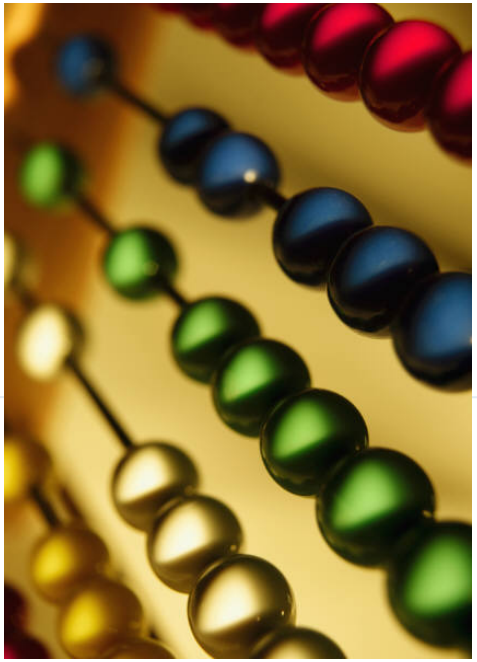
SUNY Korea

Stony Brook University

ARRAYS



Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

ARRAY DEFINITION

- An ***array*** is a sequenced **collection** of variables all of the same type. Each variable, or ***cell***, in an array has an ***index***, which uniquely refers to the value stored in that cell. The cells of an array, A, are numbered 0, 1, 2, and so on.
- Each value stored in an array is often called an ***element*** of that array.

ARRAY LENGTH AND CAPACITY

- Since the length of an array determines the maximum number of things that can be stored in the array, we will sometimes refer to the length of an array as its *capacity*.
- In Java, the length of an array named *a* can be accessed using the syntax *a*.length. Thus, the cells of an array, *a*, are numbered 0, 1, 2, and so on, up through *a*.length−1, and the cell with index *k* can be accessed with syntax *a*[*k*].

DECLARING ARRAYS (FIRST WAY)

- The first way to create an array is to use an assignment to a literal form when initially declaring the array, using a syntax as:

$$elementType[]\ arrayName = \{initialValue_0, initialValue_1, \ldots, initialValue_{N-1}\};$$

- The *elementType* can be any Java base type or class name, and *arrayName* can be any valid Java identifier. The initial values must be of the same type as the array.

DECLARING ARRAYS (SECOND WAY)

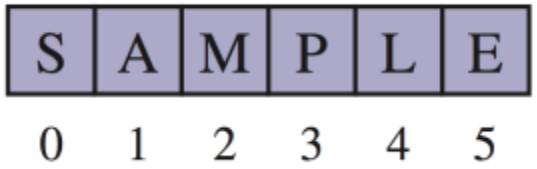
- The second way to create an array is to use the **new** operator.
  - However, because an array is not an instance of a class, we do not use a typical constructor. Instead we use the syntax:

new *elementType*[*length*]

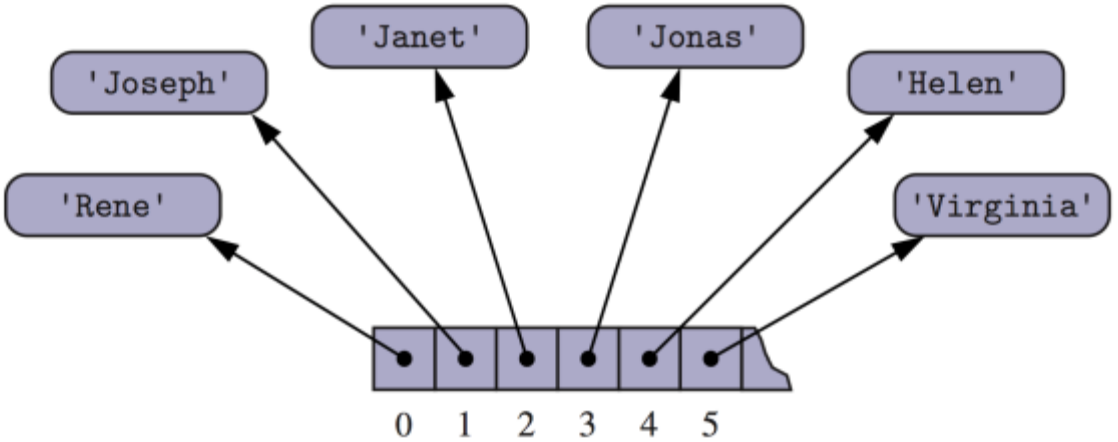
- *length* is a positive integer denoting the length of the new array.
- The **new** operator returns a reference to the new array, and typically this would be assigned to an array variable.

