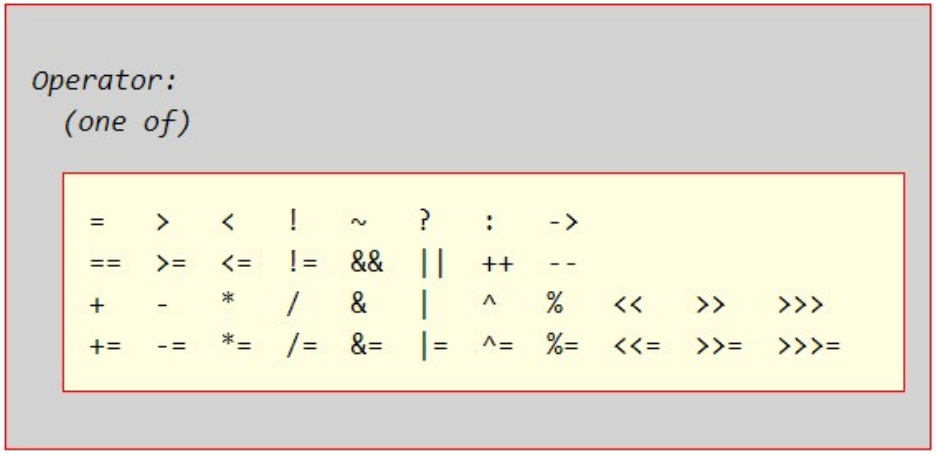
Java Operators

- A Java operator is a special symbol that can be applied to a set of variables, values, or literals—referred to as operands—and that returns a result.

- 38 tokens, formed from ASCII characters, are the operators.



Order of operator precedence

- If two operators have the **same** level of precedence, then Java guarantees **left-to-right** evaluation.

Operator	Symbols and examples
Post-unary operators	expression++, expression--
Pre-unary operators	++expression, --expression
Other unary operators	+, -, !,~
Multiplication/Division/Modulus	*, /, %
Addition/Subtraction	+, -
Shift operators	<<, >>, >>>
Relational operators	<, >, <=, >=, instanceof
Equal to/not equal to	==, !=
Logical operators	&, ^, |
Short-circuit logical operators	&&, ||
Ternary operators	boolean expression ? expression1 : expression2
Assignment operators	=, +=, -=, *=, /=, %=, &=, ^=, !=, <<=, >>=, >>>=

Binary Arithmetic Operators

- Include addition (+), subtraction (-), multiplication (*), division (/), and modulus (%)
  - int x = 2 * 5 + 3 * 4 - 8;
- you can change the order of operation explicitly by wrapping parentheses around the sections you want evaluated first.
  - int x = 2 * ((5 + 3) * 4 – 8);

Numeric Promotion Rules

1. If two values have different data types, Java will **automatically promote one of the values to the larger** of the two data types:
   - If one of the values is integral and the other is floating-point, Java will automatically promote the integral value to the floating-point value's data type.
2. **Smaller data types**, namely byte, short, and char, are **first promoted to int any time** they're used with a Java binary arithmetic operator, **even if neither** of the operands **is int**.
3. After all promotion has occurred and the operands have the same data type, **the resulting value** will have the **same** data type as its **promoted operands**

Numeric Promotion examples

- What is the data type of x * y?

```
int x = 1;
long y = 33;
```

- What is the data type of x + y?

```
double x = 39.21;
float y = 2.1;
```

- What is the data type of x / y?

```
short x = 10;
short y = 3;
```

- What is the data type of x * y / z?

```
short x = 14;
float y = 13;
double z = 30;
```

Unary operator

Unary operator	Description
+	Indicates a number is positive, although numbers are positive by default in Java
-	Indicates a literal number is negative or negates an expression
++	Increments a value by 1
--	Decrements a value by 1
~	Bitwise unary not operator
!	Inverts a Boolean's logical value

- What is the value of x?

```
boolean x = false;
x = !x;
```

- What is the value of x?

```
double x = 1.21;
x = -x;
```

- What is the value of x, y?

```
int x = 3;
int y = ++x * 5 / x-- + --x;
```

```
int x = !5
boolean y = -true;
boolean z = !0;
```

Assignment Operators

- An assignment operator is a binary operator that modifies, or assigns, the variable on the left-hand side of the operator, with the result of the value on the right-hand side of the equation.
  - The simplest assignment operator is the = assignment

    ```
    int x = 1;
    ```
  - Compound Assignment Operators

    ```
    int x = 2, z = 3;
     x = x * z;          // Simple assignment operator
    x *= z;              // Compound assignment operator
    ```

Relational Operators

- Relational operators, which compare two expressions and return a boolean value.

Relational operator	Description
<	Strictly less than
<=	Less than or equal to
>	Strictly greater than
>=	Greater than or equal to
==	Equal to
!=	Not qual to
a instanceof b	True if the reference that a points to is an instance of a class, subclass, or class that implements a particular interface, as named in b

Relational Operators

- The equality operators are used in one of three scenarios:
  1. Comparing two numeric primitive types.
  2. Comparing two boolean values.
  3. Comparing two objects, including null and String values.

  For object comparison, the equality operator is applied to the **references** to the objects, not the objects they point to. Two references are equal if and only if they point to the same object, or both point to null.

```
File x = new File("myFile.txt");
File y = new File("myFile.txt");
File z = x;
System.out.println(x == y); // Outputs false
System.out.println(x == z); // Outputs true
```

- The logical operators, (&), (|), and (^), may be applied to both numeric and boolean data types.
- The conditional operators (short-circuit operators), && and ||.
  - They operates only on boolean data types.
  - the right-hand side of the expression may never be evaluated if **the final result can be determined by the left-hand side** of the expression.

x & y (AND)

	y = true	y = false
x = true	true	false
x = false	false	false

x | y (INCLUSIVE OR)

	y = true	y = false
x = true	true	true
x = false	true	false

x ^ y (EXCLUSIVE OR)

	y = true	y = false
x = true	false	true
x = false	true	false

Example:

```
boolean x = true || (y < 4);
//y < 4 will never be evaluated
```

Quick exercise

- Evaluate this expression:

```
3 + 4 * 4 > 5 * (4 + 3) - 1 && (4 - 3 > 5)
```

Operator	Symbols and examples
Post-unary operators	expression++, expression--
Pre-unary operators	++expression, --expression
Other unary operators	+, -, !,~
Multiplication/Division/Modulus	*, /, %
Addition/Subtraction	+, -
Shift operators	<<, >>, >>>
Relational operators	<, >, <=, >=, instanceof
Equal to/not equal to	==, !=
Logical operators	&, ^, |
Short-circuit logical operators	&&, ||
Ternary operators	boolean expression ? expression1 : expression2
Assignment operators	=, +=, -=, *=, /=, %=, &=, ^=, !=, <<=, >>=, >>>=

Java Statements

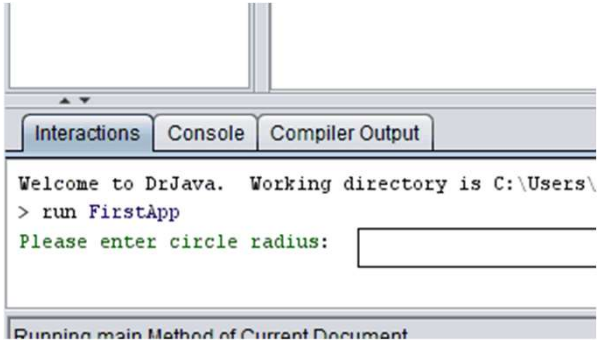
Selections

Why we use selections?

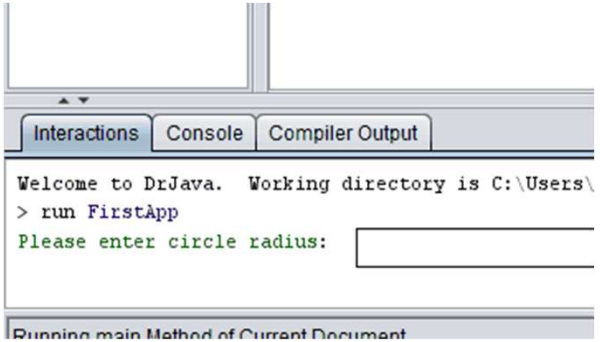
- The program can decide which statements to execute based on a condition.
- For example :

  the program of computing circle area: in case of the user input is a negative circle radius, the program must deal with this case. But how?

