# CSIS 3103

Ch 4: Deques
Queues and Simulation

## Deque Interface

- Short for double-ended queue
- Allows insertions and removals from both ends
- The Java Collections Framework provides two implementations of the `Deque` interface
  - `ArrayDeque`
  - `LinkedList`
- `ArrayDeque` uses a resizable circular array

| Method | Behavior |
|---|---|
| `boolean offerFirst(E item)` | Inserts `item` at the front of the deque. Returns **`true`** if successful; returns **`false`** if the item could not be inserted. |
| `boolean offerLast(E item)` | Inserts `item` at the rear of the deque. Returns **`true`** if successful; returns **`false`** if the item could not be inserted. |
| `void addFirst(E item)` | Inserts `item` at the front of the deque. Throws an exception if the item could not be inserted. |
| `void addLast(E item)` | Inserts `item` at the rear of the deque. Throws an exception if the item could not be inserted. |
| `E pollFirst()` | Removes the entry at the front of the deque and returns it; returns **`null`** if the deque is empty. |
| `E pollLast()` | Removes the entry at the rear of the deque and returns it; returns **`null`** if the deque is empty. |
| `E removeFirst()` | Removes the entry at the front of the deque and returns it if the deque is not empty. If the deque is empty, throws a `NoSuchElementException`. |
| `E removeLast()` | Removes the item at the rear of the deque and returns it. If the deque is empty, throws a `NoSuchElementException`. |
| `E peekFirst()` | Returns the entry at the front of the deque without removing it; returns **`null`** if the deque is empty. |
| `E peekLast()` | Returns the item at the rear of the deque without removing it; returns **`null`** if the deque is empty. |
| `E getFirst()` | Returns the entry at the front of the deque without removing it. If the deque is empty, throws a `NoSuchElementException`. |
| `E getLast()` | Returns the item at the rear of the deque without removing it. If the deque is empty, throws a `NoSuchElementException`. |
| `boolean removeFirstOccurrence(Object item)` | Removes the first occurrence of item in the deque. Returns **`true`** if the item was removed. |
| `boolean removeLastOccurrence(Object item)` | Removes the last occurrence of item in the deque. Returns **`true`** if the item was removed. |
| `Iterator<E> iterator()` | Returns an iterator to the elements of this deque in the proper sequence. |
| `Iterator<E> descendingIterator()` | Returns an iterator to the elements of this deque in reverse sequential order. |

## Deque Interface

| Deque Method | Deque d | Effect |
|---|---|---|
| `d.offerFirst('b')` | b | `'b'` inserted at front |
| `d.offerLast('y')` | by | `'y'` inserted at rear |
| `d.addLast('z')` | byz | `'z'` inserted at rear |
| `d.addFirst('a')` | abyz | `'a'` inserted at front |
| `d.peekFirst()` | abyz | Returns `'a'` |
| `d.peekLast()` | abyz | Returns `'z'` |
| `d.pollLast()` | aby | Removes `'z'` |
| `d.pollFirst()` | by | Removes `'a'` |

## Deque Interface

A deque can be used as a stack if elements are pushed and popped from the front of the deque (preferable to using the legacy `Stack` class)

| Stack Method | Equivalent Deque Method |
|---|---|
| `push(e)` | `addFirst(e)` |
| `pop()` | `removeFirst()` |
| `peek()` | `peekFirst()` |
| `empty()` | `isEmpty()` |

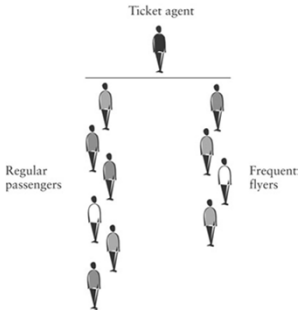
## Simulating Waiting Lines Using Queues

- Simulation is used to study the performance characteristics of a physical system by using a computer model of the system
  - Airline check-in counter for example
- A special branch of mathematics called queuing theory studies such problems

## Simulate a Strategy for Serving Airline Passengers

**Strategy:** Serve a certain number of consecutive frequent flyers between regular passengers.

Let this number vary and measure the effect on the time it takes to serve customers.

