JavaWiz Tutorial

Dr. Herbert Prähofer
Institut für Systemsoftware
Johannes Kepler Universität Linz

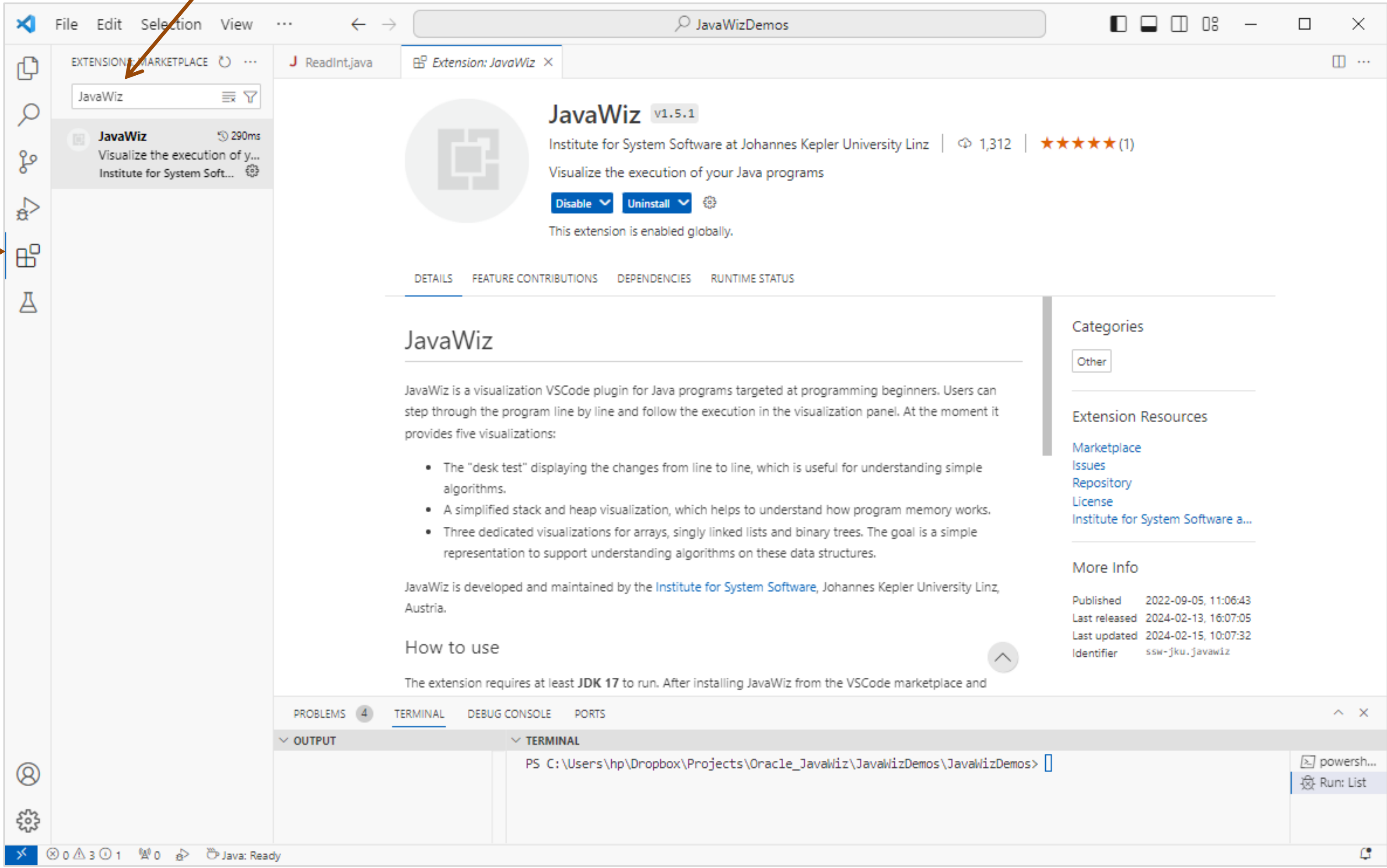
herbert.praehofer@jku.at

JavaWiz Visual Studio Code Extension

Install JavaWiz VS Code extension

JavaWiz

Extensions



JavaWiz v1.5.1

Institute for System Software at Johannes Kepler University Linz | 1,312 | ★★★★★ (1)

Visualize the execution of your Java programs

This extension is enabled globally.

JavaWiz

JavaWiz is a visualization VSCode plugin for Java programs targeted at programming beginners. Users can step through the program line by line and follow the execution in the visualization panel. At the moment it provides five visualizations:

- The "desk test" displaying the changes from line to line, which is useful for understanding simple algorithms.
- A simplified stack and heap visualization, which helps to understand how program memory works.
- Three dedicated visualizations for arrays, singly linked lists and binary trees. The goal is a simple representation to support understanding algorithms on these data structures.

JavaWiz is developed and maintained by the Institute for System Software, Johannes Kepler University Linz, Austria.

How to use

The extension requires at least **JDK 17** to run. After installing JavaWiz from the VSCode marketplace and

Categories

Other

Extension Resources

Marketplace
Issues
Repository
License
Institute for System Software a...

More Info

Published	2022-09-05, 11:06:43
Last released	2024-02-13, 16:07:05
Last updated	2024-02-15, 10:07:32
Identifier	ssw-jku.javawiz

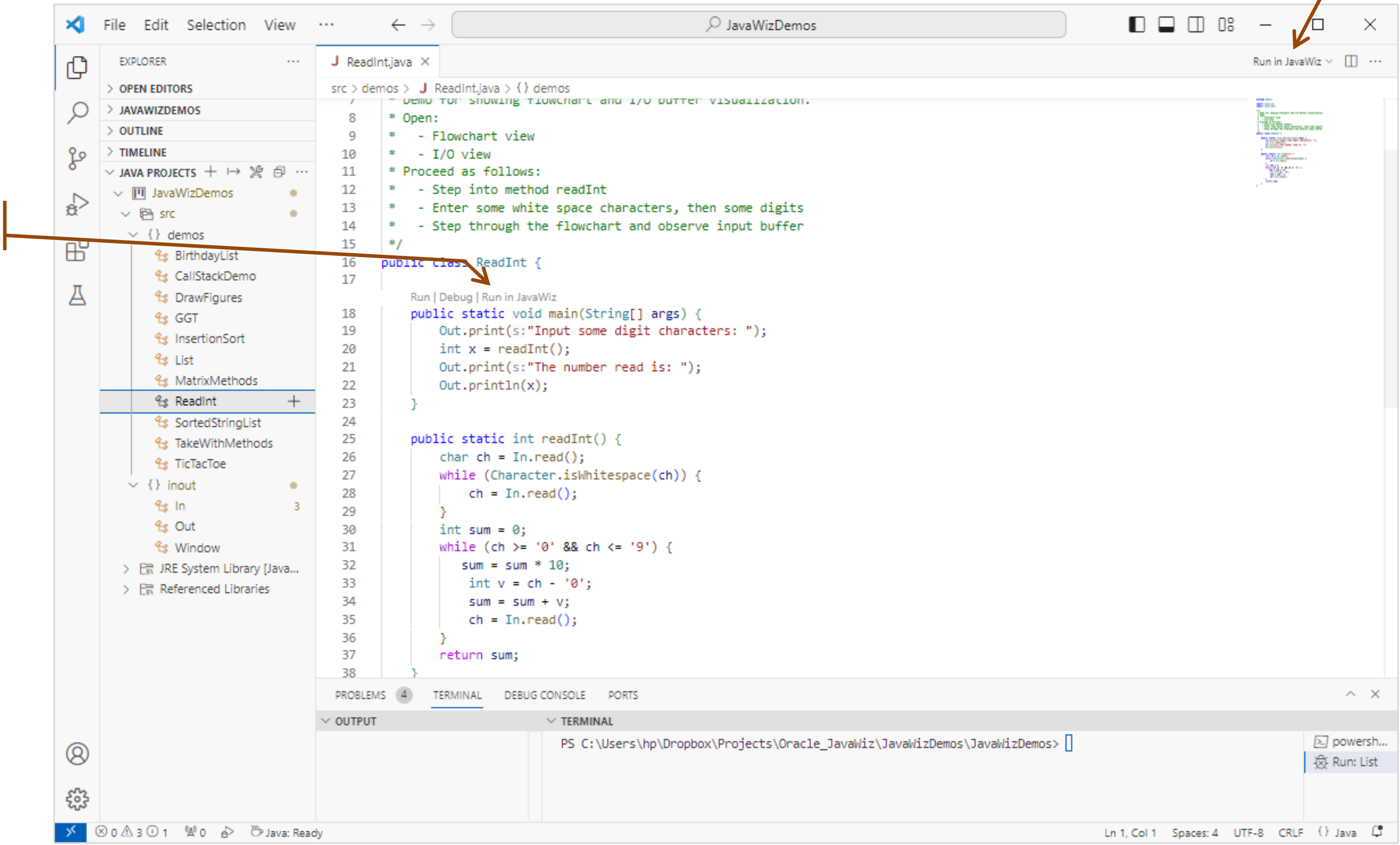
```
PS C:\Users\hp\Dropbox\Projects\Oracle_JavaWiz\JavaWizDemos\JavaWizDemos>
```