✓ Correct



× Wrong



Boolean Data Type

- Selection statements use conditions that are Boolean expressions. A Boolean expression is an expression that evaluates to a Boolean value: true or false.
- The **boolean data type** declares a variable with the Boolean value (value either true or false).
- Java provides six relational operators (also known as comparison operators), shown the following table , which can be used to compare two values (assume radius is 5 in the table)

Java Operator	*Mathematics Symbol*	*Name*	*Example (radius is 5)*	*Result*
<	<	less than	`radius < 0`	`false`
<=	≤	less than or equal to	`radius <= 0`	`false`
>	>	greater than	`radius > 0`	`true`
>=	≥	greater than or equal to	`radius >= 0`	`true`
==	=	equal to	`radius == 0`	`false`
!=	≠	not equal to	`radius != 0`	`true`

Variable: is a named address in memory that hold value and the Data Type: specifies this value type.

IF Statement – One-Way if Statement

- A one-way if statement executes an action if and only if the condition is true. If the condition is false, nothing is done.

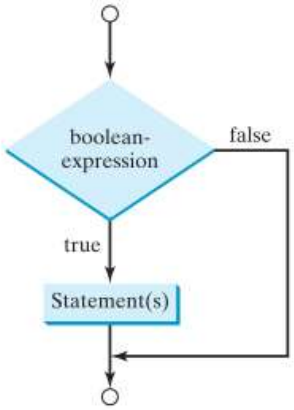
```
if (boolean-expression) {
  statement(s);
}
```

Syntax

rule:

ex:

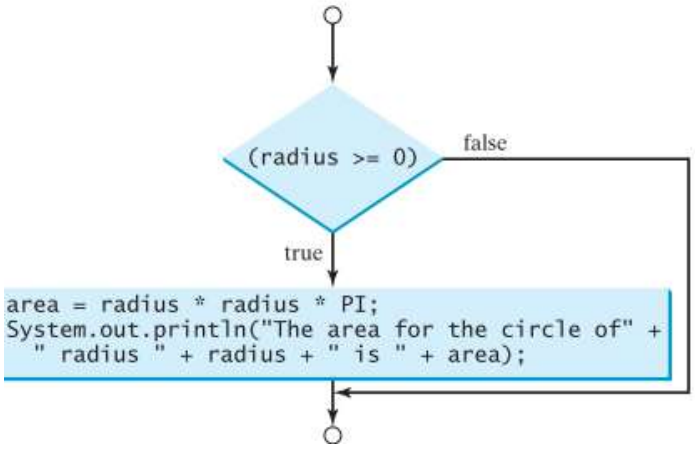
```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area for the circle of radius " +
    radius + " is " + area);
}
```



Flowchart

rule:

ex:



IF Statement – Quick notes

- The boolean-expression must be enclosed in parentheses "()".

× Wrong

```
if i > 0 {
  System.out.println("i is positive");
}
```

✓ Correct

```
if (i > 0) {
  System.out.println("i is positive");
}
```

- The block braces can be omitted if they enclose a single statement.

```
if (i > 0) {
  System.out.println("i is positive");
}
```

Equivalent

```
if (i > 0)
  System.out.println("i is positive");
```

IF Statement – Two-Way if Statement

- A two-way if statement executes an action if the condition is true. But if it executes an alternative actions when the condition is false.

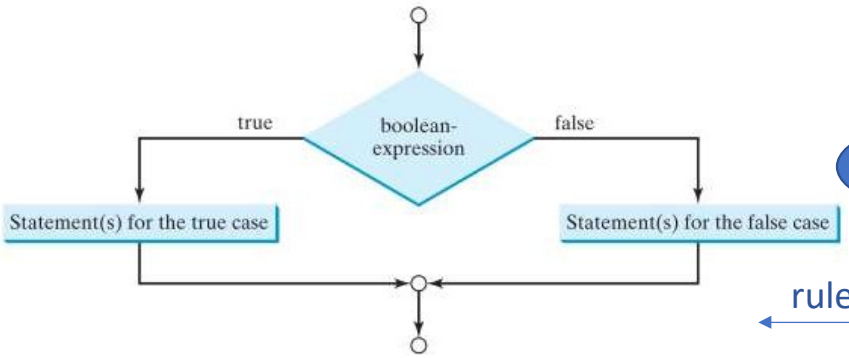
```
if (boolean-expression) {
  statement(s)-for-the-true-case;
}
else {
  statement(s)-for-the-false-case;
}
```

Syntax

rule:

ex:

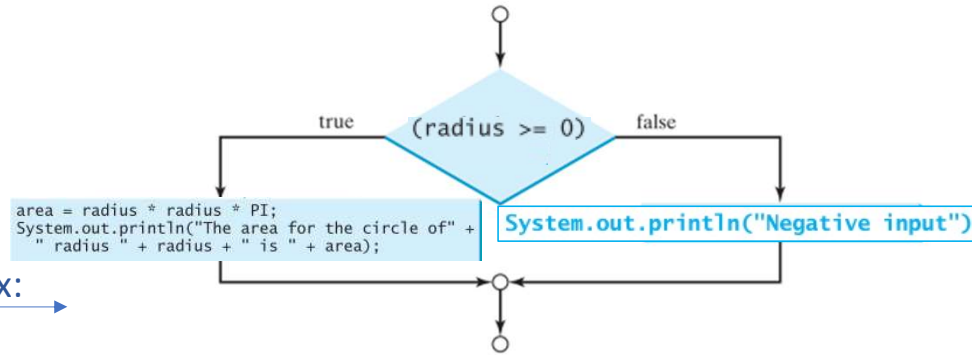
```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area for the circle of radius " +
    radius + " is " + area);
}
else {
  System.out.println("Negative input");
}
```

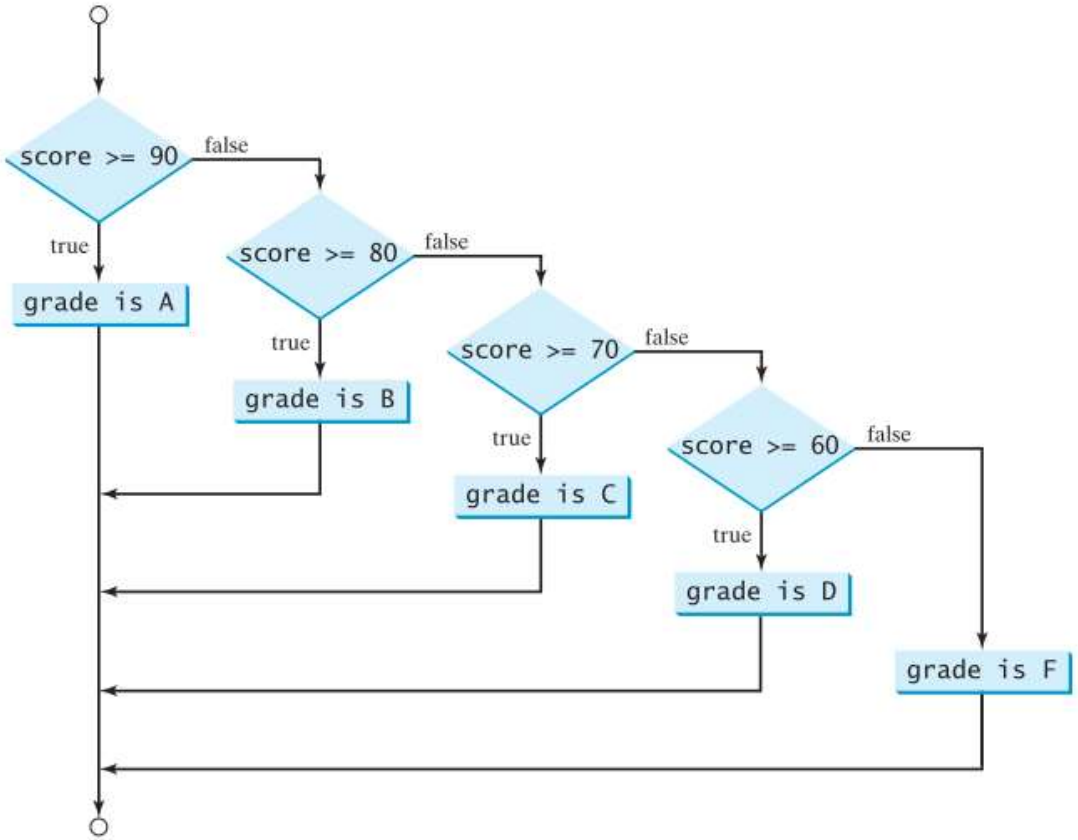


Flowchart

rule:

ex:





How can we implement this flowchart?

