Lesson 2 (cont'd)
Fundamental Programming Structures

Transcendental Consciousness is the simplest form of awareness

Conditional Logic (if)

if(*condition*) *statement*

```
if (condition) {
        statement1;
        statement2;
        …
}

if (sales >= target) {
        performance = “Satisfactory”;
        bonus = 100;
}
```

Conditional Logic (if … else)

if (*condition*) *statement1*
else *statement2*

```
if (sales >= target) {
        performance = “Satisfactory”;
        bonus = 100;
}
else  {
        performance = “Unsatisfactory”;
        bonus = 0;
}
```

Conditional Logic (Repeated else if)

```
if(sales >= 2*target) {
        performance = “Excellent”;
        bonus = 100;
}
else if (sales >= target) {
        performance = “Satisfactory”;
        bonus = 50;
}
else {
        performance = “Unsatisfactory”;
        bonus = 0;  // :(
}
```

Conditional Logic (Repeated else if)

An “else” is associated with nearest previous “if”. Therefore, these statements are read by the compiler as:

```
if(sales >= 2*target) {
        performance = “Excellent”;
        bonus = 100;
}
else {
        if (sales >= target {
                performance = “Satisfactory”;
                bonus = 500;
        }
        else {
                performance = “Unsatisfactory”;
                bonus = 0;
        }
}
```

While loop

The general form of a while loop is

while(*condition*) {*statements*}

where *condition* is a Boolean expression.

The general form of a do…while loop is

do {*statements*} while(*condition*)

Typically, do..while is used in place of while when it is necessary for the *statement* to execute at least once (even if *condition* is always false).

while loop

```
//while loop
while(balance < goal) {
        balance += payment;
        double interest = balance * interestRate  / 100;
        balance += interest;
        years++;
}
 System.out.println(years + " years");
```

do … while loop

```
//do..while loop
do{
        System.out.print("Payment amount? ");
        payment = sc.nextDouble();
        balance += payment;
        double interest = balance * interestRate  / 100;
        balance += interest;
        years++;
        System.out.println("Your balance including latest "+
                                        "payment and interest:
"+balance);

        System.out.println("Make another payment? (Y/N)");
        input = sc.next(); //Gets the next token.
}
while(input.equals("Y"));
```

While loop - break

Use the while(true) form when the *statement* requires processing before a condition can be evaluated. To exit the loop, use a break statement.

```
int sum = 0;
Scanner sc = new Scanner(System.in);
while(true) {
        System.out.print (“Enter a positive Number to sum: “);
        int value = sc.nextInt();
        if(value <= 0){
                break;
        }
        sum = sum + value;
}
System.out.println(“The sum is  ” + sum);
```

for loop

General form of the for loop:

for (*initialization*; *condition*; *increment*) *statement*

Infinite loop1

```
for ( ; ; ) ;
```

Infinite loop2

```
for(;true;);
```

means the same as

while (true) *statement*

for loop

```
for (int i = 0; i < max; i++) {
      //do something
}
```

Note: Since i is declared in the for expression, it cannot be referenced outside of the for block (scope – ‘where can it be seen’). If you need to use it outside the block, declare it outside the block.

```
int i;
for(i = 0; i < max; i++) {
      //do something
}
```

OR

```
int i=0;
for( ; i < max; i++) {
      //do something
}
```

for loop

More than one variable can be initialized, and more than one increment statement can be used; commas separate such statements.

```
for(int i = 1, j = max; i * j <=
balance; i++, j--) {
    //do something
}
```

Complex conditions are allowed in the condition slot:

```
for(int i = 0; (i+1) * value > min &&
i * value < max; i = i +2) {
    //do something
}
```

Two Control Variable - Example

```
for(i=0, j=10; i < j; i++, j--)
System.out.println("i and j: " + i + " " + j);
```

What is the output from the program:

Nested for loop

```
for (int i = 0; i < n; ++i){
        for (int j = 0; j < n; ++j){
                System.out.print("  *");
        }
        System.out.println();
}

//output for n = 5

* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

Nested for loop

```
for (int i = 0; i < n; ++i) {
      for (int j = 0; j <= i; ++j) {
            System.out.print(“*");
      }
      System.out.println();
}
 //output for n = 5
*
* *
* * *
* * * *
* * * * *
```

The switch Statement

The switch statement is a convenient shorthand for writing “if…else” statements, when the values being tested are byte, short, char, int, enum or string.

The break in each case ensures that only one case is executed. If you forget to insert the break, later cases will continue to be executed.

A default case should typically be provided, to handle all cases not specified in the case statements.

“Fallthrough behavior” when break statements are omitted: Cases are examined and, as soon as a match is found, the corresponding statement is executed, and all subsequent case statements are also executed, until a break is encountered. If no matches are found, then the default statement is executed if there is one.

General form of the switch statement

```
switch(val) { //val must be an int or char (or enum instance)
    case x:
        statement_x;
        break;
    case y:
        statement_y;
        break;
    …
    default:
        default_statement;
        break;
}
```

The switch Statement(Fallthrough behavior)

