The Simple AP® Computer Science A Unit 1 Cheat Sheet

1.1–1.2 Programs, Variables & Types

Algorithms & Compiling

Algorithm: step-by-step steps to solve a problem.
Compiler: checks code + translates it to run.

Error	When found	Example
Syntax	Won't compile	Missing ;
Logic	Wrong result	+ not *
Run-time	Crashes	int ÷ 0

Primitive Types (AP subset)

Type	Holds	Example
int	Whole numbers	-7, 42
double	Decimals	2.5
boolean	true / false	true

Reference type: stores the address of an object (e.g., **String**). No object yet = **null**.
Literal: a fixed value written in code (**5**, **2.5**, **"hi"**).
Strongly typed: Java checks types at compile time, before the code ever runs.

Declaring Variables

Declare + initialize: **int score = 0;**
Names: case-sensitive, no spaces or keywords; use **camelCase** (e.g., totalScore).
final: a constant that can't change after it's set.
final double TAX = 0.08; (ALL CAPS)
double d = 7; works: stores 7.0.
Use before assigning → compile error.

⚠ int x = 3.5; won't compile. A double doesn't fit in an int unless you cast it: (int) 3.5.

✓ "Won't compile" Qs: check the types first.

1.3–1.4 Expressions, Output & Assignment

Arithmetic Operators

Operators: + − * / %. *** / %** go before **+ −**.
Integer division: int / int drops the decimal. **7 / 2 = 3**. With a double: **7 / 2.0 = 3.5**, **7.0 / 2 = 3.5**.
Modulus %: remainder. **17 % 5 = 2**. Even test: **n % 2 == 0**. Last digit: **n % 10**; drop it: **n / 10**.
int ÷ 0 → **ArithmeticException** (crash).
Order: **5 + 10 % 3 * 2** → 1 * 2 = 2 → **7**.
Negatives: **−7 / 2 = −3** (truncates toward 0).

Expression	Result
10 / 4	2
10 / 4.0	2.5
10 % 4	2
2 + 3 * 4	14
1 / 2 * 4.0	0.0

Output & Strings in Expressions

print: same line. **println**: moves to a new line after.
+ with a String joins text, left to right: **"A" + 1 + 2 = "A12"**; **1 + 2 + "A" = "3A"** (adds first).
println("x" + 3 * 4) → **x12** (* happens before +).
Escape sequences: **\"** = quote, **** = backslash, **\n** = new line, all inside a String literal.

Assignment & Input

= stores the right side in the left variable. **x = x + 1;** is fine: the right side is evaluated first, then stored.

⚠ 1 / 2 * 4.0 = 0.0, not 2.0. Integer division (1 / 2 = 0) happens first, before any double.

1.5–1.6 Casting, Range & Compound Ops

Casting

Widening (automatic): int → double. **double d = 5;** stores 5.0; no cast is needed for this.
Narrowing (needs a cast): **(int) 3.99 = 3**. Casting cuts off the decimal; it never rounds.
Round (x ≥ 0): **(int)(x + 0.5)**; e.g., 2.6 → 3.
Cast first? (double) 7 / 2 = 3.5 but **(double)(7 / 2) = 3.0**: the cast came too late to help.

Expression	Result
(int) 9.7	9
(double) 9 / 2	4.5
(double)(9 / 2)	4.0
(int)(2.6 + 0.5)	3

Range & Precision

Overflow: going past **Integer.MAX_VALUE** wraps around to a negative number. No error.
Doubles are approximate (0.1 + 0.2 ≠ 0.3 exactly), so don't compare doubles w/ ==.

Compound Assignment

Operator	Same as
x += 3;	x = x + 3;
x -= 3;	x = x - 3;
x *= 3;	x = x * 3;
x /= 3;	x = x / 3;
x %= 3;	x = x % 3;
x++;	x = x + 1;

⚠ Trace: update variables after every line.

1.7–1.10 APIs, Comments & Methods

APIs & Libraries

Library: prewritten classes. **API**: docs that explain how to use their classes & methods.
Class = attributes (data) + behaviors (methods).

Comments

Comments: **//** one line, **/* */** many lines, **/** */** Javadoc. The compiler ignores all of them.
Precondition: must be true before a call (not checked). **Postcondition**: true after it runs.
Return type: **void** = nothing back; **int**, **double**, etc. must **return** a value of exactly that type.

Method Signatures

Signature = method name + parameter list. **double area(double r)** returns a double.

Term	Meaning
Parameter	Variable in header
Argument	Value you pass in
void	Returns nothing
Overloading	Same name, diff. params

Calling Class (static) Methods

Static methods: **ClassName.method()**, e.g., **Math.sqrt(16)**. Use the returned value.
Call by value: changing a parameter inside a method doesn't change the caller's variable.