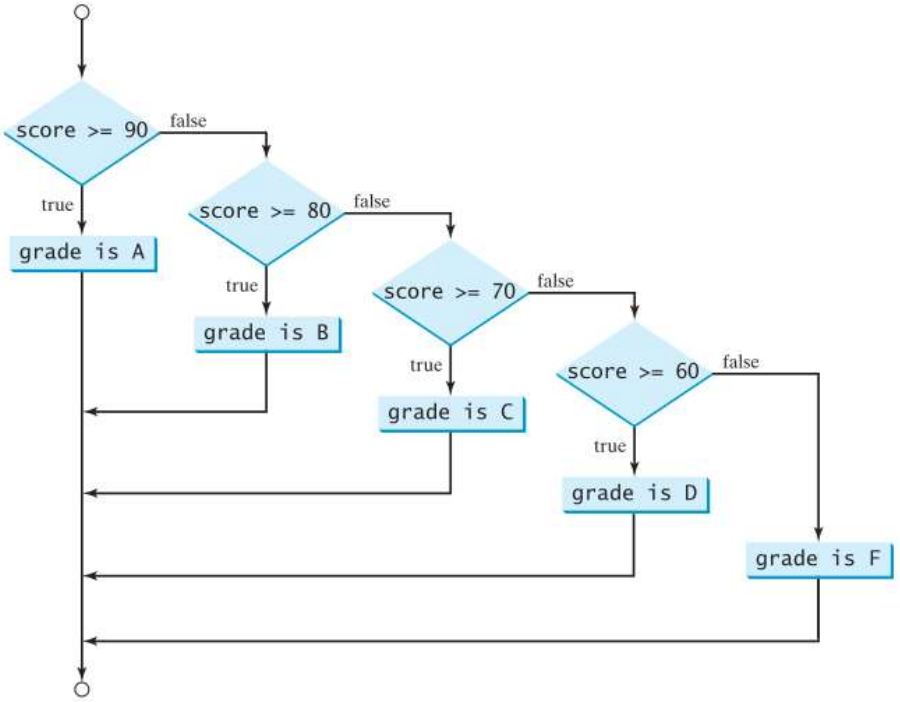
IF Statement – Nested if Statement

- An if statement can be inside another if statement to form a nested if statement.
- For example:

```
if (i > k) {
  if (j > k)
    System.out.println("i and j are greater than k");
}
else
  System.out.println("i is less than or equal to k");
```

IF Statement – Nested if Statement

- So we can implement the previous flowchart using nested if statement as follow:




```
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");
      else
        System.out.print("F");
```

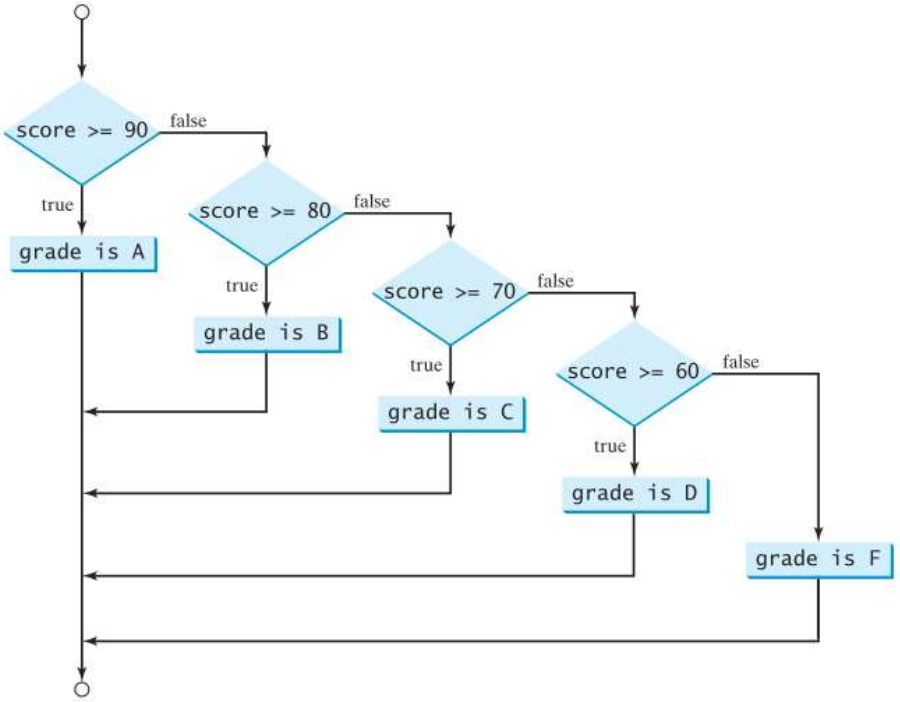
IF Statement – Multi-Way if Statement

- For example:

```
if (income <= 8350)
  tax = income * 0.10;
else if (income <= 33950)
  tax = 8350 * 0.10 + (income - 8350) * 0.15;
else if (income <= 82250)
  tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
    (income - 33950) * 0.25;
else if (income <= 171550)
  tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
    (82250 - 33950) * 0.25 + (income - 82250) * 0.28;
else if (income <= 372950)
  tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
    (82250 - 33950) * 0.25 + (171550 - 82250) * 0.28 +
    (income - 171550) * 0.33;
else
  tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
    (82250 - 33950) * 0.25 + (171550 - 82250) * 0.28 +
    (372950 - 171550) * 0.33 + (income - 372950) * 0.35;
```

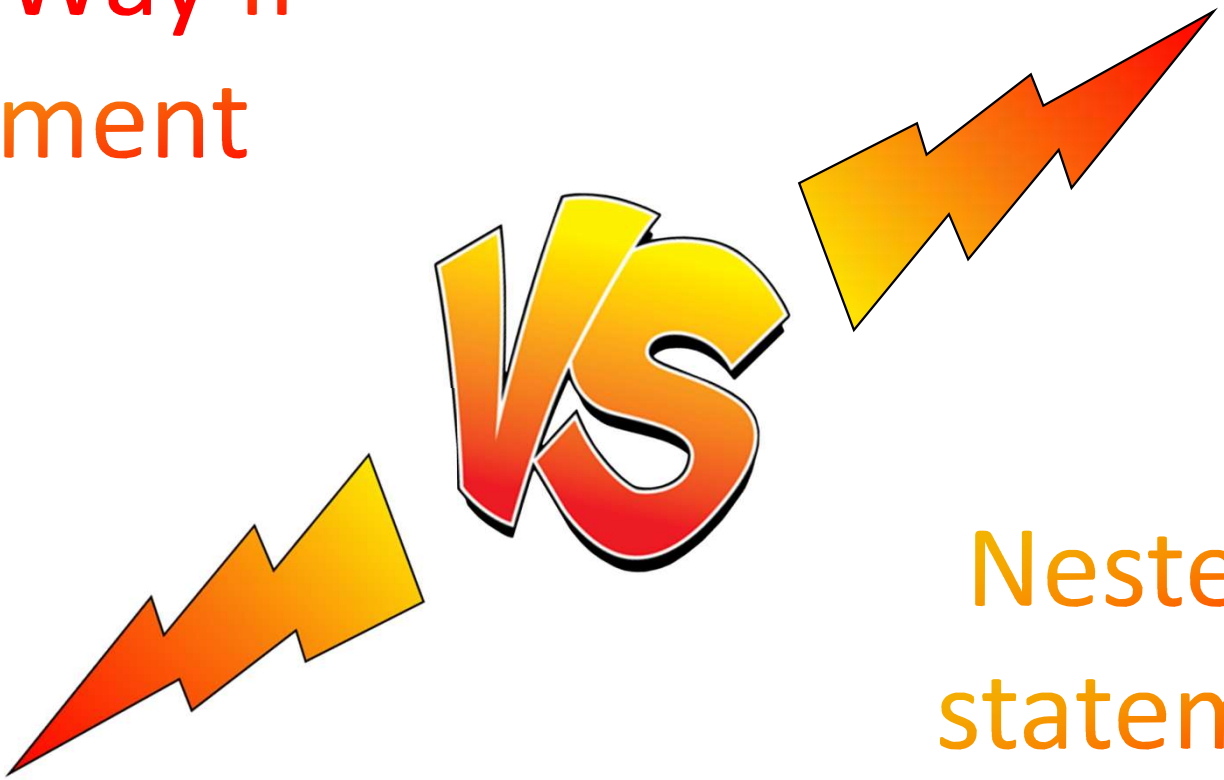
IF Statement – Multi-Way if Statement

- So we can implement the previous flowchart using Multi-Way if statement as follow:



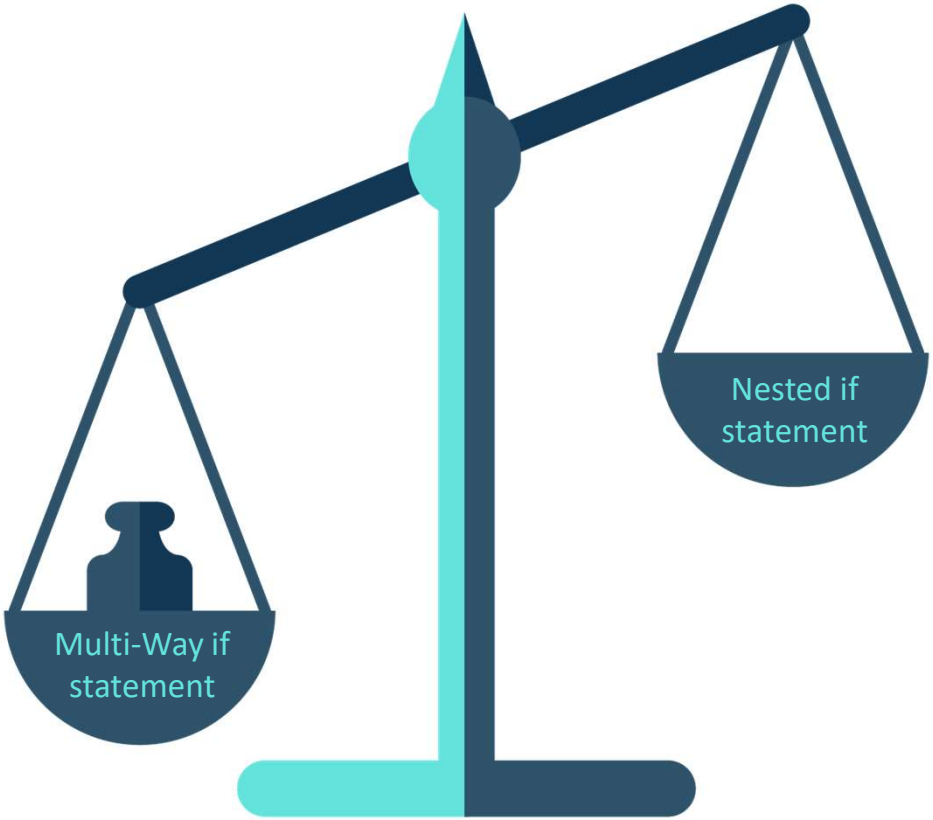

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

Multi-Way if statement



Nested if statement

- The preferred coding style is multi-way if-else statements, as it avoids deep indentation and makes the program easy to read.




```
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");
      else
        System.out.print("F");
```

Equivalent

This is better

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

Common Errors And Pitfalls

1. Forgetting Necessary Braces

× Wrong

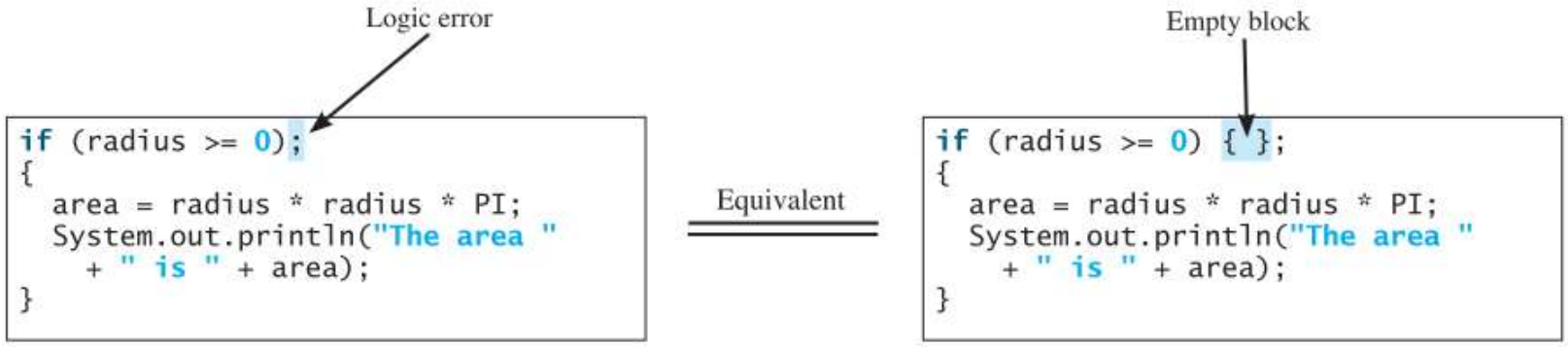
```
if (radius >= 0)
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
```

✓ Correct

```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
}
```

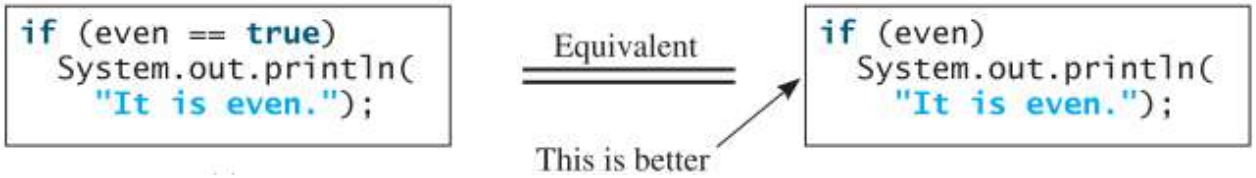
Common Errors And Pitfalls

2. Wrong Semicolon at the if Line



Common Errors And Pitfalls

3. Redundant Testing of Boolean Values



- To avoid redundancy
- To avoid errors that are difficult to detect.
  - For Example: Using the = operator instead of the == operator to compare the equality of two items in a test condition

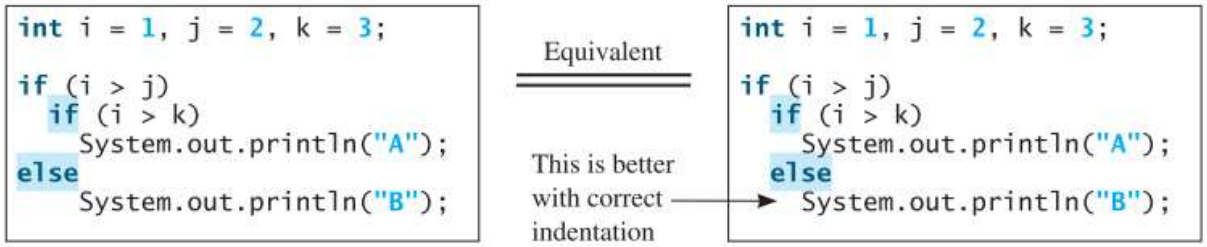
```
if (even = true)
  System.out.println("It is even.");
```

  - This statement does not have compile errors. It assigns true to even, so that even is always true.

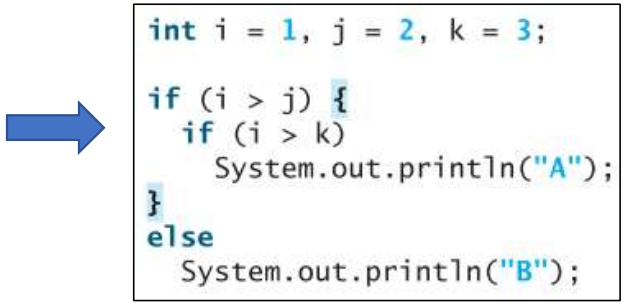
Common Errors And Pitfalls

4. Dangling else Ambiguity

- The indentation indicates that the else clause matches the first if clause. However, the else clause actually matches the second if clause.



