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

```
matrix:
1 1 1 1 ...
```

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: all elements get the default value for their type (0 for int).

Loop through Arrays

By promoting the array's index:

```java
for (int i = 0; i < months.length; i++) {
    System.out.println(months[i]);
}
```

The variable month is assigned the next element in each iteration.

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};
- Naïve copy:
  array1 = array2;
- What's wrong with this solution?

Copying Arrays

- Arrays.copyOf
  - the original array
  - the length of the copy
  ```java
  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
  ```java
  int[] arr[] = {
    1, 2, 3,
    4, 5, 6,
    7, 8, 9,
    10, 11, 12
  };
  ```

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); // array1 = array2;
  ```
  ```java
  for (int odd: newOdds) { // array2
    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][3];
  for (int i = 0; i < 3; i++)
    board[i] = new char[3];
  ```

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][];
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);
    }
}
```

### Fibonacci

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

Definition:
- fib(0) = 1
- fib(1) = 1
- fib(n) = fib(n-1) + fib(n-2)

[en.wikipedia.org/wiki/Fibonacci_number](en.wikipedia.org/wiki/Fibonacci_number)

### 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);
        }
    }
    ...
}
```

Assumption: n ≥ 0

### 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);
        }
    }
    ...
}
```

Assumption: n ≥ 0

### Iterative Fibonacci

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;
    }
    return computeElement(n-1) + computeElement(n-2);
}
```

Assumption: n ≥ 0

Can be placed outside the switch

Compilation Error: Unreachable Code
For Loop

- Printing the first n elements:

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

- It is better to use argument:

```java
public static int computeElement(int n) {
    ...
    // Use argument
    return 0;
}
```

- While:

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

- Do-While:

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

- It works since n ≥ 1

 dreadful (memorization) technique is expensive but saves time and memory.

- In fact, there are many benefits and improvements that can be made.

- The following two statements are equivalent:

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

- The following two statements are almost equivalent:

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

- Variable i is not defined outside the for block.

- Modularity, efficient code.

- The code is concise and clear.

- It is better to use arguments.

- While:

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

- Do-While:

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

- It works since n ≥ 1

- Modularity and efficiency.

- In fact, there are many benefits and improvements that can be made.

- The following two statements are essentially equivalent:

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

- The following two statements are almost equivalent:

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

- Variable i is not defined outside the for block.

- While:

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

- Do-While:

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

- It works since n ≥ 1

- for vs. while

- The following two statements are almost equivalent:

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

- 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++;
}
```

- It works since n ≥ 1

- while vs. do while

- The following two statements are essentially equivalent:

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

- The following two statements are almost equivalent:

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

- Variable i is not defined outside the for block.

- While:

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

- Do-While:

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

- It works since n ≥ 1

- for vs. while

- The following two statements are almost equivalent:

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

- 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++;
}
```

- It works since n ≥ 1

- while vs. do while

- The following two statements are essentially equivalent:

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

- The following two statements are almost equivalent:

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

- Variable i is not defined outside the for block.

- While:

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

- Do-While:

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

- It works since n ≥ 1

- for vs. while

- The following two statements are almost equivalent:

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

- 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++;
}
```

- It works since n ≥ 1

- while vs. do while

- The following two statements are essentially equivalent:

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

- The following two statements are almost equivalent:

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

- Variable i is not defined outside the for block.

- While:

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

- Do-While:

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

- It works since n ≥ 1
...משוך