Supplement II.C: Teaching/Learning Java Effectively with NetBeans

For Introduction to Java Programming
By Y. Daniel Liang

0 Introduction

Supplement II.B, “NetBeans Tutorial,” gives a brief tutorial on how to use NetBeans. NetBeans is not only a powerful Java program development tool, but it is also a valuable pedagogical tool for teaching and learning Java programming. This supplement shows how to use NetBeans effectively with the text.

The supplement is written for instructors, but it is also useful to students.

1 Important Tips

The objective of the course is to teach Java, not NetBeans. NetBeans is a complex and powerful tool. All you need for this course, however, is a minimum set of features that enable students to create, compile, run, and debug programs. So students should avoid exploring unnecessary features.

If your students follow the instructions in Supplement II.B, “NetBeans Tutorial,” or the instructions from you, students can master all essential skills in thirty minutes. It is important that your students adhere to the instructions to avoid frustrating mistakes. If a mistake is made, simply read the instructions and restart from scratch.

2 NetBeans as a Valuable Pedagogical Tool

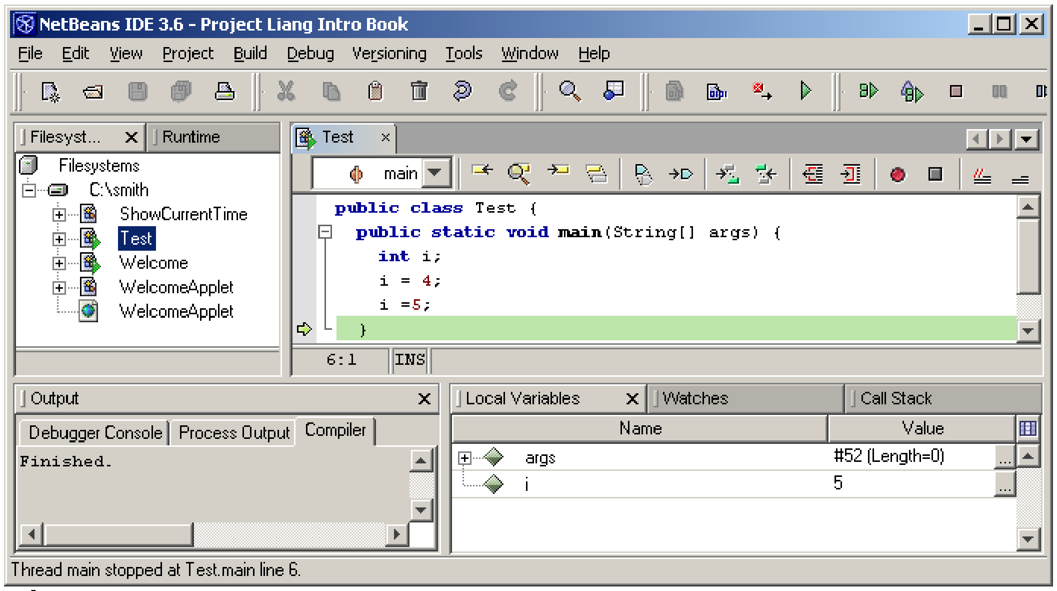
The following sections demonstrate how to utilize NetBeans in the first seven chapters.

2.1 Using NetBeans in Chapter 1

After Listing 1.1, you can start to cover how to create, compile, and run a program in NetBeans. You may also introduce how to use NetBeans online help.

2.2 Using NetBeans in Chapter 2

You may start to introduce debugging when you cover variables. You can use debug to show the value of a variable in the memory and show the change of the value during execution. Figure 1 shows a simple test program with variable *i*.


