
`boolean isEmpty()`	1
`int size()`	1
`void add(T item)`	1
`Iterator<T> iterator()`	1

`dsa.ResizingArrayBag<T> implements dsa.Bag<T>`

`ResizingArrayBag()`	constructs an empty bag
`boolean isEmpty()`	returns `true` if this bag is empty, and `false` otherwise
`int size()`	returns the number of items in this bag
`void add(T item)`	adds `item` to this bag
`String toString()`	returns a string representation of this bag
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this bag

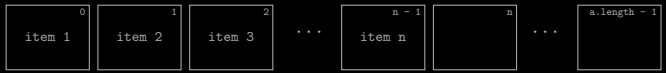
Bag · Resizing-Array Bag

`dsa.ResizingArrayBag<T> implements dsa.Bag<T>`

`ResizingArrayBag()`	constructs an empty bag
`boolean isEmpty()`	returns `true` if this bag is empty, and `false` otherwise
`int size()`	returns the number of items in this bag
`void add(T item)`	adds `item` to this bag
`String toString()`	returns a string representation of this bag
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this bag

Instance variables

- Array of items: `T[] a`



- Number of items: `int n`

Bag · Resizing-Array Bag

</> ResizingArrayBag.java 1/2

```java
package dsa;

import java.util.Iterator;
import stdlib.StdIn;
import stdlib.StdOut;

public class ResizingArrayBag<T> implements Bag<T> {
    private T[] a;
    private int n;

    public ResizingArrayBag() {
        this.a = (T[]) new Object[2];
        this.n = 0;
    }

    public boolean isEmpty() {
        return this.size() == 0;
    }

    public int size() {
        return this.n;
    }

    public void add(T item) {
        if (this.n == this.a.length) {
            resize(2 * this.a.length);
        }
        this.a[this.n++] = item;
    }

    public String toString() {
        String s = "";
        for (T item : this) {
            s += item + ", ";
        }
```

</> ResizingArrayBag.java 2/2

```
36          return this.isEmpty() ? s + "[]" : "[" + s.substring(0, s.length() - 2) + "]";
37      }
38
39      public Iterator<T> iterator() {
40          return new ArrayIterator();
41      }
42
43      private void resize(int capacity) {
44          T[] temp = (T[]) new Object[capacity];
45          for (int i = 0; i < this.n; i++) {
46              temp[i] = this.a[i];
47          }
48          this.a = temp;
49      }
50
51      private class ArrayIterator implements Iterator<T> {
52          private int i;
53
54          public ArrayIterator() {
55              this.i = 0;
56          }
57
58          public boolean hasNext() {
59              return this.i < n;
60          }
61
62          public T next() {
63              return a[this.i++];
64          }
65      }
66
67      public static void main(String[] args) {
68          // Unit tests the data type
69      }
70  }
```

Operation	T(n)
`ResizingArrayBag()`	1
`boolean isEmpty()`	1
`int size()`	1
`void add(T item)`	1*
`Iterator<T> iterator()`	1

* Amortized

Queue

Queue

A dynamic, iterable collection that can store objects of any (reference) type in first-in-first-out (FIFO) order

Queue

A dynamic, iterable collection that can store objects of any (reference) type in first-in-first-out (FIFO) order

dsa.Queue<T> extends *java.lang.Iterable<T>*

`boolean isEmpty()`	returns `true` if this queue is empty, and `false` otherwise
`int size()`	returns the number of items in this queue
`void enqueue(T item)`	adds `item` to the end of this queue
`T peek()`	returns the item at the front of this queue
`T dequeue()`	removes and returns the item at the front of this queue
`String toString()`	returns a string representation of this queue
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this queue in FIFO order

KthFromLast.java

Command-line input	*k* (int)
Standard input	a sequence of integers
Standard output	*k*th integer from the end

KthFromLast.java

Command-line input	*k* (int)
Standard input	a sequence of integers
Standard output	*k*th integer from the end

