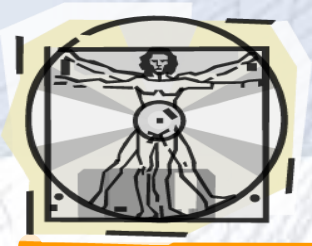
Computer Programming

Selections

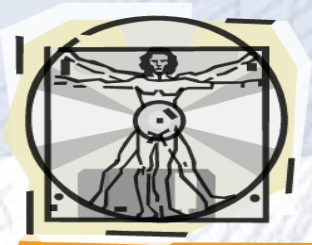
Özgür Koray ŞAHİNGÖZ
Prof.Dr.

Biruni University
Computer Engineering Department



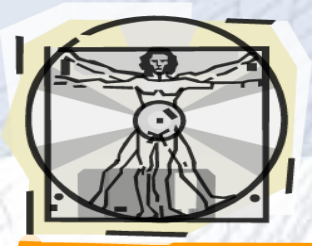
Motivations

If you assigned a negative value for radius in Listing 2.2, ComputeAreaWithConsoleInput.java, the program would print an invalid result. If the radius is negative, you don't want the program to compute the area. How can you deal with this situation?



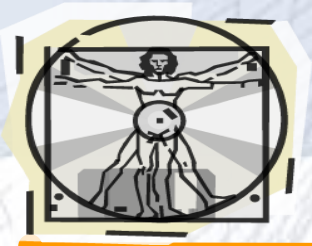
Objectives

- To declare boolean variables and write Boolean expressions using relational operators (§3.2).
- To implement selection control using one-way if statements (§3.3).
- To implement selection control using two-way if-else statements (§3.4).
- To implement selection control using nested if and multi-way if statements (§3.5).
- To avoid common errors and pitfalls in if statements (§3.6).
- To generate random numbers using the Math.random() method (§3.7).



Objectives

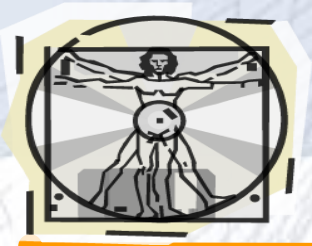
- To program using selection statements for a variety of examples (SubtractionQuiz, BMI, ComputeTax) (§§3.7–3.9).
- To combine conditions using logical operators (&&, ||, and !) (§3.10).
- To program using selection statements with combined conditions (LeapYear, Lottery) (§§3.11–3.12).
- To implement selection control using switch statements (§3.13).
- To write expressions using the conditional expression (§3.14).
- To examine the rules governing operator precedence and associativity (§3.15).
- To apply common techniques to debug errors (§3.16).



The `boolean` Type and Operators

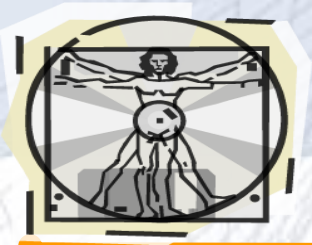
Often in a program you need to compare two values, such as whether i is greater than j. Java provides six comparison operators (also known as relational operators) that can be used to compare two values. The result of the comparison is a Boolean value: true or false.

```
boolean b = (1 > 2);
```



Relational Operators

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`



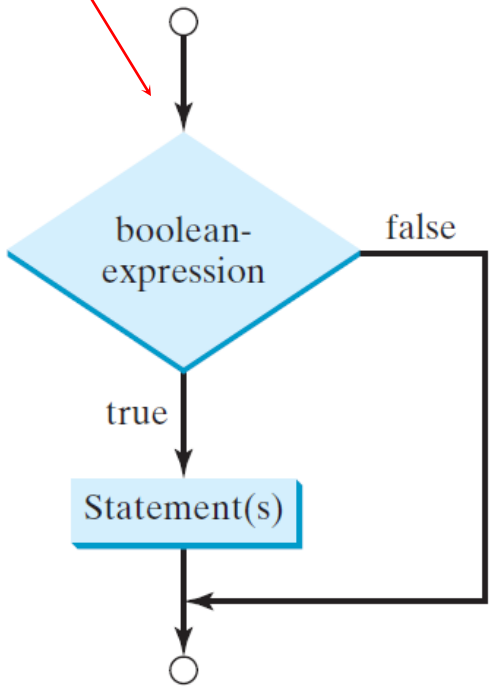
Problem: A Simple Math Learning Tool

- This example creates a program to let a first grader practice additions. The program randomly generates two single-digit integers number1 and number2 and displays a question such as “What is 7 + 9?” to the student. After the student types the answer, the program displays a message to indicate whether the answer is true or false.

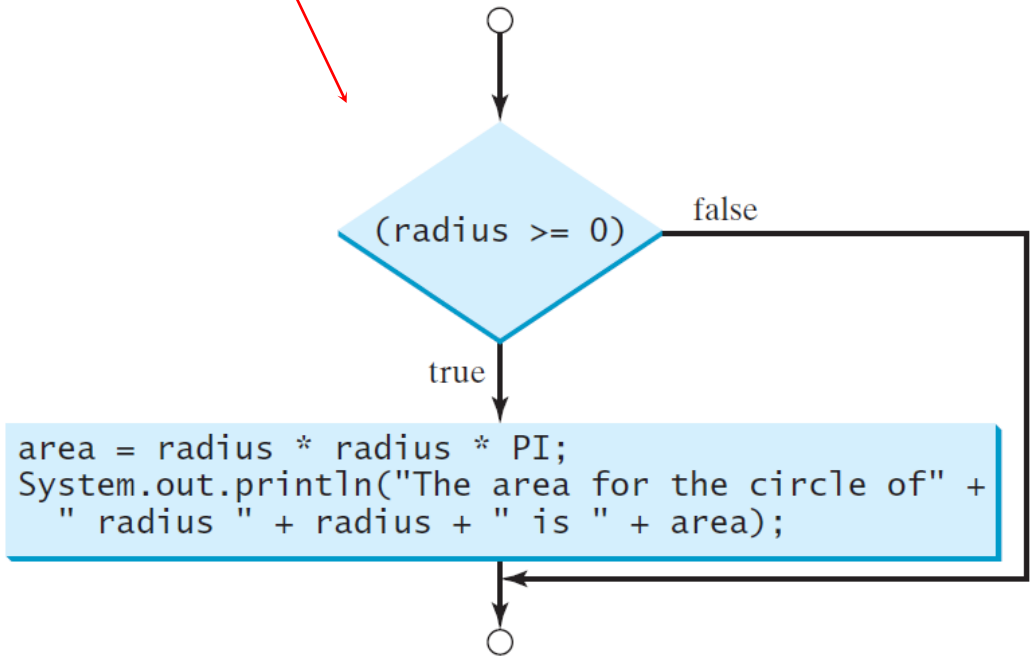
AdditionQuiz Run

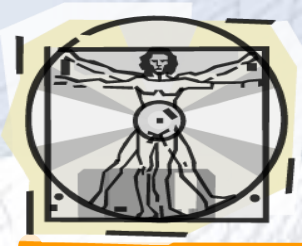
One-way `if` Statements

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




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





The if Statement

- Let's now look at the `if` statement in more detail
- The *if statement* has the following syntax:

`if` is a Java reserved word

The *`condition`* must be a boolean expression. It must evaluate to either true or false.

```
if ( condition )
   statement;
```

If the *`condition`* is true, the *`statement`* is executed.
If it is false, the *`statement`* is skipped.

