

AP COMPUTER SCIENCE A STUDY GUIDE

185 High-Yield Concepts

Advanced Placement Course | College Board CED-aligned

- All 4 CED units: objects & methods, selection & iteration, class creation, data collections
- Full Java coverage: arrays, ArrayLists, 2D arrays, searching, sorting, and recursion
 - Exam weights on every unit; real AP exam format on the how-to page
 - Perfect for the AP Computer Science A exam in May

TemplateForge | \$19

PDF Workbook | templateforge.com | (c) TemplateForge

How to Use This Guide

This guide condenses the AP Computer Science A course and exam description into high-yield concepts. It is not a textbook replacement -- it is the fastest way to review everything the exam covers and to target your weak units.

The AP Computer Science A exam is 3 hours: 42 multiple-choice questions (90 min, 55% of the score) and 4 free-response questions (90 min, 45% of the score): Methods and Control Structures, Class Design, Data Analysis with ArrayList, and 2D Array. The 4 units below map to the current College Board Course and Exam Description, with unit weights on each section title.

How to use the concepts

- Each concept = term + concise explanation + high-yield exam tip (+ example where useful).
- First pass: read every concept in order. Underline anything unfamiliar.
- Second pass: focus only on the concepts you underlined.
- Two weeks before the exam: rapid-review every tip line (they are the highest-yield).
- Use the Table of Contents to jump straight to your weakest units.

Table of Contents

1	Unit 1: Using Objects and Methods	53 concepts
2	Unit 2: Selection and Iteration	42 concepts
3	Unit 3: Class Creation	40 concepts
4	Unit 4: Data Collections	50 concepts
		Total: 185 high-yield concepts

How to use the TOC: work through the units in order on your first pass, then jump back to your weakest units for review. Each concept includes an Explanation, a high-yield exam tip, and where useful an Example. Mark the concepts you miss with a small dot in the margin and revisit them 48 hours later.

Unit 1: Unit 1: Using Objects and Methods

15-25% of the AP CSA exam. Primitive types, variables, operators, Strings, the Math class, and the object model: constructors, references, and methods.

1 Primitive Types

Explanation: The basic data types built into Java: int, double, boolean, char, long, short, byte, float. They store simple values directly in memory.

AP CSA Tip: Know int (whole numbers), double (decimals), boolean (true/false), char (single character). The AP exam uses these four.

Example: int age = 17; double price = 9.99; boolean done = false; char grade = 'A';

2 Variable Declaration and Initialization

Explanation: Declaring a variable states its type and name; initializing assigns its first value. In Java, local variables must be initialized before use.

AP CSA Tip: Declaration = type + name; initialization = value. int x; then x = 5; or int x = 5; in one line.

Example: int score = 0; declares an int named score and sets it to 0.

3 Assignment Statements

Explanation: The = operator stores a value into a variable, replacing any previous value. Assignment is right-to-left: the right side is evaluated first.

AP CSA Tip: In $x = y + 1$, compute $y + 1$ first, then store into x . The old x is overwritten.

Example: $x = x + 1$; reads x , adds 1, and stores the new value back into x .

4 Numeric Operators

Explanation: Java's arithmetic: + (add), - (subtract), * (multiply), / (divide), % (modulo/remainder). Operator precedence follows math conventions.

AP CSA Tip: Remember: * and / bind tighter than + and -. Parentheses override everything.

Example: $7 / 2$ is 3 (integer division), but $7.0 / 2$ is 3.5.

5 Integer Division

Explanation: When both operands are ints, / performs integer division, truncating the remainder: $7 / 2 = 3$, not 3.5. The % operator gives the remainder: $7 \% 2 = 1$.

AP CSA Tip: Integer division truncates toward zero; % gives the remainder. To get a decimal, make at least one operand a double.

Example: int a = 17 / 5; gives 3. int b = 17 % 5; gives 2.

6 Modulo (%) Operator

Explanation: Returns the remainder after division. Useful for even/odd checks ($n \% 2$) and digit extraction.

AP CSA Tip: $n \% 2 == 0$ means even; $x \% 10$ extracts the last digit of a positive integer.

Example: $123 \% 10$ is 3; $123 / 10$ is 12; together they peel off digits.

7 Compound Assignment Operators

Explanation: Shorthand for updating a variable: +=, -=, *=, /=, %= . $x += 5$ means $x = x + 5$.

AP CSA Tip: Compound operators modify in place; $x++$ and $x--$ add or subtract exactly 1.

Example: count += 2; is the same as count = count + 2;

8 Increment and Decrement

Explanation: $x++$ adds 1 to x ; $x--$ subtracts 1. In AP Java, use them as statements, not inside expressions, to avoid confusion.

AP CSA Tip: $x++$ and $++x$ both increment, but in expressions they differ (post vs pre). For AP, use them standalone.

Example: for (int i = 0; i < 10; i++) { ... } increments i each pass.

9 Casting Between Types

Explanation: Casting converts a value to another type: (int) 3.9 is 3 (truncation); (double) 5 is 5.0. Casting an int to double is widening; double to int is narrowing.

AP CSA Tip: *Casting double to int truncates (does NOT round). Parenthesize: (int)(7.0 / 2) is 3.*

Example: double d = (double) 5 / 2; gives 2.5 because 5 becomes 5.0 first.

10 Overflow and Rounding

Explanation: int values overflow past 2,147,483,647, wrapping around; double arithmetic can lose precision on fractions like 0.1.

AP CSA Tip: *Watch for overflow in large products; use long or double when values can grow.*

Example: int x = 2000000000; x += 2000000000; overflows to a negative value.

11 Strings

Explanation: Strings are objects (not primitives) representing sequences of characters, created with String s = "hi"; or new String("hi");

AP CSA Tip: *Strings are immutable: methods return NEW strings rather than changing the original.*

Example: String name = "Ada"; name.length() is 3.

12 String Concatenation

Explanation: The + operator joins strings; when one operand is a String, the other is converted to a String.

AP CSA Tip: *Concatenation is left-to-right: "1" + 2 + 3 is "123", but 1 + 2 + "3" is "33".*

Example: System.out.println("Total: " + 5 + 5); prints "Total: 55", not "Total: 10".

13 String Methods: length, substring

Explanation: s.length() returns the number of characters; s.substring(a, b) returns chars from index a up to (not including) b.

AP CSA Tip: *Indexes start at 0; substring(a, b) excludes b. substring(a) goes to the end.*

Example: String s = "hello"; s.substring(1, 3) is "el".

14 String Methods: indexOf, compareTo, equals

Explanation: s.indexOf(ch) returns the first index of a character (-1 if absent); s.compareTo(t) compares lexicographically; s.equals(t) checks exact content.

AP CSA Tip: *ALWAYS use equals() for String comparison, never ==. == compares references, not contents.*

Example: if (s.equals("yes")) { ... } -- the correct way to test string equality.

15 String Immutability

Explanation: String objects cannot be changed; every method returns a new String, leaving the original intact.

AP CSA Tip: *Because Strings are immutable, s.toUpperCase() does NOT modify s -- assign the result to use it.*

Example: String s = "hi"; String t = s.toUpperCase(); s is still "hi", t is "HI".

16 CharAt and Characters

Explanation: s.charAt(i) returns the char at index i. Characters can be compared and manipulated with char arithmetic.

AP CSA Tip: *charAt returns a char; compare chars with == directly (they are primitives).*

Example: "abc".charAt(1) is 'b'; (char)('A' + 1) is 'B'.

17 The Math Class

Explanation: A utility class of static methods: Math.abs(x), Math.pow(a, b), Math.sqrt(x), Math.random(), Math.max(a, b), Math.min(a, b).

