Chapter 3
Selections

CSE215 Programming Language II
Sec – 2

Motivations

- **LISTING 2.1** ComputeArea.java

```
public class ComputeArea {
  public static void main(String[] args) {
    double radius; // Declare radius
    double area; // Declare area

    // Assign a radius
    radius = 20; // radius is now 20

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```

Motivations

- If you assigned a negative value for radius in Listing 2.1, ComputeArea.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?

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);
```

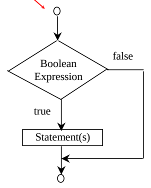
Comparison Operators

Operator	*Name*
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

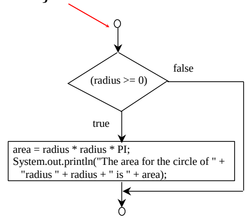
One-way `if` Statements

```
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);
}
```



(A)


(radius >= 0)
false
true
area = radius * radius * PI;
System.out.println("The area for the circle of " + "radius " + radius + " is " + area);
(B)

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

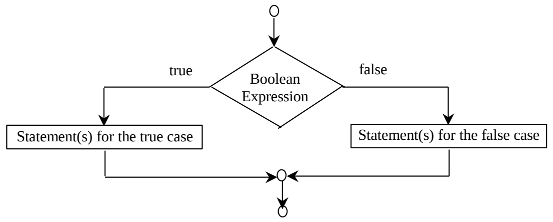
Write a program that prompts the user to enter an integer. If the number is a multiple of 5, print HiFive. If the number is divisible by 2, print HiEven.

```
import java.util.Scanner;

public class SimpleIfDemo {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.println("Enter an integer: ");
    int number = input.nextInt();
    if (number % 5 == 0)
       System.out.println("HiFive");
    if (number % 2 == 0)
       System.out.println("HiEven");
  }
}
```

The Two-way `if` Statement

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


true
false
Boolean Expression
Statement(s) for the true case
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");
}
```

Multiple Alternative if Statements

```
if (score >= 90.0)
  grade = 'A';
else
  if (score >= 80.0)
    grade = 'B';
  else
    if (score >= 70.0)
      grade = 'C';
    else
      if (score >= 60.0)
        grade = 'D';
      else
        grade = 'F';
```

Equivalent

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

Trace if-else statement

Suppose score is 70.0

The condition is false

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

Trace if-else statement

Suppose score is 70.0

The condition is false

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

Trace if-else statement

Suppose score is 70.0

The condition is true

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

Trace if-else statement

Suppose score is 70.0

grade is C

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

Trace if-else statement

Suppose score is 70.0

Exit the if statement

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

Note

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

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

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

(a)

Equivalent

```
int i = 1;
int j = 2;
int 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)

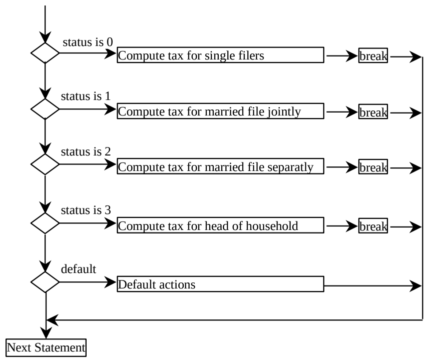
Logical Operators

Operator	*Name*
`!`	not
`&&`	and
`||`	or
`^`	exclusive or

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

`switch` Statement Flow Chart


status is 0
Compute tax for single filers
break
status is 1
Compute tax for married file jointly
break
status is 2
Compute tax for married file separatly
break
status is 3
Compute tax for head of household
break
default
Default actions
Next Statement

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.

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

The case statements are executed in sequential order, but the order of the cases (including the default case) does not matter. However, it is good programming style to follow the logical sequence of the cases and place the default case at the end.