- To force the else clause to match the first if clause, you must add a pair of braces:



Common Errors And Pitfalls

5. Equality Test of Two Floating-Point Values

- Equality test of two floating-point values is not reliable. For example, you expect the following code to display true, but surprisingly it displays false. Here, x is not exactly 0.5, but is 0.5000000000000001.

```
double x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;
System.out.println(x == 0.5);
```

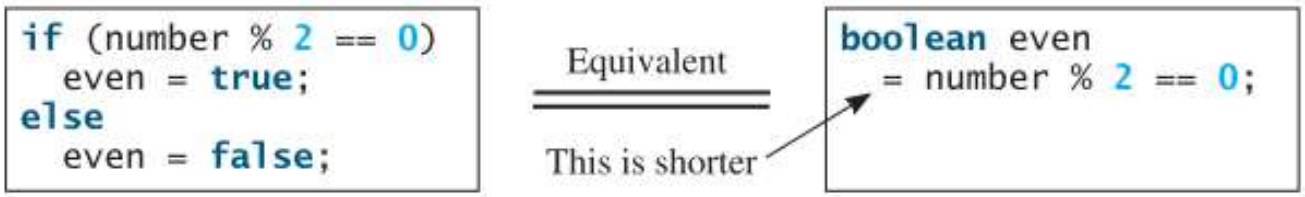
- However, you can compare whether they are close enough by testing whether the difference of the two numbers is less than some threshold.

```
final double EPSILON = 1E-14;
double x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;
if (Math.abs(x - 0.5) < EPSILON)
  System.out.println(x + " is approximately 0.5");
```

Common Errors And Pitfalls

6. Simplifying Boolean Variable Assignment

- This is not an error, but it should be better written as shown



Common Errors And Pitfalls

7. Avoiding Duplicate Code in Different Cases

- This is not an error, but it should be better written as follows:

```
if (inState) {
   tuition = 5000;
   System.out.println("The tuition is " + tuition);
}
else {
   tuition = 15000;
   System.out.println("The tuition is " + tuition);
}
```

Equivalent

```
if (inState) {
   tuition = 5000;
}
else {
   tuition = 15000;
}
System.out.println("The tuition is " + tuition);
```

This is better

Common Errors And Pitfalls

- In mathematics, the expression is correct.

```
1 <= numberOfDaysInAMonth <= 31
```

- However, it is incorrect in Java, because 1 <= numberOfDaysInAMonth is evaluated to a boolean value, which cannot be compared with 31. Here, two operands (a boolean value and a numeric value) are incompatible. The correct expression in Java is

```
(1 <= numberOfDaysInAMonth) && (numberOfDaysInAMonth <= 31)
```

SWITCH Statement

- A switch statement executes statements based on the value of a variable or an expression.

```
switch (switch-expression) {
  case value1: statement(s)1;
               break;
  case value2: statement(s)2;
               break;
  ...
  case valueN: statement(s)N;
               break;
  default:     statement(s)-for-default;
}
```

Syntax

rule:

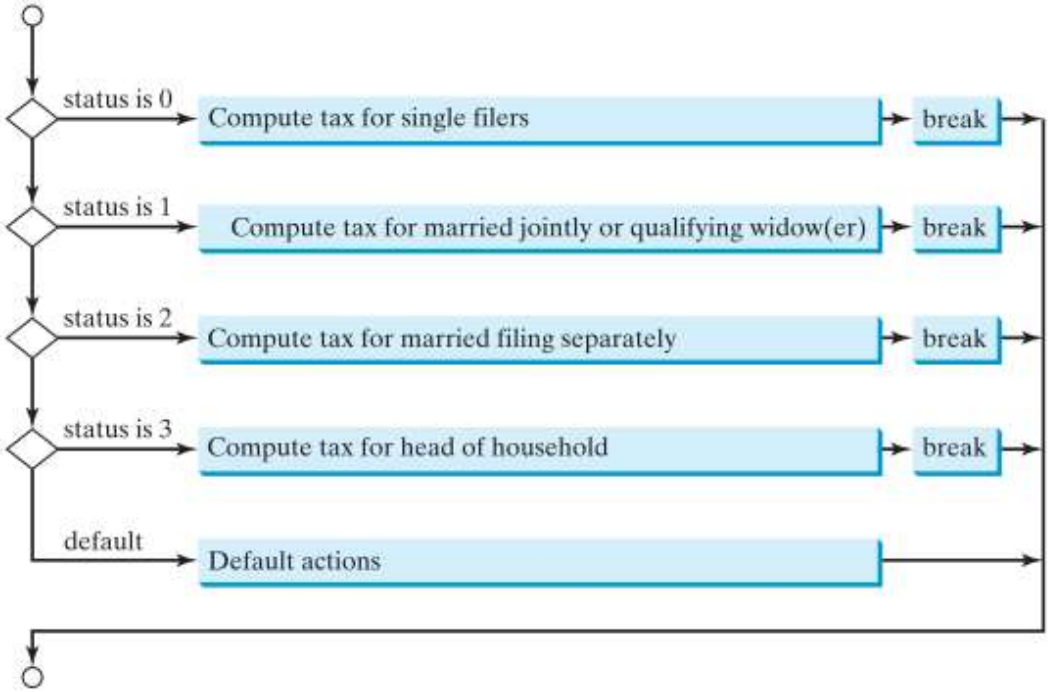
ex:

```
switch (status) {
  case 0:  compute tax for single filers;
           break;
  case 1:  compute tax for married jointly or qualifying widow(er);
           break;
  case 2:  compute tax for married filing separately;
           break;
  case 3:  compute tax for head of household;
           break;
  default: System.out.println("Error: invalid status");
           System.exit(1);
}
```

SWITCH Statement cont.

Flowchart

ex:



SWITCH Statement rules

- The switch-expression must yield a value of char, byte, short, int, or String type and must always be enclosed in parentheses. (The char and String types will be introduced in the next chapter.)
- The value1, . . ., and valueN must have the same data type as the value of the switch-expression. Note that value1, . . ., and valueN are constant expressions, meaning that they cannot contain variables, such as 1 + x.
- When the value in a case statement matches the value of the switch-expression, the statements starting from this case are executed until either a break statement or the end of the switch statement is reached.
- The default case, which is optional, can be used to perform actions when none of the specified cases matches the switch-expression.
- The keyword break is optional. The break statement immediately ends the switch statement.

Quick Exercise

- What is the output in case of 2, 5, 0, and 6?

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

- What is y after the following switch statement is executed? Rewrite the code using an if-else statement.

```
x = 3; y = 3;
switch (x + 3) {
  case 6:  y = 1;
  default: y += 1;
}
```

- What does the preceding code's running print?

```
int x = 2;
switch (x) {
  case 2:  System.out.println("2");
  default: System.out.println("default");
  case 3: System.out.println("3");
  case 4: System.out.println("4");
}
```

```
int x = 7;
switch (x) {
  case 2:  System.out.println("2");
  default: System.out.println("default");
  case 3: System.out.println("3");
  case 4: System.out.println("4");
}
```

Conditional Expressions(Ternary Operator)

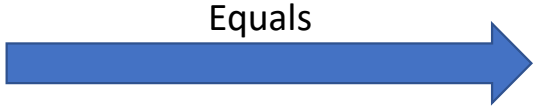
- A conditional expression evaluates an expression based on a condition.
- The result of this conditional expression is expression1 if boolean-expression is true; otherwise the result is expression2.

```
boolean-expression ? expression1 : expression2;
```

- Example:

```
if (x > 0)
  y = 1;
else
  y = -1;
```

Equals



```
y = (x > 0) ? 1 : -1;
```

Conditional Expressions cont.

- Example

```
max = (num1 > num2) ? num1 : num2;
```

```
System.out.println((num % 2 == 0) ? "num is even" : "num is odd");
```

- The symbols ? and : appear together in a conditional expression. They form a conditional operator and also called a ternary operator because it uses three operands. It is the only ternary operator in Java.

