COSC 1020

Yves Lespérance

Lecture Notes
Week 5 — Control Structures

Recommended Readings:
Horstmann: Ch. 5 and 6
Lewis & Loftus: Ch. 3

The `if` Statement

The `if` statement is used to *select* which is to be performed *among some alternative set of operations*.

There are several versions. The simplest one is used to perform a statement *only if a condition (boolean expression) is true*:

`if (`*condition*`)`
   statement

E.g.

```
double fare = 8.0;
if (age <= 17)
   fare = 5.0;
IO.println("Fare is " + fare);
```

Often, you want to perform several statements when the condition holds. You can always group several statements into one by putting them between braces. This is called a *block*. E.g.

```
IO.print("Please enter your age: ");
int age = IO.readInt();
if (age < 0)
{  IO.println("Negative age!");
   IO.print("Enter correct age: ");
   age = IO.readInt();
}
```

A second version of the `if` statement is used to perform *one statement if a condition is true* and *another statement if it is false*:

`if (`*condition*`)`
   statementIfT
`else`
   statementIfF

E.g. calculating $|x|$

```
if (x < 0)
   abs_x = -x;
else
   abs_x = x;
```

E.g.

```
final int DISCOUNT_AGE_LIMIT = 16;
IO.print("How old are you? ");
int age = IO.readInt();
if (age <= DISCOUNT_AGE_LIMIT)
{  IO.println("You get the discounted fare!");
   IO.println("Please remit $5.00.");
}
else
{  IO.println("You must pay the regular fare.");
   IO.println("Please remit $8.00.");
}
```

We can also do selection among more than two alternatives by cascading `if-elses` into one another. As well, we can tack on a “catchall” case at the end for a statement to be performed if *none* of the conditions are true.

```
if (condition1)
   statement1
else if (condition2)
   statement2
else if (condition3)
   statement3
...
else if (conditionN)
   statementN
else
   statementOtherwise
```

Note that even if more than one of the conditions is true, only the statement for the *first* true condition is performed.

Problem: Given a numerical grade, print out the letter grade equivalent.

```
IO.print("Your letter grade is ");
if (grade >= 80)
   IO.println("A");
else if (grade >= 70)
   IO.println("B");
else if (grade >= 60)
   IO.println("C");
else if (grade >= 50)
   IO.println("D");
else
   IO.println("F");
```

What is the output if the variable `grade` has value 72? What if it is 47?

You can have `if` statements nested inside other `if` statements. E.g.

```
final static int HEADS = 1;
final static int RECEIVE = 1;
IO.print("Enter 1 for heads and 2 for tails: ");
int coin = IO.readInt();
IO.print("Enter 1 to receive and 2 to kickoff: ");
int choice = IO.readInt();
if (coin == HEADS)
{ if (choice == RECEIVE)
    IO.println("You won the toss and will receive.");
  else
    IO.println("You won the toss and will kickoff.");
}
else
  IO.println("You lost the coin toss.");
```

Note that an `else` is matched with the most recent unmatched `if`.

Proper indentation is important for readability; it should reflect the structure of nesting!

Problem: for each possible value of `cond1` and `cond2`, say what the value of `r` is after the following code is executed,

```
int r = 0;
if (cond1)
   if (cond2)
      r = 1;
else
   r = 2;
```

What if we add braces?

```
int r = 0;
if (cond1)
{  if (cond2)
      r = 1;
}
else
   r = 2;
```

