Arrays (Outsource: 8-1 – 8-15)

An array is a list of data items that all have the same type and the same name. Hence, one variable name can be used to reference many values. "Array" in programming means approximately the same thing as array, matrix, or vector does in math. Unlike math, you must declare the array and allocate a fixed amount of memory for it. Subscripts are enclosed in square brackets [ ].

Declaring an array (Outsource: 8-6)

You declare an array variable in the same way as you declare any scalar variable, but you insert a pair of square brackets after the type. For example, to declare an array of integers to hold student averages you would write **int[ ] studentAverage**. After you create the array variable you still need to create the actual array. You use the same procedure to create an array that you use to create an object. For example, to allocate space for 10 student averages you would write **studentAverage = new int[10]**. You could accomplish both tasks in one statement by writing **int[ ] studentAverage = new int[10]**.

Subscripts start at zero (Outsource: 8-3)

Subscript ranges always start at zero. This isn't the way that humans normally count, but it's the way that C, C++, Java and many other languages do it. Although the element with subscript zero is allocated, it is not necessary to use it, so you can program using subscripts from 1 up. However, most Java programs use subscripts starting at 0, and we'll use that convention here.

Length of an array (Outsource: 8-14)

The length of an array in Java is stored in the member function length and can be used in a program to control access to the array. The following program declares an array of Strings and inputs ten names from the keyboard into the array. The length of an array is a measurement of the array's capacity.

```
String[ ] name = new String[10];     // declares an array of Strings
System.out.println(name.length);     // results in an output of 10
```

Initializing an Array (Outsource: 8-5)

A variable that has a primitive type, such as **int**, holds a value. A variable with a reference type, such as an array, holds a memory address where a value is stored.

Array names actually represent computer memory addresses; that is, array names are references, as are all Java objects. When you declare an array name, no computer memory address is assigned to it. Instead, the array variable name has the special value **null**.

When you define **studentAverage** as **int[ ] studentAverage = new int[10];**, then **studentAverage** has an actual memory address value. Each element of **studentAverage** has a value of zero because **studentAverage** is a numeric array. By default, numeric array elements are assigned *0*, character array elements are assigned *'\0'*, **boolean** array elements are assigned *false* and Object array elements, such as Strings, are assigned **null**.

An array can be declared and initialized using the following syntax:

String[] flintstones = {"Fred", "Wilma", "Barney", "Betty"};

The length of the array is implicitly defined by the number of elements included within the **{ }**.

Partially Filled Arrays

When working with arrays it is not always possible to know ahead of time exactly how large the array needs to be. In that case, you must declare an array that you believe will be large enough to hold all the data. In other words, take your best guess and simply make the array bigger than you think you will need.

The Scanner Class

Method Summary – Scanner class	
boolean	**hasNext** () Returns true if this scanner has another token in its input.
boolean	**hasNextDouble**() Returns true if the next token in this scanner's input can be interpreted as a double value using the nextDouble() method.
boolean	**hasNextInt**() Returns true if the next token in this scanner's input can be interpreted as an int value using the nextInt() method.
boolean	**hasNextLine**() Returns true if there is another line in the input of this scanner.
Scanner	**useDelimiter**(String pattern) Sets this scanner's delimiting pattern to a pattern constructed from the specified String.

Reading Data From A Text File Of Unknown Size

It is often necessary to read data from a text file of unknown size. The Scanner class provides a number of overloaded methods for determining if a scanner object has another token in its input. These methods are particularly useful for detecting the end of a data file. All you need to do is instantiate a Scanner object to read from a text file. Then, using a loop, read from the file as long as it has more tokens, storing the data in an array. Of course you need a counter to keep track of exactly how many items were read and stored in the array. For example, the following Java

program segment reads from a data file using a Scanner object as long as the file has another int to be scanned.

```
int[ ] list = new int[1000];
int count = 0;
Scanner reader = new Scanner(new File("data.dat"));
while (reader.hasNextInt())
    list[count++] = file.nextInt();
```

When the loop exits list is filled with count number of integers. We can now process the array using count as a loop terminator.

```
for (int i = 0; i < count; i++)
   System.out.print(list[i] + " ");
```

The following partial list of **Scanner** methods is provided for your perusal.

The Arrays Class

The Arrays class, located in the java.util package, contains various methods for manipulating arrays (such as sorting and searching).

Searching

Two array processing techniques that are particularly common are searching and sorting. Searching for an item in an array can be accomplished by calling Arrays.binarySearch. Arrays.binarySearch is an overloaded method. In the two examples below *dataType* can be replaced by any primitive data type or by the Object class.

