

Tutorial 9

November 7 & 8, 2005

Today's Topics

- Quiz 2
- Matrices
- Stacks
- Queues
- Q& A on 'Concepts you didn't get'

Quiz 2

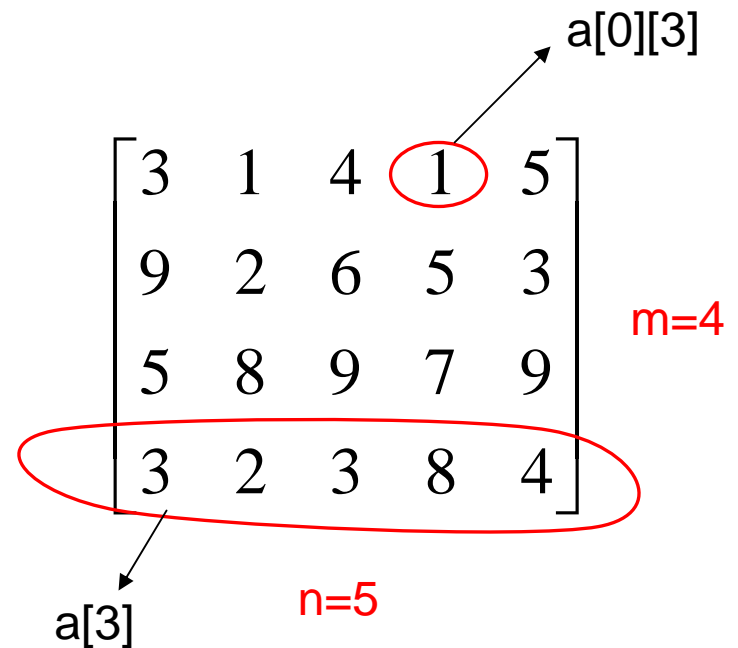
- Thursday, November 10th during class time (11 to 12:30pm)
- Topics Included
 - Lectures 12 to 24
 - PS 4 to PS 7
- Open Book, Open Notes, **NO** Laptops, **NO** Calculators, Cell Phones or other electronics
- Optional Quiz 2 Review on Tuesday, November 8th (7-10pm).

Concepts you don't get?

- Anything we need to go over again?
- Main Quiz topics:
 - Inheritance, Abstract classes
 - Interfaces
 - Swing: Containers, Components & Layout, Event Delegation, 2DAPI, MVC, Affine Transformations
 - Numerical methods: Root finding, Integration, Matrices, Linear Systems
 - Stacks & Queues

Matrices

- Represented as a 2-D array. For e.g.
int[][] a = new int[m][n]
m= no. of rows
n= no. of cols
- a[i][j] represents the element in the “i”th row and “j”th column
- a[i] represents the “i”th row



Traversing 2D Arrays

- Say `double[][] data = new double[m][n]`
- `data.length` = number of rows (m)
- `data[0].length` = number of columns (n)
- Typical operation accessing all elements of a **Matrix**

```
for(int i = 0; i < data.length; i++) {  
    for(int j = 0; j < data[0].length; j++) {  
        // Do something here  
    }  
}
```

Matrix class

- However 2-D arrays do not have default add and multiplication operations defined
- Define a Matrix class, which contains methods to add and multiply matrices as well as for any other matrix operation you need

Matrices & Linear Systems

- Matrices often used to represent a set of linear equations

$$\begin{aligned}
 a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \dots + a_{0,n-1}x_{n-1} &= b_0 \\
 a_{10}x_0 + a_{11}x_1 + a_{12}x_2 + \dots + a_{1,n-1}x_{n-1} &= b_1 \\
 \dots & \\
 a_{m-1,0}x_0 + a_{m-1,1}x_1 + a_{m-1,2}x_2 + \dots + a_{m-1,n-1}x_{n-1} &= b_{m-1}
 \end{aligned}$$

- Matrix **A** and right-hand side *b* are known

$$\begin{array}{cccccc|c|c}
 a_{00} & a_{01} & a_{02} & a_{03} \dots & a_{0,n-1} & & x_0 & b_0 \\
 a_{10} & a_{11} & a_{12} & a_{13} \dots & a_{1,n-1} & & x_1 & b_1 \\
 a_{20} & a_{21} & a_{22} & a_{23} \dots & a_{2,n-1} & & x_2 & b_2 \\
 \dots & \dots & \dots & \dots \dots & \dots & & \dots & \dots \\
 a_{m-1,0} & a_{m-1,1} & a_{m-1,2} & a_{m-1,3} \dots & a_{m-1,n-1} & & x_{n-1} & b_{m-1}
 \end{array} =$$

