CS206 Data Structures

Arrays and Linked Lists

Sung-eui Yoon (윤성의)

Department of Computer Science
KAIST

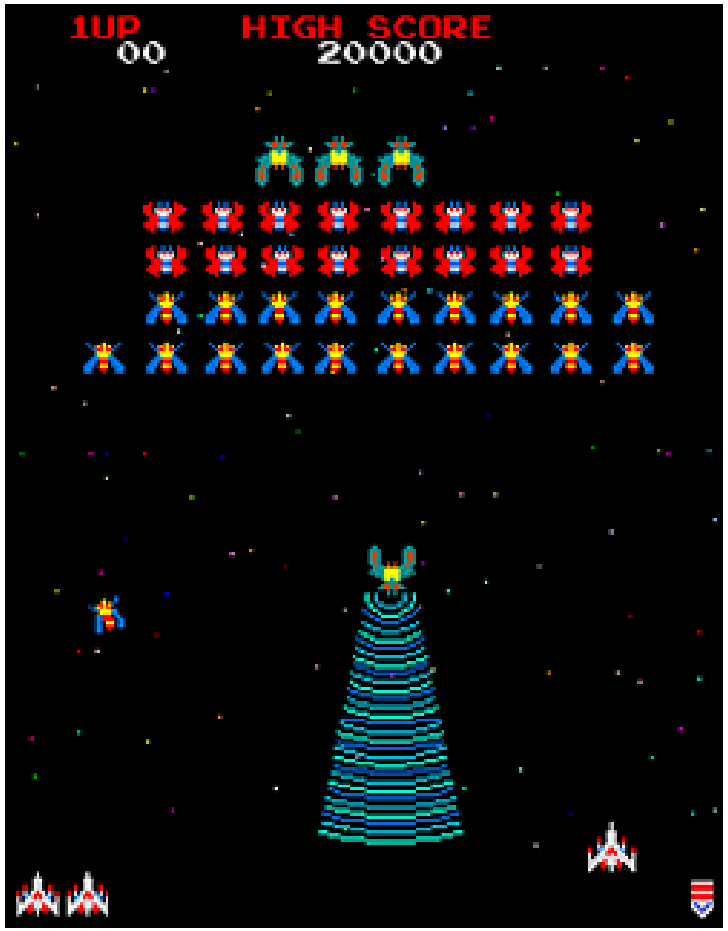
http://sglab.kaist.ac.kr/~sungeui

Class Objectives

- ☐ Understand representations and operations of arrays and singly linked lists
  - How to represent them
  - How to insert, remove, or sort them

Game High Score Entry

- ☐ Want to store high score entries
- ☐ An entry should contain:
  - An integer representing the score
  - A string representing the name of the person



Game High Score Entry

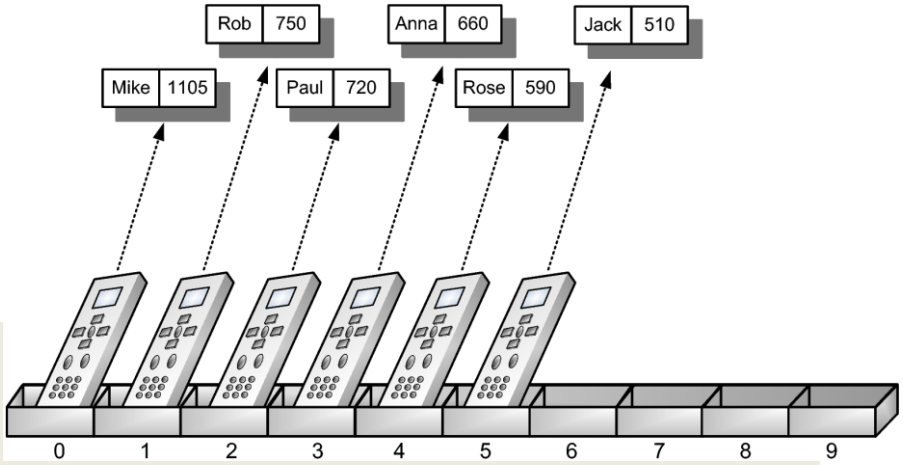
☐ Want to store high score entries; An entry should contain

- An integer representing the score
- A string representing the name of the person

```
public class GameEntry      // Define a class with a class modifier “public”
{
    protected String name; // name of the person earning this score
    protected int score; // the score value

    /** Constructor to create a game entry */
    public GameEntry(String n, int s)
    {
        name = n;
        score = s;
    }

    /** Retrieves the name field */
    public String getName()
    {
        return name;
    }

    /** Retrieves the score field */
    public int getScore()
    {
        return score;
    }

    /** Returns a string representation of this entry */
    public String toString()
    {
        return "(" + name + ", " + score + ")";
    }
}
```