E.g. MkChange Revisited

Modify the MkChange program to use singular and plural correctly, omit coins that don't have to be included, and reject negative amounts. E.g.

```
Script started on Thu Sep 30 14:57:36 1999
Terminal is : xterm
Display is: fxt23.cs.yorku.ca:0.0
tiger 41 % java MkChange2
Enter the amount in cents: 68
Change is:  2 quarters  1 dime  1 nickel  3 pennies.
tiger 42 % java MkChange2
Enter the amount in cents: 97
Change is:  3 quarters  2 dimes  2 pennies.
tiger 43 % java MkChange2
Enter the amount in cents: 12
Change is:  1 dime  2 pennies.
tiger 44 % java MkChange2
Enter the amount in cents: 0
Change is:.
tiger 45 % java MkChange2
Enter the amount in cents: -20
The amount can't be negative.  Goodbye.
tiger 46 % exit
exit

script done on Thu Sep 30 14:58:42 1999
```

```
import type.lang.*;

public class MkChange2
{  public static void main(String[] args)
   {  final int QUARTER_VALUE = 25;
      final int DIME_VALUE = 10;
      final int NICKEL_VALUE = 5;
      IO.print("Enter the amount in cents: ");
      int amount = IO.readInt();
      if (amount < 0)
         IO.println("The amount can't be negative.  Goodbye.");
      else
      {  int nQuarters = amount / QUARTER_VALUE;
         amount = amount % QUARTER_VALUE;
         int nDimes = amount / DIME_VALUE;
         amount = amount % DIME_VALUE;
         int nNickels = amount / NICKEL_VALUE;
         int nPennies = amount % NICKEL_VALUE;
         IO.print("Change is:");
         if (nQuarters > 1)
            IO.print("  " + nQuarters + " quarters");
         else if (nQuarters == 1)
            IO.print("  " + nQuarters + " quarter");
         // else when nQuarters == 0 print nothing
         if (nDimes > 1)
            IO.print("  " + nDimes + " dimes");
         else if (nDimes == 1)
            IO.print("  " + nDimes + " dime");
         if (nNickels > 1)
            IO.print("  " + nNickels + " nickels");
         else if (nNickels == 1)
            IO.print("  " + nNickels + " nickel");
         if (nPennies > 1)
            IO.print("  " + nPennies + " pennies");
         else if (nPennies == 1)
            IO.print("  " + nPennies + " pennie");
         IO.println(".");
      }
   }
}
```

Repetition E.G

Problem: print a table of the squares of positive integers from 1 to 10 as below; make sure the colums are correctly alligned.

```
 n   n^2
 1     1
 2     4
 3     9
 4    16
 5    25
 6    36
 7    49
 8    64
 9    81
10   100
```

Design

Must print one line at a time; in each line print n in a column of width 2, then 3 spaces, then n^2 in a column of width 3.

Printing the table is a repetitive task. Know in advance how many repetitions, 10 = N_MAX, so use `for` loop. n starts at 1, increases by 1 each repetition, and the loop stops when `n > N_MAX`. So use

```
for (int n = 1; n <= N_MAX; n++)
```

Algorithm in pseudocode

```
print header
for n from 1 to N_MAX incrementing by 1
{   print n in column of width 2
    print 3 spaces
    print $n^2$ in column of width 3
}
```

Code

```
import type.lang.*;

public class SquaresTbl
{  public static void main(String[] args)
   {  final int N_MAX = 10;
      IO.println(" n   n^2");
      for (int n = 1; n <= N_MAX; n++)
      {  IO.print(n, "2");
         IO.print("   ");
         IO.println(n * n, "3");
      }
   }
}
```

`for` Loops

When we want to repeatedly perform some operations — this is called a *loop* — and the number of repetitions is *known before the loop begins*, we use a loop *with a counter*.

Java provides the `for` structure for this; its syntax is:

`for (`*init*; *cond*; *update*`)`
    statement

init is an expression that is executed at the beginning of the loop, usually to initialize the counter;

cond is a condition that is tested at the beginning of each cycle — the loop continues while it remains true;

update is an expression that is executed at the end of every cycle and is typically used to update the counter for the next cycle.