ARRAYS OF CHARACTERS OR OBJECT REFERENCES

- An array can store primitive elements, such as characters.



- An array can also store references to objects.



ARRAY OF OBJECTS: GAME ENTRIES

- A game entry stores the name of a player and her best score so far in a game

```
1  public class GameEntry {
2    private String name;                    // name of the person earning this score
3    private int score;                      // the score value
4    /** Constructs a game entry with given parameters.. */
5    public GameEntry(String n, int s) {
6      name = n;
7      score = s;
8    }
9    /** Returns the name field. */
10   public String getName( ) { return name; }
11   /** Returns the score field. */
12   public int getScore( ) { return score; }
13   /** Returns a string representation of this entry. */
14   public String toString( ) {
15     return "(" + name + ",  " + score + ")";
16   }
17 }
```

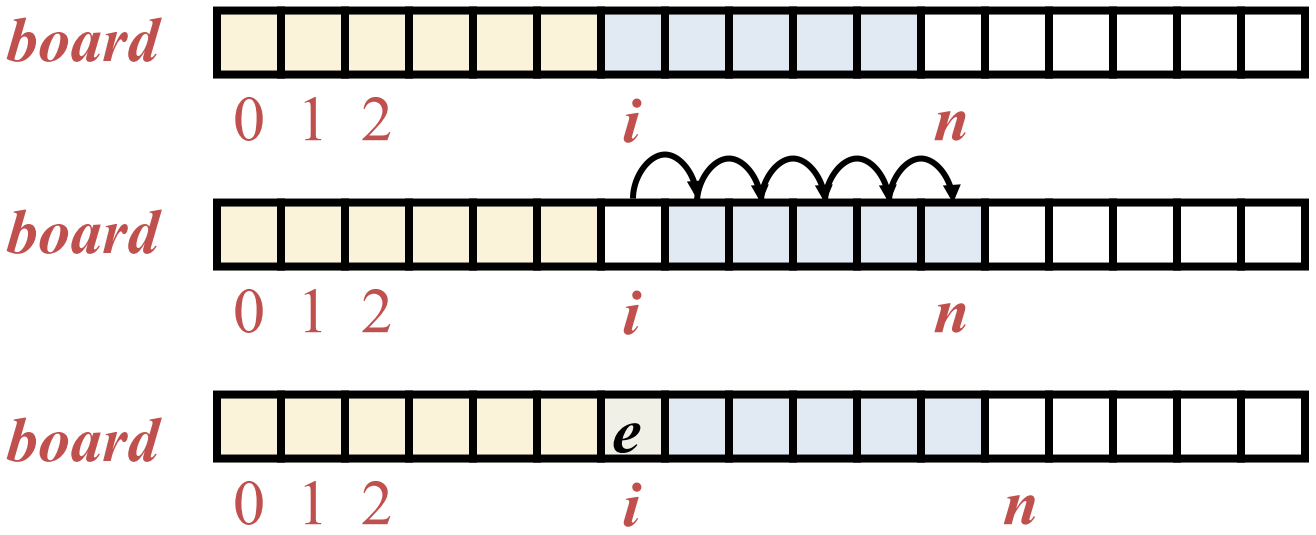
JAVA EXAMPLE: SCOREBOARD

- Keep track of players and their best scores in an array, board
  - The elements of board are objects of class GameEntry

```
1   /** Class for storing high scores in an array in nondecreasing order. */
2   public class Scoreboard {
3     private int numEntries = 0;                  // number of actual entries
4     private GameEntry[ ] board;                  // array of game entries (names & scores)
5     /** Constructs an empty scoreboard with the given capacity for storing entries. */
6     public Scoreboard(int capacity) {
7       board = new GameEntry[capacity];
8     }
...   // more methods will go here
36  }
```

