ECE25100: OBJECT ORIENTED PROGRAMMING

Note 3 _ Control Structures

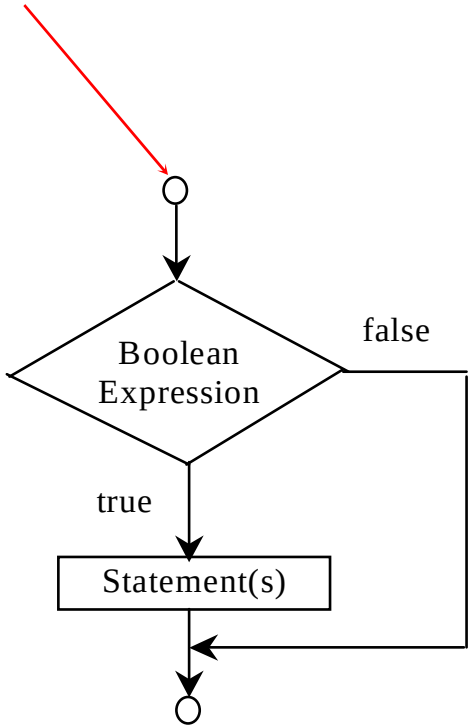
Instructor: Xiaoli Yang

Selection Statements

- if Statements
- switch Statements
- Conditional Operators

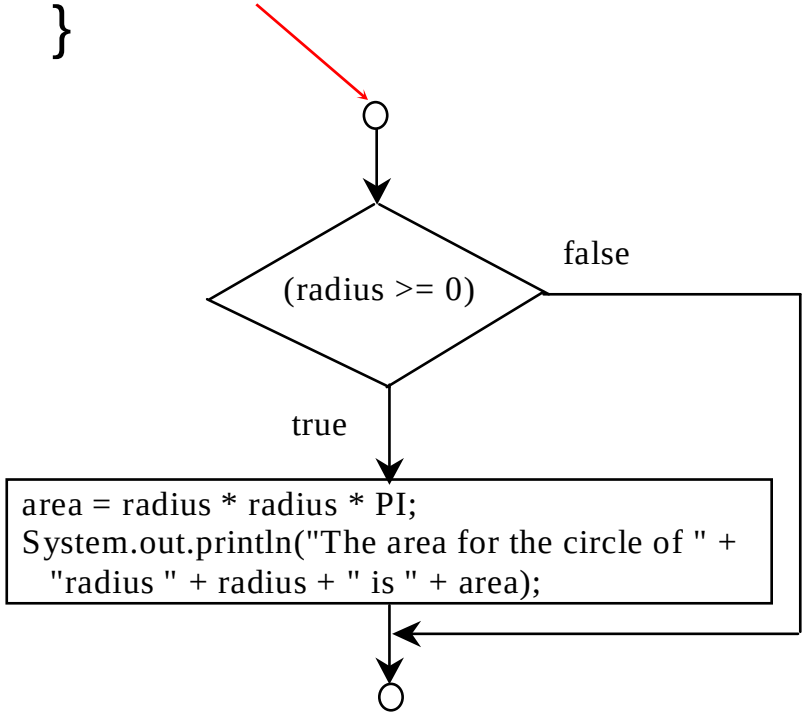
Simple `if` Statements

```
if (booleanExpression) {
  statement(s);
}
```



(A)

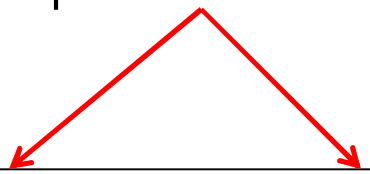
```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area"
    for the circle of radius "
    + radius + " is " + area);
}
```



(B)

Note

Outer parentheses required



Braces can be omitted if the block contains a single statement

```
if ((i > 0) && (i < 10)) {
  System.out.println("i is " +i + " ,an integer between 0 and 10");
}
```

Caution

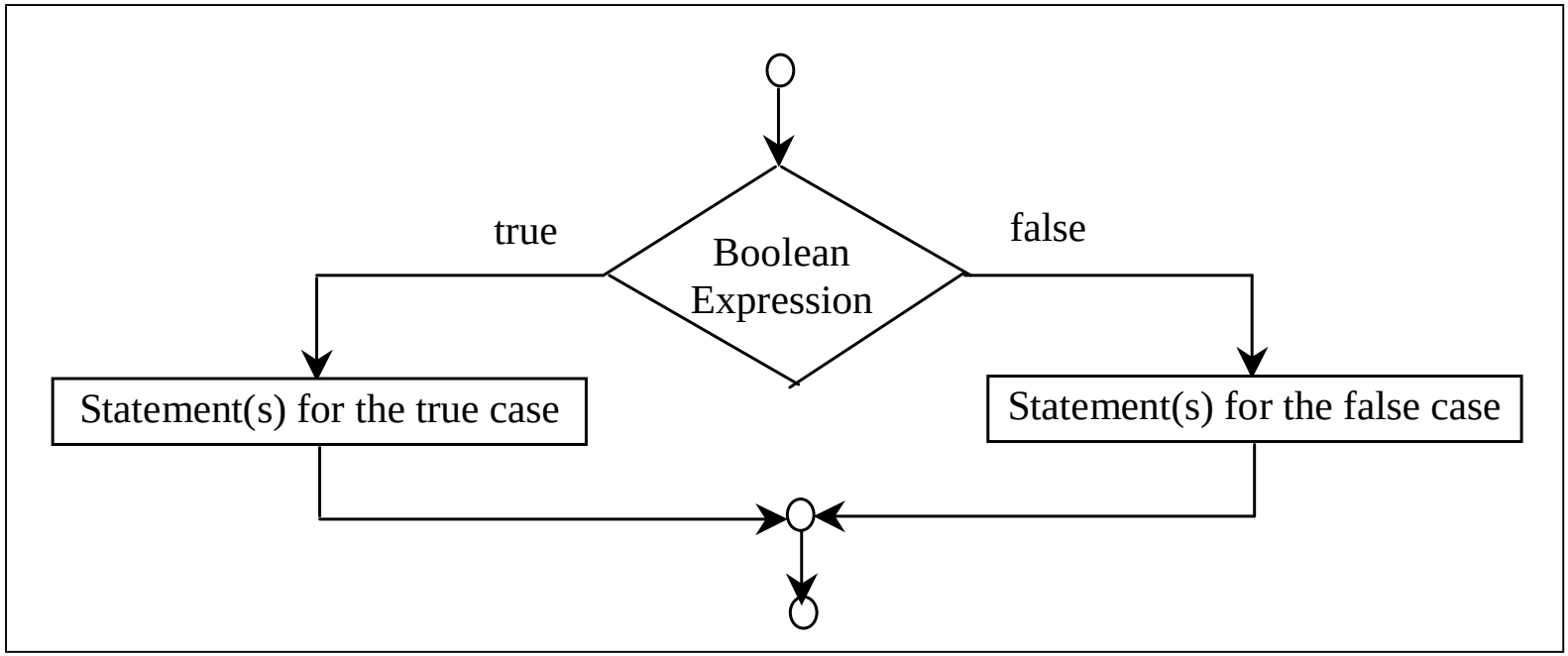
- Adding a semicolon at the end of an if clause is a common mistake.

