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

**Example**: An array of odd numbers:

- Array: indices (start from 0)
  - `odds`:
    - `1 3 5 7 9 0 2 15`
  - `odds.length == 8`
  - The type of all elements is `int`
  - The value of the element at index 4 is 9: `odds[4] == 9`

**Array Declaration**

- **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 0 3 0 5 0 7 0 9 0 11 0 13 0 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};
  ```
  ```java
  String[] months = {
  ```
  - **months**: `"Jan"`
### 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

  See also: `System.arraycopy`

### 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](http://java.sun.com/javase/6/docs/api/java/util/Arrays.html)

### 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);
    }
  }
  ```

- 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];
    ```

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

### 2D Arrays

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

- Naive copy:
  ```java
  array1 = array2;
  ```

- How would we copy an array?

### Other Manipulations on Arrays

- 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
**Fibonacci**

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

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

---

**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: $n \geq 0$

**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);
        }
    }
    ...
}
```

Compilation Error: Dead Code

Assumption: $n \geq 0$

**For Loop**

```java
static int computeElement(int n) {
    if (n == 0) n = 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: $n \geq 0$

**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);
        }
    }
    ...
}
```

Compilation Error: Dead Code

Assumption: $n \geq 0$
For Loop

Printing the first n elements:

```java
public class Fibonacci {
    public static int computeElement(int n) {
        // ...
        return fib(n); // Use arg0
    }
    public static void main(String[] args) {
        for(int i = 0; i < 10; 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);
```

while vs. do while

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

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

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

Java Code: Description

- In the `Fibonacci` class, the `computeElement` method calculates the Fibonacci numbers for a given input `n`.
- The `main` method prints the first 10 Fibonacci numbers using a `for` loop.

Modularity vs. Unroll

- Modularity (encapsulation) of a function results in cleaner and more maintainable code.
- However, for this particular example, `memoization` could be used to store previously computed values to improve efficiency.

while vs. do while

- The two statements `while (i < n) { ... }` and `do { ... } while (i<n);` are equivalent if `n > 0`.
- This is because the loop condition is checked at the start of the `while` block, while in `do while`, the loop body is executed first, and then the condition is checked.

for vs. while

- The `for` loop is typically more readable and easier to understand, especially when the loop variables are initialized and incremented within the loop.
- The `while` loop requires the loop condition to be checked at the beginning of each iteration, which might be less intuitive compared to the `for` loop.

- In the `Fibonacci` class, the `main` method uses a `for` loop to print the first 10 Fibonacci numbers.
- For efficiency, the `computeElement` method could benefit from `memoization` to store previously computed values.

- The `for` loop is preferred in this context because it clearly expresses the loop's initialization, condition, and increment, making the code more readable and maintainable.
שאלות?

הswick...