IONA

Department of Computer Science

Loops

Dr. Chia-Ling Tsai

Outline

- Logical operators
- Definite loops – the **`for`** statement
- Early termination
- Indefinite loops – the **`while`** statement

Logical Operators

- The "and" and "or" operators
- Example and a warning
- Boolean variables
- The "not" operator

The "and" and "or" Operators

- " Boolean" means "logical"
- You can combine several conditions into one using *Boolean operators*
  - For "and" use `&&`
  - For "or" use `||`
- Example
  - To say `x < 2` *or* `y < 5` write `x < 2 || y < 5`
  - To say `x < 2` *and* `y < 5` write `x < 2 && y < 5`

Example and a Warning

- Example
  - Condition 1: "x is greater than 2"
    - Java: `x > 2`
  - Condition 2: "x is less than or equal to 5"
    - Java: `x <= 5`
  - Combination: "x is greater than 2 and less than or equal to 5"
    - Java: `x > 2 && x <= 5`
- Warning
  - You have to put in the second `x`
  - You *can't* write it as in mathematics: `2 < x <= 5`

Boolean Variables

- A variable of type **`boolean`** has value either **`true`** or **`false`**
- Replace a complicated condition with a well-named **`boolean`** variable
- Example
  - English: salary is at least $50,000 and savings are at least $100,000
  - Java: `salary >= 50000 && savings >= 100000`
  - Java using Boolean variables

```
boolean highSalary = salary >= 50000;
boolean enoughSavings = savings >= 100000;
if (highSalary && enoughSavings)
  System.out.println("Eligible for loan");
```

The "not" Operator

- For "not" use `!`
- This operator is especially useful with named Boolean variables
- Example
  - Salary is not high when the condition `salary >= 50000` becomes `salary < 50000`
  - You can also just write `!highSalary`

The Truth Tables

- NOT (!)

p	!p
True	false
False	true

- AND (&&)

p	q	p && q
True	True	true
True	False	false
False	True	false
False	False	false

The Truth Tables

- OR (||)

p	q	p || q
True	True	true
True	False	true
False	True	true
False	False	false

De Morgan's Law

- not (P and Q) = (not P) or (not Q)
  - `!(highSalary && enoughSavings)= ! highSalary || ! enoughSavings`
- not (P or Q) = (not P) and (not Q)
  - `!(smart || hardWorking)= !smart && !hardWorking`

Repetition

- Repeated action in a program is common
  - E.g. comparing numbers for sorting
- Repeat
  - beat in one egg

  Until no more eggs
- Loop i from 1 to 5 do
  - Print (“*”)

  End loop i

Definite Loops

- Java does not have a simple statement of the form “Do *n* times”
- The idea of a *definite loop* is that we use a *variable* to do the counting
- You must tell Java
  - The name of the variable and what value to start with (declare and initialize the variable)
  - What must be true of the variable to continue the repetition
  - What to do to the variable between repetitions

`for` Statement Syntax

- To print a vertical line of three stars:

```
for (int count = 1; count < 4; count++)
{
  System.out.println("*");
}
```

  - "`int count = 1`" : Initialization of the counter variable
  - "`count < 4`" : Continuation test
  - "`count++`" : Update on the counter variable
- Notice that the initialization is done *once* and that the *order* within the repetition is (1) check the condition, then (2) do the statement, and then (3) increment the counter

Can we use for-loops here?

```
// A program that draws the letter E
public class DrawE
{
   public static void main(String[] args)
   {
      // Print horizontal line
      System.out.println("*******");
      // Print vertical segment
      System.out.println("*");
      System.out.println("*");
      System.out.println("*");
      // Print horizontal line
      System.out.println("*******");
      // Print vertical segment
      System.out.println("*");
      System.out.println("*");
      System.out.println("*");
      // Print horizontal line
      System.out.println("*******");
   }
}
```

More On Initialization

- The initialization of a **`for`** loop looks just like a declaration with initialization
- One difference: you may use *any* name for the counter *including one you have used before*
  - But you must avoid confusing yourself!
  - However, it can be convenient to reuse the same counter name in several loops
- Typically, short names like `i` and `n` are used, but of course there is nothing wrong with a meaningful name like `count`

Comparing Numerical Values

- In forming conditions, *comparisons* are often used

Math	Java	Description
$>$	`>`	Greater than
$\geq$	`>=`	Greater than or equal
$<$	`<`	Less than
$\leq$	`<=`	Less than or equal
$=$	`==`	Equal
$\neq$	`!=`	Not equal