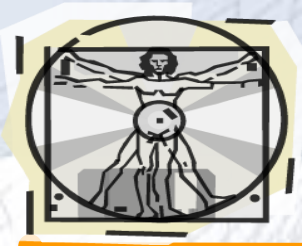
```
//********************************************************************
//  Age.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an if statement.
//********************************************************************

import java.util.Scanner;

public class Age
{
   //-----------------------------------------------------------------
   //  Reads the user's age and prints comments accordingly.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      final int MINOR = 21;

      Scanner scan = new Scanner(System.in);

      System.out.print("Enter your age: ");
      int age = scan.nextInt();

      System.out.println("You entered: " + age);

      if (age < MINOR)
         System.out.println("Youth is a wonderful thing. Enjoy.");

      System.out.println("Age is a state of mind.");
   }
}
```

Sample Run

```
Enter your age: 47
You entered: 47
Age is a state of mind.
```

Another Sample Run

```
Enter your age: 12
You entered: 12
Youth is a wonderful thing. Enjoy.
Age is a state of mind.
```



Note

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

(a) Wrong

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

(b) Correct

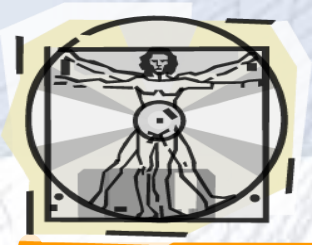
```
if (i > 0) {
   System.out.println("i is positive");
}
```

(a)

Equivalent

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

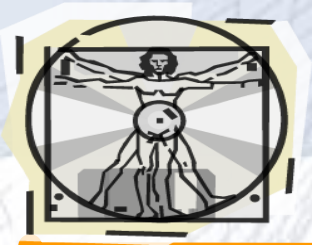
(b)



Simple if Demo

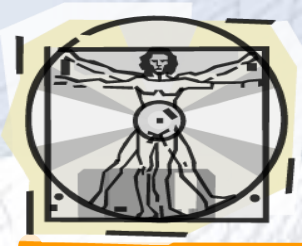
- Write a program that prompts the user to enter an integer.
  - If the number is a multiple of 5, print HiFive.
  - If the number is divisible by 2, print HiEven.

SimpleIfDemo Run

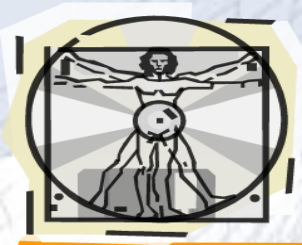


0(zero) means False

```
if(0)                 // false
if(1)                 // true
if(2)                 // true
if(0 == false)        // true
if(0 == true)         // false
if(1 == false)        // false
if(1 == true)         // true
if(2 == false)        // false
if(2 == true)         // false
```

