Olga

https://github.com/USF-CS601-Fall2023

especially: https://github.com/USF-CS601-Fall2023/final-project-declanelias

https://github.com/USF-CS601-Fall24

https://github.com/USF-CS545-Spring25

https://github.com/USF-CS545-S24

CS545 final exam:

```
/** A class that implements a Queue ADT using a circular array
 *  The code is modified from Prof. Galles's code.
 */
public class ArrayQueue implements Queue {

    static final int defaultSize = 100;

    private Object data[]; // the array that will store the queue
    private int head;
    private int tail;
    private int size; // the maximum # of elements it can hold

    ArrayQueue(int maxSize) {
        data = new Object[maxSize];
        head = 0;
        tail = 0;
        size = maxSize;
    }
```

```
public ArrayQueue() {
        data = new Object[defaultSize];
        head = 0;
        tail = 0;
        size = defaultSize;
}

/** Add an element to the end of the queue, if it's not full */
public void enqueue(Object elem) {
        // Before adding, check if the queue is full
        if ((tail + 1) % size == head) {
                System.out.println("Queue is full: Can't add an element");
                return;
        }
        data[tail] = elem;
        tail = (tail + 1) % size;
}

/**
 * Dequeues the element from the queue
 * @return element that was dequeued
 */
public Object dequeue() {
        Object retval;
```

```
        // Check if the queue is empty
        if (head == tail)
                return null;
        retval = data[head];
        head = (head + 1) % size;
        return retval;
}

public boolean empty() {
        return head == tail;
}

public String toString() {
        String result = "[";
        int tmpHead = head;
        if (tmpHead != tail) {
                result = result + data[tmpHead];
                tmpHead = (tmpHead + 1) % size;
                while (tmpHead != tail) {
                        result = result + "," + data[tmpHead];
                        tmpHead = (tmpHead + 1) % size;
                }
        }
        result = result + "]";
        return result;
}
```

```
}
/**
 * A class that represents an unweighted graph with an adjacency list
 */
public class Graph {
  private Edge[] graph; // Adjacency list for this graph.
  // For each vertexId, stores the first outgoing edge (head of the linked list)

  public static class Edge { // Class Edge
    private int neighbor; // id of the neighbor (id of the destination node)
    private Edge next; // reference to the next "edge"

    public Edge(int neighbor) {
      this.neighbor = neighbor;
      next = null;
    }
  }

  public Graph(int numVertices) {
    graph = new Edge[numVertices];
  }

  /**
   * Adds the given edge as an outgoing edge for the given vertex.
   * Modifies the adjacency list.
```

```
 *
 * @param vertexId id of the vertex
 * @param edge     outgoing edge
 *             Do not modify.
 */
public void addEdge(int vertexId, Edge edge) {
   Edge head = graph[vertexId]; // head of the linked list for this  node
   graph[vertexId] = edge; // insert in front
   if (head != null) {
      edge.next = head;
   }
}

/**
 * Use BFS (Breadth first search) to find the length of the shortest path
 * from vertex 1 to vertex 2 in the graph. Assume edges do not have weights.
 * A length of the shortest path is the number of edges in the shortest path.
 * @param vertex1 source vertex
 * @param vertex2 destination vertex
 * @return length of the shortest path (number of edges in the shortest path),
 * or -1 if no path exists between vertex 1 and vertex 2.
 */
public int findLengthOfShortestPath(int vertex1, int vertex2) {
   boolean[] visited = new boolean[graph.length];
   Queue queue = new ArrayQueue();
   visited[vertex1] = true;
```

```
    // For each vertex i, stores the number of edges on the shortest path from vertex 1 to
vertex i.
    int[] lengthOfShortestPath = new int[graph.length];
    for (int i = 0; i < lengthOfShortestPath.length; i++)
      lengthOfShortestPath[i] = -1;
    lengthOfShortestPath[vertex1] = 0;
    // FILL IN CODE:
    // Must use BFS to find the length of the shortest path from vertex 1 to vertex 2.
    // Must be efficient.


    queue.enqueue(vertex1);
