Java Day 4

Finish Objects and Constructors

Method Overloading

Stack and Heap

Remeber that Java manages memory for us (handles memory management)

**The Heap**: the term for memory in Java

* All Objects are stored in the Heap.

**The Stack**: Stack memory is used for static memory allocation and the execution of a thread.

* Fast when compared to Heap Memory
* Includes:
  + Primitives
    - Primitives never leave the stack (unless attached to an object)
    - This is what makes them fast to work with - esp. Wrappers
  + Reference variables (which point to a location in the Heap where an Object is stored)
  + LIFO: Last in first out
    - new method called => new block on top of the stack (contains values, primitive variables, references to objects) => method executes => the stack pushes off the block

**Memory Structure**

* Variables are stored in memory
* A specific location in memory is called an "address"
  + each address stores a single byte of data
  + most variables then occupy multiple addresses
* The number of addresses reserved for a single variable is determined by the variables type
* The number of addresses/bytes reserved determines value range (binary)

**Reference Variables**

* stored the memory address (or reference) to an object in memory
* Objects have to reserve enough memory to hold all the variables stored for that single object
  + the memory reserved for an object might contain references to other objects, which contain others...
* This confusion and messiness is why Java lets us ignore memory management.
* *The reference variable is NOT the object...it's the door through which the object is accessed*

**Pass-by-value**: Java is **always** pass by value.

* With Primitives:
  + Java creates a copy of the variable being passed into a method
* With Objects (References)
  + Java creates a copy of the reference and passes it to the method - but it still points to the same memory address

Strings

* see code from class
* .equals() vs ==
* .equals() compares the characters in a String
* == compares the memory address of the String's reference variable

Wrapper Classes

Pillars

Encapsulation

Access Modifiers

Getters and Setters

### Composite Relationships