Basic Data Structures

Outline

Generics in Java

Generics in Java

Generics (aka parametrized types) enable us to implement collection ADTs that can store any type of data

Generics in Java

Generics (aka parametrized types) enable us to implement collection ADTs that can store any type of data

Example

```
LinkedStack<String> s1 = new LinkedStack<String>();
LinkedStack<Date> s2 = new LinkedStack<Date>();
s1.push("03/14/1879");
s2.push(new Date(3, 14, 1879));
String s = s1.pop();
Date d = s2.pop();
```

Generics in Java

Java automatically converts a primitive type to the corresponding reference type (auto boxing) and vice versa (auto unboxing)

Generics in Java

Java automatically converts a primitive type to the corresponding reference type (auto boxing) and vice versa (auto unboxing)

Example

```
LinkedStack<Integer> stack = new LinkedStack<Integer>();
stack.push(42);      // auto boxing (int -> Integer)
int i = stack.pop(); // auto unboxing (Integer -> int)
```

Generics in Java

Java automatically converts a primitive type to the corresponding reference type (auto boxing) and vice versa (auto unboxing)

Example

```
LinkedStack<Integer> stack = new LinkedStack<Integer>();
stack.push(42);      // auto boxing (int -> Integer)
int i = stack.pop(); // auto unboxing (Integer -> int)
```

Wrapper types

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

Linked List

Linked List

A linked list is a data structure that is either empty (`null`) or a reference to a node having a generic item and a reference to the rest of the linked list

Linked List

A linked list is a data structure that is either empty (`null`) or a reference to a node having a generic item and a reference to the rest of the linked list

Example

Linked List

A linked list is a data structure that is either empty (`null`) or a reference to a node having a generic item and a reference to the rest of the linked list

Example

Linked list ADT

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

Linked List

Traverse a linked list

Linked List

Traverse a linked list

Linked List

Traverse a linked list


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

Linked List

Traverse a linked list


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

```
to be or not to be
```

Linked List

Linked List

Build a linked list

Linked List

Build a linked list

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

Build a linked list

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

Linked List

Insert at the beginning

Insert at the beginning

Linked List

Insert at the beginning


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

Linked List

Insert at the beginning


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

Linked List

Insert at the end

Linked List

Insert at the end

Linked List

Insert at the end


```
Node oldlast = last;
last = new Node();
last.item = "or";
oldlast.next = last;
```

Linked List

Insert at the end


```
Node oldlast = last;
last = new Node();
last.item = "or";
oldlast.next = last;
```

Linked List

Remove from the beginning

Remove from the beginning

Linked List

Remove from the beginning


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

Linked List

Remove from the beginning


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

Linked List

Operation	$T(n)$
Insert at the beginning	1
Insert at the end	1
Remove from the beginning	1

Bag

Bag

A `Bag` is an iterable collection that stores generic items

Bag

A `Bag` is an iterable collection that stores generic items

dsa.Bag<Item> extends java.lang.Iterable<Item>

`boolean isEmpty()`	returns `true` if this bag is empty, and `false` otherwise
`int size()`	returns the number of items in this bag
`void add(Item item)`	adds `item` to this bag
`Iterator<Item> iterator()`	returns an iterator to iterate over the items in this bag

Bag

Bag

Program: `Stats.java`

Bag

Program: `Stats.java`

- Standard input: a sequence of doubles

Program: `Stats.java`

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

Bag

Program: `Stats.java`

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

```
>_ ~/workspace/dsaj/programs
$ _
```

Bag

Program: `Stats.java`

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

```
>_ ~/workspace/dsaj/programs
$ java Stats
```

Bag

Program: `Stats.java`

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

```
>_ ~/workspace/dsaj/programs
$ java Stats
-
```

Bag

Program: `Stats.java`

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

```
>_ ~/workspace/dsaj/programs

$ java Stats
1 3 5 7 9
```

Bag

Program: `Stats.java`

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

```
>_ ~/workspace/dsaj/programs

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

Bag

Program: `Stats.java`

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