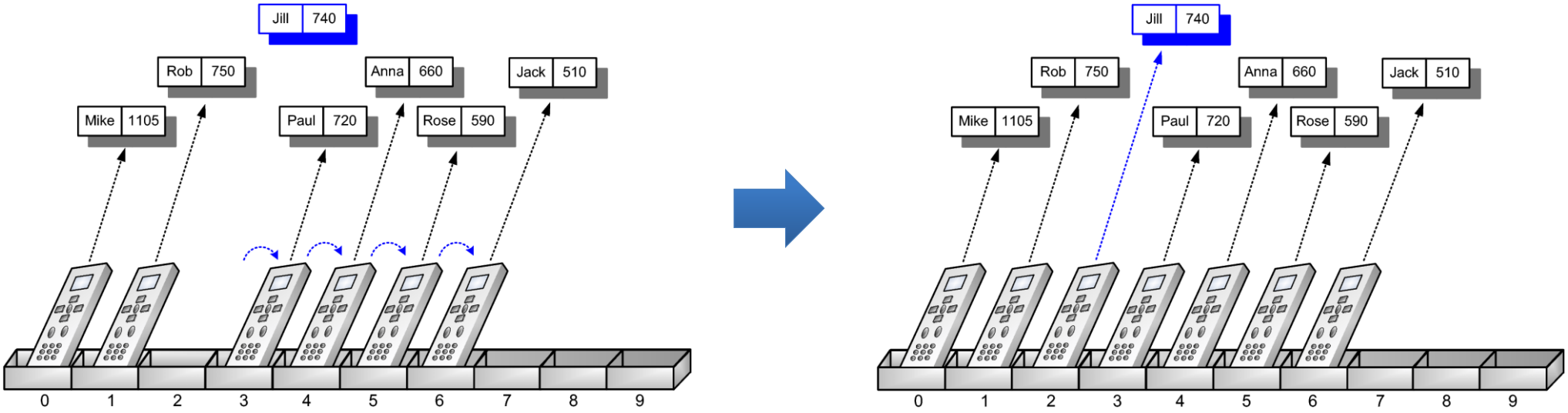
Storing High Scores in Array

- Store high scores in array ordered by `score` values:



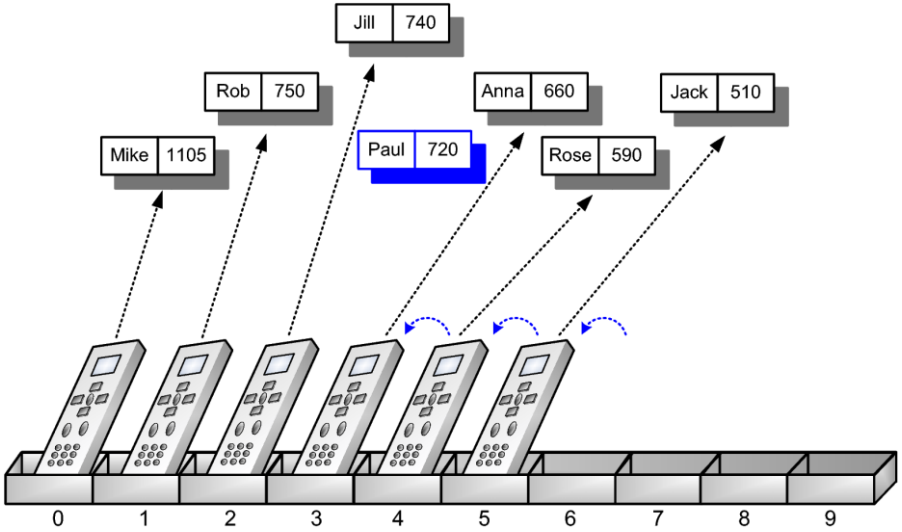

```
/** Class for storing high scores in an array in
   non-decreasing order. */
public class Scores
{
    public static final int maxEntries = 10; // number of high scores we keep
    protected int numEntries; // number of actual entries
    protected GameEntry[] entries; // array of game entries (names & scores)

    /** Default constructor */
    public Scores()
    {
        entries = new GameEntry[maxEntries];
        numEntries = 0;
    }

    /** Returns a string representation of the high scores list */
    public String toString()
    {
        String s = "[";
        for (int i = 0; i < numEntries; i++)
        {
            if (i > 0)
                s += ", "; // separate entries by commas
            s += entries[i];
        }
        return s + "]";
    }
    // ... methods for updating the set of high scores go here ...
}
```

