CSCI 151, Fall 2020

Java BootCamp

This is my version of a document that was originally written by Alexa Sharp.

1. Comments and Whitespace
    - Whitespace is significant in Python but not in Java. Tabs, returns, etc. are ignored by the Java compiler.
    - // This is a one-line comment.
    - /* This is a comment that takes up
      more than one line. */
    - There are also Javadoc comments that we will discuss in class.

2. Tokens
    - The following are *reserved words*. They have special meaning in Java and can't be used for anything else. Do not try to use one of these as a variable or class name; the error message you get may be quite cryptic. Here is a partial list of Java's reserved words:
      boolean, char, class, const, double, else, final, float, for, if, import, int, long, new, public, return, static, throws, void, while
    - Identifiers (for variable, method or class names) must begin with a letter, underscore or dollarsign and can only contain letters,digits, underscores and dollarsigns.
    - Java is case-sensitive: main() and Main() are different method names. This was also true in Python.
    - Most people capitalize the first letter of a class name (as in MyArrayList), and put the first letter of a method name in lowercase (as in mySort()).

3. Primitive Data Types
    - There are 8 primitive data types: boolean, char, byte, short, int, long, float, double. We will primarily use only 4 of these: boolean, char, int, double.
    - The boolean type has only 2 values: true and false. In Python they are capitalized; in Java they aren't.
    - The char type is not represented in Python. In Java it represents individual characters, such as 'a'. Note that the char 'a' is a very different object from the String "a".
    - ints are 32-bit positive or negative integers, which represent values up to about 2 billion. In Python there is no limit to the size of an int; in Java there is.
    - doubles are 64-bit floating point values.

4. Operators

- Increment operators: x += 5 adds 5 to the value of x. -=. *=, and so forth are similar. You can also use ++ to indicate "add 1". If x is 5, x++ changes x to 6.
- % is the modulus or remainder operator, as it is in Python. 24% 5 is the remainder when 24 is divided by 5, or 4.
- = (assign) and == (compare for equality) have the same meaning in Java as they do in Python. Be careful when you compare non-primitive types for equality, Java usually says whether the two values are stored at the same address, rather than whether they contain the same data.
- You create a new object of class Foo with `new Foo()`.
- `x instanceof Foo` is true if x is an object of class Foo.
- The dot (.) is used to access the individual fields of a class in Java, just as in Python. For example, Math.PI is the object PI from class Math.
- Array elements are accessed in Java as lists are in Python. If A is an array, A[0] is the first element of A, A[1] is the second, and so forth. Your program will crash if you try to access an element past the end of an array.
- Java has the same comparison operators as Python (<, <=, etc.)
- The boolean operators in Java are && (for *and*) and || (for *or*)

5. Punctuation
    - **Statements end with a semicolon**
    - Parentheses group operator expressions, as in 3*(4+5) and are also used in function calls: PrintPrimes(100);
    - Curly braces are used to group together a sequence of statements, such as the body of a loop:

```
while (x <  5 ) {
        System.out.println(x);
        x += 1;
}
```

6. Statements
    - An *empty statement* consists of just a semicolon
    - A *block* is any sequence of statements inside curly braces. Variables declared inside a block are not visible outside that block.
    - All variables must be *declared* before they can be used. A *declaration statement* tells Java the *type* of the variable; among other things this determines how much space needs to be allocated for the data that will be stored in the variable. For example, in the two statementsn

```
int a;
a = 5;
```

      the first statement declares variable a to be of type int, the second statement uses a by storing a value in it.

- Most other statements in Java work just like their analogs in Python:
  assignment statements store values in variables: x = 1;
  function calls invoke methods: main();
  s = smallest(A);
  if-statements provide selections, but there is no elif:

```
if (x < 10)
    System.out.println( "small" );
else if (x < 100)
    System.out.println( "medium" );
else
    System.out.println( "big" );
```

- Java has 3 kinds of loops: while-loops, do-while-loops, and for-loops

```
int x = 0;
while (x < 10) {
    System.out.println(x);
    x += 1;
}

do {
    int r;
    r = random.nextInt();
    System.out.println( r );
} while (r != 0);

for (int i = 0; i < 10; i++)
    System.out.println( i );
```

7. Methods (functions)
   - Here is the syntax of a method declaration

     [modifiers] <return-type><method-name>([parameters]) [throws-exceptions] {
     body
     }
     as in

```
public int foobar(int x, int y) {
    return x-y;
}|
```

