Selections

CSE 114, Computer Science 1

Stony Brook University

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

- 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);
  }
}
```

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 ComputeArea {
  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 `boolean` Type and Operators

- Often in a programs you need to compare values:

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

- Java provides six comparison operators (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.

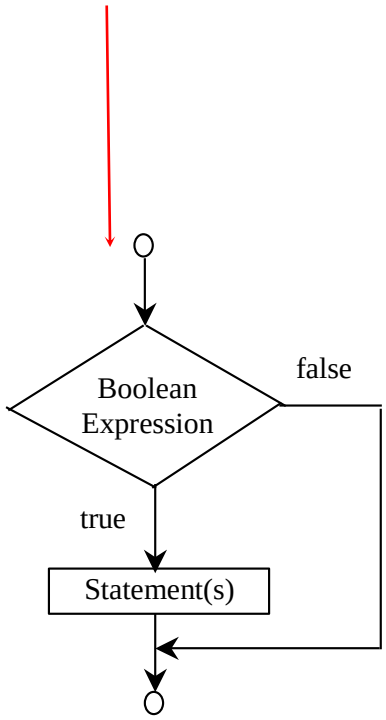
Comparison Operators

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

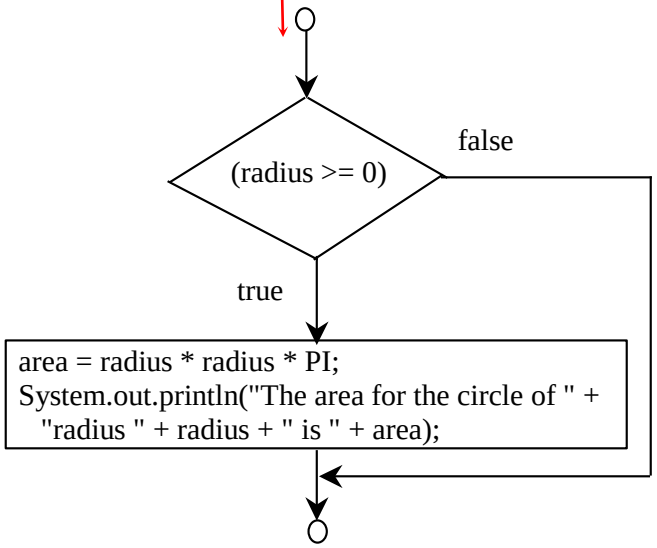
One-way `if` Statements

```
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)

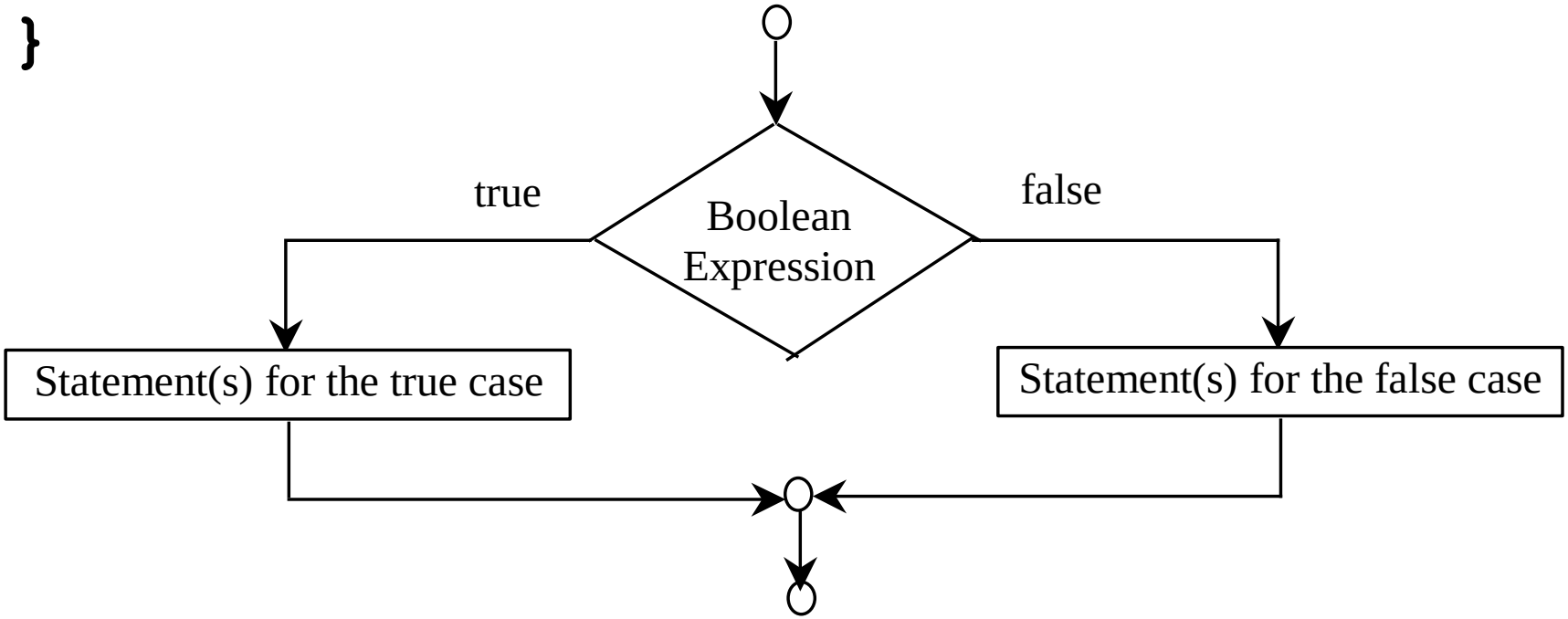
Equivalent

```
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;
}
```