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

Wrong

The `if...else` Statement

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



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';
```

- Example: multiple if-else example

  multipleIf2Tester.java

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.

CAUTION

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

(a)

Equivalent

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

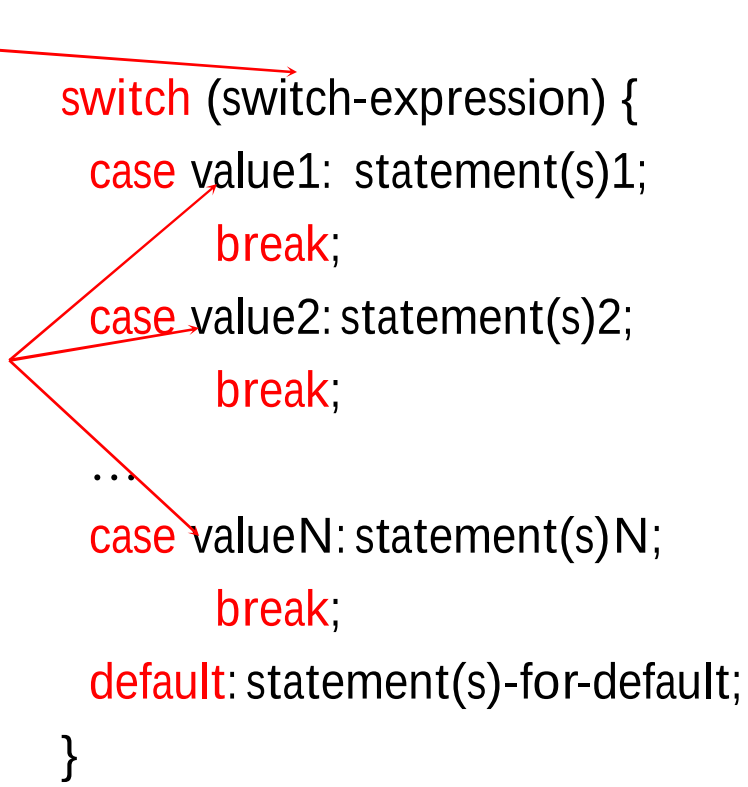
(b)

`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 Statement Rules (Cont.)

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.

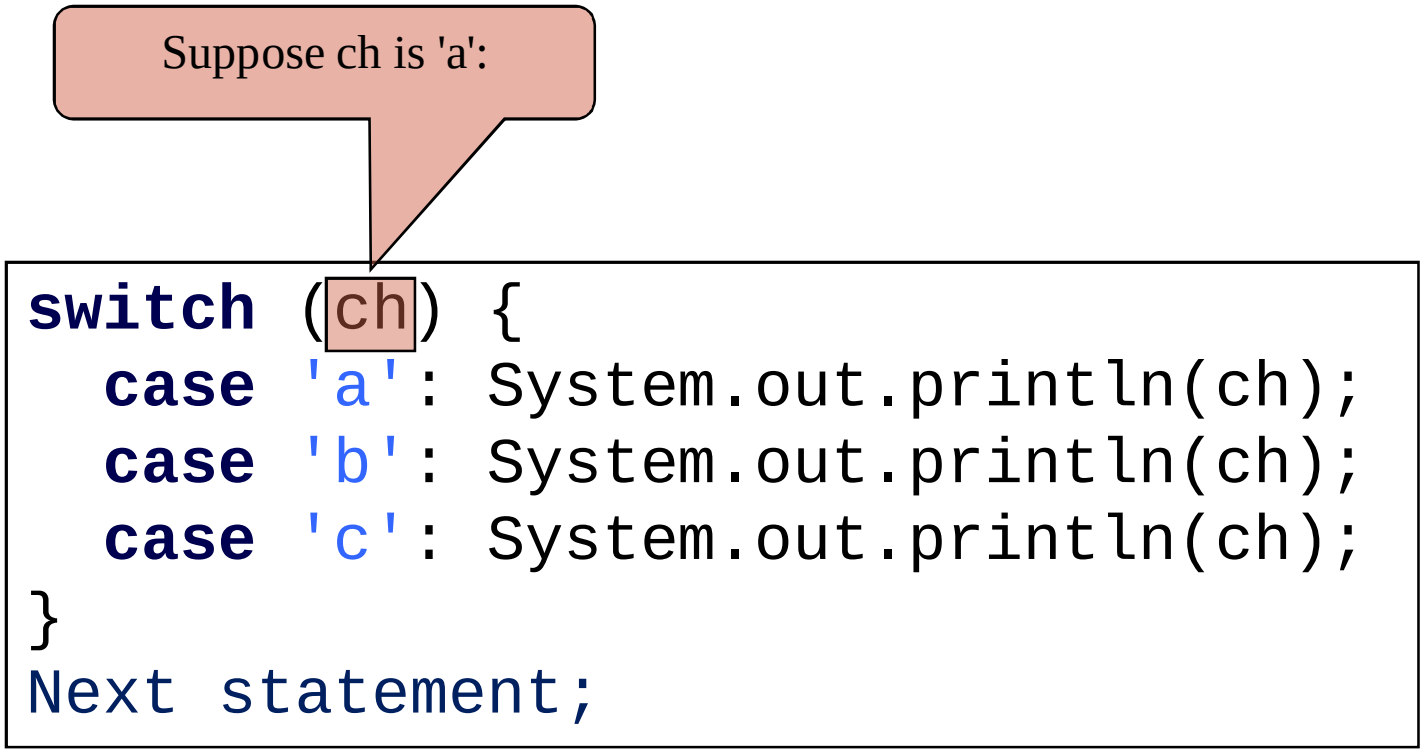
```
switch (switch-expression) {
  case value1:  statement(s)1;