AP CSA Tip: *Math.random() returns a double in [0, 1); (int)(Math.random() * n) gives 0 to n-1.*

Example: int die = (int)(Math.random() * 6) + 1; simulates a die roll.

18 Math.random() Range

Explanation: Returns a random double greater than or equal to 0.0 and less than 1.0. Scale and shift to get a desired range.

AP CSA Tip: *Formula: $(\text{int})(\text{Math.random()} * (\text{max} - \text{min} + 1)) + \text{min}$ gives integers from min to max inclusive.*

Example: To roll 1-6: $(\text{int})(\text{Math.random()} * 6) + 1$.

19 Math.pow and Math.sqrt

Explanation: Math.pow(base, exp) returns base^{exp} as a double; Math.sqrt(x) returns the square root as a double.

AP CSA Tip: *pow and sqrt return doubles; cast if an int is needed. Use multiplication for small integer powers.*

Example: Math.pow(2, 10) is 1024.0; Math.sqrt(16) is 4.0.

20 Math.abs, max, min

Explanation: Math.abs(x) returns absolute value; Math.max(a, b) and Math.min(a, b) return the larger/smaller of two values.

AP CSA Tip: *max/min can be nested: Math.max(a, Math.max(b, c)) finds the largest of three.*

Example: Math.abs(-7) is 7; Math.max(3, 9) is 9.

21 Objects and Classes

Explanation: A class is a blueprint; an object is an instance created from it. Objects bundle state (instance variables) and behavior (methods).

AP CSA Tip: *Class = blueprint; object = instance. new creates an object; the constructor runs.*

Example: String is a class; the value "hello" is an object of that class.

22 The new Keyword

Explanation: new allocates an object and calls its constructor: new ClassName(arguments).

AP CSA Tip: *new is required to create most objects (except Strings, which have special syntax).*

Example: Scanner sc = new Scanner(System.in); creates a Scanner object.

23 Constructors

Explanation: Special methods that initialize an object's state, with the same name as the class and no return type.

AP CSA Tip: *Constructors run once at creation; a class can have multiple overloaded constructors.*

Example: new Point(3, 4) calls the Point constructor with x = 3, y = 4.

24 Default Constructor

Explanation: If a class defines no constructors, Java provides a no-argument default constructor that sets fields to defaults (0, null, false).

AP CSA Tip: *Writing ANY constructor removes the default; then new MyClass() fails unless you define a no-arg version.*

Example: class Box { } -- Box b = new Box(); uses the implicit default constructor.

25 Object References

Explanation: Object variables store REFERENCES (memory addresses), not the object itself. Two variables can reference the same object.

AP CSA Tip: *Assignment copies the reference: Point p2 = p1; makes p2 and p1 point to the SAME object.*

Example: String a = "hi"; String b = a; -- a and b refer to the same String object.

26 Aliasing

Explanation: When two references point to the same object, changing it through one reference is visible through the other.

AP CSA Tip: *Aliasing means mutations via one reference affect the shared object -- a classic exam trap.*

Example: Point p = new Point(1, 2); Point q = p; q.x = 99; now p.x is also 99.

27 Calling Methods on Objects

Explanation: Use dot notation: `object.method(arguments)`. Methods can return values or just perform actions (void).

AP CSA Tip: *The dot operator accesses an object's methods; the object must be non-null.*

Example: `String s = "hi"; int len = s.length();` calls `length()` on `s`.

28 Method Signatures and Overloading

Explanation: A method signature is its name plus parameter types. Overloading = multiple methods with the same name but different parameters.

AP CSA Tip: *Overloaded methods differ in parameter types/count, not return type alone.*

Example: `Math.max(int, int)` and `Math.max(double, double)` are overloaded versions.

29 Parameters and Arguments

Explanation: Parameters are the declared inputs of a method; arguments are the values passed when calling it.

AP CSA Tip: *Primitives are passed by value (a copy); objects pass the reference (a copy of the pointer).*

Example: `void draw(int size) { ... }` -- `draw(10)` passes 10 as the size argument.

30 Return Types and void

Explanation: A method declares its return type; void methods return nothing. `return` exits a method and delivers a value.

AP CSA Tip: *Every non-void method must return a value on all paths; void methods can use bare `return`; to exit early.*

Example: `int square(int x) { return x * x; }` -- must return an int.

31 Wrapper Classes

Explanation: Classes that wrap primitives as objects: `Integer`, `Double`, `Boolean`, `Character`. Used where objects are required.

AP CSA Tip: *Auto-boxing converts int to Integer automatically; unboxing reverses it.*

Example: `Integer n = 42;` -- autoboxing wraps the int 42 in an `Integer` object.

32 Integer and Double Wrapper Methods

Explanation: `Integer.parseInt(s)` converts a `String` to int; `Double.parseDouble(s)` to double; `Integer.MAX_VALUE` is the largest int.

AP CSA Tip: *`parseInt` throws `NumberFormatException` on bad input; `Integer.MAX_VALUE` is 2147483647.*

Example: `int n = Integer.parseInt("123");` gives 123.

33 Scanner for Input

Explanation: `Scanner` reads input: `new Scanner(System.in)` for console; `sc.nextInt()`, `sc.nextDouble()`, `sc.next()`, `sc.nextLine()`.

AP CSA Tip: *`next()` reads one token; `nextLine()` reads a whole line; mixing them can skip input.*

Example: `Scanner sc = new Scanner(System.in); int n = sc.nextInt();`

34 Boolean Primitives

Explanation: boolean variables hold true or false, produced by comparisons (`==`, `<`, `>`, `<=`, `>=`, `!=`) and logic operators.

AP CSA Tip: *Booleans are the fuel of if statements and loops; compare primitives with `==` but objects with `equals()`.*

Example: `boolean adult = age >= 18;`

35 Literals and Escape Sequences

Explanation: Literals are fixed values in code (17, 3.14, 'A', "hi"). Escape sequences like `\n` (newline) and `\t` (tab) appear inside strings.

AP CSA Tip: *In code, 'a' is a char and "a" is a String -- they are different types.*

Example: `System.out.println("Line1\nLine2");` prints on two lines.

36 Trace Simple Code

Explanation: Tracing = simulating execution line by line, recording variable values, to predict output.

AP CSA Tip: *Trace systematically with a table of variables; watch for order of operations and truncation.*

Example: `int x = 3; x *= 4; x -= 2; -- trace: x = 3, then 12, then 10.`

37 Scope of Variables

Explanation: A variable's scope is where it is accessible: local variables live inside their block; instance variables live for the object's lifetime.

AP CSA Tip: *A variable declared inside a block is invisible outside it; declaring the same name in nested scopes shadows it.*

Example: `for (int i = 0; i < 5; i++) { } -- i is out of scope after the loop.`

38 Final Variables

Explanation: `final` marks a variable as constant: its value cannot change after assignment.

AP CSA Tip: *final fields (like `Math.PI`) cannot be reassigned; they are often static and public.*

Example: `public static final double PI = 3.14159;`

39 String Comparison Trap

Explanation: `==` on Strings compares references, so it often returns false for equal-looking strings; always use `equals()`.

AP CSA Tip: *When in doubt, use `equals()`; `==` for Strings is wrong unless both are interned literals.*

Example: `"hi" == new String("hi")` is false; `"hi".equals(new String("hi"))` is true.

40 Type Promotion in Expressions

Explanation: In arithmetic, ints promote to double when combined with doubles; the result type is the widest operand type.

AP CSA Tip: *Mixed int/double arithmetic yields double; assign to a double variable or cast to truncate.*

Example: `double d = 1 + 1.5; -- 1 promotes to 1.0, giving 2.5.`

41 Comparing Doubles

Explanation: Doubles are approximate; testing `==` on computed doubles can fail. Compare with a small tolerance.

