Control Flow Part II

CS 18000
Prof. Sunil Prabhakar
Department of Computer Science
Purdue University

Increment and Decrement

- The increment (++) and decrement (--) operators can precede the operand
  - x++; ++x; y--; --y;
- Their effect on the operand is the same, however, they vary only in terms of the timing of the increment or decrement.
- The postfix operators are applied AFTER the variable's value is used.
- The prefix operator are applied BEFORE

Example

```
int x=2, y=10;

x = y++;
System.out.println("X is:" + x);
System.out.println("Y is:" + y);
```

```
X is: 10
Y is: 11
```

```
int x=2, y=10;

x = y--;
System.out.println("X is:" + x);
System.out.println("Y is:" + y);
```

```
X is: 10
Y is: 9
```

```
int x=2, y=10;

x = ++y;
System.out.println("X is:" + x);
System.out.println("Y is:" + y);
```

```
X is: 11
Y is: 11
```

```
int x=2, y=10;

x = --y;
System.out.println("X is:" + x);
System.out.println("Y is:" + y);
```

```
X is: 9
Y is: 9
```

```
int x=2, y=10, z;

z = x++ * --y;
System.out.println("X is:" + x);
System.out.println("Y is:" + y);
System.out.println("Z is:" + z);
```

```
X is: 3
Y is: 9
Z is: 18
```

```
int x=2, y=10;

x = --x * ++y;
System.out.println("X is:" + x);
System.out.println("Y is:" + y);
```

```
X is: 11
Y is: 11
```

Operator Precedence Rules

Group	Operator	Order
Subexpresion	( )	**Innermost first**
Postfix increment and decrement	++, --	**Right to Left**
Unary operators Prefix increment and decrement	++, --, - , !	**Right to Left**
Multiplicative	*, /, %	Left to Right
Additive	+ , -	Left to Right
Relational	<, <=, >, <,	Left to Right
Equality	!=, ==	Left to Right
Boolean AND	&&	Left to Right
Boolean OR	II	Left to Right
Assignment	=	**Right to Left**

Precedence Examples

```
int x= 1, y=10, z=100;
boolean bool, test=false;
```

- `x = -y + y * z;`
  `x = (-y) + (y*z);`
- `x ==1 && y > 5`
  `(x ==1) && (y > 5)`
- `4 < x && !test`
  `(4<x) && (!test)`
- `bool = x!=y && y == z`
  `bool = (x!=y) && (y ==z)`
- `x==y || y>4 && z<2`
  `(x==y) || ((y>4) && (z<2))`

The char Data Type

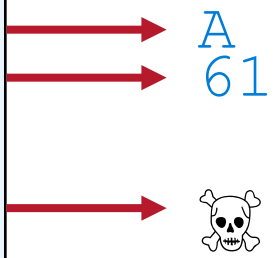
Each character of a string is an instance of a primitive type called **char**.

- In Java, a **char** variable is stored using two bytes
  - each character is encoded using an international standard called UNICODE
  - character literal are written with single quotes, e.g., 'c' 'x' 'δ' 'स' 'س' 'ש' '大' '' ''
  - some languages may use ASCII -- an older subset of UNICODE (1 byte per char).

Unicode Encoding

- Extended version of ASCII to accommodate world languages and common symbols.
  - Each character mapped to a code (2 bytes)
  - Often written in hexadecimal e.g., '\u1234'
  - Can convert between **int** and **char** types

```
char ch = 'A';
int code = '\u2620';

System.out.println(ch);
System.out.println( (int) ch);
ch = (char)code;
System.out.println(ch);
```



Characters and Relational Operators

- We can compare characters with each other or with numeric

```
char ch1='x', ch2=64, ch3='\u00a9';
int i,j;

if( ch1 == 'X' || ch2 == 99){
   …
}

if(ch2 < i && ch3 < ch2){
   …
}
System.out.println(ch1 +  ch2 + ch3);
System.out.println("" + ch1 +  ch2 + ch3);
```

→ 353

→ x@©

Strings and Characters

- We can get the character at a given index of a string using charAt()

```
char ch;
String s = "Go Purdue!!!";

ch = s.charAt(4);
System.out.println("The Character at index 4 is:" + ch);
```

- We can combine characters into a string

```
char ch1 = 'C', ch2 = 111, ch3 = '\u006c', ch4='\u0089';
String s;
s = "" + ch1 + ch2 + ch2 + ch3 + ch4;
System.out.println(s);
```

The switch statement

Earlier, we studied the **if** statement for choosing between two alternative statements (execution paths).

This week we will study another selection statement called the **switch** statement.

- Consider the problem of converting a letter grade into the corresponding grade points:
- 'A' to 4, 'B' to 3, ...,
- we could use a nested **if** statement

Converting Grades to Points

```
class StudentV5 {
 . . .
   letterGrade = JOptionPane.showInputDialog(null, "Enter Grade").charAt(0);
   if (letterGrade == 'A')
      grade = 4;
   else
      if (letterGrade == 'B')
         grade = 3;
      else
         if (letterGrade == 'C')
            grade = 2;
         else
            if (letterGrade == 'D')
               grade = 1;
            else
               grade = 0;
    . . .
```

Often written as

```
class StudentV5 {
 . . .

 public void recordGrade(){
    char letterGrade;
     . . .
    letterGrade = JOptionPane.showInputDialog(null, "Enter Grade").charAt(0);
    if (letterGrade == 'A')
        grade = 4;
    else if (letterGrade == 'B')
        grade = 3;
    else if (letterGrade == 'C')
        grade = 2;
    else if (letterGrade == 'D')
        grade = 1;
    else
        grade = 0;
     . . .
```

The switch statement

Notice the use of **else if** constructs in order to choose between multiple alternatives.

Such a situation, where only one of the various statements is executed depending upon a given value is quite common.

The **switch** statement is used to achieve achieve the same result while improving the reliability and readability of code.

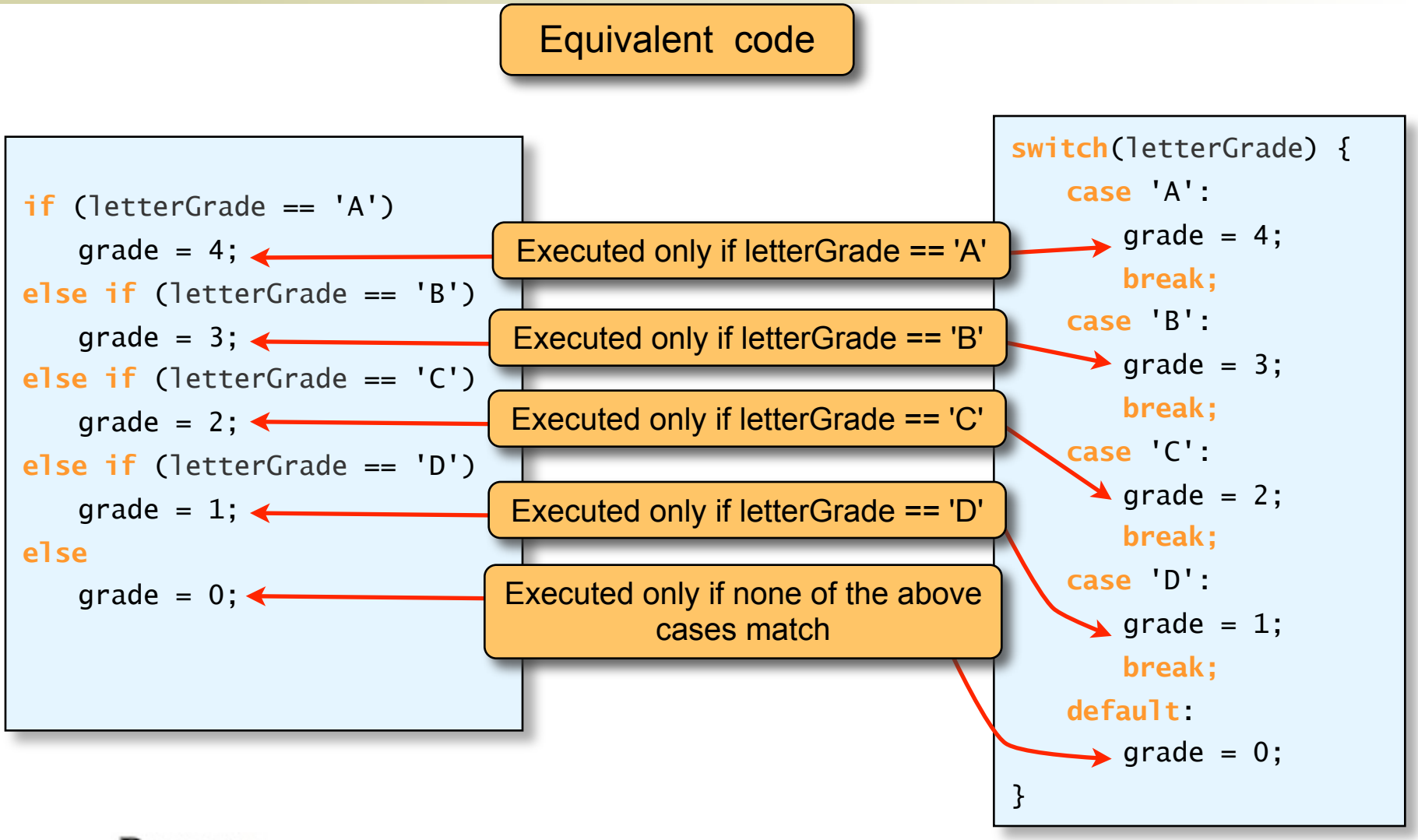
Using a switch statement

```
if (letterGrade == 'A')
    grade = 4;
else if (letterGrade == 'B')
    grade = 3;
else if (letterGrade == 'C')
    grade = 2;
else if (letterGrade == 'D')
    grade = 1;
else
    grade = 0;
```