if...else 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.

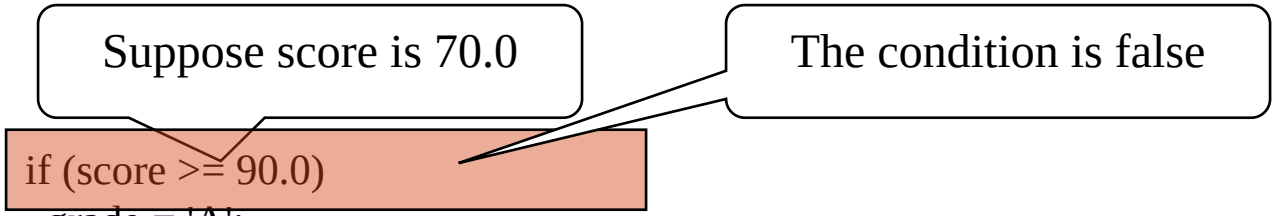
```
if (score >= 90.0)
  grade = 'A';
else
  if (score >= 80.0)
    grade = 'B';
  else
    if (score >= 70.0)
      grade = 'C';
    else
      if (score >= 60.0)
        grade = 'D';
      else
        grade = 'F';
```

Equivalent

Indentation exception for cascading else if statements:

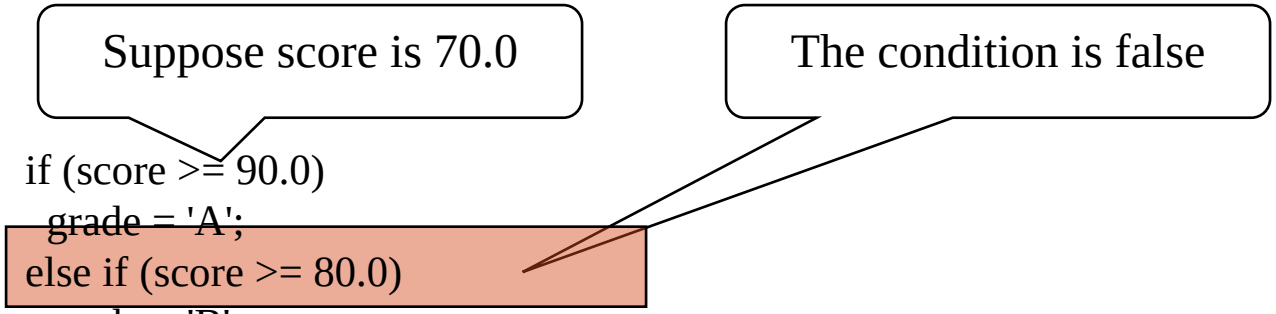
```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

Trace if-else statement



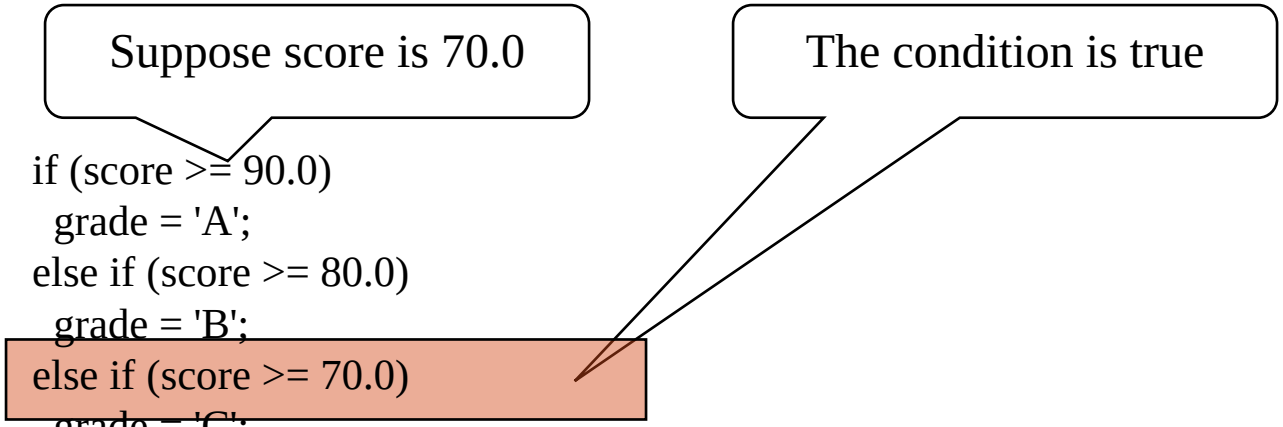

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

Trace if-else statement

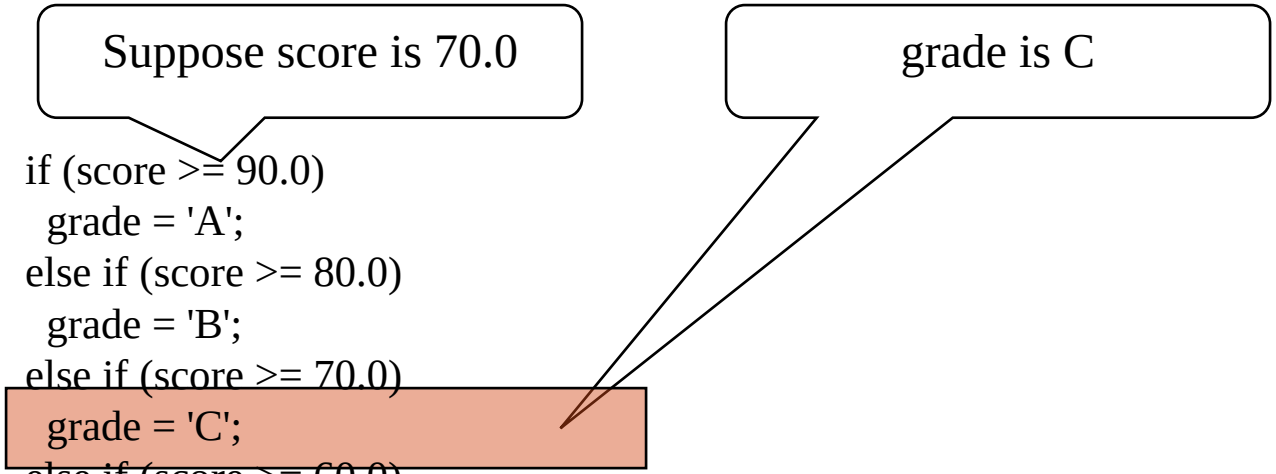