Inserting a New High Score




```
/** Attempt to add a new score to the collection (if it is high enough) */
    public void add(GameEntry e)
    {
        int newScore = e.getScore();
        // is the new entry e really a high score?
        if (numEntries == maxEntries)
        { // the array is full
            if (newScore <= entries[numEntries - 1].getScore())
                return; // the new entry, e, is not a high score in this case
        }
        else
            // the array is not full
            numEntries++;

        // Locate the place that the new (high score) entry e belongs
        int i = numEntries - 1;
        for (; (i >= 1) && (newScore > entries[i - 1].getScore()); i--)
            entries[i] = entries[i - 1]; // move entry i one to the right
        entries[i] = e; // add the new score to entries
     }
```

Removing a High Score




```
/** Remove and return the high score at index i */
    public GameEntry remove(int i) throws IndexOutOfBoundsException
    {
        if ((i < 0) || (i >= numEntries))
            throw new IndexOutOfBoundsException("Invalid index: " + i);
        GameEntry temp = entries[i]; // temporarily save the object to be removed
        for (int j = i; j < numEntries - 1; j++)
            // count up from i (not down)
            entries[j] = entries[j + 1]; // move one cell to the left
        entries[numEntries - 1] = null; // null out the old last score
        numEntries--;
        return temp; // return the removed object
    }
```

Sorting an Array: Insertion Sort

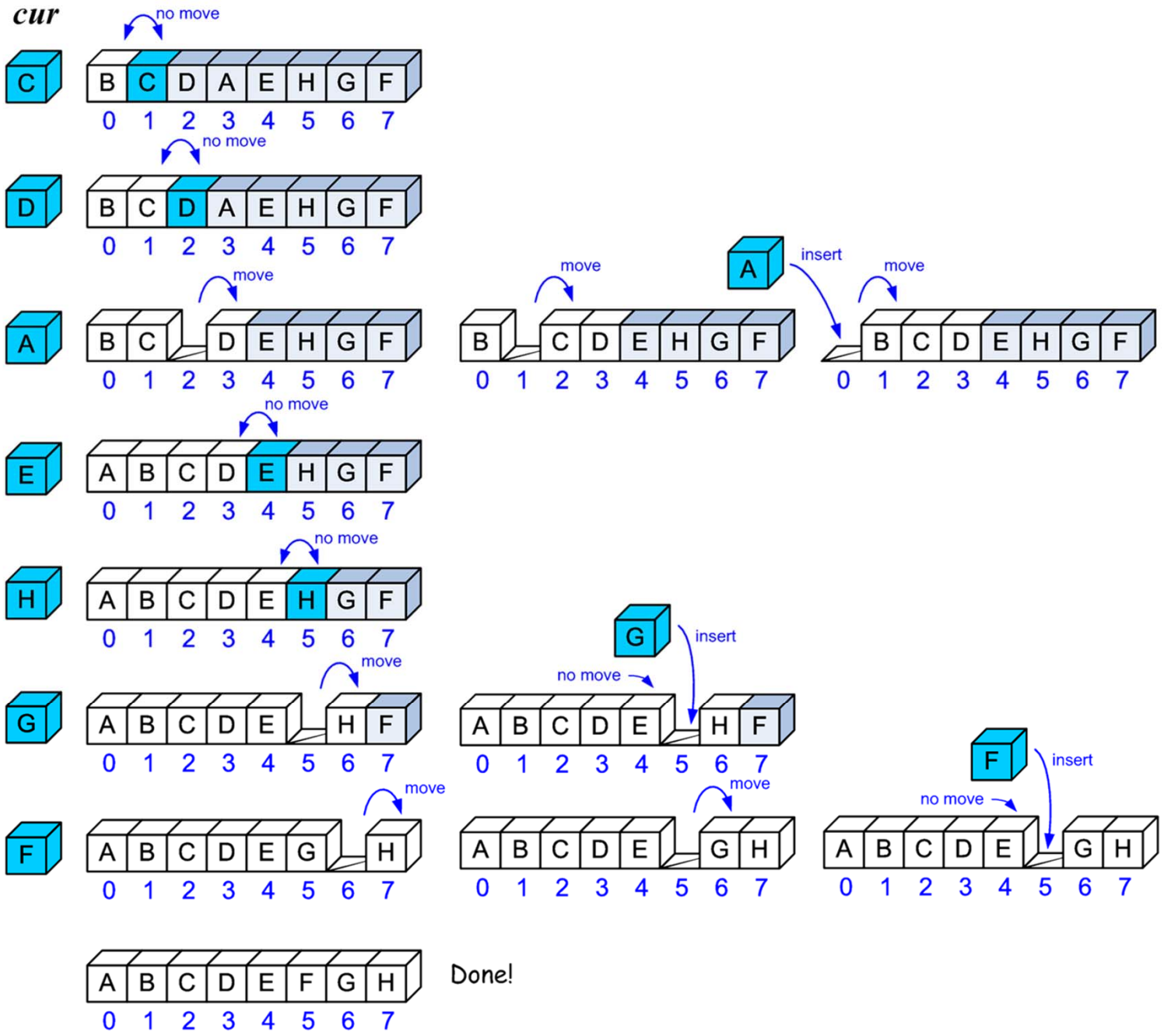
- Sort: start with an out-of-order array of objects and put them in order

