Statements and Control Flow

- The programs we have seen so far do exactly the same list of instructions every time
- What if we want to do different things for different inputs?
  - Do some action only if a specified condition is met
  - We need *conditional* statements

```
if(value < 0)
   System.out.println("Bad input");
```

Statements and Control Flow

- What if we want to repeat some set of instructions some number of times?
  - Repeat an action some number of times
  - We need *iterative* statements

```
while(i < 100)
    System.out.println(i++);
```

Statements

- *Declaration Statement*: type, followed by a comma-separated list of identifiers, followed by a semi-colon

  int foo, bar;

  String name = “Scott”;

- *Expression Statement*: expression followed by a semi-colon
  - *Assignment Expression*: size = size + 5
  - *Method Call Expression*: System.out.println(…)
  - Not all expressions can be part of an expression statement (more on this later)

Block Statement

- *Block Statement*: one or more statements inside braces, e.g.,

```
{
    int a = 4;                        // Statement
    System.out.println(a);            // Statement
}                                     // Statement
```

  - A Block Statement is a Statement
  - Block Statements can contain Block Statements
  - Variables declared within a block disappear when the block has finished executing

Empty Statement

- *Empty statement*: do-nothing statement
  ;
  - Is a statement, but does nothing
- Example:
  - Wait for a condition to become true

```
while(notTimeYet)
    ;
 <other stuff>
```

  - Question: Which is clearer, that, or this:

```
while(notTimeYet);
   <other stuff>
```

Boolean Expression

- Any expression that evaluates to true or false
  - true
  - false
  - Comparisons
  - Logical operations

Relational Operators

Name	**Symbol**	**Expression**
Equal	= =	a = = b
Not equal	!=	a != b
Less than	<	a < b
Greater than	>	a > b
Less than or equal to	<=	a <= b
Greater Than or Equal to	>=	a >= b

Comparisons

- Comparisons (using relational operators) evaluate to true or false
- Example:

```
int a = 5, b = 7;
boolean flag;
flag = (a < b);           // boolean expression
System.out.println(flag);
```

Logical Operators

- Operations on logical values

Name	**Operator**	**Expression**
NOT	!	a = !(b = = c)
AND	&&	a = (b && c)
OR	||	a = (b || c)

Logical Operations

- Example 1

```
int x, y;
boolean b;
x = in.nextInt();
y = in.nextInt();
b = (x == y);
System.out.println(b);
```

- Example 2

```
boolean b = (age >= 18 && age < 65);
System.out.println("full fare adult is " + b);
b = (age < 18 || age >= 65);
System.out.println("reduced fare is" + b);
```

Operator Precedence and Associativity

- *Operator Precedence*
  - The order in which different operators are evaluated, i.e. who goes first
  - * has higher precedence than +, both higher than =

    int x = 3 + 4 * 5; // x = 23, not 35!
- *Operator Associativity*
  - The order in which operators of the same precedence are applied
  - * and % have equal precedence, left to right associativity

    int y = 4 * 3 % 2; // y = 0, not 4!

Operator Precedence and Associativity

Operators	**Associativity**
( ) ++ (postfix) -- (postfix)	Left to right
+ (unary) - (unary) ++ (prefix) -- (prefix) !	Right to left
* / %	Left to right
+ -	Left to right
< <= > >=	Left to right
= = !=	Left to right
&&	Left to right
||	Left to right
= += -= *= /= etc.	Right to left

What if we want a different evaluation order?

- Parentheses ( ) have a higher precedence than just about everything else
  - They can be used to impose a different evaluation order

```
int x = 3 + 4 * 5;      // x = 23
int x = (3 + 4) * 5;    // x = 35
int y = 4 * 3 % 2;      // y = 0
int y = 4 * (3 % 2);    // y = 4
```

Conditional Statements

- Conditionally execute a statement based on the value of a boolean expression
  - **if statement** - decide whether or not to take a particular action
    - Execute a particular statement only if a given boolean expression is true
  - **if-else statement -** choose between two alternative actions
    - Execute one of two statements based on the value of a given boolean expression

Conditional Statements (cont.)

- **switch statement:** choose among several alternative actions
  - Execute one of a set of statements based on a specified value (not a boolean expression)
- **while statement:** repeat an action as long as a specified condition is true
  - Repeatedly execute a statement as long as the given boolean expression is true
- **for statement:** execute an action a specified number of times
  - Repeatedly execute a statement as long as the given boolean expression is true