     In the syntax template the items in square brackets are optional
   - To understand the modifiers, note that classes can be grouped in packages. It is possible for a class to be in one package and its subclass to be in a different package.
   - The possible *modifiers* are
     - public: the method can be accessed from outside the class
     - protected: The method can be accessed by anything in the method's package and

by any subclasses of the method's class.
- private: The method can be accessed only within its own class; it can't be inherited by subclasses
- static: There is only one instance of the method, belonging to the class. The method can be called without using an object of the class
- final: When this method is inherited it cannot be overwritten

- One possible return type for a method is *void*, meaning that the method doesn't return anything. Other than this, the return type must be a valid type and the system will check that you actually return a value of this type any time the method is called.
- Parameter declarations in methods look like any other declaration:
  <type> <parameter-name>
  For example
  ```
  void repeatChar(int n, char c) {
  }
  ```
- Part of the method header may indicate which exceptions are thrown by the method:
  ```
  public static readFile(String fname) throws IOException, FileNotFoundException {
  }
  ```
- Java uses call-by-value to pass values to parameters when a method is called. When a method is called, the arguments in the call are evaluated and their values are used as the initial values of the parameters in the method's header. There is no other linkage between the parameters and the arguments; changes to the parameters have no impact on the arguments. This means that a method cannot change its arguments. This is a loaded statement, however. Consider
  ```
  void changeArray( int [ ]A ) {
      A[0] = 23;
  }
  ```
  This method changes the *data stored in* array A, it doesn't change the *location where A is stored*. Arrays and all objects of all classes are actually represented by their locations (called pointers or references) rather than by their actual data. For primitive types this is easier:
  ```
  void changeValue( int x ) {
      x = 23;
  }
  ```
  This method has no effect. If, for example, we call it with
  ```
  int z = 5;
  changeValue(z);
  System.out.println(z);
  ```
  the value printed will be 5. The argument to a method, in this case 5, is only used to give an initial value to the method's parameter (here x). Changes to that parameter have no impact on the argument.

8. Objects
   - An *object* is an instance of a class. We might have a class List; any particular list is an object of that class. To create an object use the *new* construct, as in

Movie m = new Movie(“Boyhood”);

In this statement Movie is the name of a class. There is a constructor for this class that takes as an argument the name of the movie being constructed. This statement has two parts. To the left of the = operator we have a declaration of variable m: Movie m. To the right of the = operator we call the constructor:

new <class-name>(<arguments for the class constructor>)

- *null* is a value of every class; it represents “no object”. If you need to initialize a variable of some class type and you don’t have the data to construct an object of the class, you can always set the variable to null.
- The keyword *this* is analogous to Python’s *self*. It is used within a class declaration to refer to the current object.
- A variable of a class type is a *reference variable*. It holds the address of a place in memory that holds the data of an object of the class. The only way to make a new object is to use the *new* construct. For example, suppose we make 3 variables of class Alien and construct two Alien objects:

  Alien a1, a2, a3;
  a1 = new Alien(“bob”);
  a2 = new Alien(“hermione”);

  We could set a3 to a1:

  a3 = a1;

  and then both a3 and a1 would point to the same object in memory. If we then set

  a2 = a1

  we would no longer have anything pointing to the hermione alien, so that data would be lost.

9. Arrays
   - Python uses lists to store a sequence of elements. Java, and most other languages, uses arrays for this. There are two main differences between arrays and lists. One is that an array has a fixed size whereas any number of elements can be added to a Python list. The other difference is that all of the elements of a Java array must have the same type. In Lab 3 we will use arrays to make an *ArrayList* structure that functions like a Python list.
   - An *array* is an indexed sequence of values of the same type. For example, we might have an array of 10 ints, which are indexed 0 through 9. The type of the individual elements of the array is its *base type*. The length of the array is set when the array is created; this length cannot be changed.
   - An array type is <base type> [ ], as in int [ ]. Note that all arrays of the same base type (which differ only in their lengths) have the same type.
   - Arrays are similar to objects. An array variable is a reference variable; arrays are constructed with new. Here is a typical declaration of an array of 10 ints:

     int [] A = new int[10];

   - You can find the length of array A with A.length

- An exception is thrown (causing your program to crash if the exception isn't caught) if your program tries to use an improper index for an array, such as referring to A[n] if n is negative or greater than or equal to A.length.
- We can make multidimensional arrays: int[][] is the type of a rectangular array of ints. We think of this as the first index specifying the row of an element, the second index specifying the column. For example, the following line declares an array with 3 rows and 4 columns:

  int [][] M = new int[3][4];

  M[1][2] is the element of row 1, column 2.

