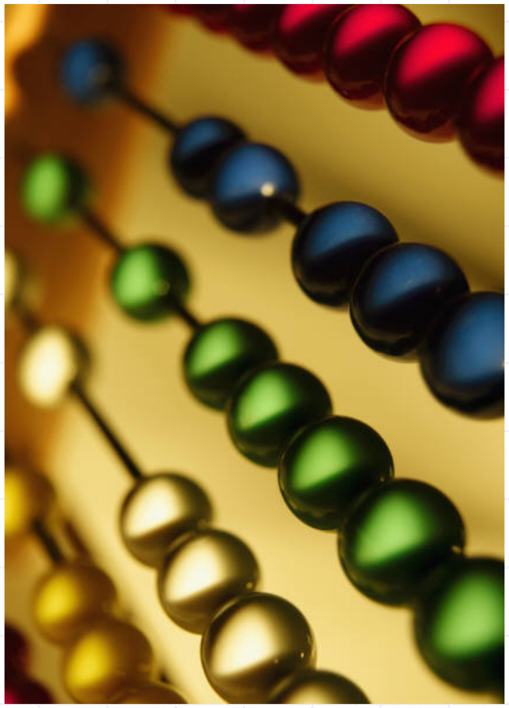
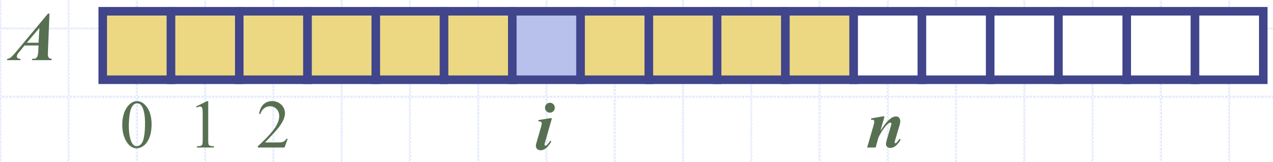
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Arrays



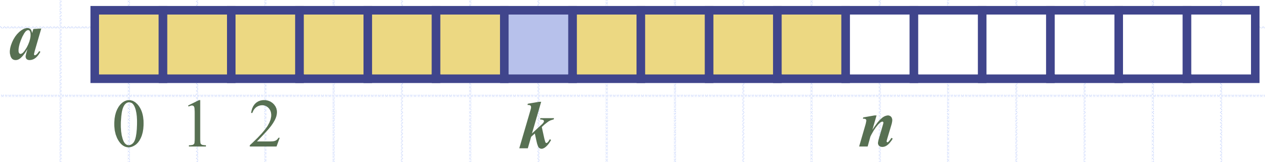
Array Definition

- 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)

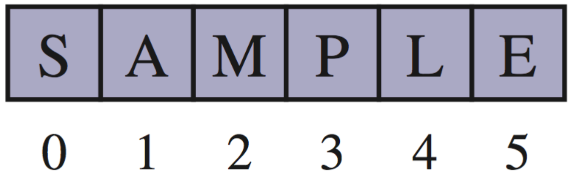
- The second way to create an array is to use the **new** operator.
  - However, because an array is not an instance of a class, we do not use a typical constructor. Instead we use the syntax:

new *elementType*[*length*]

