Project 1 for CSE 332 (Spring 2011)

Curly-Queue Jukeboxes

Phase A due Wednesday, April 6 at 11:00 PM via Catalyst CollectIt.
Phase B due Tuesday, April 12 at 11:00 PM via Catalyst CollectIt.

CSE 332: Data Abstractions
The University of Washington, Seattle, Spring 2011

Overview: In this project you will create an MP3 player application that handles queues of MP3 files to be played in an explicit manner. The queues should have a circular-array ("curly" queue) implementation. The queue or list of queues, depending on whether you are doing Phase A or Phase B, will subclass a given abstract class called VisibleDataStructure.

Purposes: (a) refresh your knowledge of Java, (b) become familiar with a framework for building visual representations of data structures and algorithms, (c) learn or gain fluency with a state-of-the-art software development tool (Eclipse), (d) understand the circular-array implementation of queues in depth, (e) learn more Java, (f) have fun using data structures to control the playing of audio media.

Resources Provided: You'll start with the following resources:
1. A Java-callable library called JLayer for processing MP3 files.
2. A "Visual Data Structure Applet" for a default data structure, which is a 2D array.
3. A Java file "VisualStackApplet.java" that shows how to adapt the Visual Data Structure Applet for a custom data structure, in this case a stack.
Your program will inherit the following behavior: scrollable display area, textual command processor, buttons to control the overall execution of commands.

Required Functionality for Phase A: You will implement the following new required functionality:
a. Queue operations – CREATE, ENQUEUE, DEQUEUE and a variation PLAY-DEQUEUE that dequeues a file name and plays the file.
b. Path operation – SET-PATH, which takes an absolute path as an argument and controls where the program looks for MP3 files to play.
c. Display of the queue and its implementation as a "circular" array.

Phase B Functionality: You'll implement the following functionality for an advanced version of your player:
d. Support for multiple queues, in which each command XCREATE, XENQUEUE, XDEQUEUE and XPLAY-DEQUEUE takes an extra argument (a word or phrase with no

spaces in it) that controls which queue is affected.
e. Command XPLAY-RR that starts dequeuing and playing from all the queues in a round-robin (turn-taking) fashion.
f. Commands XRESIZE and XDESTROY.

Additional details are given in the section “Recommended Development Sequence”.

About JLayer: You create an object to play an MP3 file by calling a constructor for the class AdvancedPlayer that is defined in the JLayer library jl1.0.1.jar. Once this object has been created, you can call its .play() method to make it start playing and its .close() method to force it to stop. The following program shows how to use the AdvancedPlayer class. In Eclipse, make a project called MySimpleMP3Player and use the source code below in a default source package file named MySimpleMP3Player.java. Put a copy of the jl1.0.1.jar in the project folder and import it into the project by going to the Project menu, Properties option, Java Build Path, and Libraries tab, and then clicking on "Add Jars...". Then select jl1.0.1.jar from the dialog box. Then put a sample MP3 file in the project folder. In MySimpleMP3Player.java, change the name of the file from "Bach.mp3" to match your sample MP3 file is named.

```
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.IOException;

import javazoom.jl.decoder.JavaLayerException;
import javazoom.jl.player.advanced.AdvancedPlayer;

public class MySimpleMP3Player {

	public static void main(String[] args) {
		BufferedInputStream bis=null;
		AdvancedPlayer player=null;
		try {
			bis = new BufferedInputStream(new FileInputStream("Bach.mp3"));
		}
		catch (IOException e) {
			System.out.println("Problem accessing file.");
		}
		try {
			player = new AdvancedPlayer(bis);
			player.play();
		}
		catch (JavaLayerException e) {
			System.out.println("Problem playing file.");
		}
		player.close();
	}
}
```

Recommended Development Sequence: The following steps are recommended.
1. Download and install the Java 6 JDK.
2. Download and install Eclipse (version 3.6 – “Helios”).