NetBeans IDE 3.6 - Project Liang Intro Book
File Edit View Project Build Debug Versioning Tools Window Help
Filesyst... Runtime
Filesystems
C:\smith
ShowCurrentTime
Test
Welcome
WelcomeApplet
WelcomeApplet
Test
main
6:1 INS
Output
Debugger Console Process Output Compiler
Local Variables Watches Call Stack

```
public class Test {
  public static void main(String[] args) {
    int i;
    i = 4;
    i =5;
  }
```

```
Finished.
```

Name	Value
args	#52 (Length=0)
i	5

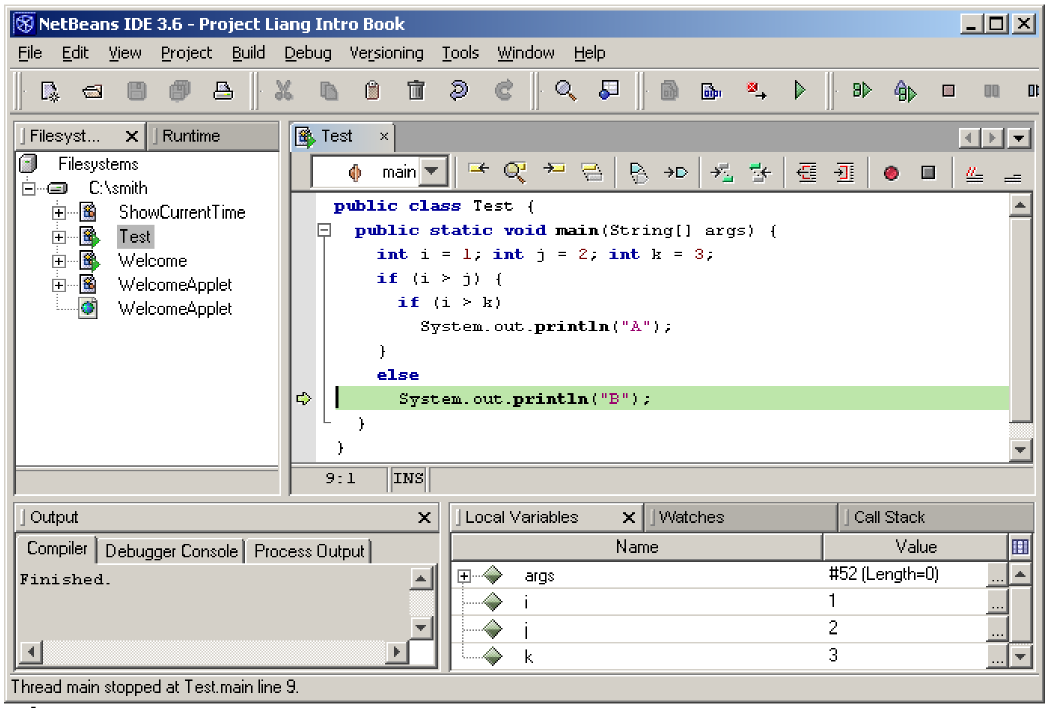
Thread main stopped at Test.main line 6.

Figure 1

Displaying values of variables in NetBeans debugger.

2.3 Using NetBeans in Chapter 3

Use the debugger to trace the if statements in Section 3.2.3, “Nested <u>if</u> Statements,” as shown in Figure 2.


NetBeans IDE 3.6 - Project Liang Intro Book
File Edit View Project Build Debug Versioning Tools Window Help
Filesyst... Runtime
Filesystems
C:\smith
ShowCurrentTime
Test
Welcome
WelcomeApplet
WelcomeApplet
Test
main
9:1 INS
Output
Compiler Debugger Console Process Output
Local Variables Watches Call Stack

```
public class Test {
  public static void main(String[] args) {
    int i = 1; int j = 2; int k = 3;
    if (i > j) {
      if (i > k)
        System.out.println("A");
    }
    else
      System.out.println("B");
  }
}
```

```
Finished.
```