Start JavaWiz in VS Code

JavaWiz VS Code extension

- Run application with JavaWiz

Run in JavaWz

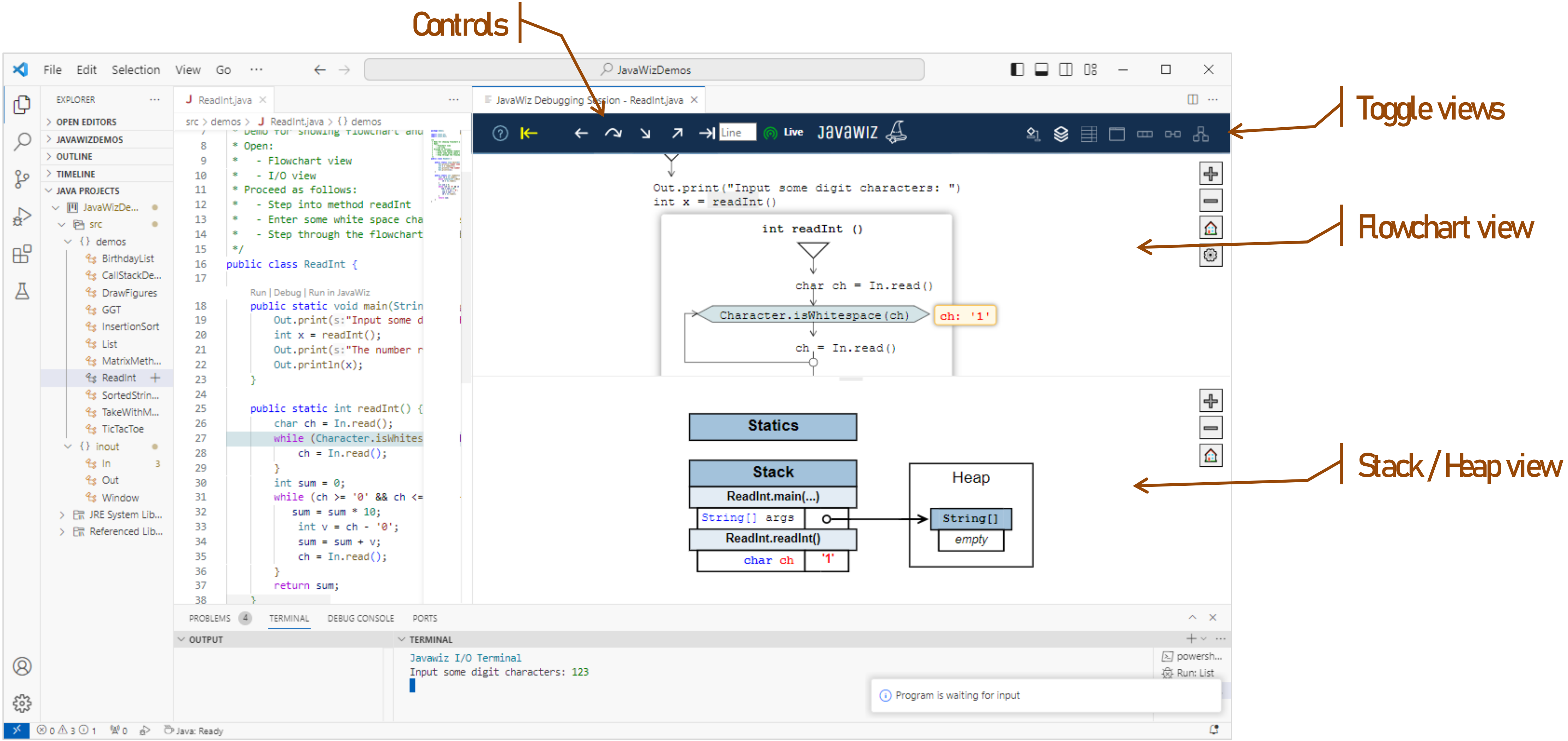


```java
 * Demo for showing flowchart and I/O buffer visualization.
 * Open:
 *   - Flowchart view
 *   - I/O view
 * Proceed as follows:
 *   - Step into method readInt
 *   - Enter some white space characters, then some digits
 *   - Step through the flowchart and observe input buffer
 */
public class ReadInt {

    Run | Debug | Run in JavaWiz
    public static void main(String[] args) {
        Out.print(s:"Input some digit characters: ");
        int x = readInt();
        Out.print(s:"The number read is: ");
        Out.println(x);
    }

    public static int readInt() {
        char ch = In.read();
        while (Character.isWhitespace(ch)) {
            ch = In.read();
        }
        int sum = 0;
        while (ch >= '0' && ch <= '9') {
            sum = sum * 10;
            int v = ch - '0';
            sum = sum + v;
            ch = In.read();
        }
        return sum;
    }
```

```
PS C:\Users\hp\Dropbox\Projects\Oracle_JavaWiz\JavaWizDemos\JavaWizDemos>
```

Run in JavaWz

JavaWiz Tool



Controls

Toggle views

Flowchart view

Stack / Heap view

```
 * Demo for showing flowchart and
 * Open:
 *   - Flowchart view
 *   - I/O view
 * Proceed as follows:
 *   - Step into method readInt
 *   - Enter some white space cha
 *   - Step through the flowchart
 */
public class ReadInt {

    public static void main(Strin
        Out.print(s:"Input some d
        int x = readInt();
        Out.print(s:"The number r
        Out.println(x);
    }

    public static int readInt() {
        char ch = In.read();
        while (Character.isWhites
            ch = In.read();
        }
        int sum = 0;
        while (ch >= '0' && ch <=
          sum = sum * 10;
            int v = ch - '0';
            sum = sum + v;
            ch = In.read();
        }
        return sum;
    }
```

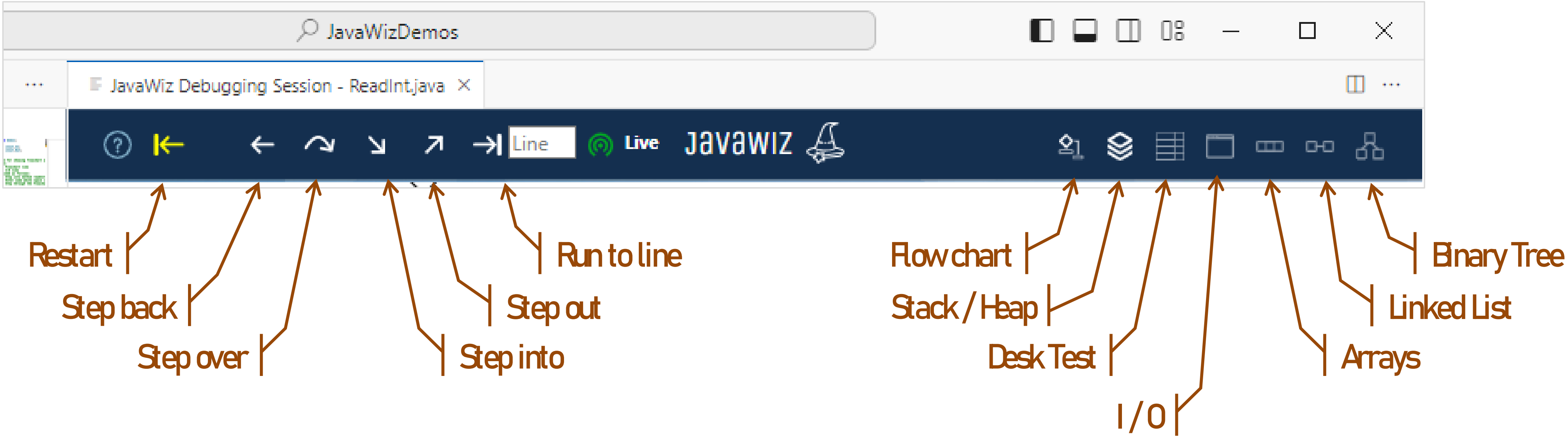
```
Out.print("Input some digit characters: ")
int x = readInt()
```

```
Javawiz I/O Terminal
Input some digit characters: 123
```