Arrays - Binary Search Method Summary

`static int`	**`binarySearch`**`(`*`dataType`*`[] a, `*`dataType`*` key)` Searches the specified array of *dataType* for the specified value using the binary search algorithm.
`static int`	**`binarySearch`**`(`*`dataType`*`[] a, int fromIndex, int toIndex,` *`dataType`*` key)` Searches a range of the specified array of *dataType* for the specified value using the binary search algorithm.

Arrays.binarySearch returns the index of the search key, if it is contained in the array; otherwise, it returns a negative number (-(insertion point) - 1). The insertion point is defined as the point at which the key would be inserted into the array: the index of the first element greater than the key, or a.length if all elements in the array are less than the specified key. Note that this guarantees that the return value will be >= 0 if and only if the key is found.

Sorting

Sorting refers to rearranging all the items in the array into increasing or decreasing order (where the meaning of increasing and decreasing can depend on the context). Sorting an array can be accomplished by calling Arrays.sort. Arrays.sort is also an overloaded method. In the two examples below *dataType* can be replaced by any primitive data type or by the Object class.

Arrays - Sort Method Summary	
`static void`	**`sort`**`(`*`dataType`*`[] a)` Sorts the specified array of *dataType* into ascending numerical order.
`static void`	**`sort`**`(`*`dataType`*`[] a, int fromIndex, int toIndex)` Sorts the specified range of the specified array of *dataType* into ascending numerical order.

LAB 10 - ASSIGNMENT

Lab 10A - 60 points

OBJECTIVE

WAP that reads a series of numbers from a data file (**"lab10a.dat"**) and stores them in an array of integers. Make the capacity of the array of size 400. Sort the numbers and output them. DO NOT output empty elements of the array. Then output the total number of values actually stored in the array and the number of unused elements.

FIELD SUMMARY

- **int[ ] list** – an array of integers.
- **int count** – an integer to keep track of the number of array elements used.

METHOD SUMMARY

- **main** – instantiate an object of your class. Make method calls to input, and output.
- **input** – declare a scanner object and read a series of integers from a data file storing them in ***list***.
- **output** – All the numbers stored in ***list*** and the *number of elements containing values* and *the number of elements that are unused*.

SAMPLE DATA FILE INPUT

<Data NOT shown>

SAMPLE OUTPUT

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346
This array contains 346 numbers.
There are 54 unused elements.

Lab 10B - 70 points

OBJECTIVE

WAP that reads a series of numbers from a data file (**"lab10b.dat"**) and stores them in an array of integers. Sort and output all of the numbers in a row followed by the sum and the average of the eight numbers.

HINT: You need a counting loop in ***process*** and ***output.***

FIELD SUMMARY

- **int[ ] list** – an array of eight integers
- **int count** – an integer to keep track of the number of array elements used.
- **Int sum** – the sum of all the numbers in *list*.
- **Double average** – the average of all the numbers stored in *list*.

METHOD SUMMARY

- **main** – instantiate an object of your class. Make method calls to `input`, `process`, and `output`.
- **input** – declare a `Scanner` object and read eight integers from a data file storing them in ***list***.
- **process** – sort list and then sum up all the values stored in *list* (A loop is required). Then calculate the *average*.
- **output** –Display all the numbers in ***list***. Then output the *sum* and *average* of all the numbers in *list*.

SAMPLE DATA FILE INPUT

16 12 8 45 27 81 44 53

SAMPLE OUTPUT

8 12 16 27 44 45 53 81
The sum of the numbers is 286
The average of the numbers is 35.8

Lab 10C - 80 points

OBJECTIVE

WAP that reads a series of numbers from a data file (**"lab10c.dat"**) into an array of doubles. Output all the numbers in ascending order. Then output the smallest value and the largest value. Format all doubles to show no more than two decimal places, suppressing trailing zeros.

HINT: You need a counting loop to process the array in ***output***. Since the array is sorted where would you expect to find the smallest number? The largest number?

FIELD SUMMARY

- **double[ ] list** – an array of ten doubles
- **int count** – an integer to keep track of the number of array elements used.

METHOD SUMMARY

- **main** – instantiate an object of your class. Make method calls to `input`, and `output`.
- **input** – declare a `Scanner` object and read a series of doubles from a data file storing them in an array.
- **output** – Sort the array and display all the numbers. Then display the smallest and largest values in the array.

SAMPLE DATA FILE INPUT

62.4 21.7 77.3 94.8 14.1 56.3 -12.9 48.6 44.4 -25.5

SAMPLE OUTPUT

The sorted list (-25.5 -12.9 14.1 21.7 44.4 48.6 56.3 62.4 77.3 94.8).
The smallest value is -25.5.
The largest value is 94.8.

