Chapter 1 Introduction to Computers and Java Applets (Main Page)

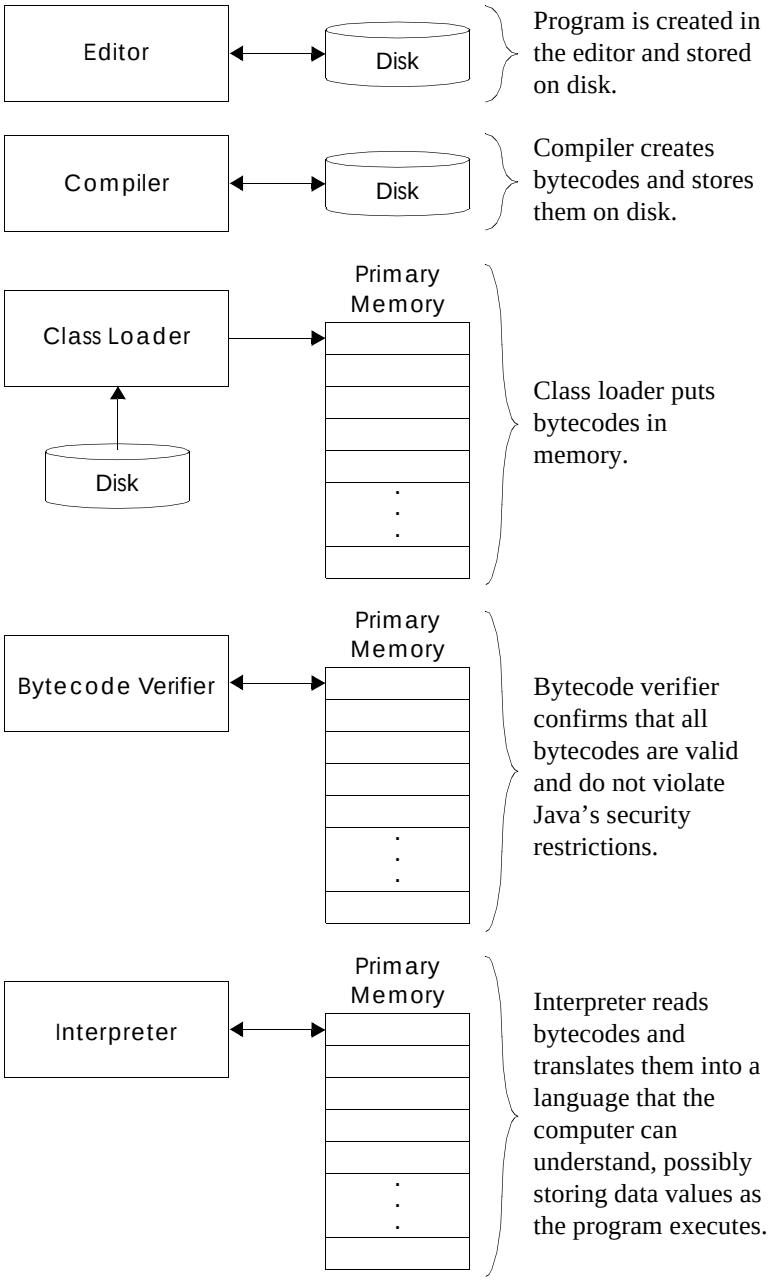


Fig. 1.1 A typical Java environment.

```
1  // A first program in Java
2  import java.applet.Applet;  // import Applet class
3  import java.awt.Graphics;   // import Graphics class
4
5  public class Welcome extends Applet {
6     public void paint( Graphics g )
7     {
8        g.drawString( "Welcome to Java Programming!", 25, 25 );
9     }
10 }
```

Fig. 1.2 A first program in Java and the program's screen output.

```
11 <html>
12 <applet code="Welcome.class" width=275 height=55>
13 </applet>
14 </html>
```

Fig. 1.3 The **`Welcome.html`** file that loads the **`Welcome`** applet class of Fig. 1.2 into a browser.

```
1  // Displaying multiple strings
2  import java.applet.Applet;  // import Applet class
3  import java.awt.Graphics;   // import Graphics class
4
5  public class Welcome extends Applet {
6     public void paint( Graphics g )
7     {
8        g.drawString( "Welcome to", 25, 25 );
9        g.drawString( "Java Programming", 25, 40 );
10    }
11 }
```

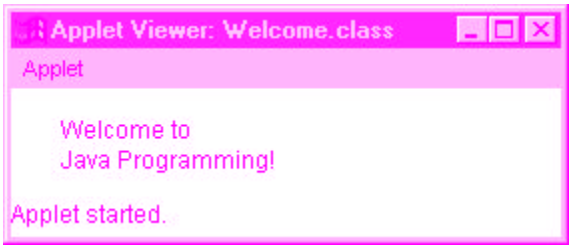


Fig. 1.4 Displaying multiple strings.