Program is waiting for input

JavaWiz Tool: Controls

Controls buttons and toggle views



JavaWiz Visualization Component: Desk Test

Demo Program: Histogram

Shows executed statements with variable values and conditions

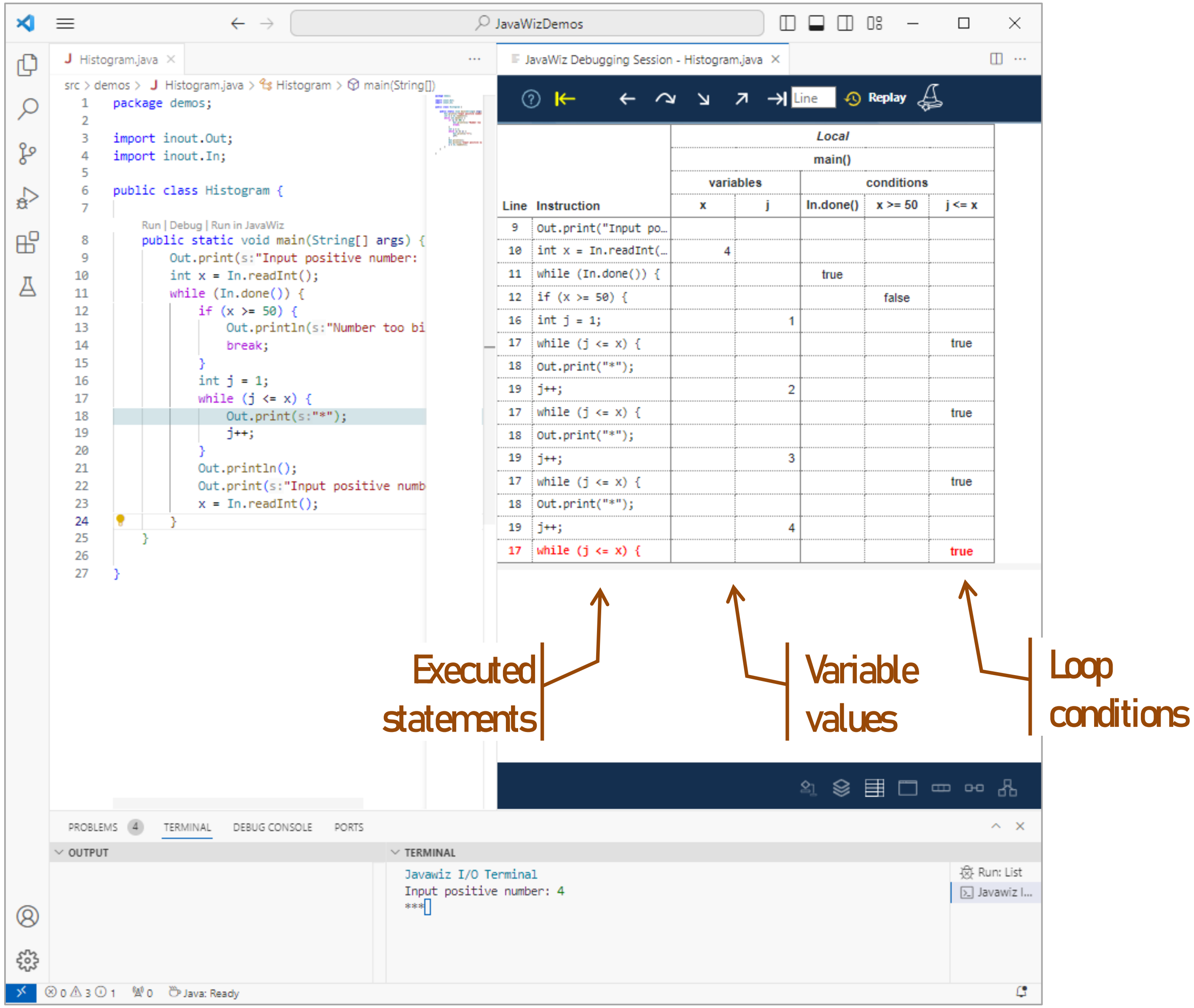
- Used to visualize control structures ifs and loops

Open Views

- Desk Test

Proceed as follows

- Step over statements
- Input positive integer values < 50
- Observe executed statements, variable values, and loop conditions in Desk Test view
- Break with input >= 50


JavaWizDemos
Histogram.java
src > demos > Histogram.java > Histogram > main(String[])
Run | Debug | Run in JavaWiz
JavaWiz Debugging Session - Histogram.java
Line
Replay
PROBLEMS 4
TERMINAL
DEBUG CONSOLE
PORTS
OUTPUT
TERMINAL
Run: List
Javawiz I...
0 3 1
0
Java: Ready

```java
package demos;

import inout.Out;
import inout.In;

public class Histogram {

    public static void main(String[] args) {
        Out.print(s:"Input positive number:
        int x = In.readInt();
        while (In.done()) {
            if (x >= 50) {
                Out.println(s:"Number too bi
                break;
            }
            int j = 1;
            while (j <= x) {
                Out.print(s:"*");
                j++;
            }
            Out.println();
            Out.print(s:"Input positive numb
            x = In.readInt();
        }
    }

}
```

		Local				
		main()				
		variables		conditions		
Line	Instruction	x	j	In.done()	x >= 50	j <= x
9	Out.print("Input po…					
10	int x = In.readInt(…	4				
11	while (In.done()) {			true		
12	if (x >= 50) {				false	
16	int j = 1;		1			
17	while (j <= x) {					true
18	Out.print("*");					
19	j++;		2			
17	while (j <= x) {					true
18	Out.print("*");					
19	j++;		3			
17	while (j <= x) {					true
18	Out.print("*");					
19	j++;		4			
17	while (j <= x) {					true

```
Javawiz I/O Terminal
Input positive number: 4
***
```

Executed statements

Variable values

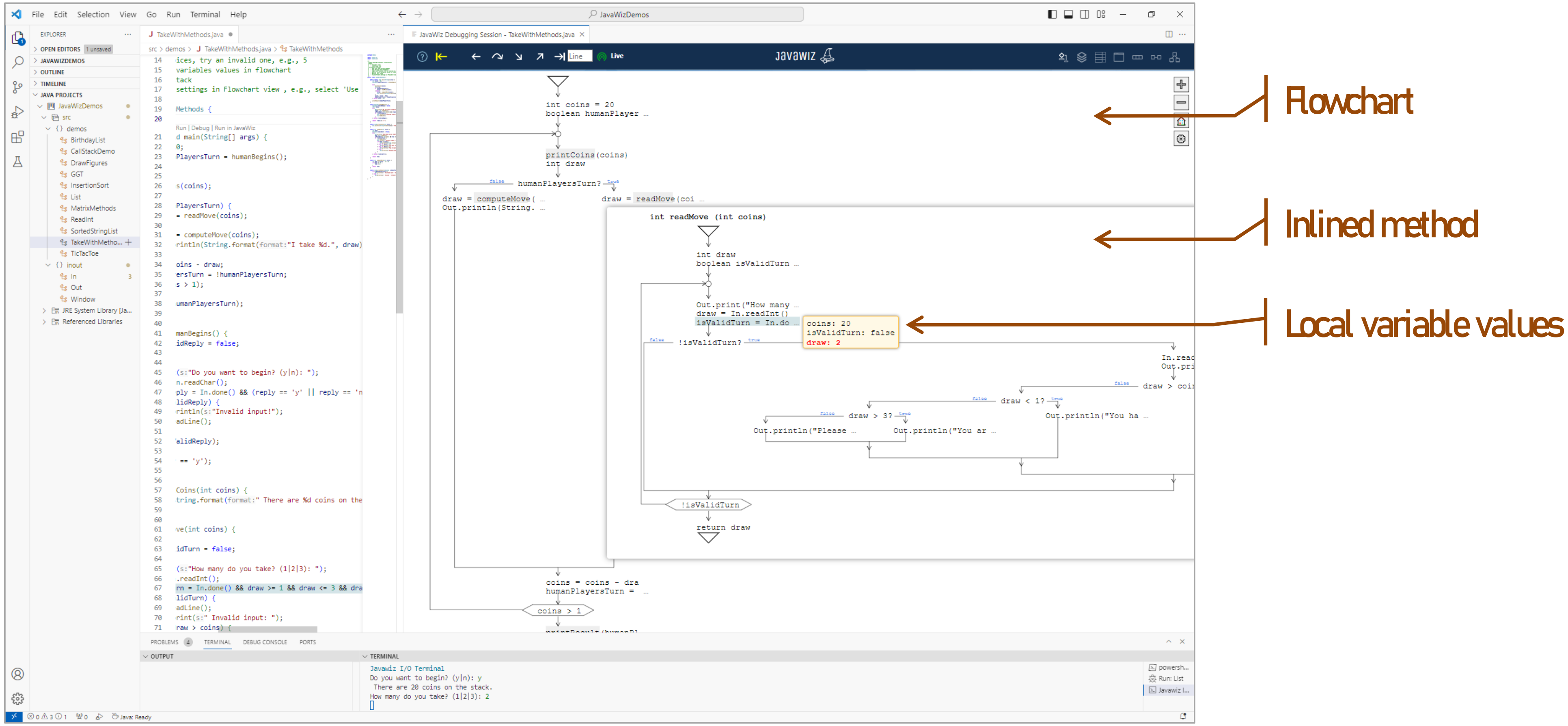
Loop conditions

JavaWiz Visualization Component: Flowchart [1/4]

Shows executed statements with variable values and conditions

- Use it for visualizing control flow and method calls

Demo Program: TakeWithMethods



```
ices, try an invalid one, e.g., 5
variables values in flowchart
tack
settings in Flowchart view , e.g., select 'Use

Methods {

Run | Debug | Run in JavaWiz
d main(String[] args) {
0;
PlayersTurn = humanBegins();


s(coins);

PlayersTurn) {
= readMove(coins);

= computeMove(coins);
rintln(String.format(format:"I take %d.", draw)

oins - draw;
ersTurn = !humanPlayersTurn;
s > 1);

umanPlayersTurn);


manBegins() {
idReply = false;


(s:"Do you want to begin? (y|n): ");
n.readChar();
ply = In.done() && (reply == 'y' || reply == 'n
lidReply) {
rintln(s:"Invalid input!");
adLine();

alidReply);

 == 'y');


Coins(int coins) {
tring.format(format:" There are %d coins on the


ve(int coins) {

idTurn = false;

(s:"How many do you take? (1|2|3): ");
.readInt();
rn = In.done() && draw >= 1 && draw <= 3 && dra
lidTurn) {
adLine();
rint(s:" Invalid input: ");
raw > coins) {
```

```
Javawiz I/O Terminal
Do you want to begin? (y|n): y
 There are 20 coins on the stack.
How many do you take? (1|2|3): 2
```

```
coins: 20
isValidTurn: false
draw: 2
```

Flowchart

Inlined method

Local variable values

JavaWiz Visualization Component: Flowchart [2/4]