        break;
  case value2: statement(s)2;
        break;
  …
  case valueN: statement(s)N;
        break;
  default: statement(s)-for-default;
}
```

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

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.

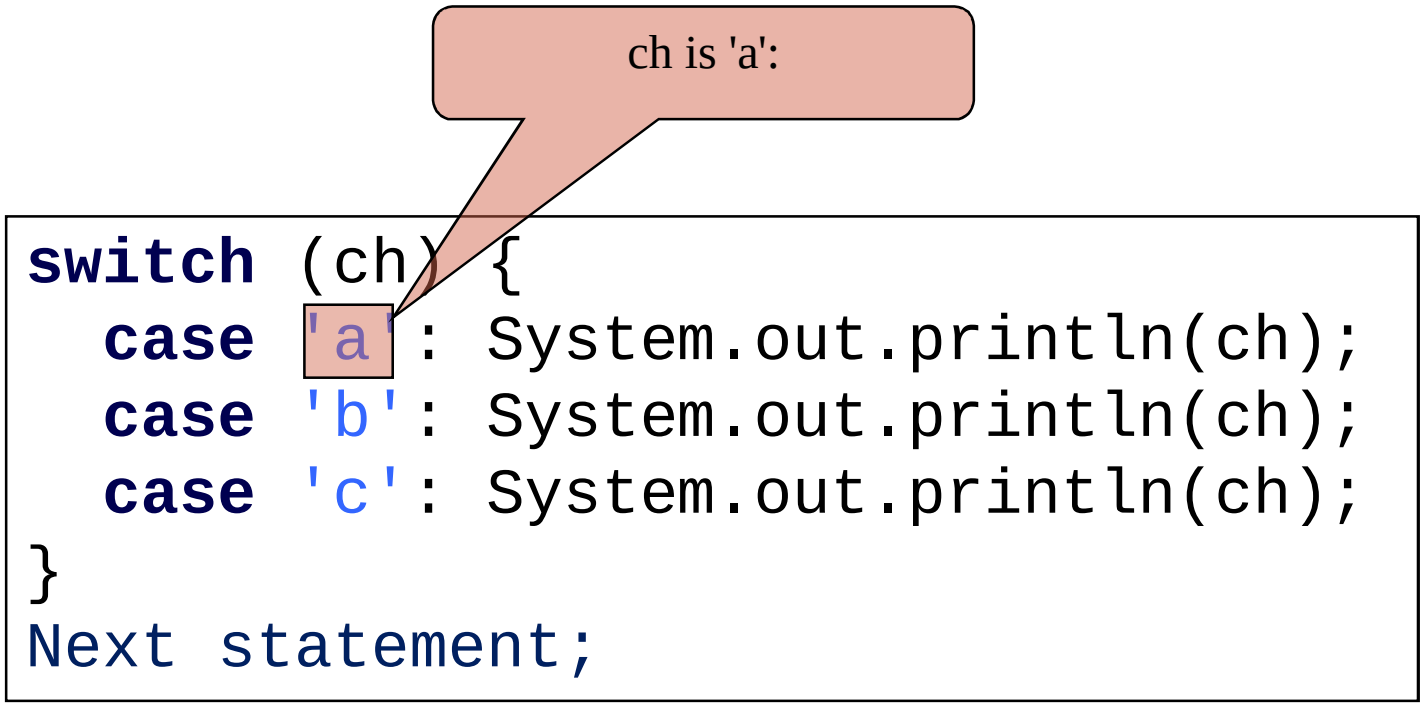
animation

Trace switch statement



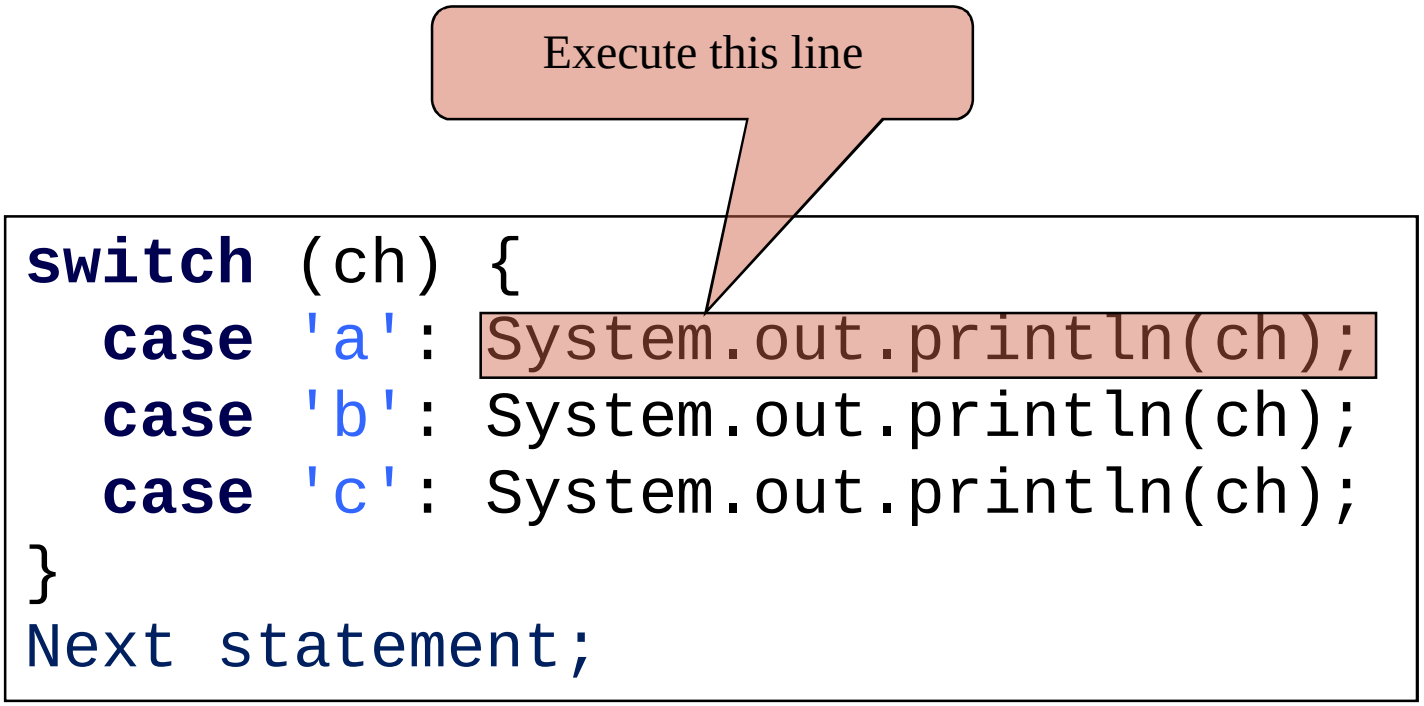
animation

Trace switch statement



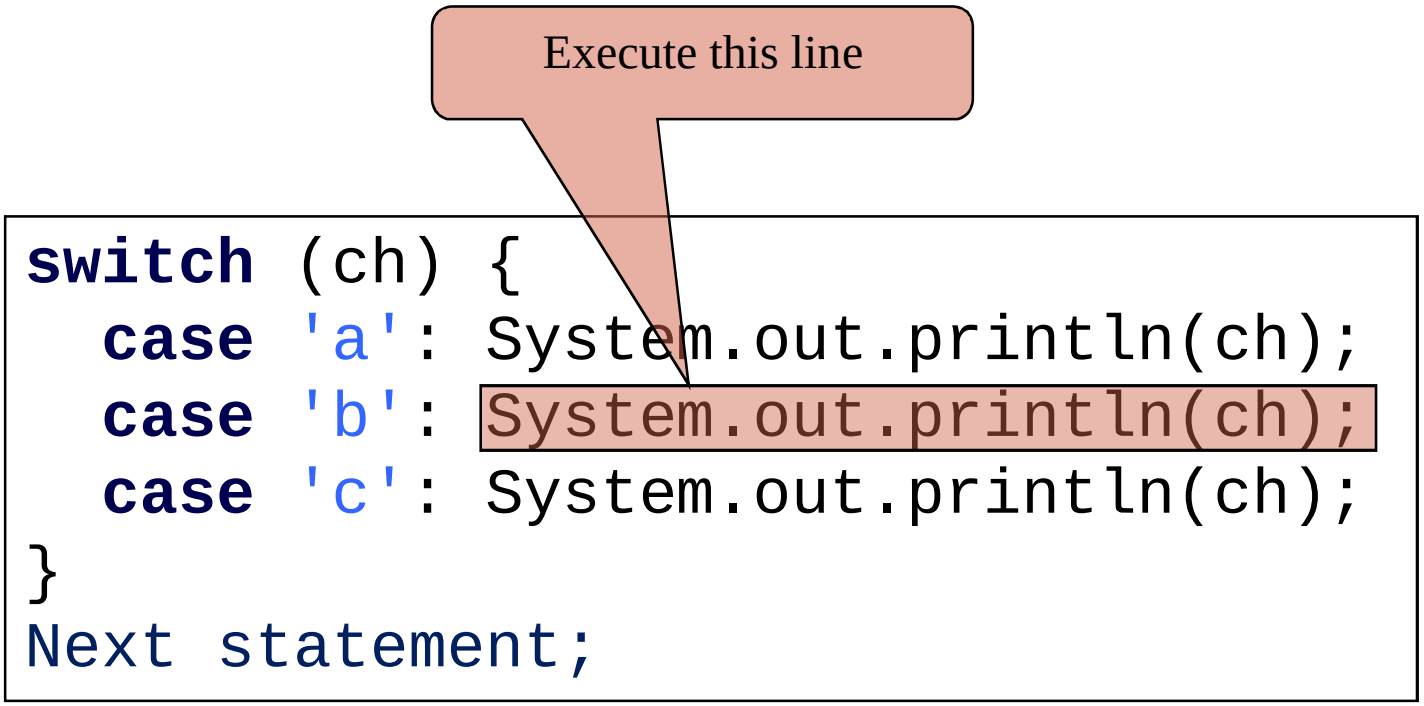
animation

Trace switch statement



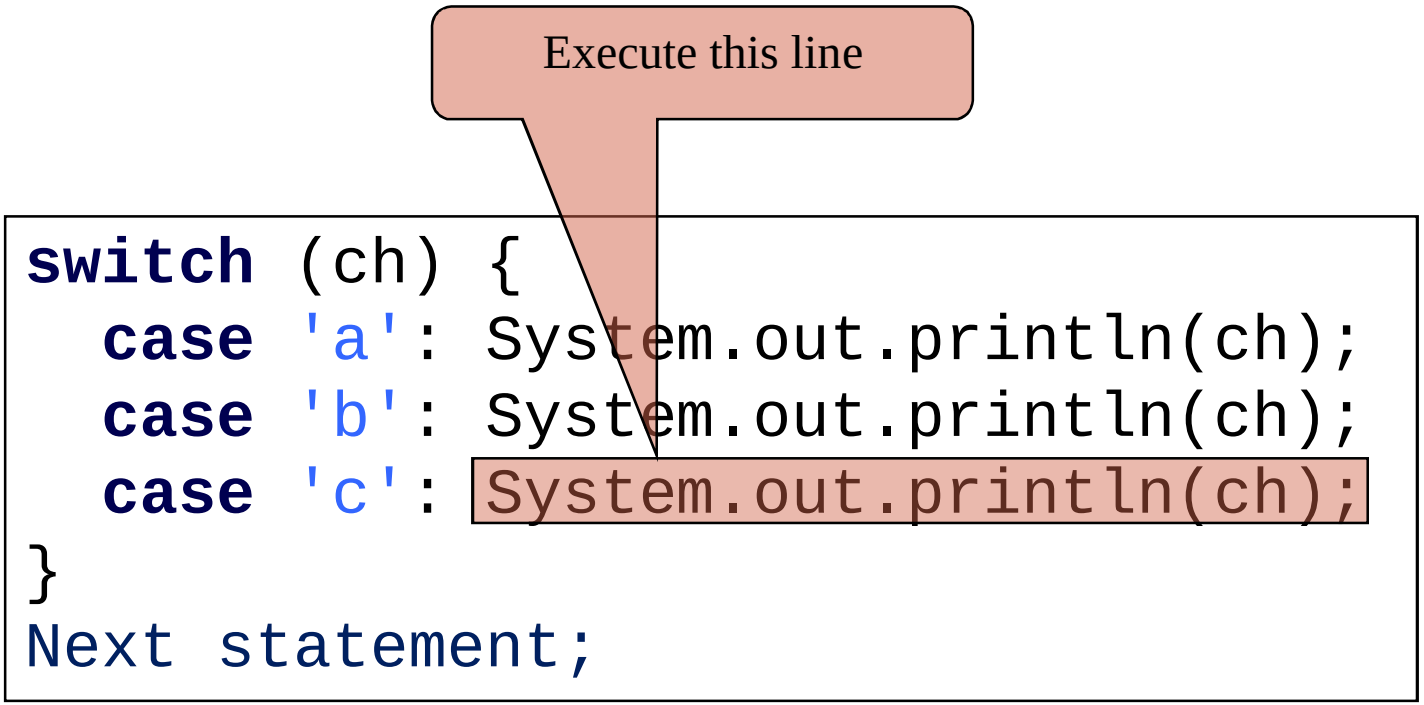
animation

Trace switch statement



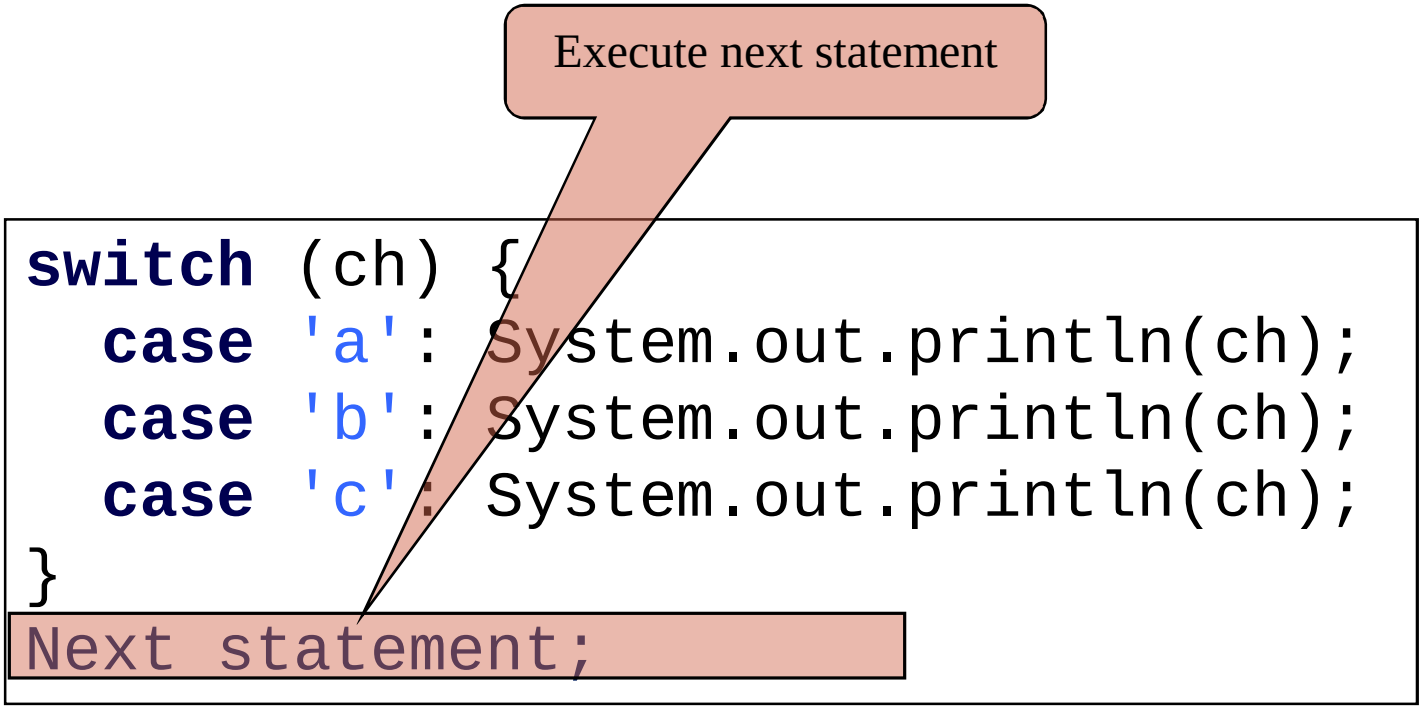
animation

Trace switch statement



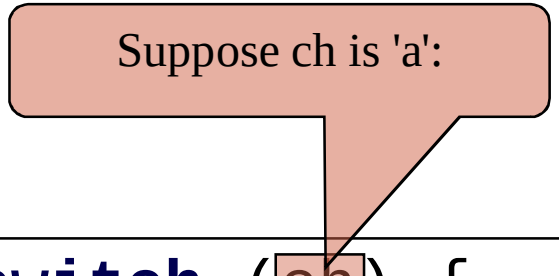
animation

Trace switch statement



animation

