

AP CS Java Syntax Summary: (version 5)

CLASSES & METHODS

Class & main:

```
public class name {
    public static void main(String[] args) {
        statement;
        ...
        statement;
    }
}
```

Method creation:

```
public static <type OR void> name(type <parameter name>, ..., type <parameter name>) {
    statement;
    ...
    statement;
    return expression;
}

name(); // Calling a Static Method with no parameters
name (value, value, ..., value); // Calling a Method with Parameter(s)
variable = name (value1, value2, ..., valueN); // calls the Method "name" with
// the parameters values: value1 – valueN and returns a value to variable
type variable = name (value1, value2, ..., valueN); // calls the Method "name"
// with the parameter values: value1 – valueN and assigns the returned value
// to "variable", which is created as the appropriate type
```

Example Method with no parameters or return value:

```
public static void printHeader() {
    System.out.println("Welcome to the wonderful Program.");
    System.out.println("Hope you enjoy our hard work");
}
```

Example Method call with no parameters or returned value (statements from another method):

```
printHeader(); // call of printHeader, simply prints out the two lines
```

Example Method with parameters and return value:

```
public static int addThree(int oneValue, int twoValue, int threeValue) {
    int sum = oneValue + twoValue;
    sum = sum + threeValue;
    return sum;
}
```

Example Method call with parameters and returned value (statements from another method):

```
mySum = addThree(100, 20, 3); // assigns integer 123 to the existing integer
// variable, mySum
int addedValues = addThree(100, 20, 3); // assigns integer 123 to newly created
// integer variable, addedValues
```

Comments:

```
// comment text, on one line
/* comment text; may span multiple lines */
```

Println – Print line:

```
System.out.println("<string>"); // Prints string then a new line
System.out.print("<string>"); // Prints just the string
```

Escape Character within a string: the backslash \

AP CS Java Syntax Summary

VARIABLES:

Primitive Variable Types:

```
int name = <value>; // creates an Integer and assigns value to it
double name = <value>; // creates an Double – real numbers, and assigns value to it
char name = '<single character>'; // creates a single character like 'a', '1', ' ', etc.
boolean name = true; // creates a Boolean of true or false, and assigns true to it
```

Other Variable Types:

```
String name = "<series of characters>"; // creates a String and assigns the string to it
```

Class Constant:

```
public static final type NAME = value; // Class constant names in all upper case letters
// usually placed just under the Class definition above main method
```

Updating Variables:

Assigning a value:

```
variable = <value or expression>; // variable's value is replaced with the new value
// or the value of an expression
```

Example Assignments:

```
x = 12; // assigns the value 12 to x
y = x + 5 * 32; // the value for the expression (x + 5 * 32) is assigned to y
z = z + 1; // z is incremented by 1
date = getDate(console); // Returned value the method getDate called with the console
// parameter is assigned to date
```

Type Casting Variables:

Type cast: A conversion from one type to another. Type casting has highest precedence and only casts the item immediately after it. It can be used to 1) promote an `int` into a `double` to get exact division from `/` & 2) To truncate a `double` from a real number to an integer

•Syntax:

```
(type) expression
```

Examples:

```
double result = (double) 19 / 5; // 3.8
int result2 = (int) result; // 3
int x = (int) Math.pow(10, 3); // 1000
```

To convert an `int` into the equivalent `char`, type-cast it.

```
(char) ('a' + 2) is 'c'
```

Shorthand

```
variable++;
variable--;
variable += value;
variable -= value;
variable *= value;
variable /= value;
variable %= value;
```

Equivalent longer version

```
variable = variable + 1;
variable = variable - 1;
variable = variable + value;
variable = variable - value;
variable = variable * value;
variable = variable / value;
variable = variable % value;
```

Logical Operators: (used in tests to determine a Boolean true or false value)

Operator	Meaning	Example	Value
<code>==</code>	equals	<code>1 + 1 == 2</code>	true
<code>!=</code>	does not equal	<code>3.2 != 2.5</code>	true
<code><</code>	less than	<code>10 < 5</code>	false
<code>></code>	greater than	<code>10 > 5</code>	true
<code><=</code>	less than or equal to	<code>126 <= 100</code>	false
<code>>=</code>	greater than or equal to	<code>5.0 >= 5.0</code>	true
<code>&&</code>	AND	<code>(2 == 3) && (-1 < 5)</code>	false
<code> </code>	OR	<code>(2 == 3) (-1 < 5)</code>	true
<code>!</code>	NOT	<code>!(2 == 3)</code>	true

