# תוכנה 1

תרגול 2: מערכים ומבני בקרה

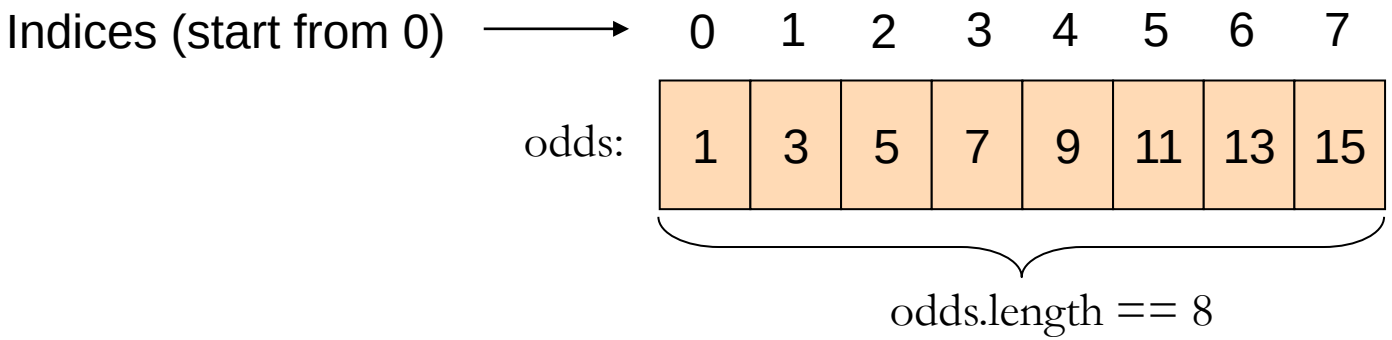
# Useful Eclipse Shortcuts

- **Ctrl+1** – quick fix for errors, or small refactoring suggestions
- **Ctrl+SPACE** – code content assist (auto-completion)
  - Auto completion for “main” – create a template for main function
  - Auto completion for “print” – system.out.println()
  - Auto completion for “for” – loop structures
  - And many more, see Window > Preferences > Java > Editor > Content assist > templates
- **Ctrl+Shift+S** – save changes in all open files
- **Ctrl+Shift+F** – auto-formatting of the code
  (always use it before you submit your HW!)
- **Ctrl+Shift+O** – organize imports
  (which allows using external classes)
- **Ctrl+F11** – run, **F11** – debug
- **Alt+Shift+R** – rename (a variable, method, class)

- All the shortcuts are listed (and can be customized in Window > Preferences > General > Keys

# מערכים

- **Array:** A fixed-length data structure for storing multiple values of the same type
- Example from last week: An array of odd numbers:

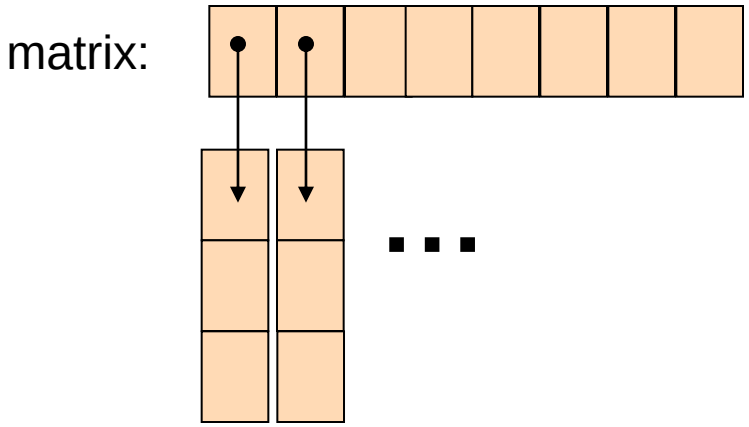



The type of all elements is `int`

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

# Array Variables

- 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)

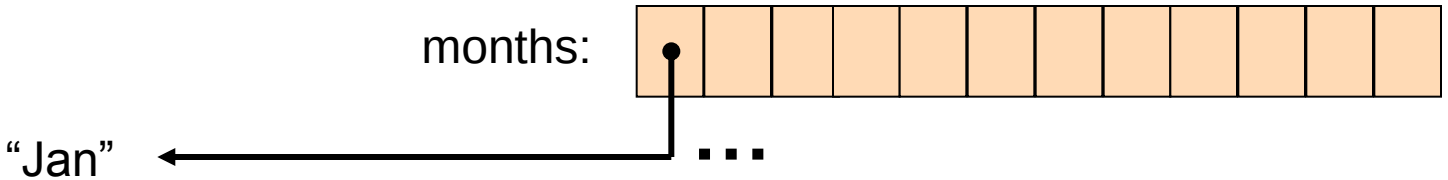
- Output:

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]);
}
```

- foreach:

```
for (String month: months) {
      System.out.println(month);
}
```

The variable month is assigned the next element in each iteration

# 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

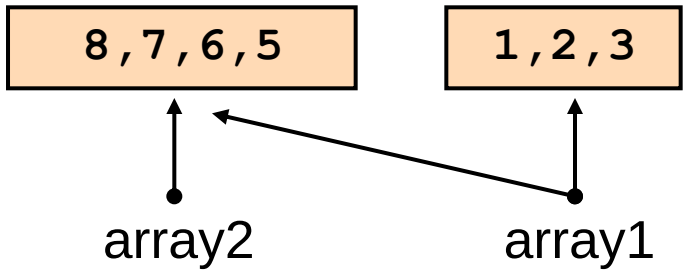
- Assume:

  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

```
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

# Question

- What is the output of the following code:

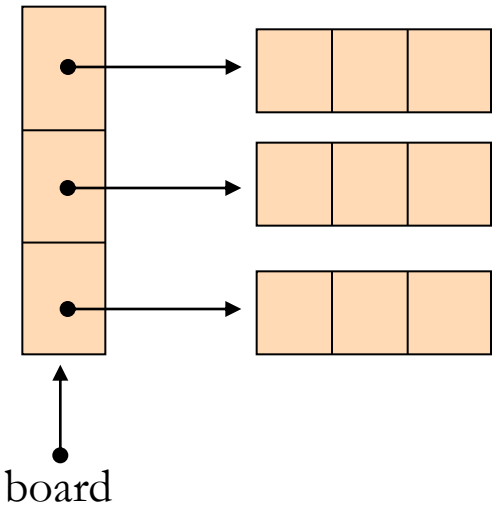
```
int[] odds = {1, 3, 5, 7, 9, 11, 13, 15};
int newOdds[] =
    Arrays.copyOfRange(odds, 1, odds.length);
for (int odd: newOdds) {
    System.out.print(odd + " ");
}
```

Output: 3 5 7 9 11 13 15

# 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];
```



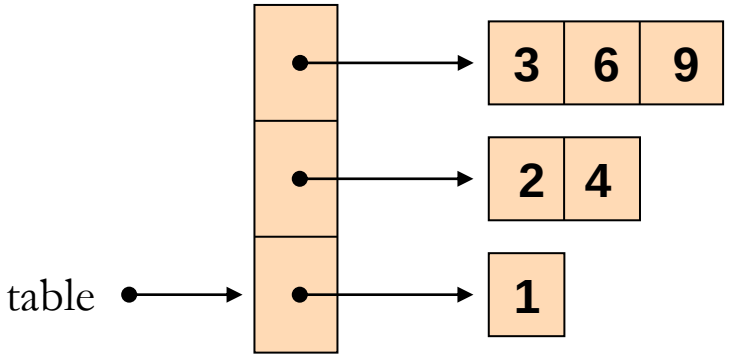

Or equivalently:

```
char[][] board = new char[3][3];
```

# 2D Arrays

- A more compact table:

```
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

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

Assumption:
n≥0

Compilation Error:
Unreachable Code

# Iterative Fibonacci

- A loop instead of a recursion

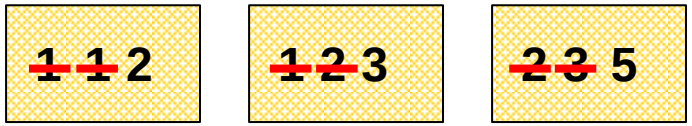
```
static int computeElement(int n) {
     if (n == 0 || n == 1)
           return 1;

     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 *<u>לחשב</u>* את האיברים וגם *<u>להדפיס</u>* אותם
  - כמו כן כדי לחשב איבר בסדרה איננו משתמשים *<u>בתוצאות שכבר חישבנו</u>* (של איברים 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);
```

# Compilation vs. Runtime Errors

- שגיאות קומפילציה (הידור): שגיאות שניתן "לתפוס" בעת קריאת הקובץ והפיכתו ל-bytecode ע"י המהדר
- דוגמאות:

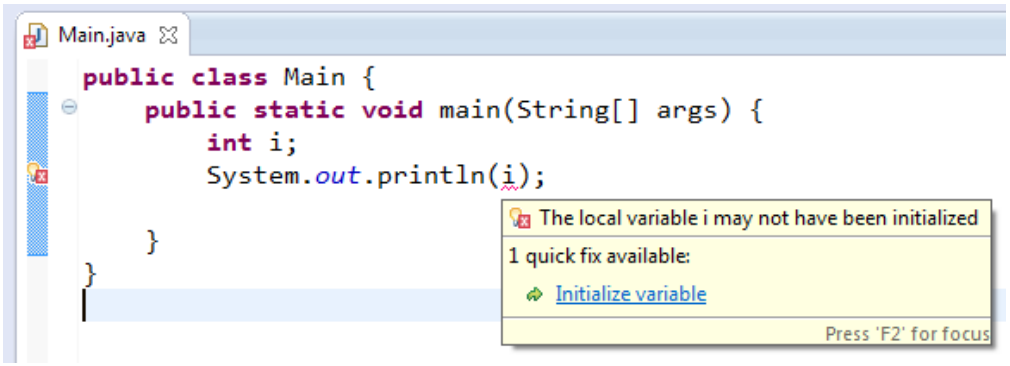
**Syntax error on token "Class", class expected**

```
Class MyClass {
    void f() {
        int n=10;

    void g() {
        int m = 20;
    }
}
```

**Syntax error, insert "}" to complete MethodBody**

```
...
int i;
System.out.println(i);
...
```




**<u>בדרך כלל קשורות ל:</u>**

**תחביר, תאימות טיפוסים, הגדרה לפני שימוש**

# Compilation vs. Runtime Errors

- שגיאות זמן ריצה: לא ניתן לדעת שתהיה שגיאה במקום ספציפי בזמן ההידור (קומפילציה)
- דוגמאות:

```
a = new int[20];
```

```
…
int a[] = new int[10];
…
a[15] = 10;
…
```

```
…
String s = null;
System.out.println(s.length())
…
```

```
Main.java
public class Main {
    public static void main(String[] args) {
        String s = null;
        System.out.println(s.length());
    }
}
```

```
Problems  Javadoc  Declaration  Console  Debug  Call Hierarchy
<terminated> Main [Java Application]
Exception in thread "main" java.lang.NullPointerException
        at Main.main(Main.java:4)
```

- מתקשר למנגנון החריגים (exceptions), עליו נלמד בהמשך

# Compilation vs. Runtime Errors

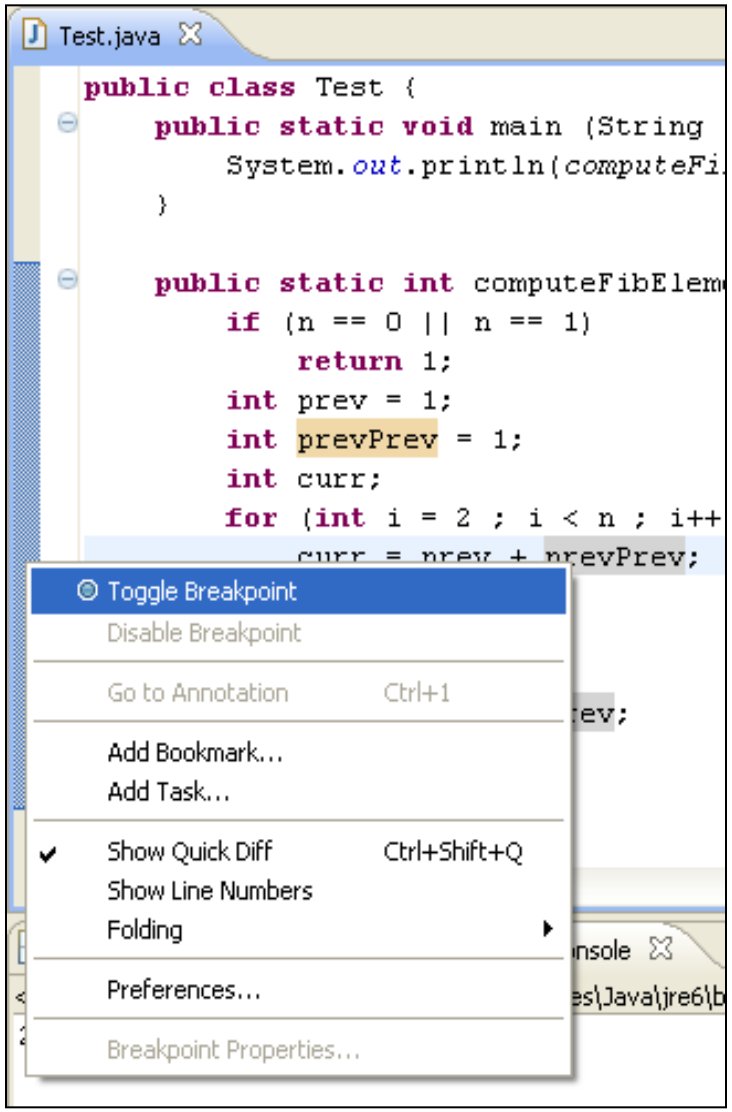
- האם יש עוד סוג של טעויות?
  - כן, הכי גרועות, טעויות לוגיות בתוכנית

```
public class Factorial {
   /** calculate x! **/
   public static int factorial(int x) {
      int f = 0;
      for (int i = 2; i <= x; i++)
         f = f * i;
      return f;
   }
}
```