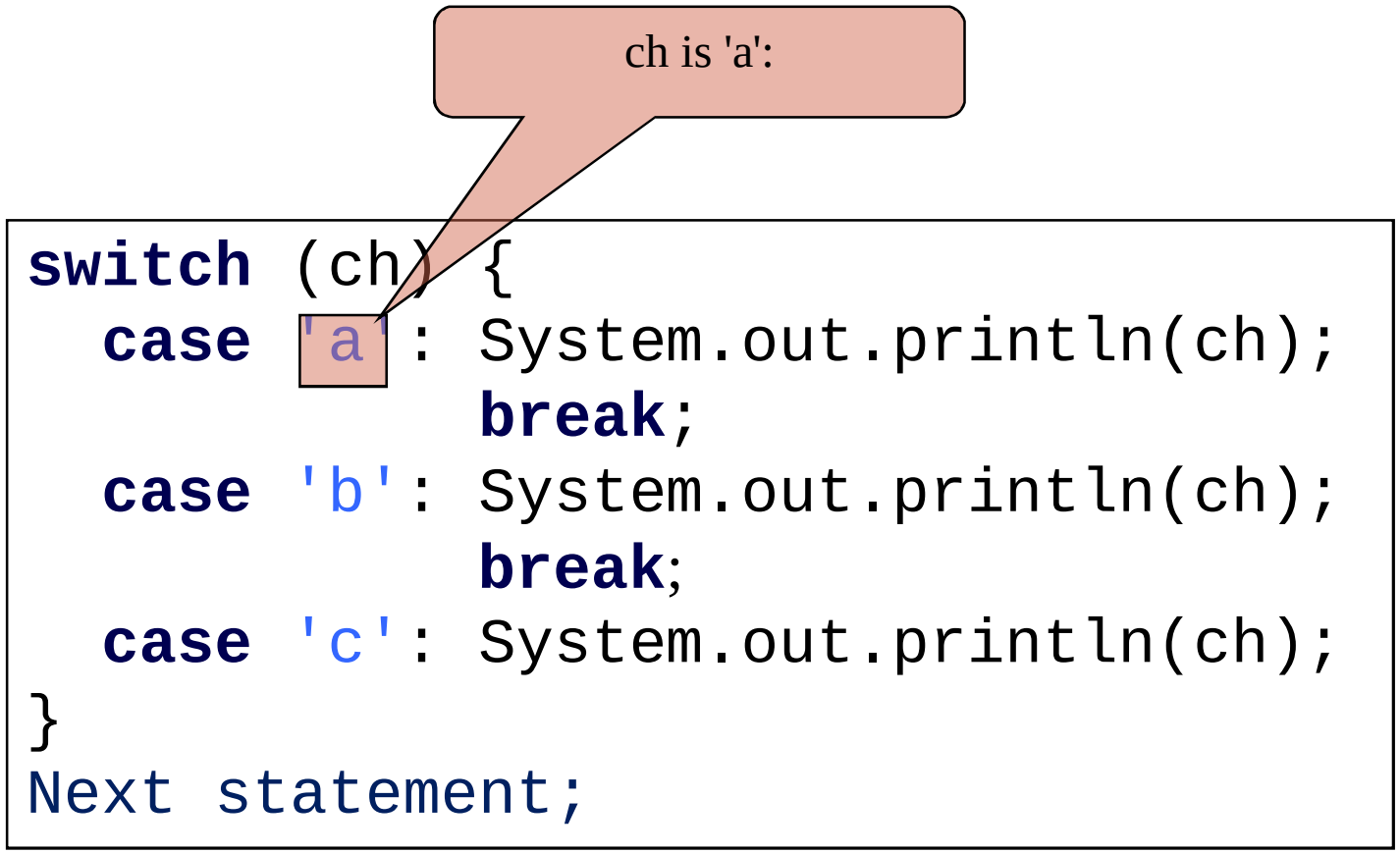
Trace switch statement




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

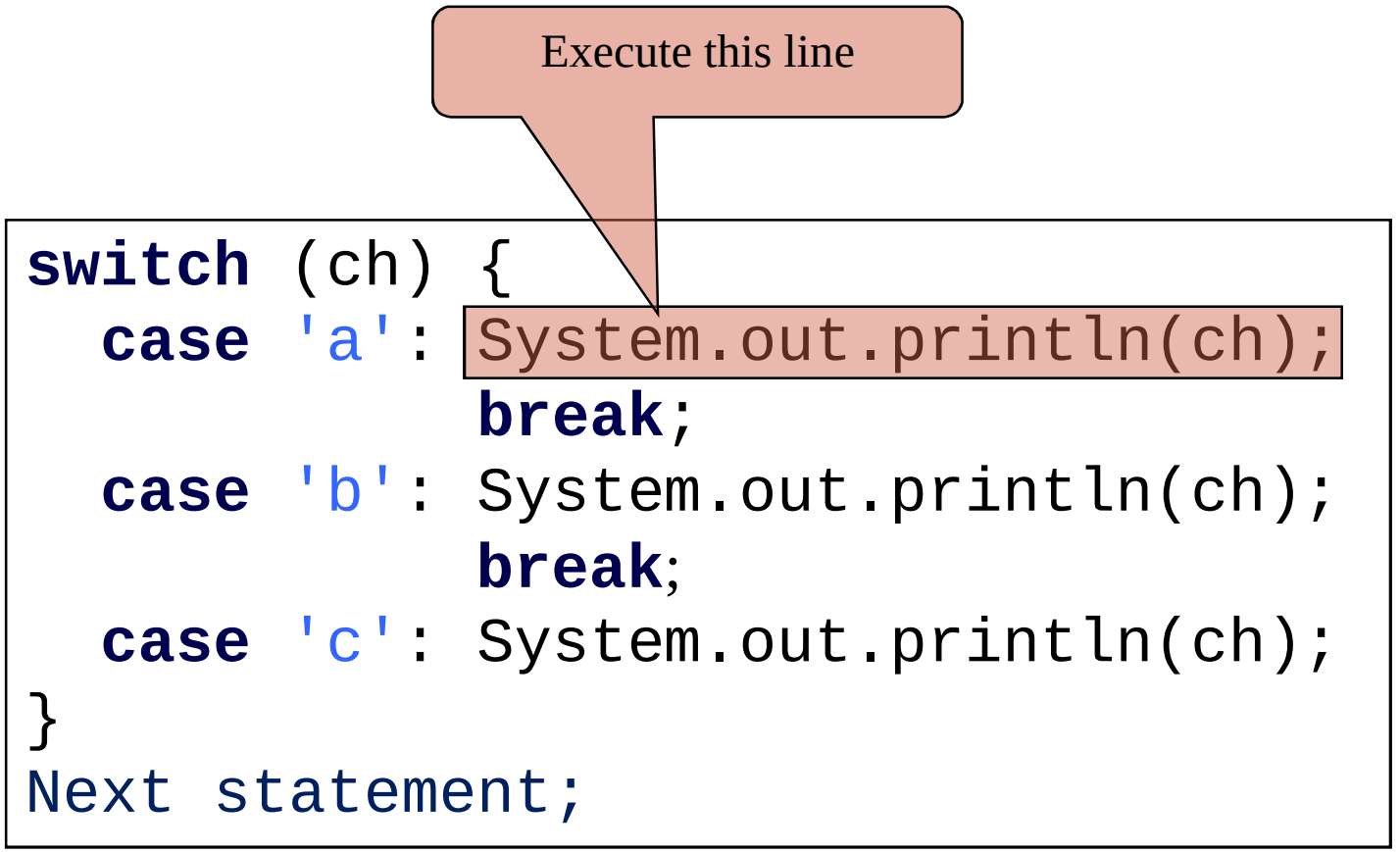
animation

Trace switch statement



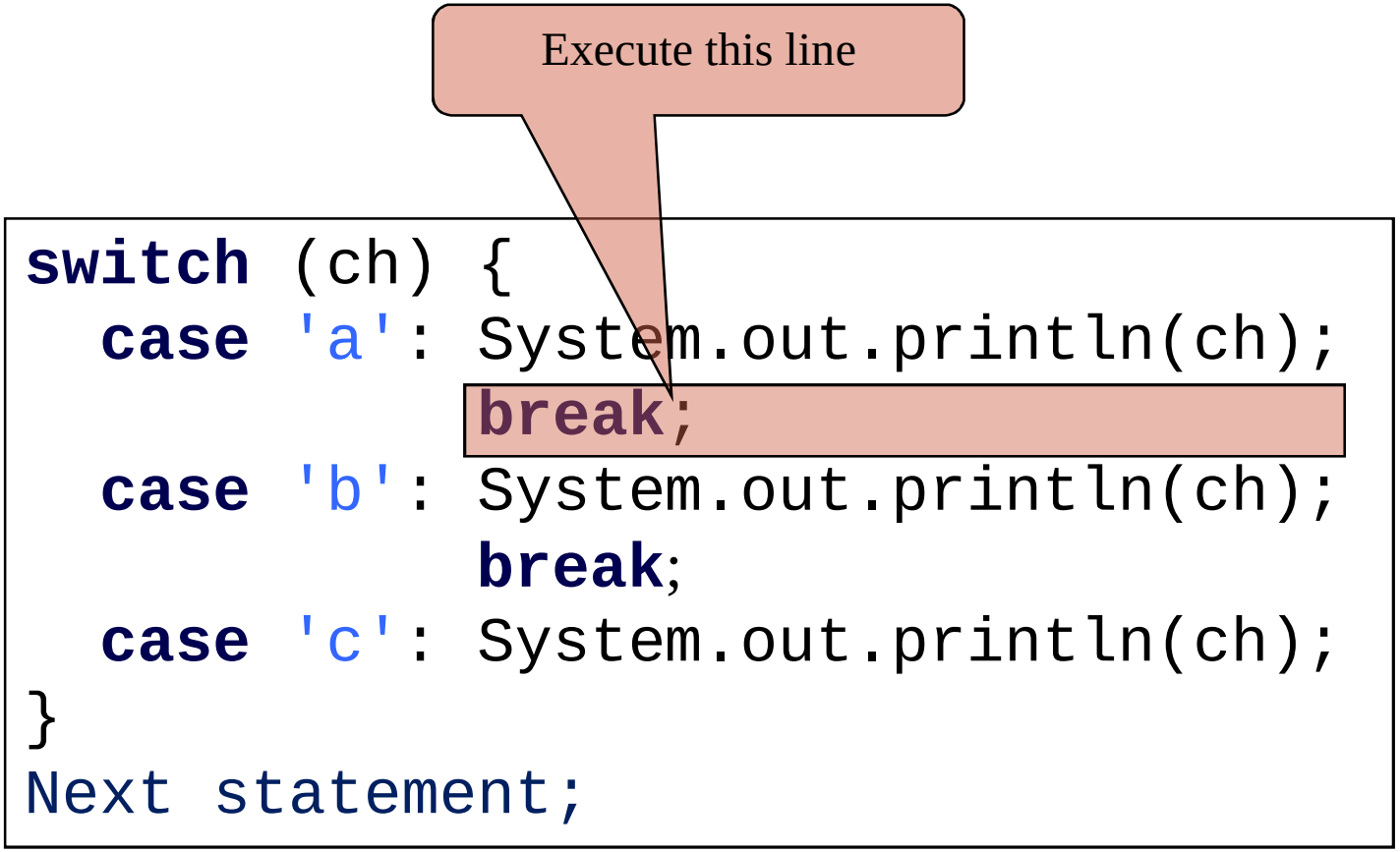
animation

Trace switch statement



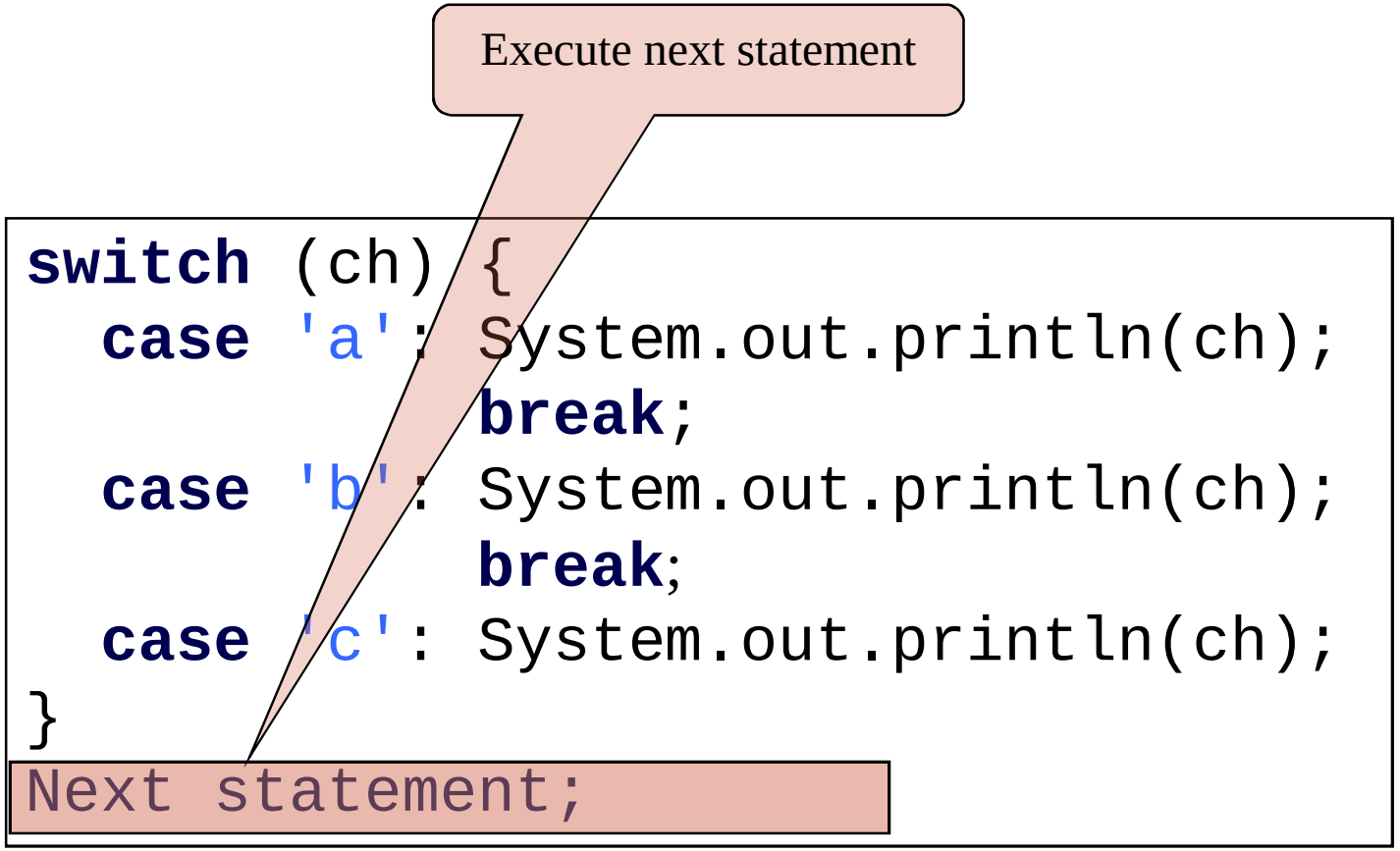
animation

Trace switch statement



animation

Trace switch statement



Conditional Operator

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

is equivalent to

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

(booleanExpression) ? expression1 : expression2

Conditional Operator Example

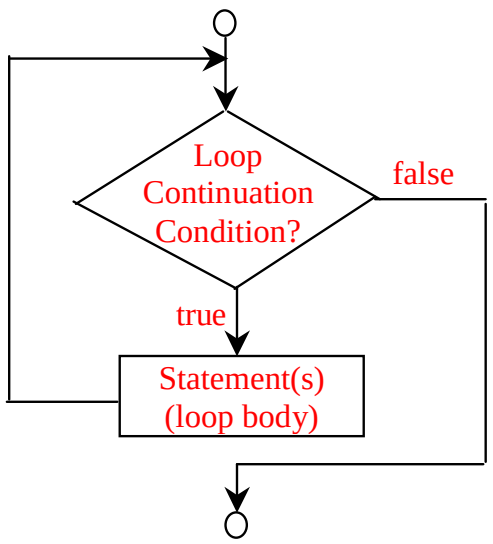
```
if (num % 2 == 0)
  System.out.println(num + “is even”);
else
  System.out.println(num + “is odd”);