```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

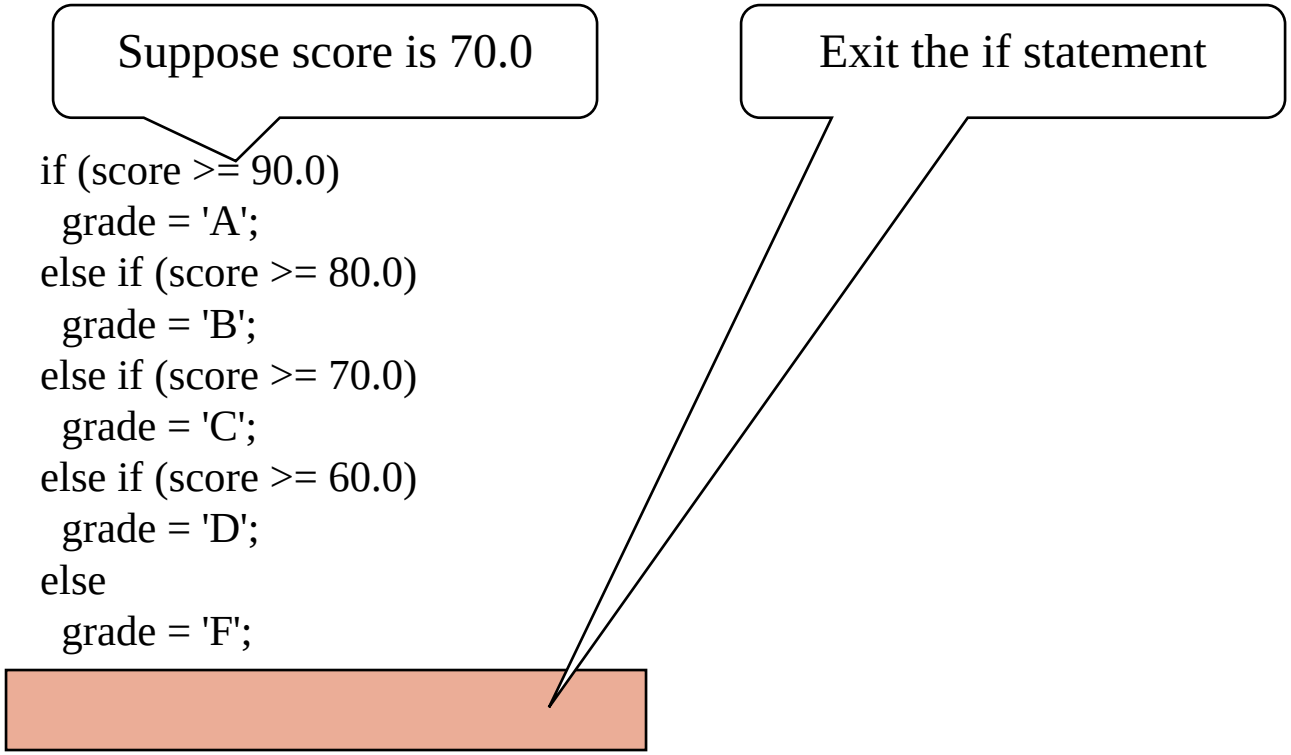
Trace if-else statement



Trace if-else statement



Trace if-else statement



if … else

Indentation mistakes 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:

```
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 or a runtime error, it is a logic error because ";" is a statement (no-operation statement)
- This error often occurs when you use the next-line block style

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 // COMPILER 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 compiler error
// It is a Bug/logical error because it says to lower
//  that 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 US federal income tax is calculated based on the filing status and taxable income (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	

Logical Operators Examples

- What is the result?

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

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

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
```

The unconditional & and | Operators

- The & operator works exactly the same as the && operator, and the | operator works exactly the same as the || operator with one exception:
  - the & and | operators always evaluate both operands