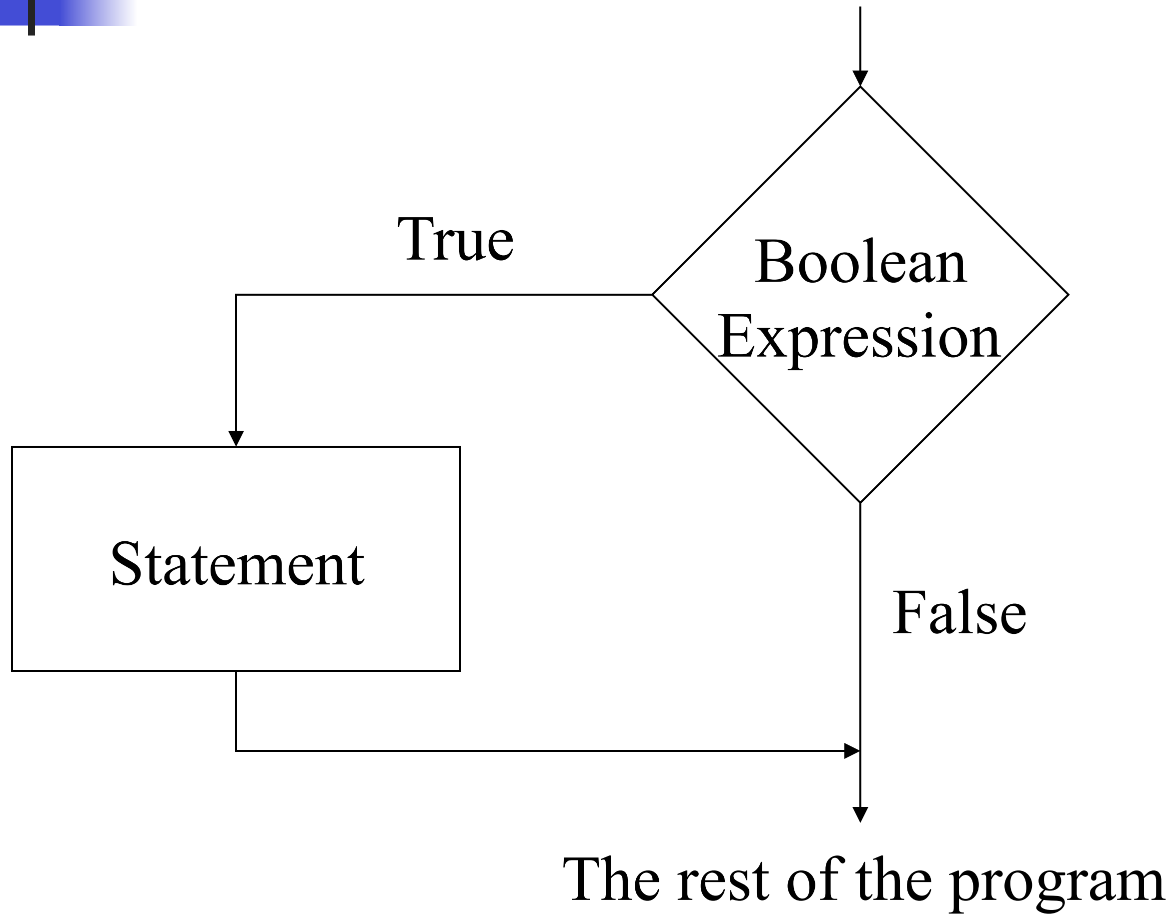
if Statement

- Used to decide whether or not to take a particular action

```
if(<boolean expression>)
    <statement>
```

- If the boolean expression is true, the *then statement* is executed, otherwise it is not

Flowchart For An if Statement



if Statements in Action

```
if(value > 50)
    System.out.println("Warning, value too big!");
```

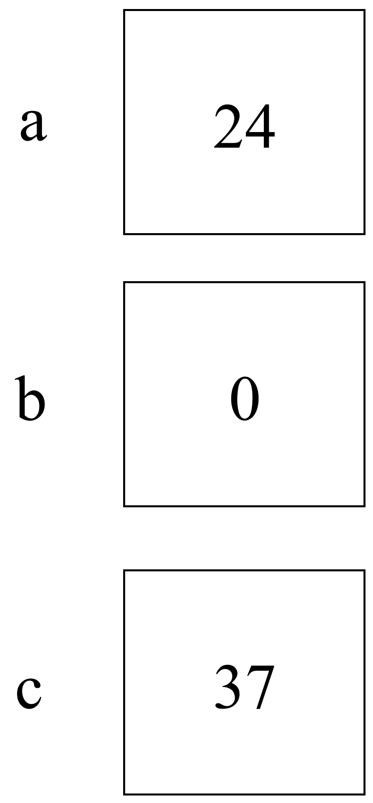
```
if(y != 0)
    z = x / y;
```

```
if(item.price < 100 && cashOnHand >= item.price) {
    item.purchase( );
    cashOnHand - = item.price;
}
```

Example: Bubblesort

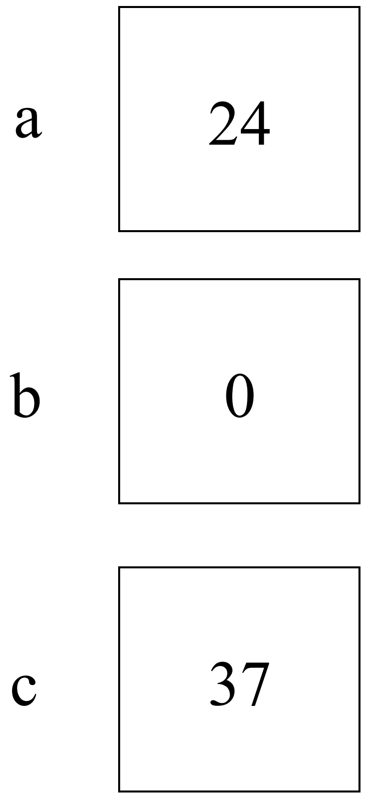
- Given three numbers, place them in increasing order
- Algorithm:
  1. Put the three numbers in *a*, *b*, and *c*
  2. if *b* is less than *a*, swap *a* and *b*
  3. if *c* is less than *b*
     1. swap *b* and *c*
     2. if *b* is less than *a*, swap *a* and *b*

