MIT-AITI Design Exam **June 29th, 2005**

This exam will test your practical Java programming and design skills. You are provided with a project specification and must implement a solution. Your exam solution will be graded on whether it meets the specified functionality, as well as its organization and clarity. Someone else should be able to understand, integrate, and maintain your solution by reading your code.

Partial credit will be given. It is recommended that you first write empty methods that return null or zero values before implementing the actual functionality. Your code should be properly commented and formatted. It is advisable to put a brief comment at the top of each method explaining what you are trying to do. This will be especially helpful for partial credit if your program does not compile. It gives us an idea of what you are trying to accomplish.

You may use any available resources on the exam, except your fellow classmates. You may not communicate with classmates in any way while the exam is in progress. Add comments to your source code citing any outside material you use. Plagiarism will not be tolerated. Students caught cheating will be immediately dismissed from the course and referred to the Dean.

Setup and Submission Instructions

- Create a new folder or project named with your first initial and last name, for example "Alyssa Hacker" would name her project "ahacker".

- In each class you implement, include the following field with your name:
  `public static final String MYNAME = "Alyssa Hacker";`

- Be sure to compile and test your code to ensure it works properly. However, you do not need to submit your test code.

- When you have completed the exam, drag the entire folder into the DropBox.

- Contact a staff member if you have difficultly submitting your exam solution or need clarification on the exam description.

Problem Motivation

You have been hired to design a new digital music management system named MusicMatatu™ for Nairobi's hottest new nightclub, the *Herbivore*. Herbivore's DJ would like to build a large library of digital music tracks and be able to searching for tracks by song or artist names. You will implement a Track and MusicMatatu in this exam.

Track Class

Each Track has a song name, artist name, a song length (in seconds), and a count of how many times it has been played (play count). Tracks are instantiated with a song name, artist name, and song length. Tracks can be played, which increments their play count. Keep track of the total number of Track objects that are created. Think about the best way to do this. Design a Track class with the following API:

`Track(String artistName, String songName, int songLength)`
Creates a new Track object and initialize the appropriate data fields.

`String getSongName()`
Returns this Track's song name.

`String getArtistName()`
Returns this Track's artist name.

`int getSongLength()`
Returns the length of this Track's song, in seconds.

`int getPlayCount()`
Returns this Track's play count.

`static int numTracks()`
Returns the number of Tracks that have been instantiated.

`void play()`
Increments this Track's play count.

`public String toString()`
Returns a string representation of this track of the form "ArtistName – SongName".

MusicMatatu Class

MusicMatatus are initialized with no arguments and initially contain zero Tracks. Each Track must be individually added, and may be returned when searching by song or artist name. Since different Tracks may have the same song name or artist name, the search methods will return an array of all the Tracks that match the query String. If no such Tracks exist, **null** is returned. MusicMatatus also have a method to return the most played song and the number of songs currently in the MusicMatatu. Design a MusicMatatu class with the following API:

`public MusicMatatu()`
Instantiates a new MusicMatatu object.

`void addTrack(Track t)`
Adds the argument Track to this MusicMatatu

`Track[] getByArtist(String artistName)`
Returns an array containing all tracks that have ***exactly*** *the same artist name as the argument. Return* **null** *if no such Tracks exist. Note: This method should not call the Track.play() method.*

`Track[] getBySong(String songName)`
Returns an array containing all tracks that have ***exactly*** *the same artist name as the argument. Return* **null** *if no such Tracks exist. Note: This method should not call the Track.play() method.*

`int numTracks()`
Returns the number of Tracks stored in this MusicMatatu.

`Track topSong()`
Returns the most played song in this MusicMatatu. In the event of a tie, return the top song with the highest alphabetical order, sorting first by artist name, then by song name. Return **null** *if no Tracks exist.*

Extra Credit

Do not attempt the extra credit until you have completed the rest of the exam. Herbivore's DJ is lazy and wants MusicMatatu to build a playlist for her. She wants you to write a method that takes as input a maximum time, then returns an array of Tracks that maximize the total play count of all tracks while keeping the total play time less than the maximum play time. You can only play each song once per playlist.

Write down a method that attempts to output a play list that maximizes song popularity. This is trickier than it looks:

`Track[] playList(int totalTime)`
Returns an array of tracks whose total play time is less than the totalTime argument, while maximizing the total of all the Tracks' play counts.

EC.S01 Internet Technology in Local and Global Communities
Spring 2005-Summer 2005