- High-level description:

Algorithm InsertionSort(A)
  Input: An array A of n comparable elements
  Output: The array A with elements rearranged in non-decreasing order
  for $i \leftarrow 1$ to $n-1$ **do**
    Insert $A[i]$ at its proper location in $A[0], A[1], \ldots, A[i-1]$
  end for

  - Some details are missing!

Insertion Sort



Insertion Sort

☐ The for loop gets now “nested”

Algorithm InsertionSort(A)

Input: An array A of n comparable elements

Output: The array A with elements rearranged in non-decreasing order

```
for i ← 1 to n − 1 do
    // Insert A[i] at its proper location in A[0], A[1], . . . , A[i − 1]
    cur ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > cur do
        A[j + 1] ← A[j]
        j ← j − 1
    end while
    A[j + 1] ← cur // cur is now in the right place
end for
```

Java Code for Insertion Sort

- We are now ready to implement the Insertion Sort algorithm in Java!

```
/** Insertion sort of an array of characters into non-decreasing order */
    public static void insertionSort(char[] a)
    {
        int n = a.length;
        for (int i = 1; i < n; i++)
        { // index from the second character in a
            char cur = a[i]; // the current character to be inserted
            int j = i - 1; // start comparing with cell left of i
            while ((j >= 0) && (a[j] > cur))
                // while a[j] is out of order with cur
                a[j + 1] = a[j--]; // move a[j] right and decrement j
            a[j + 1] = cur; // this is the proper place for cur
        }
    }
```

JUnit Testing

☐ Sorting an array is frequent task in programming

- Provided as built-in method in Java (`java.util.Arrays`)

```
public void testSort()
{
    int num[] = new int[10];
    Random rand = new Random(); // a pseudo-random number generator
    rand.setSeed(System.currentTimeMillis()); // use current time as a seed
    // fill the num array with pseudo-random numbers from 0 to 99, inclusive
    for (int i = 0; i < num.length; i++)
        num[i] = rand.nextInt(100); // the next pseudo-random number
    int[] old = (int[]) num.clone(); // cloning the num array

    System.out.println("arrays equal before sort: "
            + Arrays.equals(old, num));
    Arrays.sort(num); // sorting the num array (old is unchanged)
    System.out.println("arrays equal after sort: "
            + Arrays.equals(old, num));
    System.out.println("old = " + Arrays.toString(old));
    System.out.println("num = " + Arrays.toString(num));

    for (int i = 0; i < num.length - 1; i++)
        assertTrue(num[i] < num[i + 1]);
}
```

Two-Dimensional Arrays

☐ Uses

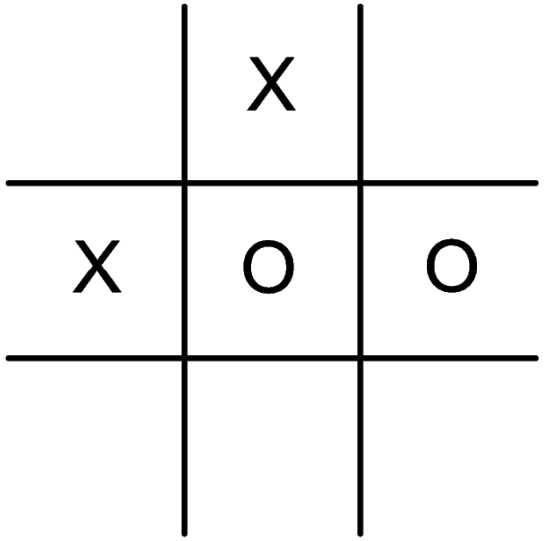
- Positional games using 2D board
- Matrix