    while (!queue.empty()) {
      int current = (int) queue.dequeue();
      Edge edge = graph[current];
      while (edge != null) {
        int neighbor = edge.neighbor;
        if (!visited[neighbor]) {
          visited[neighbor] = true;
          lengthOfShortestPath[neighbor] = lengthOfShortestPath[current] + 1;
          if (neighbor == vertex2) {
            return lengthOfShortestPath[neighbor];
          }
          queue.enqueue(neighbor);
        }
        edge = edge.next;
```

```
      }
    }

    return -1; // change as needed
}


public static void main(String[] args) {
    Graph g = new Graph(8);

    // edges going out of vertex 1
    Edge edge10 = new Edge(0);
    Edge edge12 = new Edge(2);
    g.addEdge(1, edge10);
    g.addEdge(1, edge12);

    // edge going out of 0
    Edge edge05 = new Edge(5);
    g.addEdge(0, edge05);

    //edge going out of 2
    Edge edge26 = new Edge(6);
    g.addEdge(2, edge26);

    // edges going out of 5
    Edge edge54 = new Edge(4);
```

```
        Edge edge56 = new Edge(6);
        g.addEdge(5, edge54);
        g.addEdge(5, edge56);

        // edge going out of 6
        Edge edge67 = new Edge(7);
        g.addEdge(6, edge67);

        //edge going out of 4
        Edge edge47 = new Edge(7);
        g.addEdge(4, edge47);

        // edge going out of 7
        Edge edge75 = new Edge(5);
        g.addEdge(7, edge75);

        System.out.println(g.findLengthOfShortestPath(3, 6)); // -1  (since no path exists)
        System.out.println(g.findLengthOfShortestPath(5, 4)); // 1
        System.out.println(g.findLengthOfShortestPath(7, 4)); // 2
        System.out.println(g.findLengthOfShortestPath(1, 5)); // 2
        System.out.println(g.findLengthOfShortestPath(2, 5)); // 3
        System.out.println(g.findLengthOfShortestPath(1, 4)); // 3
    }
}
import java.util.Arrays;
```

```
public class MergeSort {

  /**
   * A private mergeSort method - takes an array, and the indices that specify
   * what part of the array we are working with (from low to high)
   * Currently, it has bugs / missing code. You need to fix it.
   *
   * @param arr input array
   * @param low starting index of the subarray we need to sort
   * @param high end index of the subarray we need to sort (inclusive)
   */
  private static void mergeSort(int[] arr, int low, int high) {
    // FILL IN CODE: add a base case
    if (low >= high) return;
    int mid = (low + high) / 2;

    // FIX BUGS in how we make recursive calls:
    //mergeSort(arr, low, high);
    mergeSort(arr, low, mid);
    //mergeSort(arr, low, high);
    mergeSort(arr, mid + 1, high);
    merge(arr, low, mid, high); // See the merge method below - you need to fix bugs there
  }
```

```
/**
 * Merge two sorted subarrays together, one that goes from low to mid another
 * goes from mid+1 to high. Uses a temporary array.
 * Currently, has bugs / missing code. You need to fix them.
 * @param arr - array
 * @param low - first index of the first sorted chunk of the array
 * @param mid - the last index of the first sorted chunk
 * @param high - the last index of the second sorted chunk
 */
public static void merge(int[] arr, int low, int mid, int high) {
   int[] temp = new int[arr.length];
   int k = low;
   int i = low;
   int j = mid + 1;
   while (k <= high) {
      // FILL IN CODE: Fix bugs or add missing code
      if (i > mid) {
         temp[k] = arr[j];
         j++;
      }
      else if (j > high) {
         temp[k] = arr[i];
         i++;
      }

      else if (arr[i] < arr[j]) {
```

```
            temp[k] = arr[i];
            i++;
          } else {
            temp[k] = arr[j];
            j++;
          }
        k++;
      }
      // copy the result from temp back to arr
      for (k = low; k <= high; k++)
        arr[k] = temp[k];
    }

    public static void mergeSort(int[] arr) {
      mergeSort(arr, 0, arr.length - 1);
    }

    public static void main(String[] args) {
      int[] arr = {8, 1, 0, 5, 14, 10, 9, 7, 2, 11, 3, 25, 16};
      mergeSort(arr);
      System.out.println(Arrays.toString(arr));
    }
}
public interface Queue {
        void enqueue(Object elem);
        Object dequeue();
```

```
        boolean empty();
}
```

