Selections

CSE 114: Introduction to Object-Oriented Programming

Paul Fodor

Stony Brook University

http://www.cs.stonybrook.edu/~cse114

Contents

- Motivation for selection statements
- The Comparison Operators and the **`boolean`** Type
- One-way **`if`** Statements
- Two-way **`if`** Statement
  - Multiple Alternative if Statements
  - if … else matching
  - Common Errors
- Complex conditions with Logical Operators
  - Example: Determining Leap Year
  - The unconditional & and | Operators
- **`switch`** Statements
- The Conditional Operator
- Operator Precedence and Associativity

- Computing the Area of a Circle:

```
import java.util.Scanner;
public class ComputeArea {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter a radius: ");
    double radius = input.nextDouble();
    double area = radius * radius * 3.14159;
    // Display results
    System.out.println("The area for the circle"
      + " of radius " + radius + " is " + area);
  }
}
```

What if the user enters a negative value (i.e., an invalid value)?

The area for a circle with a negative radius does not make sense.

Motivation

If the user enterred a negative value for radius in ComputeArea.java, then you don't want the program to compute the area, but to inform the user that their input was incorrect.

- Computing the Area of a Circle:

```
import java.util.Scanner;
public class ComputeAreaNew {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter a radius: ");
    double radius = input.nextDouble();
    if(radius >= 0){
      double area = radius * radius * 3.14159;
      System.out.println("The area for the circle"
        + " of radius " + radius + " is " + area);
    } else //{
      System.out.println("The radius is negative."
         + " The area cannot be computed.");
    // }
  }
}
```

The Comparison Operators and `boolean` Type

- Often in a programs you need to compare values:

  e.g., if `x` is greater than `y` then ...

- Java provides six ***comparison operators*** (also called relational operators) to compare two values: `<`, `<=`, `>`, `>=`, `==` and `!=`
  - The result of the comparison is a Boolean value: `true` or `false`. For example,

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

`b` is `false` after the statement.

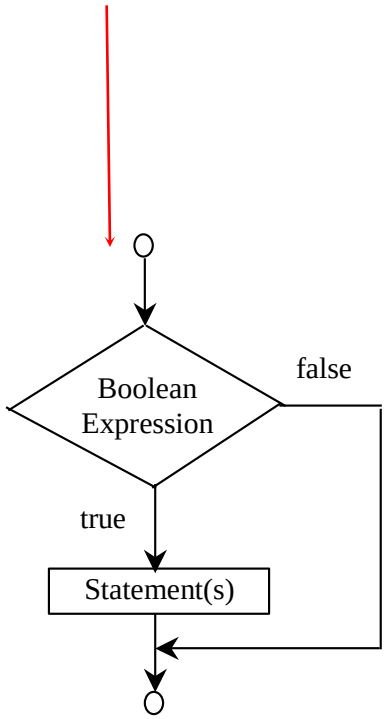
Comparison Operators

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

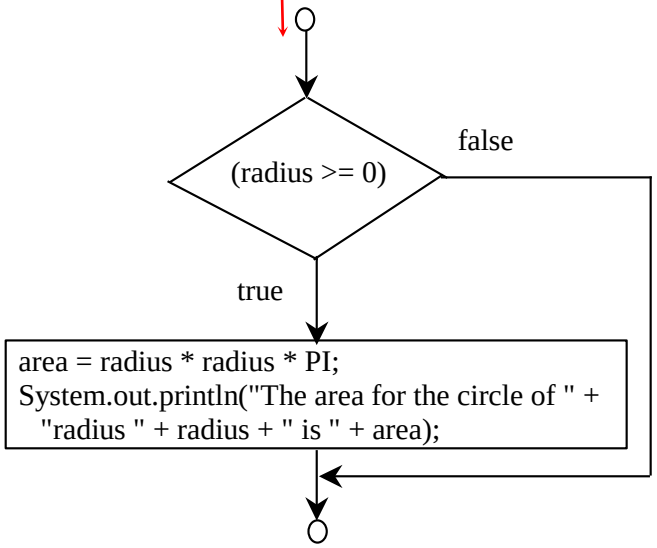
One-way `if` Statements

```
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)



(B)

One-way `if` Statements

Condition containment is necessary!

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

(a) Wrong

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

(b) Correct

Block containment is not necessary for a **single** statement!

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

(a)

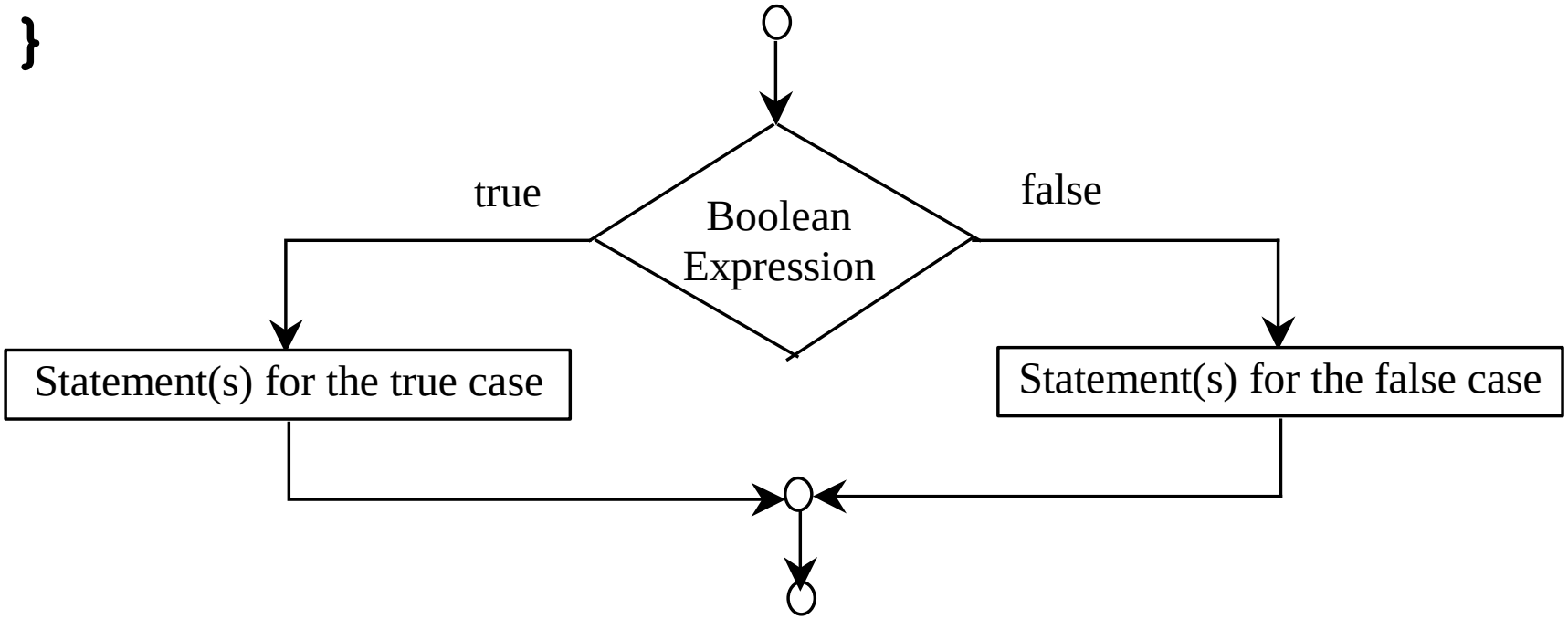
Equivalent

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

(b)

Two-way `if` Statement

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



Two-way `if` Example