Lab 10D - 90 points

OBJECTIVE

WAP that reads a series of names and ages from a data file (**"lab10d.dat"**) and stores them into two separate arrays (parallel arrays). The names in an array of Strings and the ages in an array of type int. Output the names and ages in a column after all names and ages have been entered.

FIELD SUMMARY

- **String[ ] names** – an array of five strings (names)
- **int[ ] age** – an array of five integers (ages)

METHOD SUMMARY

- **main** – instantiate an object of your class. Make method calls to `input`, and `output`.
- **input** – declare a `Scanner` object and read a series of names and ages from a data file storing them in two parallel arrays.
- **output** – Display all the names and corresponding ages. Use a counting loop that uses the array's length as a terminating value.

SAMPLE DATA FILE INPUT

Grace Kelly/28
Bob Hope/42
Frank Lovejoy/32
William Shatner/45
Howdy Doody/12
Lester Scruggs/54

SAMPLE OUTPUT

Grace Kelly is 28 years old.
Bob Hope is 42 years old.
Frank Lovejoy is 32 years old.
William Shatner is 45 years old.
Howdy Doody is 12 years old.
Lester Scruggs is 54 years old.

Lab 10E - 100 points

OBJECTIVE

WAP that reads a series of doubles from a data file (**"lab10e.dat"**) into an array of doubles. Then read two doubles from the keyboard, a value to search for and a replacement value if the search value is contained in the list. Sort the array and then search the array for the search item using the `Arrays binarySearch` method. If the search item is found, replace it with the replacement value and output the modified list, otherwise indicate the search item was not found and output the original list.

Hint: Use the **`binarySearch`** method of the **`Arrays`** class to search for a specific value. If the value is found, replace it, otherwise output an error message indicating the value was not found.

```
int pos = Arrays.binarySearch(list, 0, count, key);          // Search list for key
if (pos < 0)
        /* key was NOT found */
else
        /* key was found */
```

FIELD SUMMARY

- **double[ ] list** – an array of doubles read from a data file.
- **int count** – the number of values stored in *list*.
- **double searchItem** – the number to be found if it exists.
- **double replaceValue** – the number to be used as a replacement value if searchItem is found in the list.

METHOD SUMMARY

- **main** – instantiate an object of your class. Make a method call to fileInput, kybdInput and output.
- **fileInput** – declare a scanner object and read a series of double values from a data file ("lab16d.dat") storing them in an array of doubles.
- **kybdInput** – declare a scanner object and read a number to be found and a replacement value from the keyboard using appropriate prompts.
- **output** – Sort the array. Call the **Arrays.binarySearch** method searching for the number to be found. *If* the number is found replace it with the replacement value *otherwise* output an appropriate message, i.e. **<search item> not found**. Output the contents of the array (do not output empty elements of the array).