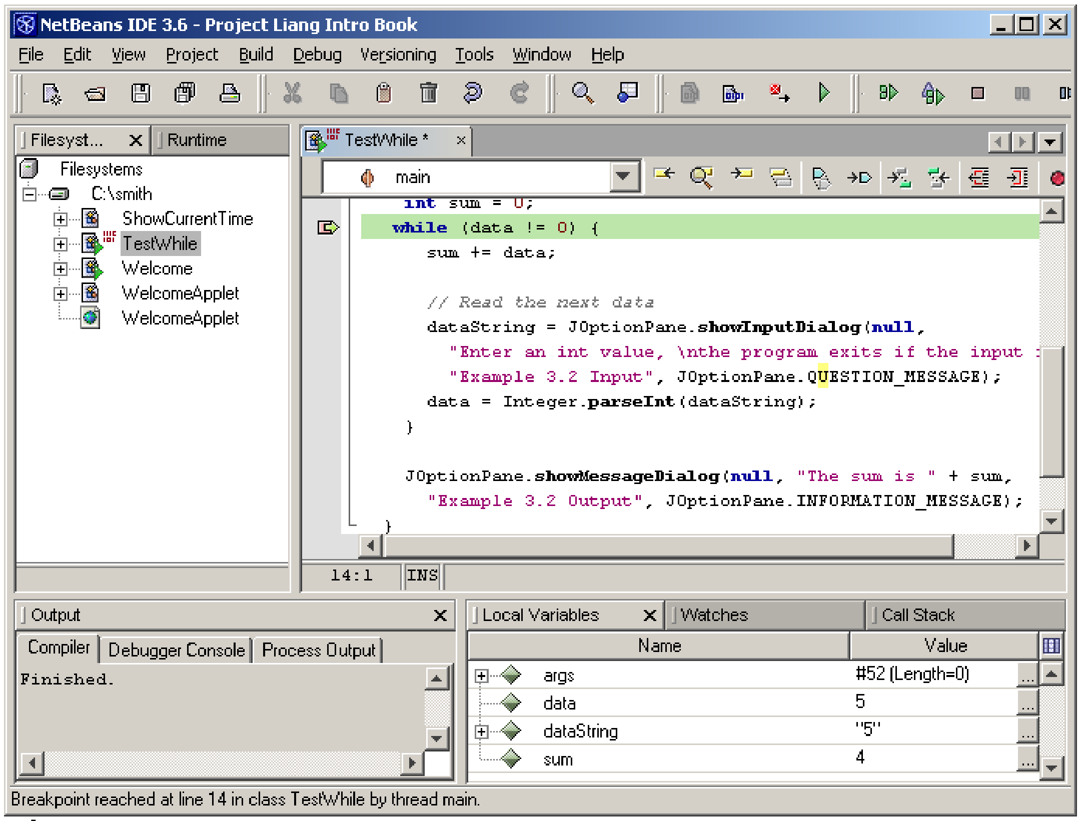
Name	Value
args	#52 (Length=0)
i	1
j	2
k	3

Thread main stopped at Test.main line 9.

Figure 2

Trace the execution of an if statement.

Use the debugger to trace the while loop in Listing 3.2 TestWhile.java (Using while Loop)," as shown in Figure 3.


NetBeans IDE 3.6 - Project Liang Intro Book
File Edit View Project Build Debug Versioning Tools Window Help
Filesyst... Runtime
Filesystems
C:\smith
ShowCurrentTime
TestWhile
Welcome
WelcomeApplet
WelcomeApplet
TestWhile *
main
14:1 INS
Output
Compiler Debugger Console Process Output
Local Variables Watches Call Stack

```
    int sum = 0;
    while (data != 0) {
      sum += data;

      // Read the next data
      dataString = JOptionPane.showInputDialog(null,
        "Enter an int value, \nthe program exits if the input :
        "Example 3.2 Input", JOptionPane.QUESTION_MESSAGE);
      data = Integer.parseInt(dataString);
    }

    JOptionPane.showMessageDialog(null, "The sum is " + sum,
      "Example 3.2 Output", JOptionPane.INFORMATION_MESSAGE);
  }
```

```
Finished.
```