E.g. read a positive integer n and print the sum of the first n positive integers:

```
int sum = 0;
IO.print("Enter a pos. integer: ");
int n = IO.readInt();
for (int i = 1; i <= n; i++)
   sum = sum + i;
IO.println("The sum of the first "
   + n + " pos. integers is " + sum);
```

The counter can start at any value, go down as well as up, and change by an arbitrary amount, e.g.

```
for (int i = 5; i <= n; i++)
for (int i = 10; i > 0; i--)
for (int i = 0; i <= n; i = i + 5)
```

Problem: write a program that prints a triangle on the screen as shown below; the size of the triangle (number of lines) should be read from the user.

```
tiger 87 % java DrawTriangle
Enter number of lines the triangle should contain: 4
*******
 *****
  ***
   *
tiger 88 % java DrawTriangle
Enter number of lines the triangle should contain: 7
*************
 ***********
  *********
   *******
    *****
     ***
      *
```

Design

First decomposition of task:

```
print prompt
read nLines
print triangle of size nLines
```

Printing triangle subtask: can't be done in one shot, must repeatedly print one line. Number of repetitions known, `nLines`. So use a `for` loop. Algorithm becomes:

```
print prompt
read nLines
for i from 1 to nLines incrementing by 1
    print line i of triangle
```

Printing line `i` of triangle subtask:

How many spaces on line `i`?
How many stars on line `i`?

Because number of spaces (stars) varies with line number, can't be done in one shot, must repeatedly print one space (star) at a time. Because know how many repetitions in advance, use `for` loops.

Algorithm

```
print prompt
read nLines
for i from 1 to nLines incrementing by 1
{  nSpaces = i - 1;
   nStars = 2 * (nLines - i) + 1;
   for j from 1 to nSpaces incrementing by 1
       print a space
   for j from 1 to nStars incrementing by 1
       print a star
   skip to next line
}
```

Code

```
import type.lang.*;
public class DrawTriangle
{  public static void main(String[] args)
   {  IO.print("Enter number of lines the triangle shou
      int nLines = IO.readInt();
      for(int i = 1; i <= nLines; i++)
      {  int nSpaces = i - 1;
         int nStars = 2 * (nLines - i) + 1;
         for(int j = 1; j <= nSpaces; j++)
            IO.print(" ");
         for(int j = 1; j <= nStars; j++)
            IO.print("*");
         IO.println();
      }
   }
}
```

Top-Down Design

Start by producing a very coarse "high-level" algorithm to solve the problem. This requires identifying steps in the solution or subproblems. Ok if subproblems must be solved later.

Then work on subproblems separately, applying the same decomposition method.

Repeat until the all subproblems have been solved and the algorithm is sufficiently detailed to be writen in code.

Top-down design can be used in combination with object-oriented design.

Problem: write a program that prints a square wave as shown below; input the amplitude and wavelength from the user; both pos. integers; wavelength should be even.

```
tiger 122 % java SquareWaves
Enter amplitude: 3
Enter wavelength: 4
******
     *
******
*
******
     *
******
*
tiger 123 % java SquareWaves
Enter amplitude: 4
Enter wavelength: 6
********
       *
       *
********
*
*
********
       *
       *
********
*
*
```

Design

Need to repeatedly print cycles of the wave, here 2 cycles.
Use constant `N_CYCLES`.
Know how many repetitions, so `for` loop.
1st decomposition:

```
print prompt & read amplitude
print prompt & read wavelength
for i from 1 to N_CYCLES
   print a cycle
```

To “print a cycle”, there are 4 subtasks.
2nd decomposition:

```
print row of stars of length 2*amplitude
print column of stars of length wavelength/2-1
      indented by 2*amplitude-1 spaces
print row of stars of length 2*amplitude
print column of stars of length wavelength/2-1
```

To “print row of stars of length `2*amplitude`”, need to repeatedly print a star `2*amplitude` times.
Algorithm:

```
for j from 1 to 2*amplitude
      print a star
skip to next line
```

To “print column of stars of length `wavelength/2-1`”, need to repeatedly print lines with a star on it that many times.
Algorithm:

```
for j from 1 to wavelength/2-1
      print a line with a star
```

To “print column of stars of length `wavelength/2-1` indented by `2*amplitude-1` spaces”, need to repeatedly print such a line that many times.
Algorithm:

```
for j from 1 to wavelength/2-1
      print 2*amplitude-1 spaces
      print a star and skip to next line
```