3. Try developing a Hello World application in Eclipse.
4. Download the VisualDataStructureApplet.java code and set it up as the source for a new project VisualDataStructureApplet. Compile and run it.
5. Copy the previous project changing the name to VisualStackApplet. Download the source code for VisualStackApplet.java and add it to this new project. Compile it and run it.
6. Carefully read the code in VisualStackApplet to see how it extends the VisualDataStructureApplet.java class. Try to understand how everything works in these two files, including the graphics calls.
7. Create a new project MySimpleMP3Player as described above, download the source code for it shown above from the Assignment 1 web page, download the Jlayer jar file, and put them into the new project in the appropriate places (Java code in the default source package, and the jar file just within the folder of the project). Import the jar file into the project. Compile and run.
8. Copy the VisualStackApplet project, with the new name of the form JohnsBasicQueueApplet. Refactor (with a rename option) to change the name to work with "Queue" instead of "Stack". Then edit the code for what will have been the inner class MyStack so that it is MyQueue and properly supports ENQUEUE and DEQUEUE instead of PUSH and POP operations. Adjust the renderDS method code if necessary to make the applet display the new data structure.
9. Copy your basic queue applet project with a new name like JohnsBasicQueueJukebox. In this project, implement the new method PLAY-DEQUEUE which should dequeue an element (which is assumed to be the name of an MP3 file) and plays it. The program should assume that there is a folder named MP3Files in the top level of the project's folder. You can put whatever MP3 files you like in that folder to test your program.
10. Copy your basic queue jukebox project with a new name like JohnsCurlyQueueJukebox. In this version make the following changes: the queue should be implemented as a circular array and displayed as such. Implement a new command CREATE that takes a single argument – an integer giving the capacity of the array. Look at the given screen shot of a sample solution. Then implement the remaining commands for Phase A.
11. Now implement the Phase B functionality. This might be the most satisfying part of the project, since the added functionality starts to go beyond what you get in your standard commercial MP3 player. Copy your Curly Queue jukebox project with a name like JohnsAdvancedCurlyQueueJukebox. In this version add multi-queue functionality and a special command XPLAY-RR that starts dequeuing and playing MP3 files from all the queues in a round-robin fashion. Your visible data structure for this version is no longer a single queue but a list of queues. The commands you will implement for this version are the following:

XCREATE *capacity queue-name*
This command creates a new queue and adds it to the list of queues. The *capacity* of the circular array implementing the queue is given by an integer. The queue name is a string containing no whitespace characters. For example, we might have:
XCREATE 15 TRADITIONAL
XCREATE 10 TECHNO

XRESIZE *capacity queue-name*
This command first checks to see that a queue already exists with name *queue-name* and if

not, shows an error message and does nothing else. Otherwise, it creates a new queue, copies all items from the existing queue to the new queue (unless the capacity is smaller than the number of items needed, in which case it copies the first k (where k = capacity of the new queue) starting at the head of the old queue. Then it deletes the old queue and gives the old name to the new queue. If any song items could not be copied due to a smaller capacity, those entries are dropped.

XENQUEUE *mp3file queue-name*
This puts the given file name onto the specified queue. For example,
XENQUEUE Auld-Lang-Syne.mp3 TRADITIONAL

