```java
// ---- Avery's Java Cheatsheet ---------------------------------

// ---- Template code -------------------------------------------

import java.util.*; // Import a bunch of useful stuff

class Main { // Should match file name (e.g. Main.java)
  public static void main(String[] args) {
    // Your code goes here
  }
}

// ---- Variables -----------------------------------------------

int myVar1; // Declaration
myVar1 = 4; // Assignment

int myVar2 = 5; // Declaration + assignment
var myVar3 = 6; // Type inferrence

final int x = 1; // Final variables and fields cannot be reassigned
x = 2; // Error

// ---- Primitives ----------------------------------------------

int i = 10;       // Whole number
double d = 3.14;  // Decimal number
boolean b = true; // true or false
char c = 'x';     // Single character

// Compare primitives with `==` and `!=`
i == 20 // false
i !=  0 // true

// Use casts to convert between primitive types
(double) 2 // 2.0
(int) 2.8  // 2   (truncates value)
(int) 'a'  // 97  (unicode value)
(char) 98  // 'b'

// ---- Math ----------------------------------------------------

int n = 10;
n + 4          // 14
n - 4          // 6
n * 4          // 40
n / 4          // 2
n / 4.0        // 2.4
n % 7          // 3
Math.pow(n, 4) // 10000.0
n > 4          // true
n <= 4         // false

int x = 0;
x++;    // x = 1
x += 4; // x = 5
x--;    // x = 4
x *= 2; // x = 8

// ---- Strings -------------------------------------------------

String s = "Hi Java";
s.equals("Hi Java") // true
s == "Hi Java"      // Unreliable, use equals()
s.split(" ")        // String[2] { "Hi", "Java" }
s.length()          // 7
s.charAt(5)         // 'v'
s + " :)"           // "Hi Java :)"

// See also: endsWith, indexOf, repeat, startsWith, strip,
// toCharArray, toLowerCase, toUpperCase

// ---- Defaults ------------------------------------------------

int defaultInt;        // defaultInt = 0
defaultInt + 4         // 4
String defaultString;  // defaultString = null
defaultString.length() // Error

// ---- Output --------------------------------------------------

String name = "Avery";
System.out.println("Hello " + name); // Prints "Hello Avery"

// ---- Input ---------------------------------------------------

Scanner scan = new Scanner(System.in); // New Scanner object
System.out.print("Enter age: ");       // (optional prompt)
String name = scan.nextLine();         // Get String input
System.out.print("Enter age: ");       // (optional prompt)
int age = scan.nextInt();              // Get int input

// ---- Random --------------------------------------------------

Random rand = new Random(); // New Random generator object
rand.nextInt(11)            // Random integer 0 through 10
rand.nextDouble()           // Random double 0.0 through 1.0
rand.nextBoolean()          // Random boolean

// ---- File IO -------------------------------------------------

import java.nio.file.*; // Add this import to the top of your file

// You'll need to add a `throws Exception` annotation to your
// method or use try/catch to avoid an `unreported exception` error

// Read lines as a List of Strings
List<String> lines = Files.readAllLines(Path.of("something.txt"));

// Read whole file as a String
String text = Files.readString(Path.of("something.txt"));

// ---- If ------------------------------------------------------

if (number > 0) {
  System.out.println("positive");
} else if (number < 0) {
  System.out.println("negative");
} else {
  System.out.println("zero");
}

// ---- Ternary -------------------------------------------------

int number = 7;
// Choose between two values based on a condition
String parity = number % 2 == 0 ? "even" : "odd"; // parity = "odd"

// ---- While ---------------------------------------------------

while (n > 10) {
  // Loop while condition is true
}

// See also: do-while loops, break, continue

// ---- For -----------------------------------------------------

for (int i = 1; i <= 10; i++) {
  // Loop over a range of numbers
}

for (int i = 0; i < names.length; i++) {
  // Loop over array indices (use `.size()` for a list)
}

for (int i = names.length - 1; i >= 0; i--) {
  // Loop over array indices in reverse (use `.size()` for a list)
}

for (String name : names) { // `names` can be a List or Array
  // Loop through a sequence of values without using indices
}

// See also: break, continue

// ---- Scope ---------------------------------------------------

int a = 0;
int b = 0;

if (true) {
  a = 1;     // Update a in outer scope
  int b = 1; // New variable b in inner scope
  int c = 1; // New variable c in inner scope
}
```

```
System.out.println(a); // Prints 1
System.out.println(b); // Prints 0 (b was only updated inside if)
System.out.println(c); // Error    (c is only defined inside if)

// ---- Arrays -------------------------------------------------

String[] dirs = { "north", "south", "east", "west" };

System.out.println(dirs); // Prints nonsense

// Prints { "north", "south", "east", "west" }
System.out.println(Arrays.toString(dirs));

dirs.length                       // 4
dirs[0]                           // "north"
dirs[dirs.length - 1]             // "west"
dirs[random.nextInt(dirs.length)] // Random direction
String.join("->", dirs)           // "north->south->east->west"

dirs[0] = "up"; // dirs = { "up", "south", "east", "west" }

// Arrays have a fixed length, no equivalent of Python's `append`

// ---- Lists --------------------------------------------------

List<String> dirs = List.of("north", "south", "east", "west");

dirs.size()                           // 4
dirs.get(0)                           // "north" (can't use [0])
dirs.get(dirs.size() - 1)             // "west"
dirs.get(random.nextInt(dirs.size())) // Random direction
String.join("->", dirs)               // "north->south->east->west"

// List.of makes an immutable list that we can't update or add to