CS545 Midterm 2

```
class BinarySearchTree {

  /** An inner class representing a node in a binary search tree */
  private class BSTNode {
    int data; // value stored at the node
    BSTNode left; // left subtree
    BSTNode right; // right subtree

    BSTNode(int newdata) {
      data = newdata;
    }
  }

  private BSTNode root; // the root of the tree, an instance variable of class
BinarySearchTree

  public BinarySearchTree() {
    root = null; // initially, the tree is empty
  }

  /**
   * Insert a given element into the BST tree
```

```
 * @param elem element to insert into the BST tree
 */
public void insert(int elem) {
   root = insert(root, elem);
}

/**
 *
 * Counts how many nodes in the tree
 * are in the range from lowValue to highValue.
 * See the private method with the same name.
 * @return number of nodes with values from lowValue to highValue
 */
public int numOfNodesInRange(int lowValue, int highValue) {
   return numOfNodesInRange(root, lowValue, highValue);
}

/**
 * Insert elem into the tree with the given root
 * @param tree root of a tree
 * @param elem element to insert
 * @return the root of a tree that contains the new node
 */
private BSTNode insert(BSTNode tree, int elem) {
   if (tree == null) {
      return new BSTNode(elem);
```

```
    }
    if (elem < tree.data) {
      tree.left = insert(tree.left, elem);
      return tree;
    } else {
      tree.right = insert(tree.right, elem);
      return tree;
    }
  }

  /**
   * Return the # of nodes with values that are greater than lowValue and smaller than highValue.
   * @param node root of the tree
   * @param lowValue lower threshold
   * @param highValue higher threshold
   * @return number of nodes in the tree with the given root whose values
   * are greater than lowValue and smaller than highValue
   */
  private int numOfNodesInRange(BSTNode node, int lowValue, int highValue) {
    // FILL IN CODE: must be efficient - make use of the fact that it is
    // a binary search tree
    // See examples in the main method

    if (node == null)
      return 0;
```

```
    if (node.data >= highValue)
      return numOfNodesInRange(node.left, lowValue, highValue);
    if (node.data <= lowValue)
      return numOfNodesInRange(node.right, lowValue, highValue);
    return 1 + numOfNodesInRange(node.left, lowValue, highValue) +
numOfNodesInRange(node.right, lowValue, highValue);
  }

  public static void main(String[] args) {
    BinarySearchTree tree = new BinarySearchTree();
    tree.insert(10);
    tree.insert(8);
    tree.insert(6);
    tree.insert(4);
    tree.insert(1);
    tree.insert(7);
    tree.insert(0);
    tree.insert(2);
    tree.insert(3);
    tree.insert(20);
    tree.insert(25);
    tree.insert(22);
    /* The tree we created is as follows (see Canvas for a better picture):
          10
        8   20
       6     25
```

```
      4 7   22
     1
   0  2
     3
     */
     System.out.print("Number of nodes in the range (0, 5) (not including 0, 5)     ");
     System.out.println(tree.numOfNodesInRange(0, 5)); // 4 nodes (it counted nodes 4, 1, 2, 3). Note that lowValue and highValue are not included.
     System.out.print("Number of nodes in the range (5, 21) (not including 5, 21)    ");
     System.out.println(tree.numOfNodesInRange(5, 21)); // 5 nodes (it counted nodes 10, 8, 6, 7, 20). Note that lowValue and highValue are not included.
     System.out.print("Number of nodes in the range (12, 50) (not including 12, 50)  ");
     System.out.println(tree.numOfNodesInRange(12, 50)); // 3 nodes
     System.out.print("Number of nodes in the range (7, 24) (not including 7, 24)    ");
     System.out.println(tree.numOfNodesInRange(7, 24)); // 4 nodes
     // I recommend testing more on your own

  }
}
public class MinHeap {
  private int[] heap; // the heap array
  private int size; // the current number of elements in the heap

  /**
   * Constructor for the min heap
   * @param maxSize the maximum size of the heap
   */
```

```
public MinHeap(int maxSize) {
   heap = new int[maxSize];
   size = 0;
   heap[0] = Integer.MIN_VALUE;
}

 /** Return the index of the parent
 *
 * @param pos the index of the element in the heap array
 * @return the index of the parent
 */
private int parent(int pos) {
   return pos / 2;
}

/** Swap given elements: one at index pos1, another at index pos2
 *
 * @param pos1 the index of the first element in the heap
 * @param pos2 the index of the second element in the heap
 */
private void swap(int pos1, int pos2) {
   int tmp;
   tmp = heap[pos1];
   heap[pos1] = heap[pos2];
   heap[pos2] = tmp;
}
```

```
/** Print the array that stores the min heap.
 * Do not print MIN_VALUE at index 0, only print the actual heap values */
public void print() {
   int i;
   for (i = 1; i <= size; i++)
      System.out.print(heap[i] + " ");
   System.out.println();
}

/**
 * Takes the index of the heap element and the new value of the key.
 * Changes the key value to updatedKey and rearranges the heap.
 * Assumes the new key value is SMALLER than the previous one.
 * @param position index of the element in the heap array
 * @param updatedKey new value of the key
 */
public void decreaseKey(int position, int updatedKey) {
   // FILL IN CODE:
   // Assume the new key value is SMALLER than the previous one
   // You do NOT need to handle the case when the key increases.

   heap[position] = updatedKey;
   while (position > 1 && heap[parent(position)] > heap[position]) {
      swap(position, parent(position));
      position = parent(position);
```

```
    }
}


/** The method creates a simple min heap */
public void createSimpleHeap() {
    size = 7;
    heap[1] = 1;
    heap[2] = 6;
    heap[3] = 10;
    heap[4] = 8;
    heap[5] = 7;
    heap[6] = 12;
    heap[7] = 20;
    /*
          1
       6      10
      8   7    12  20

     */

}

public static void main(String[] args) {
    MinHeap minheap = new MinHeap(8);
    minheap.createSimpleHeap();
```

```
      /* Heap 1:
            1
         6      10
        8   7    12  20

       */
      System.out.println("Min Heap Before :");
      minheap.print();
      minheap.decreaseKey(6, 0);
      // decrease the key at index 6 (currently the key value 12) to 0.
      System.out.println("Min Heap after decreasing the key from 12 to 0: ");
      minheap.print(); // 0 6 1 8 7 10 20
      // Test more on your own

   }
}
```

