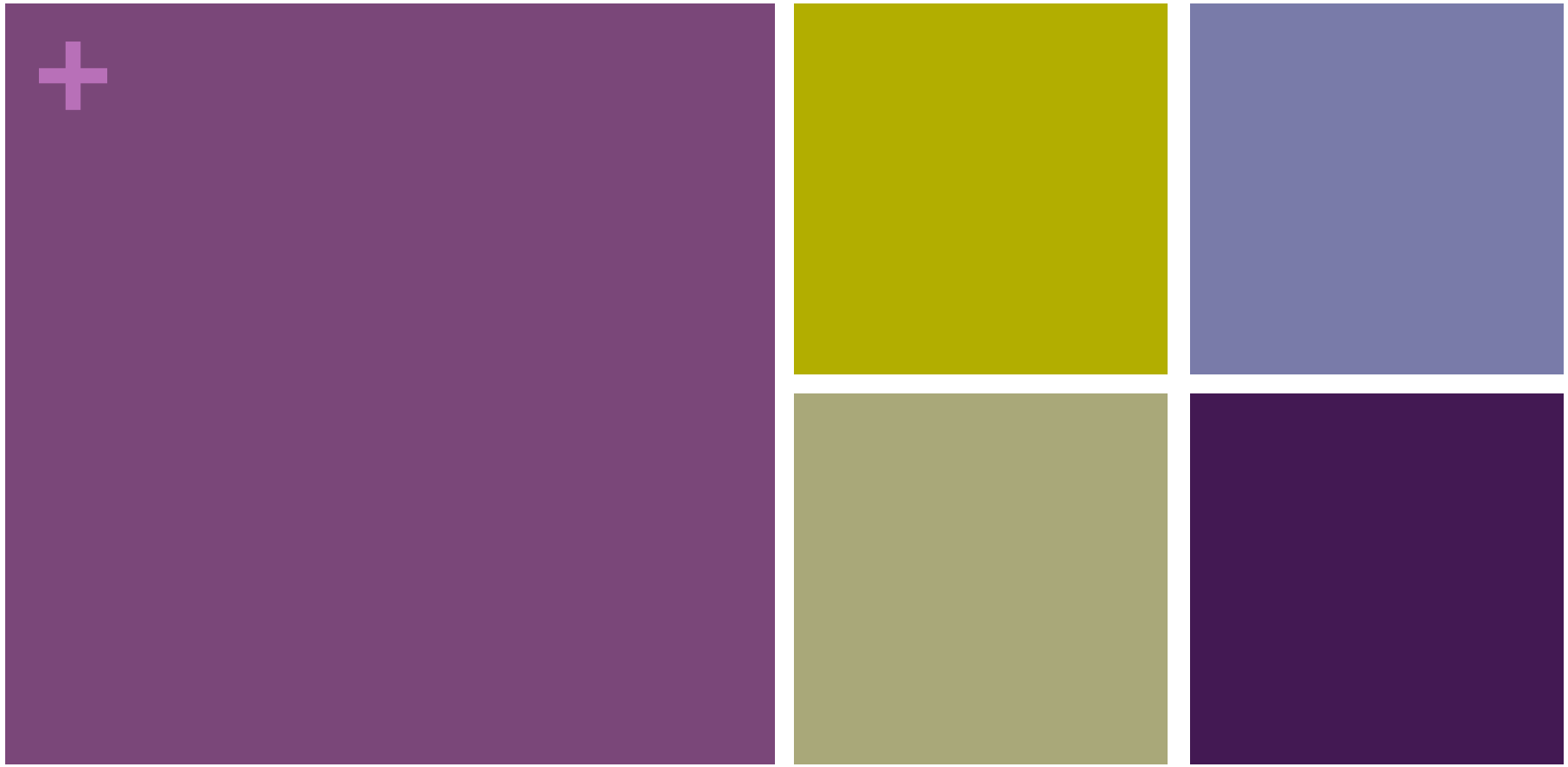
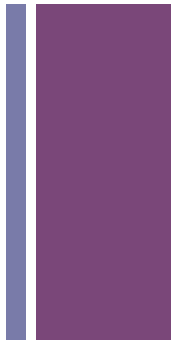


Stacks and Queues

Discrete Event simulation.

Stack



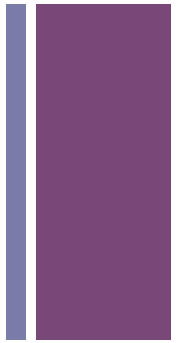
- Four Standard operations
  - push (add to top of stack)
  - pop (remove from top of stack)
  - peek* (get from top of stack)
  - empty* (test whether there are elements in the stack.)