```
>_ ~/workspace/dsaj/programs

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

Bag

Program: `Stats.java`

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

```
>_ ~/workspace/dsaj/programs

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

Bag

Bag

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

Bag

dsa.LinkedBag<Item> implements dsa.Bag<Item>	
`LinkedBag()`	constructs an empty bag
`String toString()`	returns a string representation of this bag

Bag

dsa.LinkedBag<Item> implements dsa.Bag<Item>	
`LinkedBag()`	constructs an empty bag
`String toString()`	returns a string representation of this bag

Instance variables:

Bag

dsa.LinkedBag<Item> implements dsa.Bag<Item>	
`LinkedBag()`	constructs an empty bag
`String toString()`	returns a string representation of this bag

Instance variables:

- Reference to the front of the bag: `Node first`

Bag

dsa.LinkedBag<Item> implements dsa.Bag<Item>	
`LinkedBag()`	constructs an empty bag
`String toString()`	returns a string representation of this bag

Instance variables:

- Reference to the front of the bag: `Node first`

- Number of items in the bag: `int n`

Bag

Bag

LinkedBag.java

```
package dsa;

import java.util.Iterator;
import java.util.NoSuchElementException;

import stdlib.StdIn;
import stdlib.StdOut;

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

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

    public boolean isEmpty() {
        return first == null;
    }

    public int size() {
        return n;
    }

    public void add(Item item) {
        Node oldfirst = first;
        first = new Node();
        first.item = item;
        first.next = oldfirst;
        n++;
    }

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

Bag

LinkedBag.java

```
    }

    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (Item item : this) {
            sb.append(item);
            sb.append(", ");
        }
        return n > 0 ? "[" + sb.substring(0, sb.length() - 2) + "]" : "[]";
    }

    private class Node {
        private Item item;
        private Node next;
    }

    private class ListIterator implements Iterator<Item> {
        private Node current;

        public ListIterator() {
            current = first;
        }

        public boolean hasNext() {
            return current != null;
        }

        public Item next() {
            if (!hasNext()) {
                throw new NoSuchElementException("Iterator is empty");
            }
            Item item = current.item;
            current = current.next;
            return item;
        }
```

Bag

LinkedBag.java

```
    }

    public static void main(String[] args) {
        LinkedBag<String> bag = new LinkedBag<String>();
        while (!StdIn.isEmpty()) {
            String item = StdIn.readString();
            bag.add(item);
        }
        StdOut.println(bag.size() + " items in the bag");
        StdOut.println(bag);
    }
}
```

Bag

Bag

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

Bag

Bag

dsa.ResizingArrayBag<Item> implements dsa.Bag<Item>	
`ResizingArrayBag()`	constructs an empty bag
`String toString()`	returns a string representation of this bag

Bag

dsa.ResizingArrayBag<Item> implements dsa.Bag<Item>	
`ResizingArrayBag()`	constructs an empty bag
`String toString()`	returns a string representation of this bag

Instance variables:

Bag

dsa.ResizingArrayBag<Item> implements dsa.Bag<Item>	
`ResizingArrayBag()`	constructs an empty bag
`String toString()`	returns a string representation of this bag

Instance variables:

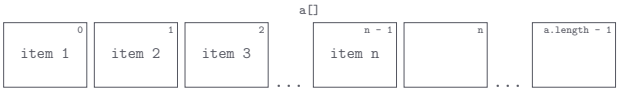
- Array of items in the bag: `Item a[]`

Bag

dsa.ResizingArrayBag<Item> implements dsa.Bag<Item>	
`ResizingArrayBag()`	constructs an empty bag
`String toString()`	returns a string representation of this bag

Instance variables:

- Array of items in the bag: `Item a[]`



- Number of items in the bag: `int n`

Bag

Bag

ResizingArrayBag.java

```
package dsa;

import java.util.Iterator;
import java.util.NoSuchElementException;

import stdlib.StdIn;
import stdlib.StdOut;

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

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

    public boolean isEmpty() {
        return n == 0;
    }

    public int size() {
        return n;
    }

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

    public Iterator<Item> iterator() {
        return new ArrayIterator();
    }
```

Bag

ResizingArrayBag.java

```
    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (Item item : this) {
            sb.append(item);
            sb.append(", ");
        }
        return n > 0 ? "[" + sb.substring(0, sb.length() - 2) + "]" : "[]";
    }

    private void resize(int capacity) {
        Item[] temp = (Item[]) new Object[capacity];
        for (int i = 0; i < n; i++) {
            temp[i] = a[i];
        }
        a = temp;
    }

    private class ArrayIterator implements Iterator<Item> {
        private int i;

        public ArrayIterator() {
            i = 0;
        }

        public boolean hasNext() {
            return i < n;
        }

        public Item next() {
            if (!hasNext()) {
                throw new NoSuchElementException("Iterator is empty");
            }
            return a[i++];
        }
```