dirs.set(0, "up"); // `UnsupportedOperationException`

// To make a mutable List, use `new ArrayList<>(...)`
List<String> mutDirs = new ArrayList<>(dirs);

mutDirs.set(0, "up"); // mutDirs = [up, south, east, west]
mutDirs.add("down");  // mutDirs = [up, south, east, west, down]

// See also: addAll, contains, equals, getLast, indexOf, isEmpty,
// lastIndexOf, removeLast, reversed, sort, subList

// We can't declare a list of primitives, we have to use wrapper
// classes like Integer, Double, Boolean, and Character

List<char> letters = List.of('a', 'b', 'c');      // Doesn't work
List<Character> letters = List.of('a', 'b', 'c'); // Works

// ---- Sets ---------------------------------------------------

Set<Character> letters = new HashSet<>();

letters.add('a');     // letters = [a]
letters.add('b');     // letters = [a, b]
letters.size()        // 4
letters.contains('a') // true
letters.remove('b');  // letters = [b]

// See also: containsAll, equals, isEmpty, removeAll, retainAll

// ---- Maps ---------------------------------------------------

Map<String, Integer> bank = new HashMap<>();

bank.put("Lamar", 100);     // bank = {Lamar=100}
bank.put("Zay", 50);        // bank = {Zay=50, Lamar=100}
bank.put("Lamar", 200);     // bank = {Zay=50, Lamar=200}
bank.get("Lamar")           // 200
bank.containsKey("Patrick") // false

// See also: compute, containsValue, entrySet, equals,
// getOrDefault, isEmpty, keySet, putAll, putIfAbsent, remove,
// size, values

// ---- Aliasing -----------------------------------------------

int[] a = { 0, 1 };
int[] b = a; // both variables point to the same array object
b[0] = 7;    // `a` and `b` both get updated

int[] c = a.clone(); // `c` is an independent copy
c[0] = 8;            // `c` changes, `a` doesn't

a.clone()            // get a copy of an array `a`
new ArrayList<>(l)   // get a copy of a list `l`
new HashMap<>(m)     // get a copy of a map `m`
new HashSet<>(s)     // get a copy of a set `s`

// ---- Enums --------------------------------------------------

// Use enums to represent sets of options, like compass directions
enum Direction {
  NORTH, SOUTH, EAST, WEST
}

Direction heading = Direction.NORTH;
// Compare Enums with `==` and `!=`
heading != Direction.WEST // true

// ---- Switch -------------------------------------------------

// Use `switch` to check a value against a set of options
switch (fileExtension) {
  case "mp3":
    System.out.println("It's autio");
    break; // Need `break`, `return`, or `yield` at end of case
  case "jpg":
  case "png":
  case "webp": // Multiple cases grouped together
    System.out.println("It's an image");
    break;
  default: // Default runs if no other case matches
    System.out.println("File type unknown");
}

// ---- Methods ------------------------------------------------

static void greet(String name) { // No return value
  System.out.println("Hello " + name);
}

static int add(int a, int b) { // Returns an int
  return a + b;
}

static double abs(double n) { // Returns a double
  if (n < 0) {
    return -n; // Return ends the method immediately
  }

  return n;
}

// ---- Classes ------------------------------------------------

class Rectangle {
  double length; // Instance field
  double width;

  static final int SIDES = 4; // Static constant

  // Constructor method
  Rectangle(double length, double width) {
    this.length = length;
    this.width = width;
  }

  // Factory method
  static Rectangle UnitSquare() {
    return new Rectangle(1, 1);
  }

  // Mutator method
  void scale(double factor) {
    length *= factor;
    width *= factor;
  }

  // Transformer method
  Rectangle rotated() {
    return new Rectangle(width, length);
  }
```

```
  // Observer method
  double area() {
    return length * width;
  }

  // You can define custom `toString`, `equals`, and `hashCode`
  // methods to change how a class is displayed, compared, and
  // hashed, but it's often easier to just use a record class
  //
  // See also: Objects.equals, Objects.hash
}

Rectangle rect = new Rectangle(4, 5);
rect.length // 4.0
rect.width  // 5.0
rect.area()  // 20.0

Rectangle unit = Rectangle.UnitSquare(); // Static method call
rect.length // 1.0
rect.width  // 1.0
unit.area() // 1.0

Rectangle.SIDES // 4

// ---- Records -------------------------------------------------

// Records are immutable classes with auto-generated constructor,
// `toString`, `equals`, `hashCode`, and getter methods

record Point(double x, double y) {
  public double distanceTo(Point other) {
    double dx = x - other.x;
    double dy = y - other.y;
    return Math.sqrt(dx * dx + dy * dy);
  }
}

Point p = new Point(0, 3);
// Use method syntax `x()` and `y()` to access fields
p.x() // 3.0
p.distanceTo(new Point(4, 0)) // 5.0

// ---- Interfaces ----------------------------------------------

interface Shape {
  double area();
}

class Circle implements Shape {
  double radius;

  Circle(double radius) {
    this.radius = radius;
  }

  public double area() {
    return Math.PI * radius * radius;
  }
}

class Rectangle implements Shape {
  double length;
  double width;

  Rectangle(double length, double width) {
    this.length = length;
    this.width = width;
  }

  public double area() {
    return length * width;
  }
}

// Different objects can implement same interface (polymorphism)
List<Shape> shapes = List.of(new Circle(10), new Rectangle(4, 5));

// ---- Generics ------------------------------------------------

// Get a random value from a generic list of any type
static <T> T getRandom(List<T> values) {
  var index = new Random().nextInt(values.size());
  return values.get(index);
}

getRandom(List.of(0, 1)) // 0 or 1
getRandom(List.of("a", "b")); // "a" or "b"

// Class that holds two values of any type
class Pair<A, B> {
  A a;
  B b;

  Pair(A a, B b) {
    this.a = a;
    this.b = b;
  }
}

var point = new Pair(1, 2);
var player = new Pair("Lamar Jackson", 8);
```