Write a static method rearrange that takes an array of integers as an argument and that rearranges the values so that all of the multiples of 3 come first followed by numbers that are one greater than a multiple of 3 followed by numbers that are two greater than a multiple of 3. For example, if a variable called "list" stores this sequence of values:

```
(23, 12, 8, 0, 4, 80, 9, 7, 30, 99, 50, 42, 13, 47, 2, 16, 87, 75)
```

Then the following call:

```
rearrange(list);
```

Should rearrange the values to look something like this:

```
(12, 0, 9, 30, 99, 42, 87, 75, 4, 13, 16, 7, 23, 47, 2, 50, 8, 80)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~  ~~~~~~~~~~~~  ~~~~~~~~~~~~~~~~~~~~
multiples of 3                1 more than a     2 more than a
                              multiple of 3     multiple of 3
```

This is only one possible arrangement. Any arrangement that puts the multiples of 3 first followed by the values that are one more than a multiple of 3 followed by the values that are two more than a multiple of 3 is acceptable. You are not allowed to use an auxiliary data structure such as a temporary array or ArrayList to solve this problem. You may assume that all values in the array are greater than or equal to 0.

Write your solution to rearrange below.

Here's the solution from class:

```
public static void rearrange(int[] list) {
    int swap = 0;
    for (int i = swap; i < list.length; i++) {
        int num = list[i];
        if (num % 3 == 0) {
            list[i] = list[swap];
            list[swap] = num;
            swap++;
        }
    }
    for (int i = swap; i < list.length; i++) {
        int num = list[i];
        if (num % 3 == 1) {
            list[i] = list[swap];
            list[swap] = num;
            swap++;
        }
    }
}
```

Here’s the more efficient solution that avoids redundancy. Either one of the two would get full credit on an exam:

```
public static void rearrange(int[] list) {
        int swap = 0;
        for (int i = 0; i < 2; i++) {
                for (int i = swap; i < list.length; i++) {
                        int num = list[i];
                        if (num % 3 == i) {
                                list[i] = list[swap];
                                list[swap] = num;
                                swap++;
                        }
                }
        }
}
```

Write a method called `interleave` that accepts two `ArrayLists` of integers *a1* and *a2* as parameters and inserts the elements of *a2*into *a1* at alternating indexes. If the lists are of unequal length, the remaining elements of the longer list are left at the end of *a1*. For example, if *a1* stores `[10, 20, 30]` and *a2* stores `[4, 5, 6, 7, 8]`, the call of `interleave(a1, a2);` should change *a1* to store`[10, 4, 20, 5, 30, 6, 7, 8]`. If *a1* had stored `[10, 20, 30, 40, 50]` and *a2* had stored `[6, 7, 8]`, the call of `interleave(a1, a2);` would change *a1* to store `[10, 6, 20, 7, 30, 8, 40, 50]`.

```
public static void interleave(ArrayList<Integer> a1, ArrayList<Integer> a2) {

        int index = 1;

        for (int i = 0; i < a2.size(); i++) {

                // This comes before to address the "empty a1" case

                if(index > a1.size()) {

                        index = a1.size();

                }

                a1.add(index, a2.get(i));

                index += 2;

        }

}
```

Write a static method named playlist that accepts as its parameter a Scanner for an input file representing a sequence of songs. Your method will reformat the song names and lengths found in the file to be more readable and output them to the console. Additionally, your method will return whether or not all the songs in the file would fit on a standard CD. A standard CD can hold up to, and including, 80 minutes of audio data.

Each line of the playlist file consists of a song name consisting of one or more words, a semi-colon (ie, ;), and the song's length as two integers, minutes and seconds. Your method should read each line and output the song information in a more readable format to the console. Separate each token of the song name with a single space, capitalize the first letter of each word, and lowercase the rest. If the total time needed to play all the songs is 80 minutes of less, return true. Otherwise, return false. The table below shows two example files. The first file has five songs which fit on a CD because the total playtime is under 80 minutes. The second file has six songs, but all six songs cannot fit on a CD because it takes more than 80 minutes to play them. Regardless of the total playtime, the reformatted playlist is always printed to the console.

Input File	Console Output	Value Returned

`Alejandro ;      8 43`
`  hOme       ; 5 3`
`WHEN yoU Love SomEBODY ;  2 5`
`   rudE to rile ; 3        30`
`CALIFORNIa   gurls ; 3      56`
	`Alejandro 8:43`
	`Home 5:3`
	`When You Love Somebody 2:5`
	`Rude To Rile 3:30`
	`California Gurls 3:56`
		`true`

`tubULaR    bElls ; 25 01`
`   reVolution 9 ;   8 22`
`lasT   caLL ; 12 41`
`ShARKS    and SAILORS ; 8 13`
` osAKA part    I    ; 38 58`
`SHINE ON YOU    crazy diamond ;`
`26 01`
	`Tubular Bells 25:1`
	`Revolution 9 8:22`
	`Last Call 12:41`
	`Sharks And Sailors 8:13`
	`Osaka Part I 38:58`
	`Shine On You Crazy Diamond`
	`26:1`
		`false`

Assume valid input. Assume the file contains data for at least one song, every line contains one song's data, and every song is in the format described above. Each song's minutes and seconds are in the range of 0-59.

```
public static boolean playlist(Scanner input) {
	int totalSeconds = 0;
	while (input.hasNextLine()) {
		String line = input.nextLine();
		Scanner lineScan = new Scanner(line);
		String nextWord = lineScan.next();
		String song = "";
		while(lineScan.hasNext() && !nextWord.equals(";")) {
			String word = lineScan.next();
			song += nextWord.toUpperCase().substring(0, 1) +
			nextWord.toLowerCase().substring(1); + " "
			nextWord = word;
		}
		int min = lineScan.nextInt();
		int sec = lineScan.nextInt();
		totalSeconds += min * 60 + sec;
		System.out.println(song + min + ":" + sec);
	}
	return totalSeconds <= 4800;
}
```