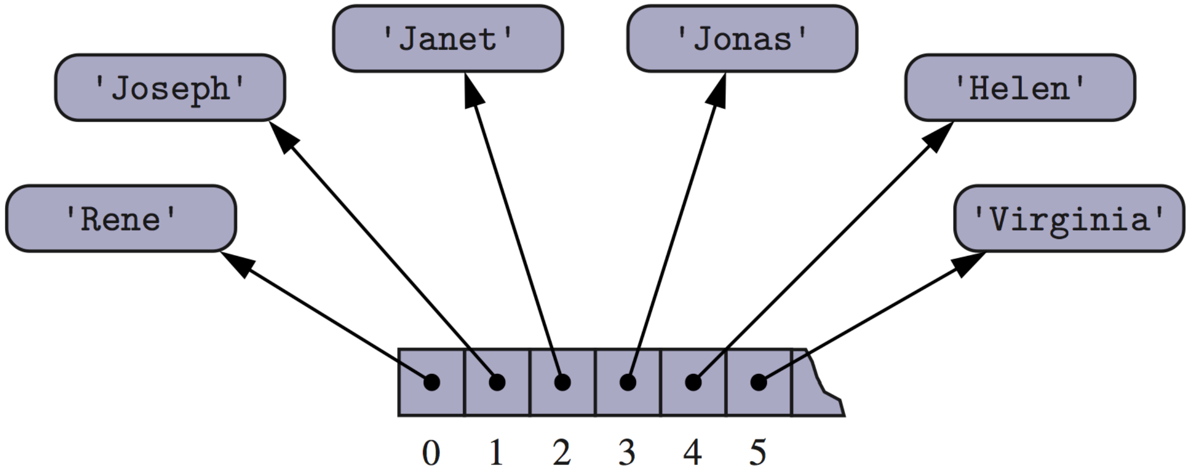
- *length* is a positive integer denoting the length of the new array.
- The **new** operator returns a reference to the new array, and typically this would be assigned to an array variable.

Arrays of Characters or Object References

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



- An array can also store references to objects.



Java Example: Game Entries

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

```
public class GameEntry {
  private String name;                          // name of the person earning this score
  private int score;                            // the score value
  /** Constructs a game entry with given parameters.. */
  public GameEntry(String n, int s) {
    name = n;
    score = s;
  }
  /** Returns the name field. */
  public String getName( ) { return name; }
  /** Returns the score field. */
  public int getScore( ) { return score; }
  /** Returns a string representation of this entry. */
  public String toString( ) {
    return "(" + name + ", " + score + ")";
  }
}
```

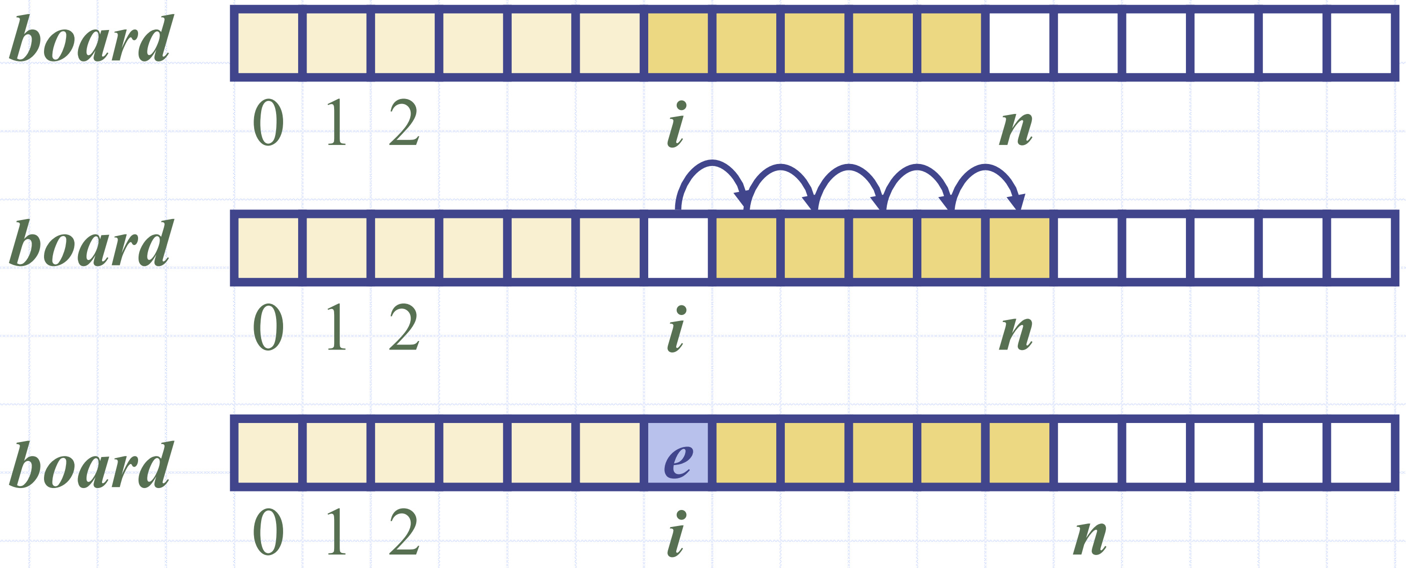
Java Example: Scoreboard

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

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

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 $board[i], \ldots, board[n-1]$

