1/31/23

Variables

- Need to declare and initialize: type name = value;
- Types: int, double, boolean, char, float, long, short, byte, *String, Array (*primitives, *objects*)
    - Double vs float vs long? Float has less memory, long has more memory
    - Char is an int but output is its unicode translation (‘a’ = 97 = ‘\u0061’)
        - int i = 97, char a = ‘a’ : i == a
    - Short is a smaller int
    - Quick type coercion: as soon as a double is introduced into an equation, whole equation gets resolved as a double
    - Signed

Naming Rules

- ONLY letters, numbers, _
- Cannot start with a number
- Conventions: lowercaseUppercase, camel_case, consistent style

Misc

- String escapes: \’, \”, \\, \t, \n, \uXXX
- Mod operator: %: remainder of division
- Typecasting: no need for putting a small type into a larger (ie double i = 10), but must convert bigger to smaller (ie int height = (int) 10.5)
- Some fractions cannot be perfectly represented in binary, so 0.3*6 = 1.799… (rounding off error), need some tolerance
- Formatting: Ctrl + Alt + Shift + L

Class

- A blueprint
- Objects are instances of a class

Methods

- Header {body}
    - Header: modifiers returnType name(type parameters)
    - Modifiers: public - accessible, static - accessible w/o object
    - Final - method/var will not change - name in all caps at the start of code
    - Void: nothing returned
    - Calling from outside: ClassName.method(parameters)

2/2/23

Classes

- Java file same name as class, uppercase, defined by class Name {}
- Contains instance variables(should always be private, only accessed by methods), constructors(initializes variables and construct objects), methods
- Constructor syntax: public ClassName(args) { }
- Initialize variables: this.var1 = arg1
- Protected: only accessible by classes in the same package (default)
- Initialize object - ClassName objName = new ClassName(args);

Accessors - can only access without changing
Mutators - changes, can also access

Strings

- Indexed as if an array of chars
- s.charAt(int i), s.indexOf(char ch), s.lastIndexOf(char ch)
- Can concatenate with +/+=, s.concat(s2)
- s.substring(int startIndex, int lastIndex+1)
- s.replace(char oldChar, char newChar)
- s.replaceAll(Str8ng regex, String replacement)
- Other methods
    - isBlank, isEmpty, length, startWith, endWith
    - s.matches(regex)

2/7/23

- Regex is a pattern - eg [0-9]
- With objects, == compares address instead of values
- p1.equals(p2) instead, needs to be implemented or created by default
- Boolean operators have a lower precedence than relational operators
- Switch - testing multiple cases - switch(varName) { case value1: branch1; break;
    - default: branch3; break }

Arrays

- Initializing methods:
    - objectType[] arrayName = {value1, value2, ...};
        - In two lines : int[ ] myInt;
          myInt = new int[ ]{1, 2, 3};
    - objectType[] arrayName = new objectType[n];

2/9/23

- Any number besides 0 is considered true
- For each loop: for (objectType varName : collectionName) { mess with each element }
- Continue - skips current interaction of for loop
- Break - leaves 1 depth
- Recursion - if base case satisfied, terminate. Else update & recurse
- Helper methods do the actual work, recursive method recurses
- Recursion runs the risk of duplicating work

2/14/23

- Access (public, private, protected) & non-access (static, final, abstract) modifiers

Scanner

- Scan (read & parse) text
- Needs to be imported java.util
- Breaks input into list of tokens (each word) delimiter whitespace
- Close scanner

- Iterator appears over first word 0
    - hasNext, hasNextInt, hasNextDouble, hasNextLine, hasNextBoolean
    - next(return and go past), nextInt, nextDouble, nextLine, nextBoolean
- useDelimiter: set using regex
    - scan.useDelimiter(“,”); scan.useDelimiter(“[a-z]”);
- Pass in System.in for user input (Scanner scan = new Scanner(System.in);)
- Can pass a string in as well to tokenize and handle w scanner

Files

- Pass in a File object when constructing the scanner object to scan the file
    - File file = new File("myFile.txt");
    - Scanner reader = new Scanner(file);
    - while(reader.hasNextLine()) System.out.println(reader.nextLine());
    - reader.close();
- Use PrintWriter to write a file

