תוכנה 1

תרגול 2: מערכים, מבני בקרה ושגיאות

הדס צור ואסף זריצקי

מערכים

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

- <u>Example</u>: An array of odd numbers:

Indices (start from 0) → 0 1 2 3 4 5 6 7

odds:	1	3	5	7	9	11	13	15

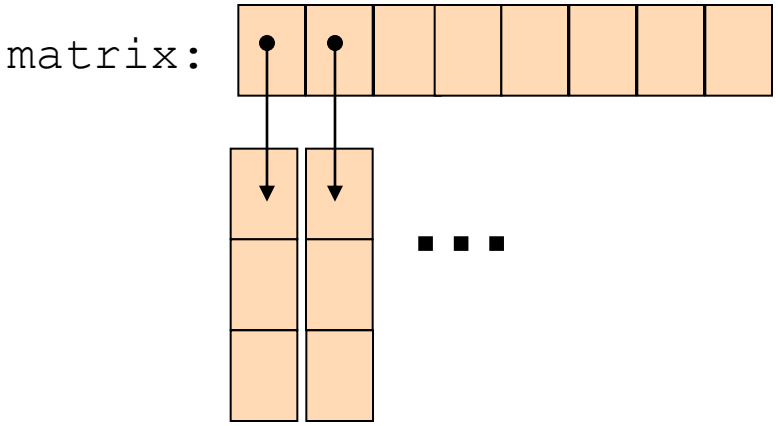
odds.length == 8

The type of all elements is `int`

The value of the element at index 4 is 9: `odds[4] == 9`

Array Declaration

- 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:

```
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] + " ");
}
```

Array creation: all elements get the default value for their type (`0` for `int`)

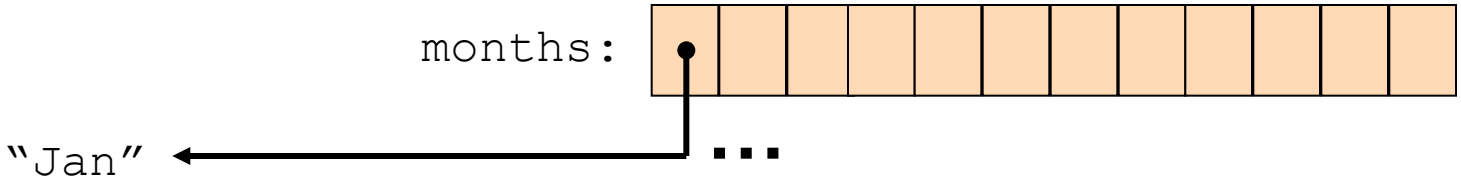
- Output:

```
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:
  - `int[] odds = {1,3,5,7,9,11,13,15};`
  - `String[] months =`

```
{"Jan", "Feb", "Mar", "Apr",
 "May", "Jun", "July", "Aug",
 "Sep", "Oct", "Nov", "Dec"};
```



Loop through Arrays

- By promoting the array's index:

```
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):

```
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

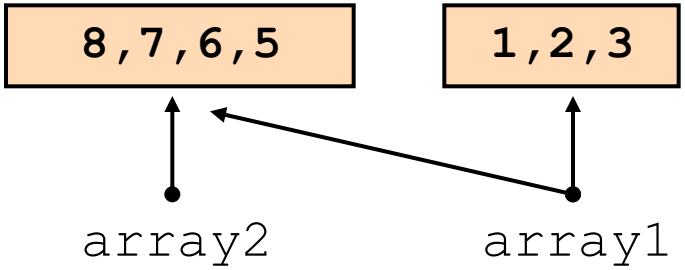
Copying Arrays

- Assume:

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

- Naïve copy:

```
array1 = array2;
```



- 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

- 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

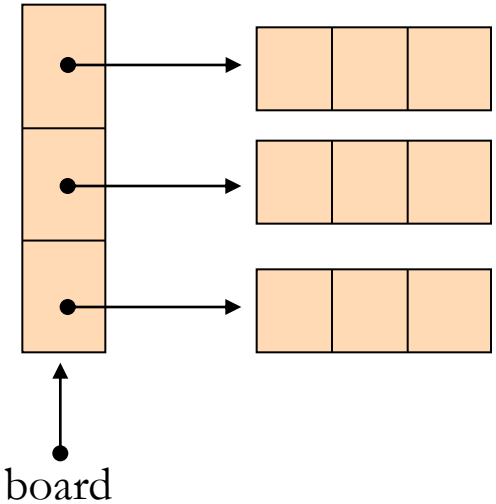
- The java.util.Arrays class has methods for sorting and searching, assigning arrays e.g.
  - public static void **sort**(int[] a)
  - public static int **binarySearch**(int[] a, int key)
  - public static void **fill**(long[] a, long val)

- 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];
```



Or equivalently:
`char[][] board = new char[3][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:
  - fib(0) = 1
  - fib(1) = 1
  - fib(n) = fib(n-1) + fib(n-2)

"Yes, you're right! We have increased our initial investment."

en.wikipedia.org/wiki/Fibonacci_number

If-Else Statement

```
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

Can be
removed

Switch Statement

```
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

can be placed
outside the switch

Switch Statement

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

Assumption:
n≥0

Compilation Error:
Unreachable Code

Iterative Fibonacci

- A loop instead of a recursion

```
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

נתונים במקום חישוב

- בתרגום רקורסיה ללולאה אנו משתמשים במשתני עזר לשמירת המצב `curr, prev` ו- `prevPrev`
- הלולאה "זוכרת" את הנקודה שבה אנו נמצאים בתהליך החישוב
- <u>דיון</u>: יעילות לעומת פשטות.

  עיקרון ה-KISS (**k**eep **i**t **s**imple **s**tupid)
- <u>תרגיל</u>: כתבו את השירות `computeElement` בעזרת `prev` ו-`prevPrev` בלבד (ללא `curr`)

For Loop

- Printing the first n elements:

```
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 args[0]

מודולריות, שכפול קוד ויעילות

- יש כאן חוסר יעילות מסוים:
  - לולאת ה-`for` חוזרת גם ב- `main` וגם ב- `computeElement`. לכאורה, במעבר אחד ניתן גם *לחשב* את האברים וגם *להדפיס* אותם
  - כמו כן כדי לחשב איבר בסדרה איננו משתמשים *בתוצאות שכבר חישבנו* (של אברים קודמים) ומתחילים כל חישוב מתחילתו

מודולריות, שכפול קוד ויעילות

- מתודה (פונקציה) צריכה לעשות דבר אחד בדיוק!
  - ערוב של חישוב והדפסה פוגע במודולריות (מדוע?)
- היזהרו משכפול קוד!
  - קטע קוד דומה המופיע בשתי פונקציות שונות יגרום במוקדם או במאוחר לבאג בתוכנית (מדוע?)
- את בעיית היעילות (הוספת מנגנון memoization) אפשר לפתור בעזרת מערכים (תרגיל)

for vs. while

- The following two statements are almost equivalent:

Variable `i` is not defined outside the for block

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

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

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 since n ≥ 1

הסוף...