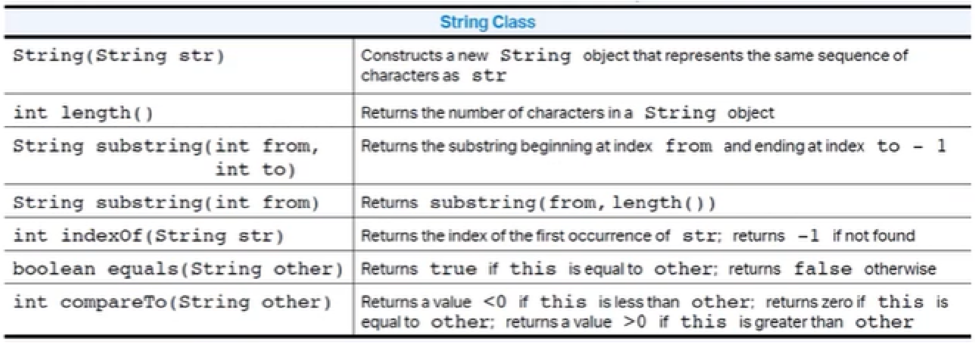
Tips

MC Exam	1. Read all of the answer choices that are provided properly to not miss key lines. 2. Be careful with answers that contain “always” or “never” 3. Answer every question but always make educated guesses 4. Read the question first and then the code to focus on the part you need 5. Read the code very carefully 6. Don’t give up after the first question in sets that are based on the same code → other questions may help you answer the first 7. Make an educated guess and then skip any question that is taking longer than 2 minutes to answer. You can return to the questions again later. a. If you took time to read the question and answers, just first make a guess to avoid forgetting it 8. Types of Questions a. What does this method do? → Follow the algorithm b. What’s wrong with this? → Fix the code c. What can go there? → Complete the missing code d. Class methods → Do you have access to private data? e. `String` manipulation & access using `String` methods f. Language-independent questions g. Array/2D-array/ArrayList traversal, access, + manipulation h. Using/accessing lists of objects i. Inheritance hierarchy relationships & visibility j. Binary/sequential search k. Sorting algorithms → various kinds (how they work, what happens on each step, etc.) l. Loop boundaries, off-by-one errors m. Recursion
FRQ Exam	1. Types of Questions a. Q1-Methods & Control Structures b. Q2-Class Implementation → write entire class

	c. Q3-Array/ArrayList → When removing certain elements from an array of arraylist, traverse backwards through it (starting from end) to avoid skipping over any as the array/arraylist changes d. Q4-2D Array 2. Read problem very carefully before starting to write any code 3. Watch out for “You will receive no credit if” → be prepared to call part (a) solution in part (b) 4. Don’t ignore preconditions & postconditions → tell you about algorithm a. Do not worry about the coding for them, though 5. Never let beginning of problem stop you from getting points at the end 6. Say within AP CSA “Java Subset” 7. Write code to satisfy the defined problem & check your code with all the provided examples 8. Have clear, concise, easy-to-read solution → organize, indent, use meaningful variable names, be neat 9. Reread problem definition after you finish to make sure you answered it properly 10. If methods are provided in given code, you will most likely have to use them 11. Don’t print anything in code unless the problem asks you to 12. Pay attention to return types for methods 13. Make all your instance variables private 14. Read the parameter list carefully and use them in the method 15. Write correct, precise code that performs the obvious tasks 16. Avoid using classes that aren’t specifically given in the exam or aren’t part of the the AP CSA “Java Subset”

Review

Primitive Data Types	❖ Integers ➢ `short` (2 bytes) ➢ `int` (4 bytes) ➢ `long` (8 bytes) ❖ Floating Point Numbers ➢ `float` (4 bytes) ➢ `double` (8 bytes) ❖ `char` → unicode encoding (2 bytes) ❖ `boolean` → true/false (1 byte)
Console Output	❖ `System.out.print()` → prints what’s in () ❖ `System.out.println()` → prints what’s in () + move to next line
Operators	❖ `*` , `-` , `*` , `/` , `%` → arithmetic operators ❖ `++` , `--` → increment/decrement by 1 ❖ `+=` , `-=` , `*=` , `/=` , `%=` → store result to same variable ➢ `x*=7` is `x=x*7` ❖ `==` , `!=` → equal to, not equal to ❖ `<` , `>` , `<=` , `>=` → comparing
Classes	❖ Classes define new data types or create “blueprint” of objects ❖ Use constructor to initialize attributes (default + overload) ❖ Have void + non-void methods
Objects	❖ Object is created as instance of the class ❖ `.equals` → true if objects have same attributes ❖ `==` → true if objects reference each other (aliases)
Strings	❖ `String` objects are immutable ❖ `String name1 = “Jadon Java”` → Create new instance ❖ `“Ja” + “va” = “Java”` → Concatenate w/ `+` or `+=` ❖ `“He said:\n/”hi\\bye/””` prints He said: “Hi\bye” ➢ `/”` → double quotes, `\\` → \\, `\n` → new line ❖ Compare with `.equals` or `.compareTo` ❖ Wrapper Classes = way to use primitive data types as objects ➢ Convert primitive types to reference types ❖ String Class



