Nathaniel Burt

CS 499

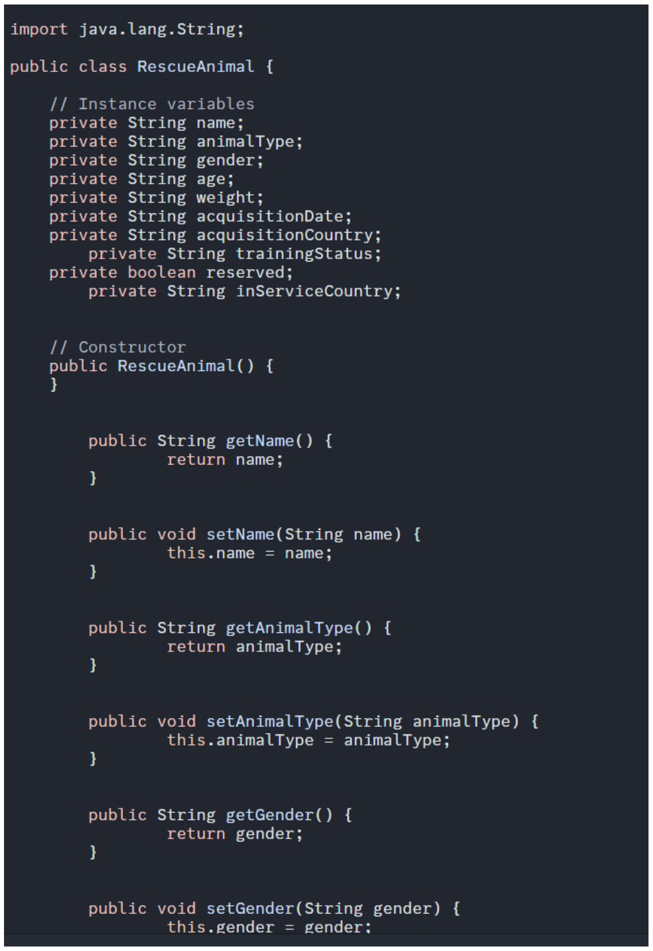
6/1/2025

Data Structures and Algorithms Enhancement Summary

Artifact Overview

The artifact is based on an earlier Java class called RescueAnimal.java, originally created as part of a foundational programming course back in June 3, 2023. It was a simple data model representing rescue animals, designed using basic object-oriented programming concepts. In May 2025, I significantly enhanced this artifact by converting it into a backend microservice using Node.js and Express. This new version focuses on implementing and demonstrating key algorithms and data structures within a RESTful API.

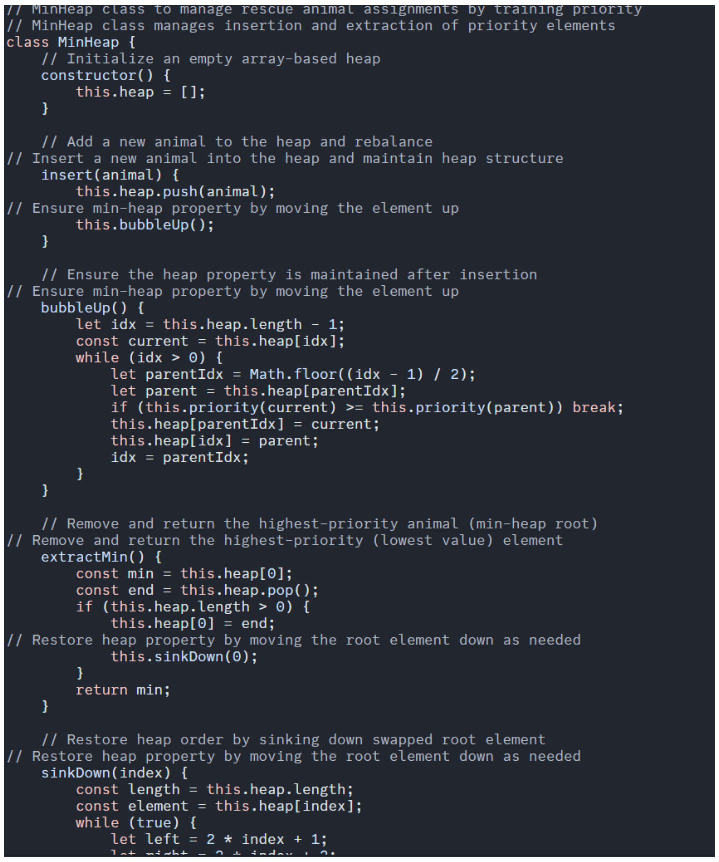
Here is a portion of the original RescueAnimal.java:



```
import java.lang.String;

public class RescueAnimal {

    // Instance variables
    private String name;
    private String animalType;
    private String gender;
    private String age;
    private String weight;
    private String acquisitionDate;
    private String acquisitionCountry;
        private String trainingStatus;
    private boolean reserved;
        private String inServiceCountry;


    // Constructor
    public RescueAnimal() {
    }


        public String getName() {
                return name;
        }


        public void setName(String name) {
                this.name = name;
        }


        public String getAnimalType() {
                return animalType;
        }


        public void setAnimalType(String animalType) {
                this.animalType = animalType;
        }


        public String getGender() {
                return gender;
        }


        public void setGender(String gender) {
                this.gender = gender;
```

