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. Furthermore, this is not an exhaustive listing of all of Java's capabilities, as we still have fundamental concepts to cover in this course and others down the road. Enjoy!

1 Comments & Whitespace

blanks and tabs ignored by Java compiler

3 types of comments:

i. `// This is a comment`

ii.
```
/* This is also a comment */
/* /* I can be nested */ */
/* And I can span
multiple lines */
```

iii. And last but not least, there is javadoc comments. See Weiss Ch 3.

2 Tokens

2.1 Reserved Words & Tokens

See p990 (Appendix 1) in Savitch, or find a list in most Java books. A sampling is:
`while, do, if, else, case, switch, class, abstract, extends, implements, this, throws, boolean, new, null, public, private...`

2.2 Identifiers

- is a name/variable that refers to something, i.e. `myVar, booYA, IHateMushroomsAndOlives,...`
- must begin with 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. `IHateMushrooms` is different from `ihatemushrooms`
- must not use reserved words as they already have special meaning

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`: are unicode characters (16-bits)
    - can convert `char`s to `int`s, and vice-versa
      ex. `char whatAmI = 97;  // This is the character 'a'! Oooo`
    i. use actual 'character', i.e. `char c = 'X';`

ii. or, use escape characters: `\t` (tab), `\n` (newline), `\"` ("), `\'` ('),\\ (\)

- `int`: 32 bits. Ranges from -2147483648 to 2147483647
- `double`: 64 bits.
  i. use decimal point (.), i.e. .10 1. 1.0
  ii. 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 (%) (remainder operator)

  ex. 24 % 5 gives 4

- beware of = (assign) vs == (equals)!

  ex.
  ```
  int a = 5
  while( a = 5 ) { a = 4; }     // This will loop forever
  ```

- instanceof (see Weiss p68)
- object creation (new)
- 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
  ```

- casting (())

  ex.
  ```
  int b = (int)5.6;             // b will have the value 5
  ```

2.5 Strings

- more on strings in a later section

2.6 Punctuation

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

3 Statements

- empty statement: ;
- block: 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, switch
  - 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 )
        Hostpital.checkIn( "Alexa" );
```

```
ex. do{
        a--;
    } while( a > 10 )
```

```
ex. for( int i = 1; i < 10; i++ ) {
        System.out.println( i );
    }
```

- others: `break, continue, return, synchronized, throw, try`

4 Methods

- syntax: `modifiers returntype name( arguments ) [throws exceptions] { statements; }`

  ex. `public int  squareMe( int i ) { return (i * i); }`

- modifiers are
  - `public`: allows method to be inherited and accessed from outside the class
  - `private`: prohibits method from being inherited or accessed from outside the class
  - `protected`: allows method to be inherited, but prohibits it from being accessed from outside the class

  - `static`: limits the number of instances of this method to one and allows access without an object
  - `final`: cannot override (inherit) this method
  - ⟨none⟩: what do you think? It does nothing! (same as `public`)
- return type can be `void` (returns no value), or any type.
- arguments are of the form `type1 var1, type2 var2,...`, or no arguments at all
- exceptions follow later in the course
- Java is call-by-value:
  - value of actual parameter is copied into the formal parameter
  - the method *cannot* 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 Classes

- blueprint/mold for creating objects
- syntax: `modifiers class name { 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 "zero":

  ints: 0, doubles: 0.0, chars: ASCII code 0 or `\u0000`, booleans: `false`

  all reference variables: `null`, Strings (which are objects): `null`
- every method can see any other method in the same class in any order
- constructors return a reference to a newly created object
  - syntax: `modifiers name( arguments ) { body }`
  - every class has at least one constructor, even if you don't write one. The default is `name() {}`, aka 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 n )                { name = n; }  \\ This is the constructor
    public  toString()                       { return name; }
    public  addStudioToName( String studio ) { name = studio + "'s "+ name; }
}
```

6 Creating Objects & References

- to create an object (which is an instance of a class), use a constuctor call:
  `new className( arguments )`

  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 alien
    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

7 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[] name;
type   name[];
```

  ex.
```
Aliens[] starsInMenInBlackII;     // An array of Aliens
Aliens   starsInMenInBlackII[];   // The exact same array
```

- syntax to assign:
```
name = new type[size];
```

ex.
```
starsInMenInBlackII = new Aliens[ 3 ];
int[]     someArray = new int[ 8 ];
```

* Note that the arrays above will be initialized with their default values: `null` in the case of the first example, and 0 in the case of the second.