# The Debugger

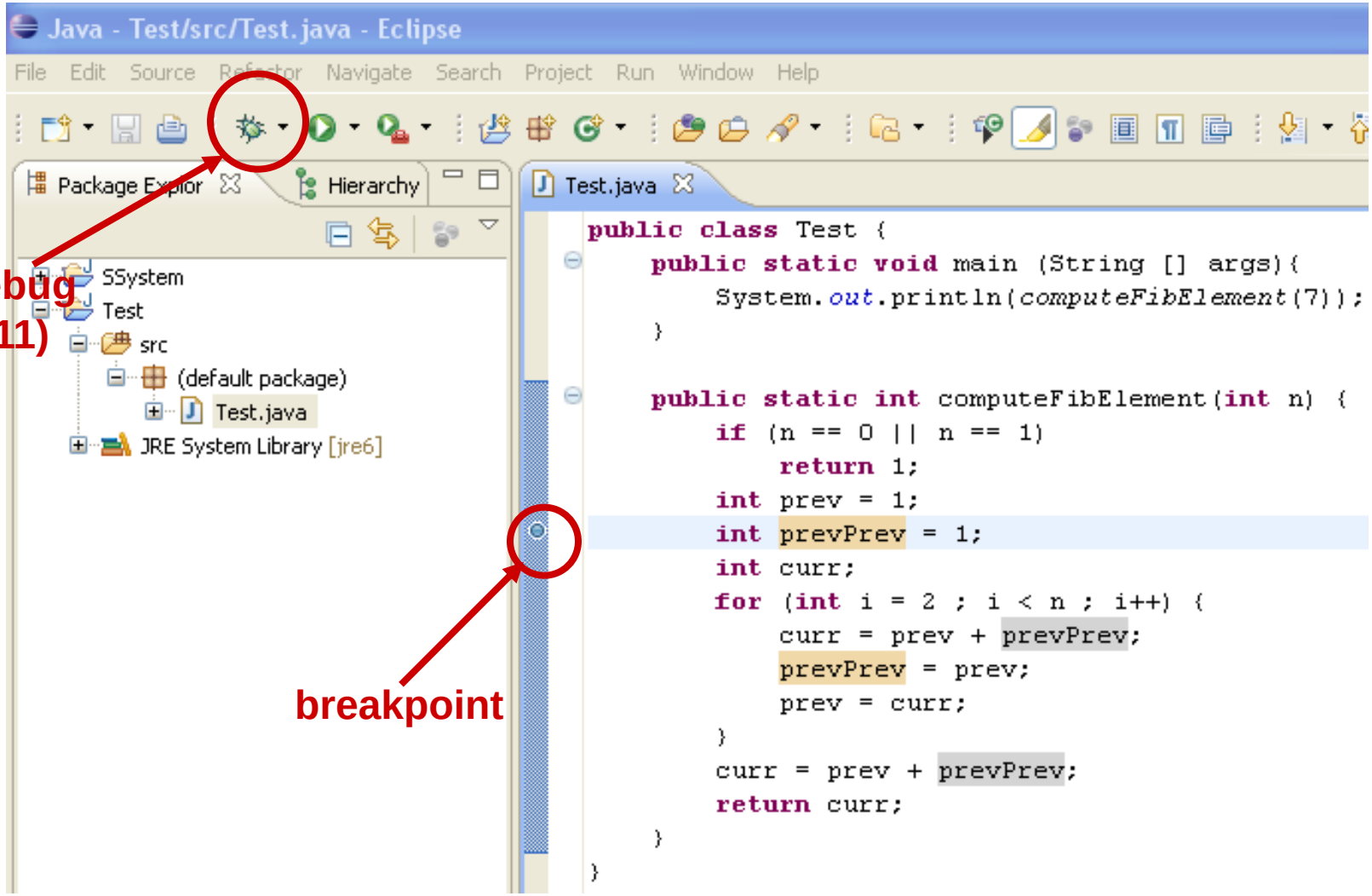
- Some programs may compile correctly, yet not produce the desirable results
- These programs are **valid** and **correct** Java programs, yet not the programs we meant to write!
- The debugger can be used to follow the program step by step and may help detecting bugs in an **already compiled** program