ADDING AN ENTRY

- To add an entry e into array board at index i, we need to make room for it by shifting forward the $n - i$ entries $board[i], \ldots, board[n-1]$

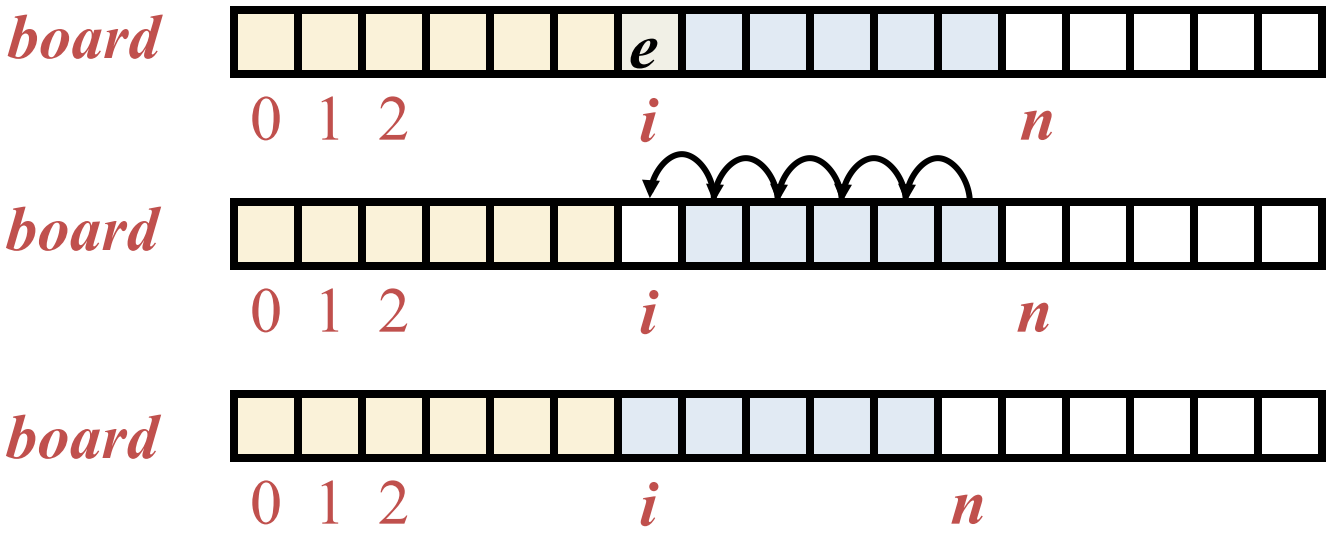


ADDING AN ENTRY: JAVA EXAMPLE

```
 9   /** Attempt to add a new score to the collection (if it is high enough) */
10   public void add(GameEntry e) {
11     int newScore = e.getScore( );
12     // is the new entry e really a high score?
13     if (numEntries < board.length || newScore > board[numEntries−1].getScore( )) {
14       if (numEntries < board.length)                    // no score drops from the board
15         numEntries++;                                   // so overall number increases
16       // shift any lower scores rightward to make room for the new entry
17       int j = numEntries − 1;
18       while (j > 0 && board[j−1].getScore( ) < newScore) {
19         board[j] = board[j−1];                          // shift entry from j-1 to j
20         j−−;                                            // and decrement j
21       }
22       board[j] = e;                                     // when done, add new entry
23     }
24   }
```

REMOVING AN ENTRY

- To remove the entry e at index i, we need to fill the hole left by e by shifting backward the $n - i - 1$ elements $board[i + 1], \ldots, board[n - 1]$



