Chap08Yourname.java

Arrays

```
import java.util.*;
//We import java.util.Scanner and java.util.Arrays
//A wildcard is a symbol used to replace or represent one or more characters.
import java.io.*;

public class Chap08 {

    private static Scanner in;

    public static void main(String[] args) {
        PrintStream out = System.out;
        in = new Scanner(System.in);
```

8.1 Creating arrays

```
out.println("8.1 Creating arrays");
//An array is a sequence of values. The values in the array are called
//elements. All the values in an array must have the same type.

int[] a;  //Declaration of a variable with an int array type.
a = new int[4]; //Assignment. Creation of the array of four integer elements.
//int[] counts = new int[4];
//The size of an array can be any non negative integer.
out.println();
```

8.2 Accessing elements

```
//Values of counts is a reference to the array.
//An array is not the same things the variable referring to it.
//We can assign different variables to refer to the same array, and we
//can change the value of a variable to refer to a different array.
```

8.2 Accessing elements

```
//The indexes of an array of n elements are: 0, 1, 2, ... n - 1.

a[0] = 1;
a[1] = 2;
a[2] = 3;
a[3] = 4;

int[] b = {5, 6, 7, 8};
//Array constants can only be used in initializers.
out.println();
```

8.3 & 8.5 Displaying arrays, array length

```
out.println("8.3 & 8.5 Displaying arrays, array length");
out.println("println(a) = " + a);
out.println("println(b) = " + b);
//The bracket indicates that the value is an array. "I" stands for
//"integer", and the rest represents the address of the array.
```

8.3 & 8.5 Displaying arrays, array length

```
public static void printArray(int[] a) {







}
```

8.3 & 8.5 Displaying arrays, array length

```
public static void printArray(int[] a) {
        System.out.print("{" + a[0]);
        for (int i = 1; i < 4; i++) {
                System.out.print(", " + a[i]);
        }
        System.out.println("}");
}
```

8.3 & 8.5 Displaying arrays, array length

```
public static void printArray(int[] a) {
        System.out.print("{" + a[0]);
        for (int i = 1; i < a.length; i++) {
                System.out.print(", " + a[i]);
        }
        System.out.println("}");
}
```

8.3 & 8.5 Displaying arrays, array length

```
printArray(a);
printArray(b);
out.println(Arrays.toString(a));
out.println(Arrays.toString(b));
out.println();
```

8.4 Copying arrays

```
out.println(“8.4 Copying arrays”);
int[] c = a;
printArray(c);
a[0] = 2;
printArray(c);
//a and c are different variables referring to the same array.
//They are called aliases. As we change the value of a, c changes with it.
out.println();
```

8.4 Copying arrays

```
int[] d = new int[a.length];
for (int i = 0; i < a.length; i++) {
        d[i] = a[i];
}
a[0] = 3;
printArray(a);
printArray(d);
//We actually copy the array that a referring to. So d does not change with a.
out.println();
```

8.4 Copying arrays

```
int[] e = Arrays.copyOf(a, 4);
a[0] = 4;
printArray(a);
printArray(e);
//We actually copy the array that a referring to. So e does not change with a.
out.println();
```

8.4 Copying arrays

```
int[] f = Arrays.copyOfRange(a, 2, 4);
//Copy the range that 2< = (index of a) < 4
printArray(f);
out.println();
```

8.6 Array traversal and search

```
//Looping through the elements of an array is called a traversal.
a[0] = 1;
printArray(a);
for (int i = 0; i < a.length; i++) {
        a[i] = (int) Math.pow(a[i], 2);
}
printArray(a);
out.println();
```

8.6 Array traversal and search

```
//Search involves traversing an array looking for a particular
//element. The following example displays the index where the
//target value first appears.
a[0] = 0;
a[1] = 0;
a[2] = 0;
a[3] = 0;
out.print("Array a = ");
printArray(a);
```

8.6 Array traversal and search

```
out.print("Search for the integer n you type in array a. n = ");
int n = in.nextInt();
if (search(a, n) >= 0) {
        out.printf("%d first appears at index = %d.\n", n, search(a, n));
} else {
        out.printf("%d does not appear in array a.\n", n);
}
out.println();
```

8.6 Array traversal and search

```
public static int search(int[] a, int n) {







}
```

8.6 Array traversal and search

```
public static int search(int[] a, int n) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] == n) {
            return i;
        }
    }

}
```