# Debugger – Add Breakpoint

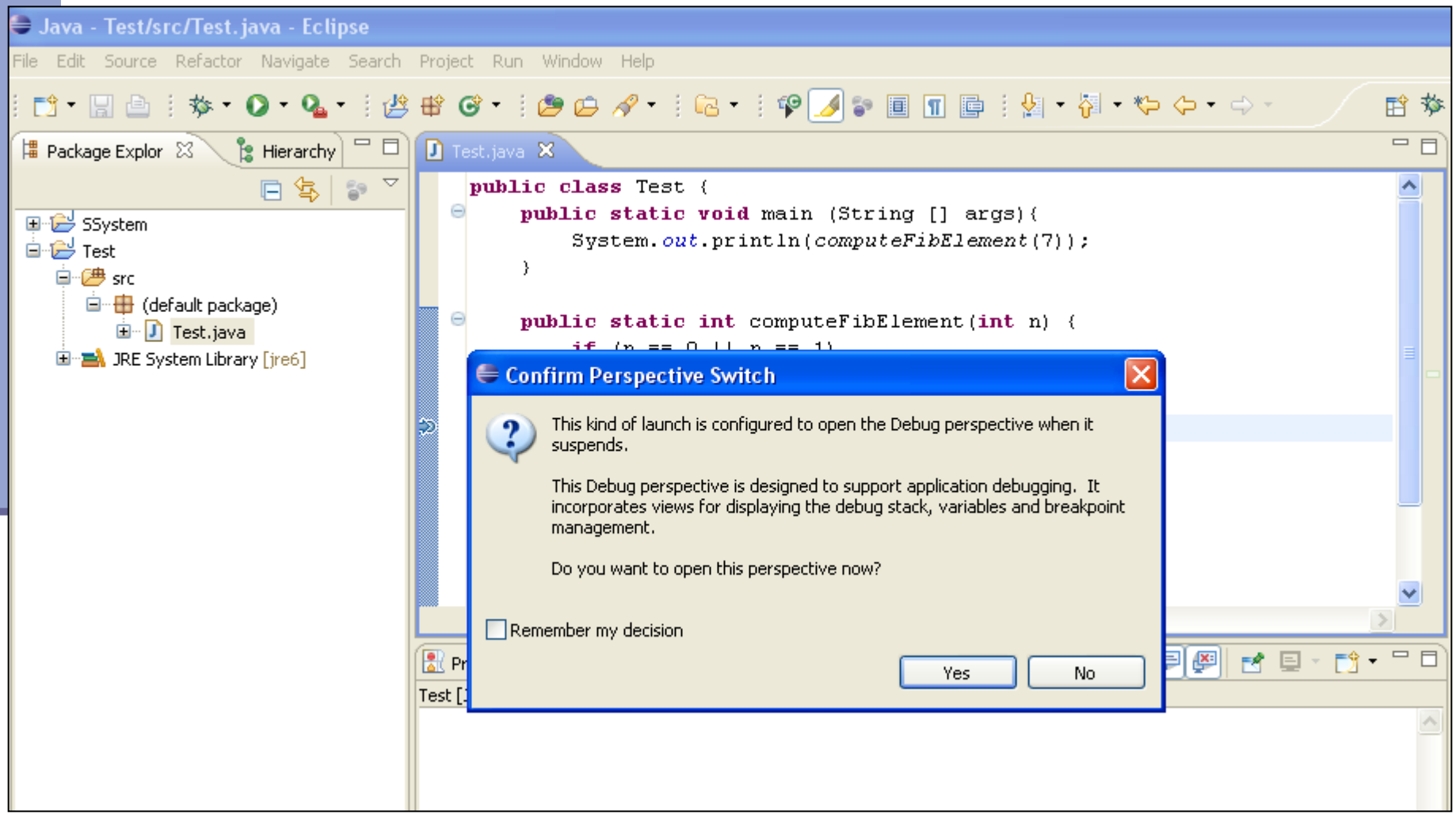



- Right click on the desired line
- “Toggle Breakpoint”