Quick exercise

- Rewrite the following if statements using the conditional operator.

```
if (ages >= 16)
  ticketPrice = 20;
else
  ticketPrice = 10;
```


TIME TO
EXERCISE

1. Determining the Chinese Zodiac sign

- Write a program to find out the Chinese Zodiac sign for a given year. The Chinese Zodiac is based on a twelve-year cycle, with each year represented by an animal: monkey, rooster, dog, pig, rat, ox, tiger, rabbit, dragon, snake, horse, or sheep—in this cycle, as shown in Figure 3.6. Note that year % 12 determines the Zodiac sign. 1900 is the year of the rat because 1900 % 12 is 4. Listing 3.9 gives a program that prompts the user to enter a year and displays the animal for the year.

year % 12 =
0: monkey
1: rooster
2: dog
3: pig
4: rat
5: ox
6: tiger
7: rabbit
8: dragon
9: snake
10: horse
11: sheep

```
Enter a year: 1963 Enter
rabbit
```

```
Enter a year: 1877 Enter
ox
```

2. Use the &&, || and ^ operators

- Write a program that prompts the user to enter an integer and determines whether it is divisible by 5 and 6, whether it is divisible by 5 or 6, and whether it is divisible by 5 or 6, but not both. Here is a sample run of this program:

```
Enter an integer: 10 Enter
Is 10 divisible by 5 and 6? false
Is 10 divisible by 5 or 6? true
Is 10 divisible by 5 or 6, but not both? true
```

3. Palindrome Number

- Write a program that prompts the user to enter a three-digit integer and determines whether it is a palindrome number. A number is palindrome if it reads the same from right to left and from left to right. Here is a sample run of this program:

```
Enter a three-digit integer: 121 ↵
121 is a palindrome
```

```
Enter a three-digit integer: 123 ↵
123 is not a palindrome
```

4. Find The Number Of Days In A Month

- Write a program that prompts the user to enter the month and year and displays the number of days in the month. For example, if the user entered month 2 and year 2012, the program should display that February 2012 had 29 days. If the user entered month 3 and year 2015, the program should display that March 2015 had 31 days.

5. Find Future Dates

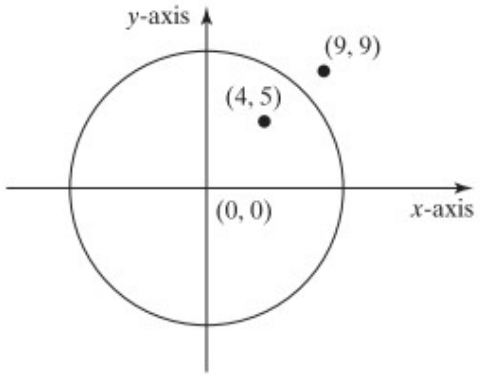
- Write a program that prompts the user to enter an integer for today's day of the week (Sunday is 0, Monday is 1, …, and Saturday is 6). Also prompt the user to enter the number of days after today for a future day and display the future day of the week. Here is a sample run:

```
Enter today's day: 1 Enter
Enter the number of days elapsed since today: 3 Enter
Today is Monday and the future day is Thursday
```

```
Enter today's day: 0 Enter
Enter the number of days elapsed since today: 31 Enter
Today is Sunday and the future day is Wednesday
```

6. Point in a circle

- (Geometry: point in a circle?) Write a program that prompts the user to enter a point (x, y) and checks whether the point is within the circle centered at (0, 0)with radius 10. For example, (4, 5) is inside the circle and (9, 9) is outside the circle.
- (Hint: A point is in the circle if its distance to (0, 0) is less than or equal to 10. The formula for computing the distance is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. Test your program to cover all cases.) Two sample runs are shown below.




```
Enter a point with two coordinates: 4 5 Enter
Point (4.0, 5.0) is in the circle
```

```
Enter a point with two coordinates: 9 9 Enter
Point (9.0, 9.0) is not in the circle
```

7. Compute The Perimeter Of A Triangle

- Write a program that reads three edges for a triangle and computes the perimeter if the input is valid. Otherwise, display that the input is invalid. The input is valid if the sum of every pair of two edges is greater than the remaining edge.

8. Solve Quadratic Equations

- The two roots of a quadratic equation $ax^2 + bx + c = 0$ can be obtained using the following formula:

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

- $b^2 - 4ac$ is called the discriminant of the quadratic equation. If it is positive, the equation has two real roots. If it is zero, the equation has one root. If it is negative, the equation has no real roots. Write a program that prompts the user to enter values for a, b, and c and displays the result based on the discriminant. If the discriminant is positive, display two roots. If the discriminant is 0, display one root. Otherwise, display "The equation has no real roots".Note that you can use Math.pow(x, 0.5) to compute 2x. Here are some sample runs.

```
Enter a, b, c: 1.0 3 1 Enter
The equation has two roots -0.381966 and -2.61803
```

```
Enter a, b, c: 1 2.0 1 Enter
The equation has one root -1
```

```
Enter a, b, c: 1 2 3 Enter
The equation has no real roots
```

9. Determining Leap Year

- Write a program that prompts the user to enter year and determined if it is a leap year or not.(A year is a leap year if it is divisible by 4 but not by 100, or if it is divisible by 400.)

```
Enter a year:  2008 ↵Enter
2008 is a leap year? true
```

```
Enter a year:  1900 ↵Enter
1900 is a leap year? false
```

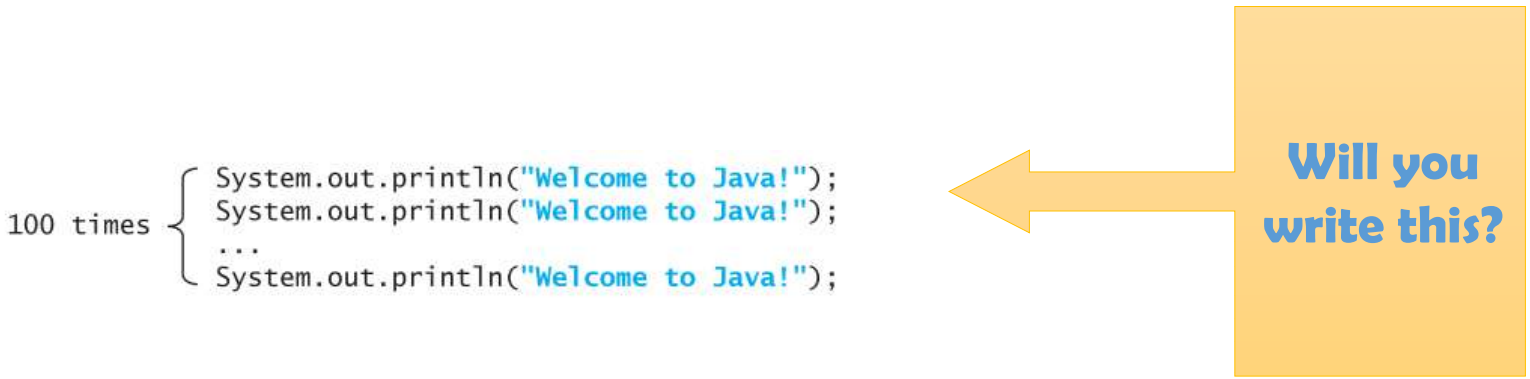
```
Enter a year:  2002 ↵Enter
2002 is a leap year? false
```

Java Statements

Repetition

Why we use repetitions?

- A loop can be used to tell a program to execute statements repeatedly.
- For example:
  - Suppose that you need to display a string (e.g., Welcome to Java!) a hundred times,

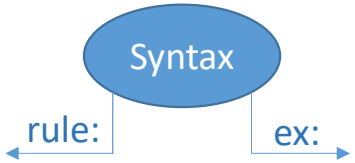


- Java provides a powerful construct called a loop that controls how many times an operation or a sequence of operations is performed in succession.

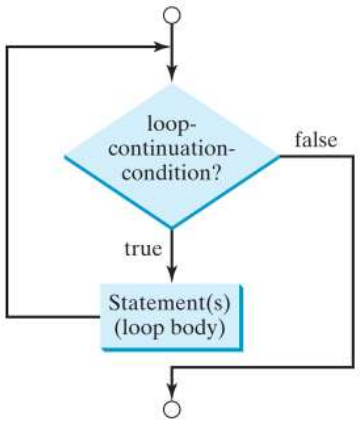
The while Loop

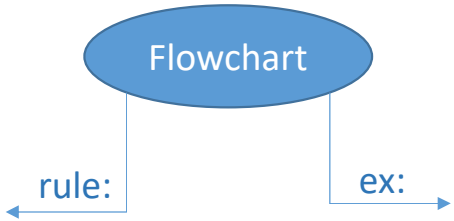
- A while loop executes statements repeatedly while the condition is true.