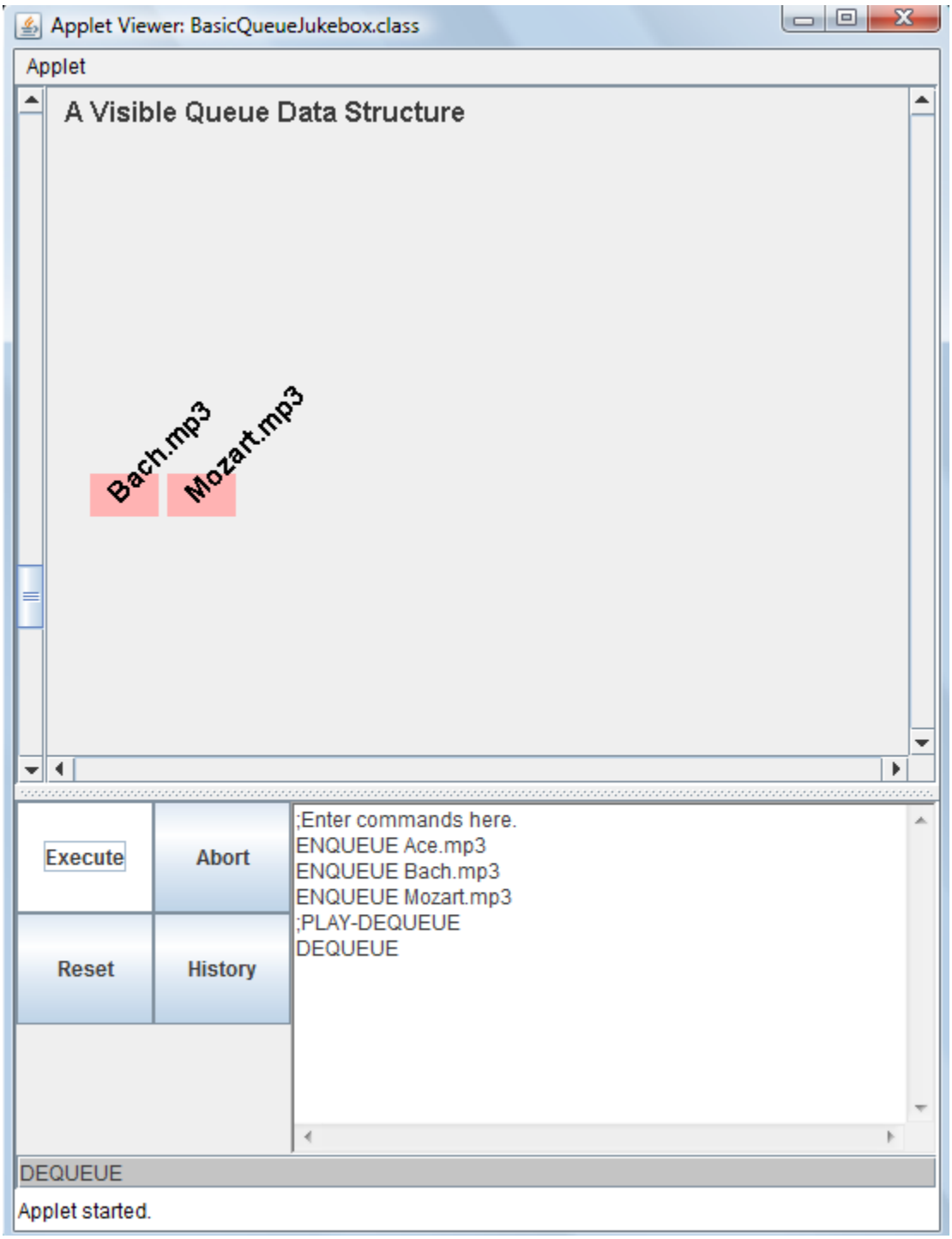
XDEQUEUE *queue-name*
Removes the next element from the queue having the given name.

XPLAY-DEQUEUE *queue-name*
Removes the next element from the queue having the given name, and plays it, assuming it is the name of an MP3 file.