~/workspace/dsaj

```
$ _
```

Queue · Example (*k*th from Last)

KthFromLast.java

Command-line input	*k* (int)
Standard input	a sequence of integers
Standard output	*k*th integer from the end

~/workspace/dsaj

```
$ java KthFromLast 4
```

Queue · Example (*k*th from Last)

KthFromLast.java

Command-line input	*k* (int)
Standard input	a sequence of integers
Standard output	*k*th integer from the end

~/workspace/dsaj

```
$ java KthFromLast 4
-
```

KthFromLast.java

Command-line input	*k* (int)
Standard input	a sequence of integers
Standard output	*k*th integer from the end

~/workspace/dsaj

```
$ java KthFromLast 4
1 2 3 4 5 6 7 8 9 10
```

KthFromLast.java

Command-line input	*k* (int)
Standard input	a sequence of integers
Standard output	*k*th integer from the end

~/workspace/dsaj

```
$ java KthFromLast 4
1 2 3 4 5 6 7 8 9 10
-
```

Queue · Example (*k*th from Last)

KthFromLast.java

Command-line input	*k* (int)
Standard input	a sequence of integers
Standard output	*k*th integer from the end

~/workspace/dsaj

```
$ java KthFromLast 4
1 2 3 4 5 6 7 8 9 10
<ctrl-d>
```

Queue · Example (*k*th from Last)

KthFromLast.java

Command-line input	*k* (int)
Standard input	a sequence of integers
Standard output	*k*th integer from the end

~/workspace/dsaj

```
$ java KthFromLast 4
1 2 3 4 5 6 7 8 9 10
<ctrl-d>
7
$ _
```

Queue · Example (*k*th from Last)

</> KthFromLast.java

```java
import dsa.LinkedQueue;

import stdlib.StdIn;
import stdlib.StdOut;

public class KthFromLast {
    public static void main(String[] args) {
        int k = Integer.parseInt(args[0]);
        LinkedQueue<String> queue = new LinkedQueue<String>();
        while (!StdIn.isEmpty()) {
            queue.enqueue(StdIn.readString());
        }
        int n = queue.size();
        for (int i = 1; i <= n - k; i++) {
            queue.dequeue();
        }
        StdOut.println(queue.peek());
    }
}
```

`dsa.LinkedQueue<T> implements dsa.Queue<T>`

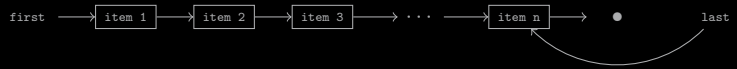
`LinkedQueue()`	constructs an empty queue
`boolean isEmpty()`	returns `true` if this queue is empty, and `false` otherwise
`int size()`	returns the number of items in this queue
`void enqueue(T item)`	adds `item` to the end of this queue
`T peek()`	returns the item at the front of this queue
`T dequeue()`	removes and returns the item at the front of this queue
`String toString()`	returns a string representation of this queue
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this queue in FIFO order

`dsa.LinkedQueue<T> implements dsa.Queue<T>`

`LinkedQueue()`	constructs an empty queue
`boolean isEmpty()`	returns `true` if this queue is empty, and `false` otherwise
`int size()`	returns the number of items in this queue
`void enqueue(T item)`	adds `item` to the end of this queue
`T peek()`	returns the item at the front of this queue
`T dequeue()`	removes and returns the item at the front of this queue
`String toString()`	returns a string representation of this queue
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this queue in FIFO order

Instance variables

- References to the front and back: `Node first` and `Node last`



- Number of items: `int n`

Queue · Linked Queue

</> LinkedQueue.java 1/3

```java
package dsa;

import java.util.Iterator;
import java.util.NoSuchElementException;
import stdlib.StdIn;
import stdlib.StdOut;

public class LinkedQueue<T> implements Queue<T> {
    private Node first;
    private Node last;
    private int n;

    public LinkedQueue() {
        this.first = null;
        this.last = null;
        this.n = 0;
    }

    public boolean isEmpty() {
        return this.size() == 0;
    }

    public int size() {
        return this.n;
    }

    public void enqueue(T item) {
        Node oldLast = this.last;
        this.last = new Node();
        this.last.item = item;
        this.last.next = null;
        if (this.isEmpty()) {
            this.first = this.last;
        } else {
            oldLast.next = this.last;
```