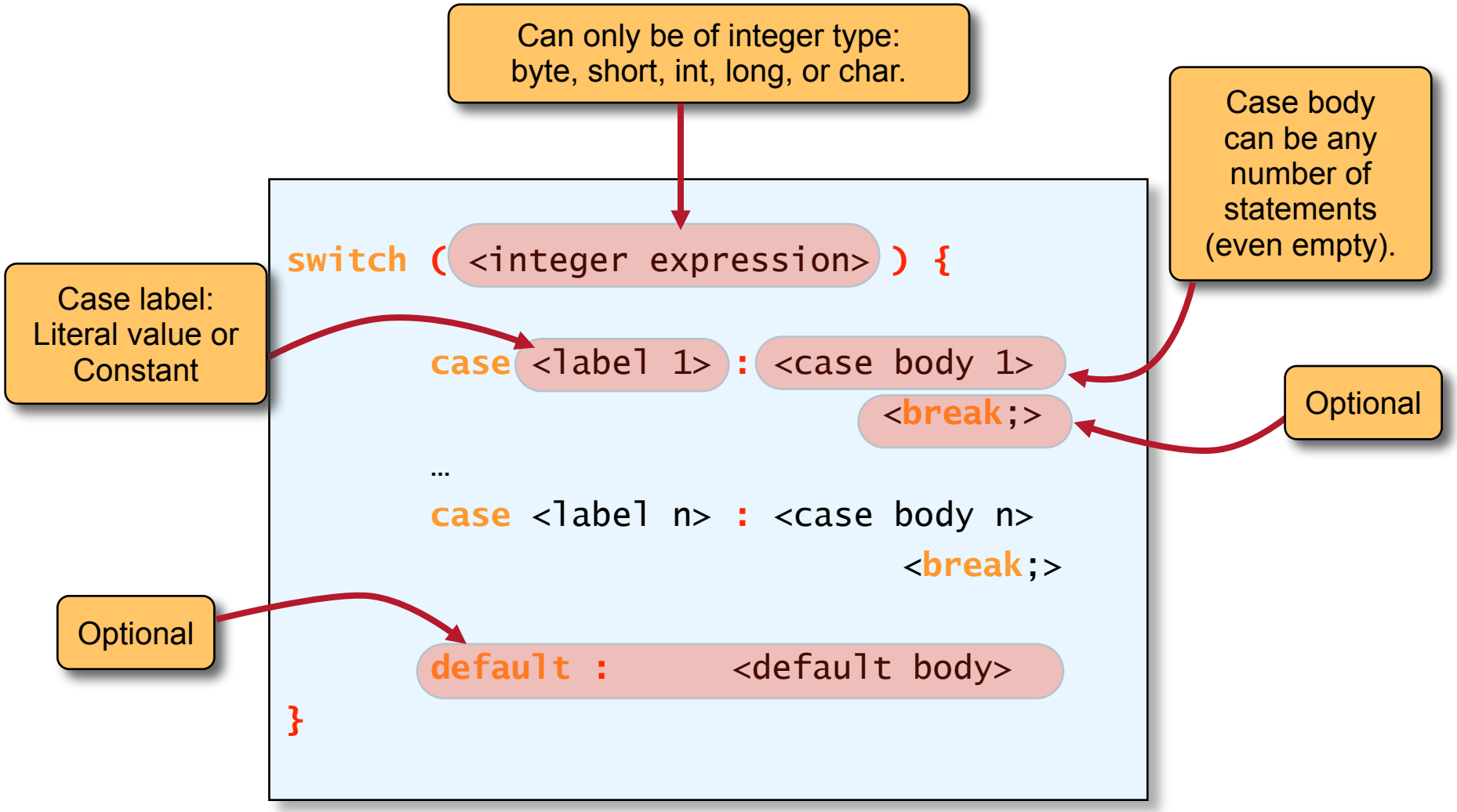
Equivalent code

```
switch(letterGrade) {
    case 'A':
        grade = 4;
        break;
    case 'B':
        grade = 3;
        break;
    case 'C':
        grade = 2;
        break;
    case 'D':
        grade = 1;
        break;
    default:
        grade = 0;
}
```

Using a switch statement



Syntax for the switch Statement

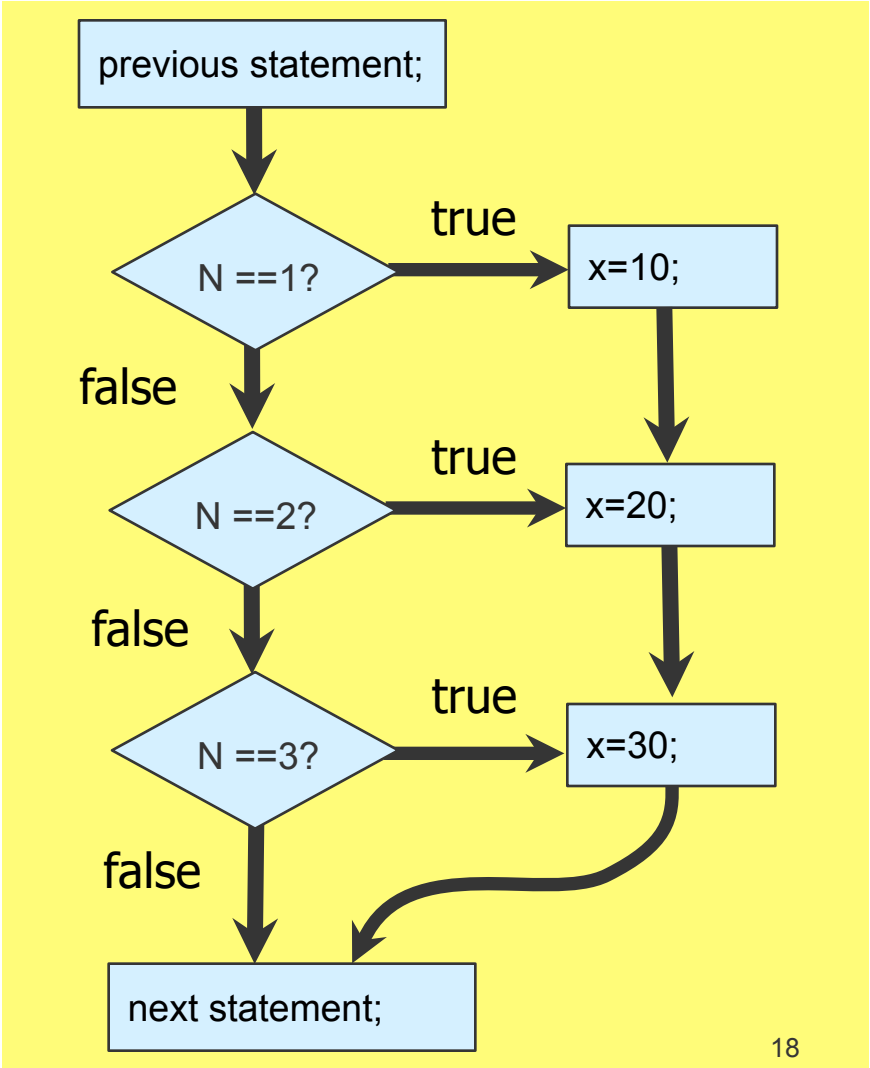


Switch statement (cont.)