Fig. 1.5 A sample Netscape Navigator window with GUI components.

```
1  // Addition program
2  import java.applet.Applet;
3  import java.awt.*;    // import the java.awt package
4  import java.awt.event.*;  // import the java.awt.event package
5
6  public class Addition extends Applet implements ActionListener {
7     Label prompt;      // message that prompts user to input
8     TextField input;   // input values are entered here
9     int number;        // variable that stores input value
10    int sum;           // variable that stores sum of integers
11
12    // setup the graphical user interface components
13    // and initialize variables
14    public void init()
15    {
16       prompt = new Label( "Enter integer and press Enter:" );
17       add( prompt );  // put prompt on applet
18
19       input = new TextField( 10 );
20       add( input );   // put input TextField on applet
21
22       sum = 0;        // set sum to 0
23
24       // "this" applet handles action events for TextField input
25       input.addActionListener( this );
26    }
27
28    // process user's action in TextField input
29    public void actionPerformed( ActionEvent e )
30    {
31       // get the number and convert it to an integer
```

```
32          number = Integer.parseInt( e.getActionCommand() );
33
34          sum = sum + number;   // add number to sum
35          input.setText( "" );  // clear data entry field
36          showStatus( Integer.toString( sum ) );  // display sum
37       }
38    }
```

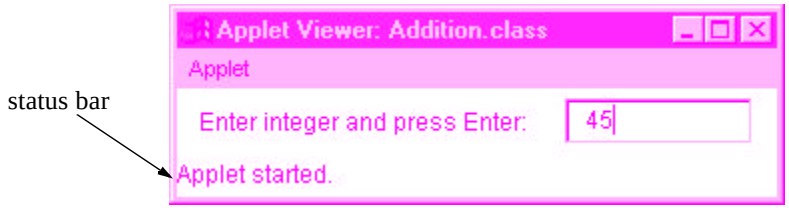


Fig. 1.6 An addition program "in action" (part 1 of 2).

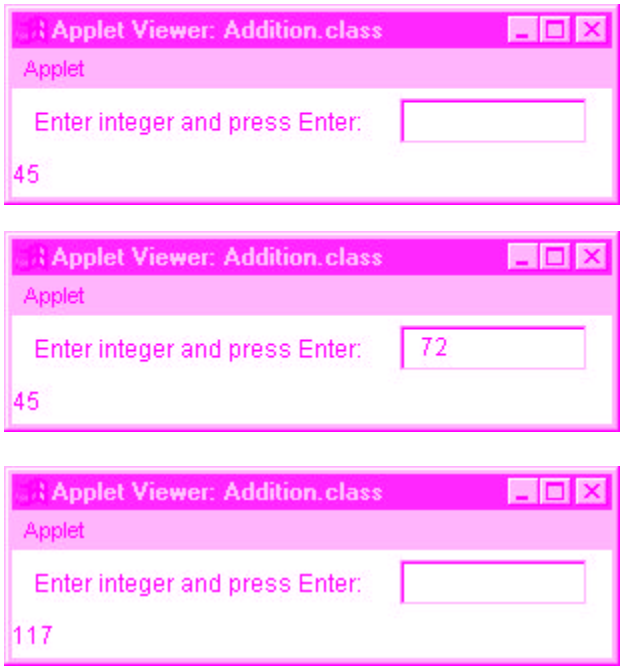


Fig. 1.6 An addition program "in action" (part 2 of 2).

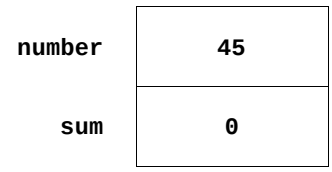


Fig. 1.7 Memory locations showing the name and value of a variable.

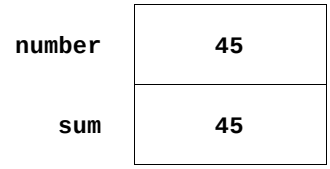


Fig. 1.8 Memory locations after a calculation.