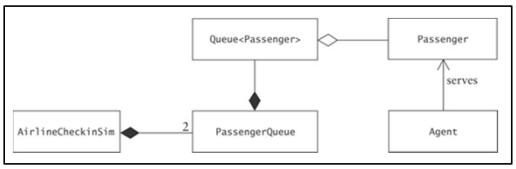



## Simulate Serving Airline Passengers

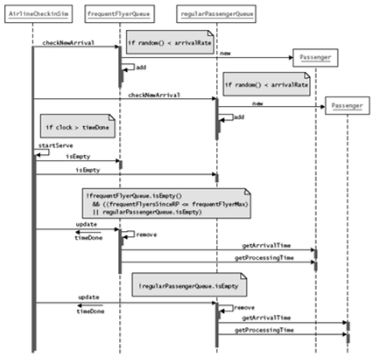
Maintain a "clock" that increases by one time unit until the simulation is finished. During each time interval, any of the following may occur:

1. New frequent flyer passenger arrives
2. New regular passenger arrives
3. Ticket agent finishes serving one passenger and begins another from the frequent flyers
4. Ticket agent finishes serving one passenger and begins another from the regular passengers

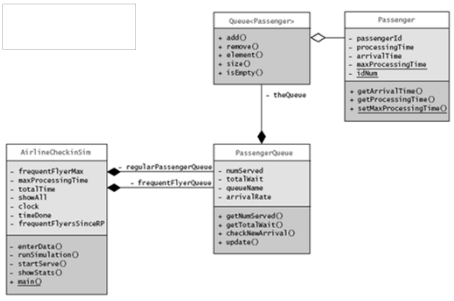
## Airline Passenger Simulation




## Airline Passenger Simulation




## Simulating Waiting Lines Using Queues




| Data Field | Attribute |
|---|---|
| `private PassengerQueue frequentFlyerQueue` | The queue of frequent flyers. |
| `private PassengerQueue regularPassengerQueue` | The queue of regular passengers. |
| `private int frequentFlyerMax` | The maximum number of frequent flyers to serve between regular passengers. |
| `private int maxProcessingTime` | The maximum time to serve a passenger. |
| `private int totalTime` | The total time to run the simulation. |
| `private boolean showAll` | A flag indicating whether to trace the simulation. |
| `private int clock` | The current clock time (initially zero). |
| `private int timeDone` | The time that the current passenger will be finished. |
| `private int frequentFlyersSinceRP` | The number of frequent flyers served since the last regular passenger. |
| **Method** | **Behavior** |
| `public static void main(String[] args)` | Starts the execution of the simulation by calling `enterData` and `runSimulation`. |
| `private void runSimulation()` | Controls the simulation. Executes the steps shown in Figure 4.15. |
| `private void enterData()` | Reads in the data for the simulation. |
| `private void startServe()` | Initiates service for a passenger. |
| `private void showStats()` | Displays the summary statistics. |

## Simulation Inputs

| Internal Variable | Attribute | Conversion |
|---|---|---|
| `frequentFlyerQueue.arrivalRate` | Expected number of frequent flyer arrivals per hour. | Divide input by 60 to obtain arrivals per minute. |
| `regularPassengerQueue.arrivalRate` | Expected number of regular passenger arrivals per hour. | Divide input by 60 to obtain arrivals per minute. |
| `maxProcessingTime` | Maximum service time in minutes. | None. |
| `totalTime` | Total simulation time in minutes. | None. |
| `showAll` | Flag. If **`true`**, display minute-by-minute trace of simulation. | Input beginning with `'Y'` or `'y'` will set this to **`true`**; other inputs will set it to **`false`**. |

| Data Field | Attribute |
|---|---|
| `private Queue<Passenger> theQueue` | The queue of passengers. |
| `private int numServed` | The number from this queue who were served. |
| `private int totalWait` | The total time spent waiting by passengers who were in this queue. |
| `private String queueName` | The name of this queue. |
| `private double arrivalRate` | The arrival rate for this queue. |
| **Method** | **Behavior** |
| `public PassengerQueue(String queueName)` | Constructs a new queue with the specified name. |
| `private void checkNewArrival(int clock, boolean showAll)` | Checks whether there was a new arrival for this queue and, if so, inserts the passenger into the queue. |
| `private int update(int clock, boolean showAll)` | Updates the total waiting time and number of passengers served when a passenger from this queue is served. |
| `public int getTotalWait()` | Returns the total waiting time for passengers in this queue. |
| `public int getNumServed()` | Returns the number of passengers served from this queue. |

| Method | Behavior |
|---|---|
| `public Passenger(int arrivalTime)` | Constructs a new `passenger`, assigns it a unique ID and the specified arrival time. Computes a random processing time in the range 1 to `maxProcessingTime`. |
| `public int getArrivalTime()` | Returns the value of `arrivalTime`. |
| `public int getProcessingTime()` | Returns the value of `processingTime`. |
| `public static void setMaxProcessingTime( int maxProcessingTime)` | Sets the `maxProcessingTime` used to generate the random processing time. |