The following notes are meant to be a quick refresher on Java. It is not meant to be a means on its own to learn Java. For that you would need a lot more detail (for which the early textbook chapters is a good source). Furthermore, this is not an exhaustive listing of all of Java’s capabilities, as we still have many a fundamental concept to cover in this course, and in other courses down the road. Enjoy!

1 Comments & Whitespace

- blanks and tabs ignored by Java compiler
- 3 types of comments:
  a) // This is a comment
  b) /* This is also a comment */
     /* /* I can be nested */ */
     /* And I can span
     multiple lines */
  c) And last but not least, there are javadoc comments. See Weiss.

2 Tokens

2.1 Keywords

These are words reserved in Java for special use, so you cannot use them as your own identifiers (variable, method, & class names). A sampling is:

boolean, char, class, const, double, else, final, float, for, if, import, int, long, new, public, return, static, throws, void, while

2.2 Identifiers

- is a name for a variable/class/method, i.e. myVar, booYA, IHateArtichokesAndOlives,...
- must begin with a letter, underscore (_), or currency symbol ($)
- may contain any number of digits, letters, underscores, or currency symbols after the first character, i.e. _83yy$z
- Java is case-sensitive, i.e. IHateArtichokes is different from iHateArtichokes
- must not use the keywords as they already have special meaning
- convention: class names should begin with an uppercase letter, as in MyClass. Method names should begin with a lowercase letter, as in main.
2.3 Primitive Data Types

- There are eight primitive types: boolean, char, byte, short, int, long, float, double
- We typically use boolean, int, double, char
  - boolean: true, false (no 0s and 1s allowed)
  - char: 'a', 'b', 'c', ... (any 16-bit unicode character)
    * Can convert chars to ints, and vice-versa
      ex. char whatAmI = 97; // This is the character's 'a'! Ooooo
    * Use actual 'character', i.e. char = 'X';
    * Or, use escape characters: \t (tab), \n (newline), " ("), ('), \/ (\)
  - int: 32 bits. Ranges from -2147483648 to 2147483647
  - double 64 bits.
    * Use decimal point i.e., .10, 1.0, 1.0
    * Use scientific notation, i.e. 1e-6 1.23E02

2.4 Operators

- Increment / decrement
  ex. x = x+1 could be written as x++ or ++x
  ex. artichoke = 1; baloney = artichoke++;
    baloney gets the current value of artichoke (i.e. 1), then artichoke increments to 2
- Modulus (%) (the remainder operator)
  ex. 24 % 5 gives 4
- Before of = (assign) vs == (equals)!
  ex. int a = 5;
    while( a = 5 ) {a = 4; } // This will loop forever
- Object creation (new)
- Instanceof (see Weiss)
- Object access (.)
  ex. Foobar fb = new Foobar(); // Creates new object
    fb.fixMe(); // Accesses a method from the object
- Array element access([])
  ex. int[] myArray = {5, 4, 3, 2, 1};
    int temp = myArray[1]; // temp will have the value 4
2.5 Boolean Expressions

- true, false are the simplest boolean expressions
- can use relational operators to construct boolean expressions
  
  ex. i < 5, s.charAt(0) == s.charAt(6), s.equals( "trogdor" )
- can use boolean operators to combine boolean expressions
  
  ex. ( (n==1) || (n==2) )

2.6 Punctuation

- use ( ) for expressions and methods
- use ; for ending statements
- use { } for blocks of statements

3 Statements

- empty statement: ;
- block: any collection of statements inside { }
- expression: assignments, increment / decrement, method calls, object creation
- declaration: must tell Java about a variable before using it
  
  ex. int a; // Declaring the variable a to be an int
  a = 5; // Using the variable a
- assignment: to store a value in a variable
- method call
- object creation
- selection: if-else, if-else if
  
  - relations: <, >, <=, >=, ==, !=
  - logic: && (and), || (or), ! (not)
  - values: true, false
  
  ex. if( b ) // Or, for example if( x < 5 )
  System.out.println( "b is true!" );
  
  ex. if( b )
  System.out.println( "b is still true!" );
else
    System.out.println( "b is false. Figures." );

- repetition: while, do-while, for
  
ex. while( ImCrazy )
    Hospital.checkIn( "Alexa" );
  ex. do{
      a--;
    } while( a > 10 )
  ex. for( int i=1; i < 10; i++ ) {
      System.out.println( i );
  }

4 Methods

- syntax: modifiers <return-type> <method-name>( arguments ) [throws exceptions] { <stmts> }
  
ex. public int squareMe( int i ) { return (i * i); }

- modifiers are
  
  - public: allows method to be inherited and accessed from outside the class
  - protected: allows method to be inherited, but prohibits it from being accessed from outside the class
  - private: prohibits method from being inherited or accessed from outside the class
  - static: limits the number of instances of this method to one and allows access without an object
  - final: prohibits method from being overridden (inherited)

- return type can be void (returns no value), or any type.