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

Loop Statements

- while Statements
- do-while Statements
- for loop statements

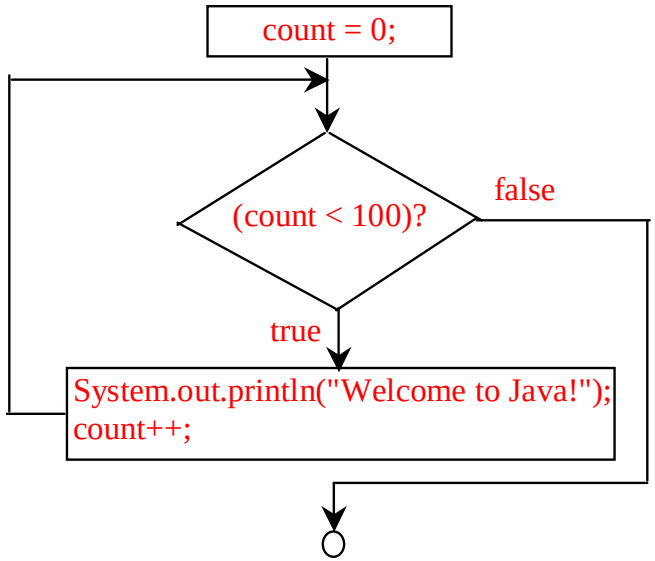
`while` Loop Flow Chart

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



(A)

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



(B)

animation

Trace while Loop

Initialize count

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
Next statement;
```

animation

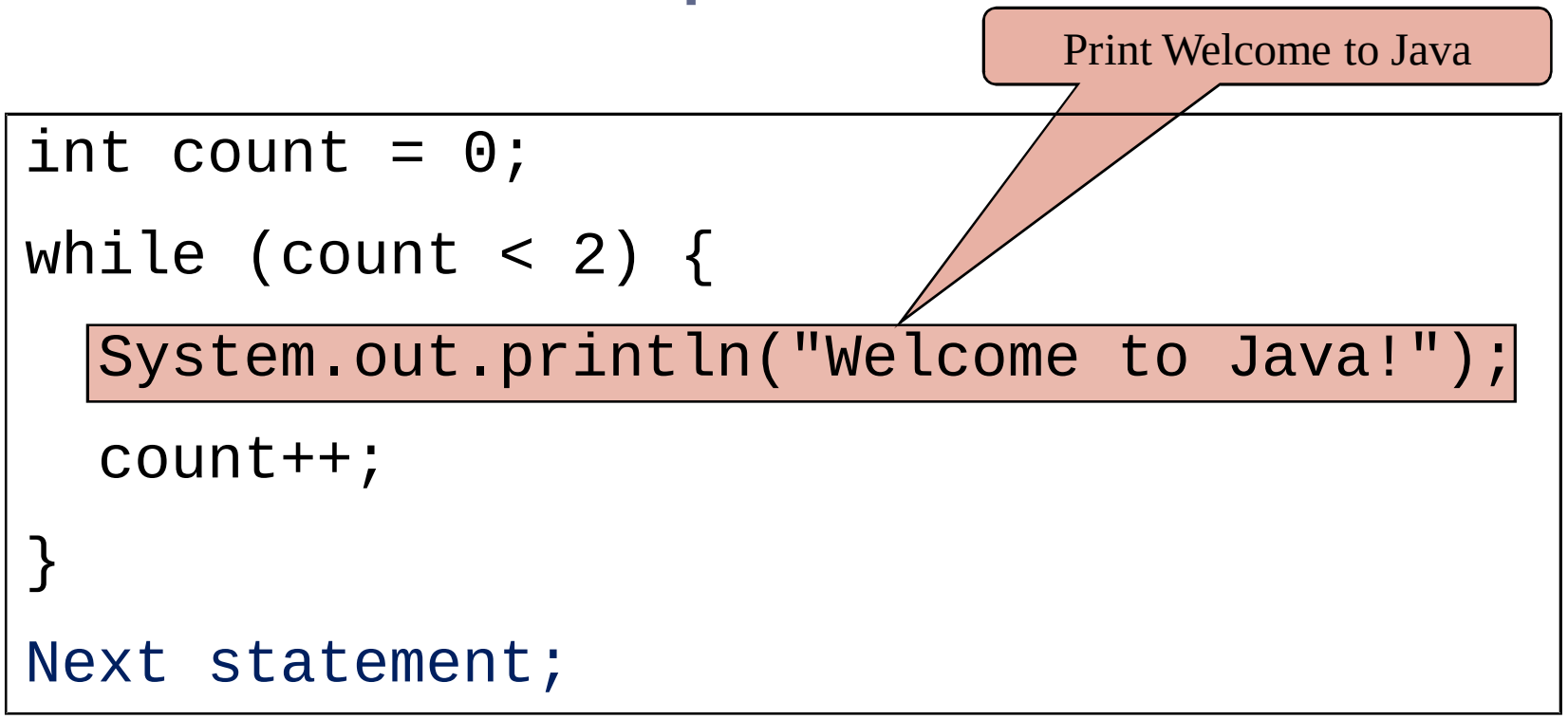
Trace while Loop, cont.

(count < 2) is true

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
Next statement;
```