(m rows x n cols)

(n x 1) = (m x 1)

$$\mathbf{Ax} = \mathbf{b}$$

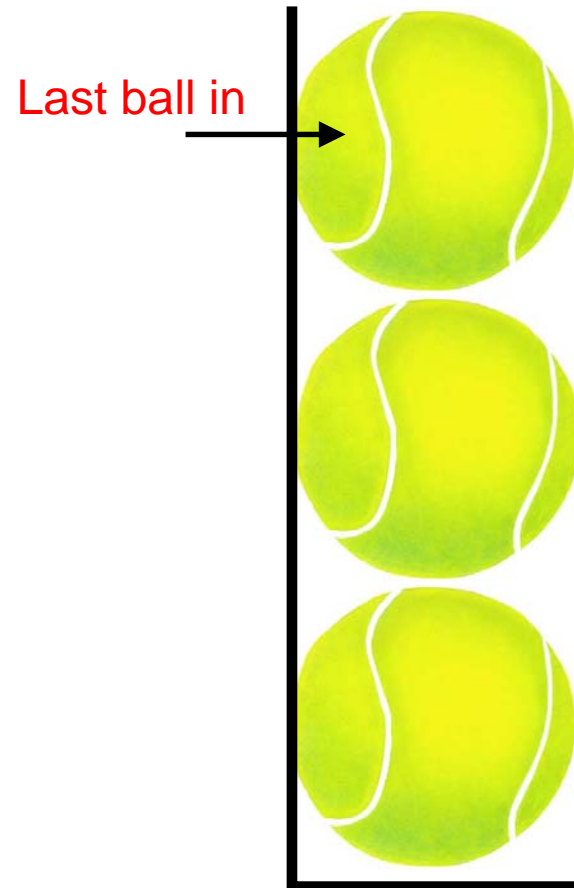
- n unknowns *x* related to each other by m equations

Exercise 1

- Write a Matrix class that creates a matrix of M rows and N columns. Then write a method that prints all its values in matrix form (i.e. as M rows and N columns).

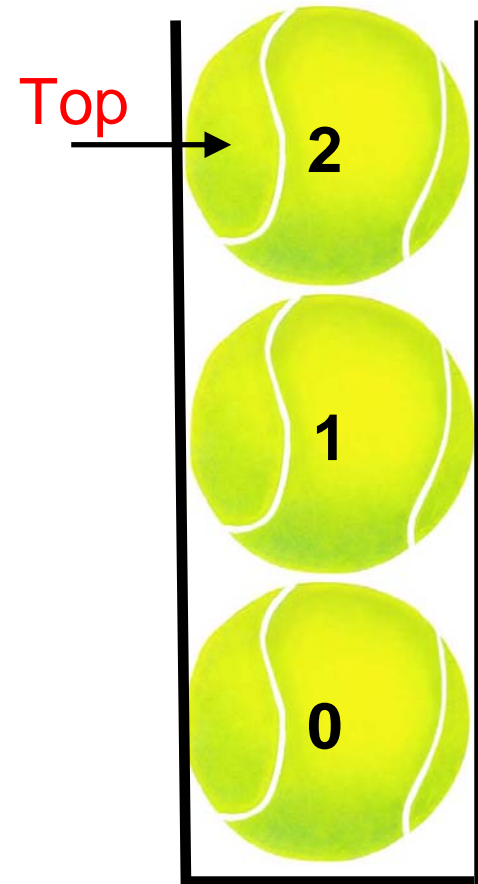
Stacks

- Consider a can of tennis balls
- The last ball you put in is now at the top
- It is also the first ball you can take out
- Model of a stack:
Last in First out (LIFO)
- Only 1 access point