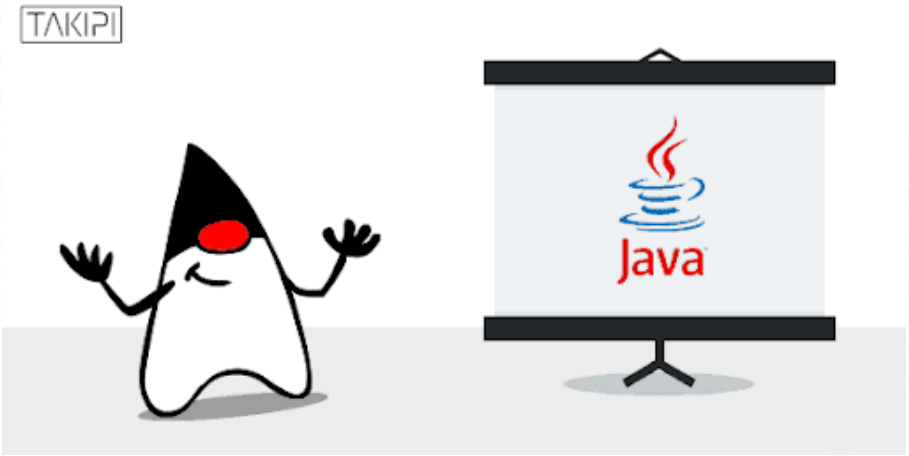


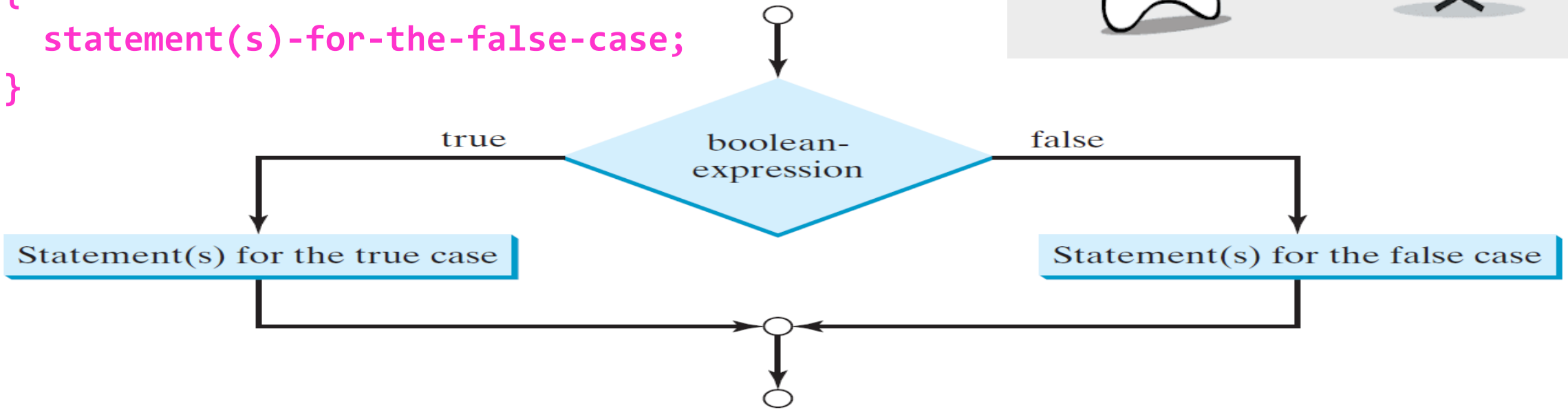
The Two-way `if` Statement

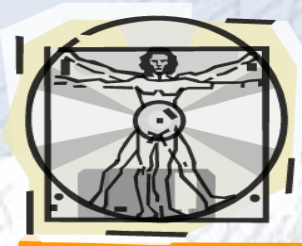


The Two-way `if` Statement

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

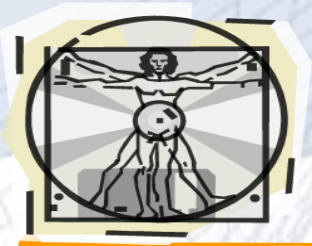






if-else Example

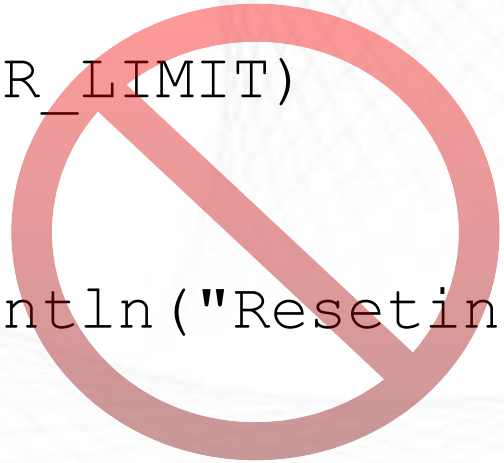
```
if (radius >= 0) {
  area = radius * radius * 3.14159;

  System.out.println("The area for the “
    + “circle of radius " + radius +
    " is " + area);
}
else {
  System.out.println("Negative input");
}
```



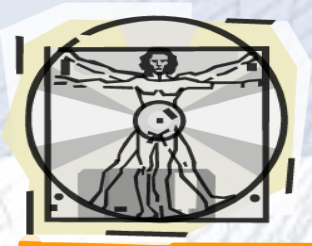
Indentation Revisited

- Remember that indentation is for the human reader, and is ignored by the compiler

```
if (depth >= UPPER_LIMIT)
   delta = 100;
else
   System.out.println("Reseting Delta");
   delta = 0;
```



- Despite what the indentation implies, `delta` will be set to 0 no matter what

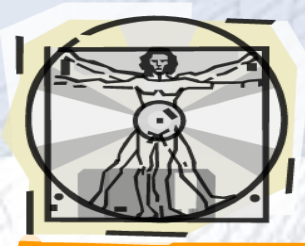


Block Statements

- Several statements can be grouped together into a *block statement* delimited by braces

- A block statement can be used wherever a statement is called for in the Java syntax rules

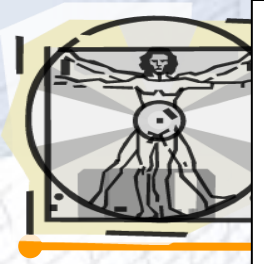
```
if (total > MAX)
{
   System.out.println("Error!!");
   errorCount++;
}
```



Block Statements

- The `if` clause, or the `else` clause, or both, could govern block statements

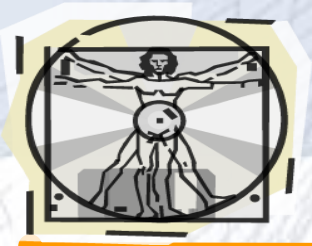
```
if (total > MAX)
{
   System.out.println("Error!!");
   errorCount++;
}
else
{
   System.out.println("Total: " + total);
   current = total*2;
}
```



```
import java.util.*;

public class Guessing {
  public static void main(Stri
      final int MAX = 10;
      int answer, guess;

      Scanner scan = new Scanner(System.in);
      Random generator = new Random();

      answer = generator.nextInt(MAX) + 1;

      System.out.print("I'm thinking of a number between 1 and "
                       + MAX + ". Guess what it is: ");

      guess = scan.nextInt();

      if (guess == answer)
         System.out.println("You got it! Good guessing!");
      else
      {
         System.out.println("That is not correct, sorry.");
         System.out.println("The number was " + answer);
      }
   }
}
```

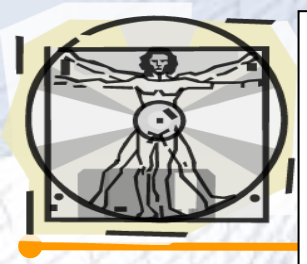
Sample Run

```
I'm thinking of a number between 1 and 10. Guess what it is: 6
That is not correct, sorry.
The number was 9
```



Nested if Statements

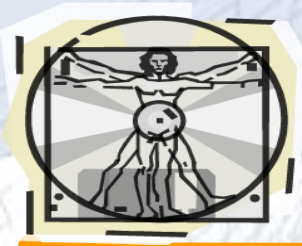
- The statement executed as a result of an `if` or `else` clause could be another `if` statement
- These are called *nested if statements*
- An `else` clause is matched to the last unmatched `if` (no matter what the indentation implies)
- Braces can be used to specify the `if` statement to which an `else` clause belongs



```
import java.util.Scanner;

public class MinOfThree {

  public static void main(String[] args)    {
      int num1, num2, num3, min = 0;

      Scanner scan = new Scanner(System.in);

      System.out.println("Enter three integers: ");
      num1 = scan.nextInt();
      num2 = scan.nextInt();
      num3 = scan.nextInt();

 if (num1 < num2)
         if (num1 < num3)
            min = num1;
         else
            min = num3;
      else
         if (num2 < num3)
            min = num2;
         else
            min = num3;

      System.out.println("Minimum value: " + min);
   }
}
```

Sample Run

```
Enter three integers:
84   69   90
Minimum value: 69
```



Multiple Alternative if Statements

```
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");
```

(a)