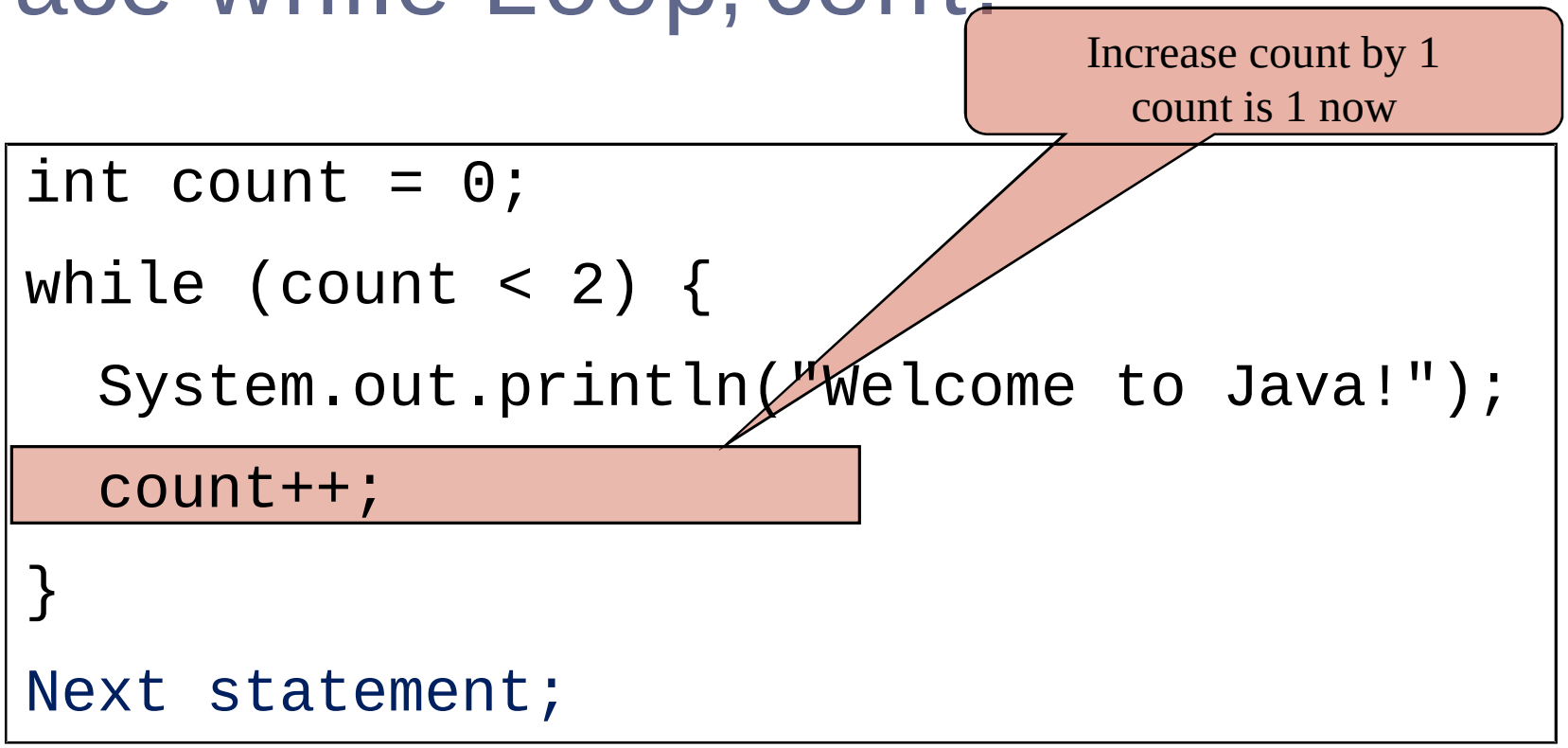
animation

Trace while Loop, cont.



animation

Trace while Loop, cont.



animation

Trace while Loop, cont.

(count < 2) is still true since count is 1

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
Next statement;
```

animation

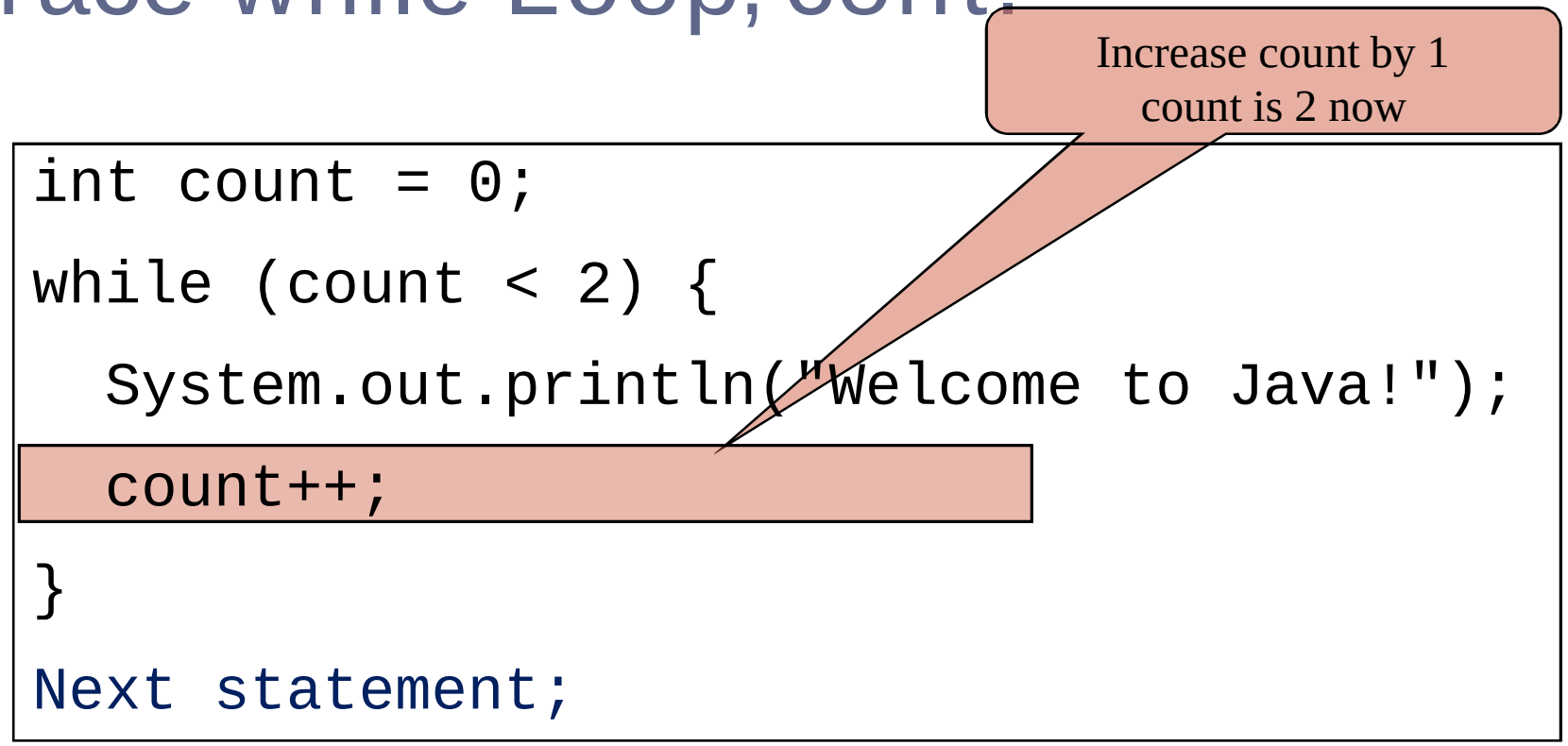
Trace while Loop, cont.

Print Welcome to Java

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
Next statement;
```

animation

Trace while Loop, cont.



animation

Trace while Loop, cont.

(count < 2) is false since count is 2 now

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
Next statement;
```

animation

Trace while Loop

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
Next statement;
```

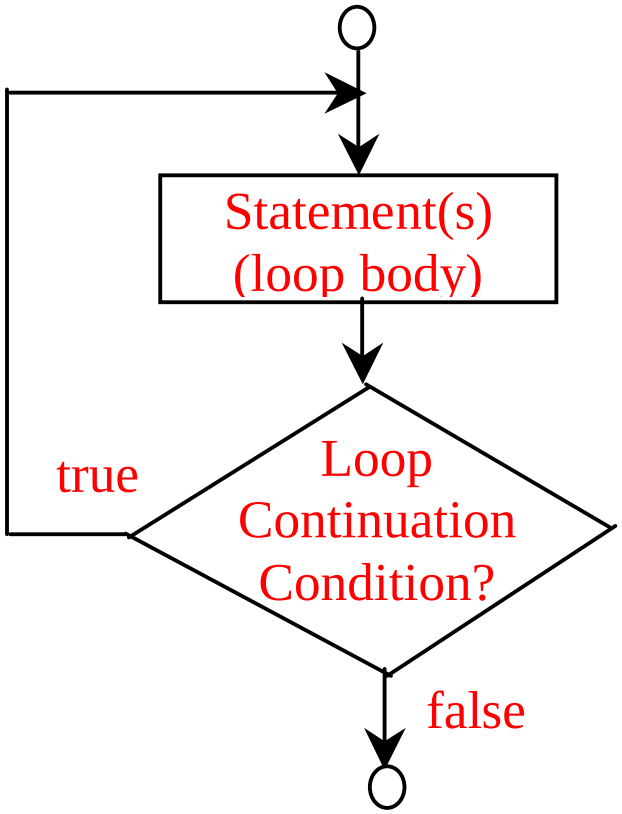
The loop exits. Execute the next statement after the loop.

Caution

- Don't use floating-point values for equality checking in a loop control. Since floating-point values are approximations, using them could result in imprecise counter values and inaccurate results. This example uses int value for data. If a floating-point type value is used for data, (data != 0) may be true even though data is 0.

```
// data should be zero
double data = Math.pow(Math.sqrt(2), 2) - 2;
if (data == 0)
        System.out.println("data is zero");
else
        System.out.println("data is not zero");
```

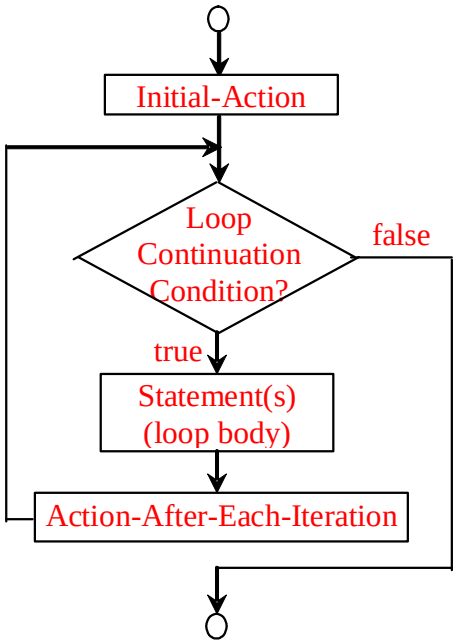
do-while Loop




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

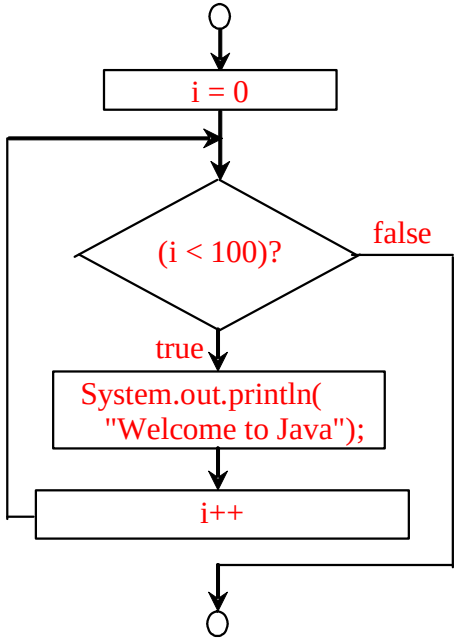
for Loops

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



(A)

