**Unit 8:** Arrays

Answer the questions and complete the programs in preparation for the end of Unit exam

**Review Questions:**

1. Consider the following array:  
   int[] a = { 1, 2, 3, 4, 5, 4, 3, 2, 1, 0 };

What are the contents of the array a after the following loops complete?

1. for(int i =1; i < 10; i++){ a[i] = a[i – 1];}

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

1. for(int i =9; i > 0; i--){ a[i] = a[i – 1];}

[1, 1, 2, 3, 4, 5, 4 ,3, 2, 1]

1. for(int i =0; i < 9; i++){ a[i] = a[i + 1];}

[2, 3, 4, 5, 4, 3, 2, 1, 0, 0]

1. for(int i =8; i >= 0; i--){ a[i] = a[i + 1];}
2. for(int i =1; i < 10; i++){ a[i] = a[i] + a[i - 1];}
3. for(int i =1; i < 10; i = i + 2){ a[i] = 0;}
4. for(int i =0; i < 5; i++){ a[i + 5] = a[i];}
5. for(int i =1; i < 5; i++){ a[i] = a[9 - i];}
6. Write the Java code for a loop that simultaneously computes both maximum and minimum values in an array.

double min = Double.POSITIVE\_INFINITY;

double max = Double.NEGATIVE\_INFINITY;

for(double n : inputs) {

if(n < min) {

min = n;

}

if(n > max) {

max = n;

}

}

1. A run is a sequence of adjacent repeated values. Give pseudocode for computing the length of the longest run in a array. For example, the longest run in the array with elements  
    1 2 5 5 3 1 2 4 3 2 2 2 2 3 6 5 5 6 3 1  
   has a length of 4.

int run\_max = 0;

int run = 0;

int previous = (int) Double.NaN;

for(int n : inputs) {

if(n == previous) {

run++;

if(run > run\_max) {

run\_max = run;

}

} else {

run = 1;

}

previous = n;

}

1. What is wrong with the following method that aims to fill an array with random numbers?

public void makeCombination(int[] values, int n)

{

Random generator = new Random();

int[] numbers = new int[values.length];  
 for( int i = 0; i < numbers.length; i++)

{

numbers[i] = generator.nextInt(n);

}

values = numbers;

}

**Exercises:**

1. Write array methods that carry out the following tasks for an array of integers by completing the ArrayMethods class below. For each method, provide a test program.

public class ArrayMethods{

private int[] values;  
 public ArrayMethods(int[] initialValues) { values = initialValues; }  
 public void swapFirstAndLast(){ …. }  
 public void shiftRight() { … }  
 …..  
}

* 1. Swap the first and last element in the array.

public void swapFirstAndLast(){

int first = values[0];

int last = values[values.length-1];

values[0] = last;

values[values.length-1] = first;

}

* 1. Shift all elements by one to the right and move the last element into the first.

public void shiftRight() {

int last = values[values.length-1];

for(int i = 1; i < values.length; i++) {

values[i] = values[i - 1];

}

values[0] = last;

}

* 1. Replace all even elements with 0.

for(int i = 0; i < values.length; i ++) {

if(values[i] % 2 == 0) {

values[i] = 0;

}

}

* 1. Replace each element except the first and last by the larger of its two neighbors.

int[] result = new int[values.length];

int lastIndex = values.length-1;

result[0] = values[0];

result[lastIndex] = values[lastIndex];

for(int i = 1; i < lastIndex; i++) {

result[i] = Math.max(values[i-1], values[i+1]);

}

values = result;

* 1. Remove the middle elements if the array length is odd, or the middle two elements if the length is even.

int[] result = new int[0];

if(values.length % 2 == 1) {

result = new int[values.length - 1];

int middle = values.length / 2;

for(int i = 0; i < values.length; i++) {

if(i != middle) {

result[i > middle ? i - 1 : i] = values[i];

}

}

} else if(values.length % 2 == 0) {

result = new int[values.length - 2];

int right = values.length / 2;

int left = right - 1;

for(int i = 0; i < values.length; i++) {

if(i != left && i != right) {

result[i > right ? i - 2 : i] = values[i];

}

}

}

values = result;

* 1. Move all even elements to the front, otherwise preserve the order of the elements.
  2. Remove the second-largest element in the array.
  3. Return true if the array is currently sorted in increasing order.
  4. Return true if the array contains two adjacent duplicate elements.
  5. Return true if the array contains duplicate elements (which need not be adjacent)

1. *Magic squares*. An *n x n* matrix that is filled with numbers 1,2,3,…,n^2 is a magic square if the sum of the elements in each row, in each column, and in the two diagonals is the same value.  
   Write a program that reads in 16 values from the keyboard and tests whether they form a magic square when put into a 4 x 4 array.

|  |  |  |  |
| --- | --- | --- | --- |
| 16 | 3 | 2 | 13 |
| 5 | 10 | 11 | 8 |
| 9 | 6 | 7 | 12 |
| 4 | 15 | 14 | 1 |