Name	Value
args	#52 (Length=0)
data	5
dataString	"5"
sum	4

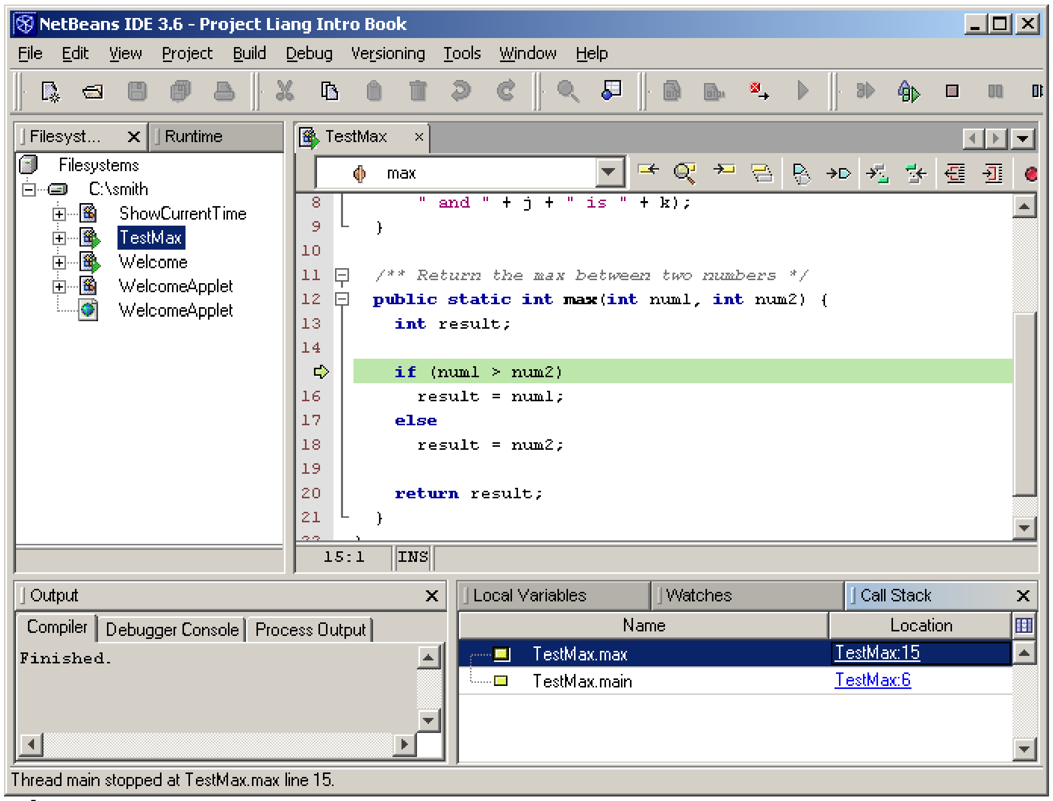
Breakpoint reached at line 14 in class TestWhile by thread main.

Figure 3

Trace the execution of a loop statement.

2.4 Using NetBeans in Chapter 5

You can use the debugger to show the call stack, which is very effective to help understand method invocation. Let us use Listing 5.1 to demonstrate method invocation. Set a breakpoint at Line 6. Start debugger, and the debugger pauses at Line 6. Choose Step into to step into the max method, as shown in Figure 4. Now in the Call Stack tab of the Debugger window, you will see method max to be invoked.


NetBeans IDE 3.6 - Project Liang Intro Book
File Edit View Project Build Debug Versioning Tools Window Help
Filesyst... Runtime
Filesystems
C:\smith
ShowCurrentTime
TestMax
Welcome
WelcomeApplet
WelcomeApplet
TestMax
max
15:1 INS
Output
Compiler Debugger Console Process Output
Local Variables Watches Call Stack

```
      " and " + j + " is " + k);
  }

  /** Return the max between two numbers */
  public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
      result = num1;
    else
      result = num2;

    return result;
  }
```

```
Finished.
```

