WELCOME TO CIS 1068!

Instructor: Alexander Yates
(yates {@} temple {.} edu)
TA: Le Shu
(tud08127 {@} temple {.} edu)
This is a difficult course!
How to do well in this course

- GO TO CLASS
- DO THE READING
- Keep up with the assignments
  - The course material is cumulative
  - From a former student: “Procrastination will eventually come around to bite you in the ass!”
- If you don’t understand something, ask questions (especially “WHY?”).
We’re here to help

- Ask questions
- Email us, come by our office hours
- We may be nerdy, but we’re not too scary
What is computer science?

- computers?
- science?
- programming?
- late lonely nights in front of the computer?

**ALGORITHMIC THINKING**

алго́ритм:  
a step-by-step procedure for solving a problem or accomplishing some end especially by a computer

- How does that relate to programming?
Just like Legos...
2 Things to Learn in this Class

1. The Java programming language
   - This is like learning a foreign language:
     - New vocabulary
     - New rules (syntax, grammar) for putting the foreign words together
   - But with a but more math

2. Using Java to solve computational problems
   - This is like learning to solve puzzles
   - This is often the hardest (but also most fun) part!
A Simple Model of a Computer
The 6 Parts
Random Access Memory (RAM, or just memory)

- Volatile storage
  - Each card can store up to a few gigabytes of data
  - Storage only lasts while computer has power

- Organization: Matrix
  - Some number of columns, either 32 or 64
  - Each cell stores a binary integer (bit): a 1 or a 0
  - Each row has an address, or row number

<table>
<thead>
<tr>
<th>Row</th>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Bit 3</th>
<th>Bit 4</th>
<th>Bit 5</th>
<th>Bit 6</th>
<th>Bit 7</th>
<th>...</th>
<th>Bit 64</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Central Processing Unit (CPU)

- Little chip, big deal: it does all the **computing**
- Think of it as a place where some instructions are stored, and carried out 1 by 1
- Instructions can do basic math, and copy data and move it around (and that’s pretty much it)

<table>
<thead>
<tr>
<th>Copy 1 to 7</th>
<th><em>Start</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Add 4 and 7, store in 8</td>
<td></td>
</tr>
<tr>
<td>Multiply 1 and 8</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Hard Disk (Disk, Hard Drive)

- Permanent (more or less) storage
  - Like memory, stores binary data
  - Unlike memory, it doesn’t forget anything when power is turned off.
- Organization: files, blocks, and cylinders
  - Complicated, we won’t get into it
  - The important point for us: 
    **It’s slow!**
    (compared to memory)
Other computer components

- Monitor
- Keyboard
- Mouse
- Graphics chip
- Serial bus
- Fan
- Disk controller
- …

All sorts of things, but for this course, we’ll just concentrate on a few of them.
Your first Java program!

- Java is a *programming language*.

```java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

- What does this code *output* (print to the user) when you *run* (execute) it?
Java code and machine code

Notice: Java instructions don’t look like CPU instructions!

- CPU instructions (aka, machine code)
  - Written in binary
  - Easy and fast for machines to work with
  - Hard for programmers to work with

- Java code
  - Written in Java, a “high-level” (English-like) programming language
  - We use the Java compiler and virtual machine to translate from Java to machine code
Compiling a program

- Before you run a program, you must compile it.
- **compiler**: Translates a computer program written in one language (i.e., Java) to another language (i.e., byte code)

Compile (javac)  Execute (java)

source code (Hello.java)  byte code (Hello.class)  output

output: Hello, world! Press any key to continue.
The Java Virtual Machine (JVM, or VM)

- The Java Virtual Machine executes byte code
  - Use the “java” command to execute it
  - It only understands byte code (“.class” files)

- The VM makes Java a bit different from older programming languages (C, C++)
  - It’s an extra step; compilers for other languages directly produce machine code
  - It’s slower
  - But it allows the same byte code to run on any machine with a VM
Program execution

- The *output* is printed to the *console*.
- Some editors pop up the console as another window.

![Console output](image-url)
public class Hello2 {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
        System.out.println();
        System.out.println("This program produces");
        System.out.println("four lines of output");
    }
}
Writing your own Java programs

```java
class <name> {
    public static void main(String[] args) {
        <statement>;
        <statement>;
        ...
        <statement>;
    }
}
```

- Every executable Java program consists of a **class**
  - that contains a **method** called `main`
    - that contains the **statements** (commands) to be executed
Syntax

- **syntax**: The set of legal structures and commands that can be used.

- Examples:
  - Every basic statement ends with a semi-colon.
  - The contents of a class occur between curly braces.
Syntax Errors

- **syntax error**: A problem in the structure of a program.