JAVA EXAMPLE

```
25    /** Remove and return the high score at index i. */
26    public GameEntry remove(int i) throws IndexOutOfBoundsException {
27      if (i < 0 || i >= numEntries)
28        throw new IndexOutOfBoundsException("Invalid index: " + i);
29      GameEntry temp = board[i];                    // save the object to be removed
30      for (int j = i; j < numEntries - 1; j++)      // count up from i (not down)
31        board[j] = board[j+1];                      // move one cell to the left
32      board[numEntries -1 ] = null;                 // null out the old last score
33      numEntries--;
34      return temp;                                  // return the removed object
35    }
```

INSERTION-SORT ALGORITHM

Algorithm InsertionSort(A):
Input: An array A of n comparable elements
Output: The array A with elements rearranged in nondecreasing order
for k from 1 to $n-1$ **do**
Insert $A[k]$ at its proper location within $A[0], A[1], \ldots, A[k]$.

Code Fragment 3.5: High-level description of the insertion-sort algorithm.

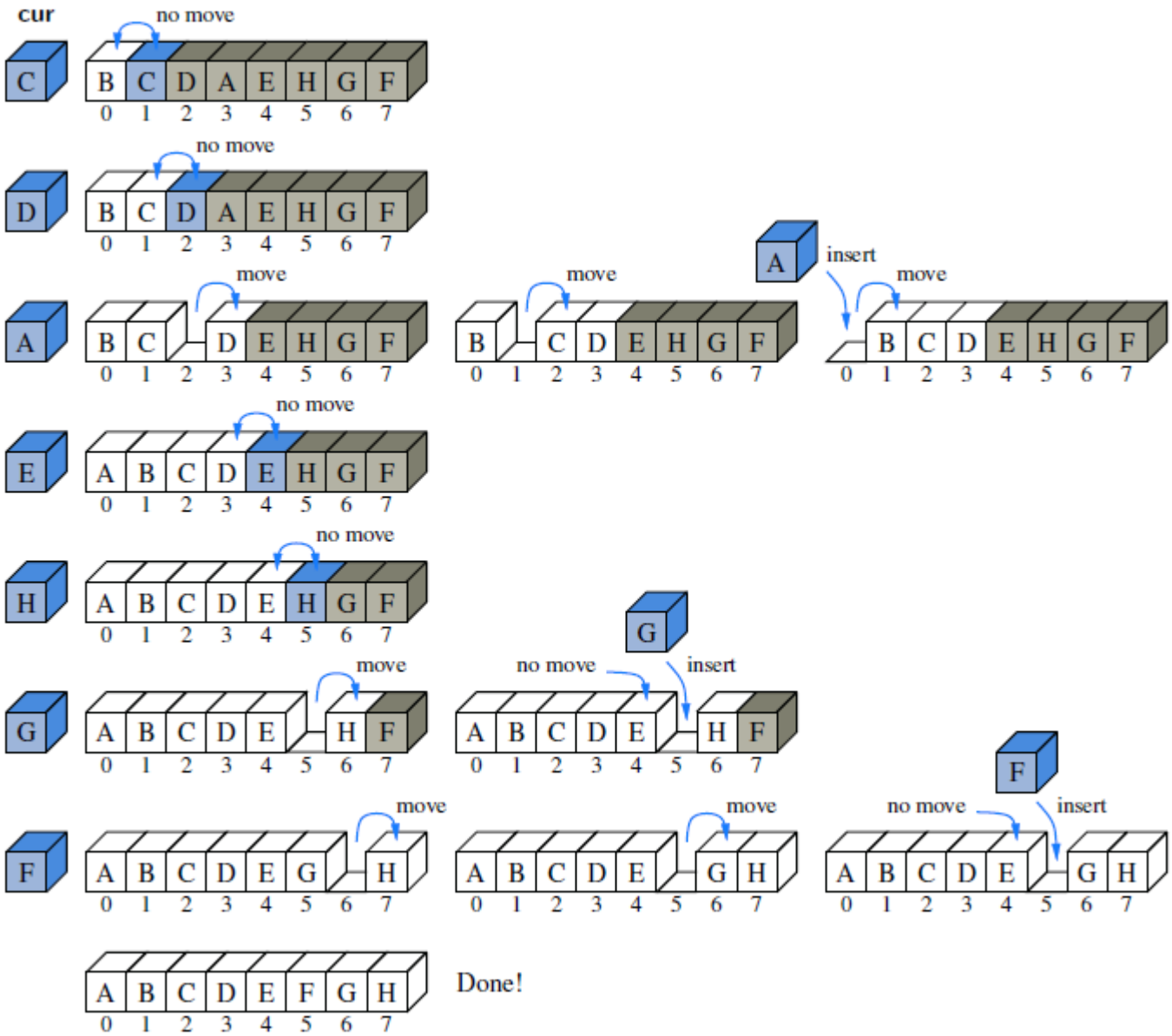
The algorithm proceeds by considering one element at a time, placing the element in the correct order relative to those before it.