- The integer expression can have only one of the following types:
  - **char**, **byte**, **short**, or **int** (and enum types)
  - Java 7 allows Strings too.
- The label must be a literal or named constant of the same type as the integer expression
  - Each label must be unique.
  - Labels may be listed in any order.
  - The **default** case applies when no label matches.
- A **break** causes execution to break out of the switch statement to the next statement.
  - each **break** is optional

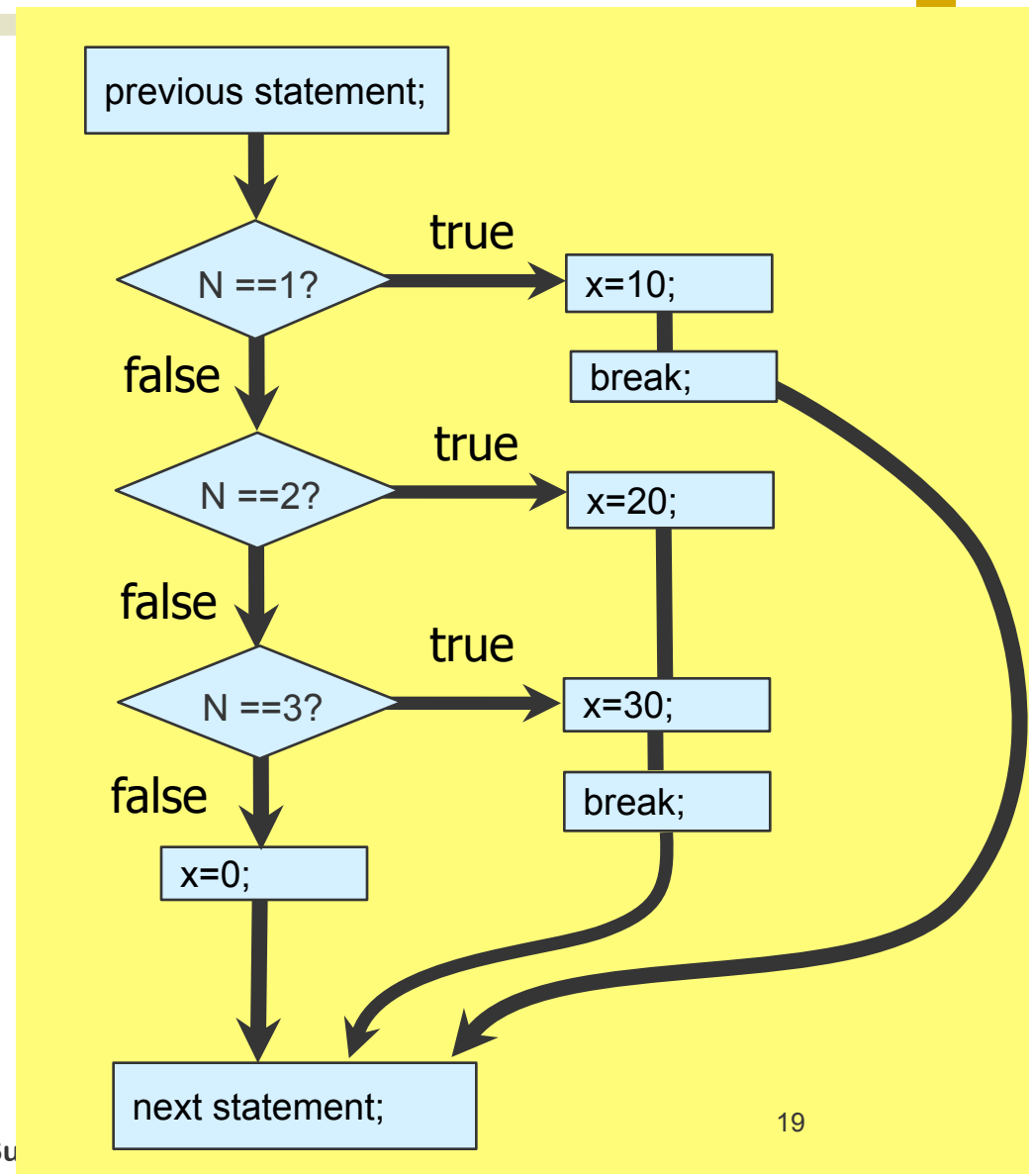
Simple switch statement

```
switch ( N ) {
    case 1: x = 10;
    case 2: x = 20;
    case 3: x = 30;
}
```



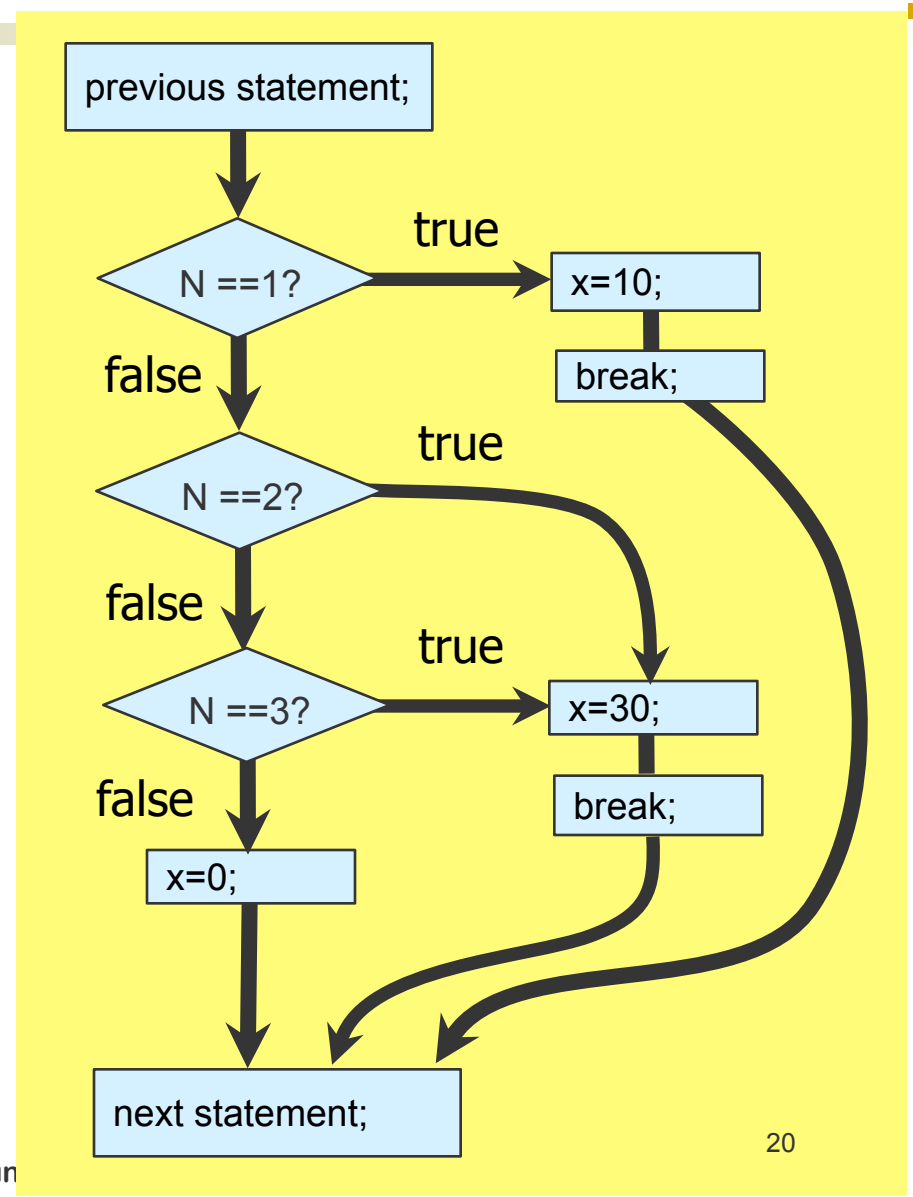
switch with break, and default

```
switch ( N ) {
    case 1: x = 10;
            break;
    case 2: x = 20;
    case 3: x = 30;
            break;
    default: x = 0;
}
```



Missing case body

```
switch ( N ) {
    case 1: x = 10;
            break;
    case 2:
    case 3: x = 30;
            break;
    default: x = 0;
}
```



DaysInMonth

```
int month, daysInMonth;
boolean isLeapYear;
 . . . // set month (1 - 12) and leapYear appropriately
 switch (month) {
   case 2:
          if(isLeapYear)
              daysInMonth = 29
           else
              daysInMonth = 28;
           break;
   case 3:
   case 5:
   case 7:
   case 8:
   case 10:
   case 12:
          daysInMonth = 31;
          break;
   default :
          daysInMonth = 30;
}
```

Problem

- *Write a program that prints out a multiplication table for a given number input by the user.*
  - *We will limit our tables to multiples up to 12.*
  - *The user input should be between 2 and 12.*