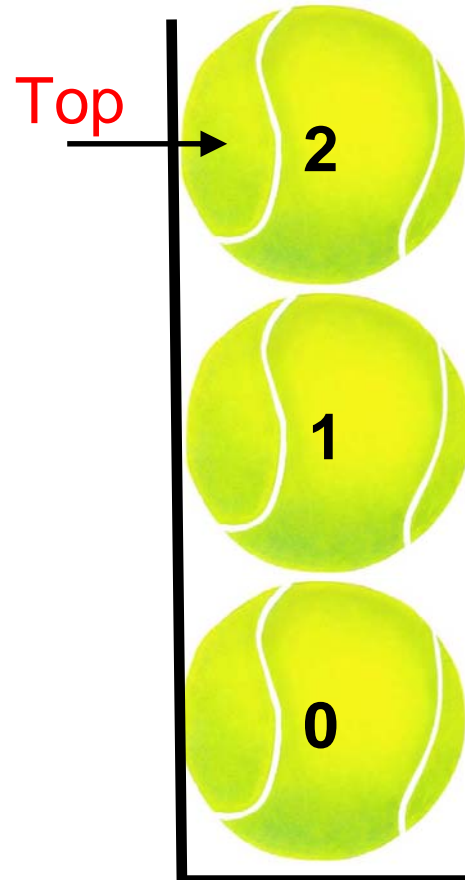
Stacks : Pop

- Pop: Removing from the top of the stack
- If you were to remove the balls one at a time, the only ball you can remove is the top one



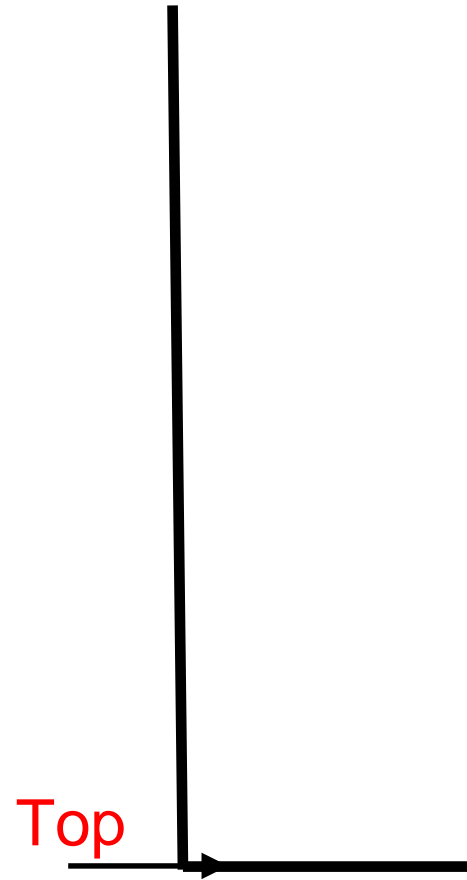
Stacks : Pop

- Pop: Removing from the top of the stack
- When you remove a ball, the top of the stack will move down to refer to the element below it
- Pop will return the object that is being removed