8.6 Array traversal and search

```
//A reduce operation reduces an array of values down to a single
//value. The following example displays the sum of elements of
//array a.
a[0] = 1;
a[1] = 2;
a[2] = 3;
a[3] = 4;
out.print("Array a = ");
printArray(a);
out.printf("The sum of elements of array a is %d.\n", sum(a));
out.println();
```

8.6 Array traversal and search

```
public static int sum(int[] a) {







}
```

8.6 Array traversal and search

```
public static int sum(int[] a) {
        int sum = 0;
        for (int i = 0; i < a.length; i++) {
                sum += a[i];
        }
        return sum;
}
```

8.6 Array traversal and search

```
//Write a method min(a) to return the minimum value of elements
//of array a.
out.print("Array a = ");
printArray(a);
out.printf("The minimum value of elements of array a is %d.\n",
                min(a));
out.println();
```

8.6 Array traversal and search

```
public static int min(int[] a) {










}
```

8.6 Array traversal and search

```
public static int min(int[] a) {
    int min = a[0];
    for (int i = 1; i < a.length; i++) {
        if (a[i] < a[i - 1]) {
            min = a[i];
        }
    }
    return min;
}
```

8.7 Random numbers

```
out.println("8.7 Random numbers");
int[] integers = ranArray(100);
out.println("We have generated an array of which the 100 elements are"
                + " random integers between 1 and 100. The array is: ");
printArray(integers);
out.println();
```

8.7 Random numbers

```
public static int[] ranArray(int n) {
        Random ran = new Random();
        int[] ranArray = new int[100];
        for (int i = 0; i < 100; i++) {
                ranArray[i] = ran.nextInt(100) + 1;
                //The method nextInt takes an integer argument n, and returns a random
                //integer between 0 and n - 1 (inclusive), that is [0, n - 1]
        }
        return ranArray;
}
```

8.8 Traverse and count

```
out.println("8.8 Traverse and count");
out.print("We want to count how many times a certain integer n appears in "
            + "the array. n = ");
n = in.nextInt();
out.printf("%d appeares %d times in the array.\n", n, count(integers, n));
out.println();
```

8.8 Traverse and count

```
public static int count(int[]a, int n) {



}
```

8.8 Traverse and count

```
public static int count(int[]a, int n) {
        int count = 0;
        for (int i = 0; i < a.length; i++) {
                if (a[i] == n) {
                        count++;
                }
        }
        return count;
}
```

8.8 Traverse and count

```
//Write a method, interval(int[]a, int low, int high),
//counting how many elements in the array fall in the interval (low, high].
out.println("We want to count how many elements in the array fall in the "
            + "inteval (low, high].");
out.print("low = ");
int low = in.nextInt();
out.print("high = ");
int high = in.nextInt();
out.printf("%d elements fall in the interval (%d, %d].\n",
            interval(integers, low, high), low, high);
out.println();
```

8.8 Traverse and count

```
public static int interval(int[]a, int low, int high) {







}
```

8.8 Traverse and count

```
public static int interval(int[]a, int low, int high) {
    int count = 0;
    for (int i = 0; i < a.length; i++) {
        if (low < a[i] && a[i] <= high) {
            count++;
        }
    }
    return count;
}
```

8.8 Traverse and count

```
//Create an array int[] frequencies = new int[10], of which the 10 elements
//counting how many random integers fall in the interval (0, 10], (10, 20]
//...(90, 100] respectively. Display array frequency.

int[] frequencies = new int[10];
```

8.8 Traverse and count

```
//Create an array int[] frequencies = new int[10], of which the 10 elements
//counting how many random integers fall in the interval (0, 10], (10, 20]
//...(90, 100] respectively. Display array frequency.

int[] frequencies = new int[10];
for (int i = 0; i < frequencies.length; i++) {
        frequencies[i] = interval(integers, 10 * i, 10 * i + 10);
}
printArray(frequencies);
out.println();
```

8.8 Traverse and count

```
//Every time the loop invokes interval(array, low, high), it traverses the
//entire array. Try to traverse the array only once:
```