	0	1	2	3	4	5	6	7	8	9
0	22	18	709	5	33	10	4	56	82	440
1	45	32	830	120	750	660	13	77	20	105
2	4	880	45	66	61	28	650	7	510	67
3	940	12	36	3	20	100	306	590	0	500
4	50	65	42	49	88	25	70	126	83	288
5	398	233	5	83	59	232	49	8	365	90
6	33	58	632	87	94	5	59	204	120	829
7	62	394	3	4	102	140	183	390	16	26

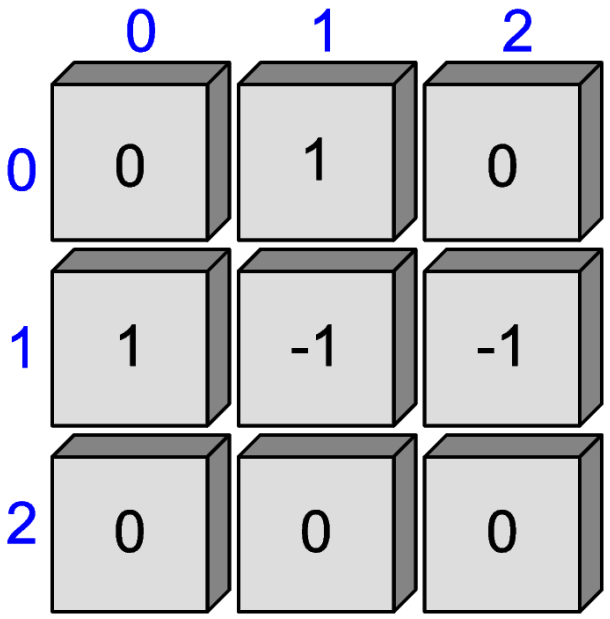
☐ Syntax

- int [][] Y = new int [8][10];
- i = Y.length; // what's the value?
- j = Y[4].length; // what's the value?
- Y[i][i+1] = Y[i][i] + 3;

Tic-Tac-Toe



playing board



board array

☐ Use 2D array “board” encoding the game statues

- empty cell (0), X (1), and O (-1)
- How do we check the winning situation?

Java Code for Tic-Tac-Toe

```
/** Simulation of a Tic-Tac-Toe game (does not do strategy). */
public class TicTacToe
{
    protected static final int X = 1, O = -1; // players
    protected static final int EMPTY = 0; // empty cell
    protected int board[][] = new int[3][3]; // game board
    protected int player; // current player

    /** Constructor */
    public TicTacToe() { clearBoard(); }

    /** Clears the board */
    public void clearBoard()
    {
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                board[i][j] = EMPTY; // every cell should be empty
        player = X; // the first player is 'X'
    }

    /** Puts an X or O mark at position i,j */
    public void putMark(int i, int j) throws IllegalArgumentException
    {
        if ((i < 0) || (i > 2) || (j < 0) || (j > 2))
            throw new IllegalArgumentException("Invalid board position");
        if (board[i][j] != EMPTY)
            throw new IllegalArgumentException("Board position occupied");
        board[i][j] = player; // place the mark for the current player
        player = -player; // switch players (uses fact that O = - X)
    }

    /** Checks whether the board configuration is a win for the given player */
    public boolean isWin(int mark)
    {
        return ((board[0][0] + board[0][1] + board[0][2] == mark * 3) // row 0
                || (board[1][0] + board[1][1] + board[1][2] == mark * 3) // row 1
                || (board[2][0] + board[2][1] + board[2][2] == mark * 3) // row 2
                || (board[0][0] + board[1][0] + board[2][0] == mark * 3) // column 0
                || (board[0][1] + board[1][1] + board[2][1] == mark * 3) // column 1
                || (board[0][2] + board[1][2] + board[2][2] == mark * 3) // column 2
                || (board[0][0] + board[1][1] + board[2][2] == mark * 3) // diagonal
        || (board[2][0] + board[1][1] + board[0][2] == mark * 3)); // diagonal
    }

    /** Returns the winning player or 0 to indicate a tie */
    public int winner()
    {
        if (isWin(X)) return (X);
        else if (isWin(O)) return (O);
        else return (0);
    }
```