- arguments are of the form <type> <varname>, <type> <varname>, ..., or no arguments at all

- before the body of the method, may possibly throw exceptions
  
ex. public static readFile( String fname ) throws FileNotFoundException {...}

- if the method parameters are primitive types then the value of the actual parameter is copied into the formal parameter, that is, the method cannot change the value of an actual parameter

- if the method parameters are object/complex/reference types, then the pointer to the actual parameter is copied into the formal parameter, and you can change the value of an actual parameter
overloading: writing multiple methods with the same name but different signature (i.e. different order of arguments, types of arguments, number of arguments, or any combination of the three)

5 Creating Objects & References

• to create an object (which is an instance of a class), use a constructor call:

  ex. Movie m = new Movie( "Revenge of the Nerds" );

• null is the value that represents no object

• the keyword this represents a reference to the current object

  ex. class Alien
  {
    private String planet; // name of home planet
    private Alien friend; // friend of current planet
    public Alien( String planet ) { this.planet = planet; }
  }

• Java does not allow you to “store” the actual object in a variable; instead, you need to use a variable of the object’s type that stores the address of this object. This kind of variable is called a reference variable.

  ex. Aardvark a = new Aardvark();

  – a is a variable of type Aardvark
  – a stores the address of a newly created Aardvark
  – if you print a, you’ll see the address value (Aardvark@....), not anything actually *useful*, God forbid.

• changing the contents of a reference variable means storing the address of a different object:

  ex. Alien bossAlien;
  Alien a1 = new Alien();
  Alien a2 = new Alien();
  bossAlien = a1; // boss now contains the address of a1

• passing an object to a method really means passing a reference to that object

• returning an object from a method really means returning a reference to that object
6 Arrays

- arrays are objects — must use `new`!
- all elements of array must have same type
- indexed from 0. Indices must be integers, or be expressions that evaluate to integers
- syntax to declare: `<type>[] <varname>;` or `<type> <varname>[];
  
  ex. Aliens[] starsInMenInBlackII; // An array of Aliens
- syntax to assign: `<varname> = new <type>[size];`
  
  ex. starsInMenInBlackII = new Aliens[ 3 ];
  
  int[] someArray = new int[ 8 ];
- can find length easily using `<varname>.length`
  
  ex. starsInMenInBlackII.length is 3.
- arrays are 0-indexed:
  
  ex. someArray[ 0 ] is the first element of someArray.
  
  ex. someArray[ someArray.length -1 ] is the last element of someArray.
- can make arrays of objects (i.e. arrays of references), and multidimensional arrays

7 Strings and Characters

- strings are reference types (use `String` class)
- string literal: "yodelayheehoo" — is an instance of class `String`
- empty string: ""
- concatenation is easy:
  
  ex. "you make me complete" + "ly miserable" → "you make me completely miserable"
- even concatenating non-strings is easy:
  
  ex. "mumbo number " + 5 → "mumbo number 5"
- must put `String` on one line. One!
- multiple `String` constructors:

  String s0 = "yo!"; // creates a string literal
  String s1 = new String(); // creates an empty string
String s2 = new String( "whassaaaap" ); // creates string of "whassaaaap"

- Strings are immutable – once created, they cannot change! But you can copy them...
- strings indexed from 0
  
  ex. String s = new String( "Ra ra Rhasputin" );
  char c = s.charAt( 1 ); // c will hold the character 'a'
  String t = s.substring( 0, 5 ); // t holds "Ra ra"

- Do not use == to compare strings, use s1.equals(s2). Yes, sometimes == will work, but not all the time, so just stick with the equals, okay!?!

8 Classes

- blueprint / mold for creating objects
- syntax: <modifiers> class <classname> { <fields>; <constructors>; <methods>; }
- modifiers same as for methods
- fields and methods are called members
- fields represent properties of a class, methods are for accessing and modifying these properties
- fields get default values of their respective types
- every method can see any other method in the same class in any order
- constructors return a reference to the newly created object – they do not have a return type
  
  - syntax: <modifiers> <classname>( <arguments> ) { <body> }
  - every class has at least one constructor, even if you don’t write one. The default is <classname>() {}, a.k.a. the empty constructor.
  - if you want to call another class constructor or the super’s constructor in your constructor, you must do it in the first line.
  - otherwise, the super’s constructor is automatically called (the empty one)

ex. class Movie {
  private String name;
  public Movie( String s ) { name = s; } \\ This is the constructor
  public toString() { return name; }
  public addStudioToName( String studio ) { name = studio + "'s " + name }  
}
9 Inheritance

• all classes inherit from class Object
• use the keyword extends to inherit from a class
• subclass extends a superclass to take advantage of superclass’s existing functionality
• subclass can extend from at most one superclass
• public and protected methods and fields are inherited unless overridden
• private fields and methods do technically do not inherit, but may be indirectly accessed using a public/protected member which does inherit
• private fields and methods are accessed from the same class that they are called from
• use super to access a superclass member, unless that member is not visible (i.e. private)

10 Useful Classes

10.1 The Math Class
• need to add import java.lang.*; at top of file.
• contains functions like abs, sqrt, pow, ...

10.2 The Random Class
• need to add import java.util.*; at top of file
• methods functions like nextInt(n)

10.3 The Scanner Class
• need to add import java.util.*; at top of file
• can construct for the console or from a File, or even from a String
  ex. Scanner input = new Scanner( new File( "input.txt" ) );
      Scanner console = new Scanner(System.in);
      Scanner console = new Scanner( "it’s just string" );
• method nextLine(), nextInt()