10. Strings and Characters
    - *String* is a class, so string variables are reference variables. A literal string value is a sequence of letters inside double-quotation marks, as in "Marvin Krislov".
    - The empty string is "" (that is 2 successive double-quotes). This has length 0. Don't confuse the empty string with " ", which has length 1.
    - As in Python, the + operator does concatenation on strings: "Bob" + "by" is the string "Bobby". You can concatenate numbers to strings: "Star Trek" + 5 is the string "Star Trek 5"
    - The charAt( ) method of the String class is used to find the element of a string at a particular index: if S is "Oberlin" then S.charAt(2) is the char 'e'. There is also a substring(a, b) method where a is the starting index of the substring and b is the stopping index.
    - As in Python, strings in Java are immutable. Instead of changing the string, copy it and make the changes as you create the copy.
    - Do not use == to compare two strings; this just says whether the two strings are stored at the same location (it compares the reference values). The String class has an *equals* method. If s1 and s2 are strings, then s1.equals(s2) if s1 and s2 contain the same text.
    - You should become friendly with the String.format( ) method. This is a static method that takes a *format string* containing *placeholders* and a list of values, one per placeholder. The most common placeholders are %d, representing an integer, %s representing a string, and %f, representing a floating point value. The placeholders are replaced by the actual values and the resulting string is returned. For example,

      String.format("I won $%d %d times for a total of $%d.", v, n, v*n);

      If v has the value 50 and n has the value 3 this produces the string

      "I won $50 3 times for a total of $150."

      You can add to the placeholders *fieldwidth designators*: %3d represents an integer whose value takes up at least 3 characters; if the integer needs fewer characters the string is padded with blanks on the left. %-3d works the same way, only the string is padded on the right. %ns works the same way as %nd only with string values. Floats are a bit more complicated: the placeholder %7.2f says to use a fieldwidth of 7 for the floating point value, with 2 places after the decimal point.

11. Classes
    - We generally use the following template for class declarations:

```
<modifiers> class <class-name> [extends parent-class-name] {
        <fields>
        <constructors>
        <methods>
}
```

- The *modifiers* in a class declaration are the same as for methods: public, private, etc.
- The *fields* of a class declaration are the instance variables of the class.
- The fields, constructors and methods together are called the *members* of the class.
- The constructors are special methods that create objects of the class. Every constructor has the same name as the class. Constructors differ in their argument types. You can't have two different constructors for the same class that have the same argument types. Constructors don't specify return types. For example, a constructor for class Student might have header

  ```
  public Student( String name );
  ```
- Class Foo is usually contained in file Foo.java
- A subclass can refer to its parent class's constructor as *super*. This must be the first line of the subclass's constructor.
- All classes have a *default constructor*, which takes no arguments and does nothing. This is the constructor that will be in effect if you don't define a constructor.

12. Subclasses and Inheritance
    - All classes are descended from class Object.
    - The keyword *extends* is used in a class definition to give its parent class:

      ```
      private class BadAlien extends Alien {
          ....
      }
      ```
    - The subclass automatically *inherits* all of the fields and methods of its parent class. The subclass has the option of *overriding* the methods it inherits.
    - You can use *super* to access a method from the parent class that has been overridden in the subclass.

13. I/O
    - You probably want to import both java.io.* and java.util.* at the top of the file.
    - The easiest way to input data is through a Scanner object:

      ```
      Scanner console = new Scanner(System.in); // reads from keyboard input
      Scanner console = new Scanner("Here is a string of text"); // takes input from
                                                                 the string
      Scanner console = new Scanner( new File("foo.txt") ); // opens the text file
                                                            foo.txt
      ```

      console can then use methods of the Scanner class such as nextInt( ) or nextLine( ) or hasNextLine( )
    - You can use the following to print to the screen:

```
System.out.print( foo ); // prints the value of foo; stays on the same line
System.out.println( foo ); // prints foo and terminates the line so the
                           // next print will be on a new line
System.out.printf( s, args) // the same as
                                    System.out.print( String.format(s, args)  )
```

- Printing the *newline character* '\n' or a string containing this character will terminate the current line and start a new one.

14. The Math class
    - Math is part of the java.lang package. This is automatically imported, so you don't need an import directive at the top of your program to use things in Math.
    - Most of the methods and constants are static; you don't need to construct a Math object to use them.
    - The methods are what you would expect: Math.sin( ), Math.cos( ), Math.tan( ), Math.abs( ), Math.sqrt( ), etc.
    - Math.PI is the value of pi: 3.1415926535..

15. The Random class
    - Add the line
      ```
      import java.util.*;
      ```
      at the top of the file.
    - You need to construct a Random generator:
      ```
      Random rand = new Random( );
      ```
    - To get a new random value use one of the methods
      ```
      rand.nextInt( max );  // gives a value between 0 and max-1
      rand.nextFloat( );    // gives a random value between 0 and 1
      rand.nextBoolean( ); // randomly gives true or false
      ```