8.8 Traverse and count

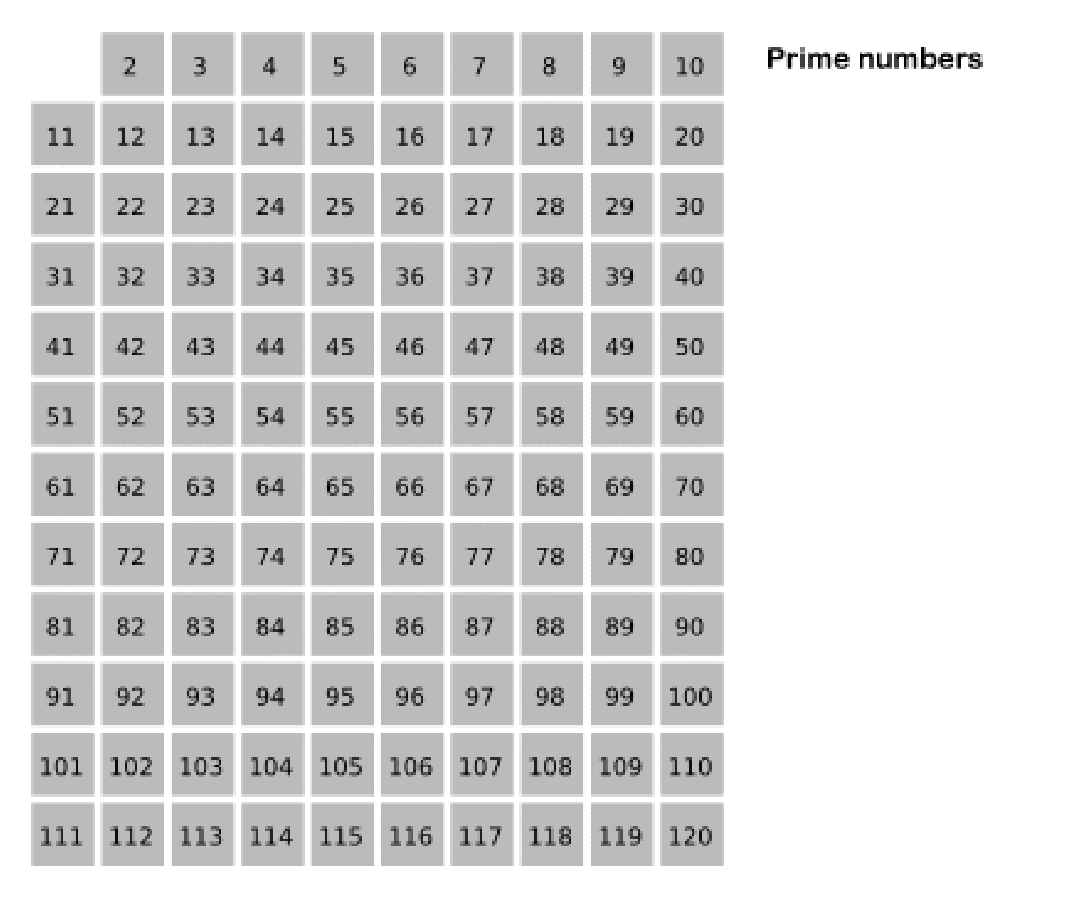
```
//Every time the loop invokes interval(array, low, high), it traverses the
//entire array. Try to traverse the array only once:

for (int iIntg = 0; iIntg < integers.length; iIntg++) {
        int iFreq = (integers[iIntg] - 1) / 10;
        frequencies[iFreq]++;
}
printArray(frequencies);
out.println();
```

8.10 The enhanced for loop

```
out.println("8.10 The enhanced for loop");
//It is conventional to use plural nouns for array variables and
//singular nouns for element variables.
for (int integer : integers) {
        int iFreq = (integer - 1) / 10;
        frequencies[iFreq]++;
}
printArray(frequencies);
out.println();
```

Ex 8.5 The Sieve of Eratosthenes

1. Create a list of consecutive integers from 1 through n: (1, 2, 3, 4, ..., n).
2. Initially, let p equal 2, the smallest prime number.
3. Enumerate the multiples of p by counting to n from 2p in increments of p, and mark them in the list (these will be 2p, 3p, 4p, ...; the p itself should not be marked).
4. Find the first number greater than p in the list that is not marked. If there was no such number, stop. Otherwise, let p now equal this new number (which is the next prime), and repeat from step 3.
5. When the algorithm terminates, the numbers remaining not marked in the list are all the primes below n.



Prime numbers

	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