</> LinkedQueue.java 2/3

```
        }
        this.n++;
    }

    public T peek() {
        if (this.isEmpty()) {
            throw new NoSuchElementException("Queue is empty");
        }
        return this.first.item;
    }

    public T dequeue() {
        if (this.isEmpty()) {
            throw new NoSuchElementException("Queue is empty");
        }
        T item = this.first.item;
        this.first = this.first.next;
        this.n--;
        if (this.isEmpty()) {
            this.last = null;
        }
        return item;
    }

    public String toString() {
        String s = "";
        for (T item : this) {
            s += item + ", ";
        }
        return this.isEmpty() ? s + "[]" : "[" + s.substring(0, s.length() - 2) + "]";
    }

    public Iterator<T> iterator() {
        return new ListIterator();
    }
```

</> LinkedQueue.java 3/3

```
71      private class Node {
72          private T item;
73          private Node next;
74      }
75
76      private class ListIterator implements Iterator<T> {
77          private Node current;
78
79          public ListIterator() {
80              this.current = first;
81          }
82
83          public boolean hasNext() {
84              return this.current != null;
85          }
86
87          public T next() {
88              T item = this.current.item;
89              this.current = this.current.next;
90              return item;
91          }
92      }
93
94      public static void main(String[] args) {
95          // Unit tests the data type
96      }
97  }
```

Operation	T(n)
`LinkedQueue()`	1
`boolean isEmpty()`	1
`int size()`	1
`void enqueue(T item)`	1
`T peek()`	1
`T dequeue()`	1
`Iterator<T> iterator()`	1

Queue · Resizing-Array Queue

`dsa.ResizingArrayQueue<T> implements dsa.Queue<T>`

`ResizingArrayQueue()`	constructs an empty queue
`boolean isEmpty()`	returns `true` if this queue is empty, and `false` otherwise
`int size()`	returns the number of items in this queue
`void enqueue(T item)`	adds `item` to the end of this queue
`T peek()`	returns the item at the front of this queue
`T dequeue()`	removes and returns the item at the front of this queue
`String toString()`	returns a string representation of this queue
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this queue in FIFO order

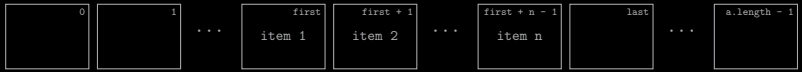
Queue · Resizing-Array Queue

`dsa.ResizingArrayQueue<T>` implements `dsa.Queue<T>`

`ResizingArrayQueue()`	constructs an empty queue
`boolean isEmpty()`	returns `true` if this queue is empty, and `false` otherwise
`int size()`	returns the number of items in this queue
`void enqueue(T item)`	adds `item` to the end of this queue
`T peek()`	returns the item at the front of this queue
`T dequeue()`	removes and returns the item at the front of this queue
`String toString()`	returns a string representation of this queue
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this queue in FIFO order

Instance variables

- Array of items: `T[] a`



- Index of the first item: `int first`
- Inded of the next available slot: `int last`
- Number of items: `int n`

Queue · Resizing-Array Queue

</> ResizingArrayQueue.java 1/3

```java
package dsa;

import java.util.Iterator;
import java.util.NoSuchElementException;
import stdlib.StdIn;
import stdlib.StdOut;

public class ResizingArrayQueue<T> implements Queue<T> {
    private T[] a;
    private int first;
    private int last;
    private int n;

    public ResizingArrayQueue() {
        this.a = (T[]) new Object[2];
        this.first = 0;
        this.last = 0;
        this.n = 0;
    }

    public boolean isEmpty() {
        return this.size() == 0;
    }

    public int size() {
        return this.n;
    }

    public void enqueue(T item) {
        if (this.n == this.a.length) {
            this.resize(2 * this.a.length);
        }
        this.a[this.last++] = item;
        if (this.last == this.a.length) {
            this.last = 0;
```