```java
1 public class Hello {
2     public static void main(String[] args) {
3         System.out.println("Hello, world!");
4     }
5 }
```

```
2 errors found:
File: Hello.java [line: 2]
Error: Hello.java:2: <identifier> expected
File: Hello.java [line: 3]
Error: Hello.java:3: ';' expected
```
Finding syntax errors

- Error messages do not always help us understand what is wrong:

  File: Hello.java  [line: 2]
  Error: Hello.java:2: <identifier> expected

  pooblic static void main(String[] args) {

  - Why can’t the computer just say “You misspelled ‘public’”?
First lesson in computer science

- Computers are stupid.
- Computers can’t read minds.
- Computers don’t make mistakes.
- If the computer is not doing what you want, it’s because YOU made a mistake.
More on syntax errors

- **Java is case-sensitive**
  - *Hello* and *hello* are not the same

```java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

<table>
<thead>
<tr>
<th>compiler output:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 error found:</td>
</tr>
<tr>
<td>File: Hello.java  [line: 1]</td>
</tr>
<tr>
<td>Error: Hello.java:1: class, interface, or enum expected</td>
</tr>
</tbody>
</table>
System.out.println

- **System.out.println**: A statement to print a line of output to the console.
  - pronounced “*print-linn*”

- **Two ways to use System.out.println**:  
  System.out.println("<message>");  
  - Prints the given message as a line of text to the console.

  System.out.println();  
  - Prints a blank line to the console.
Strings

- **string**: A sequence of text characters.
  - Start and end with quotation mark characters

**Examples:**

"hello"
"This is a string"
"This, too, is a string. It can be very long!"
Details about strings

- A string may not span across multiple lines.
  "This is not a legal string."

- A string may not contain a " character.
  - The ‘ character is okay.
    "This is not a "legal" string either."
    "This is 'okay' though."

- This begs the question…
Escape sequences

- A string can represent certain special characters by preceding them with a backslash \ (this is called an escape sequence).
  - \t tab character
  - \n newline character
  - \" quotation mark character

- Example:
  System.out.println("Hello!\nHow are \"you\"?" Tone);

- Output:
  Hello!
  How are "you"?

- This begs another question…
Questions

What is the output of each of the following `println` statements?

```java
System.out.println("\ta\tb\tc");
System.out.println("\\\\");
System.out.println("\'");
System.out.println("\\\\\\");
System.out.println("C:\nin\the downward spiral");
```

Write a `println` statement to produce the following line of output:

```
/ \ // \ \ /// \ \ 
```
Questions

What `println` statements will generate the following output?

This program prints a quote from the Gettysburg Address.

"Four score and seven years ago, our 'fore fathers' brought forth on this continent a new nation."

What `println` statements will generate the following output?

A "quoted" String is 'much' better if you learn the rules of "escape sequences."

Also, "" represents an empty String. Don't forget to use \\" instead of "! " is not the same as " 
Comments

- **comment**: A note written in the source code to make the code easier to understand.
  - Comments are not executed when your program runs.
  - Most Java editors show your comments with a special color.

- Comment, general syntax:
  
  ```
  /* <comment text; may span multiple lines> */
  ```
  
  or,
  
  ```
  // <comment text, on one line>
  ```

- Examples:
  
  ```
  /* A comment goes here. */
  /* It can even span multiple lines. */
  // This is a one-line comment.
  ```
Comments: Where do you go?

- ... at the top of each file (also called a "comment header"), naming the author and explaining what the program does.
- ... at the start of every method, describing its behavior.
- ... inside methods, to explain complex pieces of code.
Comments: Why?

- Comments provide important documentation.

- Later programs will span hundreds or thousands of lines, split into many classes and methods.

- Comments provide a simple description of what each class, method, etc. is doing.

- When multiple programmers work together, comments help one programmer understand the other's code.
That thing called style…

- What is style?
  - Indentation
  - Capitalization
  - Formatting / spacing
  - Structured code
  - No redundancy
  - …

- Why is it important?