Bubblesort (1)



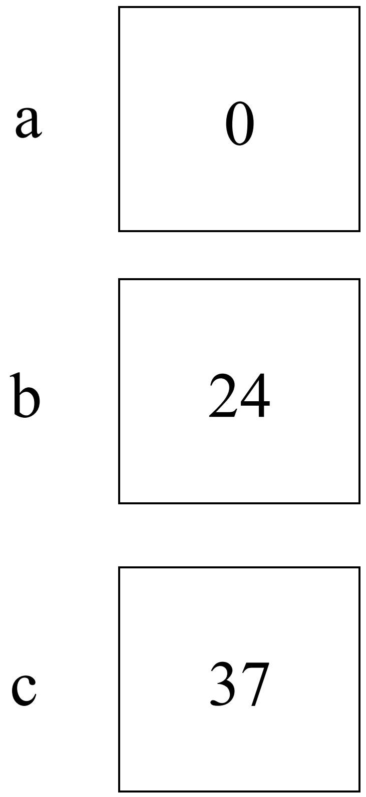
1. **Put the three numbers in *a*, *b*, and *c***
2. If *b* is less than *a*, swap *a* and *b*
3. If *c* is less than *b*
   1. swap *b* and *c*
   2. if *b* is less than *a*, swap *a* and *b*

Bubblesort (1)



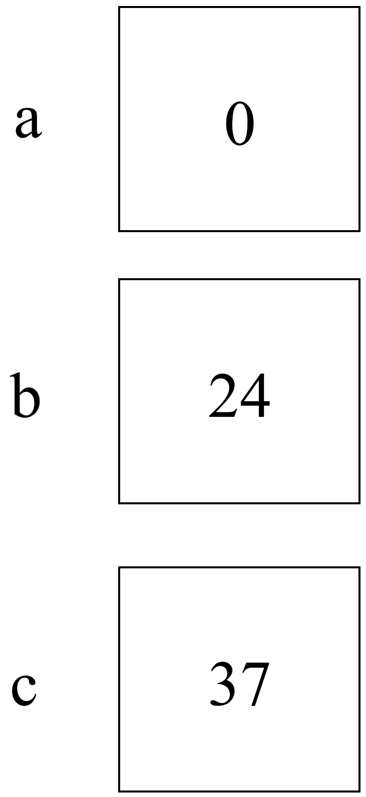
1. Put the three numbers in *a*, *b*, and *c*
2. **If *b* is less than *a***, swap *a* and *b*
3. If *c* is less than *b*
    1. swap *b* and *c*
    2. if *b* is less than *a*, swap *a* and *b*

Bubblesort (1)



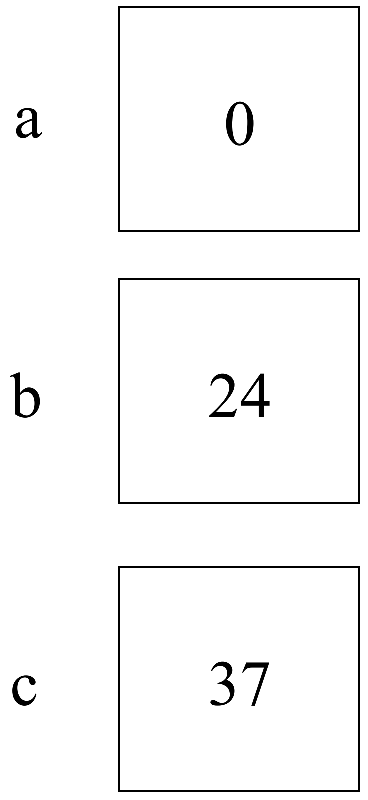
1. Put the three numbers in *a*, *b*, and *c*
2. If *b* is less than *a*, **swap *a* and *b***
3. If *c* is less than *b*
    1. swap *b* and *c*
    2. if *b* is less than *a*, swap *a* and *b*

Bubblesort (1)

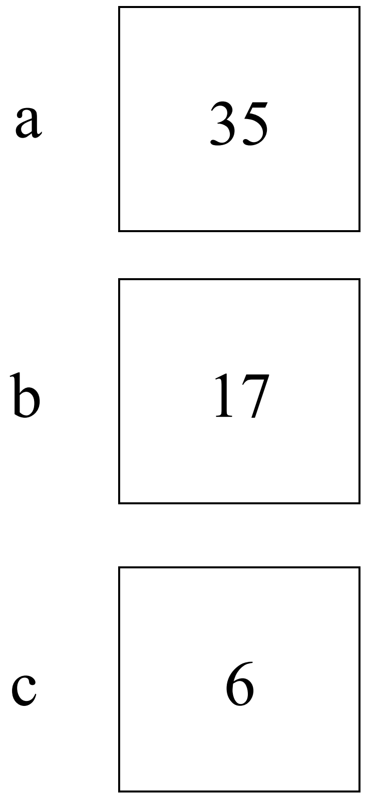


1. Put the three numbers in *a*, *b*, and *c*
2. If *b* is less than *a*, swap *a* and *b*
3. **If *c* is less than *b***
    1. swap *b* and *c*
    2. if *b* is less than *a*, swap *a* and *b*