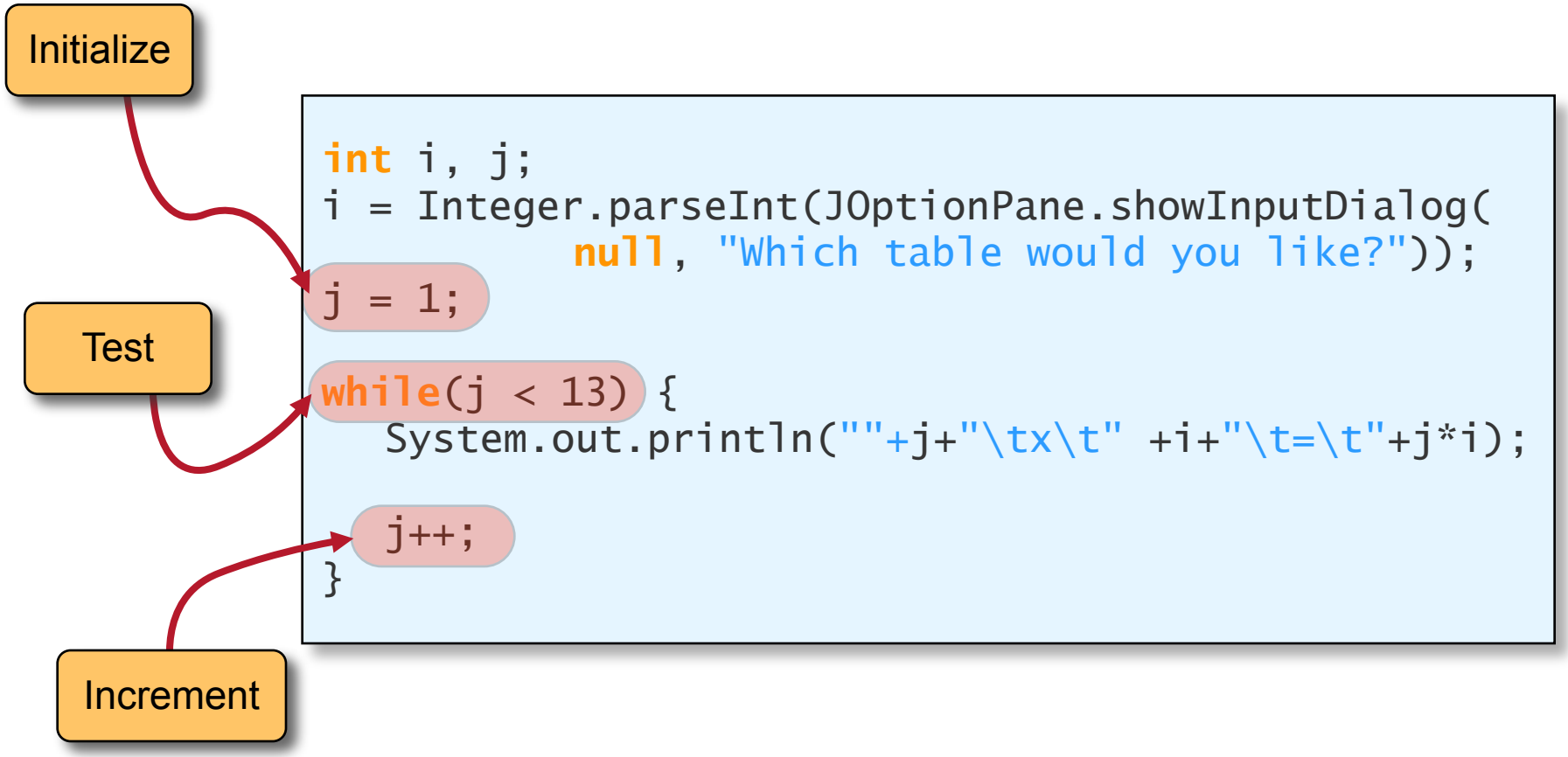
PrintOneTable

```
public class PrintOneTable {
   public static void main(String[] args){

      int i;

      i = Integer.parseInt(JOptionPane.showInputDialog(
                null, "Which table would you like?"));
      System.out.println("1\tx\t" + i + "\t=\t" + 1*i);
      System.out.println("2\tx\t" + i + "\t=\t" + 2*i);
      System.out.println("3\tx\t" + i + "\t=\t" + 3*i);
      . . .
      System.out.println("12\tx\t" + i + "\t=\t" + 12*i);

   }
}
```

Issues

- This is not very convenient.
- What if we wanted to print the table up to multiples of 1000?
  - we would have to add 1000 print statements to our code!
- What if we wanted to change the range of multiples?
- Could use a **while** loop instead.