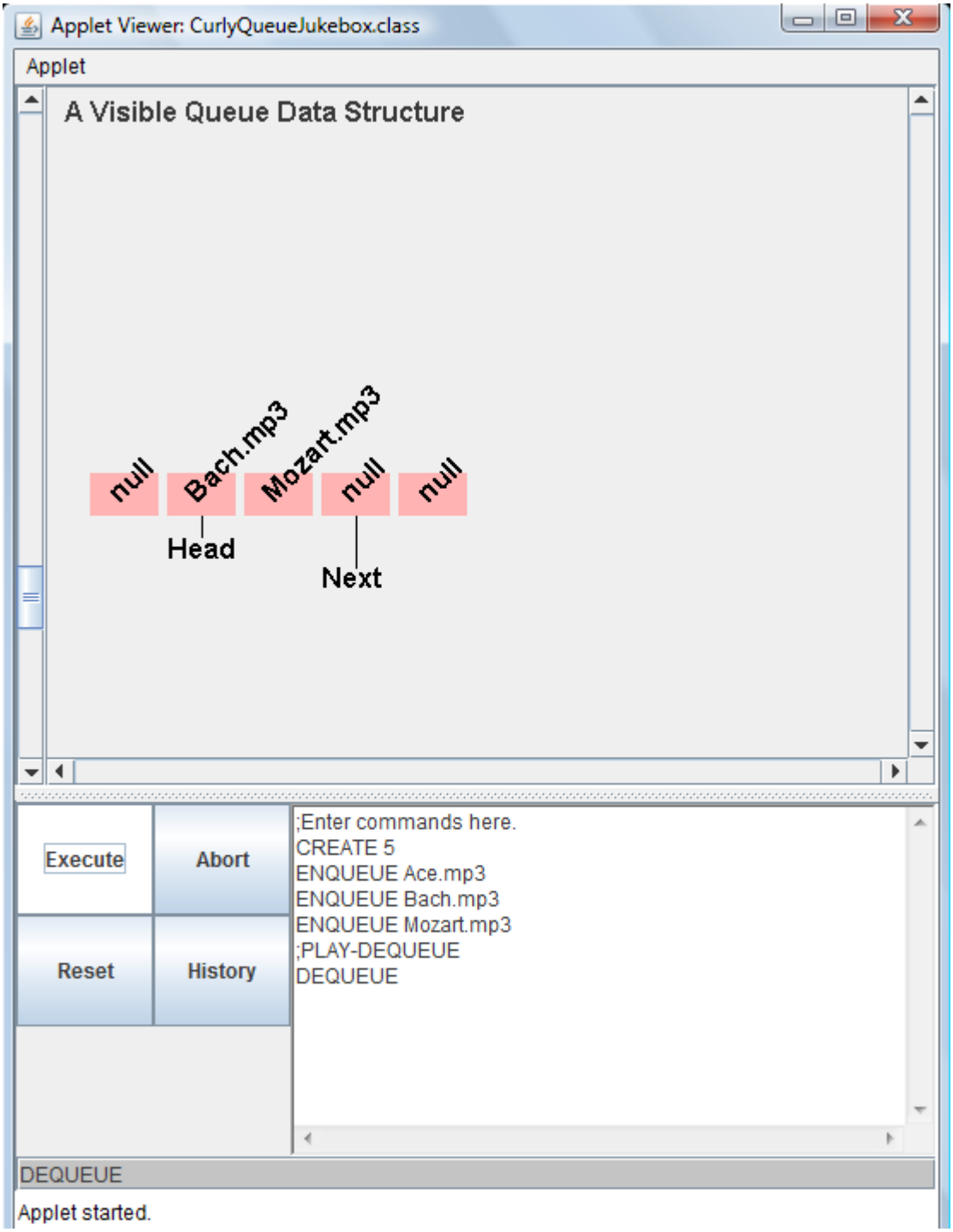
XPLAY-RR
Begins playing all the queued up items by going from one queue to the next, dequeuing and playing one piece at a time. It uses a round-robin policy, taking turns among the queues that still have entries left in them. It stops when all queues are empty or when interrupted by the user clicking on the Abort button.

In Phase B, your functionality should be a superset of the Phase A functionality. That means that all Phase A commands should still work. You may assume that Phase A commands always work on the first queue that currently exists, and that if no queue exists, a default empty queue is constructed with name QUEUE0.