FLOW CONTROL**for Loop:**

```
for (initialization; test; update) {
    statement(s);
}
```

Example:

```
for (int i = 1; i <= 6; i++) {
    System.out.println("I am so smart");
}
```

Cumulative Algorithm (or Sum): typically requires four parts, don't forget one: Initialization (outside the loop), Loop, Action, & Increment. Here is a code snippet of a Cumulative Sum:

```
int sum = 0; // Initialization - outside the loop
for (int i = 1; i <= 1000; i++) { // Loop also contains the Increment, i++
    sum = sum + i; // Action
}
System.out.println("The sum is " + sum);
```

if Statement:

```
if (test) {
    statement(s);
}
```

if Example:

```
if (GPA >= 3.5) {
    System.out.println("You are so smart");
}
```

if / else Statement:

```
if (test) {
    statement(s);
} else {
    statement(s);
}
```

if / else Example: (exactly one path will be executed)

```
if (GPA >= 3.5) {
    System.out.println("You are so smart");
} else {
    System.out.println("Study more please");
}
```

if / else / if Statement: (one or no path may be executed)

```
if (test) {
    statement(s);
} else if {
    statement(s);
}
```

Nested if / else / if Example:

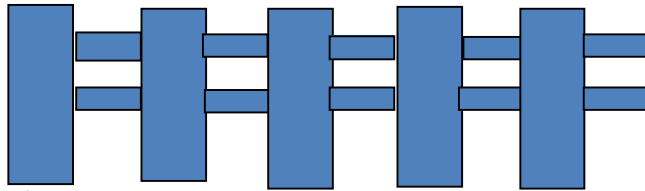
```
if (place = 1) {
    System.out.println("Gold Medal!");
} else if (place = 2) {
    System.out.println("Silver Medal!");
} else if (place = 3) {
    System.out.println("Bronze Medal!");
}
```

AP CS Java Syntax Summary

Fencepost Loop:

Adds a statement outside the loop to place the initial "post." Also called a fencepost loop or a "loop-and-a-half" solution.

```
place a post.  
for (length of fence - 1) {  
    place some wire.  
    place a post.  
}
```



```
public static void printNumbers(int max) {  
    System.out.print(1);  
    for (int i = 2; i <= max; i++) {  
        System.out.print(", " + i);  
    }  
    System.out.println(); // to end the line  
}
```

definite loop: Executes a known number of times. (i.e. most counting for loops)

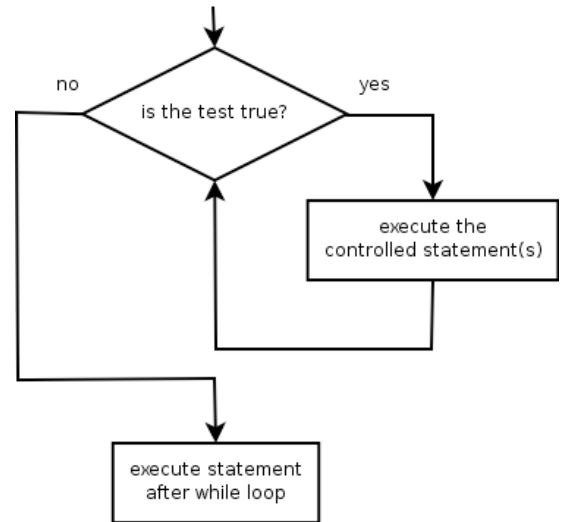
indefinite loop: One where the number of times its body repeats is not known in advance.

while Loop:

```
while (test) {  
    statement(s);  
}
```

while loop Example: prints powers of 2 less than 200

```
int num = 1; // initialization  
while (num <= 200) { // test  
    System.out.print(num + " ");  
    num = num * 2; // update  
}
```



Sentinel value: A value that signals the end of user input.