Bag

ResizingArrayBag.java

```
    }

    public static void main(String[] args) {
        ResizingArrayBag<String> bag = new ResizingArrayBag<String>();
        while (!StdIn.isEmpty()) {
            String item = StdIn.readString();
            bag.add(item);
        }
        StdOut.println(bag.size() + " items in the bag");
        StdOut.println(bag);
    }
}
```

Bag

Bag

Operation	$T(n)$
`ResizingArrayBag()`	1
`boolean isEmpty()`	1
`int size()`	1
`void add(Item item)`	1 (amortized)
`Iterator<Item> iterator()`	1
`String toString()`	n

Queue

Queue

A `Queue` is an iterable collection that stores generic items in first-in-first-out (FIFO) order

Queue

A `Queue` is an iterable collection that stores generic items in first-in-first-out (FIFO) order

dsa.Queue<Item> extends java.lang.Iterable<Item>

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

Queue

Queue

Program: `KthFromLast.java`

Queue

Program: `KthFromLast.java`

- Command-line input: k (int)

Program: `KthFromLast.java`

- Command-line input: k (int)
- Standard input: a sequence of integers

Program: `KthFromLast.java`

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

Program: `KthFromLast.java`

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

```
>_ ~/workspace/dsaj/programs
$ _
```

Program: `KthFromLast.java`

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

```
>_ ~/workspace/dsaj/programs

$ java KthFromLast 4
```

Program: `KthFromLast.java`

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

```
>_ ~/workspace/dsaj/programs

$ java KthFromLast 4
-
```

Program: `KthFromLast.java`

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

```
>_ ~/workspace/dsaj/programs

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

Program: `KthFromLast.java`

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

```
>_ ~/workspace/dsaj/programs

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

Program: `KthFromLast.java`

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

```
>_ ~/workspace/dsaj/programs

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

Program: `KthFromLast.java`

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

```
>_ ~/workspace/dsaj/programs

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

Queue

Queue

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<Integer> queue = new LinkedQueue<Integer>();
        while (!StdIn.isEmpty()) {
            queue.enqueue(StdIn.readInt());
        }
        int n = queue.size();
        for (int i = 1; i <= n - k; i++) {
            queue.dequeue();
        }
        StdOut.println(queue.peek());
    }
}
```

Queue

Queue

dsa.LinkedQueue<Item> implements dsa.Queue<Item>	
`LinkedQueue()`	constructs an empty queue
`String toString()`	returns a string representation of this queue

Queue

dsa.LinkedQueue<Item> implements dsa.Queue<Item>	
`LinkedQueue()`	constructs an empty queue
`String toString()`	returns a string representation of this queue

Instance variables:

Queue

dsa.LinkedQueue<Item> implements dsa.Queue<Item>	
`LinkedQueue()`	constructs an empty queue
`String toString()`	returns a string representation of this queue

Instance variables:

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

Queue

dsa.LinkedQueue<Item> implements dsa.Queue<Item>	
`LinkedQueue()`	constructs an empty queue
`String toString()`	returns a string representation of this queue

Instance variables:

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

- Number of items in the queue: `int n`

Queue

Queue

LinkedQueue.java

```
package dsa;

import java.util.Iterator;
import java.util.NoSuchElementException;

import stdlib.StdIn;
import stdlib.StdOut;

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

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

    public boolean isEmpty() {
        return n == 0;
    }

    public int size() {
        return n;
    }

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

Queue

LinkedQueue.java

```
            oldlast.next = last;
        }
        n++;
    }

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

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

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

    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (Item item : this) {
            sb.append(item);
            sb.append(", ");
        }
```

Queue

LinkedQueue.java

```
        return n > 0 ? "[" + sb.substring(0, sb.length() - 2) + "]" : "[]";
    }

    private class Node {
        private Item item;
        private Node next;
    }

    private class ListIterator implements Iterator<Item> {
        private Node current;

        public ListIterator() {
            current = first;
        }

        public boolean hasNext() {
            return current != null;
        }

        public Item next() {
            if (!hasNext()) {
                throw new NoSuchElementException("Iterator is empty");
            }
            Item item = current.item;
            current = current.next;
            return item;
        }
    }

    public static void main(String[] args) {
        LinkedQueue<String> queue = new LinkedQueue<String>();
        while (!StdIn.isEmpty()) {
            String item = StdIn.readString();
            if (!item.equals("-")) {
                queue.enqueue(item);
```

