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

Example from last week: 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>
<th>7</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>
<td>15</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://docs.oracle.com/javase/6/docs/api/index.html?java/util/Arrays.html

Copying Arrays

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

- Naive copy:
  array1 = array2;

- What's wrong with this solution?

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

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

Question

- What is the output of the following code:

```java
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

2D Arrays

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

- Or equivalently:
  ```java
  char[][] board = new char[3][3];
  ```

2D Arrays

- Building a multiplication table:

```java
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);
    }
}
2D Arrays

- A more compact table:

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

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>6</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fibonacci

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

- Definition:
  \[
  \begin{align*}
  \text{fib}(0) &= 1 \\
  \text{fib}(1) &= 1 \\
  \text{fib}(n) &= \text{fib}(n-1) + \text{fib}(n-2) \\
  \end{align*}
  \]

- Assumption:
  \( n \geq 0 \)

## 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);
        }
    }
}
```

## Iterative Fibonacci

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

- Assumption:
  \( n \geq 0 \)

- A loop instead of a recursion
for vs. while

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++;
```

Variable i is not defined outside the for block

while vs. do while

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

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

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

works with n=0
...חס