Getting The Count Right

- It is very easy to be *off by one* in counting
- Each of the following causes repetition *3* times

```
for (int count = 1; count < 4; count++)
  { System.out.println("*"); }
for (int count = 1; count <= 3; count++)
  { System.out.println("*"); }
for (int count = 0; count < 6; count+=2)
  { System.out.println("*"); }
for (int count = 12; count < 15; count++)
  { System.out.println("*"); }
```

- Of course the last one is silly – but each of the others is reasonable; use the one you are least likely to get wrong

---

- Our example with loops

```
// A program that draws the letter E
// Revision: Uses loops
public class DrawEWithLoops
{
   public static void main(String[] args)
   {
      // Print horizontal line
      for (int count = 1; count <= 7; count++)
      {
         System.out.print("*");
      }
      System.out.println();
      // Print vertical segment
      for (int count = 1; count <= 3; count++)
      {
         System.out.println("*");
      }
      // Print horizontal line
      for (int count = 1; count <= 7; count++)
      {
         System.out.print("*");
      }
      System.out.println();
      // Print vertical segment
      for (int count = 1; count <= 3; count++)
      {
         System.out.println("*");
      }
      // Print horizontal line
      for (int count = 1; count <= 7; count++)
      {
         System.out.print("*");
      }
      System.out.println();
   }
}
```

Named Constants

- Numbers like 7 and 3 in the previous example are sometimes called *magic numbers*
  - The exact choice of numbers to use is not a matter of any apparent logic
- Instead of using the numbers themselves, we can define names for them, just as we do for variables
- A named constant is not expected to change, so we use the word final preceding its declaration, which Java will enforce
- Differences from variable declarations
  - Use final
  - Always initialize
  - Spell name in capitals (a C tradition, whose rationale no longer exists, but....)

```
public class DrawEWithLoopsAndConstants
{
    public static void main(String[] args)
    {
        // Define sizes of drawing elements
        final int WIDTH = 7;
        final int SEGMENT_HEIGHT = WIDTH/2;
        // Print horizontal line
        for (int count = 1; count <= WIDTH; count++)
        {
            System.out.print("*");
        }
        System.out.println();
        // Print vertical segment
        for (int count = 1; count <= SEGMENT_HEIGHT; count++)
        {
            System.out.println("*");
        }
        // Print horizontal line
        for (int count = 1; count <= WIDTH; count++)
        {
            System.out.print("*");
        }
        System.out.println();
        // Print vertical segment
        for (int count = 1; count <= SEGMENT_HEIGHT; count++)
        {
            System.out.println("*");
        }
        // Print horizontal line
        for (int count = 1; count <= WIDTH; count++)
        {
            System.out.print("*");
        }
        System.out.println();
    }
}
```

- The names *document* what we were thinking when we chose the numbers
- Another benefit is that we were able to define one constant in terms of the other
- A single, simple change (of 7 to some other integer) can *rescale* the drawing

Nested Loops

- We have seen plenty already!

loop i **from** A.length **down to** 1, **begin**
  loop j **from** 1 **to** i, **begin**
    if A[j-1]>A[j], **do** swap A[j-1] and A[j]
  end loop j
end loop i

Nested Loops

- Inside a loop may be any statements
- One of the statement inside a loop may be another loop
  - This is called *nesting* loops
- Nothing new is needed *in the design*
  - Plan a loop from the "outside in"
  - If the statement of a loop involves repetition, make it a loop also
- The only thing to watch out for is that you use different variables when one loop is inside the other

Example

- Problem: print a multiplication table
- Desired output

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

Design

- We must print 9 rows
  - This is a repetition of something 9 times, so we use a counting loop

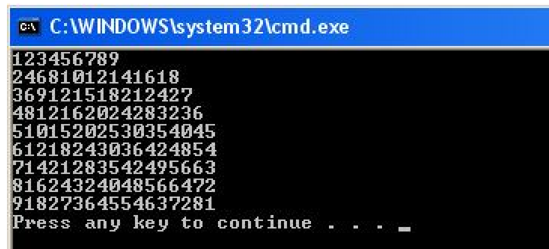
```
for (int row = 1; row <= 9; row++)
{   // print one row
}
```

- In each row, we must print 9 numbers
  - This is a repetition of something 9 times, so we use a counting loop

```
for (int column = 1; column <= 9;
column++)
 {   // print product of row and column
 }
```

