CS102

Introduction to data structures, algorithms, and object-oriented programming

DAY 5
Variations of the for loop § 3.4.1

Give three variations of for loops to print all the odd numbers between 1 and 21 (WARNING: THERE MAY BE ERRORS):

```java
for (int i = 1; i <= 21; i+=2) {
    System.out.println(i);
} // end for

for (int j = 1; j <= 21; j++) {
    if ( j % 2 ==1 ) {
        System.out.println(j);
    } // end if
} // end for

for (int k=0; k<=10; k = 2k+1) {
    System.out.println(2 * k + 1);
} // end for
```

/** Use a for loop to count 1, 3, ..., 21 */

/** Use a for loop to count 1... 21, but only print the numbers that are odd */

/** Use a for loop to count k=1...10 and * and print the numbers 2k + 1. */

THANKS TO ALISSA AND JORDY FOR FIGURING OUT THE ERROR HERE: THE K=2K+1 IN THE FOR PARENS MUST BE K = K+1. The error in DrJ was that 2k must be written 2 * k.
Control structures can contain other control structures. In particular, for loops are often nested.

```java
for ( int rowNumber = 1; rowNumber <= 12; rowNumber++ ) {
    // for each row, process all columns
    for (int n = 1; n <= 12; n++ ) {
        System.out.printf( "%4d", n * rowNumber );
        // print ints in 4-character columns; No newline
    }
    System.out.println(); // Add a newline
}
```
A while loop will repeat a statement only so long as a specified condition (boolean expression) remains true. A while loop has the form:

```java
while (boolean-expression) {  // 1. b-e
    inner block statements   // 2. b-e true
}
```

// 3. b-e false

1. boolean-expression (b-e) is evaluated
2. if b-e is true, evaluate inner block statements
3. if b-e is false, start evaluating statements after block
while flow of control diagram

while

The initialization is done outside the body of the loop.

A loop control variable is modified inside the body of the while. Eventually, the condition becomes false and the while loop ends.
while flow of control diagram

```
while

int sum = 0;
int n = 100;
int i = 1;

while (i < n) {
    sum += i;
    i++;
}
```
A while loop that stops when a positive integer is entered:

```java
int n;
while (true) {
    n = Integer.parseInt
        (JOptionPane.showInputDialog
            ("Enter a positive integer: "));
    if (n > 0) {
        break;
    }
    System.out.println("Try again.");
}
// end of while loop
JOptionPane.showMessageDialog(null,"The number is "+ n);
```
if Decision Statement

1. **if ...else**: “either-or” type statement, each with its own block of code.

2. **if** alone with a block of code, only runs block if the expression is true.

3. **if, else if, else if, ..., else**. Multi-way decision statement, each part with its own block of code.

4. **?:** Short form of if...else.

if and else are like cond in Racket. Only one clause in the group is executed and the rest are ignored. The else at the end is like that in the cond, sort of a default condition.
If..Else Flow of Control

Is condition true?

Yes
Do statement 1

No
Do statement 2

if
else
String line = JOptionPane.showInputDialog("Please enter a line of text: ");
int count = line.length();
println("Your input contains "+count+
((count>1) ? " letters.\n": " letter.\n"));

This code snippet first reads a line of text from the user and then prints the result. The ?: operator is embedded in the String that is printed. Notice that the ? and : are not consecutive...they are separated by the statement to be returned in the case the boolean expression is true.
break, continue, and return

Java provides a general method for breaking out of the middle of any loop. If you use a break statement inside a nested loop, it will only break out of innermost loop that contains the break, not out of the loop that contains the innermost loop.

A continue statement tells the computer to skip the rest of the current iteration of the loop. Instead of jumping out of the loop altogether, it jumps back to the beginning of the loop and continues with the next iteration.

Whenever a return statement is encountered in a method, control returns to the calling method.
This statement can be used in a loop to skip subsequent lines in the loop and go back to the loop start:

```java
int i = 0;
int sumEven = 0;
int count = 0;
while (true) {
    i = Integer.parseInt(JOptionPane.showInputDialog("Please enter 10 positive whole numbers");
    if (i < 0) {
        System.out.println("Oops, that was a negative number.");
        continue;
    }
    sumEven = sumEven + i;
    count++;
    if (count == 10) {
        break;
    }
}
System.out.println("The sum of the even numbers is "+ sumEven);
```
1. int i = 0;
2. int sumEven = 0;
3. int count = 0;
4. while ( true ) {
5.   i = Integer.parseInt
6.       (JOptionPane.showInputDialog
7.          ("Please enter 10 positive whole numbers");
8.   if ( i < 0) {
9.      System.out.println("Oops, that was a negative number.");
10.     continue;  // skip lines below if and go back to line 5
11.   }
12.   sumEven = sumEven + i; // i must be positive
13.   count++;
14.   if (count == 10) {
15.      break;  // need a way to break out of loop
16.   }
17. }
18. System.out.println("The sum of the even numbers is "+ sumEven);
A switch statement allows you to test the value of an expression and, depending on that value, to jump directly to some location within the switch statement.

