Lesson 3

* Code is generally executed top-down, however the control flow statements such as if-then-else break up the flow of execution
* The “==” operation is used for comparisons of primitives but the .equals() method is required when comparing objects
* .equalsIgnoreCase() is another string method
* If s1 and s2 are strings s1.compareTo(s2) returns a negative value if s1 comes before s2 in lexicographic ordering, zero if the two are equal and a positive number if s1 comes after s2
* Boolean operators:
  + && AND
  + || OR
  + ! Not
* Strings of inequalities must be joined by an AND operator ex. Min<med<max is invalid must be (min<med)&&(med<max)
* Java, like python allows for short circuit operations, for example using the || operator if the first expression evaluates to true it has no need to evaluate the second one which means it can skip division by zero errors
* To evaluate both sides of the expression regardless of the result of the first one use the bitwise operators & and |
* In Java we use else if instead of elif
* If your if statement only has one line of code in it you can omit the brace brackets

**The Ternary Operator**

int max = (n1 > n2) ? n1 : n2;

this code is equivalent to

int max =0;

if (n1 > n2)

max = n1;

else

max = n2;

place the if condition on the right side of the assignment operator followed by a “?”, then your true condition, a “:”, and your false condition

**The Switch Statement**

* When the switch statement is used Java will execute one of multiple statements or “cases” inside it
* The choice of case is determined by the *controlling expression* which must evaluate to a char, int, short or byte

Ex.

**switch (Controlling\_Expression) {**

**case Case\_Label\_1:**  
// your code for this case.  
**case Case\_Label\_2:**  
// your code for this case.

**}**

Rules for writing cases:

* Each case label must be of the same type as the controlling expression
* Case labels don’t need to be listed in order or span a complete interval, but each one may appear only once
* Each sequence of statements may be followed by a break statement

Ex.

int numberOfICeCreamFlavors;

switch (numberOfICeCreamFlavors) {

case 15:

System.out.println("Nice selection.");

break;

case 1:

System.out.println("I bet it's vanilla.");

break;

}

**The default case**

* If no cases match the switch statement the compiler can execute the default statement (written at the end of your cases) and can be used to output an error message or something
* It is important to note that once a case is executed, all lines will run until a break statement is reached. If there is no break statement the code will still run but after the case is finished the compiler will move on to another one.
* The default case is not necessary, but it is a good programming practice

**The do-while loop**

* Always executes a statement at least once and then loops

Ex. System.out.println("First do-while loop:");

countDown = 5;

do {

System.out.println("Hello");

countDown -= 1;

} while (countDown > 0);

Prints “Hello” five times

* The basic difference is it checks the condition *after* executing the statement as opposed to before

**The for loop**

* When initializing the for loop you can have multiple loop variables

Ex. for (int i = 0, j = 0; (i + j) < 6; i++) {

* I and J can also both be updated by separating them with commas at the end
* It is better programming practice to just have one loop variable and if you need another to declare it just outside the loop
* The second condition in the for loop (the Boolean expression) can only contain one expression
* Note the statement for (;;) is valid and will result in an infinite loop
* A common error is an *off-by-one error* which is when the loop executes one too many or one too few times

**The For-Each loop (advanced for loop)**

* Used to loop through an array or collection, this kind of loop can cycle through a collection even though the elements are not indexed
* An ArrayList is an example of a collection
* The following rules exist for for-each loops:
  + A loop variable cannot be assigned to. For example, in the example below we declared var as a loop variable. If you say var = anyObject that is incorrect
  + It is not possible to traverse two structures at once
  + The For-each loop can only be used for single element access
  + The For-each loop can only iterate forward by single steps

Ex.

for (type var: arr) {  
 body-of-loop  
}

Ex. Of iterating through a collection

Collection<String> myCollectionItem = new ArrayList<String> () ;

myCollectionItem.add("1 - H - Hydrogen") ;

myCollectionItem.add("2 - He - Helium") ;

myCollectionItem.add("3 - Li - Lithium") ;

myCollectionItem.add("4 - Be - Beryllium") ;

myCollectionItem.add("5 - B - Boron") ;

Iterator iter = myCollectionItem.iterator() ;

while (iter.hasNext()) {

System.out.println(iter.next()) ;

}

// Or using for-each :

System.out.println("for-each is used:") ;

for (String item: myCollectionItem)

System.out.println(item) ;

**Break identifier**

* The break statement by default is used to end any switch or loop and will end the innermost loop in the case of nested loops
* To end the outer loop in a nest you can use a *break identifier* prior to writing your loop

Ex.

endBothLoops: // this is a loop identifier.

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

System.out.println(i + " , " + j);

break endBothLoops; // use of the identifier

}

}

* Note that a break statement will end a switch or loop statement but to actually end a program use System.exit(0);
* The *exit* statement will immediately end the program as soon as it is invoked, it takes one integer as its argument and by tradition a zero is used to indicate the normal ending of a program.

**The Iterator Interface**

* As shown in the example above you can make an iterator object and use it in a while loop as opposed to using a for loop

**Assertions**

* Assertions will evaluate to either true or false and if it’s false the program will stop working and an error message will be displayed

Ex. assert database\_version == 1.2 ;

* You can write asserts like this that should be placed at the beginning of your code
* Normally, assertion checking is turned off, write

java –enableassertions ProgramName

to turn it on

**Random**

* In java you can import the random class to create and generate random number objects
* Ex. Of creating a random object: Random rand = new Random()
* rand.nextInt(n) will return a random integer from 0 to n-1

**Keywords**

* assert
* break
  + The break statement by default is used to end any switch or loop and will end the innermost loop in the case of nested loops
* Continue
  + The continue statement is used inside a loop to end the current iteration of the loop and *continue* to the following one
* do-while
  + the do-while loop is similar to the while loop except that it checks the loop condition after executing the “do” block rather than before. Code to be looped is placed inside the “do” block and then the while loop condition is placed at the end
* for-each
  + The for-each loop is an advanced for loop that is a newer edition to Java. It was created to loop through arrays or collections and is useful for traversing collections of elements that are not indexed. It can only be used for single element access and the collection can only be iterated through forward by single steps. To use it write for (type var: coll)
* If-else
* iterator
* java short-circuits
  + Java can evaluate compound Boolean expressions. To save time and compute power, if using an AND (&&) operator, if the left side is false it doesn’t matter what the right side is the expression will always be false so Java skips the right side. This saves time but also potentially skips over errors for example there may be a zero division error on the right side of the expression that is avoided but may be a future issue. Same thing for an OR but if the left side is true Java treats the whole expression as true.
* labeled break
  + The break statement by default is used to end any switch or loop and will end the innermost loop in the case of nested loops. To end the outer loop in a nest you can use a *break identifier* prior to writing your loop. Write a String as your label (for example “loops”) followed by a colon and then the loops on the next line. This labels the loops and by writing “break loops” it will end the inner AND outer loops.
* Switch
* ternary operator
* while