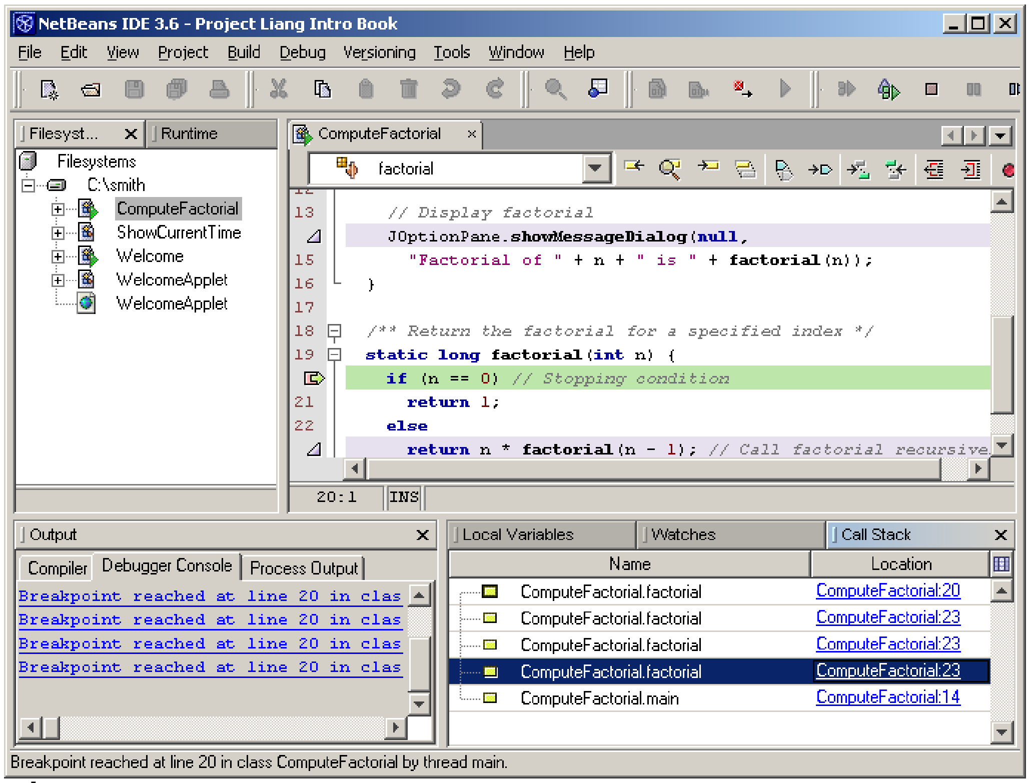
Name	Location
TestMax.max	TestMax:15
TestMax.main	TestMax:6

Thread main stopped at TestMax.max line 15.

Figure 4

Trace method invocation.

Figure 5 shows tracing recursive execution of the factorial method.


NetBeans IDE 3.6 - Project Liang Intro Book
File Edit View Project Build Debug Versioning Tools Window Help
Filesyst... Runtime
Filesystems
C:\smith
ComputeFactorial
ShowCurrentTime
Welcome
WelcomeApplet
WelcomeApplet
ComputeFactorial
factorial
20:1 INS
Output
Compiler Debugger Console Process Output
Local Variables Watches Call Stack

```
    // Display factorial
    JOptionPane.showMessageDialog(null,
      "Factorial of " + n + " is " + factorial(n));
  }

  /** Return the factorial for a specified index */
  static long factorial(int n) {
    if (n == 0) // Stopping condition
      return 1;
    else
      return n * factorial(n - 1); // Call factorial recursive
```

```
Breakpoint reached at line 20 in clas
Breakpoint reached at line 20 in clas
Breakpoint reached at line 20 in clas
Breakpoint reached at line 20 in clas
```

Name	Location
ComputeFactorial.factorial	ComputeFactorial:20
ComputeFactorial.factorial	ComputeFactorial:23
ComputeFactorial.factorial	ComputeFactorial:23
ComputeFactorial.factorial	ComputeFactorial:23
ComputeFactorial.main	ComputeFactorial:14

