CS100J Spring 2006 Assignment A5. Mozart's Musikalisches Würfelspiel. Due on the CMS at 11:59PM, Monday, 3 April

The inspiration for this assigment comes from a similar assignment given by Kevin Wayne and Robert Sedgewick in Computer Science and Princeton. The purpose of the assignment is to show you the use of two-dimensional arrays in an interesting setting, which also get you familar with random-number generation.

In 1787, Wolfgang Amadeus Mozart created a dice game (Mozart's Musikalisches Würfelspiel). In the game, you compose a two-part waltz by pasting together 32 of 272 pre-composed musical elements at random. The waltz consists of two parts: a minuet and a trio. Each is composed of 16 measures, which are generated at random according to a fixed set of rules, as described below.

- *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.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
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	34	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	31	164	144	59	173	78
12	54	130	10	103	28	37	106	5	35	20	108	92	12	124	44	131

  For example, if you roll an 11 for measure 3, then play measure 165.

- *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.

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

Example. Here is a sample waltz generated using this process and the accompanying musical score. There are 11^16 * 6^16 different possible results, some of which are more likely than others. 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. In 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 a double array that contains music in a file on your hard drive, 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. Please follow them carefully. When doing one step, don't go on to the next one until the current one is done and the method you wrote for it work!

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 in it to help out.
File waves.zip (34MB) or waves.sitx (25MB). 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. If you are on a PC, you probably have to use waves.zip; mac people can use the smaller .sitx file. 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 `generator` has methods for generating "random" numbers. The one you will use is function `nextInt`. Evaluation of a call `generator.nextInt(t)` yields an integer `i` that satisfies `0 ?… i < t`. You should use function `nextInt` to write a method with the following specification in class `Mozart`:

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

Class `Random` is in API package `java.util`. Use the link in the course webpage to obtain the API spec for `Random` and spend some time becoming familiar with it.

Function `throwDie` seems extremely simple! Nevertheless, it has to be tested to make sure that it will produce only integers in the range 1..6 and can produce all integers in that range. Test it! One way is to write a method that calls the `throwDie` 50 times, each time printing the result in the interactions pane. Then you can look at the values it produced.

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 that the method works, 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: "[first, second, what]" */
public static String toString(String[] s)

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 row 2 of the minuet and row 1 of the trio are chosen. This will allow you to concentrate on the weird part of constructing a file name, as we discuss below. Thus, write a method with this specification:

/** = A String array that contains the names of all the files
described by row 2 of array minuet and row 1 of array trio.
The filenames must be of the form
"waves/M<integer>.wav" and
"waves/T<integer>.wav",
so the files are expected to be in directory waves.*/
public static String[] create1Spiel()

We have given you arrays `minuet` and `trio` in class `Mozart`. Each element of of these arrays is an integer that indicates a file that represents a measure. For example, `minuet[2][1] = 96`, which represents file `waves/M96.wav`. So, you have to put the `String "waves/M96.wav"` into the array that this function returns. `minuet[2][1]` is one of the musical phrases that can be used in measure 1 of the minuet part. Note that "/" is used to separate folder name `waves` from file name `M96.wav`. Even if you have a PC, you must use "/" and not a backslash.

After writing this function, you can 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.

Step 5. Listen to the music! Now write the following method to play all the files whose names are in an array such as that calculated by function `createSpiel`. 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. We'll investigate eliminating the pauses later.

/** 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 a**n** 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 know how 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 first of all determine the length of the output array --by reading all the files--, then declare the output array, and finally copy the values of the .wav files given by `s` 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. Finally, write the following method:

```
/** = an array of random measure file names:16 for a minuet and 16 trios. 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[] createRandomSpiel()
```

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`.

Step 8. Saving a file. You don't have to do anything for this step. We just want to let you know that method `StdAudio.save` allows you to save a **double** array in a .wav file, so that you can play it later if you want ?\or email it home to let your family know that you have been turned on by classical music and Musikalisches Würfelspiel. You can use any file name you want, as long as it ends in ".wav". The file will be stored on your desktop.

Step 9. Submitting your assignment. Submit your file `Mozart` on the CMS by the due date: 11:59PM on Monday, 3 April. 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.