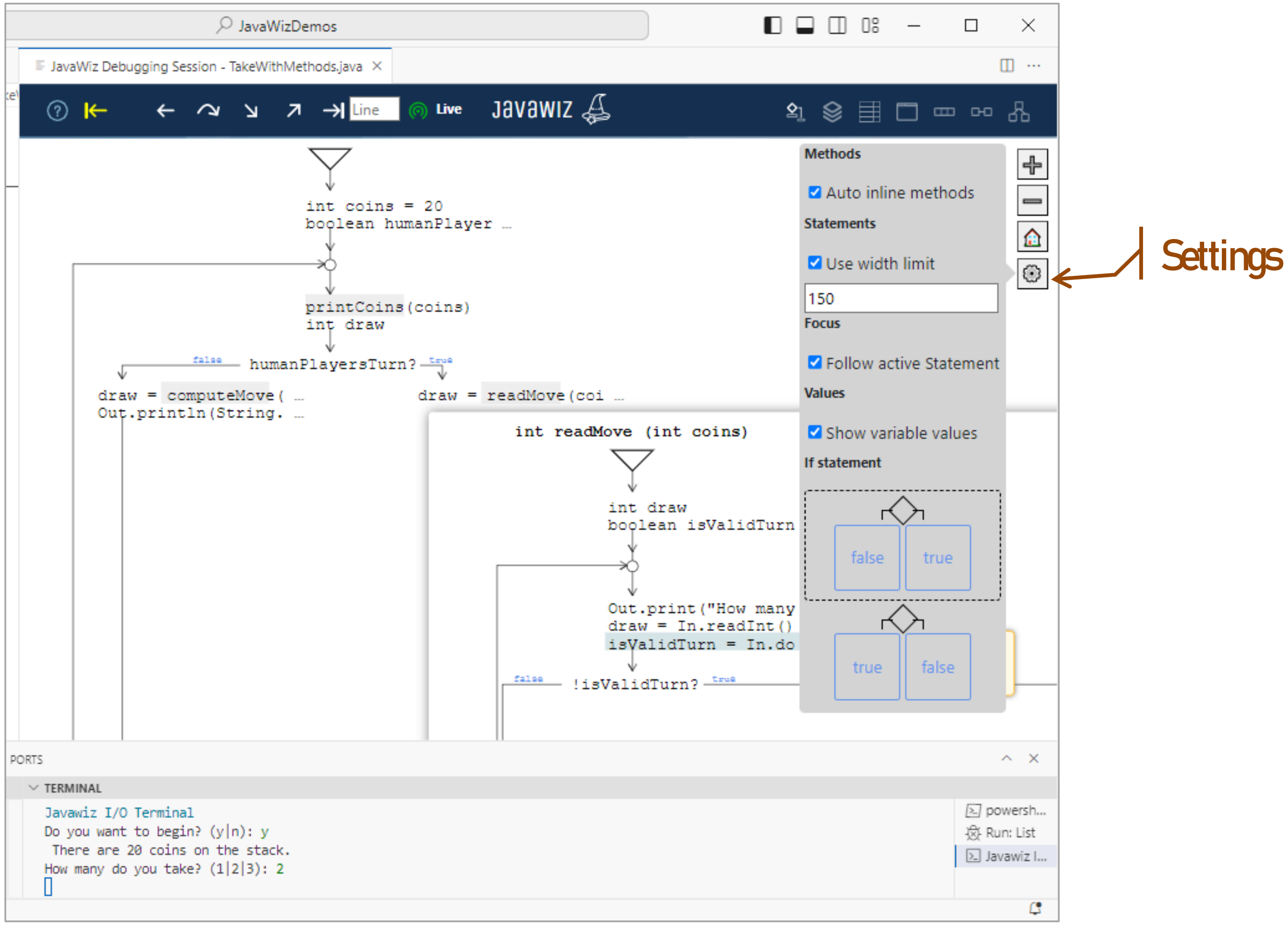
Demo Program: TakeWithMethods

Open:

- Flowchart view

Proceed as follows:

- Step into methods
- Step through the statements
- Input your choices
- Observe local variable values
- Try different settings, e.g.
  - □ Use with limit



```
int coins = 20
boolean humanPlayer …

printCoins(coins)
int draw

false  humanPlayersTurn?  true

draw = computeMove( …
Out.println(String. …

draw = readMove(coi …

int readMove (int coins)

int draw
boolean isValidTurn

Out.print("How many
draw = In.readInt()
isValidTurn = In.do

false  !isValidTurn?  true
```

Methods

☑ Auto inline methods

Statements

☑ Use width limit

150

Focus

☑ Follow active Statement

Values

☑ Show variable values

If statement

false | true

true | false

Settings

```
Javawiz I/O Terminal
Do you want to begin? (y|n): y
 There are 20 coins on the stack.
How many do you take? (1|2|3): 2
```

JavaWiz Visualization Component: Flowchart [3/4]

Show switch statements

Demo Program: SwitchDemo

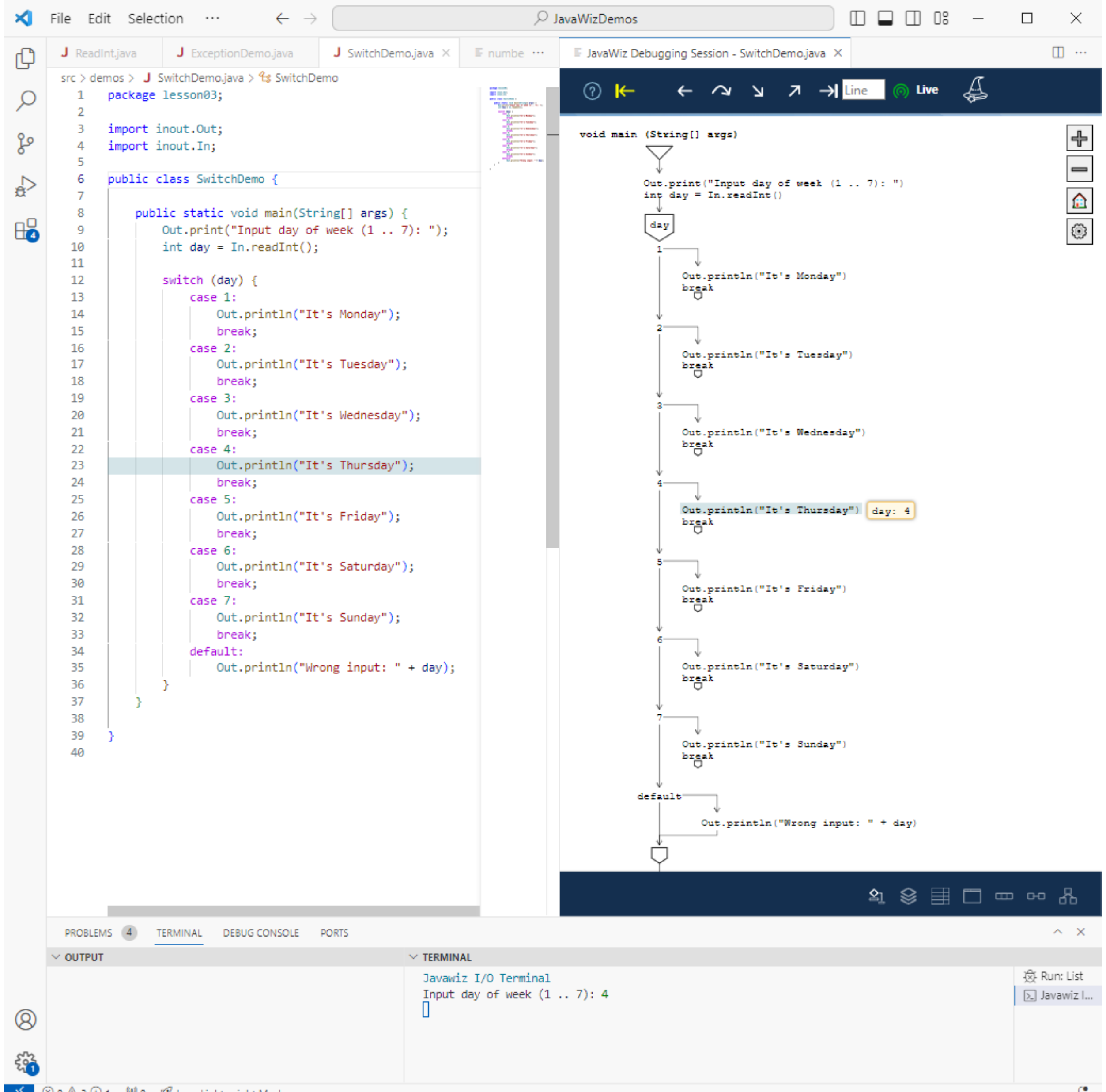
Open:

- Flowchart view

Proceed as follows:

- Input a number from 1 to 7
- Observe jump to case
- Observe break

- Input a number > 7
- Observe jump to default



File
Edit
Selection
JavaWizDemos
ReadInt.java
ExceptionDemo.java
SwitchDemo.java
numbe
JavaWiz Debugging Session - SwitchDemo.java
src > demos > SwitchDemo.java > SwitchDemo
Line
Live
void main (String[] args)
Out.print("Input day of week (1 .. 7): ")
int day = In.readInt()
day
1
Out.println("It's Monday")
break
2
Out.println("It's Tuesday")
break
3
Out.println("It's Wednesday")
break
4
Out.println("It's Thursday")
day: 4
break
5
Out.println("It's Friday")
break
6
Out.println("It's Saturday")
break
7
Out.println("It's Sunday")
break
default
Out.println("Wrong input: " + day)
PROBLEMS 4
TERMINAL
DEBUG CONSOLE
PORTS
OUTPUT
TERMINAL
Run: List
Javawiz I...

```java
package lesson03;

import inout.Out;
import inout.In;

public class SwitchDemo {

    public static void main(String[] args) {
        Out.print("Input day of week (1 .. 7): ");
        int day = In.readInt();

        switch (day) {
            case 1:
                Out.println("It's Monday");
                break;
            case 2:
                Out.println("It's Tuesday");
                break;
            case 3:
                Out.println("It's Wednesday");
                break;
            case 4:
                Out.println("It's Thursday");
                break;
            case 5:
                Out.println("It's Friday");
                break;
            case 6:
                Out.println("It's Saturday");
                break;
            case 7:
                Out.println("It's Sunday");
                break;
            default:
                Out.println("Wrong input: " + day);
        }
    }

}
```

```
Javawiz I/O Terminal
Input day of week (1 .. 7): 4
```