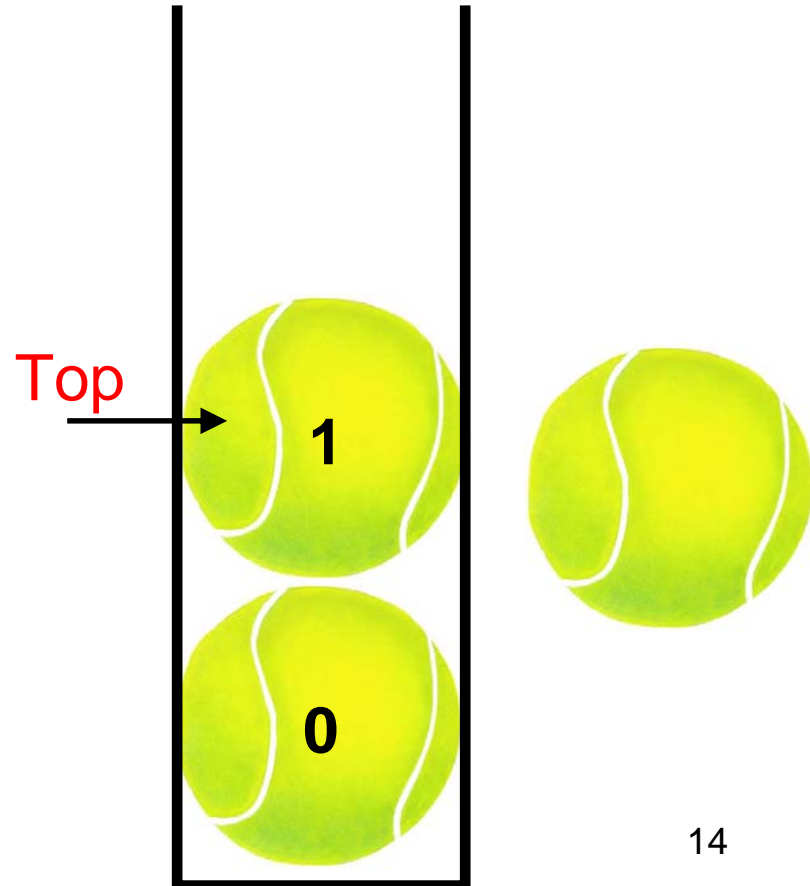
Stack: Empty stack

- What should happen when you try to remove an object from an empty stack?
- You must always check for an empty stack while trying to pop an object



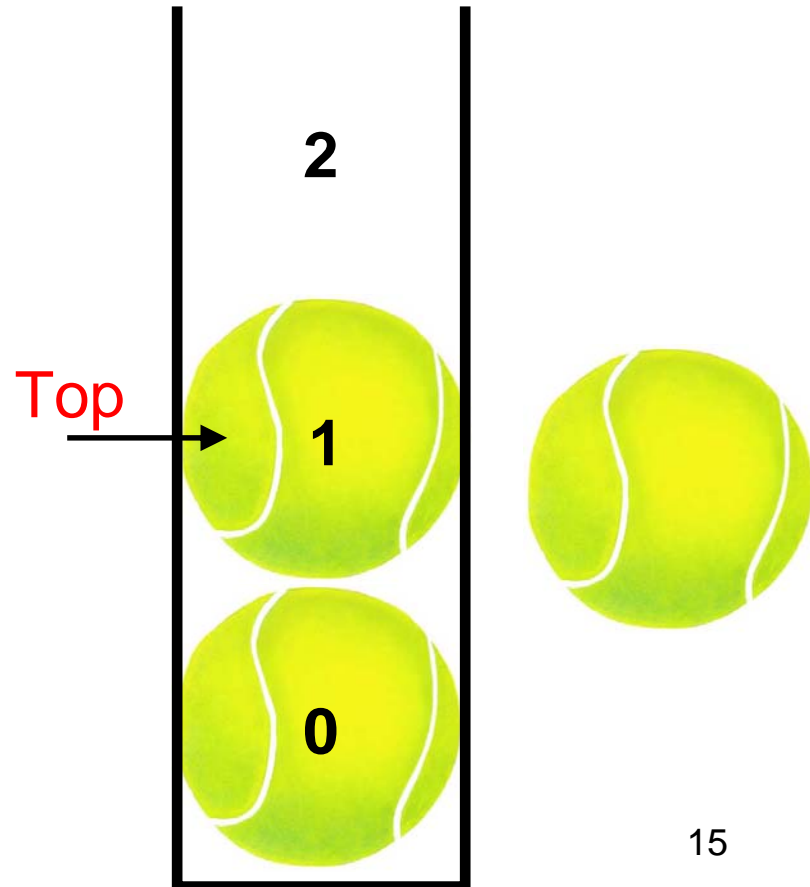
Stacks : Push

- Push: Adding to the top of the stack
- Think of the can of balls as an array with index of bottom ball = “0”
- To start with the capacity of the can is 3 balls & the top element is the 2nd ball which has an index = “1”
- You now want to add or “push” a 3rd ball into the can