Bubblesort (1)

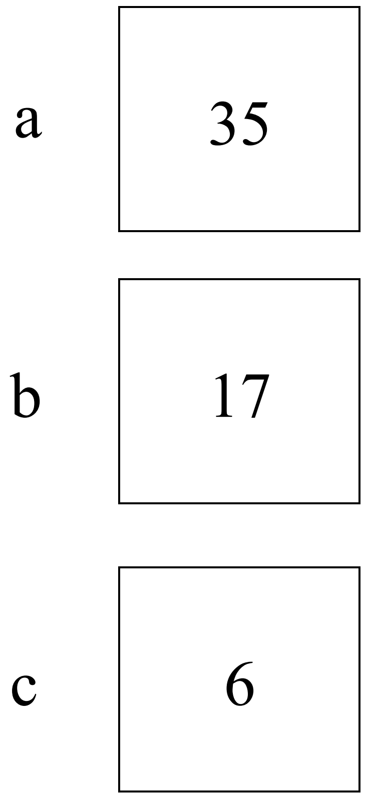


Bubblesort (2)



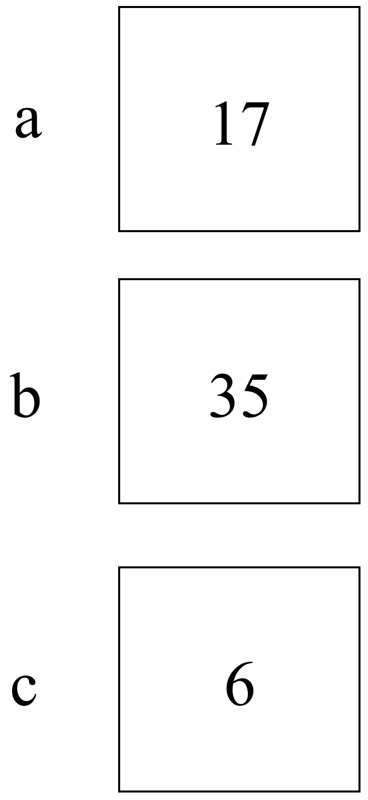
1. **Put the three numbers in *a*, *b*, and *c***
2. If *b* is less than *a*, swap *a* and *b*
3. If *c* is less than *b*
   1. swap *b* and *c*
   2. if *b* is less than *a*, swap *a* and *b*

Bubblesort (2)



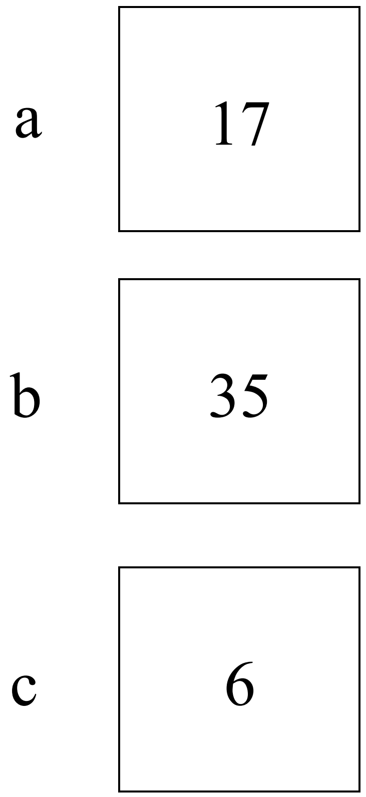
1. Put the three numbers in *a*, *b*, and *c*
2. **If *b* is less than *a***, swap *a* and *b*
3. If *c* is less than *b*
   1. swap *b* and *c*
   2. if *b* is less than *a*, swap *a* and *b*

Bubblesort (2)



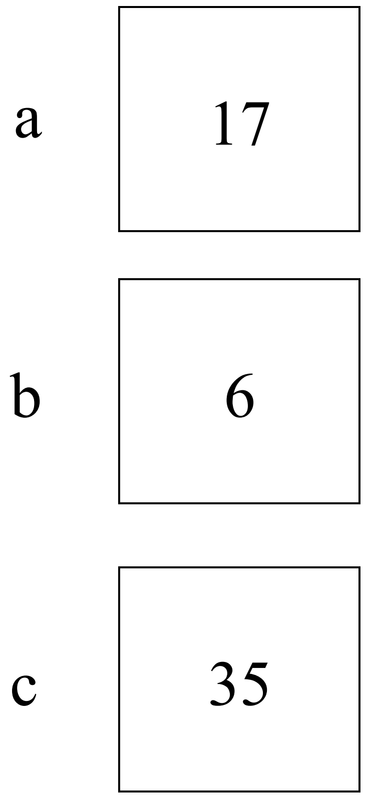
1. Put the three numbers in *a*, *b*, and *c*
2. If *b* is less than *a*, **swap *a* and *b***
3. If *c* is less than *b*
    1. swap *b* and *c*
    2. if *b* is less than *a*, swap *a* and *b*