```
if (radius >= 0) {
  double area = radius * radius * 3.1415;
  System.out.println("The area for the"
    + " circle of radius " + radius
    + " is " + area);
} else
  System.out.println("Negative input");
```

Multiple Alternative if Statements

Indentation in Java is not required, but a good programming style.

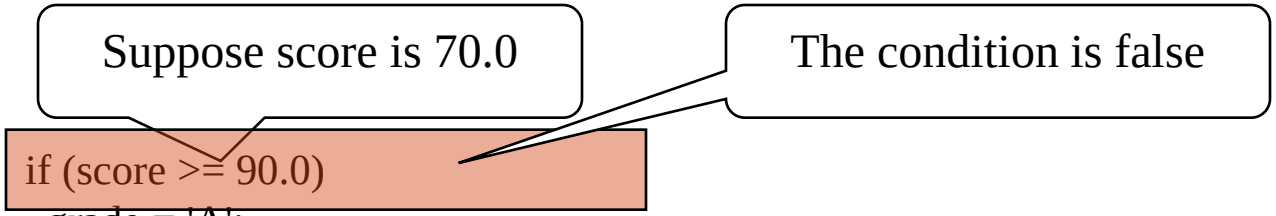
```
if (score >= 90.0)
  grade = 'A';
else
  if (score >= 80.0)
    grade = 'B';
  else
    if (score >= 70.0)
      grade = 'C';
    else
      if (score >= 60.0)
        grade = 'D';
      else
        grade = 'F';
```

Equivalent

Indentation exception for cascading if-else-if statements:

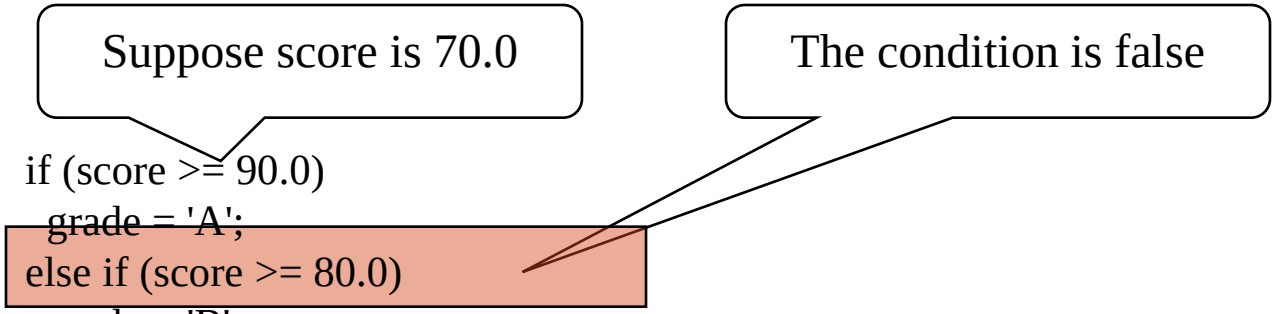
```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