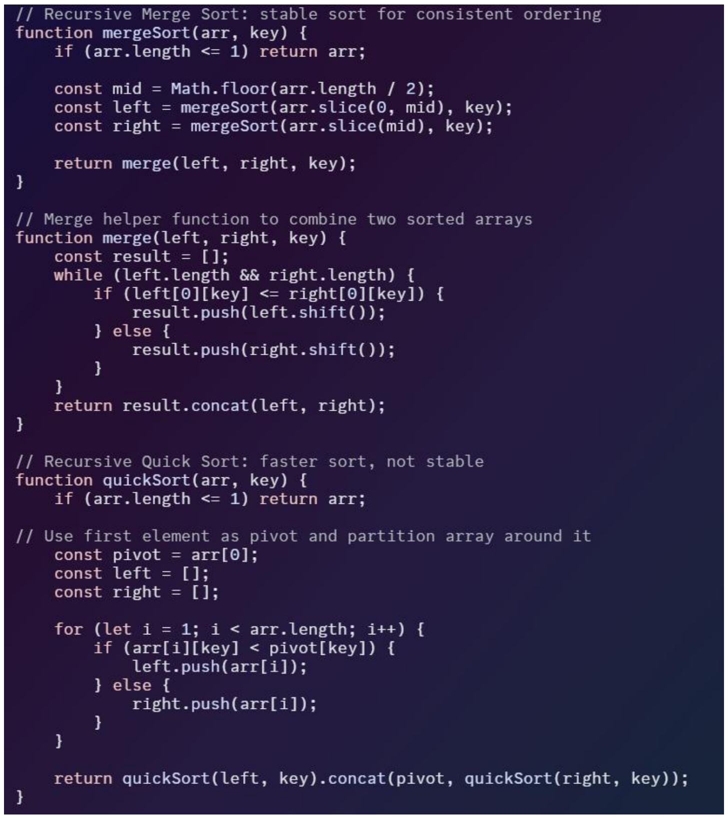
Here are some of the data structure inclusions:

MinHeap class to manage rescue animal assignments by training priority.

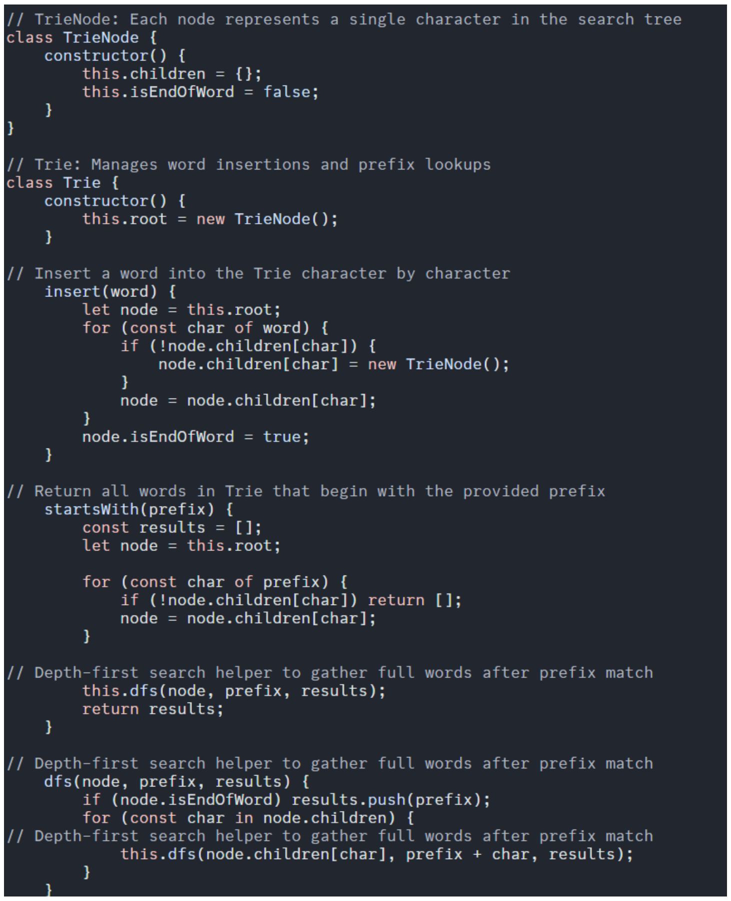


```
// MinHeap class to manage rescue animal assignments by training priority
// MinHeap class manages insertion and extraction of priority elements
class MinHeap {
    // Initialize an empty array-based heap
    constructor() {
        this.heap = [];
    }

    // Add a new animal to the heap and rebalance
// Insert a new animal into the heap and maintain heap structure
    insert(animal) {
        this.heap.push(animal);
// Ensure min-heap property by moving the element up
        this.bubbleUp();
    }

    // Ensure the heap property is maintained after insertion
// Ensure min-heap property by moving the element up
    bubbleUp() {
        let idx = this.heap.length - 1;
        const current = this.heap[idx];
        while (idx > 0) {
            let parentIdx = Math.floor((idx - 1) / 2);
            let parent = this.heap[parentIdx];
            if (this.priority(current) >= this.priority(parent)) break;
            this.heap[parentIdx] = current;
            this.heap[idx] = parent;
            idx = parentIdx;
        }
    }

    // Remove and return the highest-priority animal (min-heap root)
// Remove and return the highest-priority (lowest value) element
    extractMin() {
        const min = this.heap[0];
        const end = this.heap.pop();
        if (this.heap.length > 0) {
            this.heap[0] = end;
// Restore heap property by moving the root element down as needed
            this.sinkDown(0);
        }
        return min;
    }

    // Restore heap order by sinking down swapped root element
// Restore heap property by moving the root element down as needed
    sinkDown(index) {
        const length = this.heap.length;
        const element = this.heap[index];
        while (true) {
            let left = 2 * index + 1;
            let right = 2 * index + 2;
```