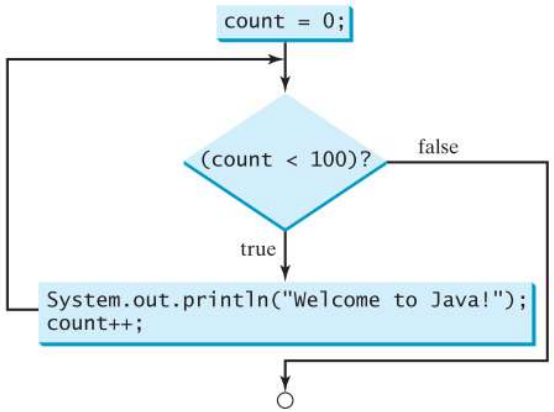
```
while (loop-continuation-condition) {
  // Loop body
  Statement(s);
}
```




```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java!");
  count++;
}
```







Loop - Quick notes

- The loop-continuation-condition must always appear inside the parentheses.
- The braces enclosing the loop body can be omitted only if the loop body contains one or no statement.
- Make sure that the loop-continuation-condition eventually becomes false so that the loop will terminate.
- If your program takes an unusually long time to run and does not stop, it may have an infinite loop. If you are running the program from the command window, press CTRL+C to stop it.
- Programmers often make the mistake of executing a loop one more or less time:

```
int count = 0;
while (count <= 100) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Loop Design Strategies

- Step 1: Identify the statements that need to be repeated.
- Step 2: Wrap these statements in a loop like this:

```
while (true) {
  Statements;
}
```

- Step 3: Code the loop-continuation-condition and add appropriate statements for controlling the loop.

```
while (loop-continuation-condition) {
  Statements;
  Additional statements for controlling the loop;
}
```

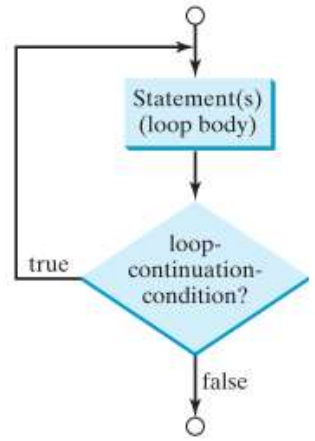
The do-while Loop

- A do-while loop is the same as a while loop except that it executes the loop body first and then checks the loop continuation condition.

```
do {
  // Loop body;
  Statement(s);
} while (loop-continuation-condition);
```

Syntax

rule:



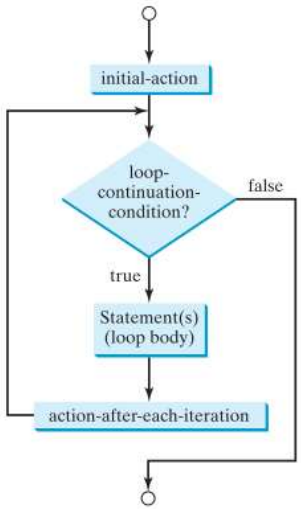
Flowchart

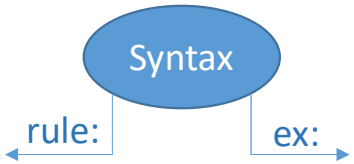
rule:

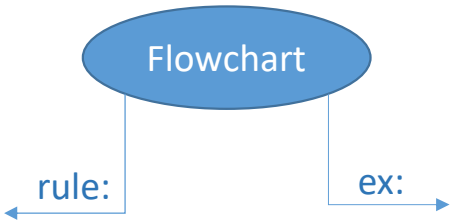
The for Loop

- A for loop has a concise syntax for writing loops.

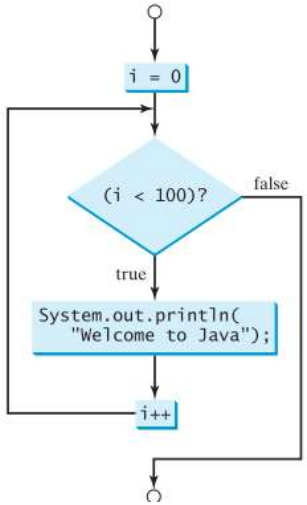
```
for (initial-action; loop-continuation-condition;
     action-after-each-iteration) {
  // Loop body;
  Statement(s);
}
```








```
for (i = initialValue; i < endValue; i++)
  // Loop body
  ...
}
```



Quick notes

- The control variable must be declared inside the control structure of the loop or before the loop. If the loop control variable is used only in the loop, not elsewhere.
- The initial-action in a for loop can be a list of zero or more comma-separated variable declaration statements or assignment expressions. For example:

```
for (int i = 0, j = 0; i + j < 10; i++, j++) {
  // Do something
}
```

- The action-after-each-iteration in a for loop can be a list of zero or more comma-separated statements. For example:

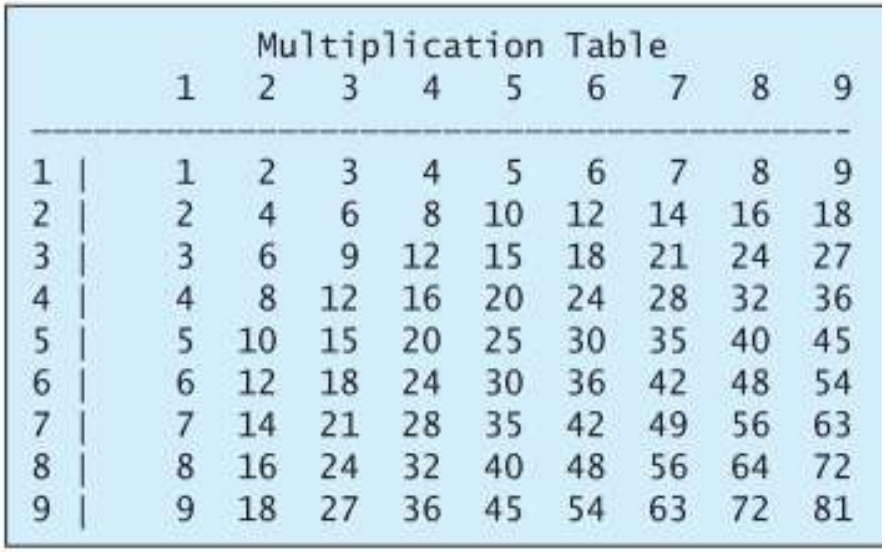
```
for (int i = 1; i < 100; System.out.println(i), i++);
```

Which Loop To Use?

- A for loop may be used if the number of repetitions is known in advance, as, for example, when you need to display a message a hundred times.
- A while loop may be used if the number of repetitions is not fixed, as in the case of reading the numbers until the input is 0.
- A do-while loop can be used to replace a while loop if the loop body has to be executed before the continuation condition is tested.

Nested Loops

- Nested loops consist of an outer loop and one or more inner loops. Each time the outer loop is repeated, the inner loops are reentered, and started anew.
- For example: printing multiplication table.



Multiplication Table

	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

Quick Exercise

- How many times is the println statement executed?

```
for (int i = 0; i < 10; i++)
  for (int j = 0; j < i; j++)
    System.out.println(i * j)
```

- Show the output of the following programs. (Hint: draw a table and list the variables in the columns to trace these programs.)

```
public class Test {
  public static void main(String[] args) {
    for (int i = 1; i < 5; i++) {
      int j = 0;
      while (j < i) {
        System.out.print(j + " ");
        j++;
      }
    }
  }
}
```

(a)

```
public class Test {
  public static void main(String[] args) {
    int i = 0;
    while (i < 5) {
      for (int j = i; j > 1; j--)
        System.out.print(j + " ");
      System.out.println("*****");
      i++;
    }
  }
}
```

(b)

```
public class Test {
  public static void main(String[] args) {
    int i = 5;
    while (i >= 1) {
      int num = 1;
      for (int j = 1; j <= i; j++) {
        System.out.print(num + "xxx");
        num *= 2;
      }

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

(c)

```
public class Test {
  public static void main(String[] args) {
    int i = 1;
    do {
      int num = 1;
      for (int j = 1; j <= i; j++) {
        System.out.print(num + "G");
        num += 2;
      }

      System.out.println();
      i++;
    } while (i <= 5);
  }
}
```

(d)

Keyword “break”

- You can also use break in a loop to immediately terminate the loop.

```
public class TestBreak {
  public static void main(String[] args) {
    int sum = 0;
    int number = 0;

    while (number < 20) {
      number++;
      sum += number;


    System.out.println("The number is " + number);
    System.out.println("The sum is " + sum);
  }
}
```

Without if

```
The number is 20
The sum is 210
```

```
The number is 14
The sum is 105
```