Bubblesort (2)



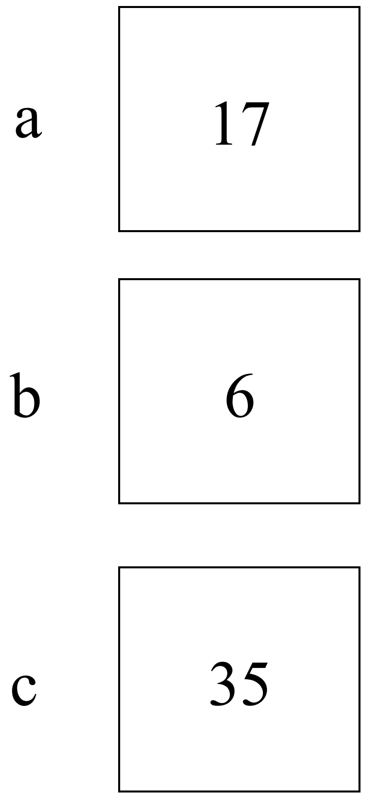
1. Put the three numbers in *a*, *b*, and *c*
2. If *b* is less than *a*, swap *a* and *b*
3. **If *c* is less than *b***
    1. swap *b* and *c*
    2. if *b* is less than *a*, swap *a* and *b*

Bubblesort (2)



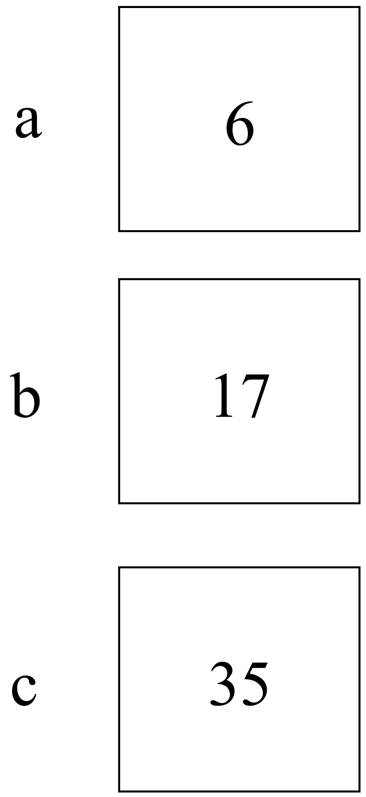
1. Put the three numbers in *a*, *b*, and *c*
2. If *b* is less than *a*, swap *a* and *b*
3. If *c* is less than *b*
    1. **swap *b* and *c***
    2. if *b* is less than *a*, swap *a* and *b*

Bubblesort (2)



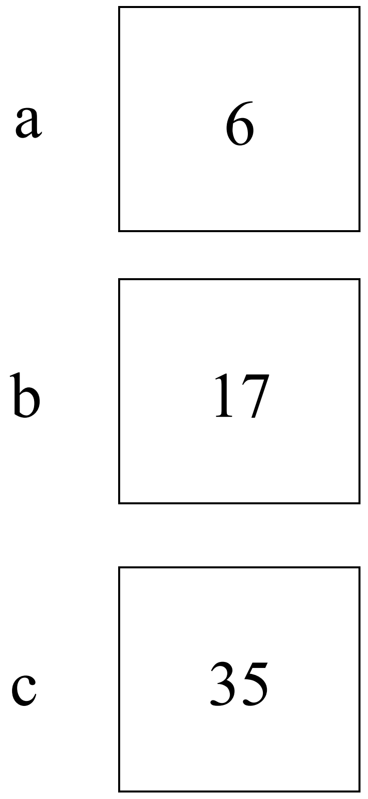
1. Put the three numbers in *a*, *b*, and *c*
2. If *b* is less than *a*, swap *a* and *b*
3. If *c* is less than *b*
   1. swap *b* and *c*
   2. **if *b* is less than *a***, swap *a* and *b*

Bubblesort (2)



1. Put the three numbers in *a*, *b*, and *c*
2. If *b* is less than *a*, swap *a* and *b*
3. If *c* is less than *b*
    1. swap *b* and *c*
    2. if *b* is less than *a*, **swap *a* and *b***

Bubblesort (2)



```
// SortInput.java - sort three numbers
import java.util.*; // for Scanner

class SortInput {
    public static void main (String[] args) {
        int a, b, c, temp;
        Scanner in = new Scanner (System.in);

        // Get three numbers from the user
        System.out.println("type three integers:");
        a = in.nextInt();
        b = in.nextInt();
        c = in.nextInt();

        // If b is less than a, swap a and b
        if (b < a) {
            temp = a;
            a = b;
            b = temp;
        }
        // If c is less than b, swap b and c
        if (c < b) {
            // swap b and c
            temp = b;
            b = c;
            c = temp;

            // if (the new) b is less than a, swap a and b
            if (a > b) {
                temp = a;
                a = b;
                b = temp;
            }
        }

        System.out.print("The sorted order is : ");
        System.out.println(a + ", " + b + ", " + c);
    }
}
```