String Class	
`String(String str)`	Constructs a new `String` object that represents the same sequence of characters as `str`
`int length()`	Returns the number of characters in a `String` object
`String substring(int from, int to)`	Returns the substring beginning at index `from` and ending at index `to - 1`
`String substring(int from)`	Returns `substring(from, length())`
`int indexOf(String str)`	Returns the index of the first occurrence of `str`; returns `-1` if not found
`boolean equals(String other)`	Returns `true` if `this` is equal to `other`; returns `false` otherwise
`int compareTo(String other)`	Returns a value `<0` if `this` is less than `other`; returns zero if `this` is equal to `other`; returns a value `>0` if `this` is greater than `other`

Booleans (Truth Tables)

x	Y	X && Y	X || Y	! (X && Y)	! (X || Y)
T	T	T	T	F	F
T	F	F	T	T	F
F	T	F	T	T	F
F	F	F	F	T	T

Control Flow - if…else

- `else` is optional, `{}` optional but recommended when if-body is only one line
-

```
if (myAge < 2) {
     return “You must sit in a car seat.”;
}
if (myAge >=16)
{
     return “You can learn to drive a car.”;
} else {
     return “You are not old enough for a
license.”;
}
```

- When a condition is met and a `return` statement is run, the rest of the code is not run after

Control Flow - Loops

- 3 main types

- For Loop

```
for (i=1; i<=5; i++) {
     System.out.println(i);
}
```

- While Loop

```
int value = 1;
while (value<=5) {
     System.out.println(value);
     value++;
}
```

- Do-While Loop

```
do {
     System.out.println(value);
     value++;
} while (value<=5);
```

- Enhanced For Loop

```
int[] numbers = {1, 2, 3, 4, 5}
for (int n : numbers ) {
     System.out.println(n);
}
```

  - Use when traversing through an iterable interface (array, arraylist, 2D array)

Defining + Using Classes

-

```
public class Snack {
     // private instance variables
     private String name;
     private int calories;

     // Default Constructor
     public Snack() {
          // Calls the overload constructor
          this(“Snack”, 0);
     }
     // Overload Constructor
     public Snack(String n, int c) {
          name = n;
          calories = c;
     }
     // accessor method
     public String getName() {
          return name;
     }
     // mutator method
```

	`    public int setCalories(int cal) {`
	`        calories = cal;`
	`    }`
	`    public boolean isHealthy() {`
	`        return (calories < 200);`
	`    }`
	`}`
	❖
	`public static void main(String[] args) {`
	    // Create new object with reference to class
	`    Snack chips = new Snack(“Chips”, 150);`
	    // Use object by calling methods
	`    chips.setCalories(259);`
	`    if (!chips.isHealth) {`
	`        System.out.println(“This isn’t healthy”);`

Arrays	❖ Behave like an object
	➢ Elements starts at index 0, and exception thrown if exceed bounds
	➢ Hold reference types or primitive types
	➢ Size can’t be changed
	❖ Ways to initialize:
	➢ `int[] nums = {1, 2, 3, 4};`
	➢ `int[] nums = new int[4];`
	`for (int i=0; i<nums.length; i++) {`
	`    nums[i]=i+1;`
	`}`
	❖ 2D arrays
	➢ Traverse through with nested loops (row-major or column-major
	➢ `2Darray.length` gives # rows
	➢ `2Darray[0].length` gives # columns

ArrayList	❖ Mutable object
	➢ Has object references
	➢ Resizable
	➢ Class w/ methods
	❖ Using ArrayLists:
	// creates new ArrayList of Doubles
	`ArrayList<Double> nums = new ArrayList<Double>();`
	// nums: [3.0]
	`nums.add(3.0);`
	// nums: [3.0, 4.0]

	`nums.add(4.0);`
	// nums: [3.0, 5.0, 4.0]
	`nums.add(1, 5.0);`
	// nums: [1.0, 5.0, 4.0]
	`nums.set(0, 1.0);`
	// nums: [1.0, 5.0]
	`nums.remove(2);`
	// nums: [1.0]
	`nums.remove(5.0);`