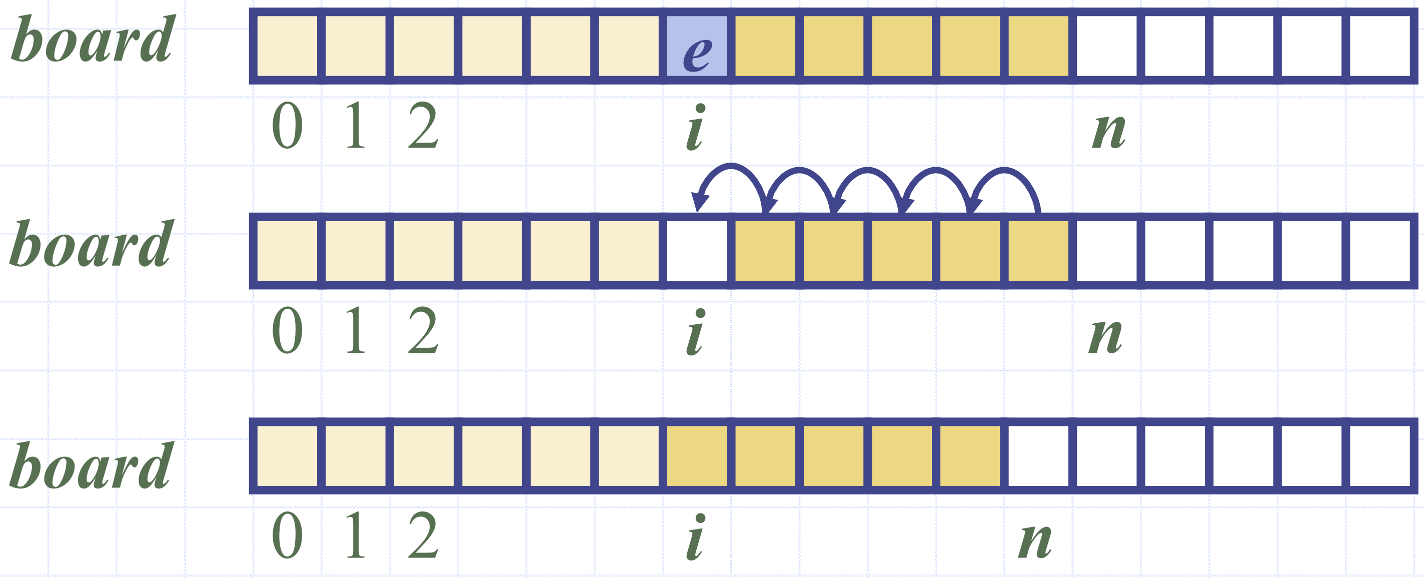


Java Example

```
/** 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 < board.length || newScore > board[numEntries−1].getScore( )) {
    if (numEntries < board.length)                  // no score drops from the board
      numEntries++;                                 // so overall number increases
    // shift any lower scores rightward to make room for the new entry
    int j = numEntries − 1;
    while (j > 0 && board[j−1].getScore( ) < newScore) {
      board[j] = board[j−1];                        // shift entry from j-1 to j
      j−−;                                          // and decrement j
    }
    board[j] = e;                                   // when done, add new entry
  }
}
```

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 $\textit{board}[i + 1], \ldots, \textit{board}[n - 1]$



Java Example

```
/** 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 = board[i];                      // save the object to be removed
  for (int j = i; j < numEntries − 1; j++)        // count up from i (not down)
    board[j] = board[j+1];                        // move one cell to the left
  board[numEntries −1 ] = null;                   // null out the old last score
  numEntries−−;
  return temp;                                    // return the removed object
}
```