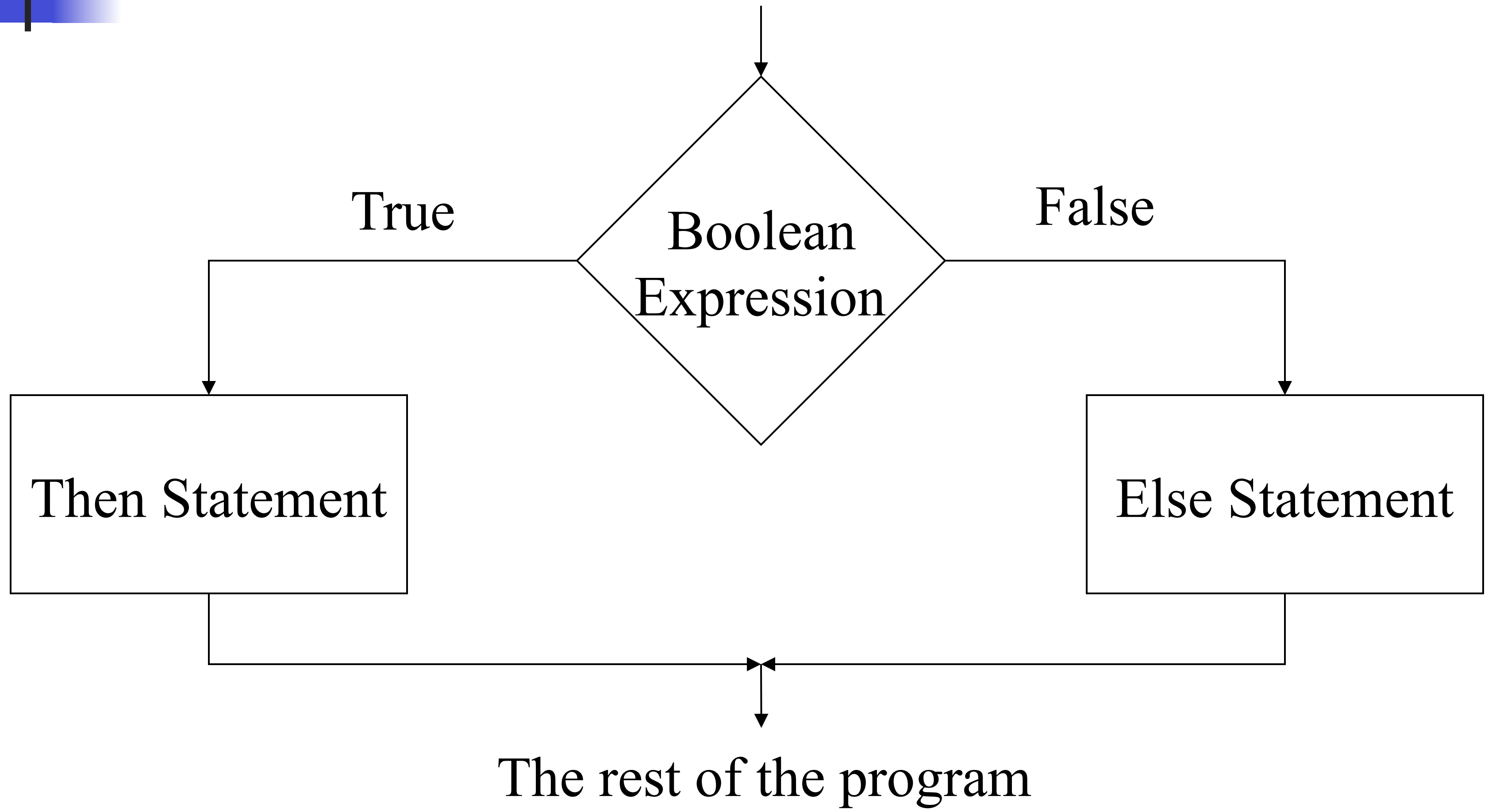
If-else Statement

- Used to choose between two alternative actions

```
if(<boolean expression>)
    <statement>
else
    <statement>
```

- If the boolean expression is true, the *then statement* is executed, otherwise the *else statement* is executed

Flowchart For An if-else Statement



If-else Statements in Action

```
if (x < y)
    min = x;
else
    min = y;

System.out.println("min = " + min);
```

```
if(y == 0)
   System.out.println("Divide by zero error!");
else
    z = x / y;
```

Details

- Any statement can be a then or an else statement
  - Expression Statement, Block Statement, Conditional Statement (including if or if-else Statements), etc.
- Common errors
  - Look at the ones listed in the book
    - They are exactly right

if-else-if-else

- If you string if-elses together, each if-else is the statement for the previous else

```
if(<boolean expression>)
    <statement>
else if(<boolean expression>)
    <statement>
else <statement>
<etc.>
```

Dangling else

- An *else* always binds to the nearest previous unmatched *if* in its block

```
if(<boolean expression1>)  {  // if 1
    if(<boolean expression2>)    // if 2
       <statement>
   else                                        // binds to "if 2"
       <statement>
}
```