AP CSA Tip: *Avoid `d1 == d2` for computed doubles; use `Math.abs(d1 - d2) < 0.0001`.*

Example: `0.1 + 0.2 != 0.3` exactly in floating point.

42 Short-Circuit Evaluation

Explanation: `&&` and `||` stop evaluating once the result is known: false `&&` anything is false, true `||` anything is true.

AP CSA Tip: *Short-circuit can avoid errors: `s != null && s.length() > 0` never calls `length()` on null.*

Example: `if (x != 0 && 10 / x > 2) -- safe because x == 0 short-circuits the division.`

43 String Concatenation Order

Explanation: Expressions evaluate left to right; once a String appears, everything after converts to String.

AP CSA Tip: *To force arithmetic first, parenthesize: `"sum: " + (a + b)`.*

Example: `"" + 1 + 2` is "12"; `"" + (1 + 2)` is "3".

44 Integer.MAX_VALUE and MIN_VALUE

Explanation: The largest int is 2,147,483,647 and the smallest is -2,147,483,648; arithmetic past these overflows.

AP CSA Tip: *Use long when values may exceed int range; overflow wraps silently.*

Example: `Integer.MAX_VALUE + 1` equals `Integer.MIN_VALUE`.

45 Enhanced Clarity in Code

Explanation: Readable code: meaningful names, consistent indentation, and clear structure reduce bugs and aid tracing.

AP CSA Tip: *The exam rewards correct logic, not cleverness; clarity helps you trace your own code.*

Example: `int numberOfStudents = 25; beats int n = 25;`

46 Math.round vs Casting

Explanation: `Math.round(2.5)` rounds to the nearest long (3); `Math.round(2.49)` rounds down (2). Casting `(int) 2.9` truncates to 2, dropping the fraction.

AP CSA Tip: *Casting truncates; Math.round rounds to nearest. Use Math.round when a problem says 'round', casting when it says 'truncate'.*

Example: `int a = (int) 3.9; gives 3. long b = Math.round(3.5); gives 4.`

47 The char Type

Explanation: `char` stores a single 16-bit Unicode character: `char c = 'A';`. Chars are numeric and can be cast to `int`.

AP CSA Tip: *Chars compare with relational operators by their numeric code; 'A' is 65, 'a' is 97.*

Example: `(char)('a' - 32)` is 'A' -- converting lowercase to uppercase by arithmetic.

48 String Index Bounds

Explanation: `charAt(i)` and `substring` must stay inside `0..length-1`; going past throws `StringIndexOutOfBoundsException`.

AP CSA Tip: *The last valid index is `length() - 1`; `substring(a, b)` requires `0 <= a <= b <= length()`.*

Example: `"hi".charAt(2)` throws `StringIndexOutOfBoundsException`.

49 Naming Conventions

Explanation: Java conventions: classes in PascalCase (`StudentRecord`), variables and methods in camelCase (`getScore`), constants in UPPER_SNAKE (`MAX_VALUE`).

AP CSA Tip: *Conventions are not syntax, but the exam expects conventional names in FRQ code.*

Example: `public class BankAccount { private double balance; }`

50 Default Values of Fields

Explanation: Instance fields default to 0 (numeric), false (boolean), and null (objects) if not explicitly initialized.

AP CSA Tip: *Local variables have NO defaults and must be initialized before use; fields do have defaults.*

Example: `private int count; -- count starts at 0.`

51 Reading Input with Scanner

Explanation: `sc.nextInt()` reads an int, `sc.nextDouble()` a double, `sc.next()` one token, `sc.nextLine()` a whole line.

AP CSA Tip: *After `nextInt()`, the leftover newline makes the next `nextLine()` return an empty string -- call `nextLine()` once to consume it.*

Example: `int age = sc.nextInt(); String rest = sc.nextLine();`

52 Chaining Method Calls

Explanation: Call methods on method results left to right: `s.trim().toUpperCase()` trims `s`, then uppercases the result.

AP CSA Tip: *Each call must be valid on the previous result's type; chains read left to right.*

Example: `String t = " hi ".trim().toUpperCase(); -- t is "HI".`

53 equalsIgnoreCase

Explanation: `s.equalsIgnoreCase(t)` compares strings ignoring case, useful for user input like 'yes', 'YES', 'Yes'.

AP CSA Tip: *Use `equalsIgnoreCase` when case should not matter; `equals` when it should.*

Example: `"YES".equalsIgnoreCase("yes")` is true.

Unit 2: Unit 2: Selection and Iteration

25-35% of the AP CSA exam. Boolean expressions, if/else logic, and while/for loops with the standard algorithms.

54 if Statements

Explanation: if (condition) { block } executes the block only when the condition is true.

AP CSA Tip: The condition must be a boolean expression; braces are optional for a single statement but recommended.

Example: if (score >= 90) { grade = 'A'; }

55 if-else Statements

Explanation: if (cond) { A } else { B } runs A when true, B when false -- exactly one branch runs.

AP CSA Tip: else attaches to the nearest unmatched if; the else branch handles the 'otherwise' case.

Example: if (x % 2 == 0) { parity = "even"; } else { parity = "odd"; }

56 else if Chains

Explanation: Multiple mutually exclusive conditions: if / else if / else. The first true branch runs; the rest are skipped.

AP CSA Tip: Order matters: put the most specific or most common conditions first in a chain.

Example: if (n < 0) { ... } else if (n == 0) { ... } else { ... }

57 Boolean Expressions with Relational Operators

Explanation: Relational operators: == (equal), != (not equal), <, >, <=, >=. They produce booleans.

AP CSA Tip: Use == for primitives, equals() for objects; <= and >= include the boundary.

Example: age >= 18 && age < 65 tests working-age adults.

58 Compound Boolean Expressions

Explanation: Combining conditions with && (AND), || (OR), and ! (NOT).

AP CSA Tip: && requires both true; || requires at least one; ! negates. Know the truth tables.

Example: if (temp > 80 || humidity > 90) { alert(); }

59 De Morgan's Laws

Explanation: !(A && B) is equivalent to (!A || !B); !(A || B) is equivalent to (!A && !B).

AP CSA Tip: Use De Morgan to simplify or rewrite negated conditions.

Example: !(x > 0 && y > 0) is the same as (x <= 0 || y <= 0).

60 Nested Conditionals

Explanation: if statements inside if statements, testing multiple levels of conditions.

AP CSA Tip: Nested ifs create a decision tree; they can often be flattened with && or else-if chains.

Example: if (x > 0) { if (y > 0) { print("quadrant 1"); } }

61 Comparing Objects with equals

Explanation: Objects (String, Point, etc.) must be compared with equals(), which compares content, not references.

AP CSA Tip: equals() checks logical equality; == checks whether two references point to the same object.

Example: if (name.equals("Alex")) { ... } -- correct object comparison.

62 Comparing String Content Lexicographically

Explanation: `compareTo` returns a negative number if this < other, 0 if equal, positive if this > other, in dictionary order.

AP CSA Tip: *`compareTo < 0` means alphabetically earlier; case matters (uppercase sorts before lowercase).*

Example: `"apple".compareTo("banana")` is negative because apple comes first.

63 while Loops

Explanation: `while (condition) { block }` repeats the block as long as the condition is true; it may run zero times.

AP CSA Tip: *The condition is checked BEFORE each iteration; ensure the loop body moves toward false.*

Example: `int i = 0; while (i < 5) { i++; } --` runs 5 times.

64 Infinite Loops

Explanation: A loop whose condition never becomes false, running forever. Caused by not updating the loop variable.

AP CSA Tip: *Check that the body changes a variable used in the condition; off-by-one bugs often cause them.*

Example: `int i = 0; while (i < 10) { } --` i never changes, infinite loop.

65 for Loops

Explanation: `for (init; condition; update) { block }` bundles initialization, condition, and update in one header.