ParseInt - to convert string to int

Exceptions

- A runtime error that will crash the program
- Throw: detect and signal an exception
- Catch: handle exception
- 3 types:
    - Internal errors: fatal errors (mainly in hardware)
    - Unchecked exception: runtime exception
    - Checked exception: checked by the compiler
- throw new ExceptionClass(message);
- Common types
    - NullPointer, Arithmetic, NumberFormat, IllegalArgument, fileNotFound
- Need a label a method that throws an exception and doesn’t handle it (*throws ExceptionClass* after parameters)
- Try-Catch (try{statements}catch(ExceptionClass exceptionObject){statements}
    - Multiple exceptions: separate ExceptionClasses by “|”
- Default: Exception E
- Try with resource : declare and initialize resource variable(s) in a pair of parentheses after keyword try

2/16/23

Documentation

- Multiline comments: /* */
- Javadoc: /** */
- Javadoc online, export tool, comments
- Format:
    - /**
    - *Summary
    - *@param

    - *@return
    - */

Testing

- Roughly 3 types (small-large)
    - Unit tests
    - Integration tests (interactions with external services)
    - End-to-End (backend to frontend, entire UX)
- Testing pyramid: 70% small, 20% medium, 10% large
- Write tests first
- JUnit: java's class that is built for testing java
    - Implement a test class that would test one class
    - Annotate each method as @Test
    - Each method tests one behavior of a method
    - Assert statements
    - assertEquals(expected, actual, 0.01f);

Inheritance

- Subclasses inherit superclasses
- class SubclassName extends SuperclassName
- Share common behaviors and data inherited from superclass
- "Is a" relationship
- Subclasses use methods, IV's, constructors that are not private (use getters to access IV's)
- Modify (override) what is not final
- Substitution principle: can also use a subclass object when a superclass object is expected (Animal pan = new Panda ✓, Panda pan = new Animal ❌)
    - But, can only use superclass features
- Polymorphism: diff object do the same task diff ways
- Access parent's instance vars through getters
- Only default constructor inherited
    - Use super to access/reuse constructors of superclass (must be first line of constructor)
- @Override

2/23/23

- Object class: clone, equals, toString, wait, hashCode, getClass
    - Final
    - toString default: className@hashcode
    - Equals default == addresses
        - Implementation must satisfy properties of equality (trans, reflex, symm)

Abstraction

- Does not have implementation details
- Method:
    - Concrete - with implementation
    - Abstract - declared but no implementation

- Cannot be called unless it's overridden
- public abstract class Name
- Can be subclassed (must have all outlined methods)
- Can have concrete methods?
- abstract returnType methodName(parameters);
- Interfaces don't need abstract keyword
    - Interfaces are like adjectives (can be shared between classes)
    - Interface in place of class
    - Most only have abstract classes
- A class "implements" interfaces (comma between)

Comparing Objects

- Classes must be comparable
    - Implement Comparable
    - Override compareTo
    - public class Name implements Comparable<Name> {}
- Comparator
    - NameComparator implements Comparator<ObjType>
    - Override compare
    - Use when objects do not have comparable, compares two given objects of ideally the same type

2/28/23

Overloading - explicitly name multiple methods the same thing w diff parameters
Overriding - explicitly name a method the same as its parent class to replace it (same parameters)

Overriding	Overloading
The method call is determined at runtime based on the object type	The method call is determined at compile time
Occurs between superclass and subclass	Occurs between the methods in the same class
Have the same signature (name and method arguments)	Have the same name, but the parameters are different

Anonymous Variable - variable without a name that gets directly passed into a method or used in "arithmetic" to define a variable (ie String mom = "m" + "om"; "m" and "om" are anonymous)

Anonymous class

- Define a class and construct an object at the same time
- Only used for a subclass of a known class or interface
- No constructors allowed

```
Animal dog = new Animal("dog"){
    @Override
    public void says() {
        System.out.println("bark!");
    }
};
```

-

Functional Interfaces - only one method

Lambda

- (parameter) -> { method body }
- May further simplify using :: (method reference operator)
- (e.g.: String::toUpperCase() is a shorthand for)