The unconditional & and | Operators

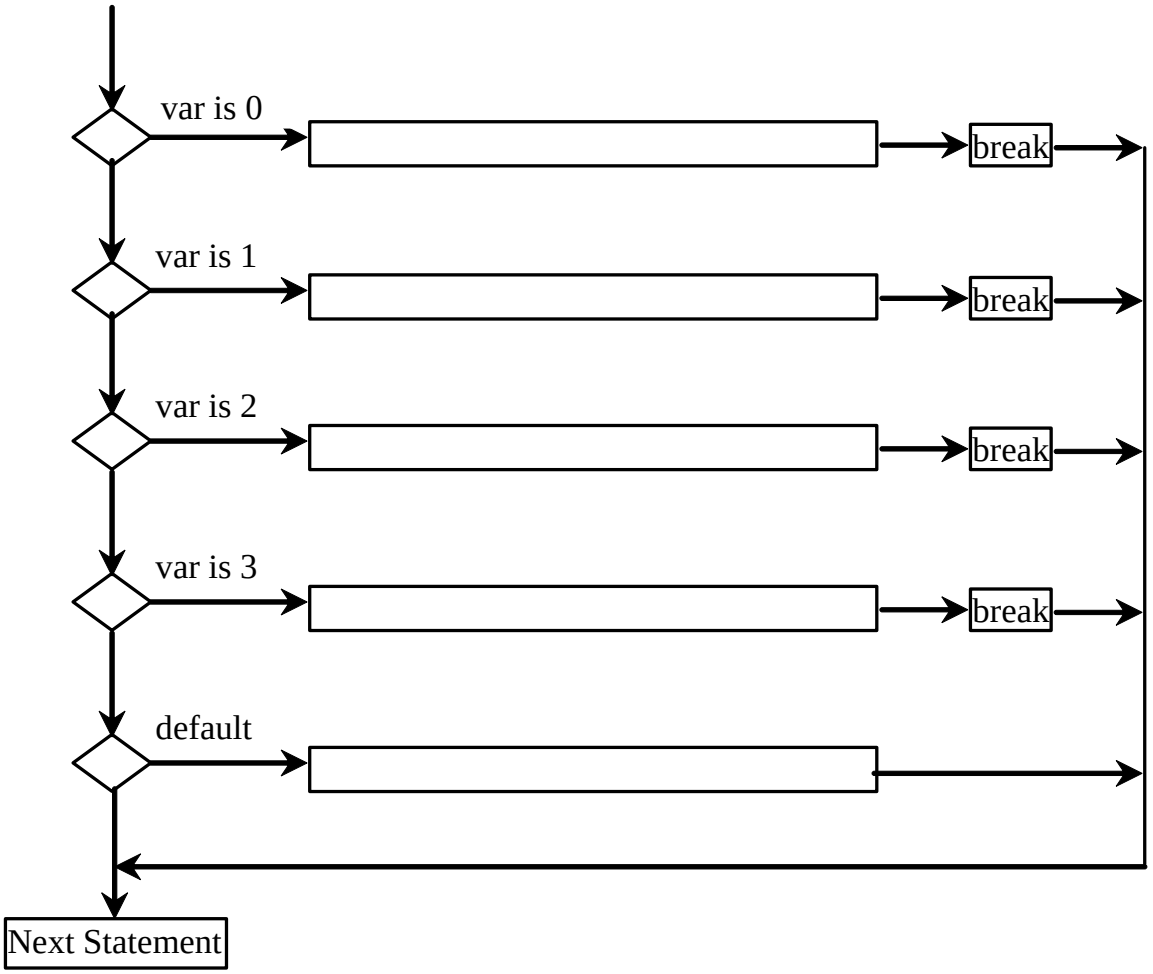
If x is 1, what is x after these expressions:

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

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


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

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