Java operation	Arithmetic operator	Algebraic expression	Java expression
Addition	`+`	f + 7	`f + 7`
Subtraction	`-`	p – c	`p - c`
Multiplication	`*`	bm	`b * m`
Division	`/`	x / y *or* $\frac{x}{y}$ *or* x ÷ y	`x / y`
Modulus	`%`	r mod s	`r % s`

Fig. 1.9 Arithmetic operators.

Operator(s)	Operation(s)	Order of evaluation (precedence)
`( )`	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses “on the same level” (i.e., not nested), they are evaluated left to right.
`*, /, or %`	Multiplication Division Modulus	Evaluated second. If there are several, they are evaluated left to right.
`+ or -`	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Fig. 1.10 Precedence of arithmetic operators.

Step 1. `y = 2 * 5 * 5 + 3 * 5 + 7;`

`2 * 5 is 10` (Leftmost multiplication)

Step 2. `y = 10 * 5 + 3 * 5 + 7;`

`10 * 5 is 50` (Leftmost multiplication)

Step 3. `y = 50 + 3 * 5 + 7;`

`3 * 5 is 15` (Multiplication before addition)

Step 4. `y = 50 + 15 + 7;`

`50 + 15 is 65` (Leftmost addition)

Step 5. `y = 65 + 7;`

`65 + 7 is 72` (Last addition)

Step 6. `y = 72;` (Last operation—assignment)

Standard algebraic equality operator or relational operator	Java equality or relational operator	Example of Java condition	Meaning of Java condition
Equality operators			
=	`==`	`x == y`	**x** is equal to **y**
≠	`!=`	`x != y`	**x** is not equal to **y**
Relational operators			
>	`>`	`x > y`	**x** is greater than **y**
<	`<`	`x < y`	**x** is less than **y**
≥	`>=`	`x >= y`	**x** is greater than or equal to **y**
≤	`<=`	`x <= y`	**x** is less than or equal to **y**

Fig. 1.12 Equality and relational operators.

```
1  // Using if statements, relational
2  // operators, and equality operators
3  import java.applet.Applet;
4  import java.awt.*;
5  import java.awt.event.*;
6
7  public class Comparison extends Applet
8                implements ActionListener {
9     Label prompt1;     // prompt user to input first value
10    TextField input1;  // input first value here
11    Label prompt2;     // prompt user to input second value
12    TextField input2;  // input second value here
13    int number1, number2;  // store input values
14
15    // setup the graphical user interface components
16    // and initialize variables
17    public void init()
18    {
19       prompt1 = new Label( "Enter an integer" );
20       add( prompt1 );  // put prompt1 on applet
21
22       input1 = new TextField( 10 );
23       add( input1 );   // put input1 on applet
24
25       prompt2 =
26          new Label( "Enter an integer and press Enter" );
27       add( prompt2 );  // put prompt2 on applet
28
29       input2 = new TextField( 10 );
30       input2.addActionListener( this );
31       add( input2 );   // put input2 on applet
32    }
33
34    // display the results
35    public void paint( Graphics g )
36    {
37       g.drawString( "The comparison results are:", 70, 75 );
38
39       if ( number1 == number2 )
40          g.drawString( number1 + " == " + number2, 100, 90 );
41
42       if ( number1 != number2 )
43          g.drawString( number1 + " != " + number2, 100, 105 );
44
45       if ( number1 < number2 )
46          g.drawString( number1 + " < " + number2, 100, 120 );
47
48       if ( number1 > number2 )
49          g.drawString( number1 + " > " + number2, 100, 135 );
50
```

Fig. 1.13 Using equality and relational operators (part 1 of 4).

```
51       if ( number1 <= number2 )
52          g.drawString( number1 + " <= " + number2, 100, 150 );
53
54       if ( number1 >= number2 )
55          g.drawString( number1 + " >= " + number2, 100, 165 );
56    }
57
58    // process user's action on the input2 text field
59    public void actionPerformed( ActionEvent e )
60    {
61       number1 = Integer.parseInt( input1.getText() );
```

```
62          number2 = Integer.parseInt( input2.getText() );
63          repaint();
64       }
65    }
```