Trace if-else statement



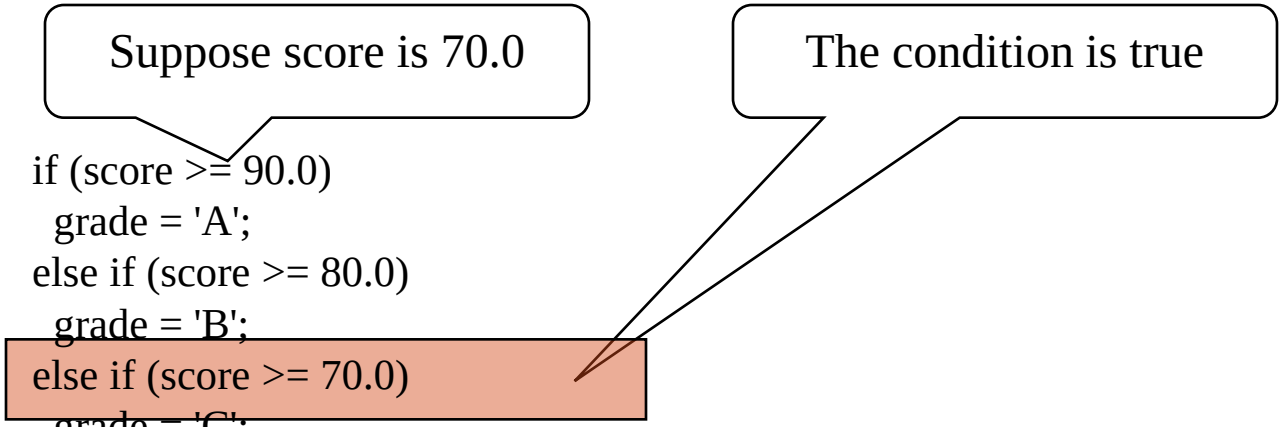

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

Trace if-else statement

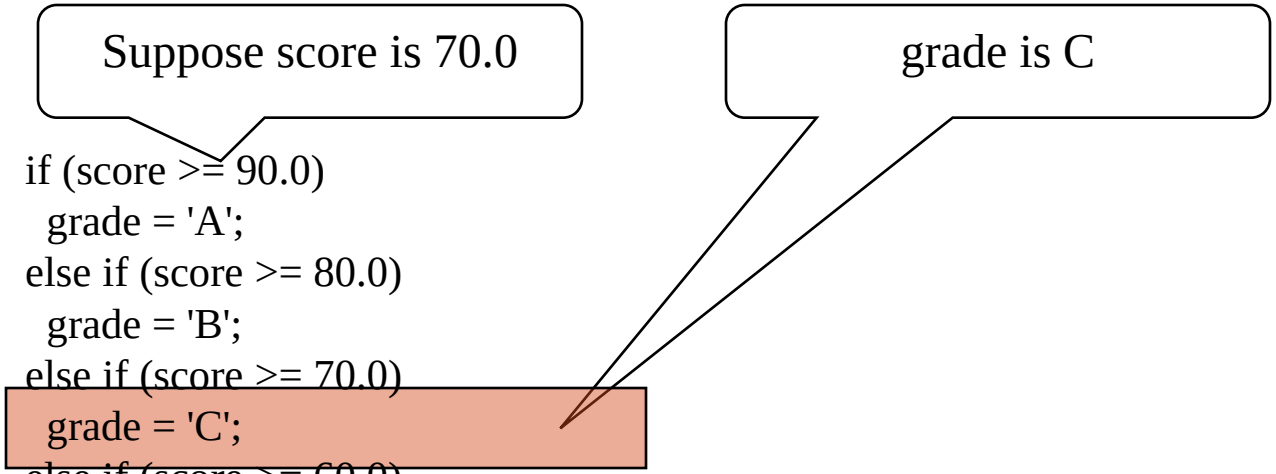



```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

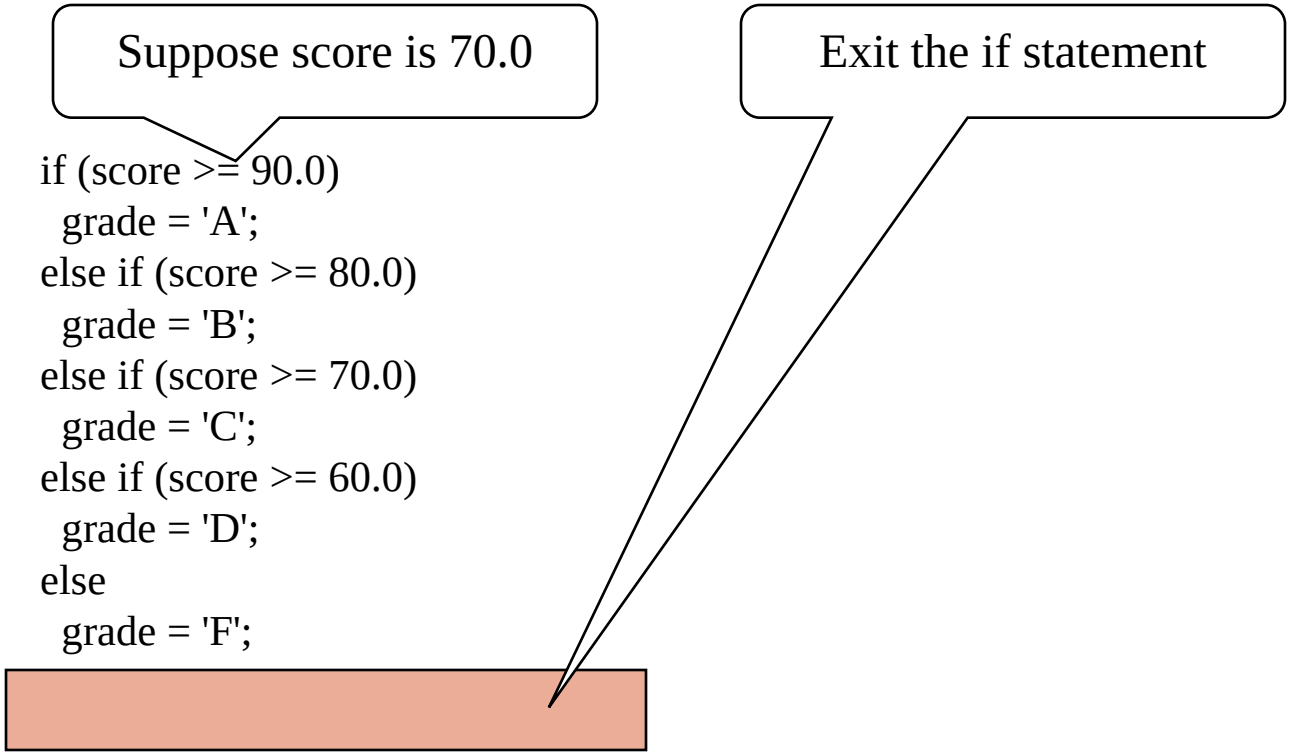
Trace if-else statement



Trace if-else statement



Trace if-else statement



if … else

Inconsistent indentation can get us confused, so the rule is that 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)

Wrong indentation

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)

Correct indentation

This does not print anything!

if … else

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 code prints B.

Common Error

- Adding a semicolon at the end of an if clause is a common mistake (often occurs when you use the next-line block style):

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

Wrong

  - It is not a compilation error and it is not a runtime error
  - It is a logic error because ";" is a statement (the no-operation/no-op statement)

What's wrong here?