Program

```
for (int row = 1; row <= 9; row++)
{ // print one row
   for (int column = 1; column <= 9; column++)
   { // print product of row and column
      int product = row * column;
      System.out.print(product);
   }
   // end row
   System.out.println();
}
```



Closer than it looks – all the numbers are correct
Only the spacing (none here) is wrong

Prettier Output

```
// A program that prints the multiplication table
public class TablePrinter
{
   public static void main(String[] args)
   {
      for (int row = 1; row <= 9; row++)
      { // print one row
       for (int column = 1; column <= 9; column++)
       { // print product of row and column
        int product = row * column;
        System.out.printf("%7d", product);
       }
       // end row
       System.out.println();
      }
   }
}
```

Use formatted printing to line things up

Early Termination

- What if we want to break out of the loop earlier?
  - Use “break” to end the loop immediately.
- What if we want to skip the rest of the current iteration.
  - Use “continue” to skip the iteration.

Example: Prime Numbers

- A prime number is divisible only by 1 and itself.
- Problem: Print out all the prime numbers between 2 and n.
- Algorithm:

The `while` Statement

- Not all loops needed in programs are *definite*
  - We don't know how many times to execute the loop
  - Example: Given the annual rate of growth in a population, how many years will it take for the population to double?
- A statement suitable for indefinite loops is has the form

```
while (condition)
{
    statements to repeat
}
```

- Important: The *statements to repeat* must change the *condition* or the repetition will never end

Sample Program

```
public class PopulationDoubler {
    public static void main(String[] args) {
        final int INIT_POPULATION = 100; //in million
        final double RATE = 0.1;
        int years = 0;
        int population = INIT_POPULATION;
        while (population < 2*INIT_POPULATION){
            years++;
            int increase = (int)(RATE * population);
            population = population + increase;
        }
        // Report result
        System.out.println("The population will double in "+years+" years");
    }
}
```

A Closer Look

```
int years = 0;
int population = INIT_POPULATION;
while (population < 2*INIT_POPULATION){
    years++;
    int increase = (int)(RATE * population);
    population = population + increase;
}
```

Variables must be defined before loop

The *condition*

Ensures that *condition* will change

```
while (population < 2*INIT_POPULATION){
    years++;
    int increase = (int)(RATE * population);
    //population = population + increase;
}
```

Error: condition never changes
An *infinite loop*

Another Example: Square Root

- Heron's iterative method to calculate the square root of a positive number
- Problem: To find the square root of an input number *n*

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{n}{x_n}\right)$$

Updated guess on (N+1)-th iteration

The guess of the N-th iteration

SquareRoot

```
// Finding the square root using Heron's method
public class SquareRoot {
    public static void main(String[] args) {
        final int number=10;

        // Initializing the guess;

        while ( /*condition*/ ) {

            // updating

        }
        System.out.println("The square root = "+ guess);
    }
}
```

Greatest Common Divisor

- Given two non-zero natural numbers *a* & *b*
- Euclid's method:

  1st try: a=15, b=9, 15 mod 9 = 6

  2nd try: a=9, b=6, 9 mod 6 = 3

  3rd try: a=6, b=3, 6 mod 3 = 0!

  GCD = 3.

GCD

```java
// Using the Euclid method to find the greatest common divisor
public class GreatestCommonDivisor {
    public static void main(String[] args) {
        int num1= 9;
        int num2=6;
        // Initialization

        while ( /* condition */) {

            //updating

        }
        System.out.println("GCD = "+num2);
    }
}
```

Variation—**`do/while`** loop

- The while loop tests at the “top” of the loop
- The do/while loop tests at the “bottom” of the loop.
- Example:

```java
int number = 1;
do{
  number *= 2;
} while (number <= 100);
```

- You decides which works better for you in a specific instance.

SquareRoot Revisited

- Let's implement the Heron's method again using do/while.

Lab5: squares

- 1*1 = 0 + 1
- 2*2 = 0 + 2 + 2
- 3*3 = 0 + 3 + 3 + 3
- 4*4 = 0 + 4 + 4 + 4 + 4
- 5*5 = 0 + 5 + 5 + 5 + 5 + 5
- n*n = 0 + n + n + … + n = answer

Addition is done n times

Summary

- Boolean logic
- Definite loops – the **`for`** statement
- Indefinite loops – the **`while`** statement

Common Syntax Mistakes

- Too many ";" or not enough
  - for (int i = 0; i<max; i++);
  - ans = ans + n
- {} missing or unmatched
  - Using Ctrl-m to check the matching one