Sentinel loop: Repeats until a sentinel value is seen.

KEY OBJECTS & THEIR METHODS

Objects:

Constructing (creating) an object:

```
Type objectName = new Type(parameters);
```

Calling an object's method:

```
objectName.methodName(parameters);
```

Drawing Panel & Graphics Objects:

Required Library for these:

```
import java.awt.*; // this import is required for Graphics above the Class
```

Object Creations:

```
DrawingPanel panelName = new DrawingPanel(width, height); // creates Drawing Panel  
Graphics graphicName = panelName.getGraphics(); // creates the graphics object  
Color colorName = new Color(red, green, blue); // creates a color with RGB values  
Polygon polygonName = new Polygon(); // creates a polygon
```

Key Drawing Panel Methods:

```
panelName.setBackground(colorName); // sets the background color of the panel  
panelName.clear(); // Erases any shapes that are drawn on the drawing panel.  
panelName.setWidth(width); // Changes the drawing panel's width  
panelName.setHeight(height); // Changes the drawing panel's height  
panelName.setSize(width, height); // Changes the drawing panel's width & height  
panelName.save(filename); //Saves the image on the panel to the given filename  
panelName.sleep(ms); //Pauses the drawing for the given number of milliseconds
```

Key Graphics Methods:

```
graphicName.drawLine(x1, y1, x2, y2); // draws a line from points 1 to 2  
graphicName.drawOval(x, y, width, height); // draws an Oval's outline  
graphicName.drawRect(x, y, width, height); // draws an Rectangle's outline  
graphicName.drawString(text, x, y); // draws out the text string  
graphicName.fillOval(x, y, width, height); // draws a filled Oval  
graphicName.fillRect(x, y, width, height); // draws a filled Rectangle  
graphicName.setColor(Color); // Sets the color for drawing  
graphicName.fillPolygon(polygonName); // fills the Polygon with the current color
```

Key Color Methods/values:

```
Color.CONSTANT_NAME // values for preset colors, where CONSTANT_NAME is:  
BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE,  
PINK, RED, WHITE, YELLOW
```

Key Polygon Method:

```
polygonName.addPoint(x, y); // adds a point to the Polygon at x,y coordintate
```

Integer Object Values:

```
Integer.MAX_VALUE - the Maximum value that can be stored in an Integer: 2147483647
```

```
Integer.MIN_VALUE - the Minimum value that can be stored in an Integer: -2147483648
```

AP CS Java Syntax Summary

Key Math Methods: (of the Math Class)

```
Math.abs(value) // absolute value
Math.ceil(value) // rounds up
Math.floor(value) // rounds down
Math.log10(value) // logarithm, base 10
Math.max(value1, value2) // larger of two values
Math.min(value1, value2) // smaller of two values
Math.pow(base, exp) // base to the exp power
Math.random() // random double between 0 and 1
Math.round(value) // nearest whole number
Math.sqrt(value) // square root
Math.sin(value) // sine of an angle in radians
Math.cos(value) // cosine of an angle in radians
Math.tan(value) // tangent of an angle in radians
Math.toDegrees(value) // convert radians to degrees
Math.toRadians(value) // convert degrees to radians
```

Scanner & File Objects:

Required Library for these: (placed above the Class creation

```
import java.util.*; // required for Scanner Object
import java.io.*; // required for File Object
```

Scanner & File Object Creations:

```
Scanner name = new Scanner(source);
File name = new File("file name");
```

Examples:

```
Scanner console = new Scanner(System.in); // creates Scanner named "console" that
// reads from the input (keyboard)
File fileHere = new File("mydata.txt"); // creates File named "fileHere" that
// accesses the file mydata.txt
Scanner input = new Scanner(fileHere); // creates Scanner named "input" that
// reads from the file "fileHere", which is accessing "mydata.txt"
```

Scanner Methods: (s is the Scanner object)

```
s.nextInt() reads an int from the user and returns it
s.nextDouble() reads a double from the user
s.next() reads a one-word String from the user
s.nextLine() reads a one-line String from the user
```

