Array: A fixed-length data structure for storing multiple values of the same type

Example: An array of odd numbers:

<table>
<thead>
<tr>
<th>Indices (start from 0)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>odds</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>13</td>
</tr>
</tbody>
</table>

The type of all elements is int
The value of the element at index 4 is 9: odds[4] == 9

---

Array Declaration

- An array is denoted by the [] notation
- Examples:
  - int[] odds;
  - int odds[];  // legal but discouraged
  - String[] names;
  - int[][] matrix; // an array of arrays

---

Array Creation and Initialization

What is the output of the following code:
```java
int[] odds = new int[8];
for (int i = 0; i < odds.length; i++) {
    System.out.print(odds[i] + " ");
    odds[i] = 2 * i + 1;
    System.out.print(odds[i] + ");
}
```

Output:
0 1 3 5 7 9 11 13 15

---

Array Creation and Initialization

Creating and initializing small arrays with a-priori known values:
```java
int[] odds = {1,3,5,7,9,11,13,15};
String[] months = {
```

---

Loop through Arrays

By promoting the array's index:
```java
for (int i = 0; i < months.length; i++) {
    System.out.println(months[i]);
}
```

foreach (since Java 5.0):
```java
for (String month: months) {
    System.out.println(month);
}
```
Operations on arrays

- The class Arrays provide operations on array
  - Copy
  - Sort
  - Search
  - Fill
  - ...

java.util.Arrays
http://java.sun.com/javase/6/docs/api/java/util/Arrays.html

Copying Arrays

- Assume:
  int[] array1 = {1,2,3};
  int[] array2 = {8,7,6,5};

- Naïve copy:
  array1 = array2;

- How would we copy an array?

Copying Arrays

- Arrays.copyOf
  the original array
  the length of the copy

int[] arr1 = {1, 2, 3};
int[] arr2 = Arrays.copyOf(arr1, arr1.length);

- Arrays.copyOfRange
  the original array
  initial index of the range to be copied, inclusive
  final index of the range to be copied, exclusive

What is the output of the following code:

int[] odds = {1, 3, 5, 7, 9, 11, 13, 15};
int newOdds[] =
Arrays.copyOfRange(odds, 1, odds.length);
for (int odd: newOdds) {
    System.out.print(odd + " ");
}

Output: 3 5 7 9 11 13 15

Other Manipulations on Arrays

- The java.util.Arrays class has methods for sorting and searching, assigning arrays e.g.
  - public static void sort(int[] a)
  - public static int binarySearch(int[] a, int key)
  - public static void fill(long[], long val)

More details in JDK 6.0 documentation
http://java.sun.com/javase/6/docs/api/java/util/Arrays.html

2D Arrays

- There are no 2D arrays in Java but …
  - you can build array of arrays:
    char[][] board = new char[3][];
    for (int i = 0; i < 3; i++)
        board[i] = new char[3];
2D Arrays

Building a multiplication table:
```
int[][] table = new int[10][10];
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        table[i][j] = (i+1) * (j+1);
    }
}
```

Fibonacci

Fibonacci series
1, 1, 2, 3, 5, 8, 13, 21, 34

Definition:
- \( \text{fib}(0) = 1 \)
- \( \text{fib}(1) = 1 \)
- \( \text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2) \)

If-Else Statement

```java
public class Fibonacci {
    /** Returns the n-th Fibonacci element */
    public static int computeElement(int n) {
        if (n==0)
            return 1;
        else if (n==1)
            return 1;
        else
            return computeElement(n-1) + computeElement(n-2);
    }
}
```

Switch Statement

```java
public class Fibonacci {
    /** Returns the n-th Fibonacci element */
    public static int computeElement(int n) {
        switch(n) {
            case 0:
                return 1;
            case 1:
                return 1;
            default:
                return computeElement(n-1) + computeElement(n-2);
        }
    }
}
```

Compilation Error: Dead Code

Assumption: \( n \geq 0 \)
For Loop

A loop instead of a recursion

```java
static int computeElement(int n) {
    if (n == 0 || n == 1)
        return 1;
    int prev = 1;
    int prevPrev = 1;
    int curr;
    for (int i = 2; i < n; i++) {
        curr = prev + prevPrev;
        prevPrev = prev;
        prev = curr;
    }
    curr = prev + prevPrev;
    return curr;
}
```

Assumption: n ≥ 0

Discussion: Efficiency vs. Simplicity.