switch Statements

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

`switch` Statement Flow Chart



switch Statement Rules

char, byte, short, int, String

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

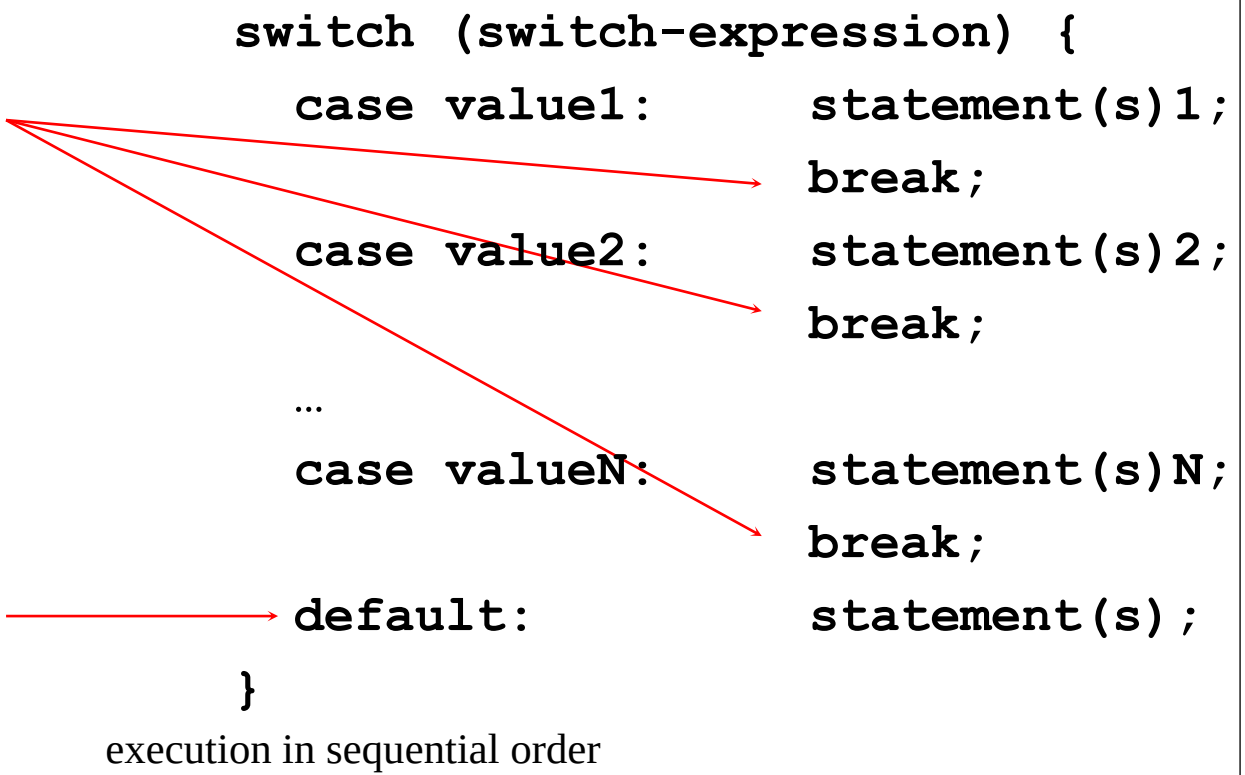
constant = they cannot contain variables in the expression, such as x+y