Queue

LinkedQueue.java

```
            } else if (!queue.isEmpty()) {
                StdOut.print(queue.dequeue() + " ");
            }
        }
        StdOut.println();
        StdOut.println(queue.size() + " items left in the queue");
        StdOut.println(queue);
    }
}
```

Queue

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

Queue

Queue

dsa.ResizingArrayQueue<Item> implements dsa.Queue<Item>	
`ResizingArrayQueue()`	constructs an empty queue
`String toString()`	returns a string representation of this queue

Queue

dsa.ResizingArrayQueue<Item> implements dsa.Queue<Item>	
`ResizingArrayQueue()`	constructs an empty queue
`String toString()`	returns a string representation of this queue

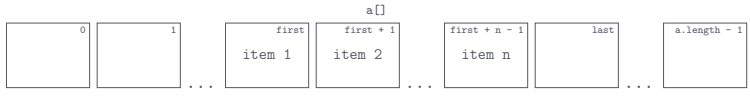
Instance variables:

Queue

dsa.ResizingArrayQueue<Item> implements dsa.Queue<Item>	
`ResizingArrayQueue()`	constructs an empty queue
`String toString()`	returns a string representation of this queue

Instance variables:

- Array of items in the queue: `Item a[]`

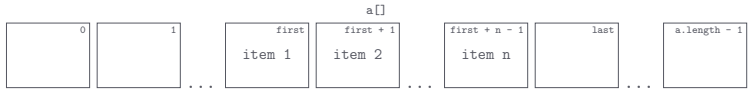


Queue

dsa.ResizingArrayQueue<Item> implements dsa.Queue<Item>	
`ResizingArrayQueue()`	constructs an empty queue
`String toString()`	returns a string representation of this queue

Instance variables:

- Array of items in the queue: `Item a[]`



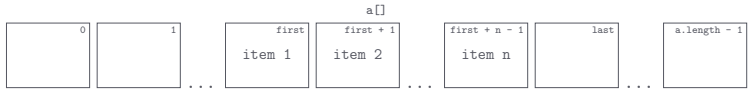
- Index of the first item: `int first`

Queue

dsa.ResizingArrayQueue<Item> implements dsa.Queue<Item>	
`ResizingArrayQueue()`	constructs an empty queue
`String toString()`	returns a string representation of this queue

Instance variables:

- Array of items in the queue: `Item a[]`



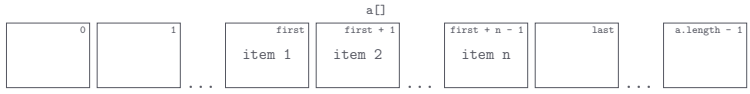
- Index of the first item: `int first`
- Index of the next new item: `int last`

Queue

dsa.ResizingArrayQueue<Item> implements dsa.Queue<Item>	
`ResizingArrayQueue()`	constructs an empty queue
`String toString()`	returns a string representation of this queue

Instance variables:

- Array of items in the queue: `Item a[]`



- Index of the first item: `int first`
- Index of the next new item: `int last`
- Number of items in the queue: `int n`

Queue

Queue

ResizingArrayQueue.java

```
package dsa;

import java.util.Iterator;
import java.util.NoSuchElementException;

import stdlib.StdIn;
import stdlib.StdOut;

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

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

    public boolean isEmpty() {
        return n == 0;
    }

    public int size() {
        return n;
    }

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

Queue

ResizingArrayQueue.java

```
            last = 0;
        }
        n++;
    }

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

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

    public Iterator<Item> iterator() {
        return new ArrayIterator();
    }

    public String toString() {
        StringBuilder sb = new StringBuilder();
```

Queue

ResizingArrayQueue.java

```
        for (Item item : this) {
            sb.append(item);
            sb.append(", ");
        }
        return n > 0 ? "[" + sb.substring(0, sb.length() - 2) + "]" : "[]";
    }

    private void resize(int capacity) {
        Item[] temp = (Item[]) new Object[capacity];
        for (int i = 0; i < n; i++) {
            temp[i] = a[(first + i) % a.length];
        }
        a = temp;
        first = 0;
        last = n;
    }

    private class ArrayIterator implements Iterator<Item> {
        private int i;

        public ArrayIterator() {
            i = 0;
        }

        public boolean hasNext() {
            return i < n;
        }

        public Item next() {
            if (!hasNext()) {
                throw new NoSuchElementException("Iterator is empty");
            }
            Item item = a[(i + first) % a.length];
            i++;
            return item;
```

Queue

ResizingArrayQueue.java