JavaWiz Visualization Component: Flowchart [4/4]

Show exception handling

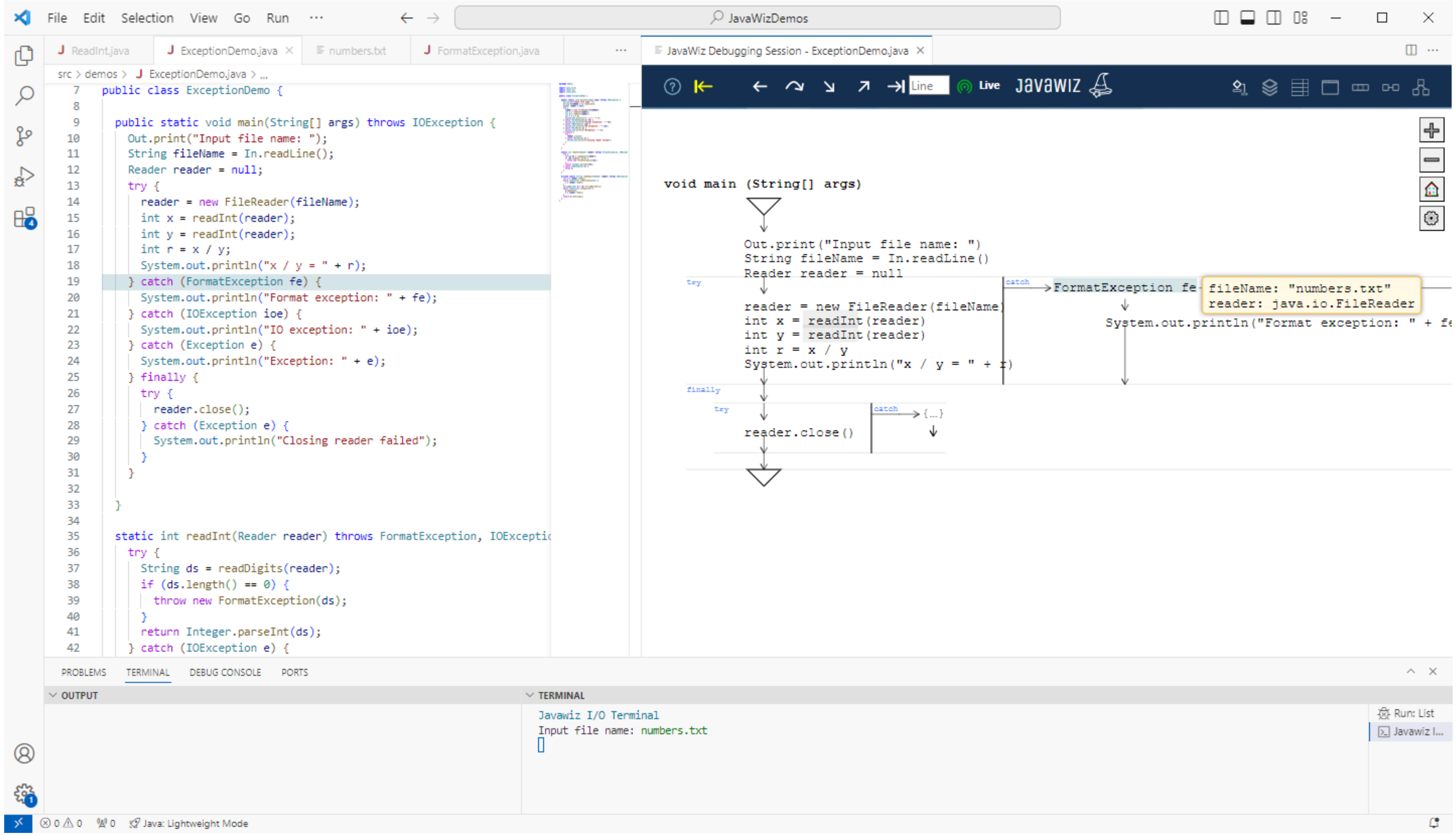
Demo Program: ExceptionsDemo

Open:

- Flowchart view

Proceed as follows:

- Input file name "numbers.txt"
- Observe exception in second readInt
- Observe execution of finally block



```java
public class ExceptionDemo {

  public static void main(String[] args) throws IOException {
    Out.print("Input file name: ");
    String fileName = In.readLine();
    Reader reader = null;
    try {
      reader = new FileReader(fileName);
      int x = readInt(reader);
      int y = readInt(reader);
      int r = x / y;
      System.out.println("x / y = " + r);
    } catch (FormatException fe) {
      System.out.println("Format exception: " + fe);
    } catch (IOException ioe) {
      System.out.println("IO exception: " + ioe);
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    } finally {
      try {
        reader.close();
      } catch (Exception e) {
        System.out.println("Closing reader failed");
      }
    }

  }

  static int readInt(Reader reader) throws FormatException, IOExceptic
    try {
      String ds = readDigits(reader);
      if (ds.length() == 0) {
        throw new FormatException(ds);
      }
      return Integer.parseInt(ds);
    } catch (IOException e) {
```

```
void main (String[] args)

Out.print("Input file name: ")
String fileName = In.readLine()
Reader reader = null

reader = new FileReader(fileName)
int x = readInt(reader)
int y = readInt(reader)
int r = x / y
System.out.println("x / y = " + r)

FormatException fe
System.out.println("Format exception: " + fe

reader.close()
```

```
fileName: "numbers.txt"
reader: java.io.FileReader
```

```
Javawiz I/O Terminal
Input file name: numbers.txt
```

JavaWiz Visualization Component: Stack

Showing call stack

- Use it for
  - □ visualizing call stack
  - □ showing difference of static and local variables

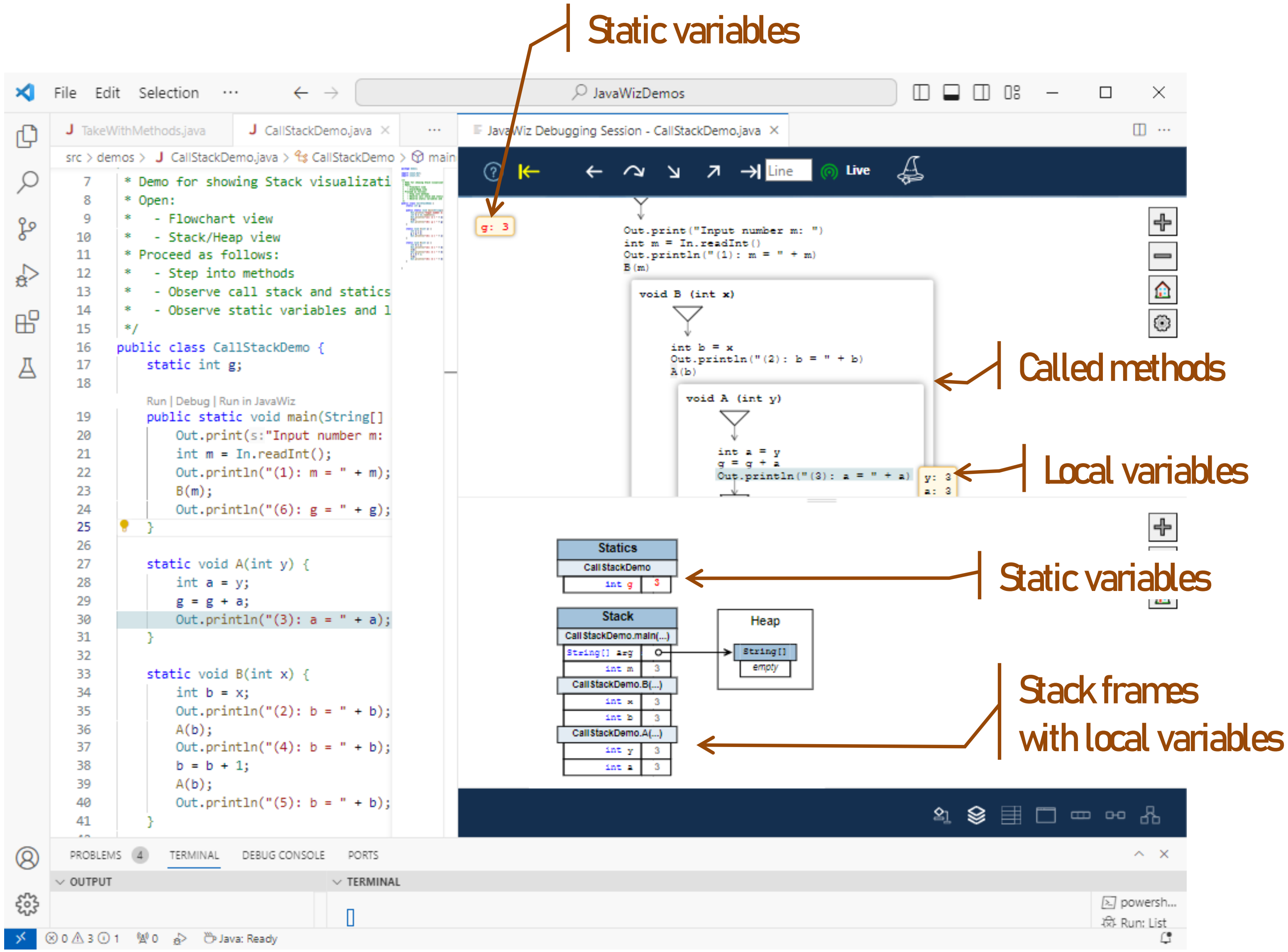
Demo Program: CallStackDemo

Open views:

- Flowchart
- Stack/Heap view

Proceed as follows:

- Step into methods
- Observe call stack with frames and statics in Stack/Heap visualization
- Observe static variables and local variables



```
 * Demo for showing Stack visualizati
 * Open:
 *   - Flowchart view
 *   - Stack/Heap view
 * Proceed as follows:
 *   - Step into methods
 *   - Observe call stack and statics
 *   - Observe static variables and l
 */
public class CallStackDemo {
    static int g;

    Run | Debug | Run in JavaWiz
    public static void main(String[]
        Out.print(s:"Input number m:
        int m = In.readInt();
        Out.println("(1): m = " + m);
        B(m);
        Out.println("(6): g = " + g);
    }

    static void A(int y) {
        int a = y;
        g = g + a;
        Out.println("(3): a = " + a);
    }

    static void B(int x) {
        int b = x;
        Out.println("(2): b = " + b);
        A(b);
        Out.println("(4): b = " + b);
        b = b + 1;
        A(b);
        Out.println("(5): b = " + b);
    }
```

```
g: 3

Out.print("Input number m: ")
int m = In.readInt()
Out.println("(1): m = " + m)
B(m)

void B (int x)
    int b = x
    Out.println("(2): b = " + b)
    A(b)

    void A (int y)
        int a = y
        g = g + a
        Out.println("(3): a = " + a)      y: 3
                                          a: 3
```

Statics	
CallStackDemo	
int g	3

Stack	
CallStackDemo.main(...)	
String[] arg	○
int m	3
CallStackDemo.B(...)	
int x	3
int b	3
CallStackDemo.A(...)	
int y	3
int a	3

Heap

String[]
empty

Static variables

Called methods

Local variables

Static variables

Stack frames with local variables

JavaWiz Visualization Component: Heap

Showing call heap

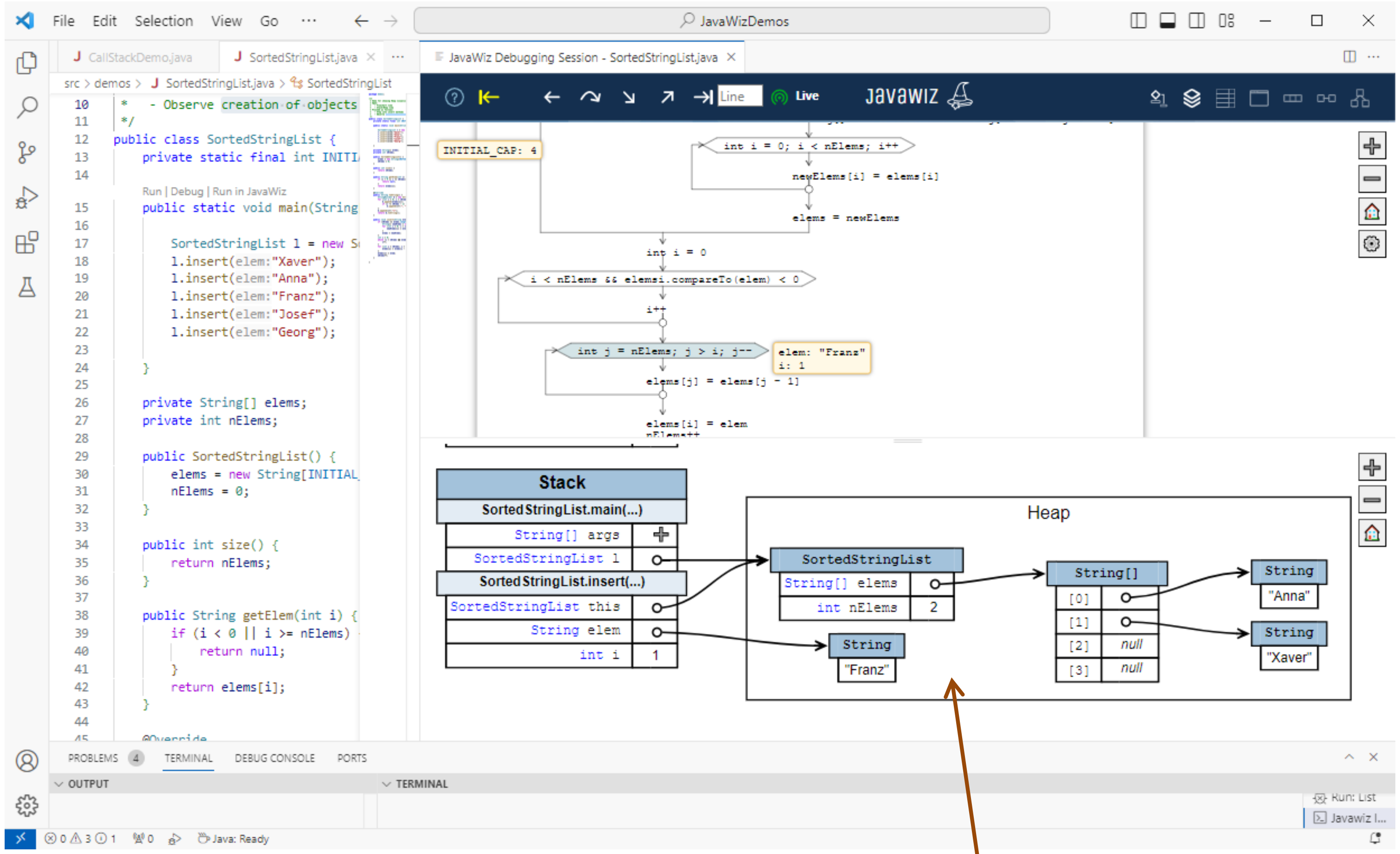
- Use it for
  - □ visualizing heap
  - □ explaining references
- Demo Program: **SortedStringList**

Open views:

- Flowchart
- Stack/Heap

Proceed as follows:

- Step into insert methods
- Observe creation of objects on heap
- Observe references to objects



```
 *   - Observe creation of objects
 */
public class SortedStringList {
    private static final int INITI

    Run | Debug | Run in JavaWiz
    public static void main(String

        SortedStringList l = new S
        l.insert(elem:"Xaver");
        l.insert(elem:"Anna");
        l.insert(elem:"Franz");
        l.insert(elem:"Josef");
        l.insert(elem:"Georg");

    }

    private String[] elems;
    private int nElems;

    public SortedStringList() {
        elems = new String[INITIAL_
        nElems = 0;
    }

    public int size() {
        return nElems;
    }

    public String getElem(int i) {
        if (i < 0 || i >= nElems)
            return null;
        }
        return elems[i];
    }

    @Override
```

Heap objects with references

JavaWiz Visualization Component: Arrays [1/2]

Showing array algorithms

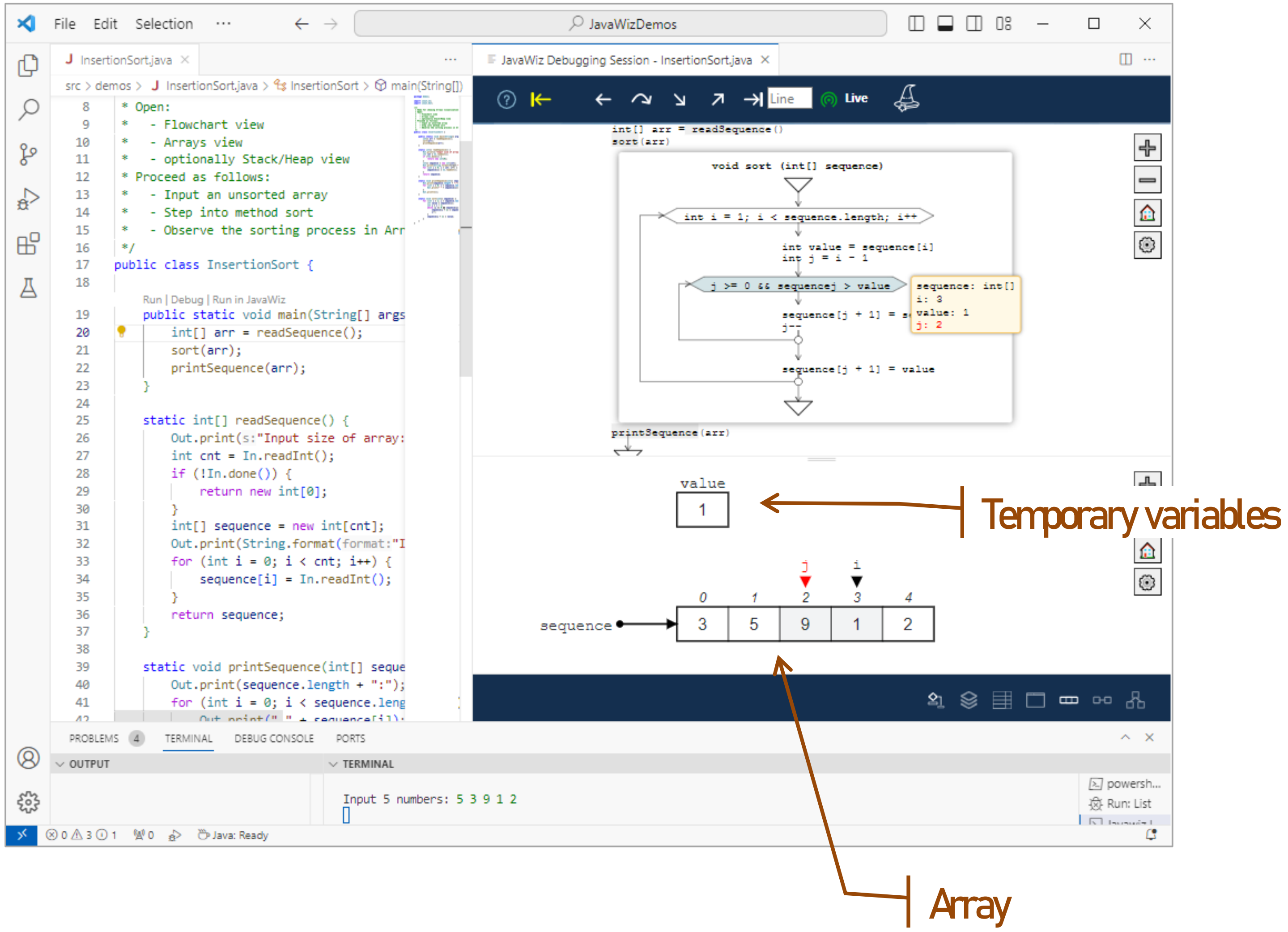
- Demo Program: **InsertionSort**

Open views:

- Flowchart
- Arrays
- optionally Stack/Heap

Proceed as follows:

- Input an unsorted array
- Step into method sort
- Observe the sorting process in Arrays view Step into insert methods
- Observe index variables



File
Edit
Selection
JavaWizDemos
InsertionSort.java
src > demos > InsertionSort.java > InsertionSort > main(String[])
JavaWiz Debugging Session - InsertionSort.java
Line
Live
Run | Debug | Run in JavaWiz

```
 * Open:
 *   - Flowchart view
 *   - Arrays view
 *   - optionally Stack/Heap view
 * Proceed as follows:
 *   - Input an unsorted array
 *   - Step into method sort
 *   - Observe the sorting process in Arr
 */
public class InsertionSort {

    public static void main(String[] args
        int[] arr = readSequence();
        sort(arr);
        printSequence(arr);
    }

    static int[] readSequence() {
        Out.print(s:"Input size of array:
        int cnt = In.readInt();
        if (!In.done()) {
            return new int[0];
        }
        int[] sequence = new int[cnt];
        Out.print(String.format(format:"I
        for (int i = 0; i < cnt; i++) {
            sequence[i] = In.readInt();
        }
        return sequence;
    }

    static void printSequence(int[] seque
        Out.print(sequence.length + ":");
        for (int i = 0; i < sequence.leng
```

int[] arr = readSequence()
sort(arr)
void sort (int[] sequence)
int i = 1; i < sequence.length; i++
int value = sequence[i]
int j = i - 1
j >= 0 && sequencej > value
sequence[j + 1] = se
j--
sequence[j + 1] = value
printSequence(arr)

```
sequence: int[]
i: 3
value: 1
j: 2
```

0	1	2	3	4
3	5	9	1	2

```
Input 5 numbers: 5 3 9 1 2
```

Temporary variables

Array

JavaWiz Visualization Component: Arrays [2/2]

Showing array algorithms

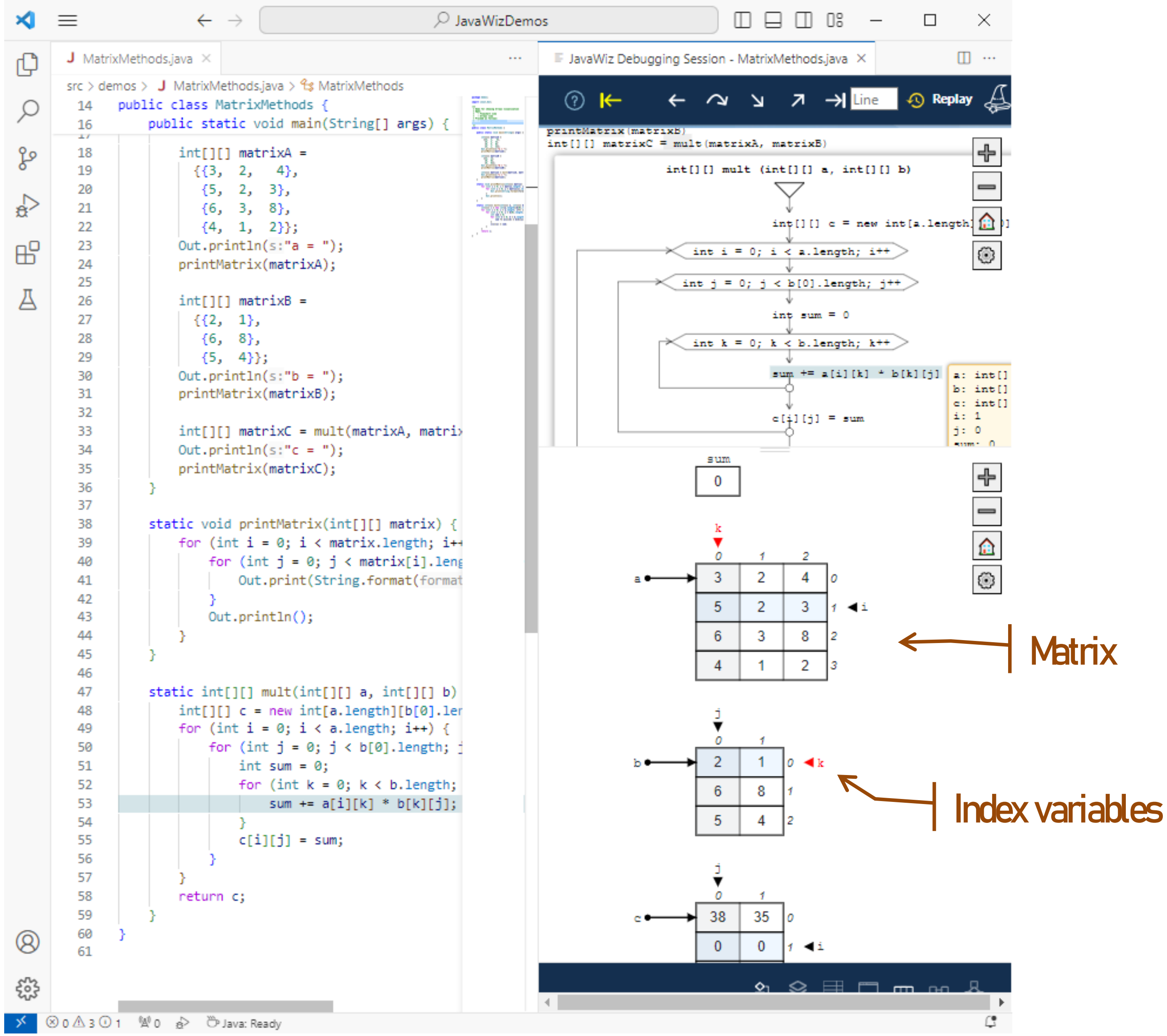
- Visualizing two-dimensional arrays
- Demo Program: **MatrixMethods**

Open views:

- Flowchart
- Arrays

Proceed as follows:

- Step into method mult
- Observe the multiplication process in Arrays view
- Observe index variables



JavaWizDemos
MatrixMethods.java
src > demos > MatrixMethods.java > MatrixMethods
JavaWiz Debugging Session - MatrixMethods.java
Line
Replay

```java
public class MatrixMethods {
    public static void main(String[] args) {

        int[][] matrixA =
          {{3,  2,   4},
           {5,  2,  3},
           {6,  3,  8},
           {4,  1,  2}};
        Out.println(s:"a = ");
        printMatrix(matrixA);

        int[][] matrixB =
          {{2,  1},
           {6,  8},
           {5,  4}};
        Out.println(s:"b = ");
        printMatrix(matrixB);

        int[][] matrixC = mult(matrixA, matri
        Out.println(s:"c = ");
        printMatrix(matrixC);
    }

    static void printMatrix(int[][] matrix) {
        for (int i = 0; i < matrix.length; i++
            for (int j = 0; j < matrix[i].leng
                Out.print(String.format(format
            }
            Out.println();
        }
    }

    static int[][] mult(int[][] a, int[][] b)
        int[][] c = new int[a.length][b[0].ler
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < b[0].length; j
                int sum = 0;
                for (int k = 0; k < b.length;
                    sum += a[i][k] * b[k][j];
                }
                c[i][j] = sum;
            }
        }
        return c;
    }
}
```

```
printMatrix(matrixB)
int[][] matrixC = mult(matrixA, matrixB)
```

int[][] mult (int[][] a, int[][] b)
int[][] c = new int[a.length][b[0]]
int i = 0; i < a.length; i++
int j = 0; j < b[0].length; j++
int sum = 0
int k = 0; k < b.length; k++
sum += a[i][k] * b[k][j]
c[i][j] = sum

```
a: int[]
b: int[]
c: int[]
i: 1
j: 0
sum: 0
```

```
⊗ 0 ⚠ 3 ⓘ 1   0   Java: Ready
```