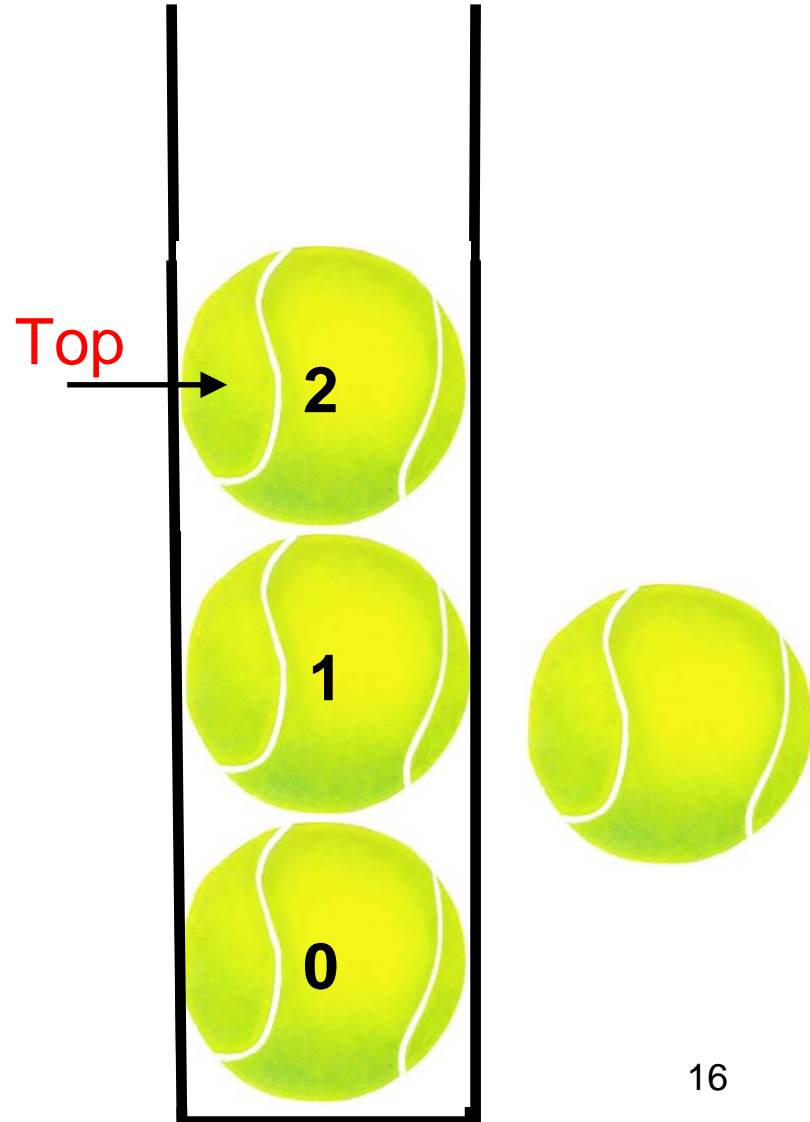
Stacks : Push

- Push: Adding to the top of the stack
- When you add the 3rd ball, the top of the stack now moves up to refer to it



Stack: Overflow

- What happens when you try to add a 4th ball?
- You can either grow the stack (suspend disbelief for a moment here)
- Or you can say that the stack is full
- Choose one of the above depending on your context
- Always check for overflow when you want to “push”



Standard Stack methods

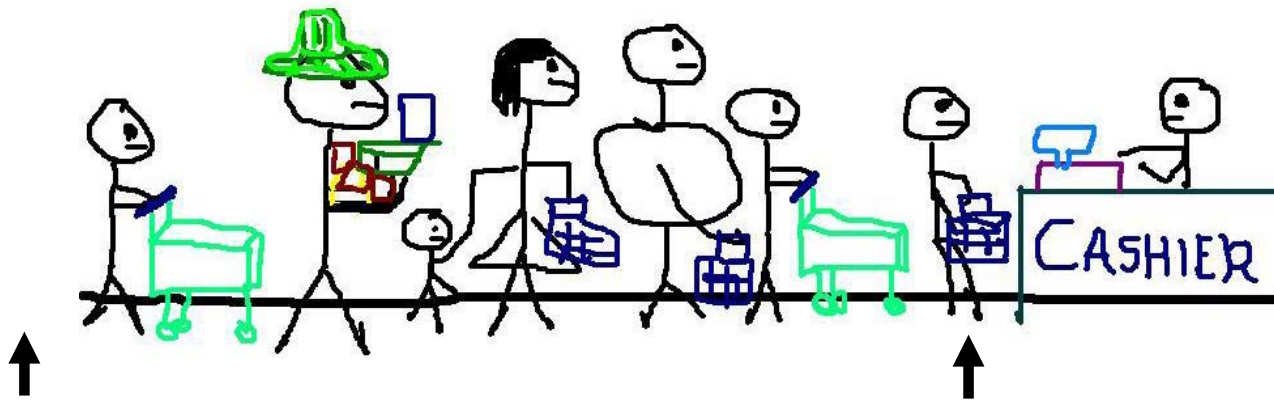
```
public interface GenericStack<E>
{
    public boolean isEmpty();
    public void push(E o);
    public E pop();
    public void clear();
}
```

Exercise 2

- From Quiz 2, Spring 2004
- Use stacks to check if a given string is a palindrome or not. You can use the `StringStack.java` class to do this.