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

switch Statement Rules

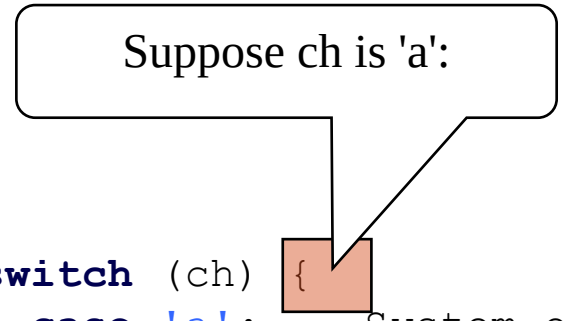
break is optional, but it terminates the remainder of the switch statement

default is optional - executed when none of the specified cases matches the switch-expression.



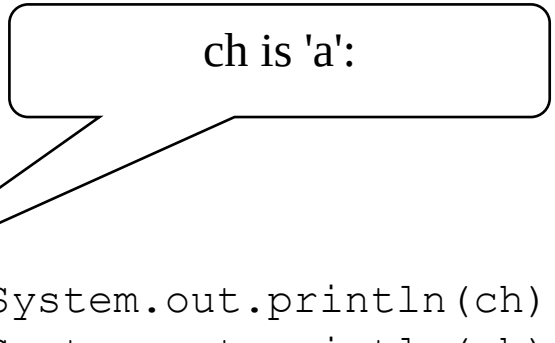
execution in sequential order

Trace switch statement



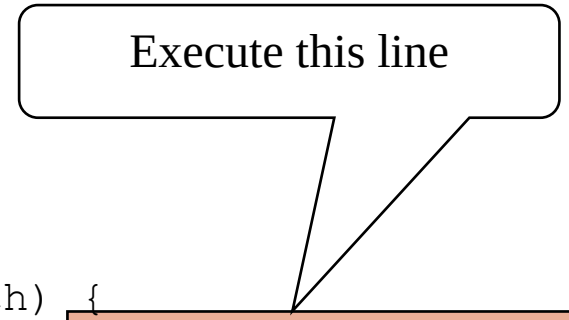

```
switch (ch) {
  case 'a':     System.out.println(ch);
  case 'b':     System.out.println(ch);
  case 'c':     System.out.println(ch);
}
```

Trace switch statement



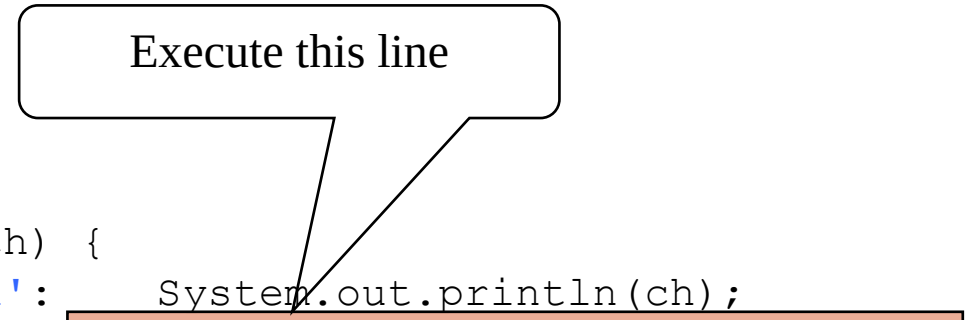

```
switch (ch) {
  case 'a':     System.out.println(ch);
  case 'b':     System.out.println(ch);
  case 'c':     System.out.println(ch);
}
```