```
Scanner sc = new Scanner(System.in);
System.out.print("Pick an integer in the range 1..9");
int val = sc.nextInt();
System.out.println();
switch(val) {
        case 2:
        case 4:
        case 6:
        case 8:
                System.out.println("You chose an even number.");
                break;
        default:
                System.out.println("You chose an odd number.");
}
```

Main Point 3

Control flow is supported in Java via the *if..else*, *for*, *while*, *do..while*, *switch*, and *for each* language elements. *Loops are the CS analogue to the self-referral performance at the basis of all creation, whereas branching logic mirrors the tree-like hierarchy of natural laws that guide the activity in each layer of creation.*

Arrays

An array is a data structure that stores a collection of values of the same type and that supports *random access* of its elements (the element at position i in an array arr is retrieved using the syntax arr[i]).

Declaration of arrays	int[] arr;
Initialization of arrays	int[] arr = new int[100];
	int arr[] = new int[100];

100 cells, numbered 0 to 99, are created and by default, each cell contains the value 0.

All numeric arrays (for primitive types) are filled with 0 when initialized. String arrays (and arrays of objects of other kinds) are filled with the value null.

Arrays

Alternate notation for declaration: int arr[]; //less desirable

Setting values in an array arr[5] = 30;

Retrieving values in an array int positionFour = arr[4];

for each loop

new in J2SE5.0

Syntax:

```
for(variable : collection) statement
```

```
for(int x: arr) {
    System.out.println(x);
}
```

(As with ordinary for loops, the variable declaration can occur inside or outside the for expression)

The *collection* must either be an array or an instance of a class that implements the Iterable interface.

Array Initializers

When first created, can initialize an array like this (called an *array initializer*):

```
int[] somePrimes = {2, 3, 5, 7, 9, 11};
String[] names = {"Bob", "Harry", "Sue"};
```

double[] myList = {1.9, 2.9, 3.4, 3.5};

Application of Arrays-the split function of the String class

- Use `split` to break up a `String` into `tokens` based on a set of *delimiters.*
- The statement

```
String[] parsedVals =      s.split(",");
```

will split the String s into tokens, using "," as delimiter, and will place the tokens in the array `parsedVals`

Example:

```
String s = "hello,how,are,you,today";
String[] parsedVals =      s.split(",");
```

The elements of `parsedVals` are:

```
hello
how
are
you
today
```

- To treat more than one character as a delimiter for a `String`, list them in a String, using *bitwise or* (|) as a separator. To indicate a dot (.), you must use a double backslash (in Java's regular expression syntax, '\.' matches a dot; in Java, a backslash must be coded by "\\").

```
String t = "Hello,strings can be fun.They have many uses."
String[] result = t.split(",|\\.| "); //3 delimiters here
```

Here, `result` stores the array: [Hello, strings, can, be, fun, They, have, many, uses]

- NOTE: As of jdk1.4, the `split` method replaces the use of the class `StringTokenizer`

Array Methods : length

```
int x[]= new int[10];
System.out.println("Length of the Array : " + x.length);
```

Output : Length of the Array : 10

```
int nums[] = { 99, -10, 100123, 18, -978,
5623, 463, -9, 287, 49 };
System.out.println("Length of the Array : " +
nums.length);
```

Output : Length of the Array : 10

Array Copying and Sorting

```
Arrays.sort(arr)
System.arraycopy(Source, fromIndex, Target,
  toIndex, count)
int[] smallPrimes = { 2, 3, 5, 7, 11};
int[] copy = new int[5];
System.arraycopy(smallPrimes, 0, copy, 0, 5);
 // copy is [ 2, 3, 5, 7, 11 ]
 int[] smallPrimes = { 2, 4, 5, 7, 11};
int[] luckyNums = {350, 400, 150, 200, 250};
System.arraycopy(smallPrimes, 1, luckyNums, 3,
  2);
//luckyNums is now [350, 400, 150, 4, 5]
Arrays.sort(luckyNums);
//luckyNums is now [4, 5, 150, 350, 400]
```

Sorting Strings

When you used Arrays.sort on an array of Strings, the JVM automatically uses the compareTo method to compare Strings and to put them in alphabetical order.

- Example:

```
public static void main(String[] args) {
    String[] names = {"Steve", "Joe", "Alice", "Tom"};
    //sorts the array in place
    Arrays.sort(names);
    System.out.println(Arrays.toString(names));
}
```

//output

```
[Alice, Joe, Steve, Tom]
```

Multidimensional Arrays

Declaration: `int[][] twoD;`