CS545 midterm 1

Give Theta of the running time as a function of n for the method shown below and provide an explanation to get full credit.

```
public static int func1(int n) {
   int sum = 0;
   for (int i = 1; i <= n; i++) {
     for (int j = 1; j <= n; j = j * 2) { // note: j is multiplied by 2
       for (int k = n; k >= 1; k = k - 2) {
```

```
            sum  = sum + i + j*k;
        } // end of k loop
      } // end of j loop
   } // end of i loop

   // Note that l goes to n*n
   for (int l = 1; l <= n*n; l++) {
      sum++;
   }
   return sum;
}
```

Your answer:

(a) Answer (0.5 pt)

(b) Explanation (1.5 pt):

Make sure you explain the running time of each loop and explain how to combine them.

You may type your answers here, or write your solution on paper and submit it to the instructor at the end of the exam.

You may also type your answer in IntelliJ and commit&push it to Github.

Consider the following function, func2, that takes two arrays of integers,

a and b, of the same length n, and returns the number of elements in the b array

that are equal to one of the prefix sums of a.

A prefix sum at a given index i is the sum of elements from index 0 to index i.

Example: If the array is {1, 3, 4, 2}, then prefix sums will be 1, 4, 8, 10.

The function will count how many of those numbers occur in an array b.

```
public static int func2(int[] a, int[] b){
    int count = 0;
    int n = a.length; // assume the length of b is also n.
    for (int i = 0; i < n; i++) {
      int prefixSum = 0;
      for (int j = 0; j <= i; j++) {
        prefixSum = prefixSum + a[j];
      }
      // prefixSum is the sum of elements of a from index 0 to index i.
      System.out.println("prefix sum = " + prefixSum);
      for (int k = 0; k < n; k++) {
        if (b[k] == prefixSum) {
          count++;
        }
      }
    } // closed for i loop
    return count;
  }
```

Give Theta of the running time of func2 in terms of n, where n is the length of the array a (assume the length of the array b is also n).

(a) Answer: 0.3 pt

(b) Explanation: 0.7 pt

(Reminder: for nested loops where j depends on i, show the summation)

You may type your answers on Canvas, or write your solution on paper and submit it to the instructor at the end of the exam.

You may also type your answer here and commit&push this file to Github.

```
import java.util.Arrays;

/* The starter code shows the method called countingSortForCharacters which
  sorts a given array of characters in ascending order
  using the modified counting sort we covered in class.
  Example:
  If the input array is [g, b, a, c, e, m, a, p, a], the method will
  sort it as following: [a, a, a, b, c, e, g, m, p].

  Your goal is to implement another method, radixSortStrings,
  so that it uses radix sort to sort an array of strings in ascending order.
  Assume that all strings in the array have the same length, and it is small.
  Example: if the input array is
  [cat, cot, moo, pet, den, dog, pen, ale, red],
  after calling radixSortStrings, it should be sorted:
  [ale, cat, cot, den, dog, moo, pen, pet, red]
  To implement this method, you may use & modify parts of the method
countingSortForCharacters.
 */
// (4 pt)
public class Question3 {
```

```
public static void countingSortForCharacters(char[] characters) {
 int[] counterArr = new int[26]; // 26 letters in English alphabet
 for (int i = 0; i < characters.length; i++) {
  char ch = characters[i];
  int val = (int) ch - (int) 'a';
  // The formula above maps a letter to an index from 0 to 25
  // for instance, 'a' will be mapped to 0, 'b' to 1, 'c' to 2 etc.

  counterArr[val]++;
 }

 // Modify the counter array by combining counterArr[j] with counterArr[j-1]
 for (int j = 1; j < counterArr.length; j++)
  counterArr[j] = counterArr[j] + counterArr[j-1];

 // Write elements into the sorted array called "result"
 // Use the counter array to find the index of each character in the sorted array.
 char[] result = new char[characters.length];
 for (int i = characters.length - 1; i >= 0; i--) {
  char ch = characters[i];
  int val = (int) ch - (int) 'a';
  counterArr[val]--;
  result[counterArr[val]] = ch;
 }

 // Copy elements back to arr
```

```
   for (int k = 0; k < result.length; k++) {
     characters[k] = result[k];
   }
 }

  /** Should perform radix sort on strings of the same length.
   *  Intermediate steps of radix sort should be done using modified counting sort
   *  (see the starter code that shows how to do it for characters).
   *
   * @param strings input array of strings. Assume all strings have the same length, and it is small.
   */

  //public static void radixSortStrings(String[] strings) {
     // FILL IN CODE: implement radix sort for strings of the same length
     // Sort by character (which one will you start with?) using the modified counting sort.

    // Do NOT hardcode the number of characters in the String, like 3.
    // It should work on an array of strings of any length as long as it is significantly smaller than the number of strings).
    // Assume each string has the same length.



  //}

  public static void radixSortStrings(String[] strings) {
    int stringLength = strings[0].length();
```

```
  for (int i = stringLength - 1; i >= 0; i--) {
    countingSortForStrings(strings, i);
  }
}

public static void countingSortForStrings(String[] str, int position) {
  int[] counterArr = new int[500];
  String[] result = new String[str.length];
  for (String string : str) {
    char ch = string.charAt(position);
    counterArr[ch]++;
  }
  for (int j = 1; j < counterArr.length; j++) {
    counterArr[j] += counterArr[j - 1];
  }
  for (int i = str.length - 1; i >= 0; i--) {
    char ch = str[i].charAt(position);
    counterArr[ch]--;
    result[counterArr[ch]] = str[i];
  }
  System.arraycopy(result, 0, str, 0, str.length);
}


public static void main(String[] args) {
  // countingSortForCharacters is provided in the starter code
```

```
    char[] characters = {'g', 'b', 'a', 'c', 'e', 'm', 'a', 'p', 'a'};
    System.out.println(Arrays.toString(characters));
    countingSortForCharacters(characters);
    System.out.println(Arrays.toString(characters));

    // Call your radixSortStrings on the following array
    String[] strings = {"cat", "cot", "moo", "pet", "den", "dog", "pen", "ale", "red"};
    System.out.println(Arrays.toString(strings));
    radixSortStrings(strings);
    System.out.println(Arrays.toString(strings));
    // The expected result: [ale, cat, cot, den, dog, moo, pen, pet, red]

    // Test more on your own.

  }
}
```