Dangling else

- An *else* always binds to the nearest unmatched *if* in its block

```
if(<boolean expression1>)  {  // if 1
if(<boolean expression2>)     // if 2
    <statement>
}
else                                    // binds to “if 1”
    <statement>
```

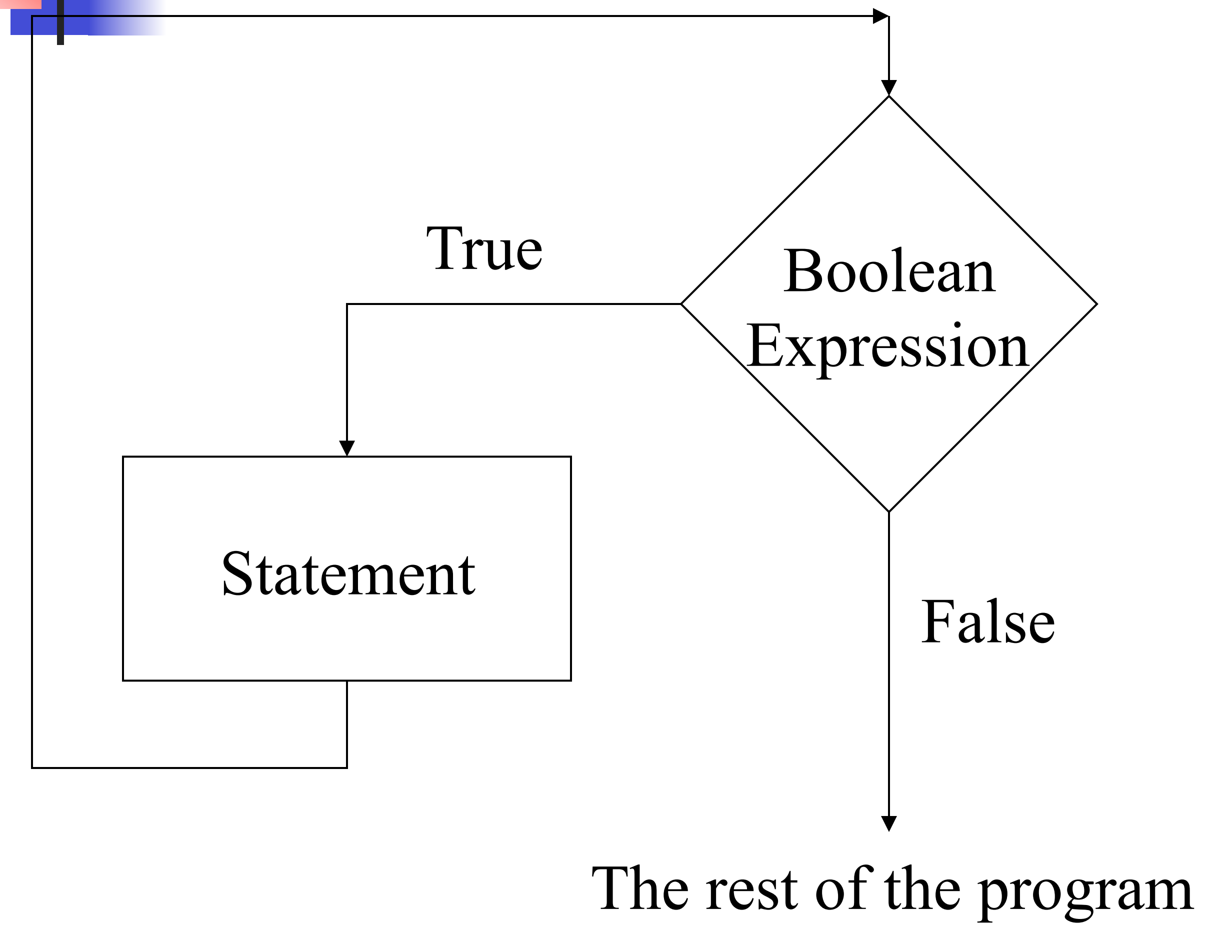
while Statement

- Repeat some action as long as a specified condition is true

```
while(<boolean expression>)
    <statement>
```

- Repeatedly execute <statement> until <boolean expression> is false
  - May not execute <statement> at all

Flowchart For A while Statement



while Statements in Action

```
int value = 0;
while(value < 5)
    System.out.println(value++);
```

```
char c = ‘a’;
while(c != ‘x’) {
    c = in.nextChar();
    System.out.println(c);
}
```