Inheritance	❖
	`public class Musician extends Performer {`
	`    private String instrument;`
	
	`    public Musician() {`
	`        super();`
	`        instrument = “Piano”;`
	`    }`
	`    public Musician(String inst) {`
	`        super();`
	`        instrument = inst;`
	`    }`
	`    public Musician(String n, int a, String inst) {`
	`        super(n,a);`
	`        instrument = inst;`
	`    }`
	`}`
	
	❖ Child class inherits all of the data and methods from the superclass
	➢ Methods from superclass can be overwritten by the subclass

`static`	❖ `static` variable ➢ Like a global variable ➢ Have only one instance of the variable, regardless of the number of objects ❖ `static` methods ➢ Can only access `static` items ➢ Invoked by [Class].[methodName]()
`final`	❖ `final` class, methods, and variables

Sorts

- Insertion Sort

```
public void insertionSort() {
     for (int i=1; i<list.length; i++) {
          int key = list[i];
          int j = i-1;
          while (j>=0 && list[j]>key) {
               list[j+1]=list[j];
               j--;
          }
          list[j]=key;
     }
}
```

  - Starts at index 1 as target and works forward, assumes everything to left is sorted, inserts target value where it belongs

- Selection Sort

```
public void selectionSort() {
     for (int i=0; i<list.length-1; i++) {
          int minIdx = i;
          for (int j=i+1; j<list.length;j++) {
               if (list[j] < list[minIdx]) {
                    minIdx = j;
               }
          }
          int x = list[i];
          list[i] = list[min];
          list[min] = x;
```

  - Starts at index 0 as target, finds min value to right of target, swaps min and target

- Merge Sort

```
public void sort(int[] list, int e) {
     if (e < 2) {
          return;
     }
     int m = e/2
     int[] left = new int[m];
     int[] right = new int[e-m];
     for (int i=0; i<left.length; i++) {
          left[i] = list[i];
     }
     for (int i=0; i<right.length; i++) {
          right[i] = list[i+m];
     }
     sort(left, m);
     sort(right, e-m);
     mergeSort(list, right, left, m, e-m);
}
public void mergeSort(int[] list, int[] left,
                  int[] right, int o, int t) {
     int i=0, j=0, k=0;
     while (i<o && j<t) {
          if (left[i] <= right[j]) {
               list[k] = left[i];
               i++;
          } else {
               list[k] = right[j];
          }
          k++;
     }
}
```

- “dive & conquer” → breaks list into pieces by halving, puts pieces back in order → done recursively

Java Quick Reference

Accessible methods from the Java library that may be included in the exam

Class Constructors and Methods	Explanation
String Class	
String(String str)	Constructs a new String object that represents the same sequence of characters as str
int length()	Returns the number of characters in a String object
String substring(int from, int to)	Returns the substring beginning at index from and ending at index to - 1
String substring(int from)	Returns substring(from, length())
int indexOf(String str)	Returns the index of the first occurrence of str; returns -1 if not found
boolean equals(String other)	Returns true if this is equal to other; returns false otherwise
int compareTo(String other)	Returns a value <0 if this is less than other; returns zero if this is equal to other; returns a value >0 if this is greater than other
Integer Class	
Integer(int value)	Constructs a new Integer object that represents the specified int value
Integer.MIN_VALUE	The minimum value represented by an int or Integer
Integer.MAX_VALUE	The maximum value represented by an int or Integer
int intValue()	Returns the value of this Integer as an int
Double Class	
Double(double value)	Constructs a new Double object that represents the specified double value
double doubleValue()	Returns the value of this Double as a double
Math Class	
static int abs(int x)	Returns the absolute value of an int value
static double abs(double x)	Returns the absolute value of a double value
static double pow(double base, double exponent)	Returns the value of the first parameter raised to the power of the second parameter
static double sqrt(double x)	Returns the positive square root of a double value
static double random()	Returns a double value greater than or equal to 0.0 and less than 1.0
ArrayList Class	
int size()	Returns the number of elements in the list
boolean add(E obj)	Appends obj to end of list; returns true
void add(int index, E obj)	Inserts obj at position index (0 <= index <= size), moving elements at position index and higher to the right (adds 1 to their indices) and adds 1 to size
E get(int index)	Returns the element at position index in the list
E set(int index, E obj)	Replaces the element at position index with obj; returns the element formerly at position index
E remove(int index)	Removes element from position index, moving elements at position index + 1 and higher to the left (subtracts 1 from their indices) and subtracts 1 from size; returns the element formerly at position index
Object Class	
boolean equals(Object other)	
String toString()	