Trace switch statement



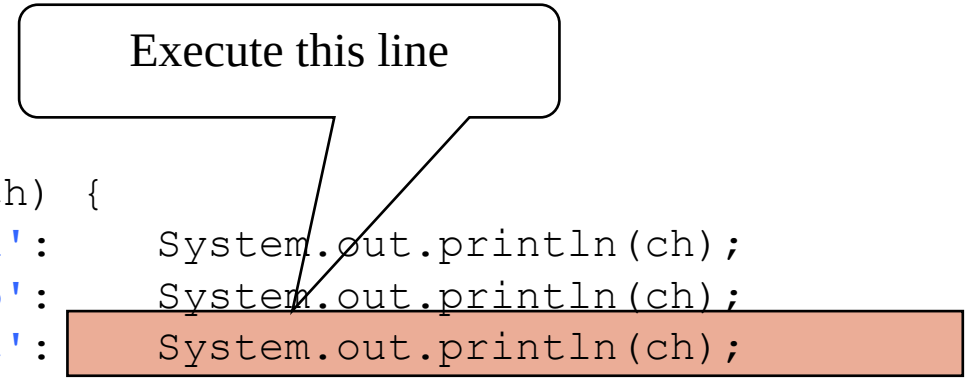

```
switch (ch) {
  case 'a':    System.out.println(ch);
  case 'b':    System.out.println(ch);
  case 'c':    System.out.println(ch);
}
```

Trace switch statement




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

Trace switch statement

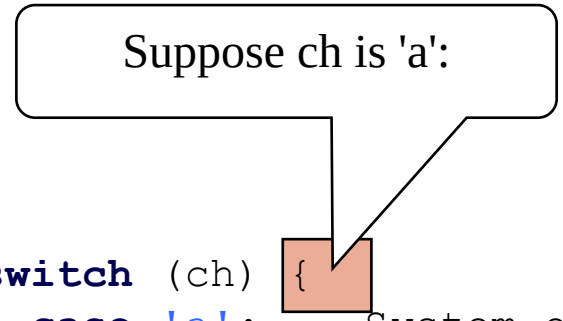



```
switch (ch) {
  case 'a':    System.out.println(ch);
  case 'b':    System.out.println(ch);
  case 'c':    System.out.println(ch);
}
```

Trace switch statement

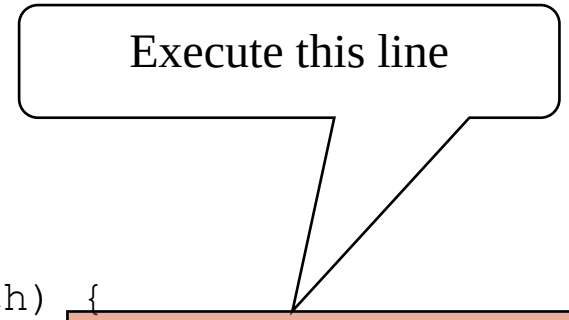
```
switch (ch) {
  case 'a':    System.out.println(ch);
  case 'b':    System.out.println(ch);
  case 'c':    System.out.println(ch);
}
```

Trace switch statement 2



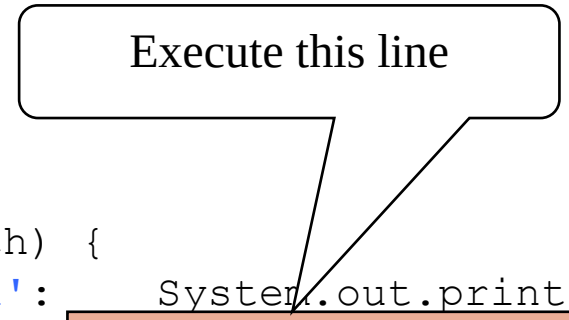

```
switch (ch) {
  case 'a':     System.out.println(ch);
                break;
  case 'b':     System.out.println(ch);
                break;
  case 'c':     System.out.println(ch);
}
```