AP CSA Tip: *All three parts are optional; the classic form counts from 0 to n-1.*

Example: `for (int i = 0; i < n; i++) { System.out.println(i); }`

66 Loop Control: break and continue

Explanation: `break` exits the loop immediately; `continue` skips to the next iteration.

AP CSA Tip: *`break` jumps out; `continue` jumps to the condition/update. Use them sparingly for clarity.*

Example: `for (int i = 0; i < 10; i++) { if (i == 5) break; } --` stops at 5.

67 Nested Loops

Explanation: Loops inside loops: the inner loop completes fully for each iteration of the outer loop.

AP CSA Tip: *Nested loops multiply iterations: outer n, inner m gives n*m total executions.*

Example: `for (int r = 0; r < 3; r++) { for (int c = 0; c < 4; c++) { ... } } --` 12 iterations.

68 Using Loops to Process Data

Explanation: Standard loop algorithms: counting, summing, finding min/max, searching, and accumulating results.

AP CSA Tip: *Identify which algorithm a problem needs: count? sum? find? compare pairs?*

Example: `Sum: int total = 0; for each value, total += value;`

69 Counting with Loops

Explanation: Counting occurrences: initialize a counter to 0 and increment it when the condition holds.

AP CSA Tip: *Set the counter to 0 BEFORE the loop; increment inside an if.*

Example: `int evens = 0; for (int x : arr) { if (x % 2 == 0) evens++; }`

70 Summing with Loops

Explanation: Accumulate a total: `int sum = 0; sum += value;` inside the loop.

AP CSA Tip: *Initialize the accumulator to the identity (0 for sum, 1 for product).*

Example: `for (int i = 1; i <= n; i++) { sum += i; } --` sums 1..n.

71 Finding Maximum and Minimum

Explanation: Track the best so far: start with the first element, compare each subsequent element, update if better.

AP CSA Tip: *Initialize max to the FIRST value (or Integer.MIN_VALUE), not 0 -- negatives break 0.*

Example: `int max = arr[0]; for (int x : arr) { if (x > max) max = x; }`

72 Linear Search

Explanation: Scanning elements one by one until the target is found or the list ends; returns the index or -1.

AP CSA Tip: *Linear search needs no sorting; it stops early on success but may check every element.*

Example: `for (int i = 0; i < arr.length; i++) { if (arr[i] == target) return i; } return -1;`

73 Comparing Adjacent Values

Explanation: Loop algorithms that compare neighbors: counting increases, finding peaks, checking sortedness.

AP CSA Tip: *Loop from index 1 to n-1, comparing arr[i] with arr[i-1].*

Example: `for (int i = 1; i < arr.length; i++) { if (arr[i] < arr[i-1]) sorted = false; }`

74 Sentinel-Controlled Loops

Explanation: A loop that reads until a sentinel value (like -1 or "quit") signals the end of input.

AP CSA Tip: *The sentinel is a value that cannot be a real data item; check it in the loop condition.*

Example: `while (sc.nextInt() != -1) { process(value); }`

75 Reading Until Input Ends

Explanation: Loops over input until there is no more data, often using hasNext() or hasNextInt().

AP CSA Tip: *hasNextInt() returns false when input runs out, safely ending the loop.*

Example: `while (sc.hasNextInt()) { int v = sc.nextInt(); ... }`

76 Loop Invariant Thinking

Explanation: A condition that stays true before and after each loop iteration, used to reason about correctness.

AP CSA Tip: *State what the loop maintains (e.g., 'sum holds the total so far') to check correctness.*

Example: In a sum loop, the invariant is: `sum == total of elements processed so far.`

77 String Traversal with Loops

Explanation: Iterating over characters with `for (int i = 0; i < s.length(); i++)` and `charAt(i)`.

AP CSA Tip: *Traverse strings by index; count chars, reverse strings, or build new strings.*

Example: `for (int i = 0; i < s.length(); i++) { if (s.charAt(i) == 'a') count++; }`

78 Building a Reversed String

Explanation: Construct a new string by prepending each character, or append while traversing backward.

AP CSA Tip: *String result = ""; for (int i = s.length() - 1; i >= 0; i--) { result += s.charAt(i); }*

Example: reversing "abc" gives "cba".

79 The for-each Loop

Explanation: `for (Type var : collection)` iterates every element in order; the loop variable is a copy of each element.

AP CSA Tip: *For-each cannot modify the collection or know the index; use a regular for loop when you need indexes.*

Example: `for (String s : names) { System.out.println(s); }`

80 for-each vs Indexed for

Explanation: for-each is simpler and safer for reading; indexed for is needed for writing, positions, or parallel arrays.

AP CSA Tip: Choose indexed when the index matters; choose for-each for read-only traversal.

Example: To set `arr[i] = 0` for all `i`, you need the indexed loop.

81 Logical Errors vs Runtime Errors

Explanation: Logical errors compile and run but give wrong results; runtime errors (exceptions) crash at execution.

AP CSA Tip: Logic errors = wrong output; runtime = crash (`ArrayIndexOutOfBoundsException`, `NullPointerException`).

Example: Using `=` instead of `==` is a logic error; `arr[5]` on a 5-element array is a runtime error.

82 ArrayIndexOutOfBoundsException

Explanation: Thrown when accessing `arr[i]` with `i` outside `0..length-1`. The classic off-by-one crash.

AP CSA Tip: Valid indexes are 0 to `length-1`; the last element is `arr[arr.length - 1]`.

Example: `int[] a = new int[3]; a[3] = 5; --` throws `ArrayIndexOutOfBoundsException`.

83 NullPointerException

Explanation: Thrown when calling a method or accessing a field on a null reference.

AP CSA Tip: Guard with null checks; an uninitialized object variable is null.

Example: `String s = null; s.length(); --` throws `NullPointerException`.

84 Trace Loop Execution

Explanation: Simulate loops by tracking the loop variable and any accumulators at each iteration.

AP CSA Tip: Make a table: iteration | `i` | sum; predict output before running.

Example: `for (int i = 0; i < 3; i++) { sum += i; } --` sum ends at 3.

85 Algorithm Efficiency Basics

Explanation: Counting iterations: a single loop over `n` items is $O(n)$; nested loops are $O(n^2)$; halving is $O(\log n)$.

AP CSA Tip: Identify the dominant cost: one loop = n , two nested = n^2 .

Example: Two nested loops each of size `n` run $n*n$ times.

86 Choosing the Right Loop

Explanation: Use `for` when the count is known; `while` when the condition is data-driven; `for-each` when reading all elements.

AP CSA Tip: Match the loop to the problem: fixed count -> `for`; unknown stop -> `while`.

Example: Reading until a sentinel needs a `while` loop; printing 1..10 needs a `for` loop.

87 Compound Condition Ordering

Explanation: In `&&` expressions, place the cheaper or guard condition first; short-circuit prevents later evaluation.

AP CSA Tip: Guard before access: `arr != null && arr.length > 0` evaluates safely.

Example: `if (i < arr.length && arr[i] == 7) --` index checked before access.

88 Nested Conditional vs Compound Boolean

Explanation: Nested ifs and compound booleans can express the same logic; choose whichever is clearer.

AP CSA Tip: `A && B` can replace `if (A) { if (B) { ... } }`; `!` operator flips conditions.

Example: `if (x > 0 && y > 0)` is equivalent to the nested version.

89 Variable Update Patterns

Explanation: Common updates: counter (`x++`), accumulator (`sum += v`), tracker (`best = x`), and flag (`found = true`).

AP CSA Tip: *Identify the pattern a problem needs; flags start false and flip true when a condition is met.*

Example: `boolean found = false; ... found = true; -- a flag pattern.`

90 Counting Vowels in a String

Explanation: Traverse the string and count matching characters with an if inside a for loop.

