**Contracts**

A contract between a supplier and a customer defines:

- **Precondition** - "Precondition"
- **Postcondition** - "Postcondition"

A customer contract:

- Defines the terms of service for each service:
  - **Precondition** - "Precondition"  
  - **Postcondition** - "Postcondition"

**Preconditions**

- **Preconditions** define the supplier’s obligations.
- In most cases, these conditions describe situations in which the program in which the supplier must deliver the service is acceptable.
  - In simple and common cases, these conditions only affect the input passed to the service.
  - In general, these conditions also refer to the program’s state, such as global variables.

- The precondition can be a combination of several conditions (AND).

**Postconditions**

- **Postconditions** define the service’s obligations.
- If the precondition is met, the supplier must honor the postcondition.
- If the precondition is not met, the service is not guaranteed to complete:
  - Maybe the service will complete without errors.
  - Maybe the service will get stuck in an infinite loop.
  - Maybe the service will immediately terminate.
  - Maybe an incorrect value will be returned.
  - Maybe the service will complete without errors, but the program will terminate or get stuck after that.

**Example 1**

```java
public static double min1(double[] arr) {
    double m = Double.POSITIVE_INFINITY;
    for (double x : arr)
        m = (x < m ? x : m);
    return m;
}
```

The implementation does not check if the precondition is met:

- arr == null
- arr.length == 0
- arr contains only NaNs
- arr contains only infinite values

**Example 2**

```java
public static double min2(double[] arr) {
    double m = Double.POSITIVE_INFINITY;
    for (double x : arr)
        m = (x < m ? x : m);
    return m;
}
```

In comparison with Example 1, this contract is less strict for the customer.
Max Span

* This is the max span of a given sequence of integers.
* The span is defined as the number of consecutive elements between two occurrences of the same integer.
* The span is calculated as the difference between the positions of the two occurrences.

Example:
* Input: [1, 2, 2, 3]
  * Output: 2

Span

* This function finds the maximum span of a given sequence of integers.
* The maximum span is defined as the largest span in the sequence.

Example:
* Input: [1, 2, 3, 4, 5]
  * Output: 3

MaxSpan

* This class contains methods for calculating the maximum span of a given sequence of integers.
* The methods are defined in the MaxSpanTests package.

Test MaxSpan

* This is a test class for the MaxSpan class.
* The test cases are defined in the TestMaxSpan class.

Documentation:

Development:

Conclusion:
* The MaxSpan class provides methods for calculating the maximum span of a given sequence of integers.
* The class is useful for analyzing sequences and finding patterns in the data.

Notch in the code

* This is a notch in the code of the MaxSpan class.
* The notch is located in line 3 of the code.
Compilation vs. Runtime Errors

Syntax error on token "Class", class expected

```
public class MyClass {
  void f1() {
    short x = 5;
    short y = 10;
    short z = x * y;
  }

  void f2() {
    int m = 20;
    int n = 5;
    int i;
    System.out.println(i);
  }
}
```

Type mismatch: cannot convert from int to short

```
short z = x * y;
```

The local variable i may not have been initialized

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

We also need to adjust the output array size.

Conflicts with the array

```
int[] array = null;
int maxSpan;
array = new int[1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
maxSpan = Arrays.maxSpan(array);
if (maxSpan == 1) {
  System.out.println(Arrays.toString(array) + " expected: 1, result: " + maxSpan);
} else {
  System.out.println(Arrays.toString(array) + " expected: 7, result: " + maxSpan);
}
array = new int[1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
maxSpan = Arrays.maxSpan(array);
if (maxSpan == 1) {
  System.out.println(Arrays.toString(array) + " expected: 1, result: " + maxSpan);
} else {
  System.out.println(Arrays.toString(array) + " expected: 7, result: " + maxSpan);
}
array = new int[1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
maxSpan = Arrays.maxSpan(array);
if (maxSpan == 1) {
  System.out.println(Arrays.toString(array) + " expected: 1, result: " + maxSpan);
} else {
  System.out.println(Arrays.toString(array) + " expected: 7, result: " + maxSpan);
}
```
Compilation vs. Runtime Errors

- **Compilation Errors**: Cannot determine whether there is an error at a specific location during compilation (compile-time).
- **Examples**:
  - Calling a method that doesn’t exist (exceptions), as mentioned later.

```java
data = new int[10];
data[15] = 10;
d = new int[20];

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

**Debugger**
- **Add Breakpoint**
  - Right click on the desired line
  - "Toggle Breakpoint"

**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.
Using the Debugger: Video Tutorial

* תוכלו למצוא מצגות וידאו מצוינות המדריכות כיצד להשתמש ב debugger באתר:
  http://eclipsedtutor.sourceforge.net/debugger.html*

* מומלץ לצפות至少 בארבעת הסרטונים הראשונים*

* המורשת המורה בעיבוד הפרוטוקולים והאורחונים

* הקורס מומצא עם ב ישנם היכרות בבלוק על הפרוג'קט

Debugger – Debugging

Debugger – Debugging

Debugger – Debugging