Queues

- Consider a queue of people waiting in line to get billed.
- The person who gets billed first is the one at the front of the line
- Model of a queue: First in First Out (FIFO)
- 2 points of interest, front & end of the queue

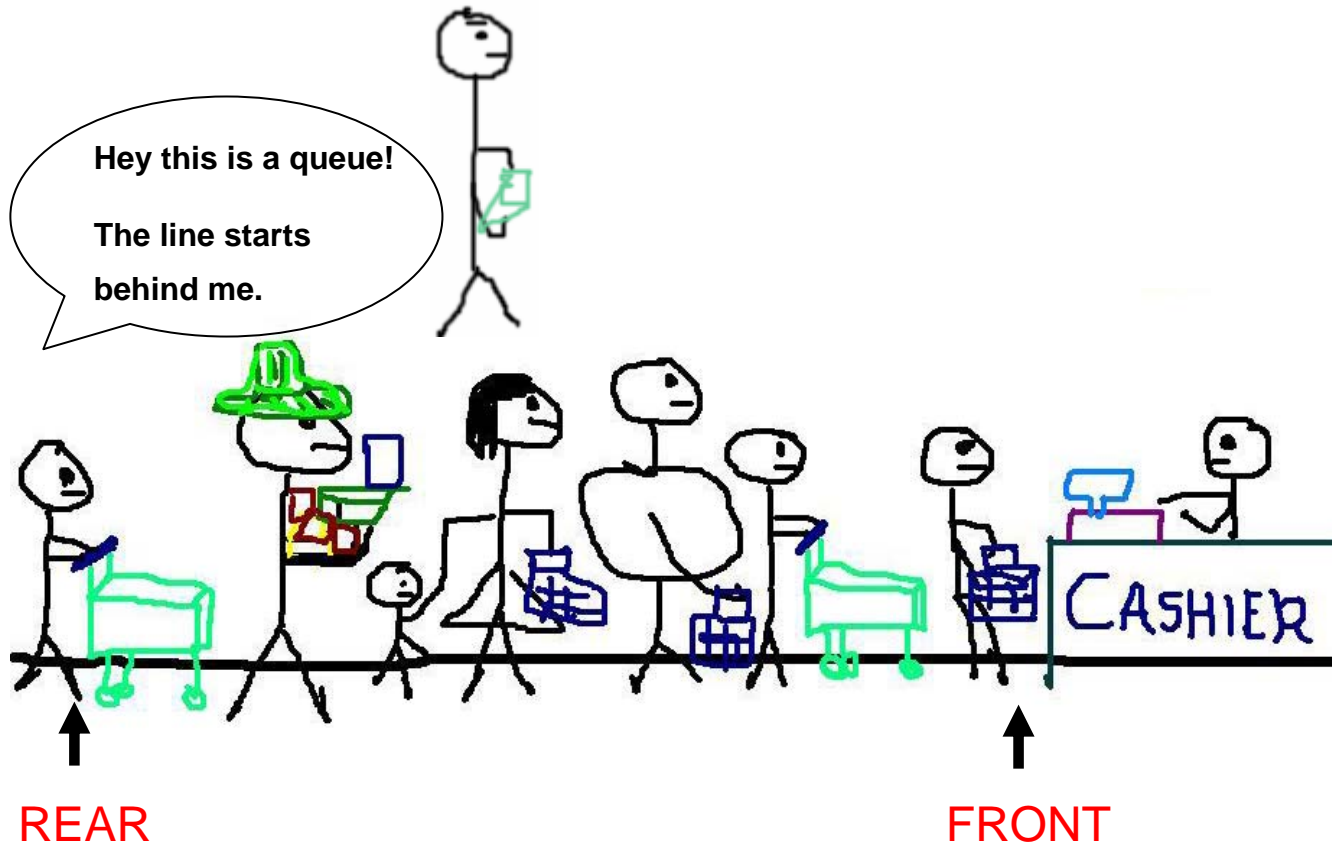


Next person will come here

Gets billed first

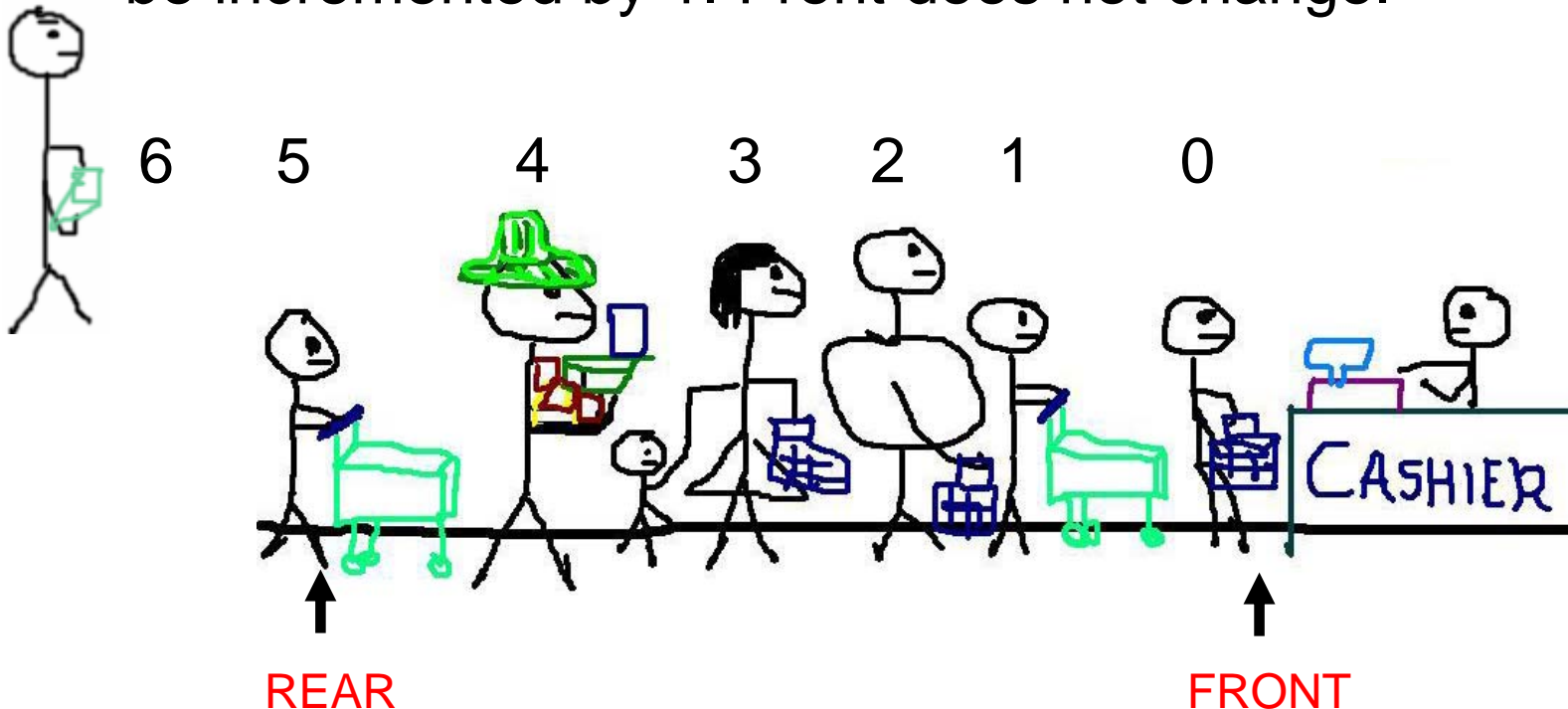
Queues: Enqueue (Add)

- You should only add to the end of the queue
- You cannot add to the front of the queue



Queues: Enqueue (Add)

- Think of the queue as an array with the index of the front element as 0.
- Rear is the index of the last element in the array
- When an element is added to the array, the rear should be incremented by 1. Front does not change.



Standard Queue Methods

```
public interface GenericQueue<E>
{
    public boolean isEmpty();
    public void enqueue(E o); //or "add"
    public E dequeue(); // or "remove"
    public void clear();
}
```

Overflow & Empty Queue

- You must always check for queue overflow while trying to enqueue
- You must check for an empty queue while trying to dequeue

Stacks vs. Queues

- What data structure (stack or queue) should be used to represent the following?
 1. A pile of books
 2. Managing events in java
 3. Print jobs
 4. Office hour lines
 5. People getting into an elevator