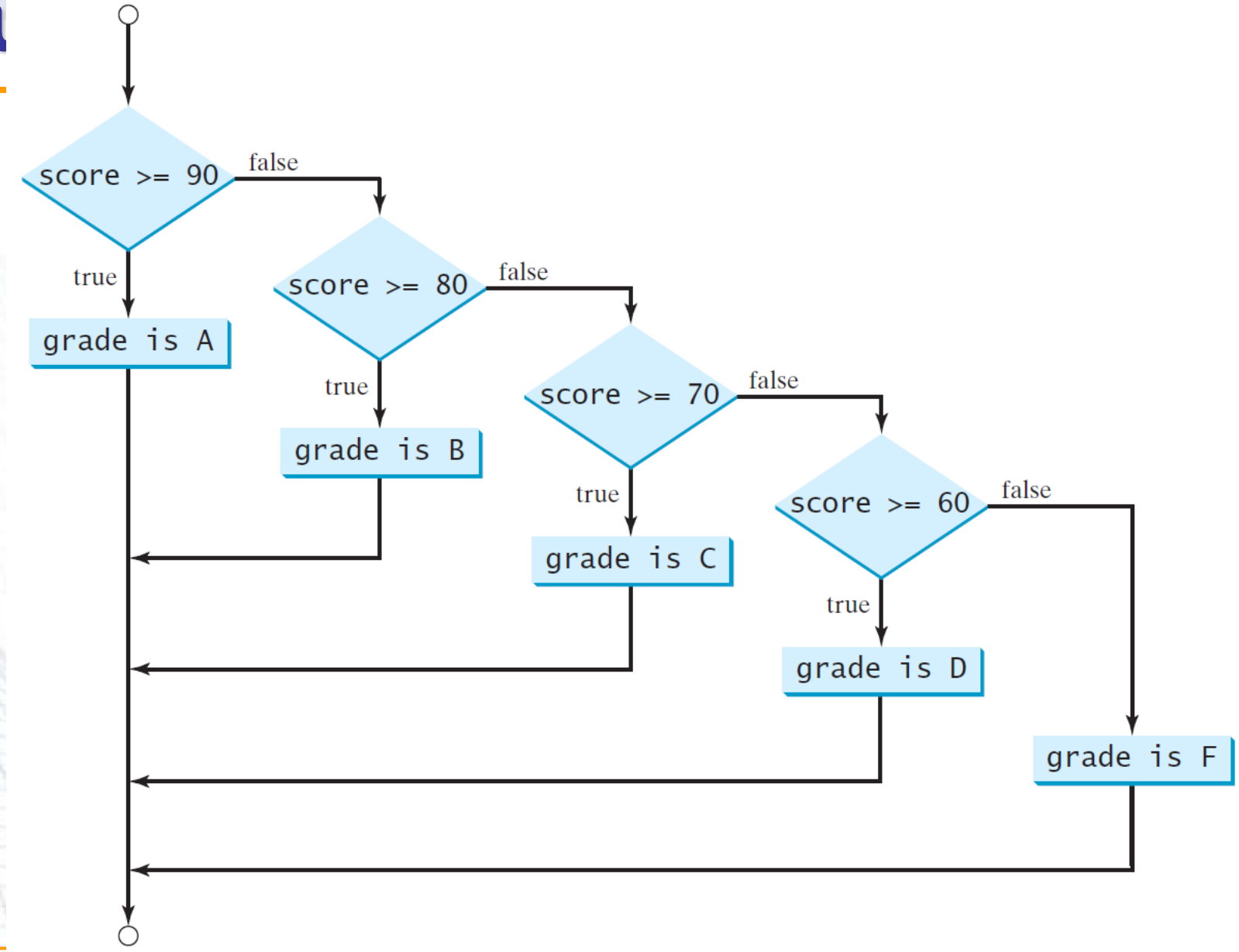
Equivalent

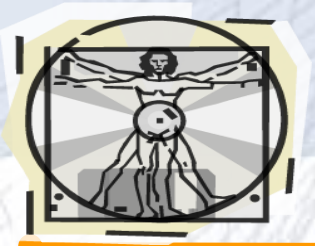
This is better

```
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");
```

(b)

Mu


score >= 90
false
true
grade is A
score >= 80
false
true
grade is B
score >= 70
false
true
grade is C
score >= 60
false
true
grade is D
grade is F

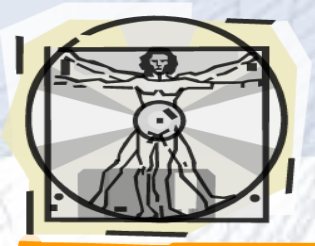


Trace if-else statement

Suppose score is 70.0

The condition is false

```
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");
```

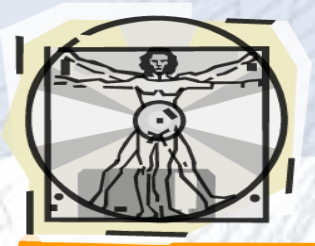


Trace if-else statement

Suppose score is 70.0

The condition is false

```
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");
```

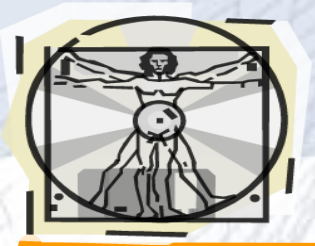


Trace if-else statement

Suppose score is 70.0

The condition is true

```
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");
```

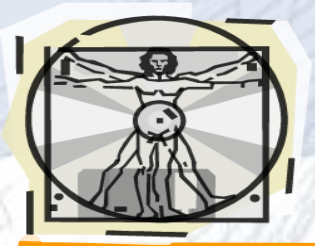


Trace if-else statement

Suppose score is 70.0

grade is C

```
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");
```

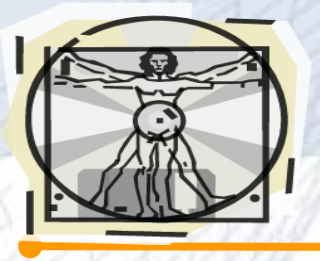


Trace if-else statement

Suppose score is 70.0

Exit the if statement

```
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");
```



Note

The else clause matches the most recent if clause in the same block.

```
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
else
    System.out.println("B");
```

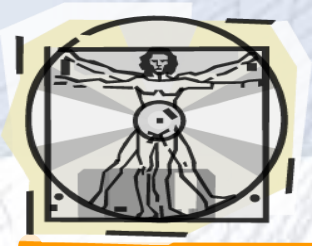
(a)

Equivalent

This is better with correct indentation

```
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
  else
    System.out.println("B");
```

(b)

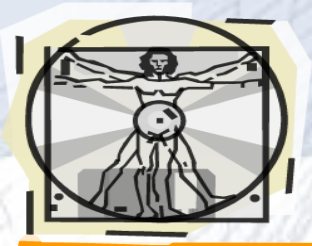


Note, cont.

