AP CS A Power Ups

Review these one line tips and tricks before the test!

Math & Numbers

- Random [0, n): `(int)(Math.random() * n)`
- Random integer [min, max]: `(int)(Math.random()*(max-min+1))+min`
- Do something 30% of the time:
  `if (Math.random() * < 0.30) {//do it}`
- Integer division truncates — 7/2 → 3, not 3.5
- Get the ones digit: `n % 10`;
- Remove the ones digit: `n = n / 10` (repeat to walk all digits)
- Math methods: `Math.abs()` `Math.pow(b,e)` `Math.sqrt()` — all return `double`
- *room to add your own tips*
-
-

Strings

- Use `.equals()` not `==`
- `.substring(a, b)` — a inclusive, b exclusive
- `"hello".substring(1, 3)` → "el"
- `.length()` is a method (parens!) — unlike array `arr.length`
- `.indexOf()` returns -1 if not found — useful in an if-check
- Strings are immutable — every method returns a new String; original never changes
- `Concat: "val: " + 3 + 4` → "val: 34" not "val: 7"
-
-

Scanner (I/O)

- `new Scanner(System.in)` for keyboard;
- `new Scanner(new File("f.txt"))` for file
- `nextLine()` reads the whole line;
- `next()` reads one whitespace-delimited token
- `nextInt()` / `nextDouble()` reads and converts the next word into a number
- `sc.hasNextLine()` loop over all lines
- `sc.hasNext()` loop over all tokens/words

```
Scanner sc = new Scanner(System.in);
String line = sc.nextLine();          // full line
int n       = sc.nextInt();           // one int token
sc.nextLine();                        // flush leftover
newline

String[] parts = line.split(" ");    // split on space
int val = Integer.parseInt(parts[0]); // String → int

while (sc.hasNextLine()) {            // loop until EOF
    process(sc.nextLine());
}
```

Arrays

- `arr.length` — field, no parens (unlike `list.size()`)
- Defaults values:
- `int[] =` 0,
- `boolean[] =` false,
- `Object[] =` null
- Enhanced for loop can't modify the array — use indexed loop when assigning: `for(int x: list) {x = 3}` doesn't change the list!
- To change values in a loop use `list.set(i, 3)` for ArrayLists or `list[i] = 3)` for Arrays
-
-

2D Arrays

- Declare: `int[][] g = new int[rows][cols];`
- Or `int[][] g = {{1,2},{3,4}};`
- `g.length` = rows
- `g[0].length` = cols
- access as `g[row][col]`
- Standard traversal: outer loop rows (i), inner loop cols (j) — swap order for column-major
- How to check all 4 neighbors:

```
if ((r-1 > 0 && g[r-1][c] == true) ||
   (r+1 < g.length && g[r+1][c] == true) ||
   (c-1 > 0 && g[r][c-1] == true) ||
   (c+1 < g[r].length && g[r][c+1] == true) )
```

ArrayLists

- `list.size()` — method (parens), not a field
- No primitives types — use `Integer`, `Double`, `Boolean`
- `.remove(2)` removes index 2;
  `.remove(Integer.valueOf(2))` removes value 2
- Don't add/remove while iterating forward — always delete in a reverse loop
-
-

Array Bounds Checking

- Guard before accessing a neighbor — short-circuit `&&` keeps it safe:
- `i-1 >= 0 && a[i-1] < a[i]` ← left neighbor
- `i+1 < a.length && a[i+1] > a[i]` ← right neighbor
- Bounds check MUST come first — that's the whole point of short-circuit
- `i-1 > 0` misses index 0! Always use `i-1 >= 0`
- 2D: stack both — `r-1 >= 0 && c+1 < grid[0].length && grid[r-1][c+1] > 0`
-
-

AP CS A Power Ups

Casting & Numeric Promotion

- Promotion (implicit widening): when `int` and `double` mix, the `int` silently becomes `double`
- `int/int` → `int` **always** — both sides must involve a `double` (or cast) to get a decimal

```
int a = 7, b = 2;
double x = a / b;           // 3.0  ❌ int division happens first
double y = (double) a / b;  // 3.5  ✅ cast promotes a before dividing
double z = (double)(a / b); // 3.0  ❌ cast applied after — too late!

int t = (int) 3.9;          // 3    truncates toward zero
int u = (int) -3.9;         // -3   NOT -4!  always toward zero
```

- Cast affects the very next value only — `(double) a / b` ≠ `(double)(a / b)`

Boolean & Conditionals

- **= assigns, == compares**
- `&&` stops at first `false`; `||` stops at first `true` (short-circuit)
- Null guard pattern: `obj != null && obj.getValue() > 0`
- ! precedence: `!a && b` means `(!a) && b`, not `!(a && b)`

Object & Reference

- Null pointer: calling any method on `null` crashes — always check first
- **.equals() vs ==** applies to ALL objects, not just Strings
- `toString()` is called implicitly when you print an object or concatenate an object with `+`

Loops

- Last valid index: `arr.length - 1` — use `< arr.length`
- Enhanced for loops cannot change the original list
- Reverse: `for (int i = arr.length-1; i >= 0; i--)`
- Searching: usually you can return inside loop on first match; return -1 / false after the loop ends
- Use an enhanced for loop for ArrayLists unless you need to change the list!

Recursion

- **Always needs a base case** — no base case = StackOverflowError
- The call stack builds up, then unwinds — code AFTER the recursive call runs in reverse order (deepest call finishes first)
- Print BEFORE the call → outermost first (3, 2, 1)
- Print AFTER the call → innermost first (1, 2, 3)

```
void forward(int n) {
  if (n == 0) return;
  System.out.println(n); // BEFORE → 3, 2, 1
  forward(n - 1);
}

void backward(int n) {
  if (n == 0) return;
  backward(n - 1);
  System.out.println(n); // AFTER  → 1, 2, 3
}
```

- Return values also accumulate on the way back up: `return 1 + f(n-1)` adds 1 at every level

Classes

- Instance variables MUST be `private` — omitting it loses a rubric point on the FRQ
- Constructor name = class name, no return type; initializes every instance variable
- Two constructors are common, they just need a different number or type of parameters
- Use `this.field` when a parameter name is the same as a field name — otherwise the parameter is assigned to itself
- Getters return a value; setters are `void`
- `static`: only ONE per class — called as `ClassName.method()`, or `ClassName.field()`
- Staic methods can't use `this`

```
public class Dog {
  private String name;  // PRIVATE
  private int age;

  public Dog() { this("Unknown", 0); } // calls below
  public Dog(String name, int age) {
    this.name = name;   // this. needed: names clash
    this.age  = age;
  }
  public String getName() { return name; } // getter
  public void setAge(int a) { age = a; }   // setter

  public static int dogYears(int h) {      // static
    return h * 7;
  }
}
```

- `// Dog.dogYears(5)  not  d.dogYears(5)`

SORTING ALGORITHMS

	Selection	Insertion	Merge
Strategy	Find min, swap to front	Take next, slide left into sorted half	Divide, sort halves, merge back
Best case	O(n²)	O(n) ← nearly sorted!	O(n log n)
Worst case	O(n²)	O(n²)	O(n log n)
Swaps / moves	≤ n swaps total	Many element shifts	No swaps — copies to aux array
Extra memory	In-place	In-place	Needs O(n) extra array
Key insight	Always same work — never lucky	Fast on nearly-sorted data	Guaranteed fast; costs memory