Recursive Merge Sort for consistent ordering.


```
// Recursive Merge Sort: stable sort for consistent ordering
function mergeSort(arr, key) {
    if (arr.length <= 1) return arr;

    const mid = Math.floor(arr.length / 2);
    const left = mergeSort(arr.slice(0, mid), key);
    const right = mergeSort(arr.slice(mid), key);

    return merge(left, right, key);
}

// Merge helper function to combine two sorted arrays
function merge(left, right, key) {
    const result = [];
    while (left.length && right.length) {
        if (left[0][key] <= right[0][key]) {
            result.push(left.shift());
        } else {
            result.push(right.shift());
        }
    }
    return result.concat(left, right);
}

// Recursive Quick Sort: faster sort, not stable
function quickSort(arr, key) {
    if (arr.length <= 1) return arr;

// Use first element as pivot and partition array around it
    const pivot = arr[0];
    const left = [];
    const right = [];

    for (let i = 1; i < arr.length; i++) {
        if (arr[i][key] < pivot[key]) {
            left.push(arr[i]);
        } else {
            right.push(arr[i]);
        }
    }

    return quickSort(left, key).concat(pivot, quickSort(right, key));
}
```

Trie Node used for prefix based search on animal names.



```
// TrieNode: Each node represents a single character in the search tree
class TrieNode {
    constructor() {
        this.children = {};
        this.isEndOfWord = false;
    }
}

// Trie: Manages word insertions and prefix lookups
class Trie {
    constructor() {
        this.root = new TrieNode();
    }

// Insert a word into the Trie character by character
    insert(word) {
        let node = this.root;
        for (const char of word) {
            if (!node.children[char]) {
                node.children[char] = new TrieNode();
            }
            node = node.children[char];
        }
        node.isEndOfWord = true;
    }

// Return all words in Trie that begin with the provided prefix
    startsWith(prefix) {
        const results = [];
        let node = this.root;

        for (const char of prefix) {
            if (!node.children[char]) return [];
            node = node.children[char];
        }

// Depth-first search helper to gather full words after prefix match
        this.dfs(node, prefix, results);
        return results;
    }

// Depth-first search helper to gather full words after prefix match
    dfs(node, prefix, results) {
        if (node.isEndOfWord) results.push(prefix);
        for (const char in node.children) {
// Depth-first search helper to gather full words after prefix match
            this.dfs(node.children[char], prefix + char, results);
        }
    }
```

Why This Artifact Was Chosen

I included this artifact in my ePortfolio because it demonstrates my ability to apply data structure and algorithmic concepts in a practical development setting. The enhanced version features a Trie for fast prefix-based searching, merge sort and quicksort for sorting animals by attributes like age and weight, and a min-heap (priority queue) for task assignment based on training status. These additions reflect advanced proficiency in designing and implementing efficient algorithms and structures from scratch, and show my ability to integrate them into a working API. These improvements also highlight my understanding of runtime complexity, stability in sorting, and modular code design.

Course Outcomes Alignment

I met all the course outcomes I originally planned for this enhancement. Specifically, I demonstrated my ability to apply algorithmic problem-solving techniques, select and implement appropriate data structures, and analyze trade-offs between competing solutions. The enhancement remains aligned with my outcome-coverage plan, and no updates are currently necessary.

Reflection on the Enhancement Process

Through this enhancement, I deepened my understanding of how algorithms work under real-world constraints, especially when integrated into a backend API. Implementing the Trie and priority queue from scratch helped me refresh and apply concepts I had previously only studied in theory. One challenge was ensuring that the added complexity of these data structures did not interfere with the clarity and modularity of the backend service. Managing performance, code readability, and maintainability simultaneously was an important learning experience. Overall, this process strengthened both my technical skills and my confidence in designing scalable backend solutions.