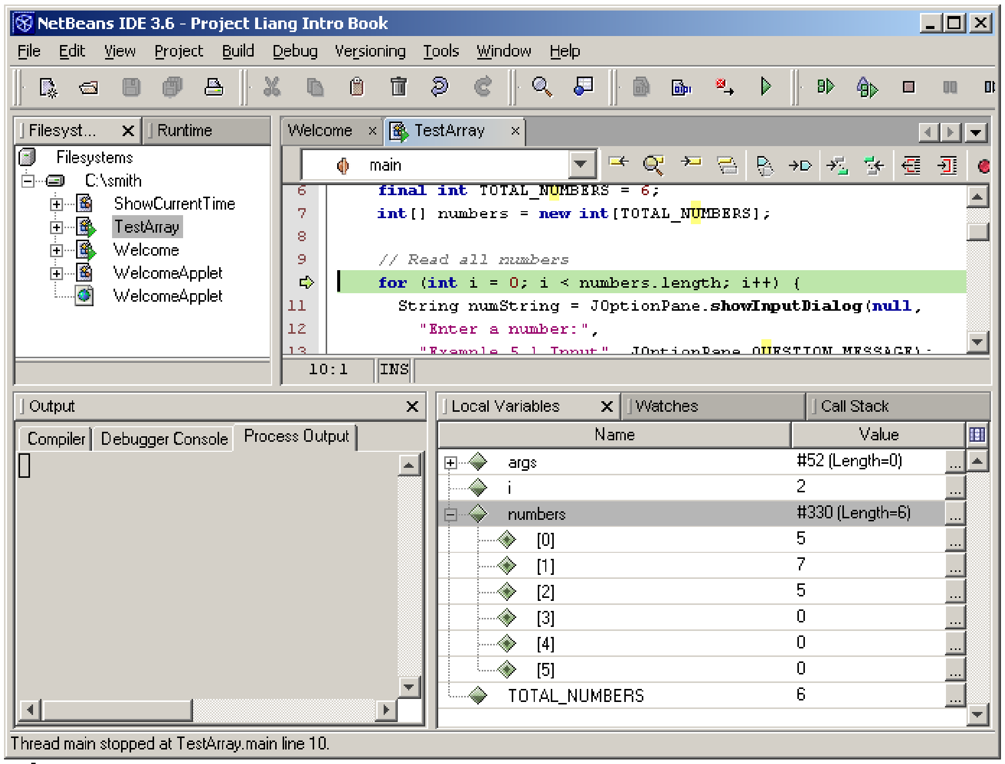
Breakpoint reached at line 20 in class ComputeFactorial by thread main.

Figure 5

Trace a recursive method invocation.

2.5 Using NetBeans in Chapter 6

You can use the debugger to show the values of all the elements in an array. Figure 6 shows debugging Listing 6.1.


NetBeans IDE 3.6 - Project Liang Intro Book
File Edit View Project Build Debug Versioning Tools Window Help
Filesyst... Runtime
Filesystems
C:\smith
ShowCurrentTime
TestArray
Welcome
WelcomeApplet
WelcomeApplet
Welcome TestArray
main
Output
Compiler Debugger Console Process Output
Local Variables Watches Call Stack

```
    final int TOTAL_NUMBERS = 6;
    int[] numbers = new int[TOTAL_NUMBERS];

    // Read all numbers
    for (int i = 0; i < numbers.length; i++) {
      String numString = JOptionPane.showInputDialog(null,
        "Enter a number:",
        "Example 5.1 Input", JOptionPane.QUESTION_MESSAGE);
```

10:1 INS

Name	Value
args	#52 (Length=0)
i	2
numbers	#330 (Length=6)
[0]	5
[1]	7
[2]	5
[3]	0
[4]	0
[5]	0
TOTAL_NUMBERS	6

Thread main stopped at TestArray.main line 10.