PrintOneTable

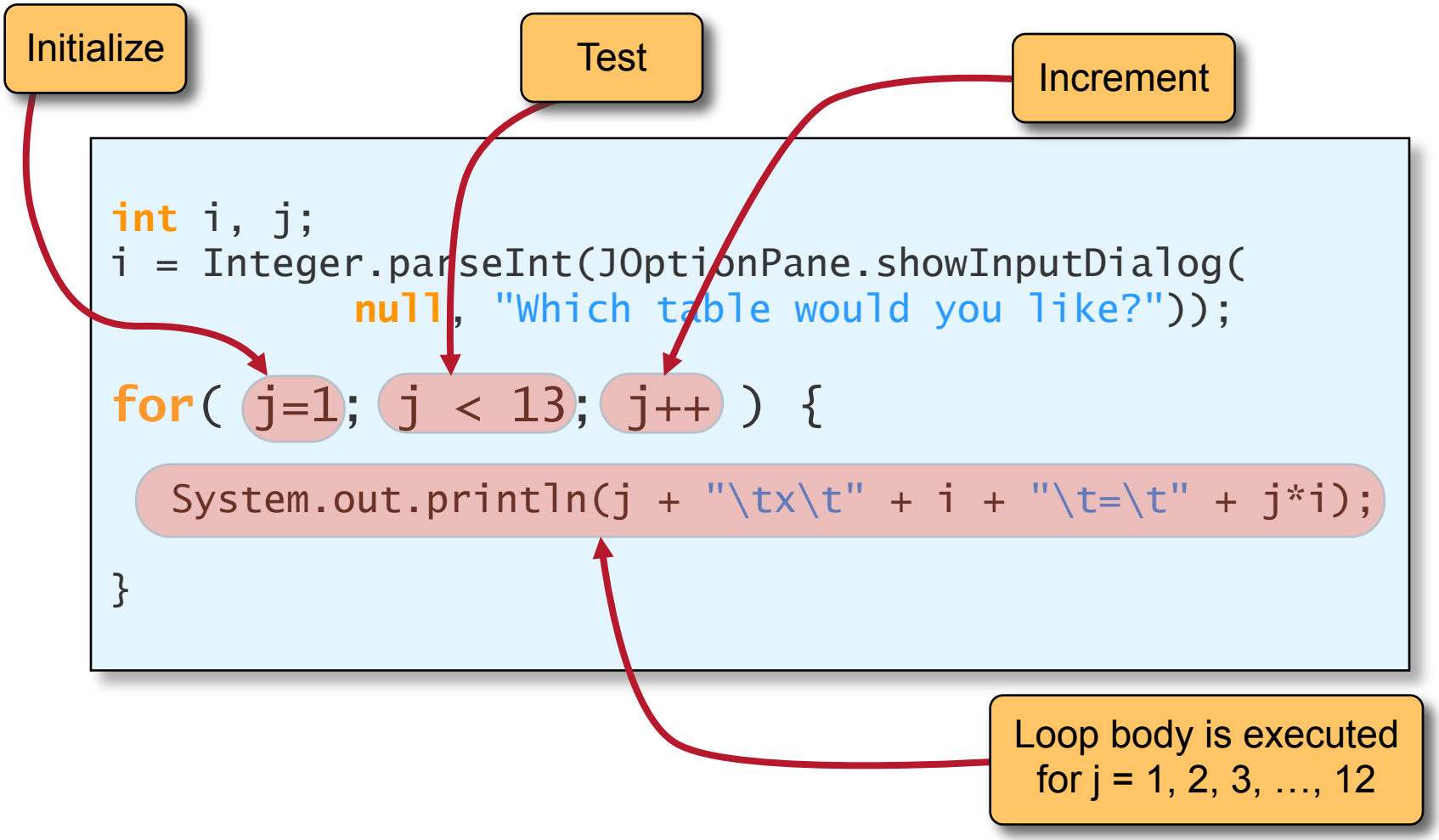


Initialize-Test-Increment

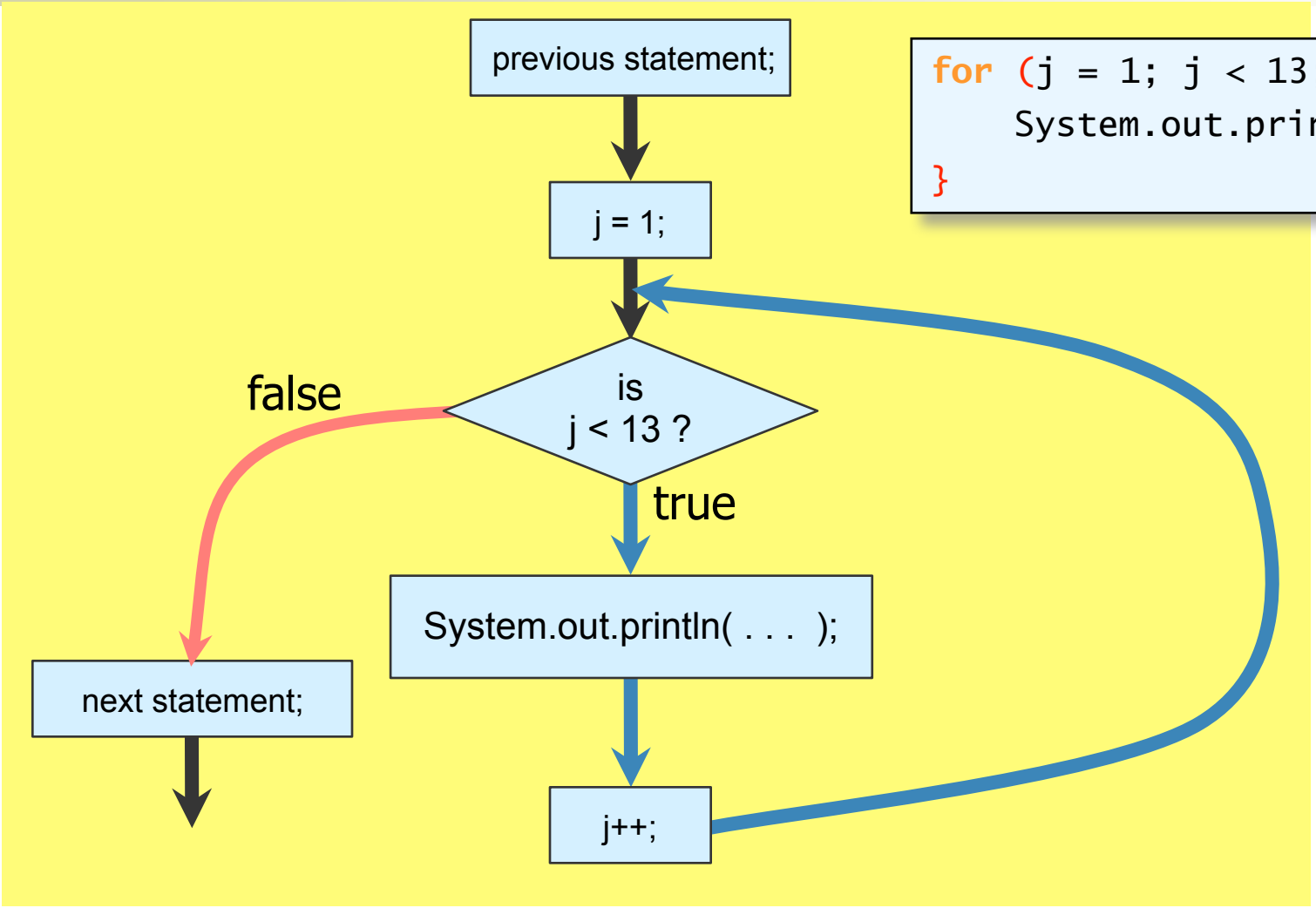
- This is a very common situation:
  - initialize a variable
  - repeat a loop body until some condition is true
  - update variable in each loop
- A **for** loop can be used in this situation.
  - makes the three steps explicit
  - separate from the loop body

```
for ( <initialization>; <boolean expression>; <increment>  )
                                <statement>
```

PrintOneTable



Control flow of for




```
for (j = 1; j < 13; j++) {
    System.out.println(…);
}
```

More for Loop Examples

```
int sum = 0;
for (i = 1; i <= 1000; i++)
   sum += i;
```

Sum of the first 1000 integers

```
int product = 0;
for (i=2; i<=100;i+=2)
   product += i;
```

Product of the even number below 100

```
for (j = 2; j < 40; j *= 2)
```

```
for ( int k = 100; k > 0; k--)
```

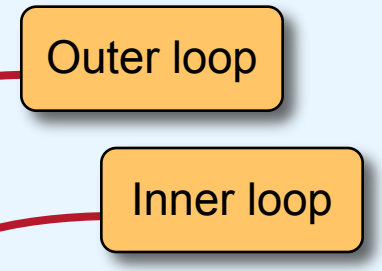
Problem

- *Write a program to print out a multiplication tables from 1 through 12.*

	1	2	3	4	5	6	7	8	9	10	11	12
1	1	2	3	4	5	6	7	8	9	10	11	12
2	2	4	6	8	10	12	14	16	18	20	22	24
3	3	6	9	12	15	18	21	24	27	30	33	36
4	4	8	12	16	20	24	28	32	36	40	44	48
5	5	10	15	20	25	30	35	40	45	50	55	60
6	6	12	18	24	30	36	42	48	54	60	66	72
7	7	14	21	28	35	42	49	56	63	70	77	84
8	8	16	24	32	40	48	56	64	72	80	88	96
9	9	18	27	36	45	54	63	72	81	90	99	108
10	10	20	30	40	50	60	70	80	90	100	110	120
11	11	22	33	44	55	66	77	88	99	110	121	132
12	12	24	36	48	60	72	84	96	108	120	132	144

Generating the Table

```
int row, col;

for(col=1;col<13;col++)
  System.out.print(" " +col);
System.out.print("\n");

for(row=1; row<13; row++){
  System.out.print(""+row);
  for(col=1;col<13;col++)
    System.out.print(" " + row*col);

  System.out.print("\n");
}
```



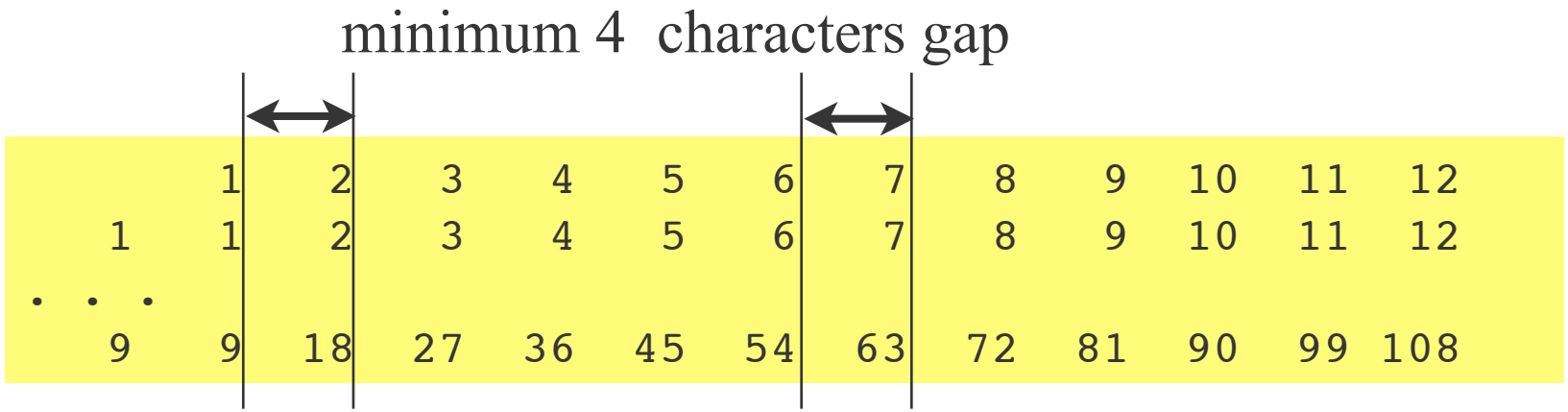
Output

```
   1  2  3  4  5  6  7  8  9  10  11  12
1  1  2  3  4  5  6  7  8  9  10  11  12
2  2  4  6  8  10  12  14  16  18  20  22  24
3  3  6  9  12  15  18  21  24  27  30  33  36
4  4  8  12  16  20  24  28  32  36  40  44  48
5  5  10  15  20  25  30  35  40  45  50  55  60
6  6  12  18  24  30  36  42  48  54  60  66  72
7  7  14  21  28  35  42  49  56  63  70  77  84
8  8  16  24  32  40  48  56  64  72  80  88  96
9  9  18  27  36  45  54  63  72  81  90  99  108
10  10  20  30  40  50  60  70  80  90  100  110  120
11  11  22  33  44  55  66  77  88  99  110  121  132
12  12  24  36  48  60  72  84  96  108  120  132  144
```