Java Code for Tic-Tac-Toe

```
    /** Returns a simple character string showing the current board */
    public String toString()
    {
        String s = "";
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                switch (board[i][j])
                {
                case X:
                    s += "X";
                    break;
                case O:
                    s += "O";
                    break;
                case EMPTY:
                    s += " ";
                    break;
                }
                if (j < 2)
                    s += "|"; // column boundary
            }
            if (i < 2)
                s += "\n-----\n"; // row boundary
        }
        return s;
    }
    /** Test run of a simple game */
    public static void main(String[] args)
    {
        TicTacToe game = new TicTacToe();
        /* X moves: */          /* O moves: */
        game.putMark(1, 1);     game.putMark(0, 2);
        game.putMark(2, 2);     game.putMark(0, 0);
        game.putMark(0, 1);     game.putMark(2, 1);
        game.putMark(1, 2);     game.putMark(1, 0);
        game.putMark(2, 0);
        System.out.println(game.toString());
        int winningPlayer = game.winner();
        if (winningPlayer != 0)
            System.out.println(winningPlayer + " wins");
        else
            System.out.println("Tie");
    }
}
```

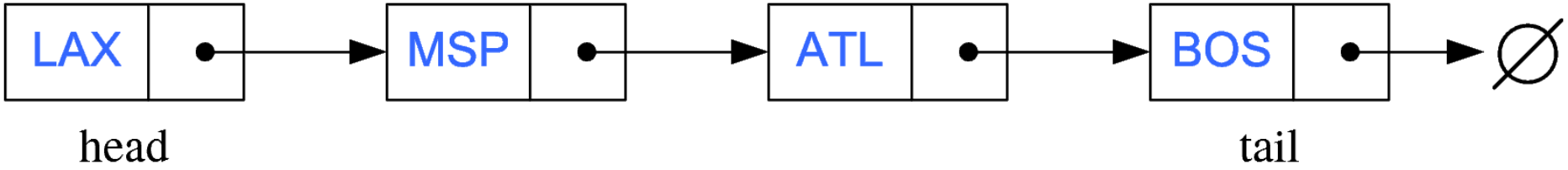
Sample Output

```
O|X|O
-----
O|X|X
-----
X|O|X
Tie
```

Singly Linked Lists

☐ Linked List

- Collection of nodes that form linear ordering
- Node = object storing the reference to an element
  + reference to another node (`next`)
- Head = first node, Tail = last node



- Useful when we only need to access the elements sequentially
- Useful when the collection grows/shrinks in size

Implementing Node

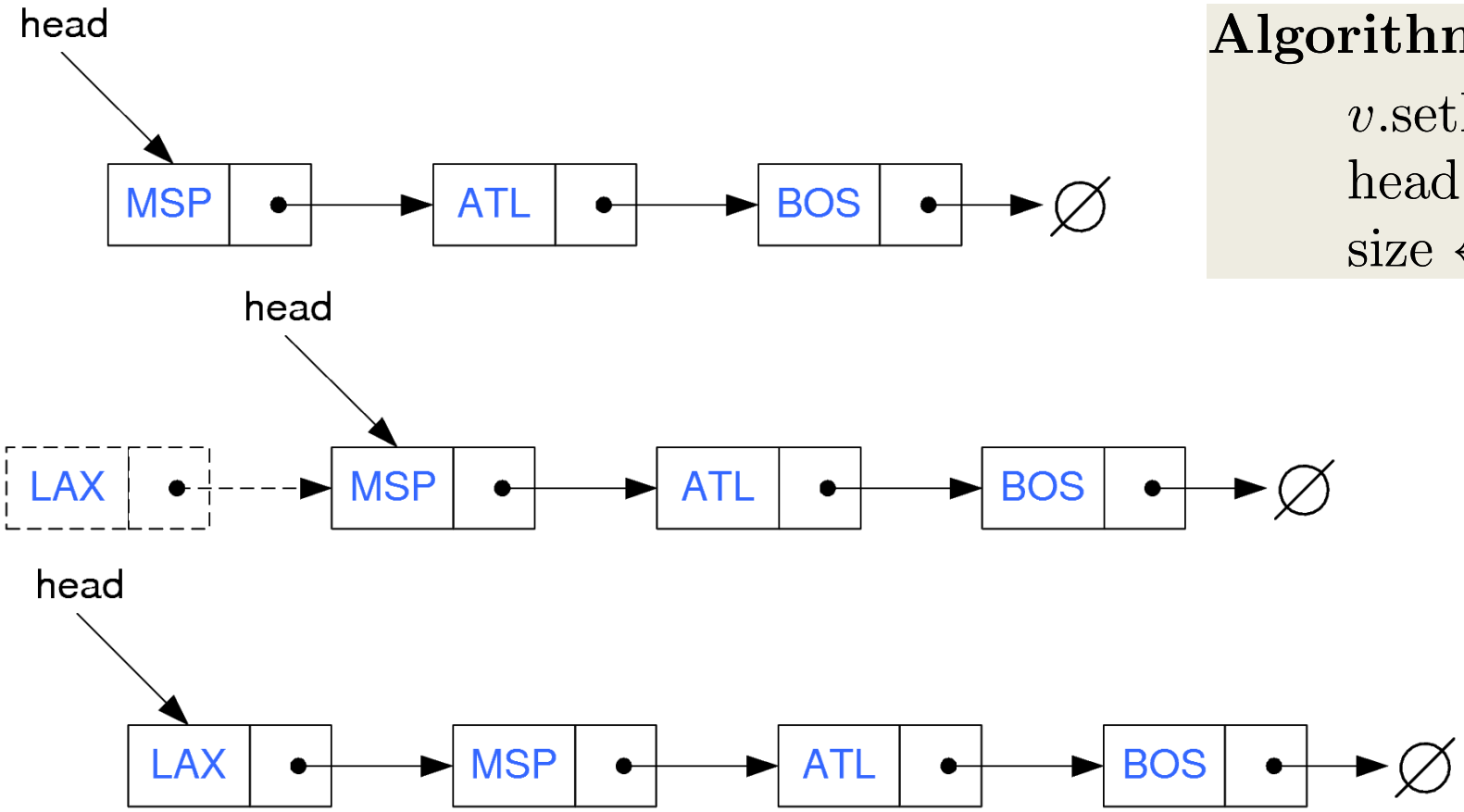
```
/** Node of a singly linked list of strings. */
public class Node
{
    private String element; // we assume elements are character strings
    private Node next;

    /** Creates a node with the given element and next node. */
    public Node(String s, Node n)
    {
        element = s;
        next = n;
    }

    /** Returns the element of this node. */
    public String getElement()
    {
        return element;
    }

    /** Returns the next node of this node. */
    public Node getNext()
    {
        return next;
    }

    // Modifier methods:
    /** Sets the element of this node. */
    public void setElement(String newElem)
    {
        element = newElem;
    }

    /** Sets the next node of this node. */
    public void setNext(Node newNext)
    {
        next = newNext;
    }
}
```