Trace switch statement

Suppose ch is 'a':

```
switch  (ch)  {
   case  'a':  System.out.println(ch);
   case  'b':  System.out.println(ch);
   case  'c':  System.out.println(ch);
}
```

Trace switch statement

ch is 'a':

```
switch  (ch)  {
   case  'a':  System.out.println(ch);
   case  'b':  System.out.println(ch);
   case  'c':  System.out.println(ch);
}
```

Trace switch statement

Execute this line

```
switch (ch) {
  case 'a': System.out.println(ch);
  case 'b': System.out.println(ch);
  case 'c': System.out.println(ch);
}
```

Trace switch statement

Execute this line

```
switch (ch) {
  case 'a': System.out.println(ch);
  case 'b': System.out.println(ch);
  case 'c': System.out.println(ch);
}
```

Trace switch statement

Execute this line

```
switch (ch) {
  case 'a': System.out.println(ch);
  case 'b': System.out.println(ch);
  case 'c': System.out.println(ch);
}
```

Trace switch statement

Execute next statement

```
switch (ch) {
  case 'a': System.out.println(ch);
  case 'b': System.out.println(ch);
  case 'c': System.out.println(ch);
}

Next statement;
```

Trace switch statement

Suppose ch is 'a':

```
switch (ch) {
  case 'a': System.out.println(ch);
            break;
  case 'b': System.out.println(ch);
            break;
  case 'c': System.out.println(ch);
}
```

Trace switch statement

ch is 'a':

```
switch (ch) {
  case 'a': System.out.println(ch);
            break;
  case 'b': System.out.println(ch);
            break;
  case 'c': System.out.println(ch);
}
```

Trace switch statement

Execute this line

```
switch (ch) {
  case 'a': System.out.println(ch);
            break;
  case 'b': System.out.println(ch);
            break;
  case 'c': System.out.println(ch);
}
```

Trace switch statement

Execute this line

```
switch (ch) {
  case 'a': System.out.println(ch);
            break;
  case 'b': System.out.println(ch);
            break;
  case 'c': System.out.println(ch);
}
```

Trace switch statement

Execute next statement

```
switch (ch) {
  case 'a': System.out.println(ch);
            break;
  case 'b': System.out.println(ch);
            break;
  case 'c': System.out.println(ch);
}
```

Next statement;

Conditional Operator

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

is equivalent to

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

(boolean-expression) ? expression1 : expression2

Conditional Operator, cont.

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

Conditional Operator

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

is equivalent to

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

Formatting Output

Use the printf statement.

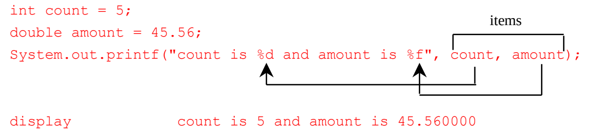
```
System.out.printf(format, items);
```

Where *format* is a string that may consist of substrings and format specifiers. A format specifier specifies how an item should be displayed.

An *item* may be a numeric value, character, boolean value, or a string. Each specifier begins with a percent sign.

Frequently-Used Specifiers

Specifier	Output	Example
%b	a boolean value	true or false
%c	a character	'a'
%d	a decimal integer	200
%f	a floating-point number	45.460000
%e	a number in standard scientific notation	4.556000e+01
%s	a string	"Java is cool"



```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);

display          count is 5 and amount is 45.560000
```

Operator Precedence

- `var++, var--`
- `+, - (Unary plus and minus), ++var,--var`
- `(type) Casting`
- `! (Not)`
- `*, /, % (Multiplication, division, and remainder)`
- `+, - (Binary addition and subtraction)`
- `<, <=, >, >= (Comparison)`
- `==, !=; (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

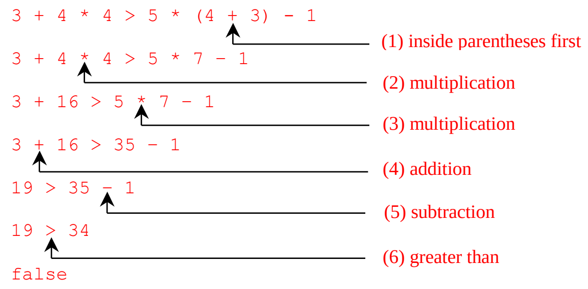
- When two operators with the same precedence are evaluated, the *associativity* of the operators determines the order of evaluation.
- All binary operators except assignment operators are *left-associative*.

  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