Formatting Output using printf

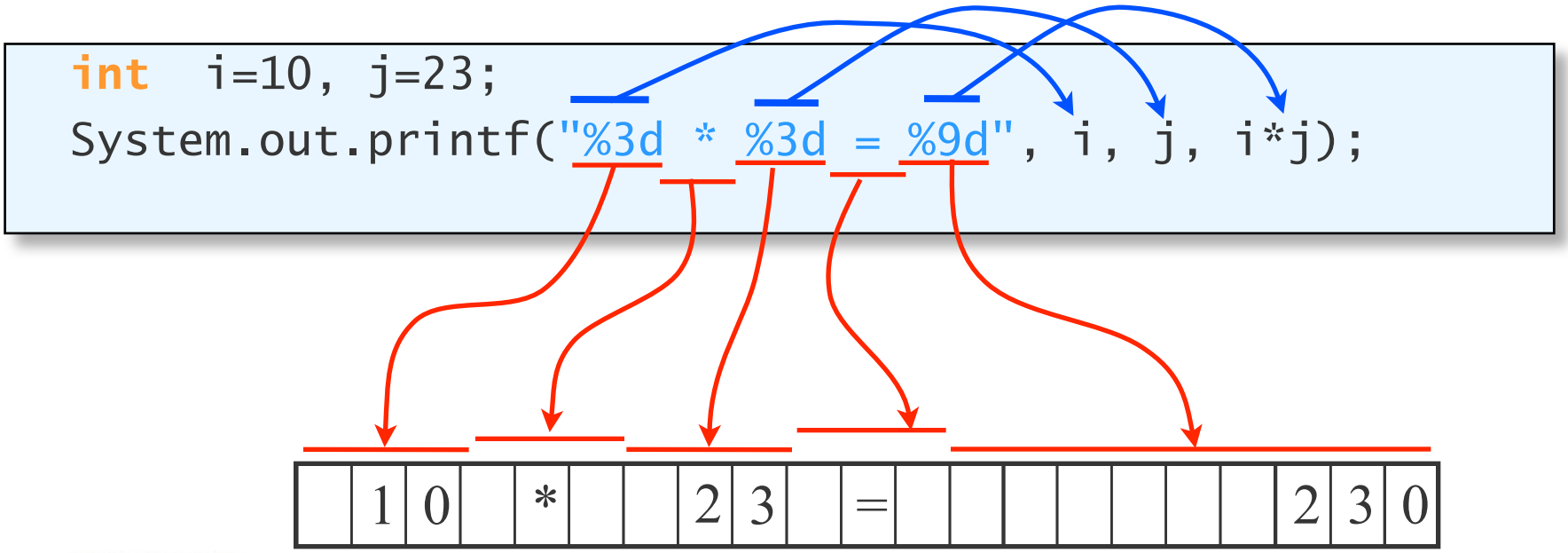
In order to control output, we can use the printf() function.

```
for(row=1; row<13; row++){
  System.out.printf("%4d", row);
  for(col=1;col<13;col++)
     System.out.print("%4d", row*col);
  System.out.print("\n");
}
```



Formatting Output

- The first argument to printf() is the string to be displayed.
- Each occurrence of a format specifier (e.g., %4d) is replaced by a matching argument.



Format Specifiers

- Integer data
  - %<min width>d e.g., %5d
- Real numbers (float and double)
  - %<min width>.<decimal places>f
  - e.g., %3.5f -- use 5 decimal places
- String
  - %s or %10s or %10.3s %-10s
- Character
  - %c or %3c

The format method of PrintStream

- Instead of using the printf() method of PrintStream, we can achieve the same result by using the format method of PrintStream or a Formatter object.

```
System.out.printf ("%6d", 498);
```

is equivalent to

```
Formatter formatter = new Formatter(System.out);
formatter.format("%6d", 498);
```

and equivalent to

```
System.out.format("%6d", 498);
```

See API for details.

breaking out of a loop

- In some cases, it is necessary to get out of a loop.
- This is achieved using a break statement.

```
for(row=1; row<13; row++){
   System.out.print(""+row);
   for(col=1;col<13;col++){
      System.out.print(" " + row*col);
      if(col>=row)
         break;
    }
   System.out.print("\n");
}
```

```
     1   2   3   4   5   6   7   8
 1   1
 2   2   4
 3   3   6   9
 4   4   8  12  16
 5   5  10  15  20  25
 6   6  12  18  24  30  36
 7   7  14  21  28  35  42  49
 8   8  16  24  32  40  48  56  64
 9   9  18  27  36  45  54  63  72  8
10  10  20  30  40  50  60  70  80  9
11  11  22  33  44  55  66  77  88  9
12  12  24  36  48  60  72  84  96 10
```

breaking Out of Outer Loop

```
for(row=1; row<13; row++){
   System.out.print(""+row);
   for(col=1;col<13;col++)
      System.out.print(" " + row*col);
   System.out.print("\n");
   if(col*row>30)
         break;
}
```

	1	2	3	4	5	6	7	8	9	10	11	12
1	1	2	3	4	5	6	7	8	9	10	11	12
2	2	4	6	8	10	12	14	16	18	20	22	24
3	3	6	9	12	15	18	21	24	27	30	33	36

Skipping an iteration

- We can skip the current iteration of a loop using a continue statement.
- A continue transfers control to the test statement of the loop.

```
for(row=1; row<13; row++){
  if(row%2==0)
     continue;
   System.out.printf("%4d", row);
   for(col=1;col<13;col++)
      System.out.printf("%4d", row*col);
   System.out.print("\n");
}
```

```
     1   2   3   4   5   6   7   8   9
 1   1   2   3   4   5   6   7   8   9
 3   3   6   9  12  15  18  21  24  27
 5   5  10  15  20  25  30  35  40  45
 7   7  14  21  28  35  42  49  56  63
 9   9  18  27  36  45  54  63  72  81
11  11  22  33  44  55  66  77  88  99
```

Multiple statements in for loop

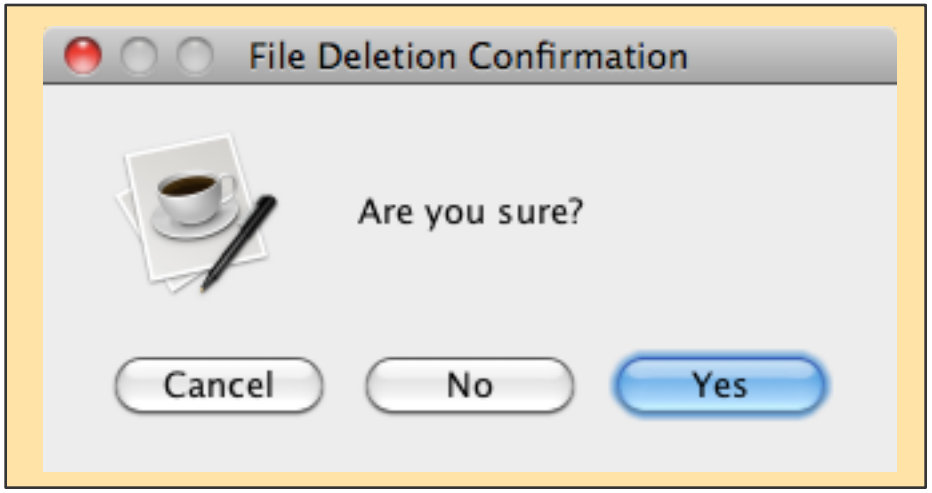
- The initialization and increment of a for loop can contain multiple statements separated by commas.

```
int sum;
for (int i=0,sum=0; i<=10;  sum+=i,i++){

}
System.out.println("Sum from 1 to 20 is:" + sum);
```

Confirmation Dialog

- Used to give the user a choice between different buttons.

```
JOptionPane.showConfirmDialog(null,
                    "Are you sure?",
                   "File Deletion Confirmation",
                   JOptionPane.YES_NO_CANCEL_OPTION);
```



Example: Confirmation Dialog

```
int choice;
choice = JOptionPane.showConfirmDialog(null,
                    "Are you sure?",
                    "File Deletion Confirmation",
                    JOptionPane.YES_NO_CANCEL_OPTION);

if(choice == JOptionPane.YES_OPTION)
   System.out.println("You chose to delete the file");
else if (choice == JOptionPane.NO_OPTION)
   System.out.println("You chose not to delete the file");
else
   System.out.println("You chose to cancel");
```