```
int[][] twoDspecified = new int[3][5];
```

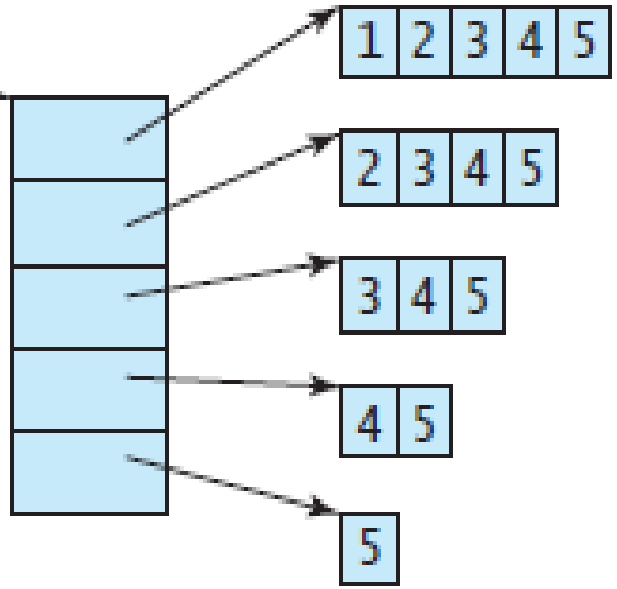
Array initializers

```
String[][] teams={
            {"Joe", "Bob", "Frank", "Steve"},
      {"Jon", "Tom","David","Ralph"},
      {"Tim", "Bev","Susan","Dennis"}
};
```

Ragged Arrays

- Each row in a two-dimensional array is itself an array. Thus, the rows can have different lengths. An array of this kind is known as a *ragged array*.

```
int[][] triangleArray = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```



Command line Parameters

The `main` method is designed to read input from the user when the program is executed.

```
class ParameterExample {
  public static void main(String[] args) {
      int len = 0;
      if(args != null) len = args.length;
      for(int i = 0; i < len; ++i) {
            System.out.println("position " + i + ": " +
  args[i]);
}
  }
}
```

Commandline parameters can be inserted into a Run Configuration in Eclipse. Right click class > Run As > Run Configurations, set name of configuration, add Program Arguments in Arguments tab.

Avoiding Costly Concatenation of Strings with StringBuilder

- **Example**: You are writing an application that will receive an unknown number of `Strings` as command-line arguments. These `Strings`, when pieced together, will form a sentence. Your job is to concatenate all these `Strings` and output to console the final sentence, with the correct sentence structure. (Since we are assuming just one sentence is formed, the only adjustments we need to make to the input are to put spaces between the words and a period at the end.)

First Try

```
public static void main(String[] args) {
   if(args == null || args.length == 0) {
      System.out.println("<no input>");
   }
   String finalSentence = "";
   len = args.length;
   for(int i = 0; i < len-1; ++i) {
      finalSentence += (args[i] + " ");
//inefficient
   }
   finalSentence += (args[len-1] + ".");
   System.out.println(finalSentence);
}
```

Problem: Concatenation becomes very slow with many arguments because each concatenation creates a new `String` (which requires allocating new memory for the new object), and compared to other steps, this is a costly operation.

Solution: `StringBuilder`

StringBuilder represents a "growable String" – can append characters and `Strings` without significant cost.

Note: `StringBuilder` is designed to be used for single-threaded applications – it is not thread-safe. This means that a single `StringBuilder` instance must not be shared between two or more competing threads. If multithreaded access is needed, a class with the same method names, `StringBuffer`, can be used, but it is less efficient in the single-threaded case.

Better Solution

```
public static void main(String[] args) {
    if(args == null || args.length == 0) {
        System.out.println("<no input>");
    }
    StringBuilder finalSentence = new StringBuilder();
    int len = args.length;
    for(int i = 0; i < len-1; ++i) {
        finalSentence.append(args[i]);
        finalSentence.append(" ");  //much more efficient
    }
    finalSentence.append(args[len-1]);
    finalSentence.append(".");

    // Convert the StringBuilder to a String at the end.
    String finalSentenceAsString = finalSentence.toString();
    System.out.println(finalSentenceAsString);
}
```

Main Point 4

- *Arrays* in Java support storage of multiple objects of the same type. Java supports multi-dimensional and ragged arrays; array copy and sort functions (accessible through the System and Arrays classes); and supports convenient forms of declaration and initialization. *All CS data structures mirror the "existence" aspect of consciousness – the nervous system – whereas the contents of these structures mirrors the "intelligence" aspect; the pure potentiality of a data structure is as if brought to life by filling it with real data.*

Connecting the Parts of Knowledge With the Wholeness of Knowledge

Role and utility of structures in Java

1. In Java, variables of primitive type can be combined using operators to form expressions, which may be evaluated to produce well-defined output values.
2. On a broader scale, objects in Java are "combined" by way of "messages" between objects, which collectively result in the behavior of a Java application.

---

3. Transcendental Consciousness *is the experience of the simplest and most abstract state of awareness which underlies all states of greater excitation.*
4. *Impulses within the Transcendental field: When consciousness knows itself it creates the lively impulses of pure knowledge within the field of silent Being.*
5. Wholeness moving within itself: *In Unity Consciousness, one observes that this unbounded silent quality of awareness is spontaneously present at all levels of action in the world, and not just relegated to the transcendental field.*