SAMPLE DATA FILE INPUT

<Data NOT shown>

SAMPLE KEYBOARD INPUT

Enter a value to search for: **62.91**
Enter a replacement value: **999.99**

SAMPLE OUTPUT

The Modified List:
1.01 1.03 1.05 1.12 1.13 1.18 1.33 1.37 1.41 1.46 1.49 1.77 1.84 2.01 2.07 2.16 2.17 2.27 2.28 2.34 2.4 2.41 2.42 2.47 2.55 2.58 2.6 2.83 2.85 2.9 3.1 3.2 3.24 3.32 3.55 3.62 3.68 4.45 4.46 4.63 4.68 4.69 4.93 5.15 5.36 5.5 5.54 5.59 5.64 5 .72 5.74 5.75 5.79 5.82 6.05 6.07 6.09 6.11 6.15 6.24 6.34 6.43 6.77 6.89 7.15 7.22 7.58 7.78 8.03 8.24 8.4 8.43 8.48 8.49 8.61 8.7 8.94 9.15 9.22 9.48 9.53 9.62 9.67 9.72 9.73 9.96 10.09 10.25 10.42 10.49 10.57 10.68 10.72 10.74 10.9 10.92 11.0 11.3 11.45 11.58 11.62 11.68 11.74 11.9 11.93 12.02 12.17 12.25 12.51 12.56 12.77 12.92 13.04 13.07 13.08 13.19 13.32 13.51 13.73 13.84 13.85 13.9 13.92 14.09 14.27 14.37 14.38 14.6 14.72 14.87 14.9 15.1 15.23 15.35 15.39 15.5 15.53 15.61 15.65 15.69 15.74 15.95 16.2 16.24 16.39 16.45 16.77 17.18 17.26 17.33 17.41 17.49 17.66 17.9 18.18 18.22 18.26 18.31 18.32 18.42 18.43 18.48 18.55 18.62 18.72 18.77 18.95 18.97 19.11 19.12 19.24 19.27 19.33 19.39 19.43 19.53 19.59 19.73 19.74 19.79 19.98 20.07 20.11 20.29 20.36 20.44 20.7 21.02 21.09 21.23 21.29 21.31 21.4 21.42 21.76 21.8 22.07 22.5 22.84 22.85 23.03 23.04 23.08 23.33 23.42 23.48 23.49 23.5 23.57 23.66 23.67 23.72 23.73 23.76 23.99 24.05 24.07 24.08 24.09 24.11 24.18 24.24 24.4 24.74 24.76 24.82 24.96 24.97 25.03 25.26 25.49 25.58 25.62 25.81 25.82 25.99 26.2 26.22 26.23 26.32 26.43 26.44 26.49 26.59 26.6 26.75 26.83 26.85 26.95 26.99 27.1 27.16 27.44 27.58 27.64 27.82 27.88 27.89 27.95 28.21 28.38 28.49 28.58 28.59 28.65 28.67 28.69 28.93 28.96 29.2 29.41 29.43 29.68 29.94 30.17 30.36 30.43 30.53 30.63 30.73 30.76 30.78 30.96 31.3 31.37 31.43 31.53 31.61 31.71 31.73 31.74 31.78 32.02 32.03 32.05 32.11 32.24 32.45 32.63 32.68 32.69 32.74 33.05 33.07 33.24 33.27 33.38 33.5 33.57 33.63 33.65 33.81 33.83 34.0 34.07 34.09 34.16 34.24 34.32 34.52 34.58 34.64 34.71 34.72 34.93 34.96 35.3 35.33 35.34 35.39 35.48 35.57 35.66 35.69 35.81 35.92 35.94 36.03 36.09 36.12 36.22 36.24 36.3 36.33 36.36 36.41 36.44 36.45 36.47 36.62 36.66 36.73 36.78 36.79 36.8 36.84 36.94 37.01 37.07 37.39 37.41 37.44 37.55 37.57 37.6 37.75 37.97 38.07 38.09 38.28 38.31 38.34 38.44 38.58 38.65 38.74 38.78 38.81 38.91 38.96 39 .27 39.35 39.41 39.47 39.55 39.71 39.82 39.84 39.99 40.08 40.11 40.23 40.36 40.39 40.55 40.69 40.77 40.87 40.91 40.95 41.01 41.47 41.52 41.66 42.08 42.18 42.2 42.26 42.32 42.39 42.4 42.51 42.54 42.6 42.79 42.88 42.92 42.98 43.04 43.11 43.14 43.16 43.34 43.49 43.96 44.17 44.2 44.22 44.33 44.43 44.79 44.85 44.96 44.99 45.23 45.31 45.34 45.36 45.43 45.46 45.54 45.72 45.8 45.86 45.99 46.31 46.34 46.4 46.49 46.59 46.74 46.8 46.84 46.89 46.96 47.02 47.15 47.8 47.82 47.89 47.92 48.06 48.2 48.26 48.33 48.39 48.59 48.67 49.0 49.02 49.29 49.37 49.42 49.69 49.78 49.86 49.91 49.95 50.02 50.03 50.16 50.26 50.27 50.57 50.75 50.81 50.85 50.9 51.26 51.26 51.29 51.35 51.37 51.41 51.46 51.47 51.68 51.78 51.92 52.19 52.42 52.61 52.65 52.66 52.67 52.69 52.94 52.98 53.0 53.12 53.24 53.4 53.52 53.63 53.74 53.82 53.9 54.1 54.14 54.19 54.22 54.31 54.33 54.36 54.38 54.42 54.46 54.48 54.52 54.55 54.58 54.62 54.65 54.66 54.81 54.84 54.87 54.9 54.97 55.09 55.28 55.29 55.45 55.73 55.94 56.21 56.27 56.35 56.41 56.44 56.5 56.59 56.61 57.14 57.19 57.22 57.29 57.3 57.31 57.39 57.42 57.65 57.68 57.7 57.95 57.97 57.98 58.06 58.09 58.21 5 8.25 58.42 58.49 58.6 58.66 58.77 58.82 58.83 58.86 58.93 58.99 59.0 59.15 59.2 59.37 59.6 59.72 59.87 59.9 59.95 60.05 60.07 60.13 60.15 60.24 60.27 60.43 60.48 60.64 60.77 60.94 61.07 61.16 61.36 61.47 61.56 61.57 61.62 61.71 61.74 61.76 62.2 62.32 62.34 62.38 62.4 62.44 62.49 62.51 62.52 62.64 62.65 62.69 62.74 62.78 62.8 62.81 62.87 999.99