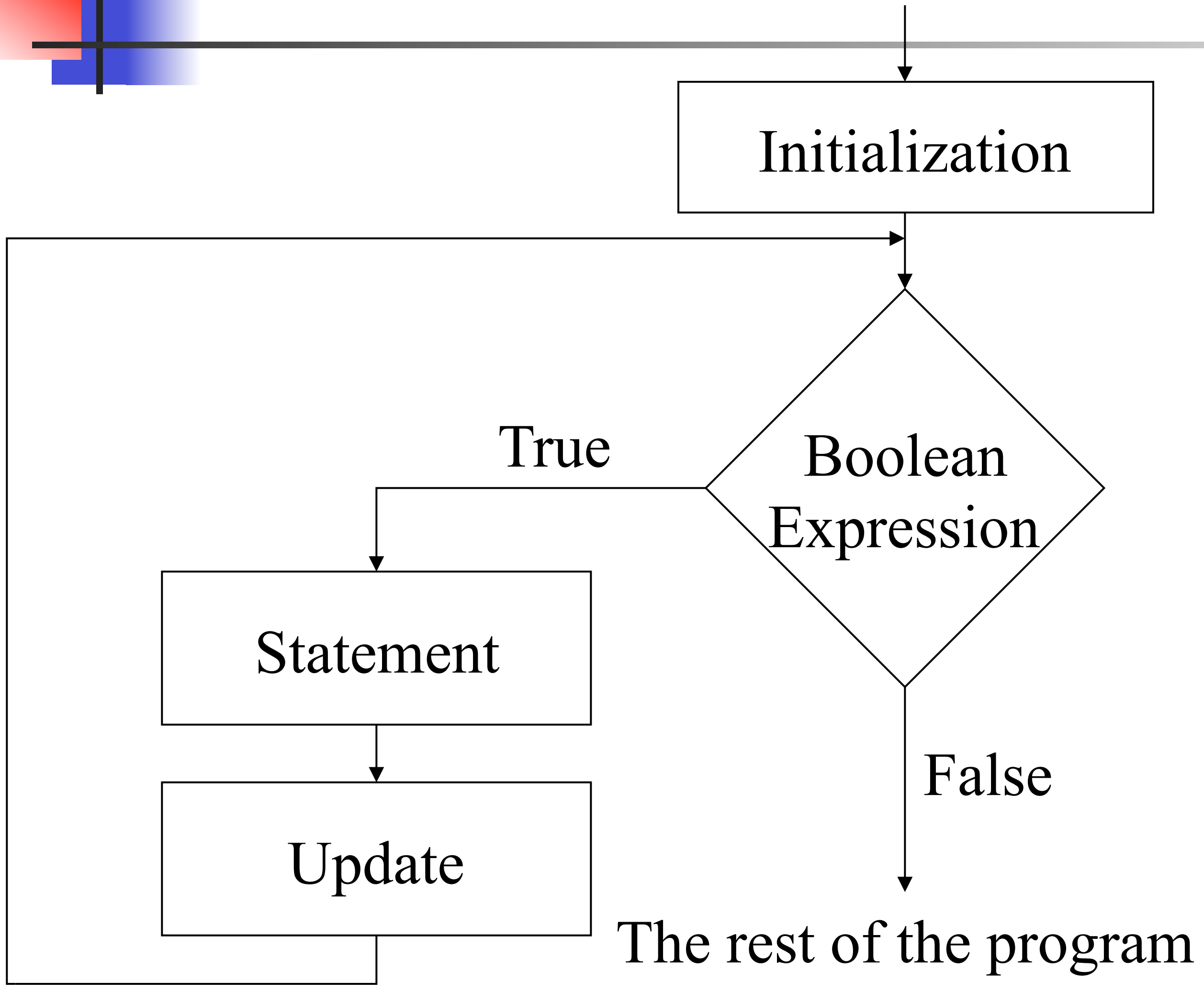
for Statement

- Repeat some action as long as a specified condition is true

```
for(<init>; <boolean>; <update>)
    <statement>
```

- <init> - executed once, at the beginning
- <boolean> - checked each time through the loop, before <statement>
- <update> - executed each time, after <statement>
- for statements are the same as while statements, except that init and update are explicitly included

Flowchart For A for Statement



for Statements in Action

```
for(int value = 0; value < 5; value++)
    System.out.println(value);
```

```
for(char c = 'a'; c != 'x'; System.out.print(c))
    c = in.next().charAt(0);
```

```
for(int i = 1, j = 1,t; i < 100; j = t +j) {
    System.out.println(i);
    t = i;
    i = j;
}
```

break and continue

- break and continue interrupt the flow of control in a while loop, for loop, or switch statement
- break
  - Jumps out of a while or for loop
    - With nested loops, jumps out of innermost one only
  - Causes a switch statement to terminate
    - If you omit it after a case, control drops into the next case
- continue
  - Terminates the *current* iteration of a loop

break Statement in Action

```
char c = 'a';
while(c != 'x') {
    c = in.next().charAt();
    System.out.print(c);
}
```

```
while(true) {
    c = in.next().charAt();
    System.out.println(c);
    if(c == 'x')
        break;
}
```

continue Statement in Action

```
for(int i = 0; i < 100; i++) {
    if(i % 2 == 1)
        continue;
    System.out.println(i);
}
```

switch Statement

- Choose among several alternative actions

```
switch(<controlling expression>) {    // integer variable or value
   case value1:                  // if <ce> == value1, do <statement1>
       <statement1>
       break;
   case value2 :                  // if <ce> == value2, do <statement2>
        <statement2>
        break;
   <… more cases here …>
   case valuen :                  // if <ce> == valuen, do <statementn>
         <statementn>
         break;
}
```

switch Statements in Action

```
switch(dayOfWeek) {
    case 1:
        System.out.println(“Sunday”);
        break;
    case 2:
        System.out.println(“Monday”);
        break;
    <etc.>
    default:
        System.out.println(“Huh?”);
        break;
}
```