CS100J Fall 2006 Assignment A5. Mozart's Musikalisches Würfelspiel. Due Sunday, 5 November

The inspiration for this assigment comes from a similar assignment given by Kevin Wayne and Robert Sedgewick in Computer Science, Princeton. The assignment illustrates the use of two-dimensional arrays and also random-number generation in an interesting setting.

In 1787, Wolfgang Amadeus Mozart created a dice game (Mozart's Musikalisches Würfelspiel). In the game, one composes a two-part waltz by pasting together 32 of 272 precomposed measures (written by Mozart) at random. We will do a variation of Mozart's original Würfelspiel.

Your program will generate a waltz consisting of a trio followed by a minuet. Each is 16 measures long, and the measures are generated at random according to a fixed set of rules.

- *Trio.* The trio consists of 16 measures. There are 96 possible Trio measures, named `T1.wav` through `T96.wav`. To determine which one to play, roll one fair die and use the following table.

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	measure number
1	72	6	59	25	81	41	89	13	36	20	46	79	30	95	19	66	
2	56	82	42	74	14	7	26	71	76	5	64	84	8	35	47	88	
3	75	39	54	1	65	43	15	80	9	34	93	48	57	58	90	21	
4	40	73	16	68	29	55	2	61	22	67	49	77	69	87	33	10	
5	83	3	28	53	37	17	44	70	63	85	32	96	12	23	78	91	
6	18	45	62	38	4	27	52	94	11	92	24	86	51	60	50	31	

  For example, if you roll 5 for measure 3, then play measure 28.

- *Minuet.* The minuet consists of 16 measures. There are 176 possible Minuet measures, named `M1.wav` through `M176.wav`. To determine which one to play, roll two fair dice, and use the following table.

	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	measure
2	96	22	141	41	105	122	11	30	70	121	26	9	112	49	109	14	
3	32	6	128	63	146	46	134	81	117	39	126	56	174	18	116	83	
4	69	95	158	13	153	55	110	24	66	139	15	132	73	58	145	79	
5	40	17	113	85	161	2	159	100	90	176	7	4	67	160	52	170	
6	148	74	163	45	80	97	36	107	25	143	64	125	76	136	1	93	
7	104	157	27	167	154	68	118	91	138	71	150	29	101	162	23	151	
8	152	60	171	53	99	133	21	127	16	155	57	175	43	168	89	172	
9	119	84	114	50	140	86	169	94	120	88	48	166	51	115	72	111	
10	98	142	42	156	75	129	62	123	65	77	19	82	137	38	149	8	
11	3	87	165	61	135	47	147	33	102	4	108	164	144	59	173	78	
12	54	130	10	103	28	37	106	5	35	20	31	92	12	124	44	131	

Example. This sample waltz is generated using mozart's original process; here is the accompanying musical score.

There are 6**16 * 11**16 different possible waltzes. Since this is over 10**23 different possibilities, each time you play the game you are likely to compose a piece of music that has never been heard before. Mozart carefully constructed the measures to obey a rigid harmonic structure, so each waltz reflects Mozart's distinct style.

About this assignment. You will use a number of ideas for the first time: generation of random numbers, two-dimensional arrays, playing music stored in a .wav file, saving in a file on your hard drive a **double** array, and more. To help you learn all this and also to give you more advice on how to go about developing and testing programs, we lead you through this assignment in a series of steps. Follow them carefully. When doing one step, don't start the next one until the current one is done and the methods you wrote for it work!

You will also have to submit a file MozartTester.java, which will contain test cases that you used in testing methods.

Step 1. Download the following and place them all in a new folder for the assignment.
File StdAudio.java. Class `StdAudio` contains methods for manipulating and playing music in wave (.wav) format.
File Mozart.java. You will be writing class `Mozart`. We have provided you with a few components to help out.

File waves.zip (34MB) OR waves.sitx (25MB). If you have a PC, you probably have to use waves.zip; mac people can use the smaller .sitx file. After downloading a file, unpack it into a folder `waves` that contains all the .wav files for the measures used in Mozart's Musikalisches Würfelspiel. Put folder `waves` in the folder with the two .java files.

Step 2. Generating rolls of a die. Your program will have to "roll a die" to produce a random number in the range 1..6. At the beginning of class `Mozart`, there is a declaration of a static `Random` variable `generator`. An object of class `java.util.Random` has methods for generating "random" numbers. Please bring up the Java 1.5 API package in your favorite browser, find class `Random`, and read about it. You will use function `nextInt`. Evaluation of a call `generator.nextInt(t)` yields an integer `i` that satisfies `0 <= i < t`. Use function `nextInt` in writing a method with the following specification in class `Mozart`:

/** = a roll of a die --an int in the range 1..6 */
public static int throwDie()

Function `throwDie` has to be tested to make sure that it will (1) produce only integers in the range 1..6 and (2) will produce all integers in that range at least once. In JUnit class `MozartTester`, write a method that will test `throwDie`. In the method, have a loop that calls `throwDie` 200 times and tests each value it produces. In addition to method `assertEquals`, you can use method `assertTrue(b)`, which passes the test if `b` is true and stops testing if `b` is false. For example, you could use something like this:

```
assertTrue(1 <= die && die <= 6);
```

Step 3. A method for printing a String array. Later, you will be writing a function that produces a `String` array of file names corresponding to measures to be played. In order to see the results of the method, you have to see the array of `Strings`. For that purpose, write and test the following function.

