

The following notes are meant to be a quick cheat sheet for Java. It is *not* meant to be a means on its own to learn Java or this course. For that you should look at your textbook and the sample code from class. 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. These notes will be updated as we learn more syntax. Enjoy!

1 Comments & Whitespace

- blanks and tabs ignored by Java compiler
- 2 types of comments:

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

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). They are listed on p17 of your text, but 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`. Variable and method names should begin with a lowercase letter, as in `main`. Class constants should be all uppercase, as in `WIDTH`.
-

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'! 00000`
 - * 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`

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
- integer division
 - ex. `10/4` gives 2, not 2.5

2.5 Punctuation

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

3 Statements

- 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
- for-loops

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

4 Methods

- syntax: `public static <return-type> <method-name>(arguments) { <statements> }`

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

- 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
- 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
- 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 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!
-

6 Useful Classes

6.1 The Math Class

- contains functions like `abs`, `sqrt`, `pow`, `sin`...
- contains constants like `Math.PI`, `Math.E`,...

6.2 The Random Class

- need to add `import java.util.*;` at top of file
- methods functions like `nextInt(n)`,...

6.3 The Scanner Class

- need to add `import java.util.*;` at top off 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()`, `nextDouble()`,...

6.4 The File Class

- need to add `import java.io.*;` at top of file
-