JavaWiz Visualization Component: Linked List

Showing linked list algorithms

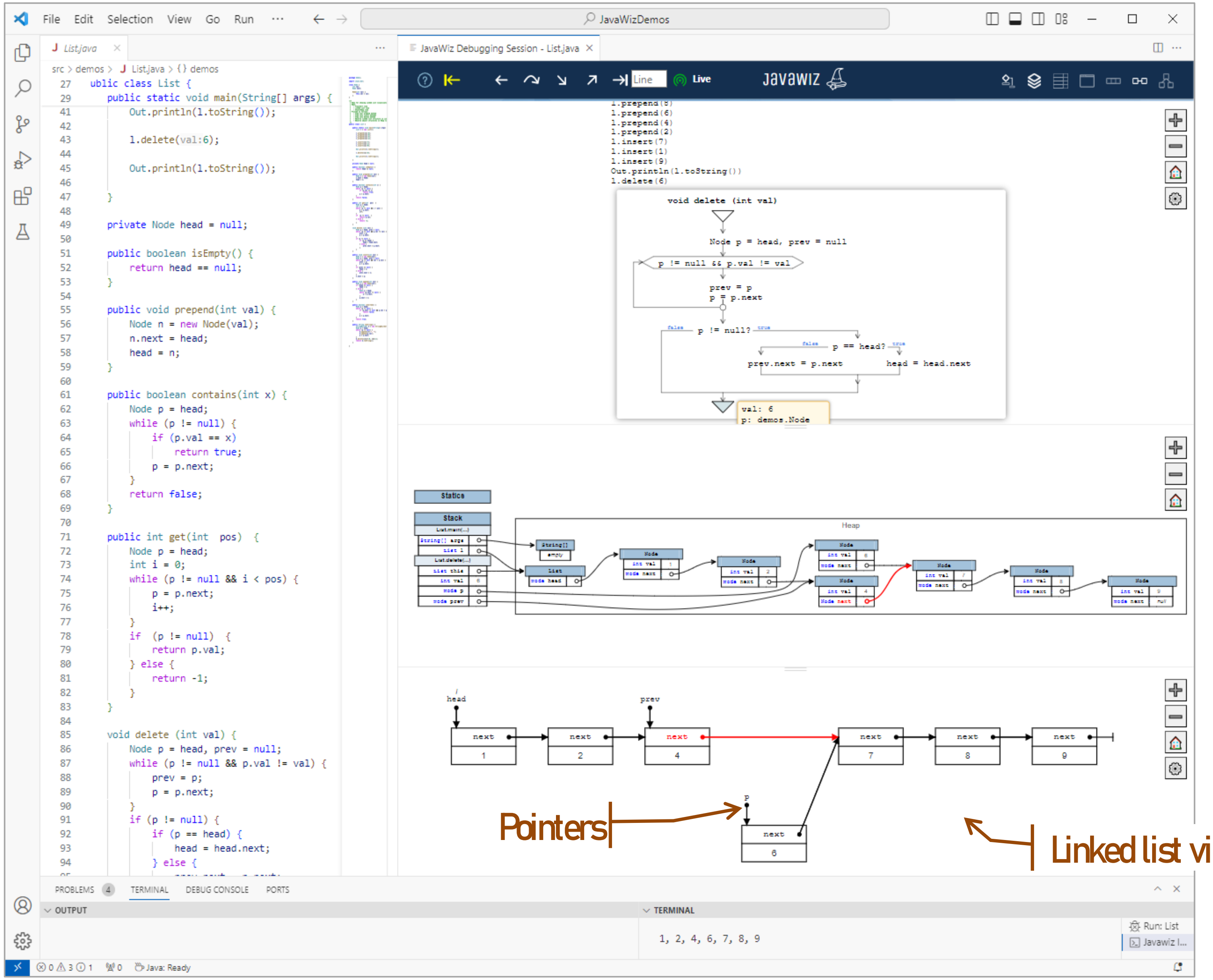
- Visualizing linked list structures
- Demo Program: **List**

Open views:

- Flowchart
- Stack / Heap
- Linked List

Proceed as follows:

- Step over prepend method
- Step into insert methods
- Step into delete method
- Observe linked list structure in Linked List view
- Observe pointers to nodes
- Observe object structures in Heap view



```java
ublic class List {
    public static void main(String[] args) {
        Out.println(l.toString());

        l.delete(val:6);

        Out.println(l.toString());

    }

    private Node head = null;

    public boolean isEmpty() {
        return head == null;
    }

    public void prepend(int val) {
        Node n = new Node(val);
        n.next = head;
        head = n;
    }

    public boolean contains(int x) {
        Node p = head;
        while (p != null) {
            if (p.val == x)
                return true;
            p = p.next;
        }
        return false;
    }

    public int get(int  pos)  {
        Node p = head;
        int i = 0;
        while (p != null && i < pos) {
            p = p.next;
            i++;
        }
        if  (p != null)  {
            return p.val;
        } else {
            return -1;
        }
    }

    void delete (int val) {
        Node p = head, prev = null;
        while (p != null && p.val != val) {
            prev = p;
            p = p.next;
        }
        if (p != null) {
            if (p == head) {
                head = head.next;
            } else {
```

```
l.prepend(6)
l.prepend(4)
l.prepend(2)
l.insert(7)
l.insert(1)
l.insert(9)
Out.println(l.toString())
l.delete(6)
```

```
void delete (int val)
Node p = head, prev = null
p != null && p.val != val
prev = p
p = p.next
p != null?   false / true
p == head?   false / true
prev.next = p.next
head = head.next
```

```
val: 6
p: demos.Node
```

```
1, 2, 4, 6, 7, 8, 9
```

Pointers

Linked list view

JavaWiz Visualization Component: I / O

Showing input

- Visualizing input buffer
- Demo Program: **ReadInt**

Open views:

- Flowchart
- I / O

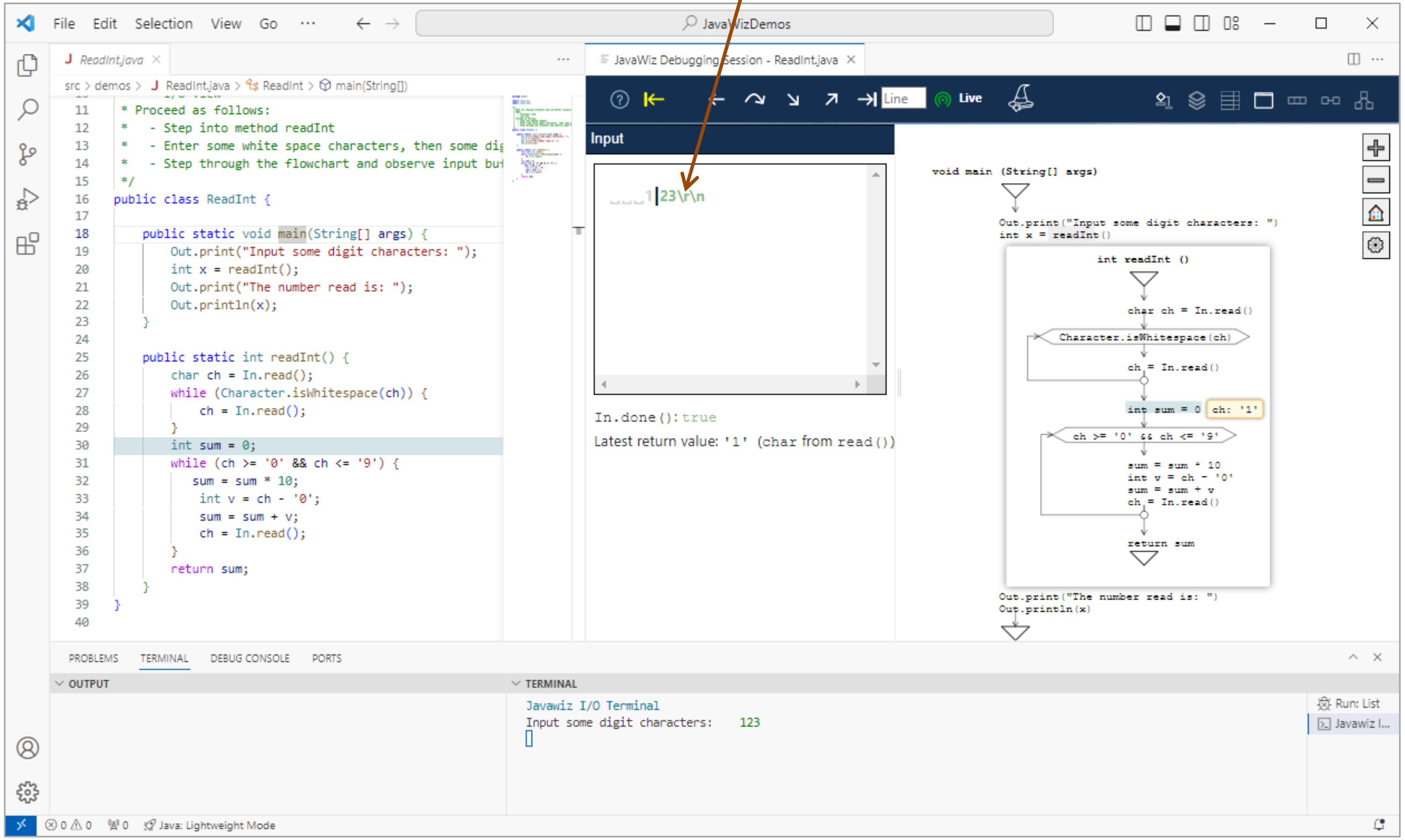
Proceed as follows:

- Step into readInt method
- Input some blanks, then some digits, then 'Enter', e.g.,

  ␣␣␣123↲

- Observe input buffer with cursor

Input buffer



```
 * Proceed as follows:
 *   - Step into method readInt
 *   - Enter some white space characters, then some dig
 *   - Step through the flowchart and observe input bu
 */
public class ReadInt {

    public static void main(String[] args) {
        Out.print("Input some digit characters: ");
        int x = readInt();
        Out.print("The number read is: ");
        Out.println(x);
    }

    public static int readInt() {
        char ch = In.read();
        while (Character.isWhitespace(ch)) {
            ch = In.read();
        }
        int sum = 0;
        while (ch >= '0' && ch <= '9') {
           sum = sum * 10;
            int v = ch - '0';
            sum = sum + v;
            ch = In.read();
        }
        return sum;
    }
}
```

Input

```
␣␣␣123\r\n
```

In.done(): true

Latest return value: '1' (char from read())

```
Javawiz I/O Terminal
Input some digit characters:    123
```