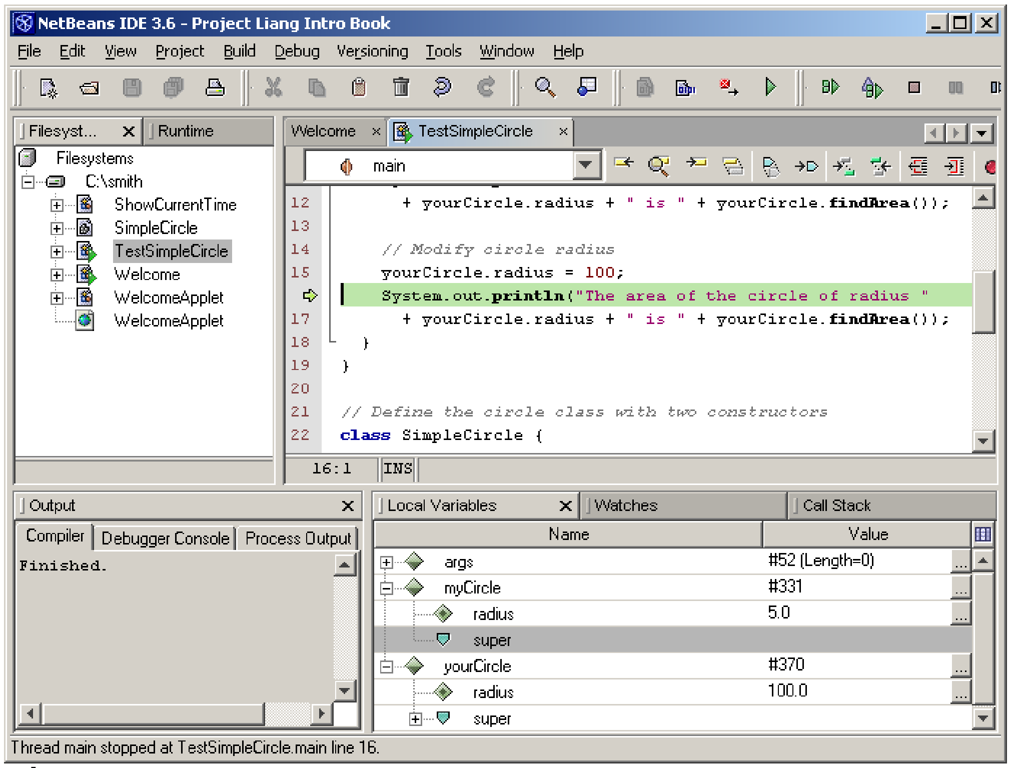
Figure 6

You can see the change of values in an array.

You can use the debugger to demonstrate how arguments are passed and to see the differences between passing primitive type values and arrays.

2.5 Using NetBeans in Chapter 7

You can use the debugger to show the contents of an object. Figure 7 shows debugging Listing 7.1.


NetBeans IDE 3.6 - Project Liang Intro Book
File Edit View Project Build Debug Versioning Tools Window Help
Filesyst... Runtime
Filesystems
C:\smith
ShowCurrentTime
SimpleCircle
TestSimpleCircle
Welcome
WelcomeApplet
WelcomeApplet
Welcome TestSimpleCircle
main
16:1 INS
Output
Compiler Debugger Console Process Output
Local Variables Watches Call Stack

```
        + yourCircle.radius + " is " + yourCircle.findArea());

      // Modify circle radius
      yourCircle.radius = 100;
      System.out.println("The area of the circle of radius "
        + yourCircle.radius + " is " + yourCircle.findArea());
    }
  }

  // Define the circle class with two constructors
  class SimpleCircle {
```

```
Finished.
```

Name	Value
args	#52 (Length=0)
myCircle	#331
radius	5.0
super	
yourCircle	#370
radius	100.0
super	

```
Thread main stopped at TestSimpleCircle.main line 16.
```

Figure 7

You can see the change of values in an object.

You can use the debugger to demonstrate how arguments are passed and to see the differences between passing primitive type values and objects.