Consider the following recursive function that computes the power of n (computes x^n) for a positive n:

```
public static long power(int x, int n) {
      if (n < 0)
         throw new IllegalArgumentException();
      if (n == 0)
         return 1;
      if (n % 2 == 0) {
         return power(x, n/2) * power(x, n/2);
```

```
    }
    else
      return x * power(x, n/2) * power(x, n/2);

  }
```

(a) (0.5pt) Give a recurrence relation for this function (give both base and recursive cases).

(b) (1.5pt) Solve the recurrence relation using either repeated substitution

or recursion tree to get Theta of the running time.

(a) (1pt) Show intermediate steps of sorting the subarray

from low = 2 to high = 5 (inclusive) with selection sort in DESCENDING order.

8 7 6 1 9 2 17 0

Show the array after each pass of selection sort (4 passes total).

Show the whole array even if some elements do not change

because they are outside the [low, high] range.

You can type your answer on Canvas or write on paper, or submit it via Github.

```
import java.util.Arrays;

// Change the given code so that it sorts the subarray from index low to index high
(inclusive)
// in descending order using selection sort.
```

```
//You may NOT do an ascending selection sort and then just reverse the array, such solutions will not get credit.
public class Question5b {

  public static void selectionSort(int[] arr, int low, int high) {
    // TODO: change the code so that it sorts
    // the subarray from index = low to index = high (inclusive)
    // in DESCENDING order.
    for (int i = low; i <= high; i++) {
      int ind = i;
      for (int j = i; j <= high; j++) {
        if (arr[j] > arr[ind]) {
          ind = j;
        }
      }
      int tmp = arr[ind];
      arr[ind] = arr[i];
      arr[i] = tmp;
    }
  }

  public static void main(String[] args) {
    int[] arr = {8, 7, 6, 1, 9, 2, 17, 0};
    selectionSort(arr, 2, 5);
    System.out.println(Arrays.toString(arr)); //expected 8, 7, 9, 6, 2, 1, 17, 0
```

}

}

Read the description of this question on Canvas.

Solve the following recurrence relation using the Master method.

T(0) = T(1) = C1

T(n) = 3T(n/3) + n^(1.5)

Explain your answer. What are the two functions you are comparing?

Specify which case applies; if applying case 1 or 3, give epsilon; in case 3, also give C.

Please refer to the description of this question on Canvas

where the instructor listed three cases of the Master theorem.

CS601 midterm 1

Consider the following statements about the design patterns covered in class. Select all that are true.

The Singleton pattern restricts instantiation to one object and ensures that other classes cannot access this instance.

Correct!

The Adapter pattern allows incompatible interfaces to work together by acting as a bridge between them.

In the MVC (Model-View-Controller) pattern, the View is responsible for calling methods on the model based on user's actions.

Correct!

The Observer pattern allows one-to-many relationships so that when one "subject" changes state, all its listeners are notified.

The Decorator pattern relies solely on inheritance to add new behaviors to children of the parent class.

Question 2

We want to be able to compare Actor objects based on the number of movies they have appeared in. If two actors have the same number of movies, they should be compared alphabetically by their names.