```
System.out.print("Enter your total cholesterol level: ");
int totalCholesterol= input.nextInt();

if (totalCholesterol>= 200)
  System.out.println("Your cholesterol is too high.");
  System.out.println("You need to lower that.");
else
  System.out.println("Good, eat away!");
```

What's wrong here?

```
System.out.print("Enter your total cholesterol level: ");
int totalCholesterol= input.nextInt();

if (totalCholesterol>= 200)
  System.out.println("Your cholesterol is too high.");
System.out.println("You need to lower that.");
else // SYNTAX ERROR HERE: this else does not match any if
  System.out.println("Good, eat away!");
```

What's wrong here?

```
System.out.print("Enter your total cholesterol level: ");
int totalCholesterol= input.nextInt();


if (totalCholesterol>= 200) { // Now it is correct
  System.out.println("Your cholesterol is too high.");
  System.out.println("You need to lower that.");
} else
  System.out.println("Good, eat away!");
```

Why is this worse?

```
System.out.print("Enter your total cholesterol level:");
int totalCholesterol= input.nextInt();

if (totalCholesterol>= 200)
  System.out.println("Your cholesterol is too high.");
  System.out.println("You need to lower that.");
```

Why is this worse?

```
System.out.print("Enter your total cholesterol level:");
int totalCholesterol= input.nextInt();

if (totalCholesterol>= 200)
  System.out.println("Your cholesterol is too high.");
System.out.println("You need to lower that.");
// NO SYNTAX ERROR
// NO RUNTIME ERROR
// It is a Bug/logical error because it says to lower
//  the cholesterol even if it is fine.
```

Why is this worse?

```
System.out.print("Enter your total cholesterol level: ");
int totalCholesterol= input.nextInt();


if (totalCholesterol>= 200){ // correct
  System.out.println("Your cholesterol is too high.");
  System.out.println("You need to lower that.");
}
```

What about complex conditions?

- For example: Computing Taxes: the income tax rate is calculated based on the filing status and taxable income (so, we need multiple / complex logical conditions)
  - There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household combined with earnings.

Marginal Tax Rate	Single (0)	Married Filing Jointly or Qualified Widow(er) (1)	Married Filing Separately (2)	Head of Household (3)
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	$68,525 – $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+

Logical Operators

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

Truth Table for Operator !

p	!p	Example (assume age = 24, gender = 'F')
`true`	`false`	!(age > 18) is false, because (age > 18) is true.
`false`	`true`	!(gender != 'F') is true, because (grade != 'F') is false.

Truth Table for Operator &&

p1	p2	p1 && p2	Example (assume age = 24, gender = 'F')
false	false	false	(age > 18) && (gender == 'F') is true, because (age > 18) and (gender == 'F') are both true.
false	true	false	
true	false	false	(age > 18) && (gender != 'F') is false, because (gender != 'F') is false.
true	true	true	

Truth Table for Operator ||

p1	p2	p1 || p2	Example (assume age = 24, gender = 'F')
false	false	false	(age > 34) || (gender == 'F') is true, because (gender == 'F') is true.
false	true	true	
true	false	true	(age > 34) || (gender == 'M') is false, because (age > 34) and (gender == 'M') are both false.
true	true	true	

Truth Table for Operator ^

p1	p2	p1 ^ p2	Example (assume age = 24, gender = 'F')
false	false	false	(age > 34) ^ (gender == 'F') is true, because (age > 34) is false but (gender == 'F') is true.
false	true	true	
true	false	true	(age > 34) ^ (gender == 'M') is false, because (age > 34) and (gender == 'M') are both false.
true	true	false	

p1 ^ p2 = p1 != p2

Logical Operators Examples

- What is the result?

```
boolean result;
result = (5 <= 9);                  // true
result = !(5 <= 9);                 // false
result = (3.9 > 3.19);              //true
result = ('a' == 'A');              //false
result = (5 <= 9 && 8 > 9); //false
result = (5 <= 9 || 8 > 9); //true
```

Logical Operators 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)));
```

Determining Leap Year

Consider a program that 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
```

Determining Leap Year

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

2000: leap?

(true && false) || true = false || true = true

1900: leap?

(true && false) || false = false || false = false

2026: leap?

(false && NA) || false = false || false = false

2020: leap?

(true && true) || NA= true || NA= true

The unconditional & and | Operators

- false && p2 (it does not execute p2) = false
  - && is called short-cut operator
    - if the first operand is false, the conjunction is immediately false (skips the evaluation of the second operand)
  - sometimes is what we want, e.g.:

ref!=null && ref.property==value

    - ref.property is not evaluated for null objects

The unconditional & and | Operators

- true || p2 (it does not execute p2) = true
  - || is called short-cut operator
    - if the first operand is true, the disjunction is immediately true (skips the evaluation of the second operand)

The unconditional & and | Operators

If x is 1, what is x after these expressions?

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

false and NA = false

```
(x > 1) & (x++ < 10)           2
```

still evaluated

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

true or NA = true