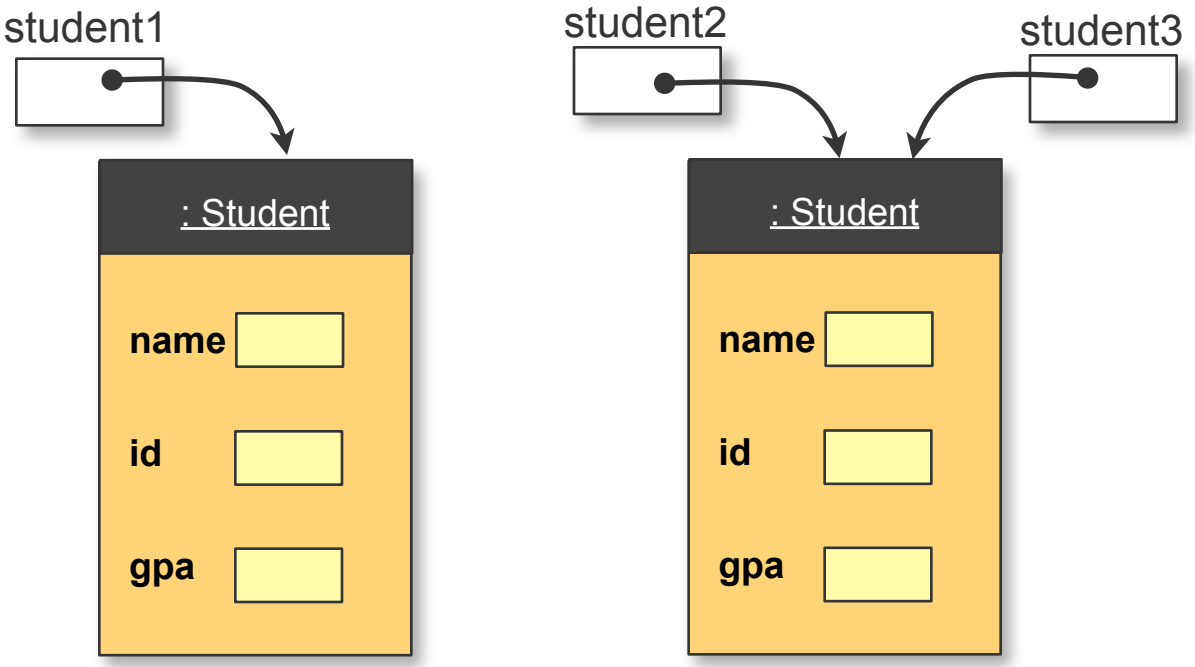
Comparing Objects

- As with numeric types, we can compare two objects for equality and inequality
  - Relational operators (<, < , <=, <=) are **not allowed** for objects.
- Recall that these are reference types.
  - Thus, we are really testing for equality of the references, i.e., are the two variable referencing the same object or not?
- If we want to compare their contents, we need special methods.

Comparing Objects

```
Student student1, student2, student3;

student1 = new Student();
student2 = new Student();
student3 = student2;

if(student1 == student2)
   System.out.println("Equal");
else
   System.out.println("Not Equal");
```

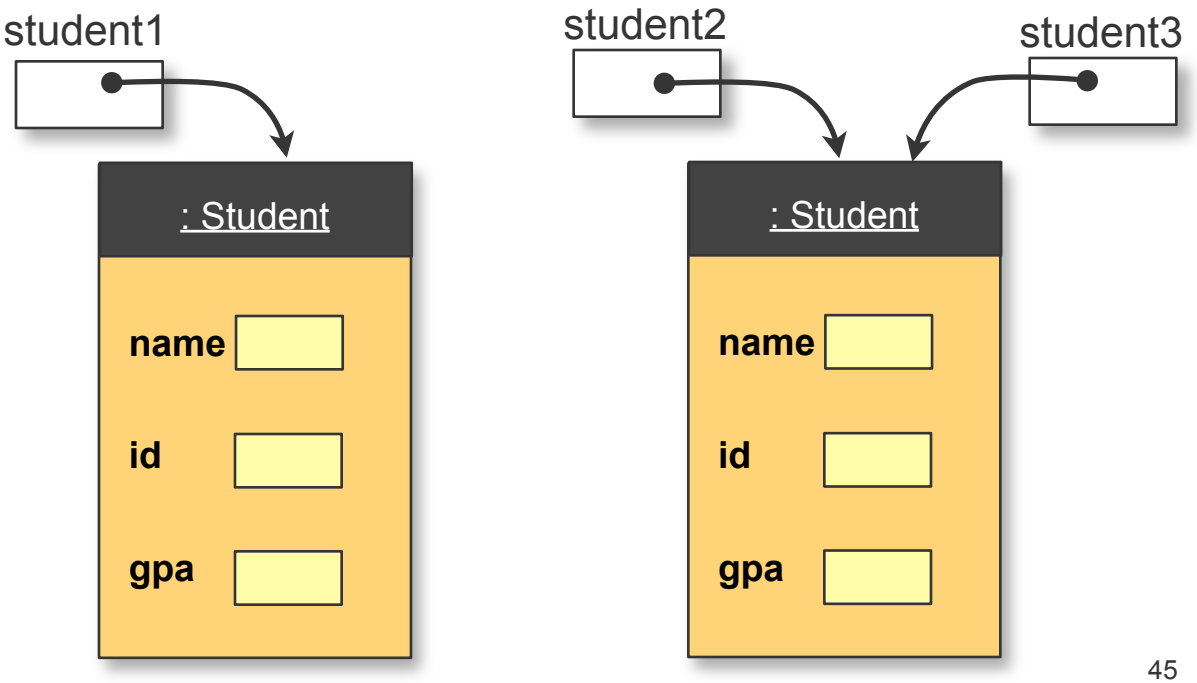
"Not Equal"



Comparing Objects

```
Student student1, student2, student3;

student1 = new Student();
student2 = new Student();
student3 = student2;

if(student3 == student2)
   System.out.println("Equal");
else
   System.out.println("Not Equal");
```

"Equal"

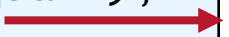


Comparing Object Contents

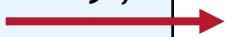
- If we want to compare the internal contents of objects, we have to use methods
- For example, with String objects, we can use
  - equals() to test equality of two strings' contents
  - equalsIgnoreCase() to test equality while treating upper and lower case of the same letter as equal
  - compareTo() to determine the relative position of two strings in lexicographic order.
  - each is called on one string with the other as an argument

Comparing Strings

```
String str1 = "Elephant", str2 = "eLePhant";

if(str1.equals(str2)){
     System.out.println("They are equal");
} else {
     System.out.println("They are not equal");
}

if(str1.equalsIgnoreCase(str2)){
     System.out.println("Equal, but for case");
} else {
     System.out.println("They are not equal");
}
```



"They are not equal"



"Equal but for case"

compareTo() method

- Strings are compared character by character. The return value is an integer that tells us their relative order.

```
String str1, str2;
int i;

i = str1.compareTo(str2);


if(i==0)
   System.out.println(str1 + " equals " + str2);
else
   if(i>0)
      System.out.println(str2 + " precedes " + str1);
   else
      System.out.println(str1 + " precedes " + str2);
```

equals() for Other Classes

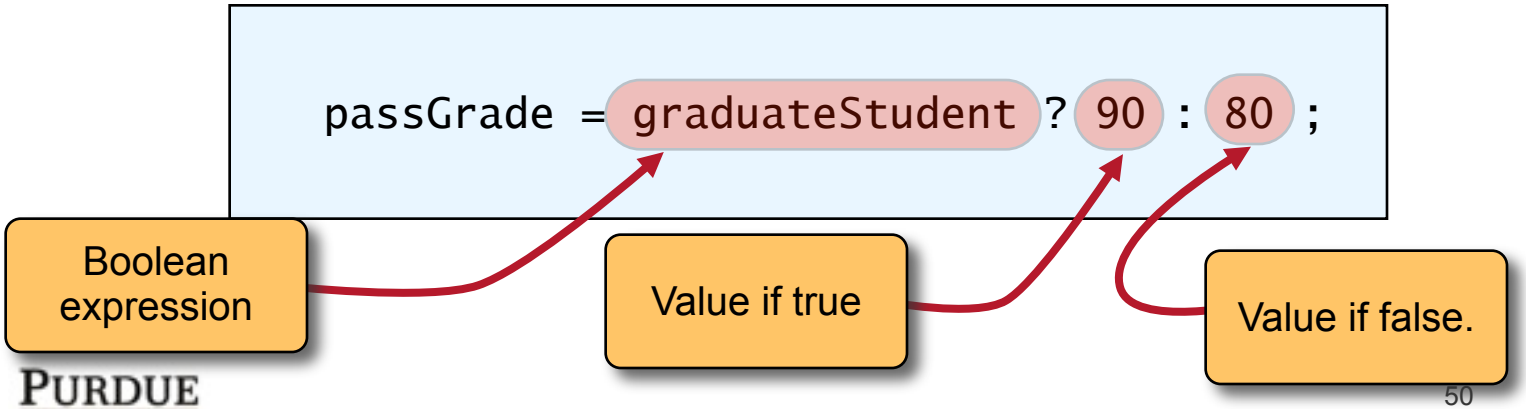
- All classes get an equals() method for free.
- However, it may not work as expected.
- If you wish to compare objects of your classes for equality of content you should write an appropriate method.
- We will see some examples later.