Insertion-Sort Java Example:

```
public class InsertionSort {

  /** Insertion-sort of an array of characters into nondecreasing order */
  public static void insertionSort(char[] data) {
    int n = data.length;
    for (int k = 1; k < n; k++) {           // begin with second character
      char cur = data[k];                   // time to insert cur=data[k]
      int j = k;                            // find correct index j for cur
      while (j > 0 && data[j-1] > cur) {    // thus, data[j-1] must go after cur
        data[j] = data[j-1];                // slide data[j-1] rightward
        j--;                                // and consider previous j for cur
      }
      data[j] = cur;                        // this is the proper place for cur
    }
  }

  public static void main(String[] args) {
    char[] a = {'B', 'C', 'D', 'A', 'E', 'H', 'G', 'F'};
    System.out.println(java.util.Arrays.toString(a));
    insertionSort(a);
    System.out.println(java.util.Arrays.toString(a));
  }
}
```



TWO DIMENSIOANL ARRAYS

- Array-of-arrays: we can define a two-dimensional array to be an array with each of its cells being another arra y

```
int[ ][ ] data = new int[8][10];
```

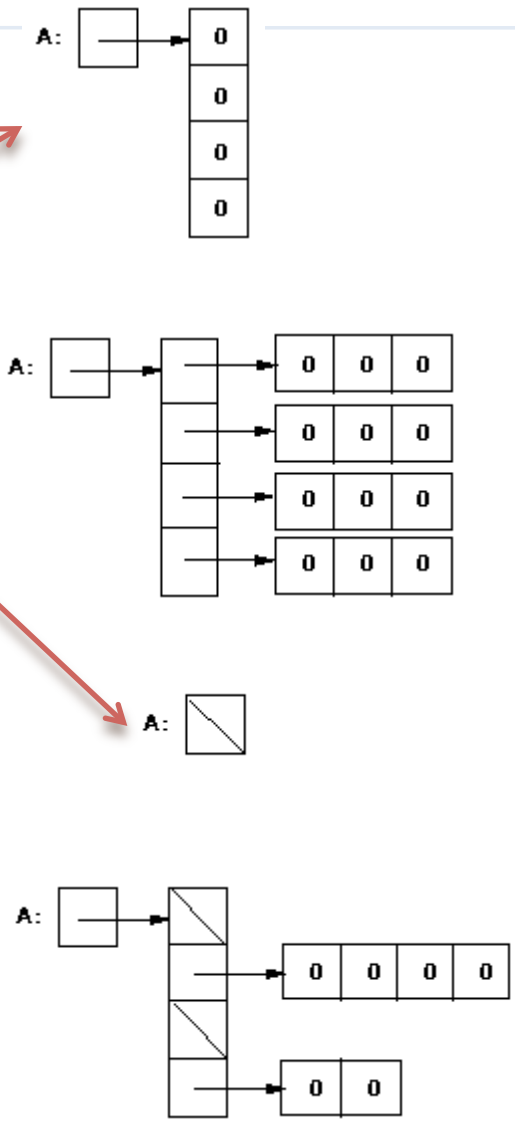
EXAMPLES WITH ARRAYS

```
int [] A;
```

```
int [] A = new int [4];
```