AP CSA Tip: *Use a helper to test vowel membership: `charAt(i) == 'a' || charAt(i) == 'e' || ...`*

Example: `for (int i = 0; i < s.length(); i++) { char c = s.charAt(i); if (isVowel(c)) count++; }`

91 Break vs Return in Loops

Explanation: `break` exits only the loop; `return` exits the entire method (and can return a value).

AP CSA Tip: *Use `return` to hand back an answer found mid-loop; use `break` to stop searching but continue the method.*

Example: `for (int i = 0; i < n; i++) { if (arr[i] == target) return i; } return -1;`

92 Boolean Flags in Loops

Explanation: A flag variable records whether something happened: `boolean found = false`; set it true when the condition holds; check it after the loop.

AP CSA Tip: *Flags summarize loop results; initialize false (or true) depending on the question asked.*

Example: `boolean allPositive = true; for (int x : arr) { if (x <= 0) allPositive = false; }`

93 while(true) Input Loops

Explanation: A loop that runs until a sentinel or break condition stops it, useful when the stop point is unknown.

AP CSA Tip: *`while (true) { int v = sc.nextInt(); if (v == -1) break; process(v); }`*

Example: Reading numbers until -1 sentinel with `while(true)` and `break`.

94 Off-by-One Loop Bugs

Explanation: Loops that run one time too few or too many, usually from using `<=` instead of `<`, or wrong start index.

AP CSA Tip: *Check bounds: for `i < n` visits all `n` elements; `i <= n` visits `n+1` and crashes arrays.*

Example: `for (int i = 0; i <= arr.length; i++) -- accesses arr[arr.length], out of bounds.`

95 Nested Loops for Tables

Explanation: Outer loop rows, inner loop columns, printing a table or grid with values computed from row and column.

AP CSA Tip: *Use the loop variables in the body: row `r`, column `c` gives cell `(r, c)`.*

Example: `for (int r = 1; r <= 3; r++) { for (int c = 1; c <= 3; c++) { print(r * c); } } -- a times table.`

Unit 3: Unit 3: Class Creation

10-18% of the AP CSA exam. Writing classes, encapsulation, constructors, static members, inheritance, and polymorphism.

96 Defining a Class

Explanation: A class bundles instance variables, constructors, and methods. Syntax: `public class Name { ... }` with fields, constructor(s), and methods.

AP CSA Tip: *One public class per file, named after the file; fields describe state, methods describe behavior.*

Example: `public class Student { private String name; public Student(String n) { name = n; } }`

97 Instance Variables (Fields)

Explanation: Variables declared inside the class but outside methods; each object gets its own copy, holding its state.

AP CSA Tip: *Fields persist for the object's lifetime; they are usually private for encapsulation.*

Example: `private int age;` -- every Student object has its own age.

98 Encapsulation

Explanation: Hiding an object's internal data by making fields private and exposing controlled access through public methods.

AP CSA Tip: *Encapsulation = private fields + public methods; it protects invariants and eases change.*

Example: A BankAccount keeps balance private and exposes `deposit()/withdraw()` instead of direct access.

99 Access Modifiers: private vs public

Explanation: private members are accessible only within the class; public members are accessible anywhere.

AP CSA Tip: *Fields = private; methods = public (unless internal helpers). The exam tests this distinction.*

Example: `private double balance; public void deposit(double amt) { ... }`

100 Accessor Methods (Getters)

Explanation: Public methods that return field values: `public int getAge() { return age; }`

AP CSA Tip: *Getters let outsiders READ private data; name them `getXxx()`.*

Example: `public String getName() { return name; }`

101 Mutator Methods (Setters)

Explanation: Public methods that change field values, often with validation: `public void setAge(int a) { if (a > 0) age = a; }`

AP CSA Tip: *Setters let outsiders WRITE private data safely; name them `setXxx()`.*

Example: `public void setScore(int s) { score = Math.max(0, s); }`

102 Constructors Initialize State

Explanation: Constructors set initial field values using parameters; they may call other constructors with `this(...)`.

AP CSA Tip: *Constructors have no return type and the class name; default values apply if fields are not set.*

Example: `public Student(String name, int id) { this.name = name; this.id = id; }`

103 The this Keyword

Explanation: `this` refers to the current object; it disambiguates parameters from fields: `this.name = name;`

AP CSA Tip: *`this.field` resolves the field when a parameter has the same name; `this(...)` calls another constructor.*

Example: `public void setName(String name) { this.name = name; }`

104 Constructor Overloading

Explanation: A class can have multiple constructors with different parameter lists, each initializing state differently.

AP CSA Tip: *Overloaded constructors give flexibility; one can delegate to another with this(...).*

Example: Student() and Student(String n) are overloaded constructors.

105 toString Method

Explanation: A public method returning a String representation of the object; called automatically when printing.

AP CSA Tip: *Override toString to show meaningful state: "Student[name=Ada, id=42]".*

Example: System.out.println(student); calls student.toString().

106 equals Method Override

Explanation: Override equals to define logical equality for objects (by field values, not references).

AP CSA Tip: *A proper equals compares class and fields; == would only compare references.*

Example: public boolean equals(Object other) { ... compare fields ... }

107 Static Variables and Methods

Explanation: static members belong to the CLASS, not any instance; accessed via ClassName.member.

AP CSA Tip: *static = one copy shared by all instances; use static for constants and utility methods.*

Example: Math.sqrt(16) is a static method call; Math.PI is a static constant.

108 Static vs Instance Context

Explanation: Instance methods can access static members, but static methods cannot access instance fields directly (no this).

AP CSA Tip: *Static methods have no object context; they need an instance or parameters to work with object data.*

Example: public static int doubleIt(int x) { return 2 * x; } -- no instance needed.

109 The Main Method

Explanation: public static void main(String[] args) is the program's entry point; it is static because no object exists yet.

AP CSA Tip: *main creates objects and calls their methods to start the program.*

Example: public static void main(String[] args) { new Game().run(); }

110 Mutator vs Accessor Side Effects

Explanation: Mutators change state; accessors return state without changing it (though returning mutable objects can leak references).

AP CSA Tip: *Getters returning object references expose internals; return copies or primitives to be safe.*

Example: public int getScore() { return score; } -- safe read-only access.

111 Immutable Objects

Explanation: Objects whose state cannot change after construction: private final fields, no mutators, defensive copies.

AP CSA Tip: *Immutability simplifies reasoning; Strings are immutable in Java.*

Example: final fields + getters only = an immutable class.

112 Object Composition

Explanation: Objects can contain other objects as fields (has-a relationship), building complex structures.

AP CSA Tip: *Composition = has-a; inheritance = is-a. Prefer composition for flexibility.*

Example: A Car has-a Engine and has-a List<Wheel>.

113 Inheritance (extends)

Explanation: A subclass inherits fields and methods from a superclass with extends, adding or overriding behavior.

AP CSA Tip: *Inheritance = is-a: class Dog extends Animal. The subclass is an Animal.*

Example: `public class SavingsAccount extends BankAccount { ... }`

114 The super Keyword

Explanation: super refers to the superclass: super(...) calls the superclass constructor; super.method() calls an overridden method.

AP CSA Tip: *The superclass constructor runs FIRST; if not called explicitly, the no-arg version runs.*

Example: `public Dog(String name) { super(name); }` -- calls Animal's constructor.

115 Overriding Methods

Explanation: A subclass redefines an inherited method with the same signature, changing its behavior.

AP CSA Tip: *Override = same signature, new body; call super.method() to extend rather than replace.*

Example: `class Dog extends Animal { public void speak() { System.out.println("Woof"); } }`

116 Polymorphism

Explanation: A superclass reference can hold a subclass object, and the CALLED method is the object's actual (overridden) version at runtime.