Your task is to implement a **Comparator** for the Actor class that performs this comparison. The required import statements are omitted for brevity.

```
public class Actor {
  private String name;
  private List<String> movies;

  public Actor(String name, List<String> movies) {
    this.name = name;
    this.movies = movies; // would be better to create a copy
  }

  public void addMovie(String movie) {
    movies.add(movie);
  }

  // Additional methods (getters, etc.) omitted for brevity
```

```
  // Assume there are methods getName(), getNumMovies()
}
```

TODO: Write a class called **ActorComparator** that implements the Comparator interface and compares two actors based on the number of movies; if the number of movies are the same, compare based on the names.

Question 3

Suppose you are given a **UserProfile** class that stores a list of messages written by a user (import statements not shown):

```
public class UserProfile {
  private List<String> messages = new ArrayList<>();

  public void addMessage(String message) {
    this.messages.add(message);
  }

  public int getNumMessages() {
    return messages.size();
  }
}
```

Write a method that takes a List of UserProfile objects and returns a Map (HashMap), where the key is the number of messages (an Integer), and the value is a list of UserProfile objects that have exactly this many messages. Assume this method is part of some class (no need to write a main method or include import statements).

```
public Map<Integer, List<UserProfile>> computeMap(List<UserProfile> profiles) {
  // TODO: return a HashMap that maps the number of messages to the list of UserProfile objects with that many messages

}
```

Example:

Suppose the profiles list contains 4 users: User1 has written 2 messages, User2 3 messages, User3 2 messages, and User4 has written 1 message.
The returned map should look like this:

{
  1 => [User4],
  2 => [User1, User3],
  3 => [User2]
}

Make sure your code is general, and can handle any list of profiles.

Question 4

Suppose there are two classes: the parent class ***Vehicle*** and its child class ***ElectricCar***.

The ***Vehicle*** class has a *non-static* method ***startEngine()*** that is overridden in ***ElectricCar***.

The ***ElectricCar*** class also has a method ***checkBattery()***, which is NOT present in the parent class *Vehicle*.

Consider the code snippet below.

```
Vehicle v = new ElectricCar();

v.startEngine();

ElectricCar e = (ElectricCar) v;

e.checkBattery();
```

- Will the startEngine method from Vehicle (parent) or ElectricCar (child) be invoked?
- Is it necessary to *downcast* v to ElectricCar in order to call checkBattery()?

Make sure both parts of the answer are correct when you select the answer.

The startEngine method from ElectricCar (child) will be invoked. Yes, downcasting is necessary to call checkBattery().

The startEngine method from Vehicle (parent) will be invoked. Yes, downcasting is necessary to call checkBattery().

The startEngine() method from Vehicle (parent) will be invoked. No, downcasting is not necessary to call checkBattery().

Vehicle v = new ElectricCar(); is NOT a valid statement, so the code will not compile.

Question 5

Consider the classes ***WaitingThreads*** and ***MyThread***. In what order will the threads finish when ***WaitingThreads*** is executed? (Note: You may NOT run the code.)

```
public class WaitingThreads {
  static class MyThread extends Thread {
    private String name;
    private Thread waitsFor;

    public MyThread(String name, Thread wa) {
      this.name = name;
      waitsFor = wa;
    }

    public void run() {
      try {
        Thread.sleep(1000);
        if (waitsFor != null) {
          waitsFor.join();
        }
        System.out.println(name + " finished");
      } catch (InterruptedException ie) {
        System.out.println(ie.getMessage());
      }
    }
  }
  public static void main (String [] args) {
    Thread t1 = new MyThread("1", null);
    Thread t2 = new MyThread("2", t1);
    Thread t3 = new MyThread("3", t2);
```

```
    t3.start(); t2.start(); t1.start();
  }
}
```

t3 will finish first, then t2, then t1.

t2 will finish first, then t1, then t3.

t3 will finish first, then t1, then t2.

Question 6

Consider the partial implementation of ***TextProcessor***, which counts the total number of words in a list of text strings. Import statements have been omitted.

```
public class TextProcessor {
  private int totalCount = 0;
  private final ExecutorService executor = Executors.newFixedThreadPool(4);

  class Worker implements Callable<Integer> {
    private final String text;
    public Worker(String text) {
      this.text = text;
    }
    public Integer call() {
      String[] words = text.split(" ");
      return words.length;
    }
  }

  public void count(List<String> texts) {
    List<Future<Integer>> futures = new ArrayList<>();
```

```
    for (String text : texts) {
      Future<Integer> future = executor.submit(new Worker(text));
      futures.add(future);
    }
    // TODO: get the results and update the totalCount
    // totalCount need to be updated in a synchronized block:
    // which lock will you use?

  }
  // other methods such as getCount, shutdownPool are not shown here; you do NOT need to write these

  public static void main(String[] args) {
    TextProcessor wordCounter = new TextProcessor();
    wordCounter.count(Arrays.asList("Java 17", "SpringBoot security"));
    // If we wanted to complete this example, we would need to get the totalCount, shutdown the pool etc - you do NOT need to add these.
  }
}
```

Answer the following questions/Fill in some missing code:

1) Will different ***Worker*** tasks be executed concurrently in this implementation? Explain your answer.