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



(B)

animation

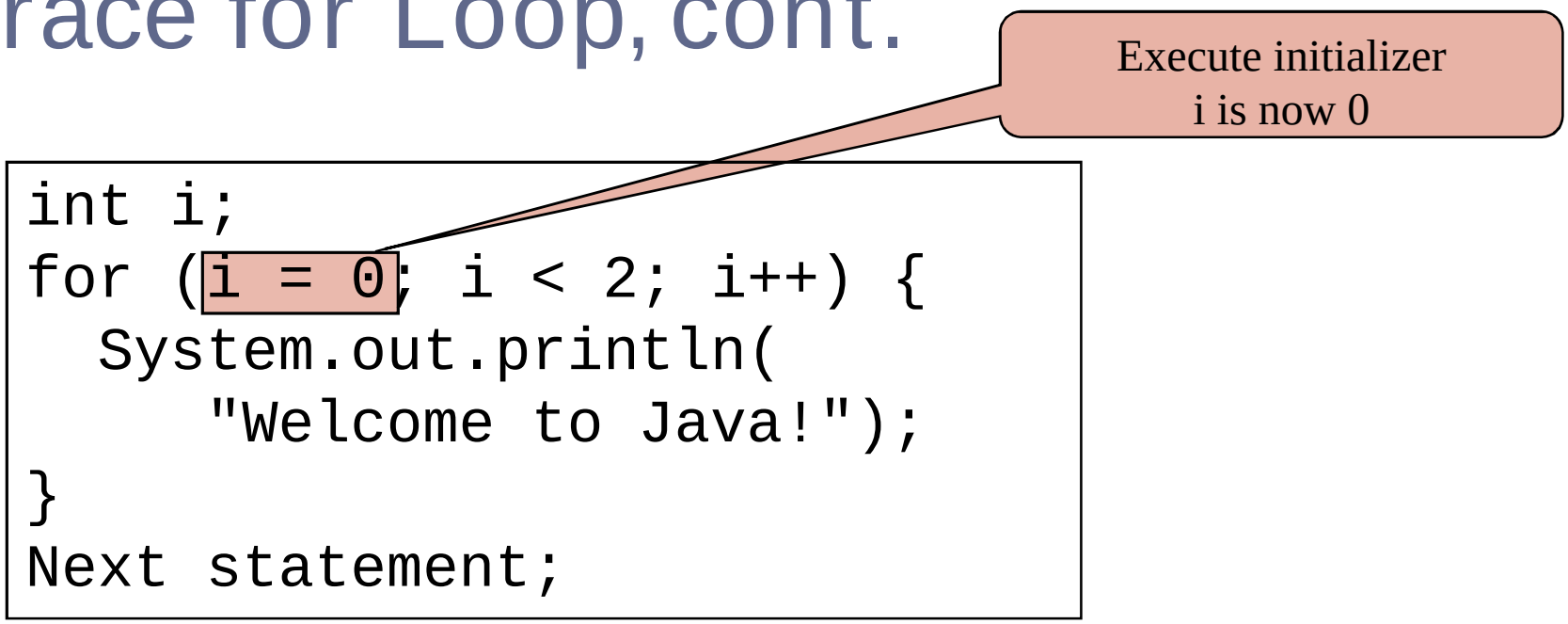
Trace for Loop

Declare i

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println(
     "Welcome to Java!");
}
Next statement;
```

animation

Trace for Loop, cont.



animation

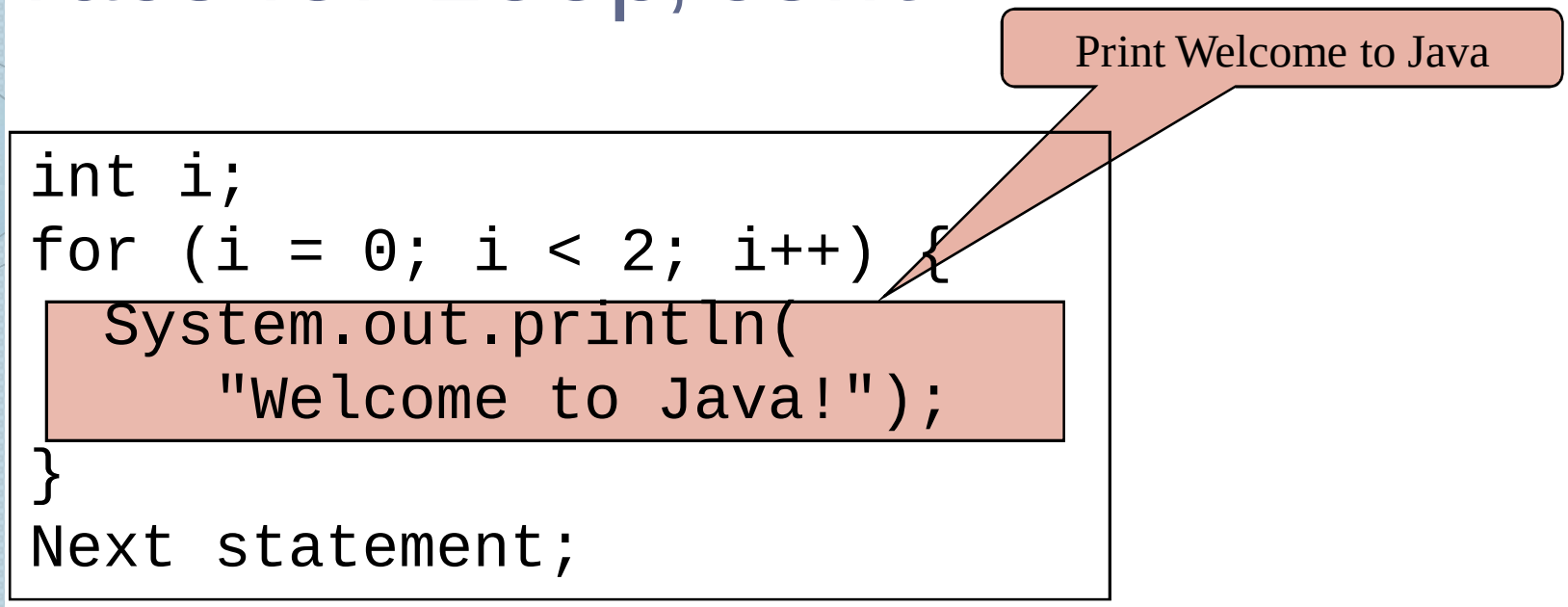
Trace for Loop, cont.

(i < 2) is true
since i is 0

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println(
     "Welcome to Java!");
}
Next statement;
```

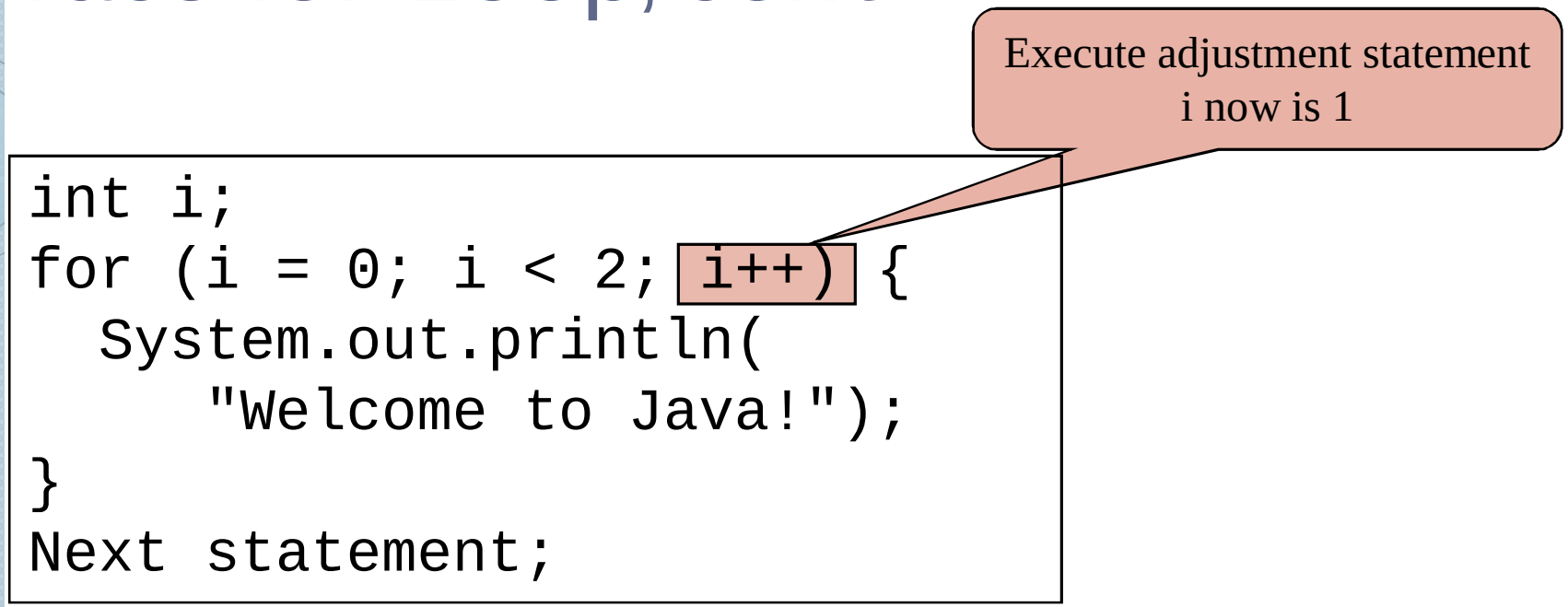
animation

Trace for Loop, cont.



animation

Trace for Loop, cont.