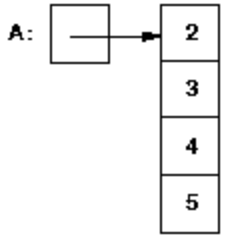
```
int [][] A = new int[4][3];
```

```
int [][] A = new int[4][];
A[1] = new int[4];
A[3] = new int[2];
```

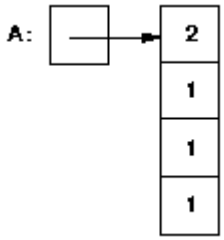


RESOURCE:
http://pages.cs.wisc.edu/~cs368-2/JavaTutorial/NOTES/Java_vs.html

```
int [] A = new int[4];
int [] B = {0,1,2,3,4,5,6,7,8,9};
System.arraycopy(B,2,A,0,4);
```




```
int [] A = new int[4];
int [] B = {2,3,4};
System.arraycopy(B,0,A,0,4);
```




```
int [] A = {1,1,1,1};
int [] B = {2,2,2};
System.arraycopy(A,0,B,1,2);
System.arraycopy(B,0,A,0,3);
```

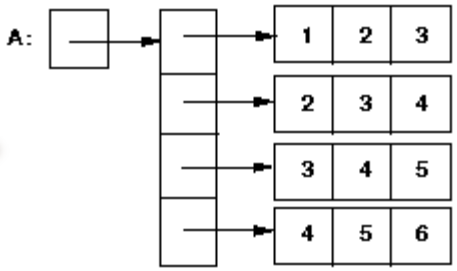
error

RESOURCE:
http://pages.cs.wisc.edu/~cs368-2/JavaTutorial/NOTES/Java_vs.html

```
int [] A = new int[4];
int [] B = {0,1,2,3,4,5,6,7,8,9};
System.arraycopy(B,8,A,0,4);
```

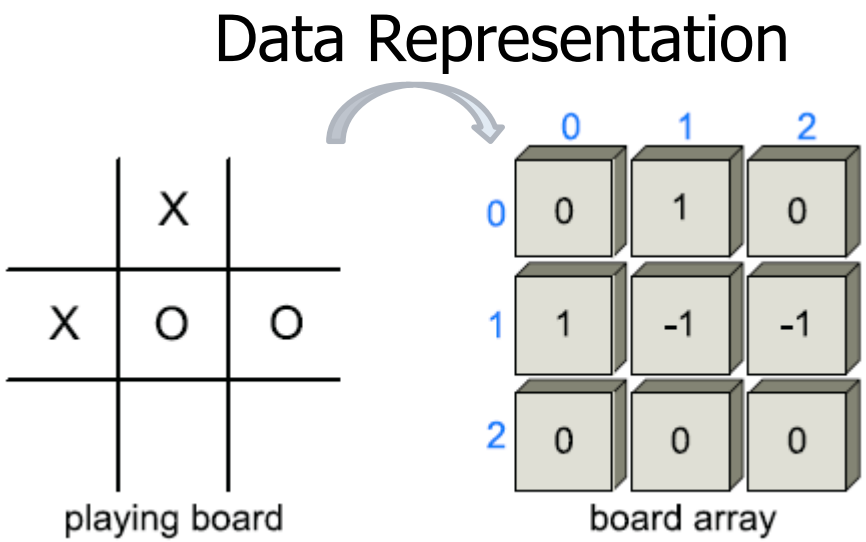
→ error

```
int [][] A = new int[4][3];
int [] B = {1,2,3,4,5,6,7,8,9,10};
System.arraycopy(B,0,A[0],0,3);
System.arraycopy(B,1,A[1],0,3);
System.arraycopy(B,2,A[2],0,3);
System.arraycopy(B,3,A[3],0,3);
```



RESOURCE:
http://pages.cs.wisc.edu/~cs368-2/JavaTutorial/NOTES/Java_vs.html

TIC-TAC-TOE USING 2D-ARRAYS



Choose to make the cells in the board array be integers, with a 0 indicating an empty cell, a 1 indicating an X, and a −1 indicating an O.