Ternary Assignment Operator

- A common situation is to assign one of two alternative values depending on a condition

```
if (graduateStudent)
   passGrade = 90;
else
   passGrade = 80;
```

- We can use the following ternary shortcut:



Short-Circuit Evaluation

- Sometimes it is unnecessary to compute all subparts of a boolean expression in order to know the overall value. E.g.,
  - i == j || k <5
    - if i is equal to j, the expression is **true** no matter what the value of k is
  - i == j && k < 5
    - if i is not equal to j, the expression is **false** no matter what the value of k is
- Most compilers will stop evaluating a expression if its overall value is clear earlier.
  - Called Short-Circuit (Lazy) Evaluation

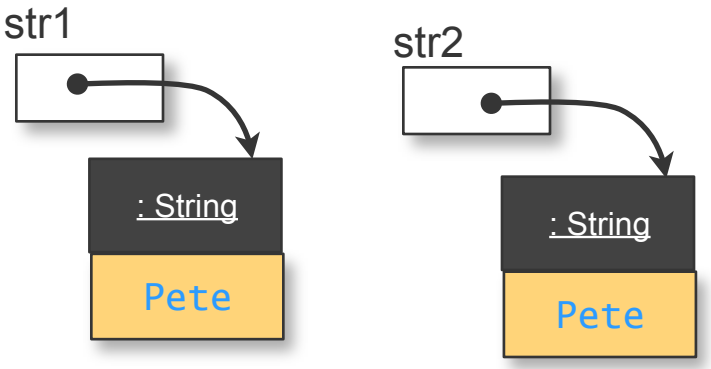
Short-Circuit Evaluation

- Why should we care?
- Can impact side effects of expressions:
  - done = (i == j) || (k++ < 5)
    - k is incremented only if i was not equal to j
- Can be useful
  - okay = (j == 0) || (i/j > 5)
    - prevents divide by 0 error
- We can force Full (Eager) Evaluation by using & instead of && and | instead of ||
- Caution:&, |, ^ also denote bitwise operations if the operands are integer values not boolean.

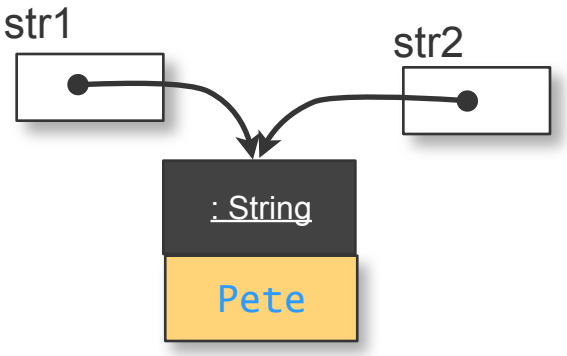
Caution: Strings

```
String str1, str2;

str1 = new String("Pete");
str2 = new String("Pete");

if(str1==str2)
    System.out.println("Equal");
else
    System.out.println("Not equal");
```

"Not equal"




```
String str1, str2;

str1 = "Pete";
str2 = "Pete";

if(str1==str2)
    System.out.println("Equal");
else
    System.out.println("Not equal");
```

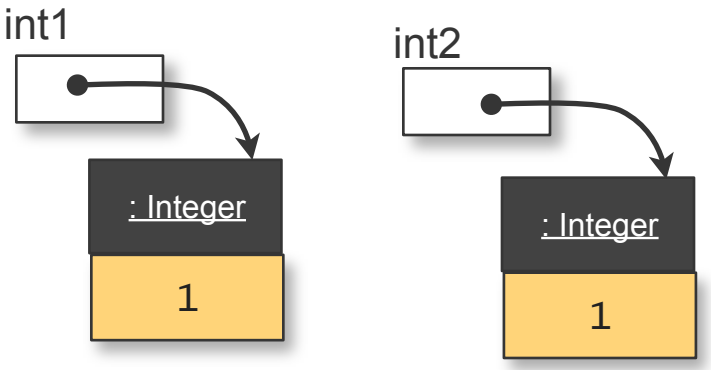


"Equal"

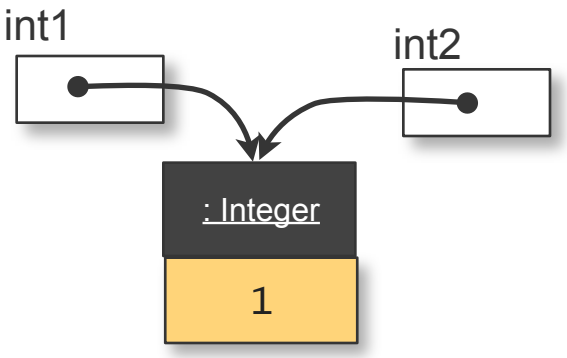
Caution: Wrapper Classes

```
Integer int1, int2;

int1 = new Integer(1);
int2 = new Integer(1);

if(int1==int2)
    System.out.println("Equal");
else
    System.out.println("Not equal");
```

"Not equal"




```
Integer int1, int2;

int1 = 1;
int2 = 1;

if(int1==int2)
    System.out.println("Equal");
else
    System.out.println("Not equal");
```



"Equal"

Caution: Wrapper Classes

```
Integer int1, int2;

int1 = 1;
int2 = 1;

if(int1==int2)
   System.out.println("Equal");
else
   System.out.println("Not equal");

int1 += 1;
if(int1==int2)
   System.out.println("Equal");
else
   System.out.println("Not equal");

int2 += 1;
if(int1==int2)
   System.out.println("Equal");
else
   System.out.println("Not equal");
```

Caution: Wrapper Classes

```
Integer int1, int2;

int1 = new Integer(1);
int2 = new Integer(1);

if(int1==int2)
   System.out.println("Equal");
else
   System.out.println("Not equal");

int1 += 1;
if(int1==int2)
   System.out.println("Equal");
else
   System.out.println("Not equal");

int2 += 1;
if(int1==int2)
   System.out.println("Equal");
else
   System.out.println("Not equal");
```

Caution: Object Equality

- Be very careful about using == and != with both Wrapper classes and Strings.
- They can have some surprising behaviors.
- In general, when using numeric values for boolean conditions, do not use Wrapper classes -- use the primitive types instead.

Side effects -- 1

```
int x= 1, y=10;
```

x = y++;	x: 10	y: 11
x = ++y;	x: 11	y: 11
x = -++y;	x: -11	y: 11
x = -y++;	x: -10	y: 11
x = -y--;	x: -10	y: 9
x = -(--y);	x: -9	y: 9
x = ++y++;	ERROR!	

Prefix vs. postfix.

- A prefix (postfix) operator is equivalent to executing the operator before (after) using the value of the variable:

```
z = x++ * --y;
```

- Is equivalent to:

```
y = y-1;
z = x * y;
x = x + 1;
```

What about:

```
z = x++ * x++;
```

PURDUE UNIVERSITY

More Examples

```
z = x++ * x++;
```

- Is equivalent to:

```
z = x * (x+1);
x = x+2;
```

```
z = x++ * --x;
```

- Is equivalent to:

```
z = x * x;
```

Can be tricky -- use with care.

Side effects -- 2

```
int x= 1, y=10, z=100;
boolean bool, test=false;
```

x = y = z;	x: 100 y: 100 z: 100
x = y = ++z;	x: 101 y: 101 z: 101
bool = (x=11)>y	x: 11 y: 10 bool: **true**
bool = (x=11)>y++	x: 11 y: 11 bool: **true**
bool = (x=11)> ++y	x: 11 y: 11 bool: **false**
bool = (x=3) > y && (z=5)<10	x: 3 y: 10 z: 10 bool: **false**
bool = (x=3) > y & (z=5)<10	x: 3 y: 10 z: 5 bool: **false**