The value of the expression can be one of the primitive integer types int, short, or byte. It can also be the primitive char type or a String.

The expression **cannot** be a double or float value.
switch ( n ) { // (Assume N is an integer variable or expression)
    case 1:
        System.out.println("The number is 1.");
        break;
    case 2:
    case 4:
    case 8:
        System.out.println("The number is 2, 4, or 8.");
        System.out.println("(That's a power of 2!)");
        break;
    case 3:
    case 6:
    case 9:
        System.out.println("The number is 3, 6, or 9.");
        System.out.println("(That's a multiple of 3!)");
        break;
    case 5:
        System.out.println("The number is 5.");
        break;
    default:
        System.out.println("The number is 7 or is outside the");
        System.out.println( " range 1 to 9.");
}
A data structure in which the items are arranged as a numbered sequence, so that each individual item can be referred to by its position number.

All the items in an array must be of the same type, and the numbering always starts at zero. An array is a list of variables, each accessible by the array name and position number of the variable.

An array is, technically, an object, so the process of creating one requires an instantiation with the keyword new.

The size of an array is fixed. It must be given when instantiating the array.
An array can be of any type and must first be declared:

String[ ] name;              // declaration of String array
int[ ] age;                       // declaration of int array
boolean[ ] leftHanded; // declaration of boolean array

Then the array must be instantiated:

name = new String[10];  // 10 memory locations for Strings
                        // each has value null to start
age = new int[5];          // 5 memory locations for int values
leftHanded = new boolean[100]; // 100 boolean locations

After instantiation, the specified number of locations will be created in memory and reserved for that type.
To put values into the array, you use the array name and position number to store a value at that position:

```java
name[5] = "Penny";
```

The length of an array is stored with the array as a field name accessible as, for example:

```java
name.length  // notice these are not method calls
age.length
```

Having access to the length of every array allows them to be easily used within a for loop to go through each element:

```java
// this for loop prints all elements in age array
for (i = 0; i < age.length; i++) {
    System.out.println( age[i] );
} // end for
```
Doesn't Java have any lists?

Java has class that is a cross between an array and a list.

ArrayLists are like arrays in that:

1. Items can be accessed directly by index number
2. When an ArrayList is declared, its data type is specified.

ArrayLists are like lists in that:

1. The size is dynamic, can be expanded and shortened. In other words, it only uses as much memory as there are items in the list.

The ArrayList is declared as follows:

```java
ArrayList<Integer> aLine = new ArrayList<Integer>();
```

Since ArrayLists can contain only data of object type, to make one that holds integers requires use of the Integer wrapper class.
for-each loop

Used to traverse array or collection of elements. The advantage of for-each loop is that it eliminates the possibility of bugs and makes the code more readable.

Syntax:  
\[
\text{for( data\_type variable : array )}\{
\}
\]

```java
class ForEachExample1{
    public static void main(String args[]){
        int arr[]={12,13,14,44};
        for(int i:arr){
            System.out.println(i);
        }
    }
}
```
for-each loop

Traversal of an ArrayList to add 1 to each element:

class ForEachExample1{
    public static void main(String args[]){

        ArrayList<Integer> intArrLst = new ArrayList<Integer>();

        for (int i = 0; i < 100; i++) {
            intArrLst.add(i);
        }
        for(Integer i : intArrLst){
            intArrLst.set(i, i+1);
        }
    }
}
2-dimensional arrays

Declaration and instantiation example:

```java
int[][] matrix = new matrix[10][5];
```

This line would create a matrix with 10 rows and 5 columns, initially all 0.

Often populated or printed in nested for loops:

```java
int[][] matrix = new matrix[10][5];
for (int row = 0; row < 5; row++) {
    for (int col = 0; col < 7; col++) {
        System.out.printf("%7d", A[row][col]);
    }
    System.out.println();
}
```
random numbers

The random method is a static member of the Math class. The call `Math.random()` produces a double between 0.0 and 1.0, inclusive. To use the `Math.random()` function to get a number between 1 and 10, for example, you would use the following call:

```java
int rNum = (int)(Math.random() * 10) + 1
```

The `(int)` operator truncates the double returned by `Math.random()` to produce an int. This type of operator is called a “cast”. The + 1 is there to avoid producing 0.
Type Conversion

- Changing a datum from one type into another.

- Explicit Conversion: Programmer uses a cast operation to perform the type conversion.

- Implicit Conversion: Compiler automatically inserts code to perform the type conversion.
Implicit Conversion from Narrow Types to Wider Types

As a convenience, rather than requiring the programmer to indicate a type conversion explicitly, the Java programming language performs an implicit conversion from the type of the expression to a type acceptable to its surrounding context.