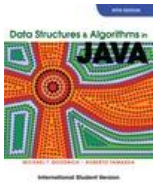
Using Arrays(Chapter 3)

Eleni Mangina

Room B2.05
School of Computer Science and Informatics
University College Dublin, Ireland

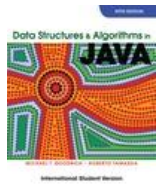


Using Arrays

- Storing Game Entries in an Array
- Sorting an Array
- java.util Methods for Arrays and Random Numbers
- Simple cryptography
- Two dimensional arrays and Positional Games

Storing Game Entries in an Array

```
public class GameEntry {
   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 + ")"; } }
```



A class for High Scores

```
/** 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 ... }
```

Visualising Game Entry Insertion

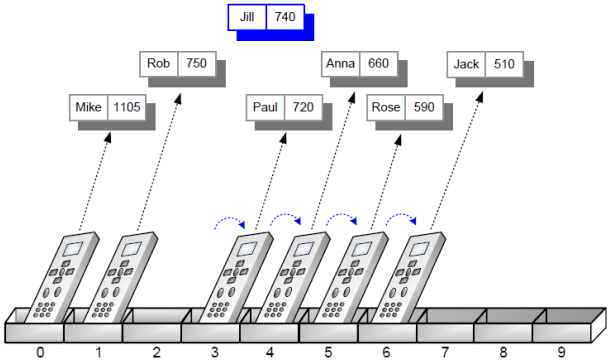


Figure 3.2: Preparing to add a new GameEntry object to the entries array. In order to make room for the new reference, we have to shift the references to game entries with smaller scores than the new one to the right by one cell.

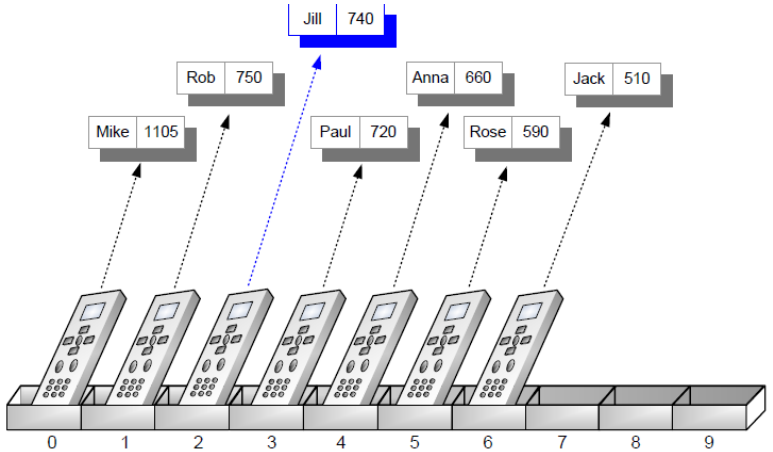


Figure 3.3: Adding a reference to a new GameEntry object to the entries array. The reference can now be inserted at index 2, since we have shifted all references to GameEntry objects with scores less than the new one to the right.

Inserting a GameEntry object

```
/** 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 }
```