The KISS (keep it simple stupid) main function:

```java
given int n
for (int i = 0; i < n; i++)
    System.out.println(computeElement(i));
```

It is better to use args[0] outside the for block...

```java
int i = 0;
while (i < n) {
    System.out.println(computeElement(i));
    i++;
}
```

Variable i is not defined outside the for block.

**Note:**

- The following two statements are almost equivalent:
- for (int i = 0; i < n; i++)
  System.out.println(computeElement(i));
- int i=0;
  while (i < n) {
    System.out.println(computeElement(i));
    i++;
  }

- The for statement is better for iteration.
- The while statement is better for testing conditions.

**Note:**

- In general, loops are more efficient than recursive calls.
- The for statement is more flexible and easier to read.
- The while statement is more suitable for complex conditions.
- The do-while statement is suitable for loops that must execute at least once.

For Loop

Printing the first n elements:

```java
public class Fibonacci {
    public static int computeElement(int n) {
        return 1;
    }
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++)
            System.out.println(computeElement(i));
    }
}
```

Modularity, Scoping, and Efficiency

- It is better to use args[0] outside the for block...
- The for statement is more efficient.
- The while statement is more flexible.
- The do-while statement is suitable for loops that must execute at least once.

Modularity, Scoping, and Efficiency

- The following statements are almost equivalent:
- for (int i = 0; i < n; i++)
  System.out.println(computeElement(i));
- int i=0;
  while (i < n) {
    System.out.println(computeElement(i));
    i++;
  }

- The for statement is better for iteration.
- The while statement is better for testing conditions.
- The do-while statement is suitable for loops that must execute at least once.

**Note:**

- In general, loops are more efficient than recursive calls.
- The for statement is more flexible and easier to read.
- The while statement is more suitable for complex conditions.
- The do-while statement is suitable for loops that must execute at least once.

```java
int i = 0;
while (i < n) {
    System.out.println(computeElement(i));
    i++;
}
```

```java
for (int i = 0; i < n; i++)
    System.out.println(computeElement(i));
```
while vs. do while

The following two statements are equivalent if and only if $n > 0$:

```java
int i = 0;
while (i < n) {
    System.out.println(computeElement(i));
    i++;
}
```

```java
int i = 0;
do {
    System.out.println(computeElement(i));
    i++;
} while (i < n);
```

works since $n \geq 1$...

Compilation vs. Runtime Errors

Shonatot ve-yom ravo: Al nit du'lut shinayeh sanoah bemikra pasif.

Shonatot kumfiltrah (hebrew): Shonatot shinot (amharic) bud.

Karatot ko'kewet lekumfiltrah (hebrew) ne derhoy.

Dugimul...

The Debugger

Some programs may compile correctly, yet not produce the desirable results.

These programs are valid and correct Java programs, yet not the programs we meant to write!

The debugger can be used to follow the program step by step and may help detecting bugs in an already compiled program.
Debugger – Add Breakpoint

- Right click on the desired line
- “Toggle Breakpoint”

Debugger – Start Debugging

- Click on the desired line
- “Debug” button

Debugger – Debug Perspective

- Open the Debug Perspective
- Navigate through breakpoints

Debugger – Debugging

- Current state
- Current location
- Back to Java perspective

Using the Debugger: Video Tutorial

- You can find great video tutorials on debugging at:
  [http://eclipsesource.sourceforge.net/debugger.html](http://eclipsesource.sourceforge.net/debugger.html)

- It is recommended to watch at least the first four videos.

* Link to video tutorials in the development course section.*