</> ResizingArrayQueue.java 2/3

```
        }
        this.n++;
    }

    public T peek() {
        if (this.isEmpty()) {
            throw new NoSuchElementException("Queue is empty");
        }
        return this.a[this.first];
    }

    public T dequeue() {
        if (this.isEmpty()) {
            throw new NoSuchElementException("Queue is empty");
        }
        T item = this.a[this.first];
        this.a[this.first] = null;
        this.n--;
        this.first++;
        if (this.first == this.a.length) {
            this.first = 0;
        }
        if (this.n > 0 && this.n == this.a.length / 4) {
            this.resize(this.a.length / 2);
        }
        return item;
    }

    public String toString() {
        String s = "";
        for (T item : this) {
            s += item + ", ";
        }
        return this.isEmpty() ? s + "[]" : "[" + s.substring(0, s.length() - 2) + "]";
    }

```

</> ResizingArrayQueue.java 3/3

```
71      public Iterator<T> iterator() {
72          return new ArrayIterator();
73      }
74
75      private void resize(int capacity) {
76          T[] temp = (T[]) new Object[capacity];
77          for (int i = 0; i < this.n; i++) {
78              temp[i] = this.a[(this.first + i) % this.a.length];
79          }
80          this.a = temp;
81          this.first = 0;
82          this.last = this.n;
83      }
84
85      private class ArrayIterator implements Iterator<T> {
86          private int i;
87
88          public ArrayIterator() {
89              this.i = 0;
90          }
91
92          public boolean hasNext() {
93              return this.i < n;
94          }
95
96          public T next() {
97              T item = a[(this.i++ + first) % a.length];
98              return item;
99          }
100     }
101
102     public static void main(String[] args) {
103         // Unit tests the data type
104     }
105 }
```

Operation	T(n)
`ResizingArrayQueue()`	1
`boolean isEmpty()`	1
`int size()`	1
`void enqueue(T item)`	1*
`T peek()`	1
`T dequeue()`	1*
`Iterator<T> iterator()`	1

* Amortized

Stack

Stack

A dynamic, iterable collection that can store objects of any (reference) type in last-in-first-out (LIFO) order

Stack

A dynamic, iterable collection that can store objects of any (reference) type in last-in-first-out (LIFO) order

dsa.Stack<T> extends *java.lang.Iterable<T>*

`boolean isEmpty()`	returns `true` if this stack is empty, and `false` otherwise
`int size()`	returns the number of items in this stack
`void push(T item)`	adds `item` to the top of this stack
`T peek()`	returns the item at the top of this stack
`T pop()`	removes and returns the item at the top of this stack
`String toString()`	returns a string representation of this stack
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this stack in LIFO order

Reverse.java

Standard input	a sequence of strings
Standard output	the strings in reverse

Reverse.java

Standard input	a sequence of strings
Standard output	the strings in reverse

>_ ~/workspace/dsaj

```
$ _
```

Reverse.java

Standard input	a sequence of strings
Standard output	the strings in reverse

>_ ~/workspace/dsaj

```
$ java Reverse
```

Reverse.java

Standard input	a sequence of strings
Standard output	the strings in reverse

```
>_ ~/workspace/dsaj

$ java Reverse
-
```

Reverse.java

Standard input	a sequence of strings
Standard output	the strings in reverse

~/workspace/dsaj

```
$ java Reverse
b o l t o n
```

Stack · Example (Reverse)