Object removal

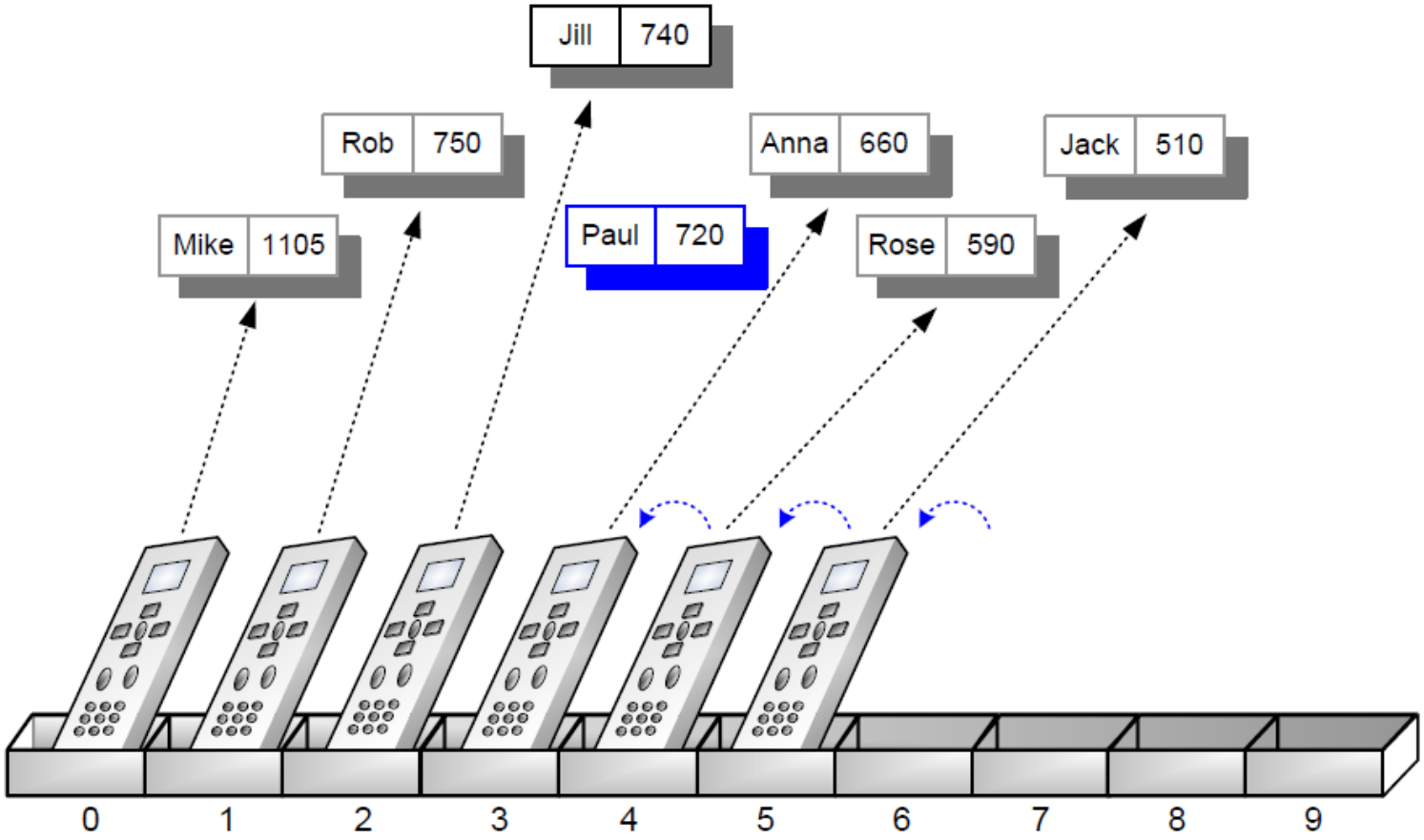


Figure 3.4: An illustration of a removal at index 3 in an array storing references to GameEntry objects.

Remove

```
/** 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 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 } }
```

Example

```
import java.util.Arrays;
import java.util.Random;
/** Program showing some array uses. */

public class ArrayTest {
    public static void main(String[] args) {
          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)); } }
```

Cryptography with Strings and Character Arrays

- Primary applications of arrays is the representation of strings of characters
- Caesar cipher!! Have you heard about it?

```
/** Class for doing encryption and decryption using the Caesar Cipher. */
public class Caesar {
      public static final int ALPHASIZE = 26; // English alphabet (uppercase only)
      public static final char[] alpha = {'A','B','C','D','E','F','G','H', 'I', 'J','K','L','M',
               'N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};
      protected char[] encrypt = new char[ALPHASIZE]; // Encryption array
      protected char[] decrypt = new char[ALPHASIZE]; // Decryption array
      /** Constructor that initializes the encryption and decryption arrays */
      public Caesar() {
               for (int i=0; i<ALPHASIZE; i++) encrypt[i] = alpha[(i + 3) % ALPHASIZE]; // rotate alphabet by 3 places
               for (int i=0; i<ALPHASIZE; i++) decrypt[encrypt[i] - 'A'] = alpha[i]; // decrypt is reverse of encrypt }
      /** Encryption method */
      public String encrypt(String secret) {
               char[] mess = secret.toCharArray(); // the message array
               for (int i=0; i<mess.length; i++) // encryption loop
                              if (Character.isUpperCase(mess[i])) // we have a letter to change
                              mess[i] = encrypt[mess[i] - 'A']; // use letter as an index
                              return new String(mess); }
      /** Decryption method */
      public String decrypt(String secret) {
               char[] mess = secret.toCharArray(); // the message array
               for (int i=0; i<mess.length; i++) // decryption loop
                              if (Character.isUpperCase(mess[i])) // we have a letter to change
                              mess[i] = decrypt[mess[i] - 'A']; // use letter as an index
               return new String(mess); }
      /** Simple main method for testing the Caesar cipher */
      public static void main(String[] args) {
               Caesar cipher = new Caesar(); // Create a Caesar cipher object
               System.out.println("Encryption order = " + new String(cipher.encrypt));
               System.out.println("Decryption order = " + new String(cipher.decrypt));
               String secret = "THE EAGLE IS IN PLAY; MEET AT JOE'S.";
               secret = cipher.encrypt(secret); System.out.println(secret); // the ciphertext
               secret = cipher.decrypt(secret); System.out.println(secret); // should be plaintext again } }
```

2D Arrays & Positional Games

- **int** [] [] Y = **new int** [8] [10];

- Y[i][i+1] = Y[i][i] + 3;

  i = a.length;

  j = Y[4].length;

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); }
```

```
/** 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"); } }
```