AP CSA Tip: *Polymorphism = one reference type, many behaviors; it is the basis of dynamic dispatch.*

Example: `Animal a = new Dog(); a.speak();` prints "Woof".

117 Is-a vs Has-a

Explanation: Is-a (inheritance): a Dog IS an Animal. Has-a (composition): a Car HAS an Engine.

AP CSA Tip: *Use extends for is-a, fields for has-a; the exam asks you to choose the right relationship.*

Example: A Student IS-A Person; a School HAS-A List<Student>.

118 Inheritance Hierarchies

Explanation: Classes can form trees: Animal -> Mammal -> Dog. Fields and methods flow down the hierarchy.

AP CSA Tip: *The most general class sits at the top; each subclass adds or refines specifics.*

Example: Animal (top) <- Mammal <- Dog, Cat; Mammal <- Whale, Bat.

119 Method Dispatch with Superclass References

Explanation: When calling a method through a superclass reference, Java uses the object's actual class to find the method.

AP CSA Tip: *Compile-time type vs runtime type: the RUNTIME object decides which method runs.*

Example: `Animal a = new Cat(); a.speak();` -- Cat's speak() runs, not Animal's.

120 Constructors in Inheritance

Explanation: The superclass constructor always runs before the subclass constructor body; super(args) must be the first statement.

AP CSA Tip: *If the superclass has no no-arg constructor, the subclass MUST call super(args) explicitly.*

Example: `class Dog extends Animal { Dog(String n) { super(n); } }`

121 Overloading vs Overriding

Explanation: Overloading = same name, different parameters (new method); overriding = same signature, new body (replaces).

AP CSA Tip: *Overload = new method; override = replace existing. Overriding uses @Override.*

Example: `void draw()` and `void draw(int n)` overload; `Dog.speak()` overriding `Animal.speak()` overrides.

122 Abstract Classes

Explanation: Abstract classes cannot be instantiated; they may declare abstract methods that subclasses must implement.

AP CSA Tip: *abstract = template; concrete subclasses implement the abstract methods.*

Example: `abstract class Shape { abstract double area(); } class Circle extends Shape { ... }`

123 Interfaces (Concept)

Explanation: Interfaces declare method signatures that implementing classes must provide, enabling polymorphism across unrelated classes.

AP CSA Tip: *Interfaces = contracts; a class can implement many interfaces but extend one class.*

Example: `interface Comparable { int compareTo(Object o); }`

124 Visibility and Inheritance

Explanation: private superclass fields are NOT directly accessible in subclasses; use public/protected accessors or protected fields.

AP CSA Tip: *Subclasses inherit what is accessible: public and protected members; private members need accessors.*

Example: Animal's private name is read by Dog via getName(), not name.

125 Designing for Reuse

Explanation: Inheritance and composition let new classes reuse tested code, reducing duplication.

AP CSA Tip: *Design classes with clear responsibilities; test subclasses inherit correct behavior.*

Example: A base Logger class is reused by FileLogger and ConsoleLogger.

126 Class Design on the FRQ

Explanation: The free-response class-design question expects: private fields, constructor(s) initializing state, and public accessor/mutator methods.

AP CSA Tip: *Follow the spec exactly: name methods as given, use the stated parameters, and validate where asked.*

Example: Write fields first, then constructor, then each required method -- check off the prompt's bullets.

127 Testing a Class

Explanation: Create objects, call methods, and verify state changes match expectations, including edge cases.

AP CSA Tip: *Test constructors, getters, setters, and special cases like zero or negative inputs.*

Example: `new BankAccount(100).withdraw(150)` should either reject or overdraft per the spec.

128 The AP Style Guideline

Explanation: AP Java uses a limited subset: no generics in FRQ signatures, no var, and specific class/method names.

AP CSA Tip: *Write AP-style code: standard loops, clear names, no fancy syntax.*

Example: Use `ArrayList<String>` and standard for-each loops as in the AP subset.

129 Field Initialization at Declaration

Explanation: Fields can be initialized where declared: `private int count = 0;` -- every object starts with that value.

AP CSA Tip: *Initialization at declaration runs before the constructor body; constructors can then override it.*

Example: `private double balance = 0.0;`

130 Method Overloading in Your Own Class

Explanation: Your class can define multiple methods with the same name and different parameters, each doing a related job.

AP CSA Tip: *Overloads must differ in parameter types or count; the compiler picks the match.*

Example: `void setColor(int r, int g, int b)` and `void setColor(String name)` are overloads.

131 Copy Constructor

Explanation: A constructor that creates a new object from an existing one, copying its field values.

AP CSA Tip: *Copy constructors give independent copies: changing one object does not affect the copy.*

Example: `public Student(Student other) { this.name = other.name; this.id = other.id; }`

132 Object References as Parameters

Explanation: Passing an object to a method passes a COPY of the reference; the method can mutate the shared object.

AP CSA Tip: *Primitives pass values; objects pass references -- mutations inside the method affect the caller's object.*

Example: `void reset(Point p) { p.x = 0; p.y = 0; }` -- changes the caller's Point.

133 The toString Contract

Explanation: Every class inherits toString from Object; overriding it gives readable output and is called automatically by println.

AP CSA Tip: *Write toString to include the key fields, e.g., "Student[name=Ada, id=42]".*

Example: `System.out.println(student);` prints what `student.toString()` returns.

134 Class Design Checklist

Explanation: From a prompt: (1) identify fields and their types, (2) write the constructor matching the given signature, (3) add each requested method exactly as named.

AP CSA Tip: *Underline the required method names and parameters in the prompt; the FRQ grades against those exact signatures.*

Example: If the prompt asks for `getScore()`, write `public int getScore()`, not `score()`.

135 Testing with a Driver Class

Explanation: A separate main method creates objects and calls methods to verify behavior.

AP CSA Tip: *The driver class is a client: it should only use public methods, never touch private fields.*

Example: `public class StudentDriver { public static void main(String[] args) { Student s = new Student("Ada"); ... } }`

Unit 4: Unit 4: Data Collections

30-40% of the AP CSA exam. Arrays, ArrayLists, 2D arrays, traversals, searching, sorting, and recursion -- the biggest unit.

136 Arrays

Explanation: Fixed-size, indexed collections of elements of ONE type: `int[] arr = new int[5]`; Elements default to 0, null, or false.

AP CSA Tip: *Array size is set at creation and cannot change; indexes run 0 to length-1.*

Example: `int[] scores = new int[10]`; -- an array of 10 ints, all 0.

137 Array Initialization

Explanation: Arrays can be created with initial values: `int[] a = {1, 2, 3}`; or with new and explicit values.

AP CSA Tip: *The shortcut {1, 2, 3} can only be used at declaration; otherwise use new int[] {1, 2, 3}.*

Example: `String[] names = {"Ada", "Bob", "Cy"};`

138 Accessing and Modifying Array Elements

Explanation: Use `arr[index]` to read or write: `arr[0] = 5`; `int x = arr[2]`;

AP CSA Tip: *Index bounds: 0 to length-1. `arr[arr.length - 1]` is the last element.*

Example: `scores[2] = 97`; updates the third element.

139 The length Attribute

Explanation: Arrays expose length (a field, not a method): `arr.length`. Strings use `length()` (a method).

AP CSA Tip: *`arr.length` has no parentheses; `s.length()` does. Mixing them up is a classic exam trap.*

Example: `for (int i = 0; i < arr.length; i++) { ... }`

140 Array Traversal

Explanation: Visiting every element, typically with an indexed for or for-each loop.

AP CSA Tip: *For-each reads cleanly; indexed for is needed when the index matters or elements are modified.*

Example: `for (int value : arr) { total += value; }`

141 Array Algorithms: Sum and Average

Explanation: Sum all elements, then divide by length for the average (cast to double to avoid truncation).