3/14/23

Linear vs Nonlinear data structure - Lists vs hashmaps and trees

Hashmaps - exist for speed of input and access

ArrayList

- More flexible, less concise syntax (ArrayList<Integer> list = new ArrayList<>(2))
- Can add items past the pre established limit
- list.set(0,1), get, add
- Must use add before getting or setting elements

LinkedList

- Made only of nodes and their neighbors
- Doubly linked
- Linear
- Next, Prev
- Add and remove only affect neighbor nodes (as opposed to shifting all other values down)
- Random access, head and tail easy to get
- Use ListIterator
    - ListIterator<Integer> = linkedList.listIterator();

Stack

- Data added from bottom to top, removed top to bottom
- Must remove top elements to access lower elements
- Push, pop, peek (pop without removal), empty, search(obj) (get distance from top: top = 1, not found = -1)
- Inherits add(index, element) avoid
- Undo, redo

Queue - “Waiting line” Added at tail, removed from head

Priority Queue

- Natural order of elements determines priority (ascendingly)
- Constructor can take nothing if objects are comparable
- Can take a comparator
- offer(element), peek, poll (get and remove head)

Deque - Double ended, sub-interface of queue (also implemented by LinkedList)
ArrayDeque - Concrete implementation of Deque, can be used as a stack or a queue, fast af

Interface	List			Queue/Deque	
Class	ArrayList	Stack	LinkedList	PriorityQueue	ArrayDeque
Order	Insertion			Natural	Insertion
Duplicates?	Yes				
Null?	Yes			No	
Commonly used Methods	Regular List operations (all implemented)	Only 5: `push`, `pop`, `peek`, `empty`, `search`	`add`/`offer` `get`/`peek` `remove`/`poll` head or tail	`offer`: add `peek`: get head `poll`: remove head	Most of the methods faster than in Stack or LinkedList
Special	Insertion/deletion in middle is NOT efficient, but get is OK	Only can add/get/remove the top (LIFO), top = 1	Insertion/deletion in middle is efficient, but not to get	(For Queue) Add to the tail, remove from head (FIFO)	Can be used as a stack or a queue
Applications	General purpose List (most commonly used)	Undo/redo, runtime stack, balanced paratheses, infix/postfix	Need to add/remove elements in the middle frequently	Printing queue, waiting list, anything related to deletion after finishing	Use as a stack or a linked list for better performance

Tree - root, leaf, nodes, edge, subtree, height, depth
Binary search tree - Left subtree only has elements less than, right subtree only greater
Map - Keys and values can be any type of object, no duplicate keys, put/remove/get methods based on keys
Hashing

- Represent an object via an integer using a hash function
- Hash Table: store elements based on their hash values
- To find an element, can easily compute hashcode instead of searching the whole array
- Code is %size to keep in range
- If collision, use a diff hash function

3/16/23

Var args

- Automates and hides process of creating and passing an array

- Can only be the last parameter
- Syntax can be ObjType[] name or ObjType… name

Generic

- Syntax (<type parameters>)
    - Class: public class Pair<S,T>
    - Method: public static <E> void print(E[] arr)
    - TSU generic
    - E elements in collection
    - KV key and values
    - Constrain types:
        - <T extends ClassName>
        - <T extends InterfaceName>
        - <T extends StructA & StructB>
    - Wildcard: a type that can remain unknown (for subtle constraints)
        - <? extends B>
        - <? super B>
        - <?>

3/21/23

Concurrency - doing multiple things at the same time
Process - An instance of a program running in a computer. Has a self-contained execution environment, at least one thread
Thread - If a thread is a worker, the process is the office
Multithreading - A process with >1 thread, each executed independently
Thread Scheduler - Manages threads, each thread runs for a short period of time (not actually concurrently - time slice)
Synchronous/Asynchronous - Execute tasks/threads one at a time or multiple at the same time
Race condition - Threads race to “win” a resource, solved by a lock
Lock - Blocks threads that want to manipulate a shared resource

3/23/23

Deadlock - Stuck waiting for each other's resources
Livelock - Stuck exchanging resources back and forth
Java supports multithreading from first version - automatic garbage collector runs in the background
Thread Class - Create and run a thread, has Runnable interface
Time complexity - Relationship between resources consumed and the size of the input (TimeTaken = (processor speed)*complexity(n) + overhead)
O(1) - constant
O(n) - linear
O(log(n)) - logarithmic, eg binary search, divides set in half each time
O(nlog(n)) - a logarithmic inside an O(n) loop
O(n!) - run away screaming