To print “`2*amplitude-1` spaces”, repeatedly print a space that many times.
Algorithm:

```
for k from 1 to 2*amplitude-1
      print a star
```

Code

```
import type.lang.*;

public class SquareWaves
{  public static void main (String[] args)
   {  final int N_CYCLES = 2;
      IO.print("Enter amplitude: ");
      int amplitude = IO.readInt();
      IO.print("Enter wavelength: ");
      int wavelength = IO.readInt();
      for (int i = 1; i <= N_CYCLES; i++)
      {  for (int j = 1; j <= 2*amplitude; j++)
            IO.print("*");
         IO.println();
         for (int j = 1; j <= wavelength/2-1; j++)
         {  for (int k = 1; k <= 2*amplitude-1; k++)
               IO.print(" ");
            IO.println("*");
         }
         for (int j = 1; j <= 2*amplitude; j++)
            IO.print("*");
         IO.println();
         for (int j = 1; j <= wavelength/2-1; j++)
            IO.println("*");
      }
   }
}
```

Other Uses of `for`

`for` loops are also used to iterate over members of a collections or input records. E.g.

```
import type.lang.*;
public class SalaryAvg
{  public static void main(String[] args)
   {  double total = 0.0;
      int count = 0;
      IO.println("Enter salaries one per line " +
                 "& end with negative amount...");
      for(double sal = IO.readDouble(); sal >= 0.0;
          sal = IO.readDouble())
      {  total = total + sal;
         count++;
      }
      IO.print("The average of all salaries is $");
      IO.println(total/count, ",.2");
   }
}
```

```
zebra 306 % java SalaryAvg
Enter salaries one per line & end with negative amount
20.50
30.50
40.50
-29.0
The average of all salaries is $30.50
```

The value used to indicate the end of input is called a *sentinel*.

File I/O Using `UniReader/UniWriter`

The `type.lang` package provides a simple way to input from a file (or URL) or output to a file (or URL). For input, use the `UniReader` class:

1. Open the file for input by calling `UniReader`'s constructor, e.g.

   ```
   UniReader myReader =
      new UniReader("myInputFile.txt");
   ```

2. Read in your data, e.g

   ```
   String name = myReader.readLine();
   int age = myReader.readInt();
   ```

3. Close the file, e.g.

   ```
   myReader.close();
   ```

E.g.

```
import type.lang.*;
public class SalaryAvgFFile
{  public static void main(String[] args)
    {  final String inputFileName = "myInputFile.txt";
       UniReader myReader =
           new UniReader(inputFileName);
      double total = 0.0;
      int count = 0;
      for(double sal = myReader.readDouble();
          !myReader.eof();
          sal = myReader.readDouble())
      {  total = total + sal;
         count++;
      }
      myReader.close();
      IO.print("The average of all salaries is $");
      IO.println(total/count, ",.2");
   }
}
```

Note that the methods in the `UniReader` class work a bit differently from those in the `IO` class. `UniReader`'s `readInt()`, `readDouble()`, etc. do not necessarily read a whole line.

You can also check whether you have reached the end of the file by calling the `eof()` method.

Writing to a file is similar:

```
UniWriter myWriter =
   new UniWriter("myOutputFile.txt");
...
myWriter.println("This is to be printed.");
...
myWriter.close();
```

Scope of Variables

The *scope* of a variable is the part of the program where it is *visible*, where it can be accessed. The variables declared inside a block `{ ... }` are *local* to the block and are only visible in the block. E.g.

```
public class ScopeEg
{  public static void main(String[] args)
   {  int v1 = 1;
      for(int v2 = 2; v2 <= 4; v2++)
      {  int v3 = 3;
         IO.println(v3); // ok
         IO.println(v2); // ok
         IO.println(v1); // ok
      }
      IO.println(v3); // not ok!
      IO.println(v2); // not ok!
      IO.println(v1); // ok
   }
}
```