Scanner Test Methods

```
s.hasNext() // returns true if there is a next token
s.hasNextInt() // returns true if there is a next token & it can be read as an int
s.hasNextDouble() // returns true if there is a next token and it can
// be read as a double
s.hasNextLine() // returns true if there are any more lines of input to read
// (always true for console input)
```

File Methods: (f is the File object)

```
f.delete() removes file from disk
f.getName() returns file's name
f.length() returns number of bytes in file
f.renameTo(file) changes name of file
```

File Test Methods

```
f.canRead() returns whether file is able to be read
f.exists() whether this file exists on disk
```

AP CS Java Syntax Summary

Key String Methods: (operates on String type, s is the object here...)

```
s.indexOf(str) // index where the start of the given string appears
                //in this string (-1 if not found)
s.length()     // number of characters in this string
s.substring(index1, index2) // the characters in this string from index1
                            // (inclusive) to index2 (exclusive);
s.substring(index1) // if index2 is omitted, grabs till end of string
s.toLowerCase()  // a new string with all lowercase letters
s.toUpperCase()  // a new string with all uppercase letters
s.charAt(int)    // accepts an int index parameter and returns the char at
                // that index
```

String Test Methods:

```
s.equals(str) // whether two strings contain the same characters
s.equalsIgnoreCase(str) // whether two strings contain the same
                        // characters, ignoring upper vs. lower case
s.startsWith(str) // whether one contains other's characters at start
s.endsWith(str)   // whether one contains other's characters at end
s.contains(str)   // whether the given string is found within this on
```

Arrays (BJP Chapter 7):

Required Library for these: (placed above the Class creation

```
import java.util.*; // required for Arrays
```

Array Creation and use:

```
type[] name = new type[length];
type[] name = {value0, value1, ..., valueN}; // initializing an array explicitly
name [index] = value; // Assigning a value to an array's indexed location
public static type methodName(type[] name) { // using an array as a parameter
public static type[] methodName(parameters) { // returning an array
methodName(arrayName); // calling a method with an array as its parameter
type[] name = methodName(parameters); // assigning an array as a returned result
Examples:
double[] numbers = double int[8]; // creates an array of doubles with 8 elements
int[] values = {11, 42, -5, 27, 0, 89}; // creates an array of integers with the
                                        // specified values of length 6
public static void actOnIt(double[] input) {} // a double array as a parameter
public static boolean[] figureItOut(int x, int y) {} // returning a boolean array
actOnIt(numbers); // calling a method with an array as its parameter
boolean[] values = figureItOut(x, y); // assigning an array as a returned result
```

Array Length Field:

```
name.length // returns the integer length of the array, note no parenthesis
```

Examples:

```
int arrayLength = numbers.length; // Assigns the length of the array to a variable
for (int i = 0; i < numbers.length; i++) { // for loop going through an array
} // one by one element at a time
```

Array Methods: (for the Arrays class)

```
Arrays.equals(array1, array2) // returns true if the two arrays contain same
                              // elements in the same order
Arrays.toString(array) // returns a string representing the array, such as
                       // "[10, 30, -25, 17]"
Arrays.copyOf(array, length) // returns a new copy of an array
Arrays.fill(array, value) // sets every element to the given value
Arrays.sort(array) // arranges the elements into sorted order
Arrays.binarySearch(array, value) // returns the index of the given value in a
                                  // sorted array (or < 0 if not found)
```

AP CS Java Syntax Summary

2D Arrays (BJP Chapter 7.5):

2D Array Creation and use:

```
type[][] name = new type[height][width];
type[][] name = { {value00, value01, ..., value0w},
                  {value10, value11, ..., value1w},
                  ...
                  {valueh0, valueh1, ..., valuehw} }
// initializing an array explicitly
name [indexh][indexw] = value; // Assigning a value to an array's indexed location
public static type methodName(type[][] name) { // using an array as a parameter
public static type[][] methodName(parameters) { // returning an array
methodName(arrayName); // calling a method with an array as its parameter
type[][] name = methodName(parameters); // assigning an array as a returned result
2D Array Examples:
char[][] board = new char[3][3]; // creates an array of chars 3 by 3, 9 elements
char[][] board = {{'X', 'O', 'X'},
                  {'O', 'X', 'O'},
                  {'X', 'O', 'X'}};
public static void PlayGame(char[][] input) {} // a char 2D array as a parameter
public static boolean[][] emptyLoc(int x, int y) {} // returning a 2D boolean array
printBoard(board); // calling a method with a 2D array as its parameter
boolean[] values = emptyLoc(x, y); // assigning a 2D array as a returned result
```