- syntax to initialize:
```
name = { value, value, ..., value };
```

ex.
```
starsInMenInBlackII = { new Alien( "Mars" ),
                        new Alien( "Pluto" ),
                        new Alien( "RD93t" ) };
someArray           = { 5, 6, 7, 8, 9, 10, 11, 12 };
```

- can find length easily using `arrayname.length`

  ex. `starsInMenInBlackII.length` is 3.

- can make arrays of objects (i.e. arrays of references), and multidimensional arrays
  - an array of objects is really an array of references... default values are `null`

8 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 members (ha ha) technically do not inherit, but may be indirectly accessed using a public/protected member which does inherit
- private members 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)
- we will talk about inheritance in detail next week

9 Strings and Characters

- strings are technically objects (use String class)
- string literal: `"yodelayheehoo"` — is an instance of class `String`
- empty string: `""`
- concatenation is easy:
  `"you make me complete"` + `"ly miserable"` $\longrightarrow$ `"you make me completely miserable"`
- even concatenating non-`Strings` is easy: `"stuff"` + primitive type promotes all to `String`:
  `"mumbo number "` + 5 $\longrightarrow$ `"mumbo number 5"`
- must put `String` on *one* line. One!
- multiple `String` constructors:

```
String s0  = "yo!";                      // creates a string literal
String s1  = new String();               // create empty string
String s2  = new String( "whassup G?" ); // create string of "whassup G?"
char[] tmp = { 'a', 'b', 'c' };          // uses an array
String s3  = new String( tmp );          // create string from chars
```

- Strings are immutable – once created, they cannot change! see `StringBuffer` class for mutable strings
- strings indexed from 0

```
String s = new String( "Ra ra Rhasputin, lover of the Russian queen" );
char   c = s[ 7 ];           // c will hold the character 'h', not 'R'
```

- 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`, ok!? Why?

10 Character Set

10.1 ASCII

- 8-bit encoding (i.e. $2^8 = 256$ encodings)
- Java will automatically understand ascii text (i.e. just use the ascii value without any escape sequences)

10.2 Unicode

- 16-bit encoding (i.e. 2^{16} encodings)
- can encode almost every character in existence, from any language
- see your text (Appendix 10) or http://www.unicode.org
- use any character in your program using `\uxxxx`

11 Packages

- a package is a collection of .java source files (and other packages and interfaces) that are grouped together in the same directory or folder
- useful packages to you:
  - `java.lang`: contains classes that you can automatically use:
    - `Integer, Boolean, Double, Float`, etc.: wrapper classes for the corresponding `int, bool,` etc.
    - `Math`: contains functions like `abs, sqrt, pow`, ...
    - `String, StringBuffer`
    - `Sytem`
    - `Exception, Throwable`
  - `java.io`: contains classes for doing input and output, including I/O to and from the console or a file
  - `java.applet`: used for applets
  - `java.awt`: used in constructing GUIs
  - `javax.swing`: contains the more modern classes for constructing GUIs

  - `java.util`: contains classes for dates, calendars, locales, random numbers, classes for vectors, sets, lists, maps, ...

- to use these packages, you use an `import` statement:
  ```
  import java.io.*;
  ```
- go to the sun API page at http://java.sun.com/j2se/1.3/docs/api/index.html to get a detail listing of what is in each package and how to use them.

12 Other Resources

12.1 On the Web

- see Java Resources link on course webpage `http://www.cs.cornell.edu/Courses/cs211/2002su`

12.2 Books

- *Java: An Introduction to Computer Science and Programming*, Walter Savitch
- *Data Structures & Problem Solving Using Java*, Mark Allen Weiss
- *Java in a Nutshell*, David Flanagan