O(2^n) - ehhh

4/4/23

Sorting

- Select sort: rearranges elements
- Quick sort: divide and conquer - call quicksort recursively on each group of elements based on a pivot number (nlogn)

4/18/23

GUIs

- Java.awt, javax.swing, javaFX (newer)
- Components
    - Frame (JFrame): main window of GUI
        - Title bar + min, max, close buttons
    - Panels (JPanel): container for all other components
        - Default layout - FlowLayout
    - Components (JComponent): individual components (text fields, buttons, labels, etc)
    - Listeners: detect and respond to an event
        - Need to attach to event source
        - Are interfaces, need to be implemented
        - Adapters implements the listener with all empty bodies
- Layouts
    - BoxLayout - single row or column
    - BorderLayout - 5 areas (south, north, east, west, center)
    - GridLayout
    - CardLayout

4/20/23

JAVA 2D API

- Extends component class
- Maintains two coordinate spaces: user & device
- (0, 0) is top left corner
    - Objects are also pinned from top left corner
- Shapes, lines, points, curves
- Graphics for art stuff

4/25/23

P vs NP

- P: a problem that can be solved in polynomial time
- NP (nondeterministic polynomial): a problem that can be verified in polynomial time
    - P assumed but not proven to be within NP
- NP hard: solved in larger than P time
- NP complete: is NP hard and can be verified in polynomial time

- Turing machine: a hypothetical computer that exists mathematically
- Polynomial transformation: a problem X can transform into another if given any input that produces a certain outcome in X, there is an input that produces the same outcome in the second problem Y.

5/2/23

Master's theorem

- T(n) - time complexity
- a - number of times it calls itself on one level
- b - ratio of how you choose to shrink the problem (>1)
- T(n) = aT(n/b) + f(n)
- f(n) = θ(n^k*log^pn)

There are 3 cases:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ with[1] $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ with $\epsilon > 0$, and $f(n)$ satisfies the regularity condition, then $T(n) = \Theta(f(n))$.
   Regularity condition: $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.

AI/ML

- AI - anything that mimics human behavior
- ML - basically the program that "learns" from use
- Deep learning - subset of ML which makes the computation of multilayered neural networks feasible
- Types of ML problems
  - Classification
  - Regression - based on certain criteria, behavior/output changes
  - Clustering - classify data into a non predefined number of classes based on certain features
- Supervised learning - algorithm learns based off of a labeled dataset that essentially provides an answer key for the algorithm, used to predict future or unseen data
- Unsupervised learning - model infers a hidden structure or pattern onto unlabeled data

5/4/23

Web Dev

- Tech you should know
  - Version control
  - Linux
  - Virtual machines
  - Cloud computing (AWS
  - Web servers
  - REST API's
    - Text URL commands to a server
    - Streaming sites - go to dev console and look up filetype .mp4
    - Save mp4 and pirate wooooo, m3u8 with a converter

    - Database
    - Browser compatibility and CSS for frontend
- Small vs large companies
    - Bigger better but harder to get in
    - Smaller companies you will learn more
- FE or BE
- Backend concepts
    - Model view controller: standard architecture - user talks to controller, which communicates with model (data) and view
    - Payments: small companies can't store credit card information, must use tools such as:
        - Asynchronous JS calls
        - Normal controller calls
        - Tokens used to confirm payments with third party payment processor
    - Login/security
        - Cookies used to track whether a user is logged in, must refresh website if logged out
- AI, game dev, and VR good opportunities

'**5/9/23**

Game Dev

- Tools
    - Maya (python tools)
    - golang/sheets
    - Powershell
    - jira(ticketing and scrum - mark tasks as in progress or complete)
    - Unity w c# (and lua)
    - **Unreal w c++
- Freya holmer - math for game devs

Pre 146 knowledge!

- Be able to write a program from scratch

Book log (CSV)
Book title, author, cover link, pdf link

Book files (PDF)
Book covers (PDF)