Nothing is printed from the preceding statement. To force the else clause to match the first if clause, you must add a pair of braces:

```
int i = 1;
int j = 2;
int k = 3;
if (i > j) {
  if (i > k)
    System.out.println("A");
}
else
  System.out.println("B");
```

This statement prints B.



Common Errors

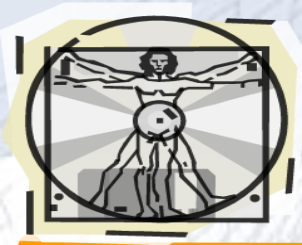
Adding a semicolon at the end of an if clause is a common mistake.

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

Wrong

This mistake is hard to find, because it is not a compilation error or a runtime error, it is a logic error.

This error often occurs when you use the next-line block style.



TIP

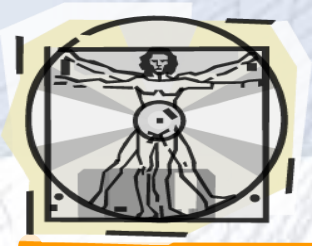
```
if (number % 2 == 0)
   even = true;
else
   even = false;
```

(a)

Equivalent

```
boolean even
   = number % 2 == 0;
```

(b)



CAUTION

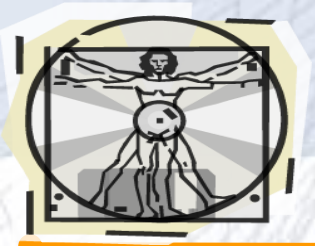
```
if (even == true)
  System.out.println(
    "It is even.");
```

(a)

Equivalent

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

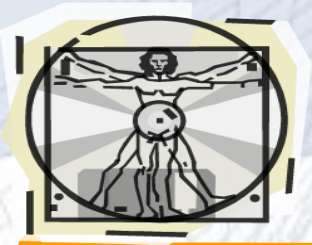
(b)



Problem: An Improved Math Learning Tool

This example creates a program to teach a first grade child how to learn subtractions. The program randomly generates two single-digit integers number1 and number2 with number1 >= number2 and displays a question such as "What is 9 – 2?" to the student. After the student types the answer, the program displays whether the answer is correct.

SubtractionQuiz **Run**

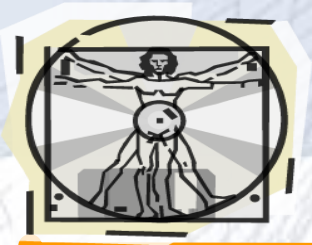


Problem: Body Mass Index

Body Mass Index (BMI) is a measure of health on weight. It can be calculated by taking your weight in kilograms and dividing by the square of your height in meters. The interpretation of BMI for people 16 years or older is as follows:

BMI	Interpretation
BMI < 18.5	Underweight
18.5 <= BMI < 25.0	Normal
25.0 <= BMI < 30.0	Overweight
30.0 <= BMI	Obese

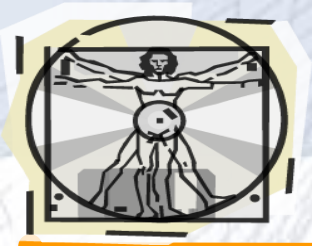
ComputeAndInterpretBMI **Run**



Problem: Computing Taxes

The US federal personal income tax is calculated based on the filing status and taxable income. There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household. The tax rates for 2009 are shown below.

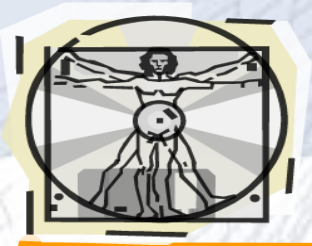
Marginal Tax Rate	*Single*	*Married Filing Jointly or Qualifying Widow(er)*	*Married Filing Separately*	*Head of Household*
10%	$0 – $8,350	$0 – $16,700	$0 – $8,350	$0 – $11,950
15%	$8,351 – $33,950	$16,701 – $67,900	$8,351 – $33,950	$11,951 – $45,500
25%	$33,951 – $82,250	$67,901 – $137,050	$33,951 – $68,525	$45,501 – $117,450
28%	$82,251 – $171,550	$137,051 – $208,850	$68,526 – $104,425	$117,451 – $190,200
33%	$171,551 – $372,950	$208,851 – $372,950	$104,426 – $186,475	$190,201 – $372,950
35%	$372,951+	$372,951+	$186,476+	$372,951+



Problem: Computing Taxes, cont.

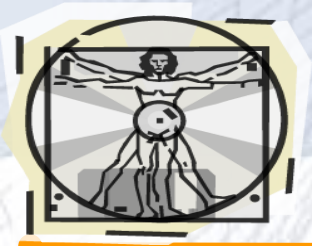
```
if (status == 0) {
  // Compute tax for single filers
}
else if (status == 1) {
  // Compute tax for married file jointly
  // or qualifying widow(er)
}
else if (status == 2) {
  // Compute tax for married file separately
}
else if (status == 3) {
  // Compute tax for head of household
}
else {
  // Display wrong status
}
```

ComputeTax Run



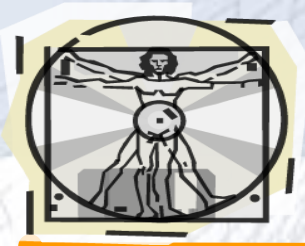
Logical Operators

Operator	Name	Description
!	not	logical negation
&&	and	logical conjunction
||	or	logical disjunction
^	exclusive or	logical exclusion



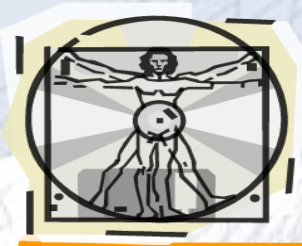
Truth Table for Operator !

p	!p	Example (assume age = 24, weight = 140)
true	false	!(age > 18) is false, because (age > 18) is true.
false	true	!(weight == 150) is true, because (weight == 150) is false.



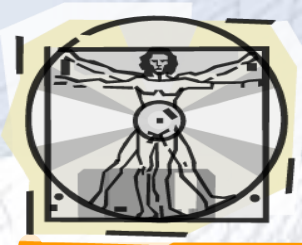
Truth Table for Operator &&

p_1	p_2	p_1 && p_2	Example (assume age = 24, weight = 140)
false	false	false	(age <= 18) && (weight < 140) is false, because both conditions are both false.
false	true	false	
true	false	false	(age > 18) && (weight > 140) is false, because (weight > 140) is false.
true	true	true	(age > 18) && (weight >= 140) is true, because both (age > 18) and (weight >= 140) are true.