Keyword Continue

- When it is encountered, it ends the current iteration and program control goes to the end of the loop body.

```
    int number = 0;

    while (number < 20) {
      number++;


      sum += number;
    }

    System.out.println("The sum is " + sum);
  }
}
```

Without if

```
The sum is 210
```

```
The sum is 189
```

Quick Exercise

- Will the following programs terminate? If so, give the output

```
int balance = 10;
while (true) {
  if (balance < 9)
    break;
  balance = balance - 9;
}

System.out.println("Balance is "
  + balance);
```

(a)

```
int balance = 10;
while (true) {
  if (balance < 9)
    continue;
  balance = balance - 9;
}

System.out.println("Balance is "
  + balance);
```

(b)

- The for loop on the left is converted into the while loop on the right. What is wrong? Correct it.

```
int sum = 0;
for (int i = 0; i < 4; i++) {
  if (i % 3 == 0) continue;
  sum += i;
}
```

Converted

Wrong conversion

```
int i = 0, sum = 0;
while (i < 4) {
  if (i % 3 == 0) continue;
  sum += i;
  i++;
}
```

Quick Exercise

- After the break statement in (a) is executed in the following loop, which statement is executed? Show the output. After the continue statement in (b) is executed in the following loop, which statement is executed? Show the output.

```
for (int i = 1; i < 4; i++) {
  for (int j = 1; j < 4; j++) {
    if (i * j > 2)
      break;

    System.out.println(i * j);
  }

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

(a)

```
for (int i = 1; i < 4; i++) {
  for (int j = 1; j < 4; j++) {
    if (i * j > 2)
      continue;

    System.out.println(i * j);
  }

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

(b)

Adding Optional Labels

- A statement can be labelled as follows.

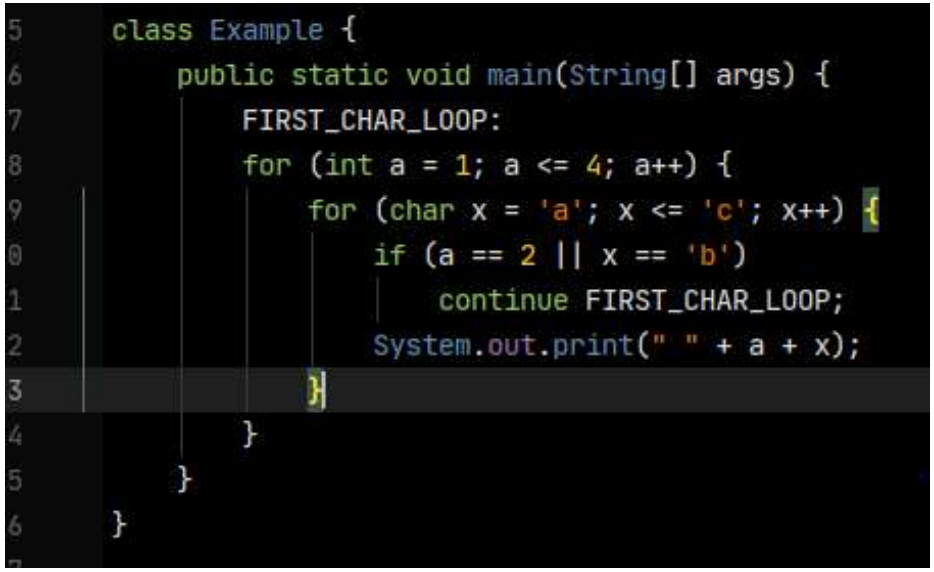
  statementName : SomeJavaStatement

- When we use break statement a long with labeles:

  break statementName;

```java
/*
This is a simple Java program.
Call this file "Example.java".
*/
class Example {
    public static void main(String args[]) {
        outer:
        for (int i = 0; i < 3; i++) {
            System.out.print("Pass " + i + ": ");
            for (int j = 0; j < 100; j++) {
                if (j == 10)
                    break outer; // exit both loops
                System.out.print(j + " ");
            }
            System.out.println("This will not print");
        }
        System.out.println("Loops complete.");
    }
}
```

Adding Optional Labels




TIME TO
EXERCISE

1. COMPUTE FACTORIAL

- Write a program that prompt the user to enter a non-negative integer and print its factorial to the user.(Note: Factorial of a non-negative integer, is multiplication of all integers smaller than or equal to n. For example factorial of 6 is 6*5*4*3*2*1 which is 720.)

```
n! = n * (n-1)!
n! = 1 if n = 0 or n = 1
```

2. Count Positive And Negative Numbers And Compute The Average Of Numbers

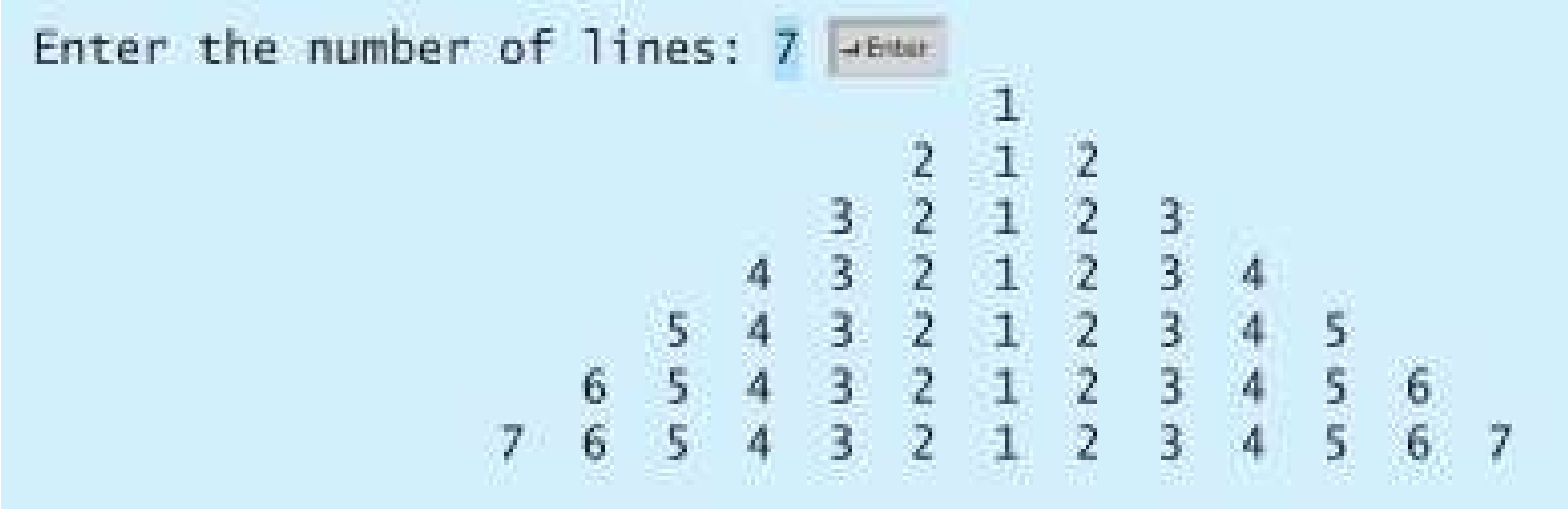
- Write a program that reads an unspecified number of integers, determines how many positive and negative values have been read, and computes the total and average of the input values (not counting zeros). Your program ends with the input 0. Display the average as a floating-point number. Here is a sample run:

```
Enter an integer, the input ends if it is 0: 1 2 -1 3 0 Enter
The number of positives is 3
The number of negatives is 1
The total is 5.0
The average is 1.25
```

```
Enter an integer, the input ends if it is 0: 0 Enter
No numbers are entered except 0
```

4. Display Pyramid

- Write a program that prompts the user to enter an integer from 1 to 15 and displays a pyramid, as shown in the following sample run:



5. Sum A Series

- Write a program to sum the following series:

$$\frac{1}{3}+\frac{3}{5}+\frac{5}{7}+\frac{7}{9}+\frac{9}{11}+\frac{11}{13}+\cdots+\frac{95}{97}+\frac{97}{99}$$

6. Prime numbers

- Write a program that displays the first fifty prime numbers in five lines, each containing ten numbers.

7. Perfect number

- A positive integer is called a perfect number if it is equal to the sum of all of its positive divisors, excluding itself. For example, 6 is the first perfect number because 6 = 3 + 2 + 1. The next is 28 = 14 + 7 + 4 + 2 + 1. There are four perfect numbers less than 10,000. Write a program to find all these four numbers.