  - * - optional

Java Stack Class

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

Ideal Queue



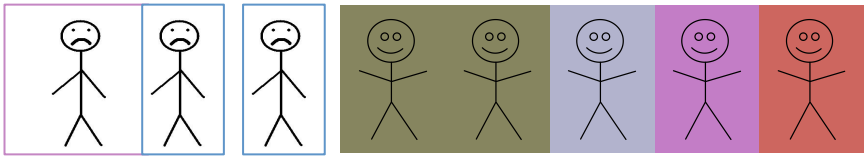
- Three operations:
  - enqueue (add (last))
  - dequeue (remove (first))
  - peek (get (first))
- Special Queues
  - Fixed Capacity Queue (limits the number of items)
  - Priority Queue (orders removal by priority)

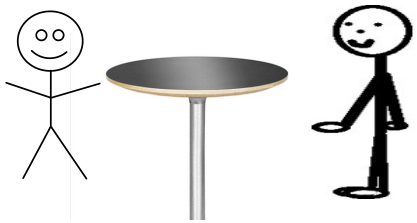
Java Queue Interface

	Throws exception	*Returns special value*
Insert	`add(e)`	`offer(e)`
Remove	`remove()`	`poll()`
Examine	`element()`	`peek()`

Discrete Event Simulation

- Single Queue, single server

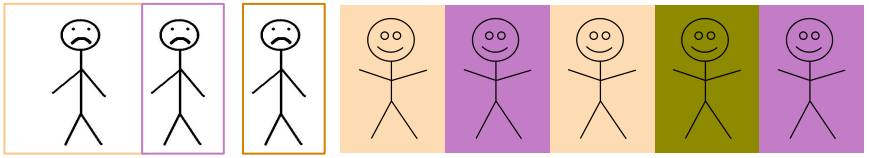


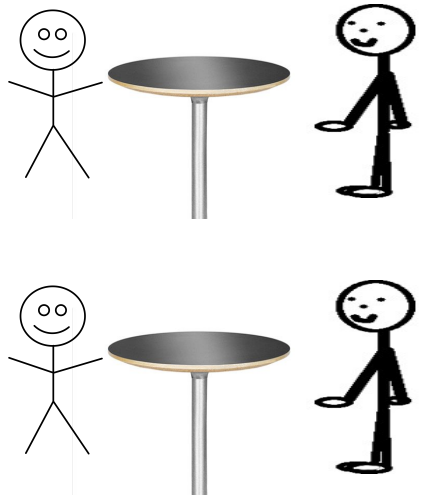


- Single Queue, multiple servers
- Multiple Queue, multiple servers

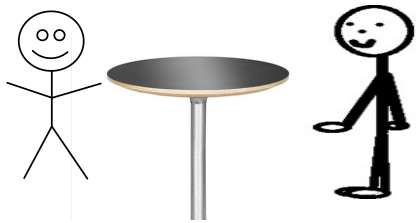
Discrete Event Simulation

- Single Queue, single server
- Single Queue, multiple servers



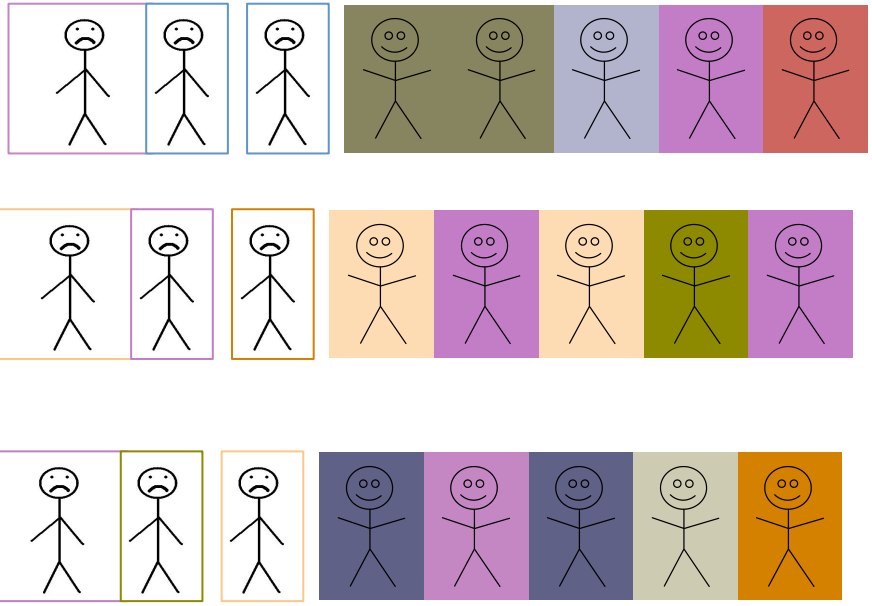


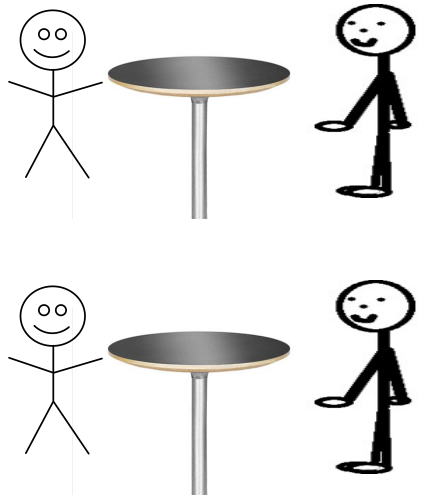
- Multiple Queue, multiple servers

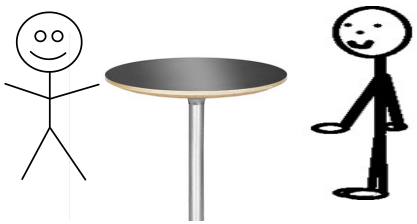


Discrete Event Simulation

- Single Queue, single server
- Single Queue, multiple servers
- Multiple Queue, multiple servers