⚠ Arguments must match the parameters in number, order, and type, or no compile.

✓ Overloads differ by parameter types or count, never by the return type alone.

1.11 The Math Class

Key Methods (on Quick Ref)

Call	Returns	Example
Math.abs(x)	|x|	abs(-4) → 4
Math.pow(b, e)	b^e	pow(2,3) → 8.0
Math.sqrt(x)	√x	sqrt(9) → 3.0
Math.random()	0.0 ≤ r < 1	0.73...

pow and **sqrt** always return a **double**, so cast to store the result in an int: **(int) Math.sqrt(n)**.
Math.abs: int in → int out; double in → double out.
Math.pow(2, 3) = 8.0, not 8: it's a double.

Random Integers

Random int: **(int)(Math.random() * range) + min**, where range = max − min + 1 values.

Goal	Code
0–9	(int)(Math.random() * 10)
1–6	(int)(Math.random() * 6) + 1
5–15	(int)(Math.random() * 11) + 5

(int) Math.random() * 10 is always 0, b/c the cast happens first. Cast the whole product in ( ).

Common Uses

Difference: **Math.abs(a − b)**. **Distance**: sqrt of the sum of the squared differences.
Round to 2 decimals: **(int)(x * 100 + 0.5) / 100.0**.

⚠ Math.random() never returns 1.0, so * 10 gives 0–9, never 10. Add 1 to get 1–10 instead.

✓ Check a range by plugging in 0.0 and 0.999.

1.12–1.13 Objects & Instantiation

Classes vs. Objects

Class: blueprint. **Object**: one **instance** made from it, w/ its own separate copy of the data.
State: an object's current data. **Behavior**: its methods, which can change state.

Creating Objects

Create w/ **new**: **Dog d = new Dog("Rex", 3);** The **constructor** sets starting values.
Overloaded constructors: same name as the class, different parameter lists (count or types).
Arguments must match a constructor's parameters, or it won't compile.

References & Memory

Aliasing: **Dog d2 = d;** → both point to the same object; a change via d2 shows up via d.
null: no object. Calling a method on null → **NullPointerException** (crash at run time).
Reassign: **d = new Dog();** points d at a new object.

	Primitive	Reference
Stores	Value	Address
Copy (=)	Copies value	Same object
Default	0 / false	null

Strings Are Objects

String is a class: **new String("hi")** works, but literals like **"hi"** are the usual way.

⚠ Dog d; makes no object. Only **new** creates an object; until then, d holds null.

✓ Draw a box for each object and an arrow from each reference variable to its box.

1.14–1.15 Instance Methods & Strings

Instance Methods

Instance method: **obj.method()**. Needs a real object, not null. Use the value it returns.

String Basics

Strings are immutable: methods return a new String. To keep it: **s = s.substring(1);**
Index runs 0 to **length() − 1**. Too big → **StringIndexOutOfBoundsException**.
String literal: **String s = "hi";** The empty string **""** has length 0, but it is not null; methods still work.

s = "APCSA"	Result
s.length()	5
s.substring(1, 3)	"PC"
s.substring(2)	"CSA"
s.indexOf("CS")	2
s.indexOf("Z")	-1
s.equals("apcsa")	false

substring(a, b): stops *before* b; length = b − a. One char at i: **s.substring(i, i + 1)**.
s.substring(s.length()) = **""**: legal, no crash.
compareTo: negative if s comes first alphabetically, 0 if equal, positive if after.
indexOf finds the *first* match. Not found → **-1**.
length() needs ( ): it's a method call.

⚠ Compare Strings with .equals(), never ==.

✓ Write the index numbers under each letter before you use substring or indexOf.

Code Tracing & Exam Strategy

Exam Format

MCQ: 42 Qs, 90 min (55%). **FRQ**: 4 Qs, 90 min (45%). Java Quick Reference given.
Unit 1 = 15–25% of the exam, and its skills (types, Strings, Math) show up in every FRQ.
Pacing: ~2 min per MCQ. Skip long traces, then come back to them at the very end.

Top Unit 1 Traps

Trap	Fix
int / int	Cuts decimal
s1 == s2	Use .equals()
substring end	Not included
Ignored return	Reassign it
null.method()	Crash

Tracing Method

Trace: make a table w/ one column per variable; update it after every line of code.
Track types: note int vs. double for every value.
Read the header: name, params, return type.

FRQ Tips

FRQ: call methods you wrote in earlier parts. Return the type the header declares.
Partial credit: write code even if unsure; correct logic earns points, small typos don't cost.

⚠ If the method says return, don't print.

✓ Know how each Quick Ref method behaves (return type, end index), not just its name.