```
(1 == x) | (10 > x++)?         2
```

still evaluated

switch Statements

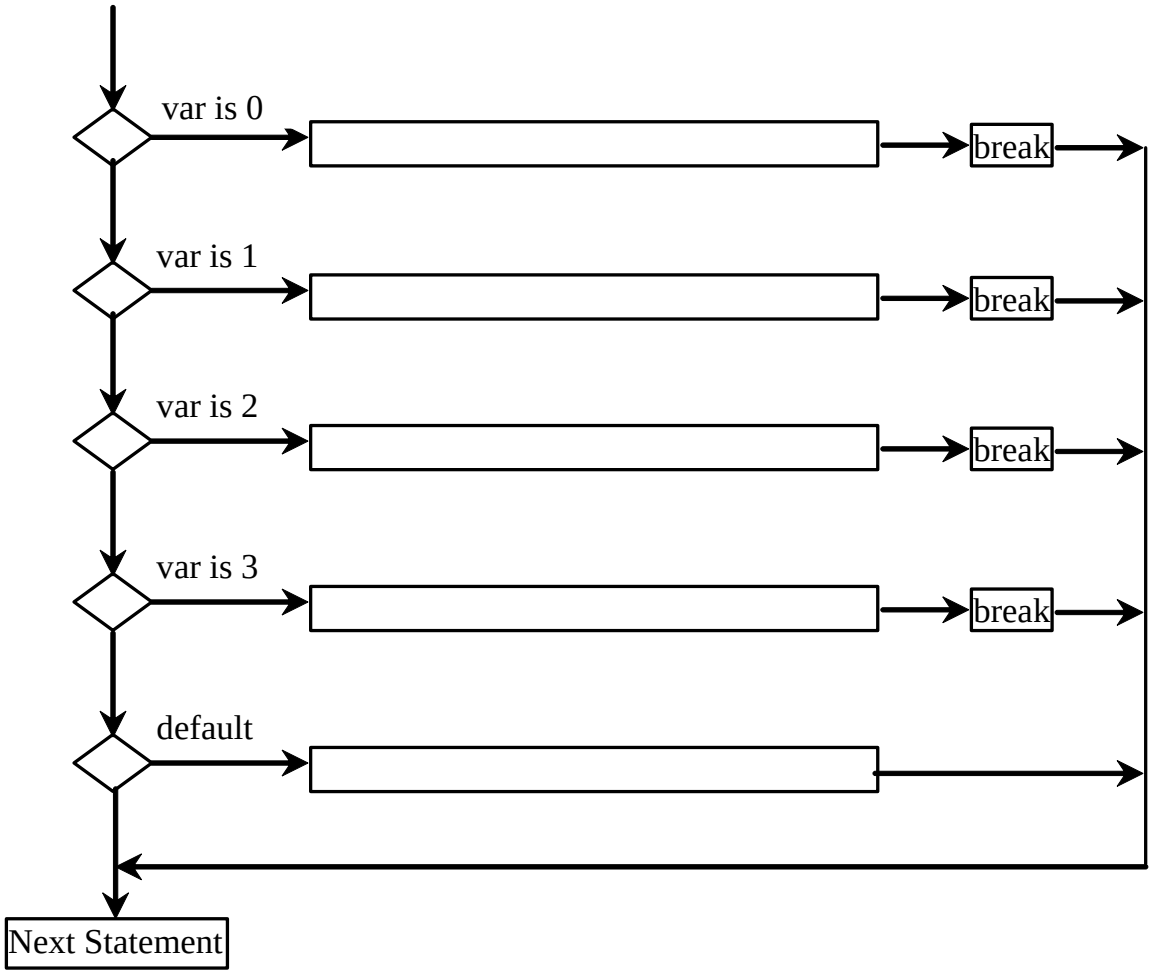
```
switch (var) {
   case 0:          ...;
                    break;
   case 1:          ...;
                    break;
   case 2:          ...;
                    break;
   case 3:          ...;
                    break;
   default:         ...;
}
```

switch Statements

```
Good for menus:
  Enter an option:
  A = burger
  B = fries
  C = exit

switch(option){
case 'A': System.out.println("burger");
          break;
case 'B': System.out.println("fries");
          break;
case 'C': System.out.println("exit");
          break;
}
```

`switch` Statement Flow Chart



switch Statement Rules

char, byte, short, int, String

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

value1, ..., and valueN are **constant expressions** of the **same data type** as the value of the switch-expression

- constant = they cannot contain variables in the expression, such as x+y

switch Statement Rules

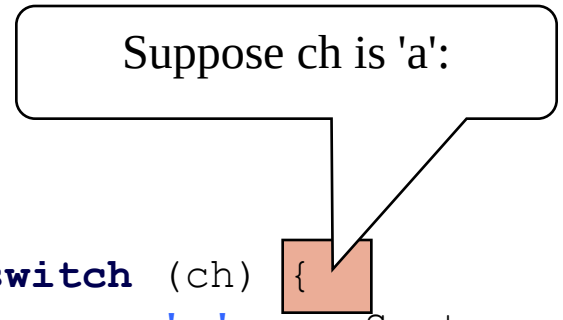
break is optional, but it terminates the remainder of the switch statement

default is optional - executed 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);
}
```