Reverse.java

Standard input	a sequence of strings
Standard output	the strings in reverse

~/workspace/dsaj

```
$ java Reverse
b o l t o n
-
```

Reverse.java

Standard input	a sequence of strings
Standard output	the strings in reverse

>_ ~/workspace/dsaj

```
$ java Reverse
b o l t o n
<ctrl-d>
```

Reverse.java

Standard input	a sequence of strings
Standard output	the strings in reverse

~/workspace/dsaj

```
$ java Reverse
b o l t o n
<ctrl-d>
n o t l o b
$ _
```

Stack · Example (Reverse)

</> Reverse.java

```java
import dsa.LinkedStack;
import stdlib.StdIn;
import stdlib.StdOut;

public class Reverse {
    public static void main(String[] args) {
        LinkedStack<String> stack = new LinkedStack<String>();
        while (!StdIn.isEmpty()) {
            String s = StdIn.readString();
            stack.push(s);
        }
        for (String s : stack) {
            StdOut.print(s + " ");
        }
        StdOut.println();
    }
}
```

dsa.LinkedStack<T> implements dsa.Stack<T>	
`LinkedStack()`	constructs an empty stack
`boolean isEmpty()`	returns `true` if this stack is empty, and `false` otherwise
`int size()`	returns the number of items in this stack
`void push(T item)`	adds `item` to the top of this stack
`T peek()`	returns the item at the top of this stack
`T pop()`	removes and returns the item at the top of this stack
`String toString()`	returns a string representation of this stack
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this stack in LIFO order

dsa.LinkedStack<T> implements dsa.Stack<T>

`LinkedStack()`	constructs an empty stack
`boolean isEmpty()`	returns `true` if this stack is empty, and `false` otherwise
`int size()`	returns the number of items in this stack
`void push(T item)`	adds `item` to the top of this stack
`T peek()`	returns the item at the top of this stack
`T pop()`	removes and returns the item at the top of this stack
`String toString()`	returns a string representation of this stack
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this stack in LIFO order

Instance variables

- Reference to the top: `Node first`

- Number of items: `int n`

</> LinkedStack.java 1/3

```java
package dsa;

import java.util.Iterator;
import java.util.NoSuchElementException;
import stdlib.StdIn;
import stdlib.StdOut;

public class LinkedStack<T> implements Stack<T> {
    private Node first;
    private int n;

    public LinkedStack() {
        this.first = null;
        this.n = 0;
    }

    public boolean isEmpty() {
        return this.size() == 0;
    }

    public int size() {
        return this.n;
    }

    public void push(T item) {
        Node oldFirst = this.first;
        this.first = new Node();
        this.first.item = item;
        this.first.next = oldFirst;
        this.n++;
    }

    public T peek() {
        if (isEmpty()) {
            throw new NoSuchElementException("Stack is empty");
```

</> LinkedStack.java 2/3

```
36          }
37          return this.first.item;
38      }
39
40      public T pop() {
41          if (isEmpty()) {
42              throw new NoSuchElementException("Stack is empty");
43          }
44          T item = this.first.item;
45          this.first = this.first.next;
46          this.n--;
47          return item;
48      }
49
50      public String toString() {
51          String s = "";
52          for (T item : this) {
53              s += item + ", ";
54          }
55          return this.isEmpty() ? s + "[]" : "[" + s.substring(0, s.length() - 2) + "]";
56      }
57
58      public Iterator<T> iterator() {
59          return new ListIterator();
60      }
61
62      private class Node {
63          private T item;
64          private Node next;
65      }
66
67      private class ListIterator implements Iterator<T> {
68          private Node current;
69
70          public ListIterator() {
```

</> LinkedStack.java 3/3

```
71              this.current = first;
72          }
73
74          public boolean hasNext() {
75              return this.current != null;
76          }
77
78          public T next() {
79              T item = this.current.item;
80              this.current = this.current.next;
81              return item;
82          }
83      }
84
85      public static void main(String[] args) {
86          // Unit tests the data type
87      }
88  }
```

Operation	T(n)
`LinkedStack()`	1
`boolean isEmpty()`	1
`int size()`	1
`void push(T item)`	1
`T peek()`	1
`T pop()`	1
`Iterator<T> iterator()`	1