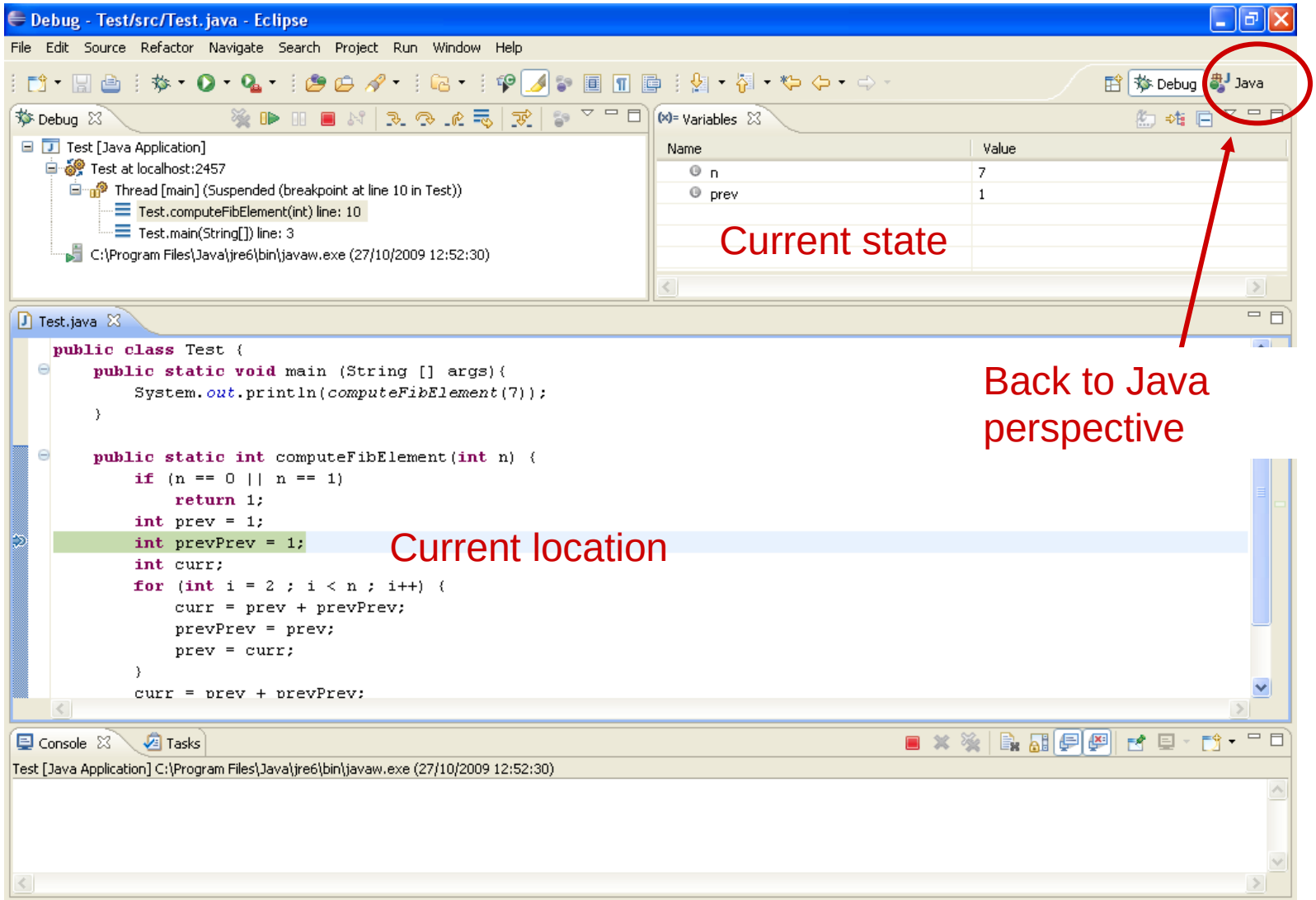
# Debugger – Start Debugging



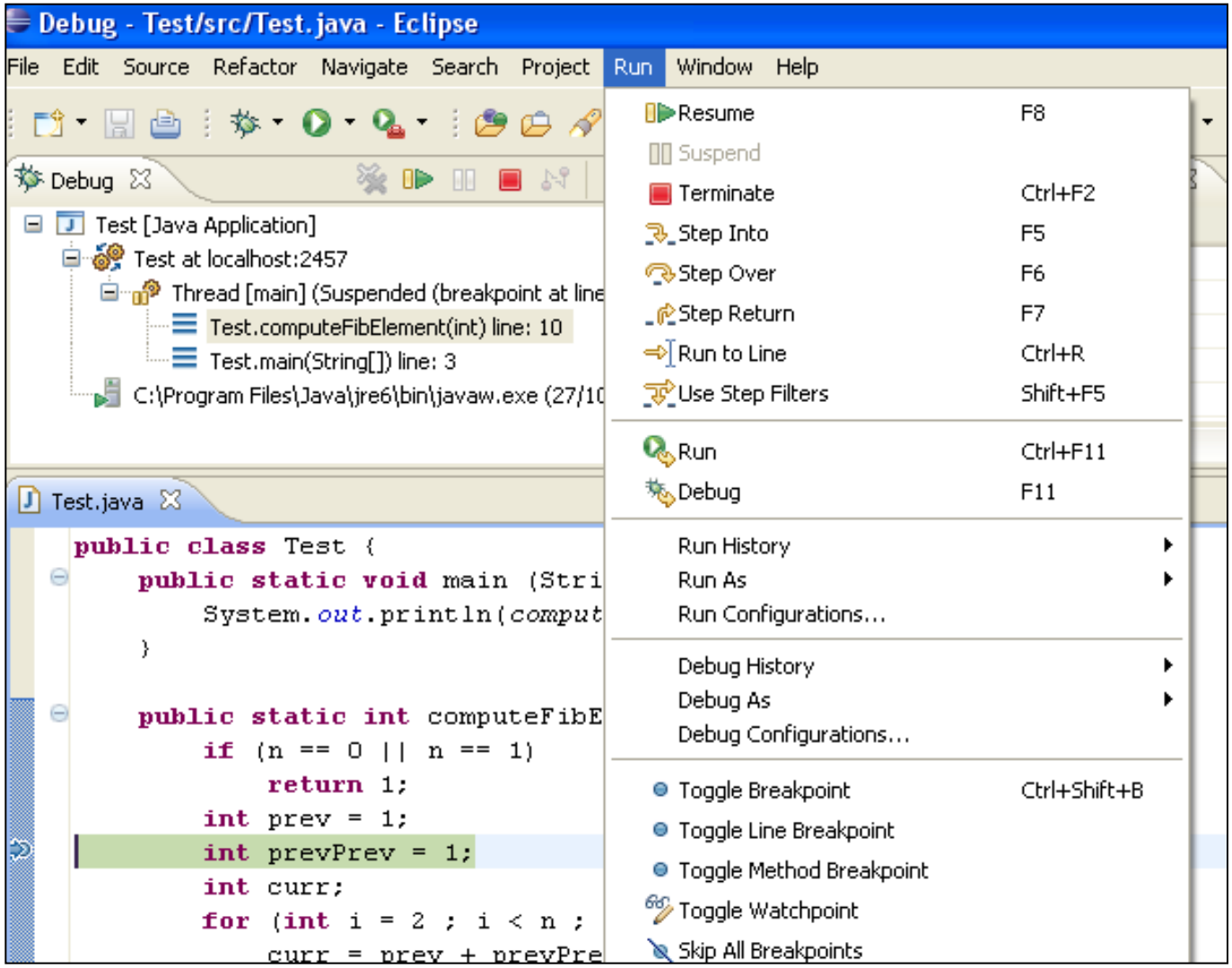
# Debugger – Debug Perspective



# Debugger – Debugging



# Debugger – Debugging



# Using the Debugger: Video Tutorial

- מצגות וידאו

http://eclipsetutorial.sourceforge.net/debugger.html

- מדריך עדכני יותר

http://www.vogella.com/tutorials/EclipseDebugging/article.html

- הקישורים נמצאים גם באתר הקורס

הסוף...