AP Computer Science Problem Set 4 **Name: ______________________________**

1. ______ Consider the following code segment.

```
int[] arr = {1, 2, 3, 4, 5, 6, 7};

for (int k = 3; k < arr.length - 1; k++)
  arr[k] = arr[k + 1];
```

Which of the following represents the contents of `arr` as a result of executing the code segment?

(A) `{1, 2, 3, 4, 5, 6, 7}`

(B) `{1, 2, 3, 5, 6, 7}`

(C) `{1, 2, 3, 5, 6, 7, 7}`

(D) `{1, 2, 3, 5, 6, 7, 8}`

(E) `{2, 3, 4, 5, 6, 7, 7}`

2. ______ Assume that `myList` is an `ArrayList` that has been correctly constructed and populated with objects. Which of the following expressions produces a valid random index for `myList` ?

(A) `(int)( Math.random() * myList.size() ) - 1`

(B) `(int)( Math.random() * myList.size() )`

(C) `(int)( Math.random() * myList.size() ) + 1`

(D) `(int)( Math.random() * (myList.size() + 1) )`

(E) `Math.random(myList.size())`

3. ______ Consider the following method.

```
public static void arrayMethod(int nums[])
{
  int j = 0;
  int k = nums.length - 1;

  while (j < k)
  {
    int x = nums[j];
    nums[j] = nums[k];
    nums[k] = x;
    j++;
    k--;
  }
}
```

Which of the following describes what the method `arrayMethod()` does to the array `nums`?

(A) The array `nums` is unchanged.

(B) The first value in `nums` is copied to every location in the array.

(C) The last value in `nums` is copied to every location in the array.

(D) The method generates an `ArrayIndexOutOfBoundsException`.

(E) The contents of the array `nums` are reversed.

4. ______ Consider the following method, which is intended to return the element of a 2-dimensional array that is closest in value to a specified number, `val`.

```
/** @return the element of 2-dimensional array mat whose value is closest to val */
public double findClosest(double[][] mat, double val)
{
  double answer = mat[0][0];
  double minDiff = Math.abs(answer - val);
  for (double[] row : mat)
  {
    for (double num : row)
    {
      if ( /* missing code */ )
      {
        answer = num;
        minDiff = Math.abs(num - val);
      }
    }
  }
  return answer;
}
```

Which of the following could be used to replace `/*` *missing code* `*/` so that `findClosest` will work as intended?

(A) `val - row[num] < minDiff`

(B) `Math.abs(num - minDiff) < minDiff`

(C) `val - num < 0.0`

(D) `Math.abs(num - val) < minDiff`

(E) `Math.abs(row[num] - val) < minDiff`

5. ______ Consider the following instance variable and method.

```
private List<String> animals;

public void manipulate()
{
  for (int k = animals.size() - 1; k > 0; k--)
  {
    if (animals.get(k).substring(0, 1).equals("b"))
    {
      animals.add(animals.size() - k, animals.remove(k));
    }
  }
}
```

Assume that `animals` has been instantiated and initialized with the following contents.

```
["bear", "zebra", "bass", "cat", "koala", "baboon"]
```

What will the contents of `animals` be as a result of calling `manipulate` ?

(A) `["baboon", "zebra", "bass", "cat", "bear", "koala"]`

(B) `["bear", "zebra", "bass", "cat", "koala", "baboon"]`

(C) `["baboon", "bear", "zebra", "bass", "cat", "koala"]`

(D) `["bear", "baboon", "zebra", "bass", "cat", "koala"]`

(E) `["zebra", "cat", "koala", "baboon", "bass", "bear"]`

6. ______ Consider the following code segment.

```
int[] arr = {7, 2, 5, 3, 0, 10};
for (int k = 0; k < arr.length - 1; k++)
{
  if (arr[k] > arr[k + 1])
    System.out.print(k + "   " + arr[k] + "   ");
}
```

What will be printed as a result of executing the code segment?

(A) 0 2 2 3 3 0

(B) 0 7 2 5 3 3

(C) 0 7 2 5 5 10

(D) 1 7 3 5 4 3

(E) 7 2 5 3 3 0

7. ______ Consider the following code segment.

```
int[] oldArray = {1, 2, 3, 4, 5, 6, 7, 8, 9};
int[][] newArray = new int[3][3];

int row = 0;
int col = 0;
for (int value : oldArray)
{
  newArray[row][col] = value;
  row++;
  if ((row % 3) == 0)
  {
    col++;
    row = 0;
  }
}

System.out.println(newArray[0][2]);
```

What is printed as a result of executing the code segment?

(A) 3

(B) 4

(C) 5

(D) 7

(E) 8

8. ______ Assume that the array `arr` has been defined and initialized as follows.

```
int[] arr = /* initial values for the array */ ;
```

Which of the following will correctly print all of the odd integers contained in `arr` but none of the even integers contained in `arr` ?

(A)
```
for (int x : arr)
  if (x % 2 != 0)
    System.out.println(x);
```

(B)
```
for (int k = 1; k < arr.length; k++)
  if (arr[k] % 2 != 0)
    System.out.println(arr[k]);
```

(C)
```
for (int x : arr)
  if (x % 2 != 0)
    System.out.println(arr[x]);
```

(D)
```
for (int k = 0; k < arr.length; k++)
  if (arr[k] % 2 != 0)
    System.out.println(k);
```

(E)
```
for (int x : arr)
  if (arr[x] % 2 != 0)
    System.out.println(arr[x]);
```

9. ______ Consider the following method.

```
public int compute(int n, int k)
{
  int answer = 1;

  for (int i = 1; i <= k; i++)
    answer *= n;

  return answer;
}
```

Which of the following represents the value returned as a result of the call `compute(n, k)` ?