2) Fill in the missing code in the ***count*** method to get the results from each ***Worker*** and update the ***totalCount***. ***totalCount*** must be updated in a thread-safe manner as it is accessed by different workers. Copy your implementation of count() here.

Question 7

Suppose we want to use a **Builder** design pattern to implement a **Triangle** class that represents a triangle with three sides a, b, c. We need to ensure that the triangle is valid - the sum of any two sides is greater then the third: (a + b > c) && (a + c > b) && (b + c > a), otherwise, the code should throw an IllegalArgumentException. Fill in the missing code in the **build** method.

```
public class Triangle {
  private final double a, b, c; // three sides of the triangle

  private Triangle(TriangleBuilder builder) {
    this.a = builder.a;
    this.b = builder.b;
    this.c = builder.c;
  }

  public static class TriangleBuilder {
    private double a, b, c;

    public TriangleBuilder setA(double a) {
      this.a = a;
      return this;
    }

    public TriangleBuilder setB(double b) {
      this.b = b;
      return this;
    }

    public TriangleBuilder setC(double c) {
      this.c = c;
      return this;
    }

    public Triangle build() {
      // TODO: Implement this method

    }
  }
}
```

Question 8

This question builds on top of the previous one (class Triangle).

Fix bugs in the following statement that was supposed to create a Triangle with sides 5, 7, 9:

Triangle triangle = new Triangle() .setA(5) .setB(7) .setC(9);

Write the correct statement in the space below.

Correct Answers

Triangle triangle=new TriangleBuilder() .setA(5) .setB(7) .setC(9).build();

CS601 Midterm 2

Question 1

Consider the database tables *teams* and *players*.

The *teams* table stores information about sports teams, including their unique ID, team name, the city where the team is based, and the sport they play.

teams

The *players* table stores information about players, including their unique ID, player name, the team they belong to (if any), and their country of origin.

players

Write a SQL query to **count the number of players for each sport, and show sport and number of players only for those sports where there are at least 3 players.**

The expected result:

teams

team_id	team_name	city	sport
1	Tigers	Los Angeles	Baseball
2	Sharks	San Francisco	Soccer
3	Eagles	Chicago	Basketball
4	Lions	New York	Football
5	Bears	Miami	Baseball

The *players* table stores information about players, including their unique ID, player name, the team they belong to (if any), and their country of origin.

players

player_id	player_name	team_id	country
1	Alice Johnson	1	USA
2	Bob Smith	1	South Africa
3	Carol Martinez	2	Spain
4	João Santos	2	Brazil
5	Emily Wilson	3	Australia
6	Hans Müller	NULL	Germany
7	Grace Lee	NULL	Turkey
8	Wei Zhang	2	China
9	Arjun Patel	3	India

Write a SQL query to **count the number of players for each sport, and show sport and number of players only for those sports where there are at least 3 players.**

The expected result:

sport	player_count
Soccer	3

Why were other sports not included? Only 2 players play basketball, and only 2 players play baseball according to the tables, and no players currently play football.

Make your query general, do not hardcode specific sports or player /team ids etc.

Question 2

Consider tables teams and players again.

teams

players

teams

team_id	team_name	city	sport
1	Tigers	Los Angeles	Baseball
2	Sharks	San Francisco	Soccer
3	Eagles	Chicago	Basketball
4	Lions	New York	Football
5	Bears	Miami	Baseball

players

player_id	player_name	team_id	country
1	Alice Johnson	1	USA
2	Bob Smith	1	South Africa
3	Carol Martinez	2	Spain
4	João Santos	2	Brazil
5	Emily Wilson	3	Australia
6	Hans Müller	NULL	Germany
7	Grace Lee	NULL	Turkey
8	Wei Zhang	2	China
9	Arjun Patel	3	India

Write a SQL query to **display the names of teams and their player names. Teams without players should also be included in the result (and show NULL instead of player names). Order the results alphabetically by team name.**

Question 3

Suppose the input to your program is a string containing email addresses and suppose we want to group these email addresses by their TLD - Top-Level Domain (such as .com, .edu, .org). Suppose we create a map that for each top-level domain stores a list of all valid emails with the this top-level domain.

Sample input: "jlee@gmail.com, priyaK@siggraph.org, admin@usfca.edu, support@AnitaB.org, questions@bbc.com"

Expected output:

{

com=[jlee@gmail.com, questions@bbc.com],

org=[priyaK@siggraph.org, support@AnitaB.org],

edu=[admin@usfca.edu]

}

```
public class EmailTLDGrouper {
   public static void main(String[] args) {
      String input = "jlee@gmail.com, priyaK@siggraph.org, admin@usfca.edu,
support@AnitaB.org, questions@bbc.com";
       Map<String, List<String>> map = new HashMap<>(); // a map that will map each top-
level-domain to the list of emails

      // FILL IN CODE:
      Pattern pattern = Pattern.compile(""); // write regex to extract an email and a top-level
domain
      Matcher matcher; // create a matcher
      // FILL IN CODE: repeatedly apply the matcher to the input and fill the map
      // the key in the map is top-level domain, and the value the list of emails with this top-
level domain

      // Print the resulting map
      System.out.println(map);
  }
```

}

Import statements are omitted. For the purpose of this question **a valid email** must include:

- at least one letter or number before @
- @ symbol
- at least one letter or number after @ and before .
- a domain name ending with a valid TLD (e.g., .com, .org, .edu, .uk).
  The TLD is the part after the last . in the domain name and should be at least 2 characters long (e.g. com, edu, org). Do NOT hard code specific TLDs, your code must be general enough to work with another input in the same format.

