CS 106
Introduction to Computer Science I

06 / 08 / 2015

Instructor: Michael Eckmann
Today’s Topics

• Questions / comments?
• continue; statement (contrasted with break;)
• Nested loops
• Arrays
• Math.random() method
The `continue;` is a way to cause your loop to skip the rest of the current iteration of the loop and continue on to the next iteration.

- It skips all the code after it and then goes to the next iteration.
- It acts slightly differently in for loops vs. while or do-while loops.
- After the rest of the current iteration is skipped, for a for loop, the next thing that happens is the `expression3` then `expression2` is evaluated to determine if it should go again.
continue;

• However, continue; within a while or a do-while loop acts like:

• After the rest of the current iteration is skipped, the next thing that happens is the condition is tested to determine if it should go again.
for (int x = 0; x <= 10; x++)
{
    if (x == 5)
        continue;
    System.out.println("x = " + x);
}

// what's this code going to do?
continue;

int x = 0;
while (x <= 10)
{
    if (x == 5)
        continue;
    System.out.println("x = " + x);
    x++;
}

// what's this code going to do?
Nested loops

• The body of a loop can contain another loop.
• We say that the inner loop is nested inside (or within) the outer loop.
• It is important to understand that the inner loop executes fully (all its iterations) within each iteration of the outer loop.
• For example, if the inner loop of a nested loop always iterates 5 times and the outer loop always iterates 10 times --- the code within the inner loop (its body) will end up executing a total of 50 times --- that is, 5 inner-loop iterations for each of the 10 outer-loop iterations.
Nested loops

• The body of a loop can contain another loop.
• We say that the inner loop is nested inside (or within) the outer loop.
• It is important to understand that the inner loop executes fully (all its iterations) within each iteration of the outer loop.
• For example, if the inner loop of a nested loop always iterates 5 times and the outer loop always iterates 10 times --- the code within the inner loop (its body) will end up executing a total of 50 times --- that is, 5 inner-loop iterations for each of the 10 outer-loop iterations.
public class Printing
{
    public static void main(String args[])
    {
        for ( int i = 1;   i <= 10;   i++)
        {
            for ( int j = 1;   j<=5;   j++)
            {
                System.out.print("@");
            }
            System.out.println();
        }
    }
}
Nested loop - what prints?

```java
public class Printing2 {
    public static void main(String args[]) {
        int i = 1, j = 1;
        for ( ; i <= 10; i++) {
            for ( ; j<=5; j++) {
                System.out.print("@");
            }
            System.out.println();
        }
    }
}
```
Nested loop - what prints?

```java
public class Printing3
{
    public static void main(String args[])
    {
        int i = 1, j = 1;
        while (i <= 10)
        {
            while (j <= 5)
            {
                System.out.print("@");
                j++;
            }
            System.out.println();
            i++;
        }
    }
}
```

Michael Eckmann - Skidmore College - CS 106 - Summer 2015
Nested loop - what prints?

```java
public class Printing4 {
    public static void main(String args[]) {
        int i = 1;
        while (i <= 10) {
            int j = 1;
            while (j <= 5) {
                System.out.print("@");
                j++;
            }
            System.out.println();
            i++;
        }
    }
}
```

Michael Eckmann  -  Skidmore
College  -  CS 106  -  Summer 2015
Pay attention to control variable

• The lesson is, to pay attention to the control variables.

• Does the inner loop's control variable get reset each time through the outer loop?, etc.
labelled break & continue statements

- labelled **break** statements within nested structures (**while**, **for**, **do while** or **switch** structure)
  - A label can be given to a block of code/structure. A block of code is defined as code enclosed in a pair of curly braces.
  - The labelled **break** statement can specify the block to “jump out of.” Non-labelled break statements are only able to “jump out of” the immediately enclosed structure, whereas, labelled break statements can “jump totally out of” a set of nested structures.
labelled break & continue statements

• labelled continue statements within nested structures (while, for, or do while structure)
  – A label can be given to a structure.
  – The labelled continue statement causes execution to skip the code in the enclosing structures and continues with the next iteration of the structure with the specified label.
labelled break & continue statements

• Let’s look at an example of how to label a block and nest some loops so that we can do a labelled break;
Arrays

• Oftentimes we find ourselves needing to store a list of items (e.g. a list of ages, grades, names, etc.)
• Up until now, if we wanted to store all the titles of each course at Skidmore in a separate variable, we would need to declare hundreds of differently named variables of type String.
• This would get out of hand quickly. Fortunately there's a data structure called an array that allows us to store a list of items of the same type.
Arrays

- A data structure of related data items of the *same type*
- They store this data in contiguous memory locations
- Each data item in an array is an *element*.
- Arrays are denoted with square brackets.
- When you want to select one element of the array, use its *index* in square brackets. *Array indices start at 0.*
- When we speak of arrays: *array* means the full array, and *array element* is one element of the array.
Arrays

• Arrays can be of any type (user-defined class types, Java API class types, or primitive types.)

