**chap 6 Making music: an on-screen piano**

**Objectives**

* Learn how to create a class that has its own private data and public constructor and methods
* Learn how to create array of objects
* Learn how to use loop to travel through an array

**Schedule:** This lesson covers Moodle folder “chap 6”. Please follow the steps below.

1, work on sec 6.1 ~ 6.6, which include scenario piano-1 to piano-4, and piano-complete.

2, Each array created in Java has a public constant data field ***length***, and it can be accessed as arrayName.length , and the value of this field is set to the array size when the array is created. That is why we have to specify the array size when creating an array. Notice that array index goes from 0 to arrayName.length -1

3, the definition of class Key can accommodate multiple key values and sounds, which makes this class versatile.

4, unzip file “**MemoryManagementForArray.zip**”, and study **MemoryManagementForArray.docx** that illustrates the memory management scheme of array, and compile and run the two java source files. You need to have good understanding of these concepts: **stack**, **heap**, **reference variable**. Here are the highlights about array :

* Array name is a reference variable, and it stays in stack
* Array body is stored in the heap, and it is being referred to by the array name.
* The ***new*** keyword in the array creation statement links the array body in the heap with the array name in the stack
* The ***new*** keyword in the array creation statement first allocates a block of memory in heap for the purpose of storing the array body, and then it returns the beginning address of this block of memory, so that it assigns this address back to the array name in the stack.
* array name actually stores the address of the block of memory in the heap, and that is why array name is called a “reference variable”, because its value is the address (reference) of the memory block in the heap.
* Just like an object variable is a reference variable, an array name is also a reference variable, because both an object and an array refers to a block of memory in the heap. If you review this document in Moodle folder “chap 3” --- **MemoryManagementForObject.zip**, you will see that there are quite some similarities between object variable and array name.

5, unzip file “**search-and-sorting**.***zip***”, you will see 5 java files for 5 search and sorting algorithms, respectively:

* two search algorithms: linear search, binary search
* three sorting algorithms: selection sort, insertion sort, bubble sort

Study these 5 algorithms, and understand how each algorithm works. You also need to learn how to trace binary search (in “test 2 review lesson”, there is an exercise for it), and understand why binary search is more efficient than linear search.

Under folder “SortComparison”, you will see one WORD document “algorithmComplexity.docx”, along with four other java source code files.

* This WORD document explains the concept of time complexity, the big-O notation, and the time complexity of all the search and sorting algorithms we have encountered so far in this chapter.
* Bubble sort, insertion sort, and selection are all the same in terms of time complexity, which is O(n2).
* Compile and run “TestQuickSort.java” and “TestBubbleSort.java”, respectively, and see how much time it takes for each sorting algorithm to finish execution, then you will notice the difference between O(n\*log(n)) time complexity for quick sort, and O(n2) for bubble sort.
* You are not required to understand the principle of quick sort, but you need to know the fact that the time complexity of quick sort is O(n\*log(n)), as explained in document “algorithmComplexity.docx”.

6, unzip file “TwoDimArrayDemo.zip”, you will see one java file “TwoDimArrayDemo.java”. This is an example of two-dimensional array, and you need to know the following about two-dimensional array:

* A two-dimension array can be viewed as “array of array”, as indicated below, because each element in the first dimension is a one-dimensional array. For instance, twoDim[0], twoDim[1], and twoDim[2], each of them is a one-dimensional array.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| twoDim[0] |  | twoDim[0][0] | twoDim[0][1] | twoDim[0][2] |
| twoDim[1] |  | twoDim[1][0] | twoDim[1][1] | twoDim[1][2] | twoDim[1][3] |
| twoDim[2] |  | twoDim[2][0] |
| twoDim[3] |  | twoDim[3][0] | twoDim[3][1] | twoDim[3][2] | twoDim[3][3] | twoDim[3][4] |
|  |
| int[][] twoDim |

You can see from the above picture that, each element in the twoDim array in a one-dimensional array, thus the two dimensional array itself becomes an array of arrays.

Notice that array reference variable ***twoDim*** locates in the stack of main method, and it points to a place in heap, where the contents of the two dimensional array locate, i.e., the four reference varaibles for the four one-dim arrays locate in the heap. Each of these four one-dim array (***twoDim[0]***, ***twoDim[1]***, ***twoDim[2]***, and ***twoDim[3]***) further points to another place in heap, where the contents of each one dimensional array locate, e.g., twoDim[0][2], twoDim[3][1]

* For initialize the dimensions, we can define the first dimension of the array, such as line 9 in file TwoDimArrayDemo.java, and define the second dimension later, such as line 13, 20, 21, and 22.
* Each individual array in the second dimension does not have to have the same array size as each other, as indicated in line 13, 20, 21, and 22, since each line has its own size. That is why two-dimensional array is called **ragged-array**, as illustrated by this picture (assume user enter 3 as size for array twoDim[0]):

|  |  |  |  |
| --- | --- | --- | --- |
| twoDim[0] | twoDim[0][0] | twoDim[0][1] | twoDim[0][2] |
| twoDim [1] | twoDim[1][0] | twoDim[1][1] | twoDim[1][2] | twoDim[1][3] |
| twoDim [2] | twoDim[2][0] |
| twoDim [3] | twoDim[3][0] | twoDim[3][1] | twoDim[3][2] | twoDim[3][3] | twoDim[3][4] |

* To navigate through a two-dimensional array, we need to use embedded regular for loops, as indicated in line 28~ 34, and line 48~53, and line 58~62, or embedded for-each loops as in line 71~75. In the embedded loops, the output loop controls the row index, and the inner loop controls the column index.
* Two-dimensional array can also be passed as actual parameter to a method, as indicated in line 38 and line 41. When receiving the actual parameter, the formal parameter has to be defined as a two-dimensional array.
* When passing an array as a method parameter, whether it is one-dimensional array, or two-dimensional array, it is similar to passing an object as a method parameter, because they are all pass-by-reference.

7, follow the instructions in file “**homework6.docx**”, and work on homework 6. After you finish it, you need to submit the solution zip file to its Moodle drop box. When coding your homework, please follow all the rules in file “RulesForIndentAndAlignCode.docx”.

8, keep working on the questions in file “Test2StudyGuide.docx”, and prepare for test 2. This file is available in Moodle folder “test 2 review lesson”.

9, next lesson, we will be working on Moodle folder “chap 7”, please study it in advance.