**DATA STRUCTURES CHEAT SHEET**

**Java Basics**

* Platform independent, auto memory mgmt., large std lib
* Single inheritance; use **Interfaces** (all abstract methods)
* Access modifiers: public, private, protected, *package-private*
* **Overloading** = same name, different params
* **Overriding** = subclass redefines parent method (@Override, super)
* JVM = runs bytecode | JRE = runtime env | JDK = dev kit
* main() inside class
* **Static** = no objects, constants/util | **Instance** = can create objects

**Data Types (defaults)**

Boolean → false, byte (8 bit) → 0, short (16bit) → 0, int (32bit) → 0, long (64bit) → 0L, float (32bit) → 0.0f, double (64bit) → 0.0, char (16bit Unicode) → '\u0000'

**Control Structures**

* if / switch / for / while / do-while
* Ternary: (condition) ? x : y

**Exceptions**

* Example: divide by 0, array bounds
* Handled with try-catch, throw

**Objects**

* Class → constructor, fields (static/instance), methods, getters/setters
* Arrays → contiguous, fixed size, objects, can be multidimensional

**Collections**

* **List**: ordered, ops: add, get, set, remove, size, contains, clear
  + **ArrayList**: resizable array, O(1) access, no primitives (uses wrappers), autobox/unbox
  + **LinkedList**: nodes w/ pointers
    - Singly = 1 direction | Doubly = 2 directions (easy mid insert)
* **Stack**: LIFO | ops = push, pop, peek, isEmpty
* **Queue**: FIFO | ops = enqueue, dequeue, peek

**Algorithms**

* Must: well-defined, finite, effective, input/output
* Approaches:
  + Brute force = try all
  + Greedy = best local choice
  + Divide & Conquer = split into subproblems
  + Dynamic Programming = store/reuse subsolutions
* **Time Complexity**:
  + O(1) = constant, O(log n) = logarithmic, O(n) = linear, O(n²) = quadratic, O(2ⁿ) = exponential (bad)
* **Space Complexity**: O(1), O(n), O(n²)

**Complexity Classes**

* P = solvable in polytime, NP = verifiable in polytime, NP-Complete = hardest NP, NP-Hard = ≥ NP-Complete
* Approximation = near-optimal fast
* Heuristic = guide, not guaranteed

**Recursion**

* Solve problem w/ smaller self-calls
* 3 Rules: 1. Base case 2. Must progress toward base 3.Must call itself