• Arrays of primitive types:
  – Might use an array of integers to store test scores. (better than creating individual variables of type int for reasons which should become clear.)
  – Might use an array of doubles to store a list of temperatures.

• Arrays of Java API class types:
  – Might use an array of Strings to store a list of people’s names.
Arrays (declaring and allocating)

• Arrays, like any variables need to be declared.
• Arrays also need to have memory allocated for a specific number of elements.

• Example:
  ```java
  int testScores[]; // declares an array of ints
  
testScores = new int[16]; // the line above allocates enough memory for 16 ints
  ```
Arrays (declaring and allocating)

• Arrays can be declared and allocated in one line.
• Example:

```
// declares an array of ints & allocates enough memory for 16 ints
int testScores[] = new int[16];
```
Arrays (allocating and initializing)

• When arrays are allocated, the array elements are all initialized to the following values:
• Arrays of numeric primitive types (byte, short, long, int, float, double, char) have their elements initialized to 0.
• Arrays of booleans (the only other primitive type) have their elements initialized to false.
• Arrays of any non-primitive data type, (classes in the Java API or user-defined classes) have their elements initialized to null.
Arrays (declaring, allocating and initializing)

• Arrays can be declared, allocated and initialized to user defined values all in one line of code.

• Example:

```c
int testScores[] = {
    130, 120, 111, 44, 181,
    183, 160, 165, 148, 147,
    146, 153, 139, 150, 153,
    122
};
```

// declares an array of ints & allocates enough memory for 16 ints and // initializes each element to the value listed
Arrays

• In our example of the testScores array, each element of this array is an `int`. So, we can refer to say the 10th element of the array by `testScores[9]` and this is an `int`, so we can use it anywhere an `int` can be used.

• REMEMBER: Array indices start at 0 (zero) and end at the (size of the array – 1).
  – E.g. Indices for an array of size 36, go from 0 to 35.

• Array values can be set one element at a time by supplying the index like so:
  
  // stores the value 130 in the 0 element of the array testScores
  testScores[0] = 130;

  // stores the value 120 in the 1 element of the array testScores
  testScores[1] = 120;

  // etc …
Arrays

• Each array knows its own length, use the array name followed by `.length` to get access to its length. In our case for the test_scores example, the length is 16 (0 to 15.)

• Arrays are easily manipulated with loops.

• Let’s look at an example.
Array code example

// Code to populate (store values into the elements of) an array

int testScores[] = new int[ 16 ];
String userInput;

for (int i=0; i < testScores.length; i++)
{
    userInput = JOptionPane.showInputDialog("Enter an integer");
    testScores[i] = Integer.parseInt(userInput);
}
Array code example

// Code to find the lowest element of an array of ints.

int least = testScores[0];

for (int i=0; i < testScores.length; i++)
{
    if (testScores[i] < least)
    {
        least = testScores[i];
    }
}

System.out.println("The lowest value in the array is: " + least);
Array code example

• Let's write a program that uses an array to store all the values
  – Compute least, greatest and average.
Example of an array of Strings

String peoplesNames[] = new String[10];
// The above line declares and allocates space for 10 Strings.

// The Strings then can have their values set like:

peoplesNames[0] = "Jerry Garcia";

peoplesNames[1] = "Bob Weir";

// etc.
Example of an array of Strings

• Just like you'd expect, not only String literals are allowed to be assigned, we could assign another String variable value to one of the elements of the peoplesNames array.

```java
String peoplesNames[] = new String[10];

for (int i=0; i < peoplesNames.length; i++)
{
    String userInputString = JOptionPane.showInputDialog("Enter a name");
    peoplesNames[i] = userInputString;
}
```
.length of an array VS. .length() method of String class

• Understand the difference between
  
  – .length the variable accessible for Arrays to tell us how many elements the array has
  – .length() the method in class String to find out how many characters are in the String.

• e.g.
  
  String csCourseNames[] = new String[20];
  // csCourseNames.length --- this is the length of the array which is 20 here.

  csCourseNames[0] = “Introduction to computer science I”;
  // csCourseNames[0].length() --- this is how many characters are in
  // the csCourseNames[0] --- which is 34
.length of an array VS. .length() method of String class

String csCourseNames[] = new String[20];
csCourseNames[0] = “Introduction to computer science I”;

System.out.println(“The number of elements in the csCourseNames array is:” +
csCourseNames.length);

System.out.println(“The number of characters in the 0 element of the array is:” +
csCourseNames[0].length());

// so what #'s will these print?
Arrays of Strings

• So, let’s write a for loop that will populate an array of Strings based on user input.

• What will the for loop’s control variable be initialized to?

• What will the condition to stop the loop be?

• What will we increment the control variable by?
Two dimensional arrays

• If, instead of a list of items, you wanted to store a grid of items, you could use a 2d array. A grid has rows and columns.