/** = a representation of array s --the list of Strings in the array, with each pair separated by ", " and the list delimited by "[" and "]".
Example: For array {"fi", "se", "th"}, return "[fi, se, th]" */
public static String arrayToString(String[] s)

Write a method in class `MozartTester` to test it. Be sure you test arrays of length 0, 1, and bigger.

Step 4. Generating a waltz. Before you work on creating a random waltz, first create a waltz assuming that each die thrown has the value 1, so that all the file names in a roll of 1 for the trio and a roll of 2 of the minuet are chosen. This will allow you to concentrate on constructing file names, as we discuss below. Thus, write a method with this specification:

/** = an array that contains the names of all the files
described by a roll of 1 for the trio and a roll of 2 for the minuet.
Each filename must be of the form

"waves/T<integer>.wav" or "waves/M<integer>.wav",

where <integer> is some positive integer.
Thus, the files are expected to be in directory waves.*/
public static String[] create1Spiel()

We have given you arrays `minuet` and `trio` in class `Mozart`. Their meaning are given as comments on their declarations. As an example, `minuet[0][2] = 22`, representing file `waves/M22.wav`, is the number to use for the musical phrase to use for a roll of a 2 in measure 2. So, you have to put the `String` `"waves/M22.wav"` into the array returned by this function. Note that "/" is used to separate folder name `waves` from file name `M22.wav`. Even if you have a PC, you must use "/" and not a backslash.

After writing this function, check it out by executing `s= Mozart.create1Spiel()` in the interactions pane and then displaying the contents of `s` using `Mozart.toString(s)` and checking to make sure that the write file names are displayed. You do not have to write a test method in `MozartTester` to test `create1Spiel`; it would be too time consuming.

Make sure that the result of `Mozart.create1Spiel()` is an array of size 32!

Step 5. Listen to the music! Now write the following method to play all the files whose names are in an array. Look in class `StdAudio` for a method that will play the music in a file given by a file name. You will notice that there is a pause between measures when the music is played. Later, we investigate eliminating the pauses. Do NOT assume that parameter s is an array of 32 elements; it can be any length.

```
/** Play the music given by files whose names are in s, in order*/
public static void play(String[] s)
```

Playing your array should produce the same music as this file: mozart12.wav.

Step 6. Wouldn't you like to get rid of the pauses? One way to do this is to build a single file that contains the music in the files given by an array like `s` in the past two steps. Write the following method:

```
/** Put the measures given by the file names in s into
a new array and return the new array */
public static double[] build(String[] s)
```

To do this, you have to read a .wav file and place its contents into a **double** array; find a method to do this in class `StdAudio`.

Your method should do the following. First, determine the length of the output array --read all the .wav files given by `s` to do this. Then, declare the output array of the appropriate size. Finally, read the .wav files (again) and copy their values into the output array, one after the other. To help you out, we already placed a method `copy` in class `Mozart` .

Step 7. Creating a Mozart Musikalisches Würfelspiel. Write the following method:

```
/** = an array of random measure file names:16 for trios followed by16 for minuets. The filenames are of
the form "waves/M<integer>.wav" and "waves/T<integer>.wav", so the files are expected
to be in directory waves, which should be in the same folder as this class. */
public static String[] randomSpiel()
```

This method should produce a random waltz as described at the beginning of this document. For this method, you can use method `throwDie`, which you wrote earlier, to throw a die to get a number in the range 1..6. Also, use arrays `minuet` and `trio` in class `Mozart` when generating random names of files, as in method `create1Spiel`. You do not need a method in `MozartTester` to test it.

Step 8. Saving a file. We write a method to generate a waltz and save its .wav file so that you can play it later --or so that you can email it home to let your family know that you have been turned on by classical music and Musikalisches Würfelspiel. You will want to use method `StdAudio.save` in writing this procedure. The method body will create a waltz, build a double array that contains all the measures, and then save the waltz on your hard drive. Be careful in testing it. Make sure the name that you use is not already the name of a file on your desktop, or you will lose that file. You do not need a method in `MozartTester` to test it.

```
/** Create a random waltz (using function randomSpiel()) and store it on the desktop of this computer
under the name filename. If filename does not end in ".wav", then put that extension at the
end of the filename before creating the file.*/
public static void randomSpiel(String filename)
```

Step 9. Submitting your assignment. Submit your files `Mozart` and `MozartTester` on the CMS by the due date and time. As usual, your methods should have precise and complete specifications, written as javadoc comments. There will be severe deductions if these javadoc comments are not suitable. We suggest that you generate the javadoc API specs and look at them carefully.