Implementing Singly Linked List

```
public class SinglyLinkedList
{
    protected Node head;         // head node of the list
    protected long size;       // number of nodes in the list

    /** Default constructor that creates an empty list */
    public SinglyLinkedList() {
      head = null;
      size = 0;
    }
    // ... update and search methods would go here ...
}
```

Insertion in Singly Linked List

☐ Inserting element at head



Algorithm addFirst(v)

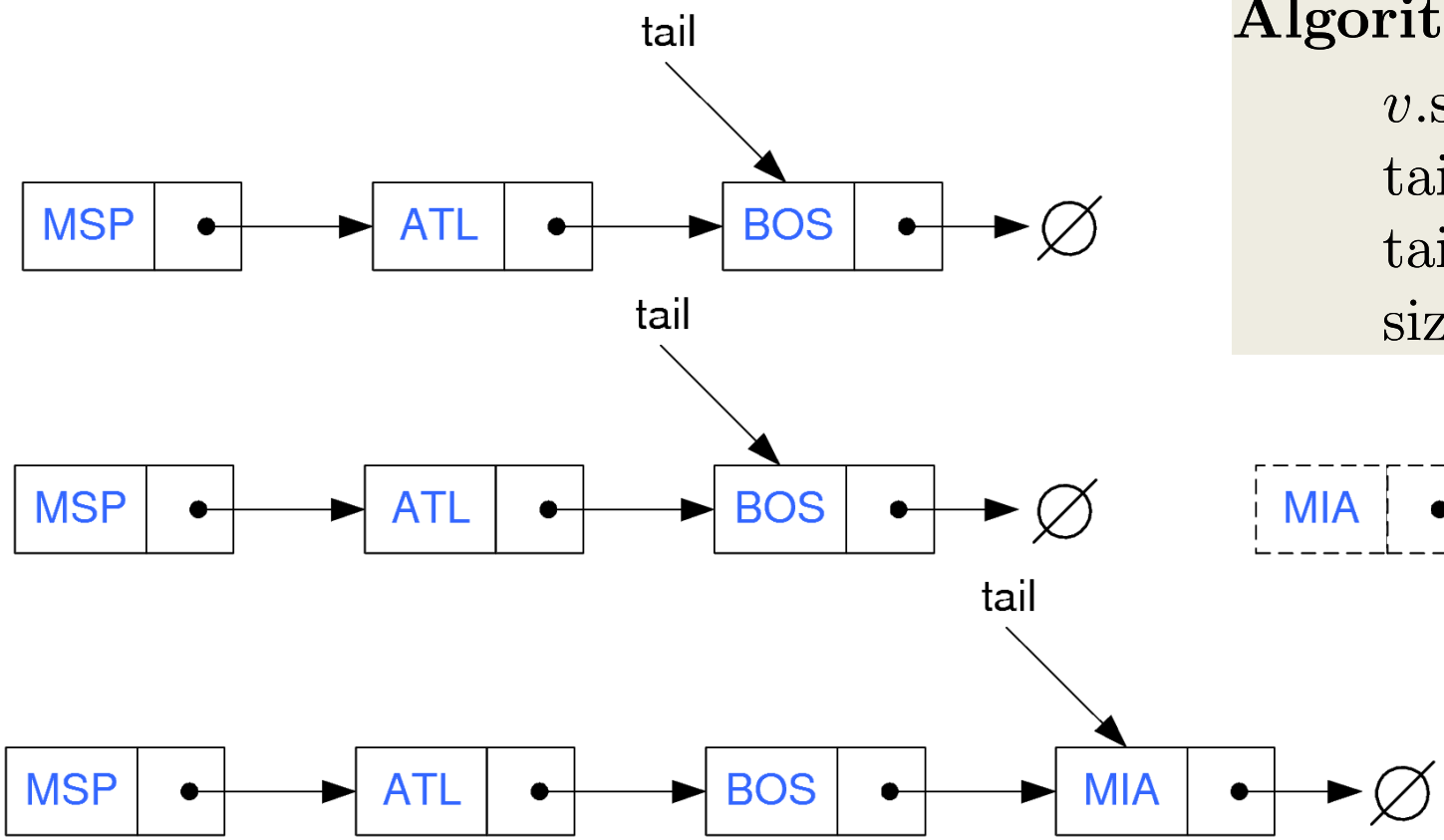
v.setNext(head)