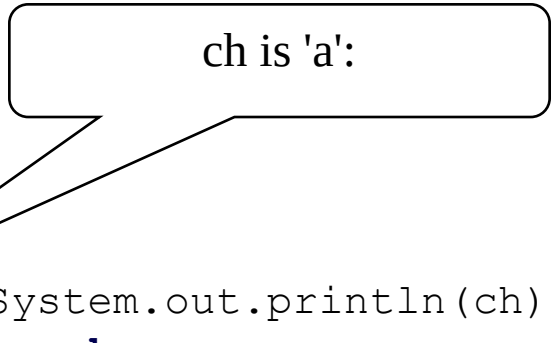
execution in sequential order

Trace switch statement 1



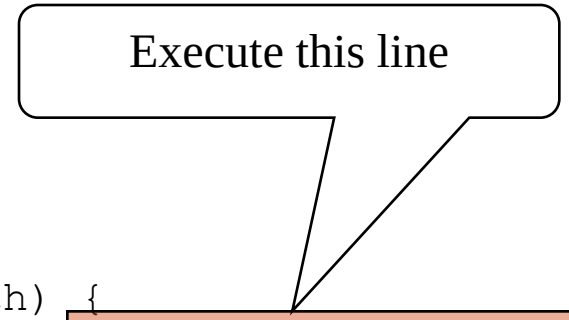

```
switch (ch) {
  case 'a':    System.out.println(ch);
               break;
  case 'b':    System.out.println(ch);
               break;
  case 'c':    System.out.println(ch);
}
```

Trace switch statement 1




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

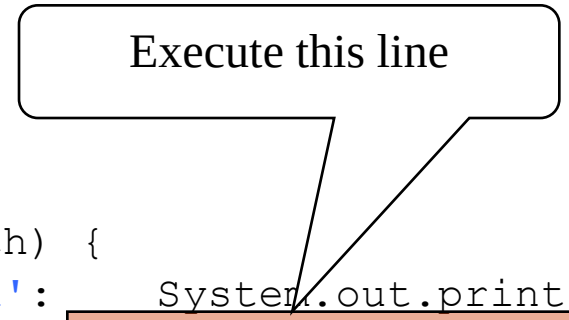
Trace switch statement 1




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

a

Trace switch statement 1




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

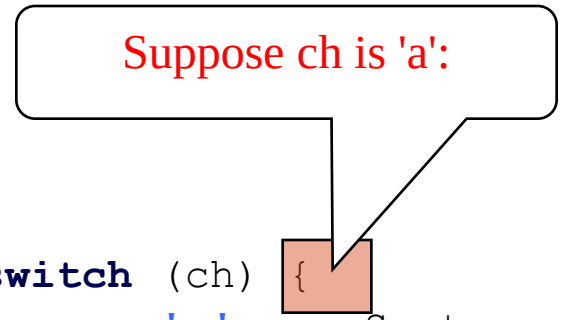
a

Trace switch statement 1

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

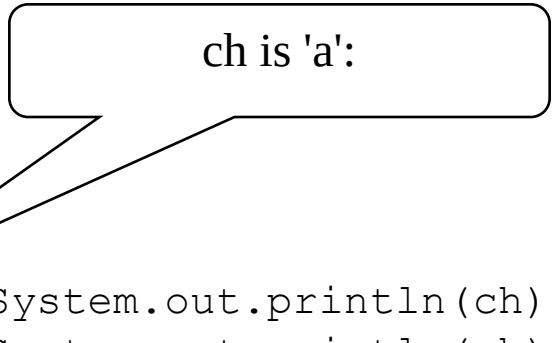
a

Trace switch statement 2



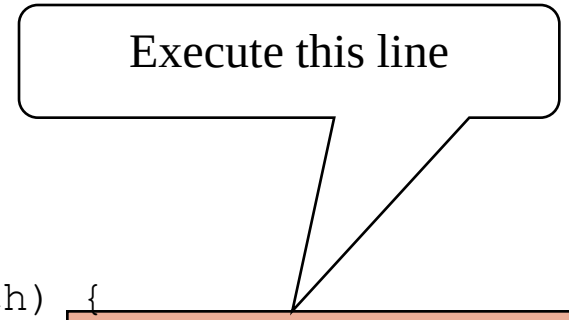

```
switch (ch) {
  case 'a':     System.out.println(ch);
  case 'b':     System.out.println(ch);
  case 'c':     System.out.println(ch);
}
```