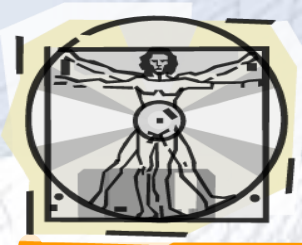
Truth Table for Operator ||

p_1	p_2	p_1 || p_2	Example (assume age = 24, weihgt = 140)
false	false	false	
false	true	true	(age > 34) || (weight <= 140) is true, because (age > 34) is false, but (weight <= 140) is true.
true	false	true	(age > 14) || (weight >= 150) is false, because (age > 14) is true.
true	true	true	



Truth Table for Operator ^

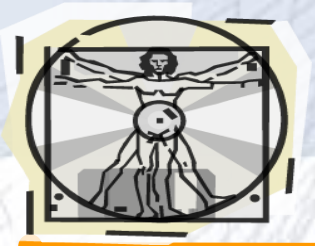
p_1	p_2	p_1 ^ p_2	Example (assume age = 24, weight = 140)
false	false	false	(age > 34) ^ (weight > 140) is true, because (age > 34) is false and (weight > 140) is false.
false	true	true	(age > 34) ^ (weight >= 140) is true, because (age > 34) is false but (weight >= 140) is true.
true	false	true	(age > 14) ^ (weight > 140) is true, because (age > 14) is true and (weight > 140) is false.
true	true	false	



Examples

Here is a program that checks whether a number is divisible by 2 and 3, whether a number is divisible by 2 or 3, and whether a number is divisible by 2 or 3 but not both:

TestBooleanOperators **Run**

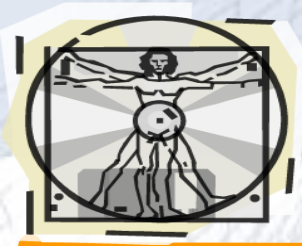


Examples

```
System.out.println("Is " + number + " divisible by 2 and 3? " +
  ((number % 2 == 0) && (number % 3 == 0)));

System.out.println("Is " + number + " divisible by 2 or 3? " +
  ((number % 2 == 0) || (number % 3 == 0)));

System.out.println("Is " + number +
  " divisible by 2 or 3, but not both? " +
  ((number % 2 == 0) ^ (number % 3 == 0)));
```

TestBooleanOperators

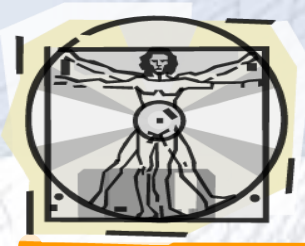
Run



Companion Website

The & and | Operators

Supplement III.B, “The & and | Operators”



Companion Website

The & and | Operators

If x is 1, what is x after this expression?

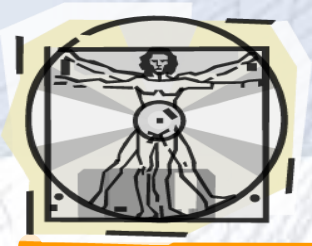
```
(x > 1) & (x++ < 10)
```

If x is 1, what is x after this expression?

```
(1 > x) && ( 1 > x++)
```

How about (1 == x) | (10 > x++)?

```
(1 == x) || (10 > x++)?
```



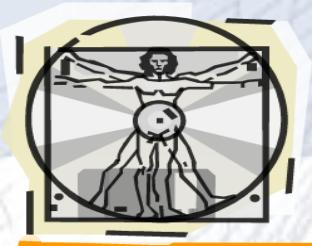
Problem: Determining Leap Year?

This program first prompts the user to enter a year as an int value and checks if it is a leap year.

A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.

```
(year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)
```

LeapYear Run

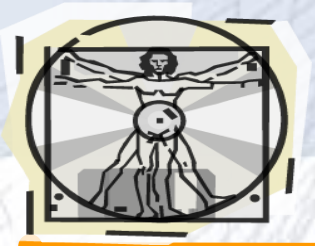


Problem: Lottery

Write a program that randomly generates a lottery of a two-digit number, prompts the user to enter a two-digit number, and determines whether the user wins according to the following rule:

- **If the user input matches the lottery in exact order, the award is $10,000.**
- **If the user input matches the lottery, the award is $3,000.**
- **If one digit in the user input matches a digit in the lottery, the award is $1,000.**