head $\leftarrow v$

size $\leftarrow$ size $+ 1$

Insertion in Singly Linked List

☐ Inserting element at tail



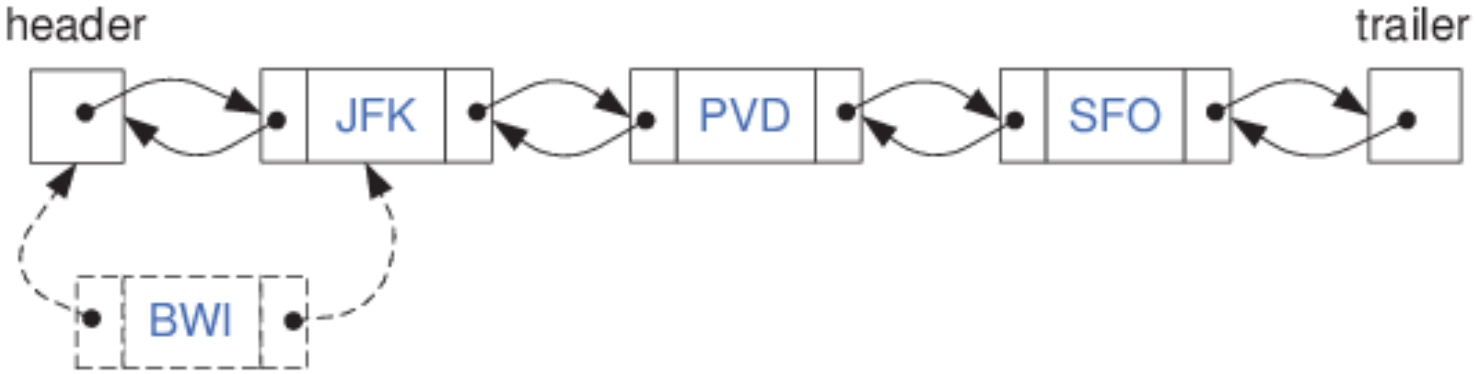
Algorithm addLast(v)

v.setNext(**null**)
tail.setNext(v)
tail $\leftarrow$ v
size $\leftarrow$ size $+ 1$

Doubly linked list

☐ Have two ways links

- Useful for inserting and deleting in a middle



Class Objectives were:

- ☐ Understand representations and operations of arrays and singly linked lists
  - How to represent them
  - How to insert, remove, or sort them

Any Questions?

☐ Come up with one question on what we have discussed in the class and submit at the end of the class

- 1 for typical questions
- 2 for questions with thoughts or that surprised me
- Anything related to class materials is okay

☐ Write questions at least 4 times

- Write a question about one out of four classes
- Unrelated questions are not counted (e.g., faculty club에 학부생도 출입이 가능한가요?)

☐ You can type at KLMS

Next Time

☐ Recursion

☐ HW:

- Go over the next lecture slides before the class
- Just 10 min ~ 20 min should be okay