```
        }
    }

    public static void main(String[] args) {
        ResizingArrayQueue<String> queue = new ResizingArrayQueue<String>();
        while (!StdIn.isEmpty()) {
            String item = StdIn.readString();
            if (!item.equals("-")) {
                queue.enqueue(item);
            } else if (!queue.isEmpty()) {
                StdOut.print(queue.dequeue() + " ");
            }
        }
        StdOut.println();
        StdOut.println(queue.size() + " items left in the queue");
        StdOut.println(queue);
    }
}
```

Queue

Operation	$T(n)$
`ResizingArrayQueue()`	1
`boolean isEmpty()`	1
`int size()`	1
`void enqueue(Item item)`	1 (amortized)
`Item peek()`	1
`Item dequeue()`	1 (amortized)
`Iterator<Item> iterator()`	1
`String toString()`	n

Stack

Stack

A `Stack` is an iterable collection that stores generic items in last-in-first-out (LIFO) order

Stack

A `Stack` is an iterable collection that stores generic items in last-in-first-out (LIFO) order

dsa.Stack<Item> extends java.lang.Iterable<Item>

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

Stack

Stack

Program: `Reverse.java`

Program: `Reverse.java`

- Standard input: a sequence of strings

Program: `Reverse.java`

- Standard input: a sequence of strings
- Standard output: the strings in reverse order

Program: `Reverse.java`

- Standard input: a sequence of strings
- Standard output: the strings in reverse order

```
>_ ~/workspace/dsaj/programs
$ _
```

Program: `Reverse.java`

- Standard input: a sequence of strings
- Standard output: the strings in reverse order

```
>_ ~/workspace/dsaj/programs
$ java Reverse
```

Program: `Reverse.java`

- Standard input: a sequence of strings
- Standard output: the strings in reverse order

```
>_ ~/workspace/dsaj/programs
$ java Reverse
-
```

Program: `Reverse.java`

- Standard input: a sequence of strings
- Standard output: the strings in reverse order

```
>_ ~/workspace/dsaj/programs

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

Program: `Reverse.java`

- Standard input: a sequence of strings
- Standard output: the strings in reverse order

```
>_ ~/workspace/dsaj/programs

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

Program: `Reverse.java`

- Standard input: a sequence of strings
- Standard output: the strings in reverse order

```
>_ ~/workspace/dsaj/programs

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

Program: `Reverse.java`

- Standard input: a sequence of strings
- Standard output: the strings in reverse order

```
>_ ~/workspace/dsaj/programs

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

Stack

Stack

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();
    }
}
```

Stack

Stack

dsa.LinkedStack<Item> implements dsa.Stack<Item>	
`LinkedStack()`	constructs an empty stack
`String toString()`	returns a string representation of this stack

Stack

dsa.LinkedStack<Item> implements dsa.Stack<Item>	
`LinkedStack()`	constructs an empty stack
`String toString()`	returns a string representation of this stack

Instance variables:

Stack

dsa.LinkedStack<Item> implements dsa.Stack<Item>	
`LinkedStack()`	constructs an empty stack
`String toString()`	returns a string representation of this stack

Instance variables:

- Reference to the top of the stack: `Node first`

Stack

dsa.LinkedStack<Item> implements dsa.Stack<Item>	
`LinkedStack()`	constructs an empty stack
`String toString()`	returns a string representation of this stack

Instance variables:

- Reference to the top of the stack: `Node first`

- Number of items in the stack: `int n`

Stack

Stack

LinkedStack.java

```
package dsa;

import java.util.Iterator;
import java.util.NoSuchElementException;

import stdlib.StdIn;
import stdlib.StdOut;

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

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

    public boolean isEmpty() {
        return n == 0;
    }

    public int size() {
        return n;
    }

    public void push(Item item) {
        Node oldfirst = first;
        first = new Node();
        first.item = item;
        first.next = oldfirst;
        n++;
    }

    public Item peek() {
        if (isEmpty()) {
```

Stack

LinkedStack.java

```
            throw new NoSuchElementException("Stack is empty");
        }
        return first.item;
    }

    public Item pop() {
        if (isEmpty()) {
            throw new NoSuchElementException("Stack is empty");
        }
        Item item = first.item;
        first = first.next;
        n--;
        return item;
    }

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

    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (Item item : this) {
            sb.append(item);
            sb.append(", ");
        }
        return n > 0 ? "[" + sb.substring(0, sb.length() - 2) + "]" : "[]";
    }

    private class Node {
        private Item item;
        private Node next;
    }

    private class ListIterator implements Iterator<Item> {
        private Node current;
```