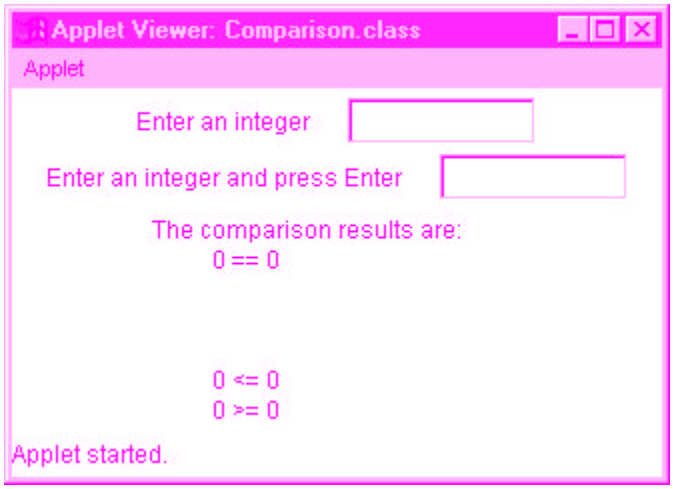
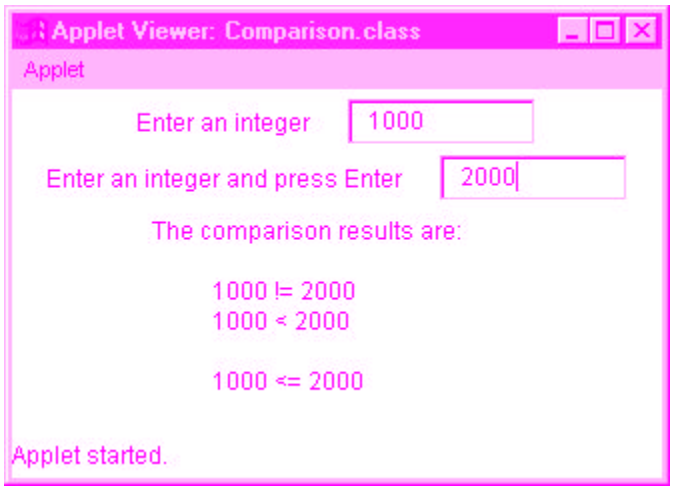


Fig. 1.13 Using equality and relational operators (part 2 of 4).



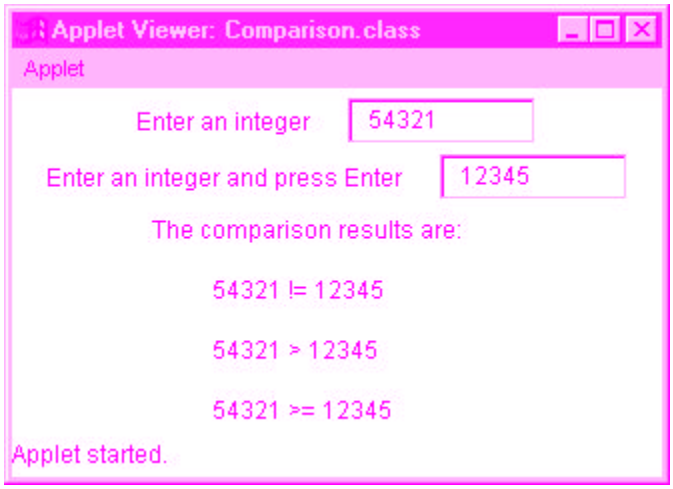


Fig. 1.13 Using equality and relational operators (part 3 of 4).

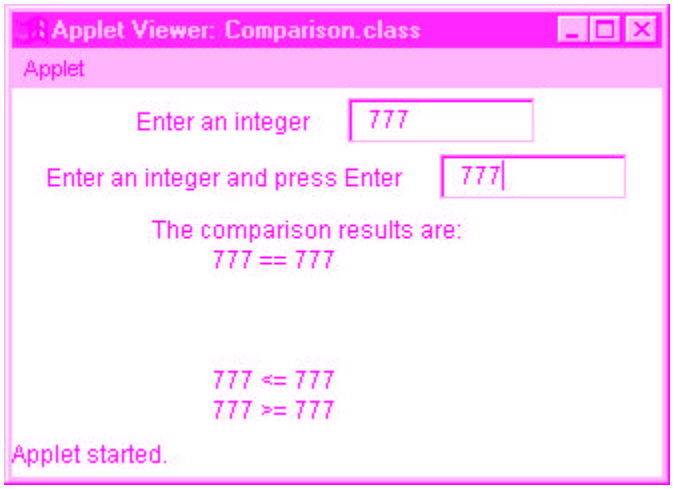


Fig. 1.13 Using equality and relational operators (part 4 of 4).

Operators	Associativity	Type
( )	left to right	parentheses
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relationals
== !=	left to right	equalities
=	right to left	assignment

Fig. 1.14 Precedence and associativity of the operators discussed so far.