• A two dimensional array is declared with two pairs of brackets.

    int array2d [][] = new int[ 2 ][ 3 ];

    // This line creates a 2-by-3 array (2 rows, 3 columns)
Two dimensional arrays

```c
int array2d [ ] [ ] = { { 5, 7, 11 }, { 62, 3, 48 } };
```

to access the element at row 0, column 1, we use:

```c
array2d [ 0 ][ 1 ]  // this holds the value 7 in our example.
```
Two dimensional arrays

• A two dimensional array might be useful for something like keeping track of all the students in a class and their grades on each of the homeworks.
• So, each student could be represented by a number (0 to 16, for 17 students) and each homework represented by a number (0 to 9, for 10 hw's).
• Each student could be a row in the 2d array and each homework could be a column.
• The values stored in the array elements are the grade a particular student got on a particular homework.
Two dimensional arrays

```java
int student_hw_grades [][] = new int [ 17 ] [ 10 ];

// The outer loop will be for the 17 students and the inner loop will
// be for the 10 hw's.

for (int row = 0; row < array2d.length; row++)
{
    for (int col = 0; col < array2d[row].length; col++)
    {
        user_str = JOptionPane.showInputDialog(
            "Enter student " + row +
            "'s homework #" + col + "grade");

        student_hw_grades[ row ][ col ] = Integer.parseInt(user_str);
    }
}
// note the use of .length in the two loop conditions
```
Multi-dimensional arrays

• Not only can we have one dimensional, and two dimensional arrays but we can have $n$ dimensional arrays, where $n$ is any positive integer.

• Example when $n=3$:

```java
double temperatures[][][][] = new double [12][31][24];
```

take a guess as to what might be stored in this array and what the indices mean
Multi-dimensional arrays

- Unfortunately the first index must go from 0 to 11, the second from 0 to 30 and the third from 0 to 23.
- What if we wanted the index to represent exactly the month (1 to 12), day (1 to 31) and hour (0 to 23)?
- Is there anything we could do to this line?

```java
double temperatures[][][] = new double [12][31][24];
```
Math class

• Let's look at the Java API for the Math class.
• Specifically these methods:

  • abs – guess what this does.
  • cos, sin, tan
  • ceil – returns smallest whole number >= parameter.
  • pow – takes two parameters – raises first to second and returns the result.
  • random – returns random # in the range: [0.0, 1.0)
  • sqrt
A few more methods in the Math class

\textbf{max} ( x, y )

method that returns the larger of x and y

\textbf{min} ( x, y )

method that returns the smaller of x and y

There are versions of these methods that work for x and y being floats, doubles, ints and longs and return a result that is same type.
example calls to static methods in the Math class

double \( w = 5.1, \ z = 10.56, \ a, \ b, \ c; \)

\[
a = \text{Math.max}( \ w, \ z ); \quad \text{// what value would } a \text{ have?}
\]

\[
a = \text{Math. max}( \ z, \ w ); \quad \text{// what value would } a \text{ have, now?}
\]

\[
b = \text{Math. min}( \ z, \ w );
\]

\[
c = \text{Math. sqrt}( \ z );
\]
double rand_num;

rand_num = Math.random ( );
// what value might rand_num have after this line of code?

// is 0.34452 a possible value?

// is 2 a possible value?

// is -14.555423 a possible value?
random( ) method in the Math class

• **random( )** returns a **double** whose value is $\geq 0$ and $< 1$, but sometimes we want a random integer.

• How might we do that?
random( ) method in the Math class

• **random ( )** returns a **double** whose value is \( \geq 0 \) and \(< 1\), but sometimes we want a random integer.

• One way to do that is to first multiply the result by some integer to get a value that isn’t necessarily between 0 and 1. Then, cast this new value to an **int** by using the **(int)** cast operator.
random( ) method in the Math class

// example:
int some_random_int;
double some_random_dbl;

some_random_dbl = Math.random ( )  *  25;
// this will result in a value >= 0 and < 25.

some_random_int = (int) (Math.random ( )  *  25);

// what is the range of values for some_random_int here?
random( ) method in the Math class

```java
int random_card_value;
int random_card_suit;

random_card_value = 1 + (int) (Math.random ( ) * 13);
random_card_suit = (int) (Math.random ( ) * 4);
```

• Let’s put this code in a program and execute it.
random( ) method in the Math class

• What if I put the cast to int without using parentheses around the rest of the expression?

• e.g.

```java
    random_card_suit = (int) Math.random ( ) * 4;
```
random( ) method in the Math class

• What if I put the cast to int without using parentheses around the rest of the expression?

• e.g.

  random_card_suit = (int) Math.random( ) * 4;

• since the cast operator (int) has higher precedence than the multiplication operator *, it would be done first, which means what?
random( ) method in the Math class

• `random_card_suit = (int) Math.random ( ) * 4;`

• The `Math.random()` method call would return a double value and immediately this value would be cast to an int. Casting a double to an int causes the truncation of any decimal portion. Recall that the double that is returned by `Math.random()` is $\geq 0.0$ and $< 1.0$

• So, what's the possible values of `(int) Math.random()`?
random( ) method in the Math class

• (int) Math.random( ) would always be zero.