animation

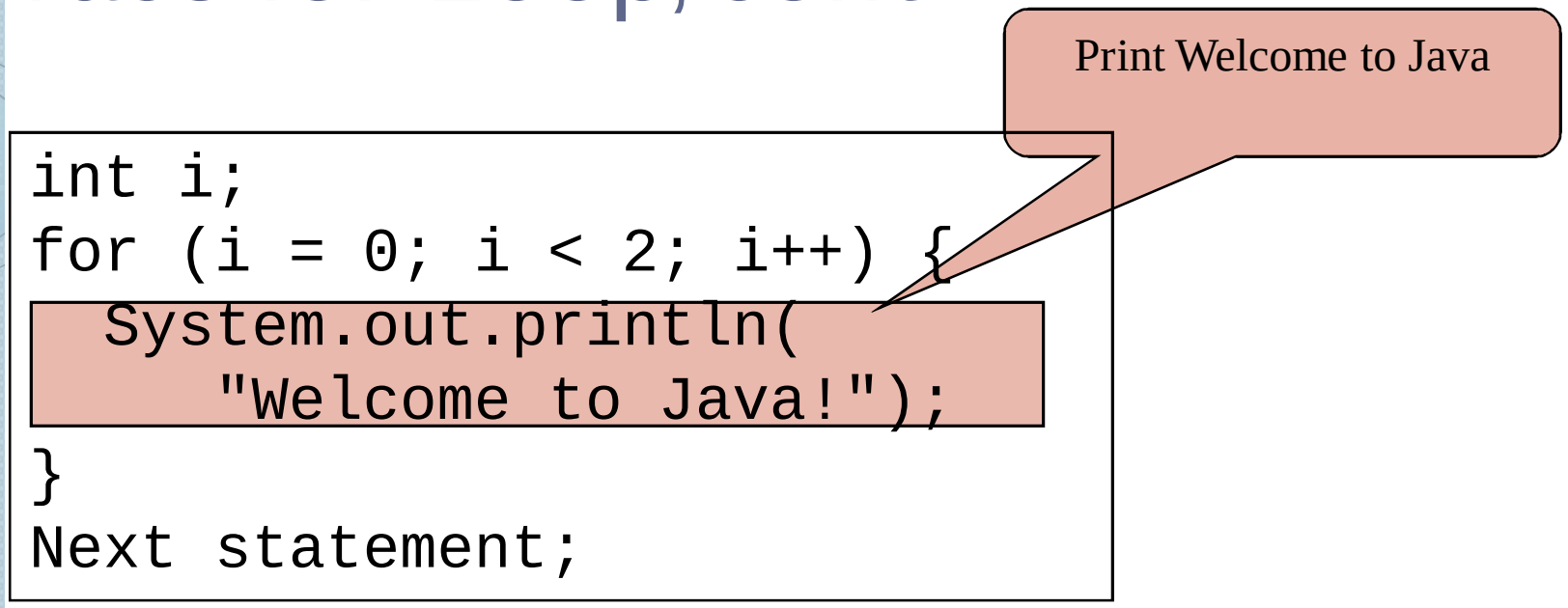
Trace for Loop, cont.

(i < 2) is still true since i is 1

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println(
     "Welcome to Java!");
}
Next statement;
```

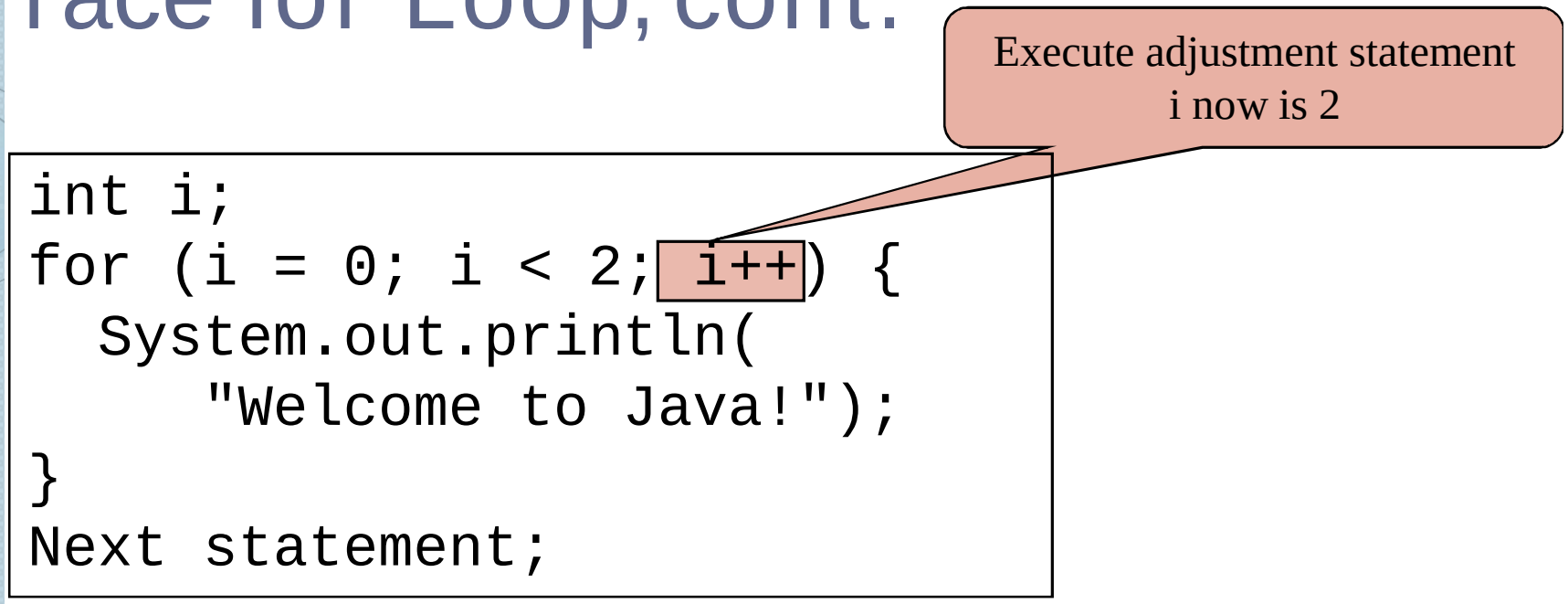
animation

Trace for Loop, cont.



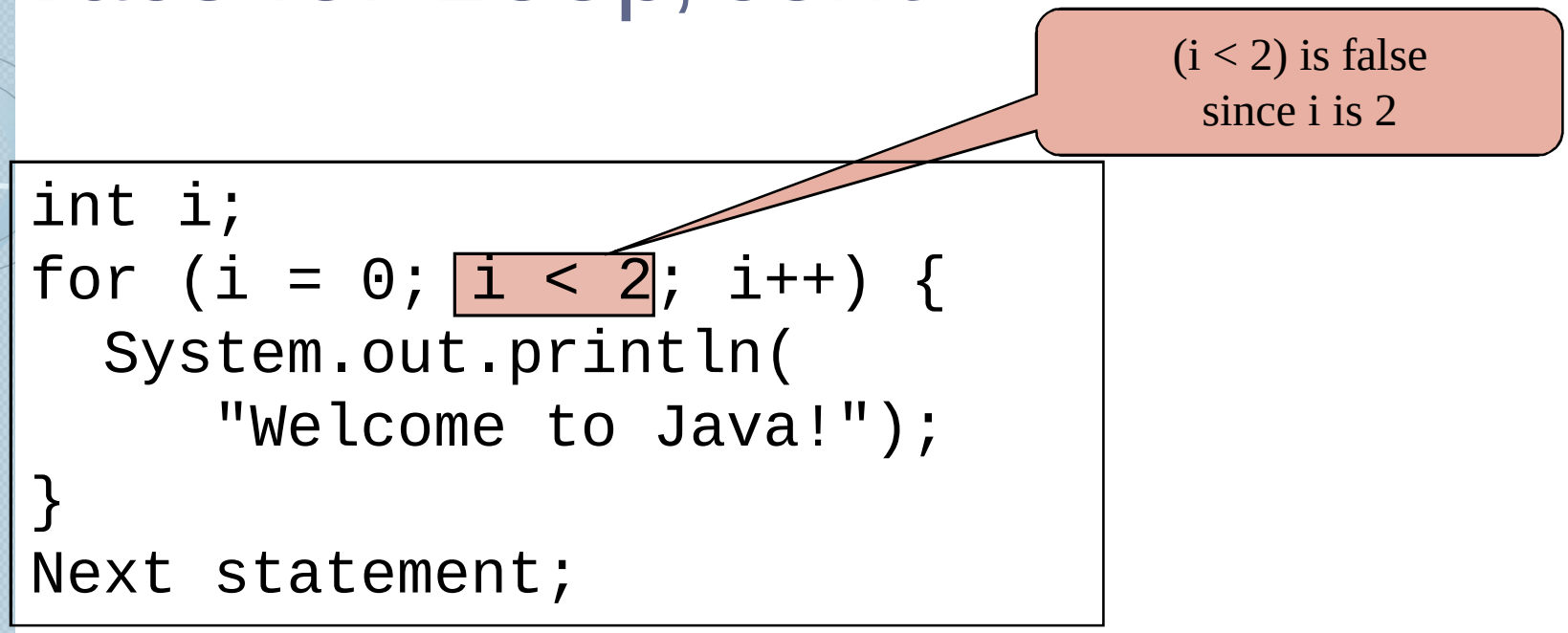
animation

Trace for Loop, cont.



animation

Trace for Loop, cont.



animation

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println(
     "Welcome to Java!");
}
Next statement;
```

Exit the loop. Execute the next statement after the loop

Note

- The initial-action in a for loop can be a list of zero or more comma-separated expressions. The action-after-each-iteration in a for loop can be a list of zero or more comma-separated statements. Therefore, the following two for loops are correct. They are rarely used in practice, however.

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

Example 2:
for (int i = 0, j = 0; (i + j < 10); i++, j++) {
  // Do something
}
```

Note (Cont.)

- If the loop-continuation-condition in a for loop is omitted, it is implicitly true. Thus the statement given below in (a), which is an infinite loop, is correct. Nevertheless, it is better to use the equivalent loop in (b) to avoid confusion:

```
for ( ; ; ) {
  // Do something
}
```

(a)

Equivalent

```
while (true) {
  // Do something
}
```

(b)

Which Loop to Use?

- The three forms of loop statements, while, do-while, and for, are expressively equivalent; that is, you can write a loop in any of these three forms. For example, a while loop in (a) in the following figure can always be converted into the following for loop in (b):

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

(a)

Equivalent

```
for ( ; loop-continuation-condition; )
  // Loop body
}
```

(b)

- A for loop in (a) in the following figure can generally be converted into the following while loop in (b) except in certain special cases:

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

(a)

Equivalent

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

(b)

Recommendations

- Use the one that is most intuitive and comfortable for you.
- In general, a for loop may be used if the number of repetitions is known, for example, when you need to print a message 100 times.
- A while loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0.
- A do-while loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.

Guess A Number Case Study:

Write a program which randomly chooses an integer from 1 to 100. The program should then tell the user: "I am thinking of a number from 1 to 100 ... guess what it is ?". The user should then enter an integer as a guess.

If the guess is above the randomly-chosen number, then the program should tell the user: "lower!" and then wait for another guess. If the guess was below the randomly-chosen number, then the program should tell the user: "higher!" and then wait for another guess. It should repeat this until the user enters the correct number. Then it should print out "Congratulations. You guessed the number with X tries!" where X is the number of guesses that the user made.

The program should then quit. You may assume that the user always enters an integer.

References

- Carleton University COMP 1005 and 1006:
  http://www.scs.carleton.ca/~lanthier/teaching/COMP1405/Notes/
  http://www.scs.carleton.ca/~lanthier/teaching/COMP1406/Notes/
- Armstrong Atlantic State University
  http://www.cs.armstrong.edu/liang/intro10e/
- Oracle Java tutorial:
  http://docs.oracle.com/javase/tutorial/
- MIT opencourseware
  https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-092-introduction-to-programming-in-java-january-iap-2010/