Jagged 2D Array Examples:

```
int[][] jagged = new int[3][];
jagged [0] = new int[2];
jagged [1] = new int[5];
jagged [2] = new int[4];
```

2D Array Length Field:

nums.length – returns array's height or number of rows (no [])
nums[0].length – returns array's width or columns - at that row [], which is the same for regular arrays.

Examples: for a 2D Array:

```
int[][] nums = new int[5][4];
```

Length results would be:

```
nums.length is 5
nums[0].length is 4
```

All rows same width here. Note: these would differ for a Jagged Array

```
nums[1].length is 4
nums[2].length is 4...
```

Acting on 2D arrays usually involves nested for loops using the length dimensions – keep track of row & columns carefully and name variables wisely.

```
public static void fillArray (int nums[][]){
    int count = 1;
    for (int row = 0; row < nums.length ; row++){
        for (int col = 0; col < nums[0].length ; col++){
            <statements...>
        }
    }
}
```


AP CS Java Syntax Summary

ArrayList: (BJP Chapter 10)

Required Library for these: (placed above the Class creation)

```
import java.util.*; // required for Arrays
```

Array Creation and use:

```
ArrayList<Type> name = new ArrayList<Type>();
name.add("Your Name"); // Assigning a value to the end of an ArrayList name
public static type methodName(ArrayList<Type> name){ // ArrayList as a parameter
public static ArrayList<Type> methodName(parameters) { // returning an ArrayList
methodName(arrayListName); // calling a method with an ArrayList as its parameter
ArrayList<String> names = methodName(parameters); // ArrayList as returned result
Examples:
```

```
ArrayList<String> names = new ArrayList<String>(); // creates ArrayList of Strings
public static void actOnIt(ArrayList<String> input) {} // ArrayList as a parameter
// returning a boolean ArrayList, using "Integer" as the wrapper for int
public static ArrayList<Integer> figureItOut(int x, int y) {}
actOnIt(names); // calling a method with an array as its parameter
ArrayList<Integer> values = figureItOut(x, y); // an ArrayList as a returned result
```

ArrayList Size Method:

```
name.size() // returns the integer size of an ArrayList, includes parenthesis
```

Examples:

```
ArrayList<Integer> list = new ArrayList<Integer>(); // Creates ArrayList of int's
for (int i = 1; i <= 10; i++) {
    list.add(10 * i); // [10, 20, 30, 40, ..., 100]
}
for (int i = 0; i < list.size(); i++) { // for loop going through an ArrayList
    ... // NOTE: that the size() may change based on what goes on in the loop.
} // one by one element at a time
```

Key ArrayList Methods: (where a is an ArrayList Object, see more in the Java API)

```
a.add(value) // appends value at end of list
a.add(index, value) // inserts given value just before the given index,
a.shifting // subsequent values to the right
a.clear() // removes all elements of the list
a.indexOf(value) // returns first index where given value is found in list
// (-1 if not found)
a.get(index) // returns the value at given index
a.remove(index) // removes & returns value at given index,
// shifting subsequent values to the left
a.set(index, value) // replaces value at given index with given value
a.size() // returns the number of elements in list
a.toString() // returns a string representation of the list such as "[3, -7, 15]"
a.addAll(list) OR a.addAll(index, list) // adds all elements from the given list to
// this list (at the end of the list, or inserts them at the given index)
a.contains(value) // returns true if given value is found somewhere in this list
a.containsAll(list) // returns true if this list contains every element
// from given list
a.equals(list) // returns true if given other list contains the same elements
a.lastIndexOf(value) // returns last index value of value found in list
// (-1 if not found)
a.removeAll(list) // removes any elements found in the given list from this list
a.retainAll(list) // removes any elements not found in given list from this list
a.subList(from, to) // returns the sub-portion of the list between indexes
// from (inclusive) and to (exclusive)
a.toArray() // returns the elements in this list as an array
```