(A) n*k

(B) n!

(C) n^k

(D) 2^k

(E) k^n

10. ______ Consider the following class declarations.

```
public class Point
{
  private double x;   // x-coordinate
  private double y;   // y-coordinate

  public Point()
  {
    x = 0;
    y = 0;
  }

  public Point(double a, double b)
  {
    x = a;
    y = b;
  }

  // There may be instance variables, constructors, and methods that are not shown.
}


public class Circle
{
  private Point center;
  private double radius;

  /** Constructs a circle where (a, b) is the center and r is the radius.
   */
  public Circle(double a, double b, double r)
  {
    /* missing code */
  }
}
```

Which of the following replacements for `/* missing code */` will correctly implement the `Circle` constructor?

I.
```
center = new Point();
radius = r;
```

II.
```
center = new Point(a, b);
radius = r;
```

III.
```
center = new Point();
center.x = a;
center.y = b;
radius = r;
```

(A) I only

(B) II only

(C) III only

(D) II and III only

(E) I, II, and III

Free Response

A two-dimensional array of temperatures is represented in the following class.

```
public class TemperatureGrid
{
   /** A two-dimensional array of temperature values, initialized in the constructor (not shown)
    *  Guaranteed not to be null
    */
   private double[][] temps;

   /** Computes and returns a new temperature value for the given location.
    *  @param row a valid row index in temps
    *  @param col a valid column index in temps
    *  @return the new temperature for temps[row][col]
    *            - The new temperature for any element in the border of the array is the
    *              same as the old temperature.
    *            - Otherwise, the new temperature is the average of the four adjacent entries.
    *  Postcondition: temps is unchanged.
    */
   private double computeTemp(int row, int col)
   {   /* to be implemented in part (a) */   }


   /** Updates all values in temps and returns a boolean that indicates whether or not all the
    *  new values were within tolerance of the original values.
    *  @param tolerance a double value >= 0
    *  @return true if all updated temperatures are within tolerance of the original values;
    *          false otherwise.
    *  Postcondition: Each value in temps has been updated with a new value based on the
    *                 corresponding call to computeTemp.
    */
   public boolean updateAllTemps(double tolerance)
   {   /* to be implemented in part (b) */   }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

(a) Write method `computeTemp`, which computes and returns the new temperature for a given element of `temps` according to the following rules.

1. If the element is in the border of the array (in the first row or last row or first column or last column), the new temperature is the same as the old temperature.

2. Otherwise, the new temperature is the average (arithmetic mean) of the temperatures of the four adjacent values in the table (located above, below, to the left, and to the right of the element).

If `temps` is the table shown below, `temps.length` is `5` and `temps[0].length` is `6`.

	0	1	2	3	4	5
0	95.5	100.0	100.0	100.0	100.0	110.3
1	0.0	50.0	50.0	50.0	50.0	0.0
2	0.0	40.0	40.0	40.0	40.0	0.0
3	0.0	20.0	20.0	20.0	20.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0

The following table shows the results of several calls to `computeTemp`.

Function Call	Result
`computeTemp(2, 3)`	`37.5` (the average of the values 50.0, 20.0, 40.0, and 40.0)
`computeTemp(1, 1)`	`47.5` (the average of the values 100.0, 40.0, 0.0, and 50.0)
`computeTemp(0, 2)`	`100.0` (the same as the old value)
`computeTemp(1, 3)`	`60.0` (the average of the values 100.0, 40.0, 50.0, and 50.0)

Complete method `computeTemp` below.

```
/** Computes and returns a new temperature value for the given location.
 *  @param row  a valid row index in temps
 *  @param col  a valid column index in temps
 *  @return the new temperature for temps[row][col]
 *           - The new temperature for any element in the border of the array is the
 *             same as the old temperature.
 *           - Otherwise, the new temperature is the average of the four adjacent entries.
 *  Postcondition: temps is unchanged.
 */
private double computeTemp(int row, int col)
```

(b) Write method `updateAllTemps`, which computes the new temperature for every element of `temps`. The new values should be based on the original values, so it will be necessary to create another two-dimensional array in which to store the new values. Once all the computations are complete, the new values should replace the corresponding positions of `temps`. Method `updateAllTemps` also determines whether every new temperature is within `tolerance` of the corresponding old temperature (i.e., the absolute value of the difference between the old temperature and the new temperature is less than or equal to `tolerance`). If so, it returns `true`; otherwise, it returns `false`.

If `temps` contains the values shown in the first table below, then the call `updateAllTemps(0.01)` should update `temps` as shown in the second table.

`temps` before the call `updateAllTemps(0.01)`

	0	1	2	3	4	5
0	95.5	100.0	100.0	100.0	100.0	110.3
1	0.0	50.0	50.0	50.0	50.0	0.0
2	0.0	40.0	40.0	40.0	40.0	0.0
3	0.0	20.0	20.0	20.0	20.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0

`temps` after the call `updateAllTemps(0.01)`

	0	1	2	3	4	5
0	95.5	100.0	100.0	100.0	100.0	110.3
1	0.0	47.5	60.0	60.0	47.5	0.0
2	0.0	27.5	37.5	37.5	27.5	0.0
3	0.0	15.0	20.0	20.0	15.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0

In the example shown, the call `updateAllTemps(0.01)` should return `false` because there are several new temperatures that are not within the given tolerance of the corresponding old temperature. For example, the updated value in `temps[2][3]` is `37.5`, the original value in `temps[2][3]` was `40.0`, and the absolute value of `(37.5 - 40.0)` is greater than the value of `tolerance (0.01)`.