The interface between the client and the provider:

**precondition** - "the client sends a request to a provider."  
**postcondition** - "the client receives a response."  

**Conditions and Constraints according to Contracts**

A contract between a provider and a client:

- **precondition** - "precondition" - precondition - "postcondition" - "postcondition."  

3. **Method to check if infinity**

Example 1:

```java
/*
 * precondition: arr != null  
 * postcondition:
 * the contract is valid if and only if:
 * 1) arr.length > 0  
 * 2) arr contains only numbers (no NaN or Infinity)
 * 3) arr contains only numbers (no NaN or Infinity)
 * postcondition: Returns the minimal element in arr
 */
 public static double min(double[] arr) {
 double m = Double.POSITIVE_INFINITY;
 for (double x : arr) {
  m = (x < m ? x : m);
 }
 return m;
}
```

Example 2 (with a different contract):

```java
/*
 * precondition: arr != null  
 * postcondition:
 * If ((arr.length==0) || (arr contains only NaNs))
 * returns Infinity.  
 * Otherwise, returns the minimal value in arr.
 */
 public static double min(double[] arr) {
 double m = Double.POSITIVE_INFINITY;
 for (double x : arr) {
  m = (x < m ? x : m);
 }
 return m;
}
```

In comparison to Example 1:

- The contract allows more flexibility for the client.
(precondition) 4 (לא נא מởא)
/* precondition: true */
* postcondition: If ((arr==null) || (arr.length==0)) returns NaN
* Otherwise, if arr contains only NaN - returns NaN
* Otherwise, returns the minimal value in arr, ignoring any NaN
*/
public static double min(double[] arr) {
    double m = Double.POSITIVE_INFINITY;
    for (double x : arr) {
        if (Double.isNaN(x))
            return x;
        m = (x < m ? x : m);
    }
    return m;
}

Compilation vs. Runtime Errors

Class MyClass {
    void f() {
        int n = 10;
        void g() {
            int m = 20;
        }
    }
    public class T {
        /** calculate x! **/
        public static int factorial(int x) {
            int f = 0;
            for (int i = 2; i <= x; i++)
                f = f * i;
            return f;
        }
    }
}

(precondition) 5 (לא נא מởא)
/* precondition: true */
* postcondition: If (arr != null) && (arr.length > 0) && (arr contains only numbers) returns the minimal value in arr.
* Else, the return value is undefined.
*/
public static double min(double[] arr) {
    if (arr == null)
        return 0;
    int i; System.out.println(i);
    int[] a = new int[10];
    a[15] = 10;
    int[] b = new int[20];
    a = new int[20];
    a[15] = 10;
    int[] a1 = new int[10];
    a1[15] = 10;
    String s = null;
    System.out.println(s.length());
}

Compilation vs. Runtime Errors

Do you have any more errors?
/* calculate x! */
public static int factorial(int x) {
    int f = 0;
    for (int i = 2; i <= x; i++)
        f = f * i;
    return f;
}

Compilation vs. Runtime Errors

Does the code compile?
/* precondition: true */
* postcondition: If (arr != null) && (arr.length > 0) && (arr contains only numbers) returns the minimal value in arr.
* Else, the return value is undefined.
*/
public static double min(double[] arr) {
    if (arr == null)
        return 0;
    int i; System.out.println(i);
    int[] a = new int[10];
    a[15] = 10;
    int[] b = new int[20];
    a = new int[20];
    a[15] = 10;
    int[] a1 = new int[10];
    a1[15] = 10;
    String s = null;
    System.out.println(s.length());

Max Span

Given an array of integers, define the span of an integer as the number of elements
(containing the integer) between two consecutive appearances of the integer in the array.

Examples:
- The array [1, 2, 1, 1, 3] and the integer 1 – the span is 4
- The array [1, 4, 2, 1, 1, 4, 1, 4] and the integer 1 – the span is 7
- The array [1, 4, 2, 1, 1, 4, 1, 4, 1] and the integer 2 – the span is 1

Max Span

The maximum span over all integers in a given array will be the

Max of Span

We want to implement a function that given an array of integers
returns the maximum span of those integers.

Examples:
- The array [1, 2, 1, 1, 3] and the integer 1 – the span is 4
- The array [1, 4, 2, 1, 1, 4, 1, 4] and the integer 1 – the span is 7
- The array [1, 4, 2, 1, 1, 4, 1, 4, 1] and the integer 2 – the span is 1

We start working

- Open a new project named MaxSpan
- Start writing the check function to solve our problem

Test function

- We define a new class for the tests
  il.ac.tau.cs.sw1.maxspan.tests.TestMaxSpan

The first part - package


Now let’s write the test cases we want to check:

```java
test = new int[] {1, 2, 1, 1, 3};
test = new int[] {1, 4, 2, 1, 1, 4, 1, 4};
```

```java
int maxSpan = MaxSpan.maxSpan(array);
if (maxSpan != 4) {
    System.out.println(Arrays.toString(array) + " expected: 4, result: " + maxSpan);
} else {
    System.out.println(Arrays.toString(array) + " correct!");
}
```

Why is the type checker complaining?

- Does not recognize Arrays?
- Does not recognize MaxSpan?

```java
import java.util.Arrays;
import il.ac.tau.cs.sw1.maxspan.MaxSpan;
```

```java
null to be defined...
```

```java
import il.ac.tau.cs.sw1.maxspan.MaxSpan;
```

```java
null to be defined...
```

```java
null to be defined...
```

```java
null to be defined...
```

```java
null to be defined...
```

```java
null to be defined...
```

```java
null to be defined...
```

```java
null to be defined...
```

```java
null to be defined...
```
public static int maxSpan(int[] array) {
    int max = 0;
    for (int i = 0; i < array.length; i++) {
        int span = 0;
        for ( ; i >= j; j--) {
            if (array[i] == array[j]) {
                break;
            }
        }
        span = j - i + 1;
        if (max < span) {
            max = span;
        }
    }
    return max;
}

Public int maxSpan(int[] nums) {
    int max = 0;
    for (int value: values(nums)) {
        max = Math.max(max, span(value, nums));
    }
    return max;
}

private static int lastIndexOf(int value, int[] nums) {
    for (int i = nums.length - 1; i >= 0; i--) {
        if (nums[i] == value) {
            return i;
        }
    }
    return -1;
}

private static int firstIndexOf(int value, int[] nums) {
    int index = -1;
    for (int i = 0; i < nums.length; i++) {
        if (nums[i] == value) {
            index = i;
            break;
        }
    }
    return index;
}

private static int[] values(int[] nums) {
    int[] values = new int[nums.length];
    int nextIndex = 0;
    for (int i = 0; i < nums.length; i++) {
        if (!contains(values, nextIndex, nums[i])) {
            add(values, nextIndex++, nums[i]);
        }
    }
    return Arrays.copyOf(values, nextIndex);
}
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 – Start Debugging

Debugger – Add Breakpoint

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

Debugger – Debugging

Debugger – Debug Perspective

- Current state
- Back to Java perspective
- Current location

The Debugger

private static void add(int[] values, int position, int value) {
    values[position] = value;
}

private static boolean contains(int[] temp, int tempLength, int value) {
    for (int i = 0; i < tempLength; i++) {
        if (temp[i] == value) {
            return true;
        }
    }
    return false;
}
Using the Debugger: Video Tutorial

You can find excellent tutorials on using the debugger at http://eclipselibrary.sourceforge.net/debugger.html.*

Recommend watching at least the first four videos.*

The link also appears in the course materials on the development environment.

Debugger – Debugging