Quiz Review
Valid Java Identifiers...

- can contain letters, numbers, the ‘_’ and ‘$’ characters
- cannot start with a number
- cannot be a reserved keyword
  - reserved keyword = int, public, static, etc

Which of these are valid java identifiers?

easy$  D15R3sPeC7  20nuggets  _underscore  @param  final
Naming Conventions

---

- Variables/Methods
  - start with lowercase, use uppercase to denote new word
  - ex: x_start, numOfElems, drawS(), getRandomNumber()

- Constants (public static final)
  - all uppercase, use underscore to denote new word
  - ex: CHAR_WIDTH, TOTAL_DELAY

- Class
  - start with uppercase, use uppercase to denote new word
  - ex: CS11Turtle
Operators

---

Basic operators:
- ‘+’, ‘-’, ‘/’, ‘*’, ‘%’
PEMDAS Rule Applies

Incrementing/Decrementing:
- x++ -> x = x+1
- x-- -> x = x-1
- x+=3 -> x = x+3
- x/=3 -> x = x/3
Post-Increment vs. Pre-Increment

There is a difference between `x++` and `++x`

- `x++` is called the post-increment
  - this increments `x` after the statement ends
- `++x` is called the pre-increment
  - this increments `x` before the statement begins

This works the same for the decrement operator
(`x--` and `--x`)
EXAMPLE

---

```java
int x = 50;
System.out.println(x); // prints out 50
System.out.println(x++); // prints out 50
System.out.println(x); // prints out 51

int y = 50;
System.out.println(y); // prints out 50
System.out.println(++y); // prints out 51
System.out.println(y); // prints out 51
```
Printing Integers (Where Things Get Weird)

- “int + int” IN PARENTHESES are interpreted as MATH
- “int + int” BEFORE STRINGS are interpreted as MATH
- “int + int” AFTER STRINGS are interpreted as CONCATENATION

Example where a = 1, b = 2

- System.out.println(a + b + " hi"); // prints “3 hi”
- System.out.println("hi "+ a + b); // prints “hi 12”
- System.out.println("hi "+ (a + b)); // prints “hi 3”
Boolean Logic

Basic Operators:
- <, <=, >, >=, ==, !=

Boolean Math:
- && (AND)
- || (OR)
- ! (NOT)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A AND B</th>
<th>A OR B</th>
<th>NOT A</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
Short Circuiting

---

false && (some boolean) will always result in false, likewise, true || (some boolean) will always result in true.

You can use this to “short circuit” the logic.

EX: false && (false || (true && false || true))

Since false is an operand for the && operator, you know the whole expression will be false.
True / False?

---

(20 < 50)
true

(20 < 50) && (77 < 30)
false

(20 < 50) || (77 < 30)
true

( (56*9 % 2 != 0) && (!(40 < 80) || true) )
true
Control Flow (if and else statements)

What it looks like:

```java
if (condition1) {
    // executes if condition1 equates to true
    // otherwise go to the next else keyword
} else if (condition2) {
    // executes if condition2 expression equates to true
    // otherwise go to the next else keyword
} else {
    // code
}
```
Conditionals/Looping WITHOUT Curly Braces (BAD)

---

- conditionals and loops will only run line IMMEDIATELY AFTER

Example:

```
int x = 0;
if (true)
    if (false)
        x = 4;
    x = 10;

// x is equal to 10 at this point!
```
Control Flow (switch-case)

Works the same as a bunch of if-else statements, but syntactically simpler. What it looks like:

```java
switch (var) {
    case (value):
        // executes code here if var == value
        break;  // exits out of the switch block
    case (value2):
        // executes code here if var == value2
        break;  // exits out of the switch block
    default:
        // executes code here if none of the values matched
        break;
}
```
Control Flow (Loops)

- not sure how many times the loop will run? Use “while” or “do-while”
- Otherwise, use “for”
- Watch for infinite loops

Ex. while (true) {
    
}

- Avoid infinite loops by making sure the loop condition changes
Break and Continue

---

Two statements you should be familiar with when working with loops:

- break;  // breaks the loop and continues
  // executing code after the loop
- continue;  // breaks only the current iteration of
  // the loop, and goes to the next iteration
  // if the conditions are still satisfied
Do

jump to statement without condition

Statements

Break  Continue

While

Pass  Fail

Condition

End

Do-While Loop
The Scanner object (imported via ‘import java.util.Scanner;’) is a way to read user input from your program. Call its constructor like so:

- \texttt{Scanner scnr = new Scanner(System.in);}

Read in the values you are looking for from command line:
- \texttt{scnr.nextInt()} to read in the next int the user types
- \texttt{scnr.nextLine()} gets all the text until enter has been pressed

***USE FOR PA3***
Unix commands

---

Key commands:
- cd = change directory
  - ‘cd foo’ will change directory to the foo directory
- mkdir = make directory
  - ‘mkdir bar’ will create a new directory named bar
- ls = list contents of directory
- pwd = prints working directory
- cp = copy file
  - ‘cp foobar foo/bar’ will copy foobar to the directory foo, and rename it bar
- mv = move file
  - ‘mv foo bar’ will move foo to directory bar
- rm = remove file
  - ‘rm foobar.java’ will remove the file foobar.java
Absolute Path vs Relative Path

- Absolute path to Landuse is /D/Data/Shapefiles/Landuse

- Relative path to Landuse from Data is Shapefiles/Landuse

- ".." refers to previous directory
- "." refers to current directory
- "~" refers to home directory NOT "/home"