You must use Pattern, Matcher and use group(s) to solve the problem. Copy/paste the code in the space below and edit. You may not run the code. Your code must be able to handle more emails in a string than shown above.

Question 4

Suppose in the Jetty server we mapped "/jokes" to a **JokeServlet** that allows users to send a joke to the server as a query parameter of the GET request.

The servlet should show the current joke sent as a query parameter and a previous joke saved in a cookie.

For instance, if the user first sends this request:

http://localhost:8082/jokes?joke=What do you call fake spaghetti?An impasta

the user will see their joke in the browser window:

Today's joke: What do you call fake spaghetti?An impasta

Since it is the first request, no previous joke was printed.

When the user sends the next request:

http://localhost:8082/jokes?joke=I am on a seafood diet. I see food I eat it.

The user should now see in the browser:

Today's joke: I am on a seafood diet. I see food I eat it.

Previous joke: What do you call fake spaghetti? An impasta.

The previous joke in the cookie would now be updated to the current joke.

```
public class JokeServlet extends HttpServlet {
  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException {
    PrintWriter pw = response.getWriter();
    // FILL IN CODE:
    // Show the previous joke if it exists in the cookie
    // Get the new joke from the query parameter "joke" sent by the client
    // Write it to the response
    // Save it in the cookie
    // (Handle the case when it's null)

    // Note: cookies do not allow white spaces, but you can ignore this issue for this
problem
    // (if we were to do it properly, we could replace spaces with %20 when we store the
cookie)

  }
}
```

Copy/paste the starter code in the space below and edit it/fill in the missing code. You may not run the code. Do not worry about formatting HTML - you may add plain text to the response as following: pw.println("Some text");

Import statements have been omitted to save space.

Question 5

Suppose we have a **train station** where a train departs only when there are exactly 3 passengers ("full" train). Suppose there
is *one **train** thread* and *multiple **passenger** threads*.

Each "passenger" can "board" the train, and the train thread "departs" once 3 passengers are on board. The same train thread can imitate multiple train "departures". **Fix the bugs** and **fill in the missing code** in the boardTrain and departTrain methods.

Use wait and notifyAll to synchronize between passengers and the train, and please make sure you **synchronize access** to shared data.

Here is a sample interaction between passengers and train:

Train: Not enough passengers, waiting for more
Passenger: I boarded. The number of boarded passengers is: 1
Passenger: I boarded. The number of boarded passengers is: 2
Passenger: I boarded. The number of boarded passengers is: 3
Train: departing with 3 passengers.
Passenger: I boarded. The number of boarded passengers is: 1
Passenger: I boarded. The number of boarded passengers is: 2
Passenger: I boarded. The number of boarded passengers is: 3
Passenger: oops, the train is full, I need to wait for next one

Train: departing with 3 passengers.

And so on. Next train should wait for 2 more passengers before departing.

Copy/paste the starter code below and edit it. You may **not** run the code.

```
public class TrainStation {
  private int numPassengers = 0; //the number of passengers who already boarded the train
  private final int MAX_PASSENGERS = 3; // How many passengers should board the train
before it departs

  public void boardTrain() throws InterruptedException {
      while (numPassengers == MAX_PASSENGERS) {
        System.out.println("Passenger: oops, the train is full, I need to wait for next one.");
        notifyAll("The train is full!);
      }
      numPassengers++;
      System.out.println("Passenger: I boarded. The number of boarded passengers is: " +
numPassengers);
      if (numPassengers == MAX_PASSENGERS) {
        notifyAll(); // The train can depart
      }
  }

  public void departTrain() throws InterruptedException {
      while (numPassengers < MAX_PASSENGERS) {
```

```
            System.out.println("Train: Not enough passengers, waiting for more");
            wait();

        }
        System.out.println("Train: departing with " + numPassengers + " passengers.");
        numPassengers = 0; // Reset passenger count for the next train

    }
```

Question 6

(a) Write an HTTP GET request that sends a request to weather.com and sends the following query parameters latitude = 37 and longitude = 122.

(b) Write an HTTP POST request to weather.com that sends the same parameters to the server (how/where will you send them?).

Do not write Java code, this is a question on HTTP protocol. Do not worry about adding Content length to the header of the request for this question. This question mostly focuses on how the parameters are passed in GET and POST.