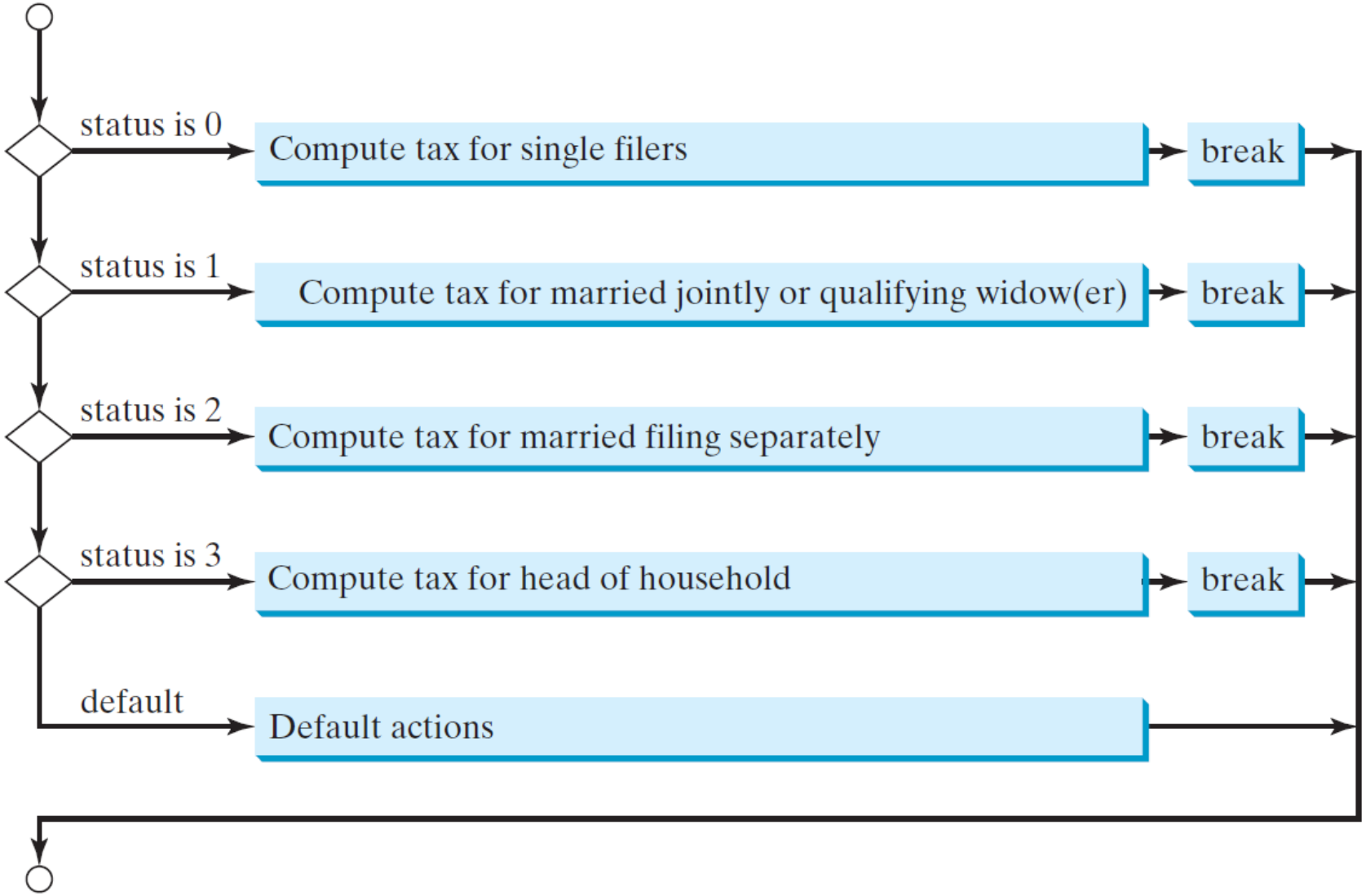
Lottery **Run**

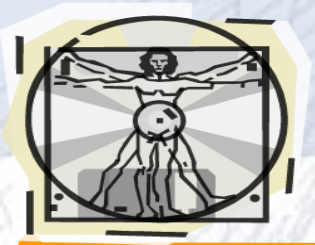


`switch` Statements

```
switch (status) {
  case 0:  compute taxes for single filers;
           break;
  case 1:  compute taxes for married file jointly;
           break;
  case 2:  compute taxes for married file separately;
           break;
  case 3:  compute taxes for head of household;
           break;
  default: System.out.println("Errors: invalid status");
           System.exit(1);
}
```

`switch` Statement Flow Chart



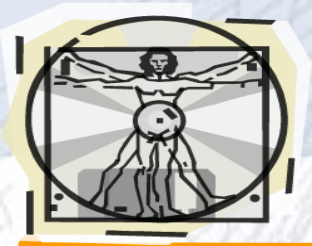


switch Statement Rules

The switch-expression must yield a value of char, byte, short, or int type and must always be enclosed in parentheses.

The value1, ..., and valueN must have the same data type as the value of the switch-expression. The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression. Note that value1, ..., and valueN are constant expressions, meaning that they cannot contain variables in the expression, such as 1 + x.

```
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;
}
```



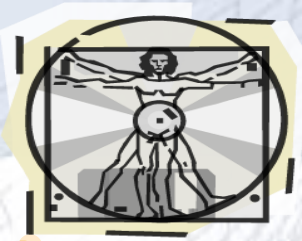
`switch` Statement Rules

The keyword break is optional, but it should be used at the end of each case in order to terminate the remainder of the switch statement. If the break statement is not present, the next case statement will be executed.

```
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;
}
```

The default case, which is optional, can be used to perform actions when none of the specified cases matches the switch-expression.

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.

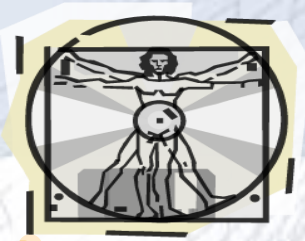


animation

Trace switch statement

Suppose day is 2:

```
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");
}
```

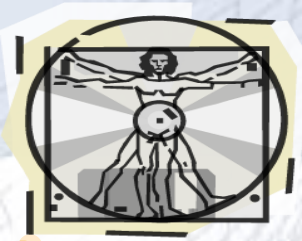


animation

Trace switch statement

Match case 2

```
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");
}
```

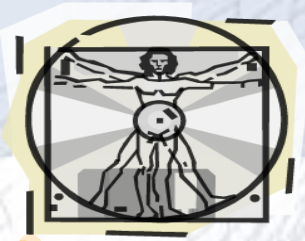


animation

Trace switch statement

Fall through case 3

```
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");
}
```

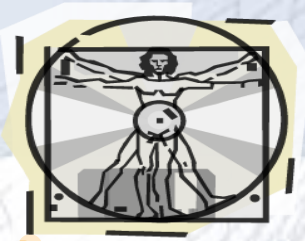


animation

Trace switch statement

Fall through case 4

```
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");
}
```

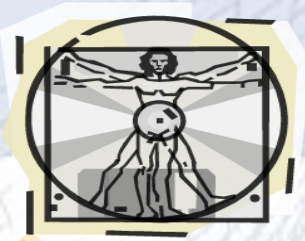


animation

Trace switch statement

Fall through case 5

```
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");
}
```

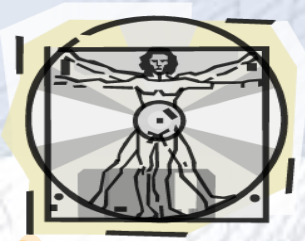


animation

Trace switch statement

Encounter break

```
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");
}
```