AP CSA Tip: *`double avg = (double) total / arr.length`; -- cast BEFORE dividing.*

Example: `int total = 0`; `for (int x : arr) total += x`; `double avg = (double) total / arr.length`;

142 Array Algorithms: Min and Max

Explanation: Track the smallest/largest seen: initialize with the first element, compare the rest.

AP CSA Tip: *Start with `arr[0]` or `Integer.MAX_VALUE/MIN_VALUE`; check every element once.*

Example: `int min = arr[0]`; `for (int x : arr) if (x < min) min = x`;

143 Array Algorithms: Count and Find

Explanation: Count elements meeting a condition, or find the first index of a target.

AP CSA Tip: *Count: increment a counter. Find: return the index immediately, or -1 if never found.*

Example: `for (int i = 0; i < arr.length; i++) if (arr[i] == target) return i`; return -1;

144 Array Algorithms: Shift and Rotate

- Explanation:** Moving elements left or right: copy each element to its neighbor position.
- AP CSA Tip:** *Shifts need a temporary variable to avoid overwriting; rotate wraps around the end.*
- Example:** `for (int i = 0; i < arr.length - 1; i++) arr[i] = arr[i + 1];` -- shifts left.
-

145 Parallel Arrays

- Explanation:** Multiple arrays of the same length storing related data by index, e.g., `names[]` and `scores[]`.
- AP CSA Tip:** *Parallel arrays share the SAME index; `names[i]` matches `scores[i]`.*
- Example:** `String[] names` and `int[] scores`, where `scores[i]` belongs to `names[i]`.
-

146 ArrayLists

- Explanation:** Resizable lists of objects: `ArrayList<Integer> nums = new ArrayList<>();` They grow and shrink automatically.
- AP CSA Tip:** *ArrayList only stores OBJECTS -- use wrapper classes (*Integer*, *Double*), not primitives.*
- Example:** `ArrayList<String> words = new ArrayList<String>();`
-

147 ArrayList Methods: add

- Explanation:** `list.add(obj)` appends; `list.add(index, obj)` inserts at a position, shifting later elements right.
- AP CSA Tip:** *add at the end is $O(1)$; insert in the middle shifts elements.*
- Example:** `words.add("hi"); words.add(0, "start");`
-

148 ArrayList Methods: get and set

- Explanation:** `list.get(i)` returns the element at index `i`; `list.set(i, obj)` replaces it and returns the old value.
- AP CSA Tip:** *ArrayList uses *get/set*, NOT brackets: `list.get(i)`, never `list[i]`.*
- Example:** `String first = words.get(0); words.set(0, "HELLO");`
-

149 ArrayList Methods: remove and size

- Explanation:** `list.remove(i)` removes by index (returning the element); `list.remove(obj)` removes the first match; `list.size()` returns the count.
- AP CSA Tip:** *`size()` has parentheses (unlike array length); removing shifts elements left.*
- Example:** `words.remove(0); int n = words.size();`
-

150 ArrayList Traversal

- Explanation:** Use *for-each* to read; use an indexed loop when removing or modifying elements.
- AP CSA Tip:** *Removing inside a *for-each* throws *ConcurrentModificationException* -- traverse backward or collect removals.*
- Example:** `for (int i = list.size() - 1; i >= 0; i--) { if (bad(list.get(i))) list.remove(i); }`
-

151 ArrayList of Primitives (Wrappers)

- Explanation:** Store `int` as `Integer` via autoboxing: `ArrayList<Integer>`; the list handles the wrapping automatically.
- AP CSA Tip:** *Autoboxing converts *int* to *Integer* on add; unboxing converts back on get.*
- Example:** `ArrayList<Integer> nums = new ArrayList<>(); nums.add(5); int x = nums.get(0);`
-

152 Array vs ArrayList

- Explanation:** Arrays: fixed size, primitives OK, bracket access. ArrayLists: resizable, objects only, method access.
- AP CSA Tip:** *Choose *ArrayList* when size changes; choose *array* for primitives or fixed-size data.*
- Example:** Scores that grow as students register need an `ArrayList`; a fixed calendar year needs an `array`.
-

153 ArrayList Algorithms

Explanation: The standard algorithms (sum, min/max, count, find) all work on ArrayLists with `get(i)/size()`.

AP CSA Tip: Translate array algorithms: `arr[i] -> list.get(i)`, `arr.length -> list.size()`.

Example: `int max = list.get(0); for (int v : list) if (v > max) max = v;`

154 2D Arrays

Explanation: Arrays of arrays: `int[][] grid = new int[rows][cols];` Accessed as `grid[r][c]`.

AP CSA Tip: `grid.length = number of rows`; `grid[0].length = number of columns`.

Example: `int[][] m = new int[3][4];` -- 3 rows, 4 columns.

155 2D Array Traversal

Explanation: Nested loops visit every cell: outer loop rows, inner loop columns.

AP CSA Tip: Row-major: outer *r*, inner *c*. Column-major: outer *c*, inner *r*. Match the question's order.

Example: `for (int r = 0; r < grid.length; r++) { for (int c = 0; c < grid[0].length; c++) { ... } }`

156 Row-Major vs Column-Major Order

Explanation: Row-major processes each full row before the next; column-major processes each column top to bottom.

AP CSA Tip: Identify which order a problem wants by what it asks to compute (row sums vs column sums).

Example: Row sums use row-major order; column sums need column-major traversal.

157 2D Array Algorithms

Explanation: Row sums, column sums, totals, and searches across a grid, using nested loops.

AP CSA Tip: For each cell `grid[r][c]`, decide whether the action is per-row, per-column, or global.

Example: `int total = 0; for (int[] row : grid) { for (int v : row) total += v; }`

158 2D Array as Table of Data

Explanation: 2D arrays model tables: rows are records, columns are fields (like a spreadsheet).

AP CSA Tip: Keep row/column semantics straight; index errors are the most common bug.

Example: A seating chart `grid[row][col]` maps to row and seat number.

159 Enhanced for with 2D Arrays

Explanation: `for (int[] row : grid) { for (int v : row) { ... } }` visits every cell without indexes.

AP CSA Tip: The outer loop variable is an entire row (an array); the inner one is each cell.

Example: `for (int[] row : grid) { for (int v : row) total += v; }`

160 Searching: Linear vs Binary

Explanation: Linear search scans in order ($O(n)$); binary search halves a SORTED collection ($O(\log n)$).

AP CSA Tip: Binary search requires sorted data; linear works on anything.

Example: Binary search on a sorted array of 1,000,000 needs about 20 comparisons.

161 Binary Search on Arrays

Explanation: Repeatedly compare the middle element, narrowing the search range by half each step.

AP CSA Tip: Update `low/high` to `mid +/- 1`; stop when `low > high` (not found).

Example: Searching 1-100: guess 50, then 25 or 75, halving each time.

162 Selection Sort

Explanation: Find the smallest element and swap it into position, then repeat on the rest; $O(n^2)$ always.

AP CSA Tip: *Selection sort: find min, swap; it is simple but slow on large data.*

Example: Sorting [5, 3, 8, 1]: first pass swaps 1 into position 0.

163 Insertion Sort

Explanation: Build the sorted part one element at a time, inserting each new element into its correct place; $O(n^2)$ worst, $O(n)$ on nearly-sorted data.

AP CSA Tip: *Insertion sort is fast on nearly-sorted data; it shifts elements right to make room.*

Example: Inserting 4 into [2, 3, 5] shifts 5 right to make [2, 3, 4, 5].

164 Merge Sort

Explanation: A divide-and-conquer sort: split in half, sort each half, merge them; guaranteed $O(n \log n)$.