Stack · Resizing-Array Stack

`dsa.ResizingArrayStack<T> implements dsa.Stack<T>`

`ResizingArrayStack()`	constructs an empty stack
`boolean isEmpty()`	returns `true` if this stack is empty, and `false` otherwise
`int size()`	returns the number of items in this stack
`void push(T item)`	adds `item` to the top of this stack
`T peek()`	returns the item at the top of this stack
`T pop()`	removes and returns the item at the top of this stack
`String toString()`	returns a string representation of this stack
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this stack in LIFO order

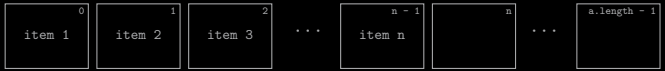
Stack · Resizing-Array Stack

`dsa.ResizingArrayStack<T> implements dsa.Stack<T>`

`ResizingArrayStack()`	constructs an empty stack
`boolean isEmpty()`	returns `true` if this stack is empty, and `false` otherwise
`int size()`	returns the number of items in this stack
`void push(T item)`	adds `item` to the top of this stack
`T peek()`	returns the item at the top of this stack
`T pop()`	removes and returns the item at the top of this stack
`String toString()`	returns a string representation of this stack
`Iterator<T> iterator()`	returns an iterator to iterate over the items in this stack in LIFO order

Instance variables

- Array of items: `T[] a`



- Number of items: `int n`

Stack · Resizing-Array Stack

</> ResizingArrayStack.java 1/3

```java
package dsa;

import java.util.Iterator;
import java.util.NoSuchElementException;
import stdlib.StdIn;
import stdlib.StdOut;

public class ResizingArrayStack<T> implements Stack<T> {
    private T[] a;
    private int n;

    public ResizingArrayStack() {
        this.a = (T[]) new Object[2];
        this.n = 0;
    }

    public boolean isEmpty() {
        return this.size() == 0;
    }

    public int size() {
        return this.n;
    }

    public void push(T item) {
        if (this.n == this.a.length) {
            this.resize(2 * a.length);
        }
        this.a[this.n++] = item;
    }

    public T peek() {
        if (isEmpty()) {
            throw new NoSuchElementException("Stack is empty");
        }
```

ResizingArrayStack.java 2/3

```
36          return this.a[this.n - 1];
37      }
38
39      public T pop() {
40          if (isEmpty()) {
41              throw new NoSuchElementException("Stack is empty");
42          }
43          T item = this.a[this.n - 1];
44          this.a[this.n - 1] = null;
45          this.n--;
46          if (this.n > 0 && this.n == this.a.length / 4) {
47              resize(this.a.length / 2);
48          }
49          return item;
50      }
51
52      public String toString() {
53          String s = "";
54          for (T item : this) {
55              s += item + ", ";
56          }
57          return this.isEmpty() ? s + "[]" : "[" + s.substring(0, s.length() - 2) + "]";
58      }
59
60      public Iterator<T> iterator() {
61          return new ReverseArrayIterator();
62      }
63
64      private void resize(int capacity) {
65          T[] temp = (T[]) new Object[capacity];
66          for (int i = 0; i < this.n; i++) {
67              temp[i] = this.a[i];
68          }
69          this.a = temp;
70      }
71
```

</> ResizingArrayStack.java 3/3

```
71      private class ReverseArrayIterator implements Iterator<T> {
72          private int i;
73
74          public ReverseArrayIterator() {
75              this.i = n - 1;
76          }
77
78          public boolean hasNext() {
79              return this.i >= 0;
80          }
81
82          public T next() {
83              return a[this.i--];
84          }
85      }
86
87      public static void main(String[] args) {
88          // Unit tests the data type
89      }
90  }
```

Operation	T(n)
`ResizingArrayStack()`	1
`boolean isEmpty()`	1
`int size()`	1
`void push(T item)`	1*
`T peek()`	1
`T pop()`	1*
`Iterator<T> iterator()`	1

* Amortized