Stack

LinkedStack.java

```
        public ListIterator() {
            current = first;
        }

        public boolean hasNext() {
            return current != null;
        }

        public Item next() {
            if (!hasNext()) {
                throw new NoSuchElementException("Iterator is empty");
            }
            Item item = current.item;
            current = current.next;
            return item;
        }
    }

    public static void main(String[] args) {
        LinkedStack<String> stack = new LinkedStack<String>();
        while (!StdIn.isEmpty()) {
            String item = StdIn.readString();
            if (!item.equals("-")) {
                stack.push(item);
            } else if (!stack.isEmpty()) {
                StdOut.print(stack.pop() + " ");
            }
        }
        StdOut.println();
        StdOut.println(stack.size() + " items left in the stack");
        StdOut.println(stack);
    }
}
```

Stack

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

Stack

Stack

dsa.ResizingArrayStack<Item> implements dsa.Stack<Item>	
`ResizingArrayStack()`	constructs an empty stack
`String toString()`	returns a string representation of this stack

Stack

dsa.ResizingArrayStack<Item> implements dsa.Stack<Item>	
`ResizingArrayStack()`	constructs an empty stack
`String toString()`	returns a string representation of this stack

Instance variables:

Stack

dsa.ResizingArrayStack<Item> implements dsa.Stack<Item>	
`ResizingArrayStack()`	constructs an empty stack
`String toString()`	returns a string representation of this stack

Instance variables:

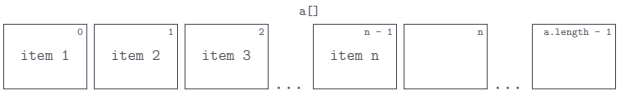
- Array of items in the stack: `Item a[]`

Stack

dsa.ResizingArrayStack<Item> implements dsa.Stack<Item>	
`ResizingArrayStack()`	constructs an empty stack
`String toString()`	returns a string representation of this stack

Instance variables:

- Array of items in the stack: `Item a[]`



- Number of items in the stack: `int n`

Stack

Stack

ResizingArrayStack.java

```java
package dsa;

import java.util.Iterator;
import java.util.NoSuchElementException;

import stdlib.StdIn;
import stdlib.StdOut;

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

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

    public boolean isEmpty() {
        return n == 0;
    }

    public int size() {
        return n;
    }

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

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

Stack

ResizingArrayStack.java

```
        }
        return a[n - 1];
    }

    public Item pop() {
        if (isEmpty()) {
            throw new NoSuchElementException("Stack is empty");
        }
        Item item = a[n - 1];
        a[n - 1] = null;
        n--;
        if (n > 0 && n == a.length / 4) {
            resize(a.length / 2);
        }
        return item;
    }

    public Iterator<Item> iterator() {
        return new ReverseArrayIterator();
    }

    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (Item item : this) {
            sb.append(item);
            sb.append(", ");
        }
        return n > 0 ? "[" + sb.substring(0, sb.length() - 2) + "]" : "[]";
    }

    private void resize(int capacity) {
        Item[] temp = (Item[]) new Object[capacity];
        for (int i = 0; i < n; i++) {
            temp[i] = a[i];
        }
```

ResizingArrayStack.java

```
        a = temp;
    }

    private class ReverseArrayIterator implements Iterator<Item> {
        private int i;

        public ReverseArrayIterator() {
            i = n - 1;
        }

        public boolean hasNext() {
            return i >= 0;
        }

        public Item next() {
            if (!hasNext()) {
                throw new NoSuchElementException("Iterator is empty");
            }
            return a[i--];
        }
    }

    public static void main(String[] args) {
        ResizingArrayStack<String> stack = new ResizingArrayStack<String>();
        while (!StdIn.isEmpty()) {
            String item = StdIn.readString();
            if (!item.equals("-")) {
                stack.push(item);
            } else if (!stack.isEmpty()) {
                StdOut.print(stack.pop() + " ");
            }
        }
        StdOut.println();
        StdOut.println(stack.size() + " items left in the stack");
        StdOut.println(stack);
```

Stack

ResizingArrayStack.java

```
    }
}
```

Stack

Operation	$T(n)$
`ResizingArrayStack()`	1
`boolean isEmpty()`	1
`int size()`	1
`void push(Item item)`	1 (amortized)
`Item peek()`	1
`Item pop()`	1 (amortized)
`Iterator<Item> iterator()`	1
`String toString()`	n