AP CSA Tip: *Merge sort = split + sort + merge; it needs extra space for merging.*

Example: Splitting [4,2,7,1] into halves, sorting, and merging gives [1,2,4,7].

165 Sorting Stability and Efficiency

Explanation: Stable sorts keep equal elements in original order; efficiency matters most for large data.

AP CSA Tip: *For AP, compare big-O: selection/insertion $O(n^2)$, merge $O(n \log n)$.*

Example: Merge sort's $O(n \log n)$ beats selection sort's $O(n^2)$ on large arrays.

166 Recursion

Explanation: A method that calls itself, solving smaller versions of the same problem until a base case stops it.

AP CSA Tip: *Recursion needs a base case (stopping condition) and a recursive call that moves toward it.*

Example: $\text{factorial}(n) = n * \text{factorial}(n - 1)$, with $\text{factorial}(1) = 1$ as the base case.

167 Base Case and Recursive Case

Explanation: The base case returns without recursing; the recursive case calls the method with a smaller/simpler input.

AP CSA Tip: *Without a base case, recursion causes StackOverflowError; every call must approach the base.*

Example: In factorial, $n == 1$ is the base case; $n > 1$ recurses.

168 Recursive Factorial

Explanation: $\text{factorial}(n) = n * \text{factorial}(n - 1)$, base $\text{factorial}(1) = 1$.

AP CSA Tip: *Trace carefully: $\text{factorial}(4) = 4 * 3 * 2 * 1 = 24$; each call waits for the next.*

Example: $\text{factorial}(3) = 3 * \text{factorial}(2) = 3 * 2 * \text{factorial}(1) = 6$.

169 Recursive Sum and Count

Explanation: Recursive methods can sum an array: $\text{sum}(\text{arr}, n) = \text{arr}[n-1] + \text{sum}(\text{arr}, n-1)$, base $\text{sum}(\text{arr}, 0) = 0$.

AP CSA Tip: *Express the problem as: current piece + recursive call on the rest.*

Example: $\text{sum of } [1,2,3] = 3 + \text{sum of } [1,2] = 3 + 2 + \text{sum of } [1] = 6$.

170 Fibonacci Recursion

Explanation: $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$, with base $\text{fib}(1) = \text{fib}(2) = 1$. It grows exponentially without memoization.

AP CSA Tip: *Fibonacci is the classic recursion example; compute small values by hand on the exam.*

Example: $\text{fib}(5) = \text{fib}(4) + \text{fib}(3) = 3 + 2 = 5$.

171 Tracing Recursive Calls

Explanation: Draw the call stack: each call waits for the calls it makes before returning.

AP CSA Tip: Evaluate the innermost call first, then unwind outward.

Example: $\text{pow}(2, 3) = 2 * \text{pow}(2, 2) = 2 * 2 * \text{pow}(2, 1) = 8$.

172 Recursion vs Iteration

Explanation: Anything recursive can be iterative (and vice versa); iteration is usually more efficient, recursion more elegant.

AP CSA Tip: The exam wants you to recognize and trace both; recursion is tested with small inputs.

Example: A sum loop is iterative; $\text{sum}(n) = n + \text{sum}(n-1)$ is recursive.

173 String Recursion

Explanation: Recursive string methods process `charAt(0)` and recurse on `substring(1)`, building toward the empty string.

AP CSA Tip: Base case: empty string. Recursive case: handle first char, recurse on the rest.

Example: length of `s` = 1 + length of `s.substring(1)`, base 0 for "".

174 StackOverflowError

Explanation: Thrown when recursion never reaches a base case and the call stack overflows.

AP CSA Tip: Check that each recursive call reduces the problem size; missing base cases cause it.

Example: factorial with no base case on `n = 100000` overflows the stack.

175 Analyzing Recursive Code on the FRQ

Explanation: FRQ recursion questions ask you to trace output or complete a recursive method per a spec.

AP CSA Tip: Write the base case first, then the recursive case matching the spec's example.

Example: If a spec says 'stop when index reaches length', that is your base case.

176 Array of Objects

Explanation: Arrays can hold object references: `Student[] roster = new Student[10]`; each slot defaults to null.

AP CSA Tip: Create each element before using it: `roster[i] = new Student(...)`.

Example: `Student[] roster = new Student[3]; roster[0] = new Student("Ada");`

177 ArrayList Remove by Value

Explanation: `list.remove(obj)` removes the first occurrence of the object; `list.remove(index)` removes by position.

AP CSA Tip: Overload ambiguity: `list.remove(5)` removes INDEX 5, not the Integer 5; use `Integer.valueOf(5)` to remove by value.

Example: `words.remove("hi")` removes the first "hi".

178 Empty Collection Handling

Explanation: Algorithms on empty arrays/lists must behave: sums are 0, max/min need care, searches return -1.

AP CSA Tip: For min/max on possibly-empty collections, handle the empty case before computing.

Example: if `(list.isEmpty())` return -1; before searching.

179 2D Array Dimensions

Explanation: `grid.length` is the row count; `grid[r].length` is the column count of row `r` (rows can differ in length).

AP CSA Tip: For rectangular grids, `grid[0].length` works for every row; check row lengths in ragged arrays.

Example: `int rows = grid.length; int cols = grid[0].length;`

180 2D Array Search

Explanation: Finding a target in a grid: nested loops compare each cell and return or set a flag when found.

AP CSA Tip: *Return the coordinates or a boolean; stop early once found.*

Example: `for (int r = 0; r < grid.length; r++) { for (int c = 0; c < grid[0].length; c++) { if (grid[r][c] == target) return true; } } return false;`

181 Binary Search as Recursion

Explanation: Binary search can be written recursively: search the half that could contain the target.

AP CSA Tip: *Recursive binary search halves the range each call: base case low > high means not found.*

Example: `binarySearch(arr, low, mid - 1)` or `(mid + 1, high)` depending on the comparison.

182 Merge Sort Space

Explanation: Merge sort uses extra memory (an auxiliary array) for merging, unlike in-place sorts.

AP CSA Tip: *Trade-off: merge sort is $O(n \log n)$ time but $O(n)$ space; selection/insertion sorts are $O(1)$ space.*

Example: Merging two halves needs a temporary array of size n .

183 ArrayList Capacity vs Size

Explanation: ArrayList has an internal capacity that grows automatically; `size()` reports the logical element count.

AP CSA Tip: *Size and capacity are different: capacity grows, size is what you put in.*

Example: `new ArrayList<String>()` starts with size 0 regardless of capacity.

184 Traversing 2D Arrays with for-each

Explanation: `for (int[] row : grid) { for (int v : row) { ... } }` visits every cell safely.

AP CSA Tip: *The outer loop variable is a whole row; use it for row-level operations.*

Example: `for (int[] row : grid) { int rowSum = 0; for (int v : row) rowSum += v; }`

185 Common Array Misconceptions

Explanation: Arrays are objects; `arr.length` has no parentheses; the size is fixed; elements are accessed by index.

AP CSA Tip: *Mixing `length/length()`, starting at 1, or using `==` on arrays are the classic traps.*

Example: `int[] a = {1, 2}; int[] b = a; -- a and b reference the same array.`

What's Next?

You have now reviewed all four units of the AP Computer Science A course, from primitive types to recursion. Go back to the Table of Contents, find the units you marked, and reread just those tip lines. Two weeks out, do a rapid pass over the entire guide; the day before, review only the tip lines. On the free-response questions, read the spec twice, use the exact method signatures given, write the simplest correct solution, and trace your code with the sample data before moving on.

Pair it with the Study Mastery Cheat Sheet (\$19) -- how to actually retain all of it

Also available: AP Computer Science Principles, AP Biology, AP Calculus AB, and more study guides

templateforge.com