animation

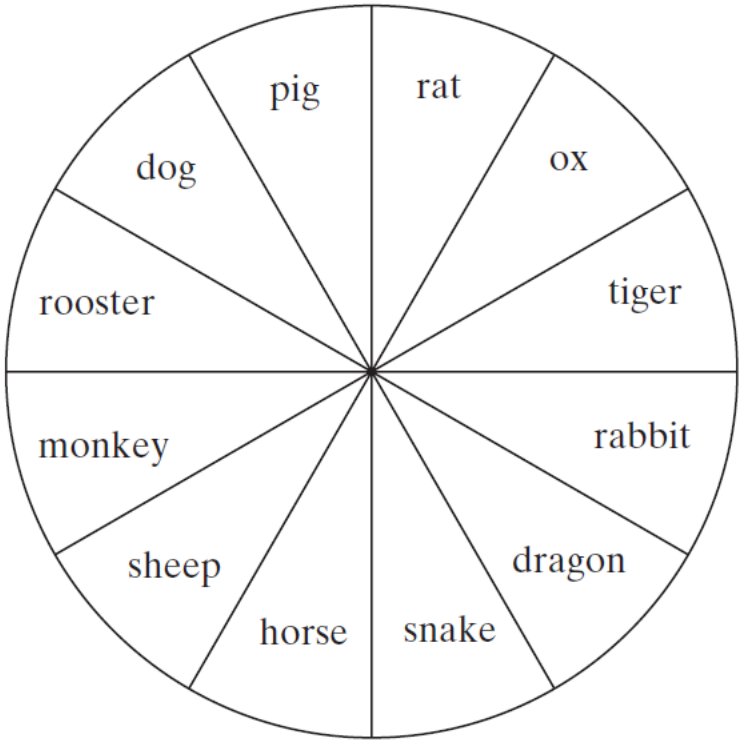
Trace switch statement

Exit the statement

```
swit   (day) {
  ca    :
  ca   2:
  c    3:
  c   e 4:
  c   se 5: System.out.println("Weekday"); break;
  c  ase 0:
  c ase 6: System.out.println("Weekend");
}
```

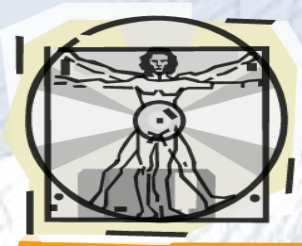
Problem: Chinese Zodiac

Write 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

ChineseZodiac **Run**



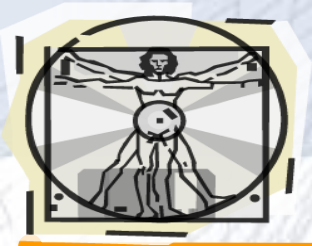
Conditional Operators

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

is equivalent to

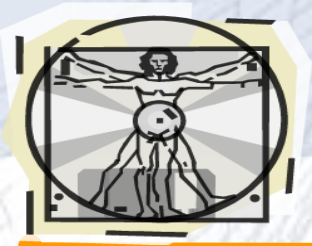
```
    y = (x > 0) ? 1 : -1;
```

(boolean-expression) ? expression1 : expression2



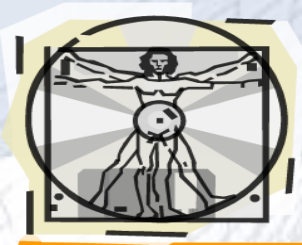
Conditional Operator

```
if (num % 2 == 0)
   System.out.println(num + “is even”);
else
   System.out.println(num + “is odd”);


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



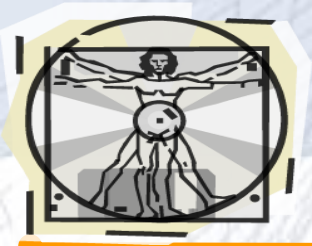
Conditional Operator, cont.

```
boolean-expression ? exp1 : exp2
```



Operator Precedence

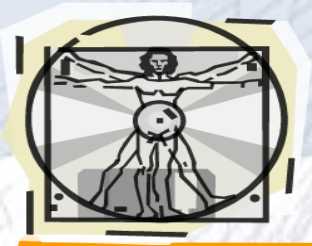
- var++, var--
- +, - (Unary plus and minus), ++var,--var
- (type) Casting
- ! (Not)
- *, /, % (Multiplication, division, and remainder)
- +, - (Binary addition and subtraction)
- <, <=, >, >= (Relational operators)
- ==, !=; (Equality)
- ^ (Exclusive OR)
- && (Conditional AND) Short-circuit AND
- || (Conditional OR) Short-circuit OR
- =, +=, -=, *=, /=, %= (Assignment operator)



Operator Precedence and Associativity

The expression in the parentheses is evaluated first. (Parentheses can be nested, in which case the expression in the inner parentheses is executed first.) When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule.

If operators with the same precedence are next to each other, their associativity determines the order of evaluation. All binary operators except assignment operators are left-associative.



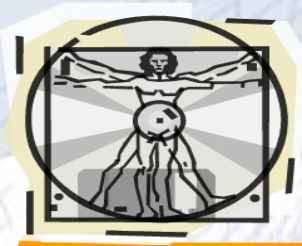
Operator Associativity

When two operators with the same precedence are evaluated, the *associativity* of the operators determines the order of evaluation. All binary operators except assignment operators are *left-associative*.

a – b + c – d is equivalent to ((a – b) + c) – d

Assignment operators are *right-associative*. Therefore, the expression

a = b += c = 5 is equivalent to a = (b += (c = 5))



Example

Applying the operator precedence and associativity rule, the expression 3 + 4 * 4 > 5 * (4 + 3) - 1 is evaluated as follows:

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

(1) inside parentheses first

```
3 + 4 * 4 > 5 * 7 - 1
```

(2) multiplication

```
3 + 16 > 5 * 7 - 1
```

(3) multiplication

```
3 + 16 > 35 - 1
```

(4) addition

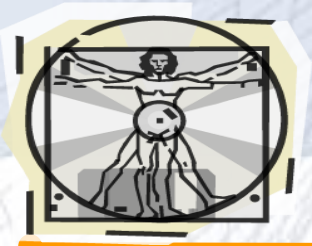
```
19 > 35 - 1
```

(5) subtraction

```
19 > 34
```

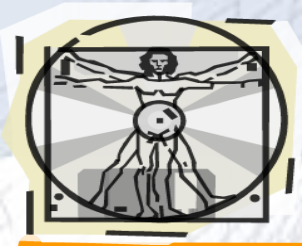
(6) greater than

```
false
```



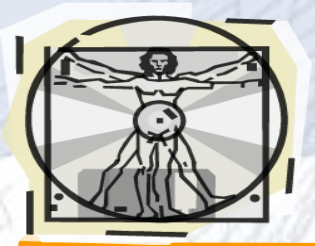
Operand Evaluation Order

Supplement III.A, “Advanced discussions on how an expression is evaluated in the JVM.”



Debugging

Logic errors are called *bugs*. The process of finding and correcting errors is called debugging. A common approach to debugging is to use a combination of methods to narrow down to the part of the program where the bug is located. You can hand-trace the program (i.e., catch errors by reading the program), or you can insert print statements in order to show the values of the variables or the execution flow of the program. This approach might work for a short, simple program. But for a large, complex program, the most effective approach for debugging is to use a debugger utility.



Debugger

Debugger is a program that facilitates debugging. You can use a debugger to

- Execute a single statement at a time.
- Trace into or stepping over a method.
- Set breakpoints.
- Display variables.
- Display call stack.
- Modify variables.