Trace switch statement 2




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

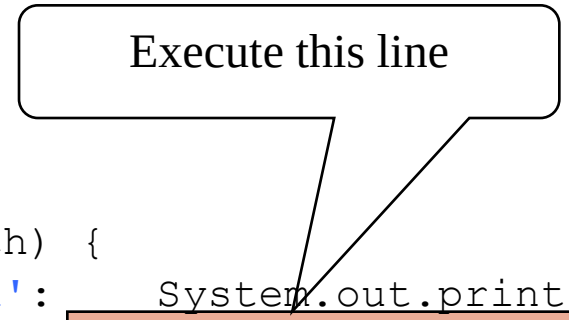
Trace switch statement 2




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

a

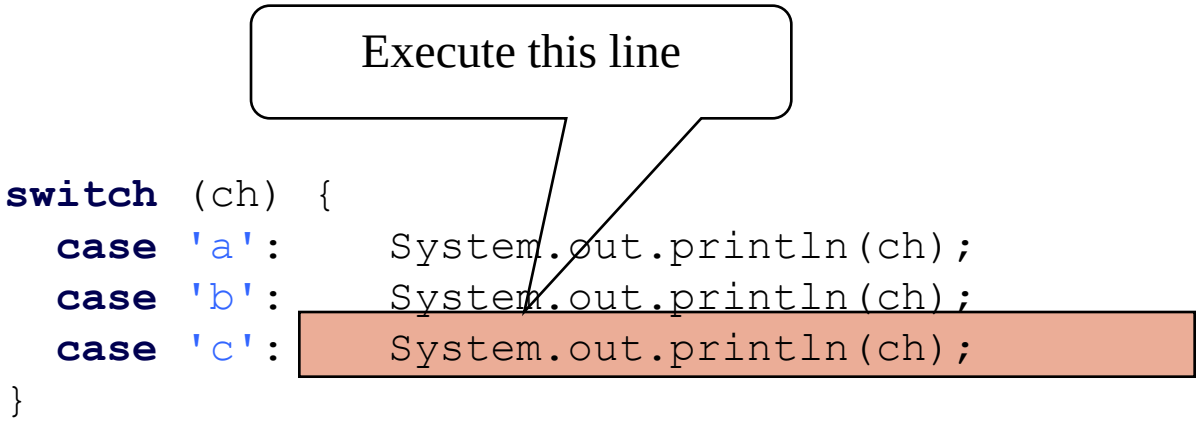
Trace switch statement 2




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

aa

Trace switch statement 2



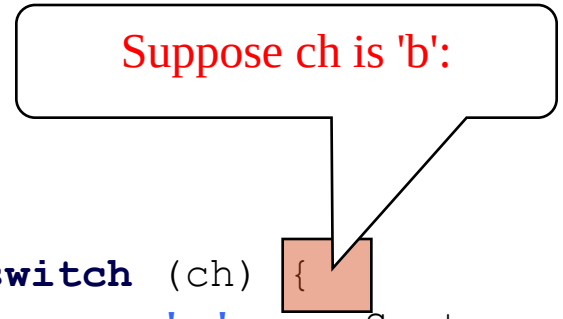
aaa

Trace switch statement 2

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

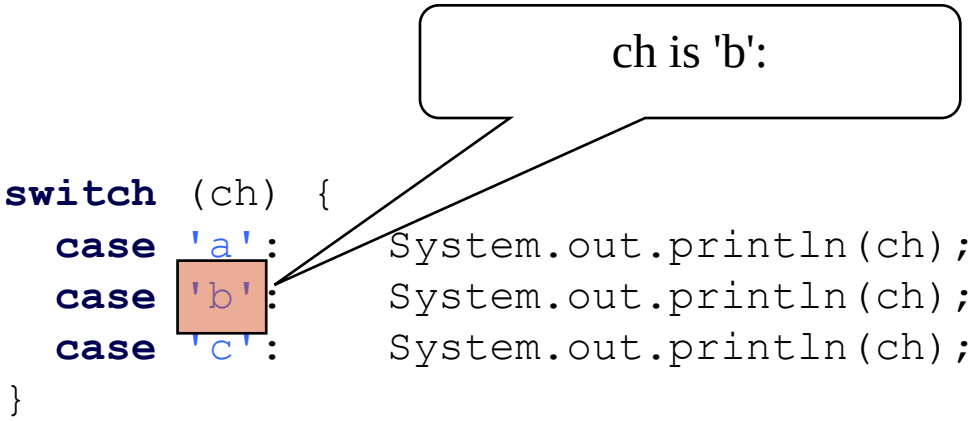
aaa

Trace switch statement 3

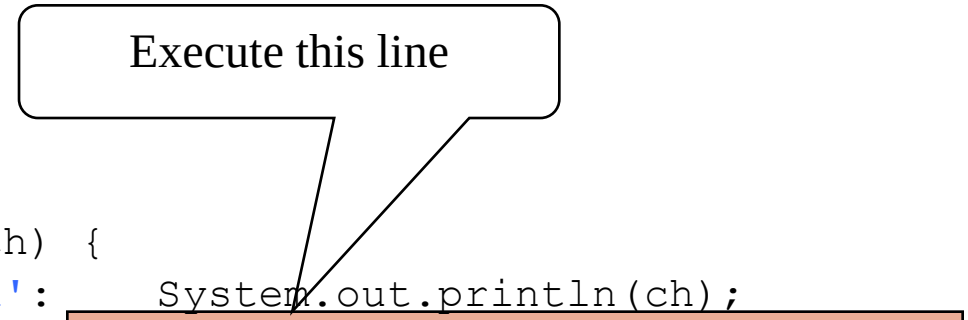



```
switch (ch) {
  case 'a':     System.out.println(ch);
  case 'b':     System.out.println(ch);
  case 'c':     System.out.println(ch);
}
```