Trace switch statement 2




```
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 2




```
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 2

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

Conditional Operator

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

is equivalent to

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

(boolean-expression) ? expression1 : expression2

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

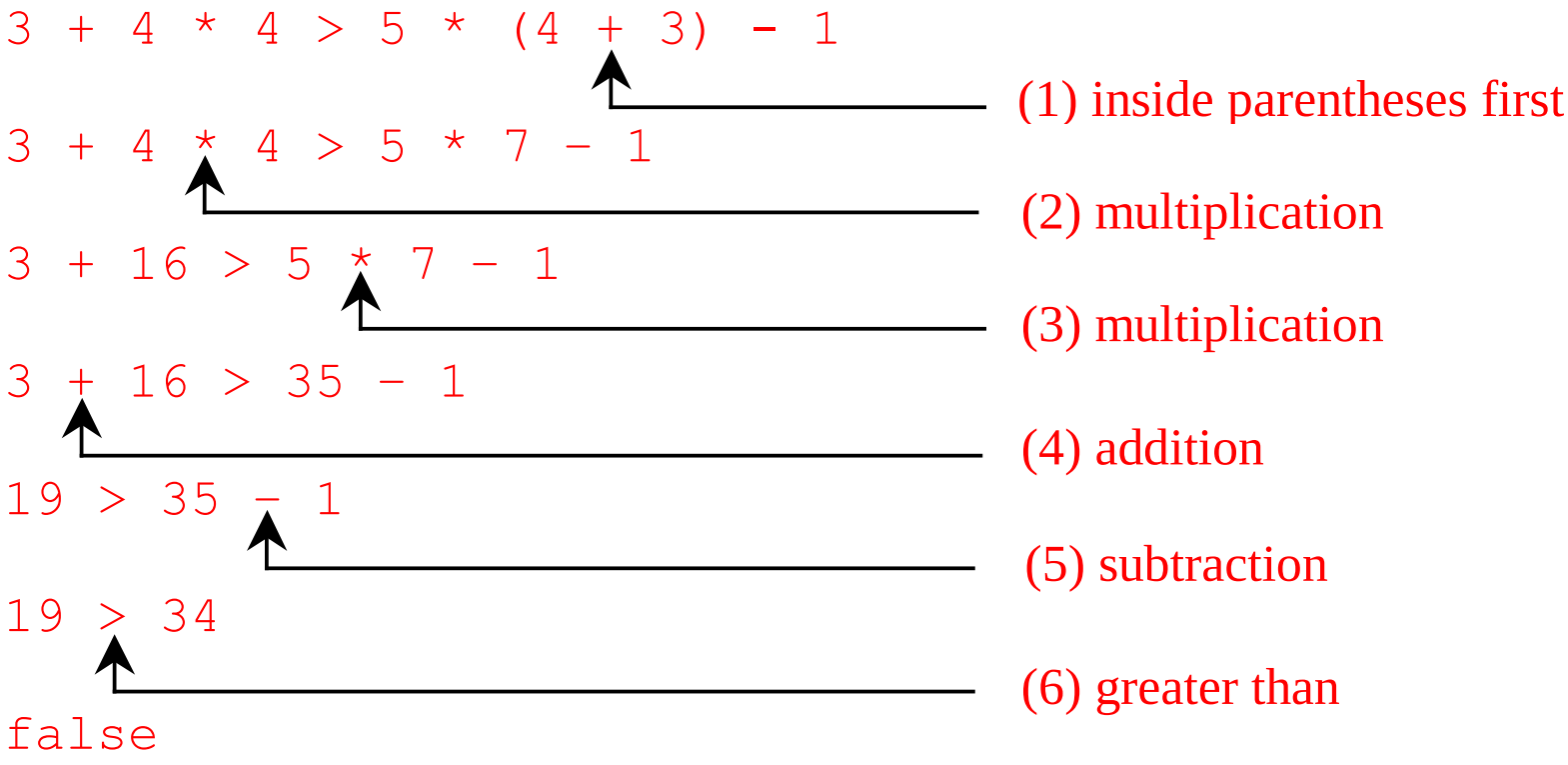
Operator Precedence

- `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)`

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

Example

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



Operator Associativity

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

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

The assignment operators are *right-associative:*

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
}
```