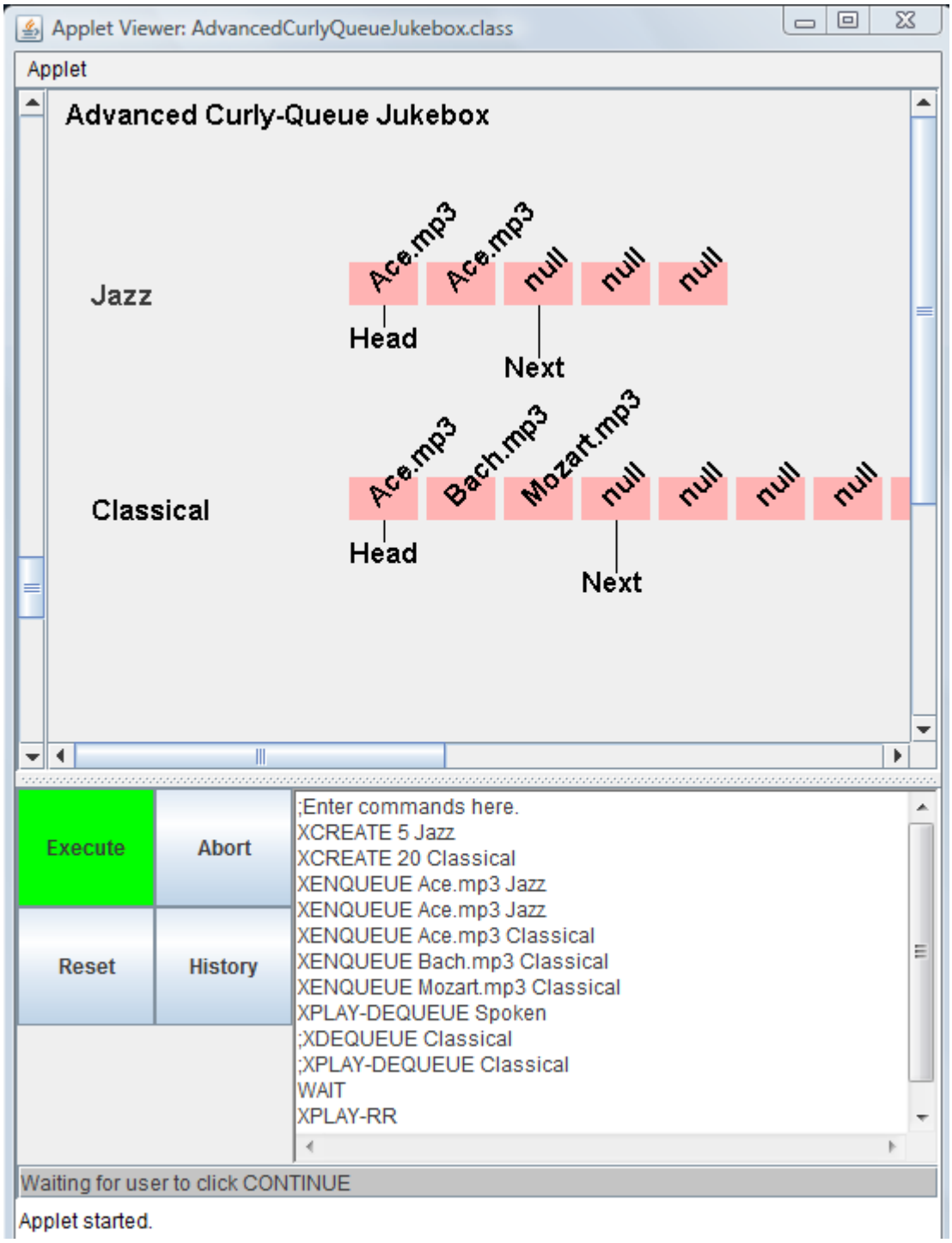
The STATS command should be updated to correctly give the total number of explicit ENQUEUE or XENQUEUE operations performed, the total number of explicit DEQUEUE, PLAY-DEQUEUE, XDEQUEUE, or XPLAY-DEQUEUE commands, and the total number of implicit DEQUEUE operations caused by executing XPLAY-RR commands. These statistics should be session totals, rather than per queue.



Screen shot of a sample solution at step 9. This basic jukebox has a queue of MP3 files but the queue is not implemented as a circular array. (It's not yet a “curly queue”.) Here the head is at the left end of the queue, and new elements are ENQUEUEd at the right end.



Screen shot of sample solution after step 10. Here the queue is implemented as a circular array. The size of the array is controlled by the parameter to the CREATE command. This version is the CurlyQueueJukebox. The “head” and “next” elements in the circular array implementation of the queue are shown explicitly with “call-out” lines and labels.



The advanced version of the jukebox allows the user to specify any number of queues and associate a name with each one. In this example, a queue for Jazz has been created with a maximum capacity of 5 entries, and a queue for Classical music has been set up for up to 20 entries. Some entries have been enqueued and execution has reached the WAIT command. Although there is an error in the user's input (asking to play from a non-existent queue “Spoken”) the program ignored that request and moved on to the next command. The Execute button is green when execution is in progress or waiting. The waiting is over when the user clicks on a “Click to Continue” dialog box (not shown in this illustration).

Further Instructions and Information: Additional instructions for “preloading” a particular demonstration into your applet will be given later. The preloaded commands are specified in the source code and placed in the text area for textual commands.
Updates and clarifications to this project will be posted in the Catalyst GoPost discussion topic “Project 1”.