Introduction:

Arrays in Java are a very useful data structure that allows the user to save multiple objects into a single instance. But for how useful they are, there are some drawbacks to them. Mainly, how they can't change in size or easily add another element to an already full array. The fix to this is a *dynamic storage* solution. You can see the advantages of these dynamic data structures, such as vectors, in applications that you use in your day to day life, such as music players: they can use vectors to store playlists and libraries that are ever-changing in size and order. In this lab, you'll create your own (slightly simpler) version of a vector that will be used in a simple mp3 player.

Instructions:

Implement the below methods so that the tester class can run without any errors, then run the MP3PlayerDriver class to see a simple mp3 player that uses your implementation of a vector to store music and playlists in a dynamic way.

UML:

(underlined means "static"; ALL CAPS means "final")

MyVector
+ DEFAULT_CAP : int + DEFAULT_CAP_INC : int - elementData : MP3File [ ] - capacityIncrement : int - elementCount : int
+ MyVector ( ) + MyVector (int) + MyVector (int, int) + add (MP3File) : boolean + add (int, MP3File) : void + capacity ( ) : int + clear ( ) : void + contains (MP3File) : boolean + ensureCapacity (int) : void + equals (MyVector) : boolean + get (int) : MP3File + isEmpty ( ) : boolean + remove (int) : MP3File + size ( ) : int + toString ( ) : String + trimToSize ( ) : void - increaseCapacity ( ) : void

Class Elements:

DEFAULT_CAP
This is completed for you. The default value to use as a the capacity of the inner array

DEFAULT_CAP_INC
This is completed for you. The default value to use as a the capacity increment

elementData
The inner array that will hold the MP3Files

capacityIncrement
The capacity increment value of this MyVector. This is the value by which the array will grow when more space is needed

elementCount
The number of elements that are contained in elementData. This value always starts at 0 and will never be larger than elementData.length

MyVector ( )
Create a new MyVector object with elementData having size DEFAULT_CAP and capacityIncrement having the value of DEFAULT_CAP_INC

MyVector (int initialCapacity)
Create a new MyVector object with elementData having size initialCapacity and capacityIncrement having the value of DEFAULT_CAP_INC. If initialCapacity is less than zero, then elementData should have size DEFAULT_CAP

MyVector (int initialCapacity, int capacityIncrement)
Create a new MyVector object with elementData having size initialCapacity and capacityIncrement having the value of capacityIncrement. If either initialCapacity or capacityIncrement are less than zero, then the instance variable should be assigned it's respective default value. This should use the other constructors

add (MP3File file)

Add ***file*** to the end of elementData. If elementData is full, then call **increaseCapacity().** This add method can call the two parameter **add** method

add (int index, MP3File file)

Add ***file*** to elementData at position ***index***. If elementData is full, then call **increaseCapacity()**

capacity ( )

Return the capacity of the inner elementData array. That is, the length of elementData

clear ( )

Remove all files from this MyVector. The capacity of the inner array will be the same as it was before this method was called. However, the elementCount should be zero and all elements should be null after this method is called

contains (MP3File file)

Returns true if this MyVector contains the MP3File ***file***, or false if it does not. This method should only check elements that are in the MyVector (only check elements that are at indices between 0 and elementCount). Note that it is possible for this method to be called with a null parameter, and it is also possible for the inner array, elementData, to contain null elements at valid indices. (Watch for possible instances where a NullPointerException could occur)

ensureCapacity (int minCapacity)

Increase the capacity of this MyVector until it is large enough to hold ***minCapacity*** number of elements. This method should call ***increaseCapacity()***, and call it repeatedly, if necessary

equals (MyVector anotherVector)

Returns true if and only if this MyVector and ***anotherVector*** have the same size, capacity, capacityIncrement, as well as the same elements, in the same order. Otherwise returns false

get (int index)

Returns the MP3File at ***index*** in the inner array, elementData. If ***index*** is out of bounds (less than zero, or greater than elementCount) then return null

isEmpty ( )

Returns true if this MyVector has zero elements in it, false otherwise

remove (int index)

Removes the element from the given ***index*** and shifts all following elements to the left one. The last element should be set to null (capacity should not change). Element count should decrease. If the given ***index*** is out of bounds, then return null and don't modify the inner array, elementData

size ( ) Returns the size of this MyVector (the number of elements in the inner array, elementData; elementCount)
toString ( ) Returns a string representation of this MyVector
trimToSize ( ) Trims this MyVector so that it's capacity and size are equal my replacing the inner array, elementData, with a small one
increaseCapacity ( ) Grows the inner array based on these rules: If capacityIncrement is equal to DEFAULT_CAP_INC (i.e. the user did not supply a capacity increment upon creation), then the inner array should double in size. Otherwise, the inner array should grow in size by capacityIncrement. The new array should contain the same elements, in the same order, as the old array. No instance variables should change besides elementData (this method should not modify elementCount or capacityIncrement.) **NOTE:** It is not up to this method to determine how many times the array should grow. One call to this method should grow the array once

Upon completion of the MyVector class, run the MyVectorTester to check your methods before executing the MP3 program.