Given board configuration, X or O wins, if the values of a row, column, or diagonal add up to 3 or −3, respectively.

TIC-TAC-TOE JAV A EXAMPLE

```
1  /** Simulation of a Tic-Tac-Toe game (does not do strategy). */
2  public class TicTacToe {
3    public static final int X = 1, O = -1;          // players
4    public static final int EMPTY = 0;              // empty cell
5    private int board[ ][ ] = new int[3][3];        // game board
6    private int player;                             // current player
7    /** Constructor */
8    public TicTacToe( ) { clearBoard( ); }
9    /** Clears the board */
10   public void clearBoard( ) {
11     for (int i = 0; i < 3; i++)
12       for (int j = 0; j < 3; j++)
13         board[i][j] = EMPTY;                      // every cell should be empty
14     player = X;                                   // the first player is 'X'
15   }
16   /** Puts an X or O mark at position i,j. */
17   public void putMark(int i, int j) throws IllegalArgumentException {
18     if ((i < 0) || (i > 2) || (j < 0) || (j > 2))
19       throw new IllegalArgumentException("Invalid board position");
20     if (board[i][j] != EMPTY)
21       throw new IllegalArgumentException("Board position occupied");
22     board[i][j] = player;              // place the mark for the current player
23     player = - player;                 // switch players (uses fact that O = - X)
24   }
25   /** Checks whether the board configuration is a win for the given player. */
26   public boolean isWin(int mark) {
27     return ((board[0][0] + board[0][1] + board[0][2] == mark*3)      // row 0
28             || (board[1][0] + board[1][1] + board[1][2] == mark*3)   // row 1
29             || (board[2][0] + board[2][1] + board[2][2] == mark*3)   // row 2
30             || (board[0][0] + board[1][0] + board[2][0] == mark*3)   // column 0
31             || (board[0][1] + board[1][1] + board[2][1] == mark*3)   // column 1
32             || (board[0][2] + board[1][2] + board[2][2] == mark*3)   // column 2
33             || (board[0][0] + board[1][1] + board[2][2] == mark*3)   // diagonal
34             || (board[2][0] + board[1][1] + board[0][2] == mark*3)); // rev diag
35   }
```

```
/** Returns the winning
player's code, or 0 to
indicate a tie (or unfinished
game).*/
public int winner( ) {
  if (isWin(X))
    return(X);
  else if (isWin(O))
    return(O);
  else
    return(0);
}
```

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

```
45   /** Returns a simple character string showing the current board. */
46   public String toString( ) {
47     StringBuilder sb = new StringBuilder( );
48     for (int i=0; i<3; i++) {
49       for (int j=0; j<3; j++) {
50         switch (board[i][j]) {
51           case X:      sb.append("X"); break;
52           case O:      sb.append("O"); break;
53           case EMPTY:  sb.append(" "); break;
54         }
55         if (j < 2) sb.append("|");              // column boundary
56       }
57       if (i < 2) sb.append("\n-----\n");        // row boundary
58     }
59     return sb.toString( );
60   }
61   /** Test run of a simple game */
62   public static void main(String[ ] args) {
63     TicTacToe game = new TicTacToe( );
64     /* X moves: */                 /* O moves: */
65     game.putMark(1,1);             game.putMark(0,2);
66     game.putMark(2,2);             game.putMark(0,0);
67     game.putMark(0,1);             game.putMark(2,1);
68     game.putMark(1,2);             game.putMark(1,0);
69     game.putMark(2,0);
70     System.out.println(game);
71     int winningPlayer = game.winner( );
72     String[ ] outcome = {"O wins", "Tie", "X wins"};  // rely on ordering
73     System.out.println(outcome[1 + winningPlayer]);
74   }
75 }
```