Arrival process

- How customers arrive: What is inter-arrival time?
  - e.g. between 1-3 min
- Service Mechanism: How long will service take?
  - e.g. 0.5-2.0 min
- Queue Characteristics: FIFO

Example Data

Customer	Inter-arrival Time	Service Time
C1	1.9	1.7 min
C2	1.3	1.8
C3	1.1	1.5
C4	1.0	0.9

Queue Simulation

T	Arrival	Queue	Server	Depart
0		[]	Idle	

Example Data

Customer	Inter-arrival Time	Service Time
C1	1.9	1.7 min
C2	1.3	1.8
C3	1.1	1.5
C4	1.0	0.9

Queue Simulation

T	Arrival	Queue	Server	Depart
0		[]	Idle	
1.9	C1	[]	C1	

Example Data

Customer	Inter-arrival Time	Service Time
C1	1.9	1.7 min
C2	1.3	1.8
C3	1.1	1.5
C4	1.0	0.9

Queue Simulation

T	Arrival	Queue	Server	Depart
0		[]	Idle	
1.9	C1	[]	C1	
3.2	C2	[C2]	C1	

Example Data

Customer	Inter-arrival Time	Service Time
C1	1.9	1.7 min
C2	1.3	1.8
C3	1.1	1.5
C4	1.0	0.9

Queue Simulation

T	Arrival	Queue	Server	Depart
0		[]	Idle	
1.9	C1	[]	C1	
3.2	C2	[C2]	C1	
3.6		[]	C2	C1

Example Data

Customer	Inter-arrival Time	Service Time
C1	1.9	1.7 min
C2	1.3	1.8
C3	1.1	1.5
C4	1.0	0.9

Queue Simulation

T	Arrival	Queue	Server	Depart
0		[]	Idle	
1.9	C1	[]	C1	
3.2	C2	[C2]	C1	
3.6		[]	C2	C1
4.3	C3	[C3]	C2	

Example Data

Customer	Inter-arrival Time	Service Time
C1	1.9	1.7 min
C2	1.3	1.8
C3	1.1	1.5
C4	1.0	0.9

Queue Simulation

T	Arrival	Queue	Server	Depart
0		[]	Idle	
1.9	C1	[]	C1	
3.2	C2	[C2]	C1	
3.6		[]	C2	C1
4.3	C3	[C3]	C2	
5.3	C4	[C4, C3]	C2	

Example Data

Customer	Inter-arrival Time	Service Time
C1	1.9	1.7 min
C2	1.3	1.8
C3	1.1	1.5
C4	1.0	0.9

Queue Simulation

T	Arrival	Queue	Server	Depart
0		[]	Idle	
1.9	C1	[]	C1	
3.2	C2	[C2]	C1	
3.6		[]	C2	C1
4.3	C3	[C3]	C2	
5.3	C4	[C4, C3]	C2	
5.4		[C4]	C3	C2

High Level Overview

Data Structures						
Structure	Add Cost	Access by value cost	Access by index cost	Sequential (Access by index)	Sorted	Built in Sort
Array	n/a	O(n)	O(1)	yes	no	Arrays.sort()
ArrayList	O(n)	O(n)	O(1)	yes	no	Collections.sort()
LinkedList	O(1)	O(n)	O(n)*	yes	no	Collections.sort()
Stack	O(1)	n/a	n/a	yes	no	Collections.sort() *

* - Note: for a LinkedList, the index is implicit by counting, either up or down, from head, where the head node is either at index 0, when counting up, or at index size – 1, when counting down.

High Level Overview

Collection Interfaces/Subclasses				
Type	Sequential	Unique Values	Sorted	Extends or Implements
Set	No	Yes	No	Collection
SortedSet	No	Yes	Yes	Set
List	Yes	No	No	Collection
Queue	No	No	No	Collection
Deque	No	No	No	Queue
ArrayList	Yes	No	No	AbstractList
LinkedList	Yes	No	No	AbstractSequentalList
Stack	No	No	No	Vector

All java.util.Collection implementations are Iterable!
All java.util.List implementations have a listIterator method

High Level Overview

- Iterator
  (required by Collection interface)
  - one direction
  - iterate
  - remove is optional
- List Iterator
  (required by List interface)
  - bi-directional
  - iterate
  - add
  - remove
  - start at any index

Questions:

- **Which iterator makes sense for a singly linked list?**
- **What, if any is a drawback of a ListIterator that traverses a singly linked list?**
- **Is one direction more efficient then the other?**