Trace switch statement 3



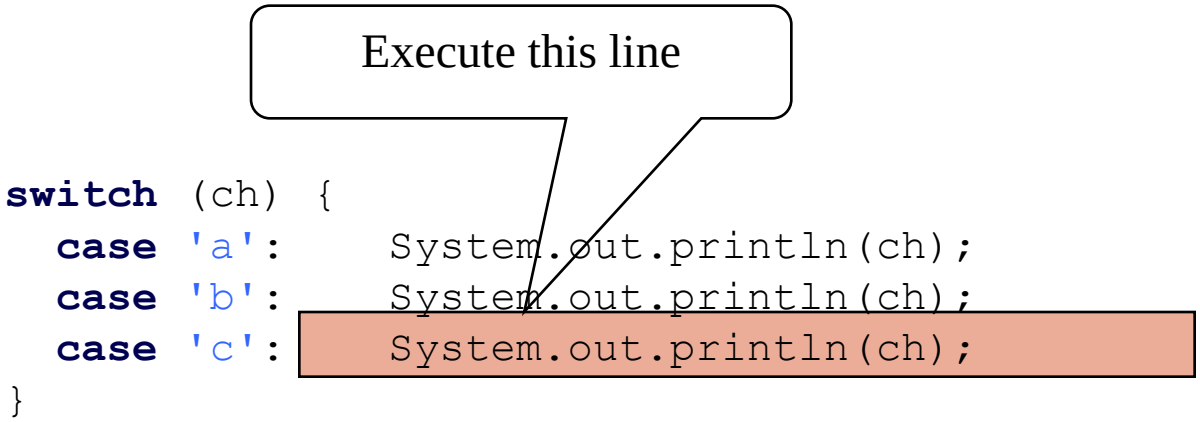
Trace switch statement 3




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

b

Trace switch statement 3



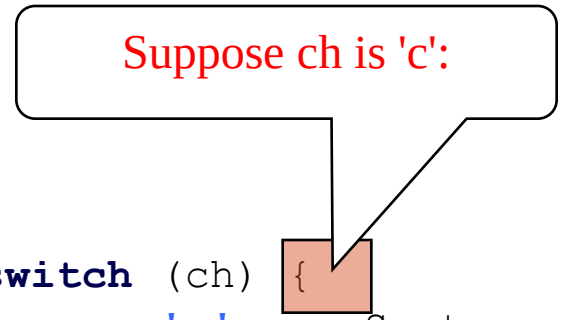
bb

Trace switch statement 3

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

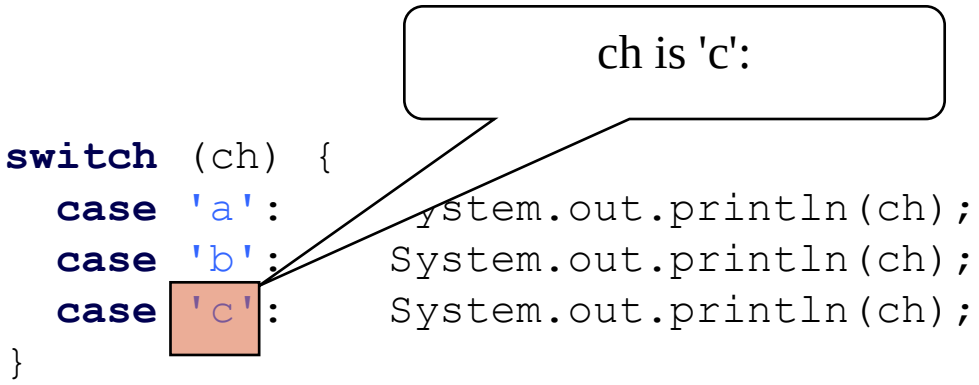
bb

Trace switch statement 4

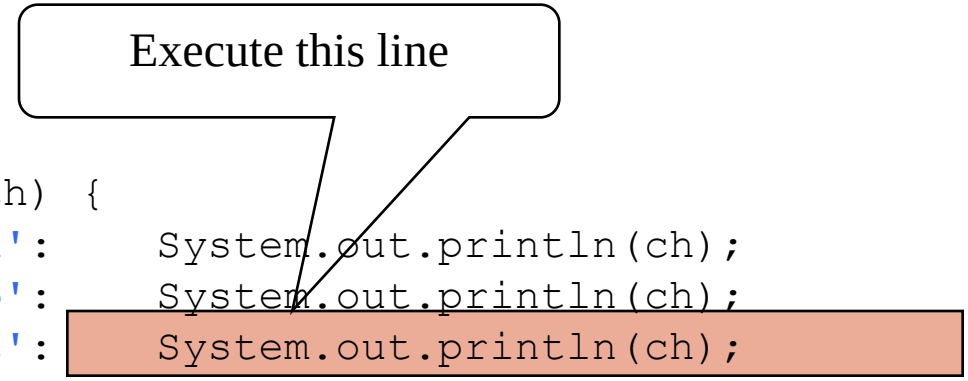



```
switch (ch) {
  case 'a':     System.out.println(ch);
  case 'b':     System.out.println(ch);
  case 'c':     System.out.println(ch);
}
```

Trace switch statement 4



Trace switch statement 4




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

c

Trace switch statement 4

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

c

Conditional Operator

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

is equivalent to

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

- Conditional Operator expression form:
  (boolean-expression) ? expression1 : expression2

Conditional Operator

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

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

Operator Precedence

1. var++, var--
2. +, - (Unary plus and minus), ++var,--var
3. (type) Casting
4. ! (Not)
5. *, /, % (Multiplication, division, and remainder)
6. +, - (Binary addition and subtraction)
7. <, <=, >, >= (Comparison)
8. ==, !=; (Equality)
9. ^ (Exclusive OR)
10. && (Conditional AND) Short-circuit AND
11. || (Conditional OR) Short-circuit OR
12. =, +=, -=, *=, /=, %= (Assignment operator)

Example

Applying the operator precedence, associativity rules and left-to-right execution, the expression 3 + 4 * 4 > 5 * (4 + 3) - 1 is evaluated as follows:

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

3 + 16 > 5 * (4 + 3) - 1

19 > 5 * (4 + 3) - 1

19 > 5 * 7 - 1

19 > 35 - 1

19 > 34

false
```

Operator Associativity

All binary operators except assignment operators are ***left-associative***.

Example:

10 - 5 - 4 = (10 - 5) - 4 = 5 - 4 = 1

The assignment operators are ***right-associative***.

Example:

a = b += c = 5;

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

ChangeMaker Example Revisited

```
System.out.print("Input change amount (1-99):");
originalAmount= scanner.readInt();
if (originalAmount< 1 || originalAmount> 99)
  System.out.println("ERROR: Out of range.");
else{
  numQuarters= originalAmount/ 25;
  remainder = originalAmount% 25;
  numDimes= remainder / 10;
  remainder = remainder % 10;
  numNickels= remainder / 5;
  numPennies= remainder % 5;
  if (numQuarters!= 0) // Do not print if zero
       System.out.println(numQuarters+ " quarters");
  if (numDimes!= 0)// Do not print if zero
       System.out.println(numDimes+ " dimes");
  if (numNickels!= 0)// Do not print if zero
       System.out.println(numNickels+ " nickels");
  if (numPennies!= 0)// Do not print if zero
       System.out.println(numPennies+ " pennies");
}
```

ChangeMaker Example Revisited

- Nested ifs:

```
if (numQuarters!= 0){ // Do not print if zero
  System.out.print(numQuarters+ " quarter");
  if (numQuarters== 1) // Do not print s if one
       System.out.println( );
  else
       System.out.println("s"); // print s if more
}
```