Criterion C

```java
/*
 * This method parses a CSV file, a format which is necessary for long-term data storage      You, now • Uncommitted changes
 */
public static void loadMessages() {
    try(Scanner in = new Scanner(new File(pathname:"blog\\src\\main\\resources\\data\\messages.csv"))) {
        while(in.hasNextLine()){
            String line = in.nextLine();
            String fields[] = line.split(regex:",");
            messages.add(new Message(Long.parseLong(fields[0]), Long.parseLong(fields[1]), fields[2], fields[3]));
```

This method is a part of my Utilities class. It handles loading the database of messages, as a persistent, yet universal form of data storage is a must.

```java
package com.InternalAssessment.blog;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.view.RedirectView;

import com.InternalAssessment.blog.Messages.Message;

import java.util.List;
You, 1 second ago | 1 author (You)
/**
 * This is the controller for SpringBoot, the external library used to display my project
 */
@Controller
public class BlogController {
    @GetMapping("/")
```

This snippet shows the required libraries that I must use to utilize SpringBoot with Thymeleaf. This library allows me to host and run the application

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.4.3)

2025-04-05T19:33:43.943-06:00  INFO 18316 --- [blog] [           main] c.I.blog.BlogApplication               : Starting BlogApplication using Java 21.0.5 with PID 18316 (G:\School\High School\2024-2025\Comp
ci IA\24-25CompSciInternalAssessment\24-25CompSciInternalAssessment\blog\target\classes started by liual in G:\School\High School\2024-2025\Comp Sci\Comp Sci IA\24-25CompSciInternalAssessment\24-25CompSciIntern
nt)
2025-04-05T19:33:43.943-06:00  INFO 18316 --- [blog] [           main] c.I.blog.BlogApplication               : No active profile set, falling back to 1 default profile: "default"
2025-04-05T19:33:44.468-06:00  WARN 18316 --- [blog] [           main] trationDelegate$BeanPostProcessorChecker : Bean 'org.springframework.ws.config.annotation.DelegatingWsConfiguration' of type [org.springfra
config.annotation.DelegatingWsConfiguration$$SpringCGLIB$$0] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying). The currently created BeanPostProcessc
ionActionEndpointMapping] is declared through a non-static factory method on that class; consider declaring it as static instead.
2025-04-05T19:33:44.490-06:00  INFO 18316 --- [blog] [           main] .w.s.a.s.AnnotationActionEndpointMapping : Supporting [WS-Addressing August 2004, WS-Addressing 1.0]
2025-04-05T19:33:44.683-06:00  INFO 18316 --- [blog] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2025-04-05T19:33:44.688-06:00  INFO 18316 --- [blog] [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2025-04-05T19:33:44.688-06:00  INFO 18316 --- [blog] [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.36]
2025-04-05T19:33:44.735-06:00  INFO 18316 --- [blog] [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2025-04-05T19:33:44.736-06:00  INFO 18316 --- [blog] [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 762 ms
2025-04-05T19:33:44.844-06:00  INFO 18316 --- [blog] [           main] o.s.b.a.w.s.WelcomePageHandlerMapping    : Adding welcome page template: index
2025-04-05T19:33:45.173-06:00  INFO 18316 --- [blog] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path '/'
2025-04-05T19:33:45.177-06:00  INFO 18316 --- [blog] [           main] c.I.blog.BlogApplication               : Started BlogApplication in 1.507 seconds (process running for 1.742)
2025-04-05T19:33:45.626-06:00  INFO 18316 --- [blog] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-04-05T19:33:45.626-06:00  INFO 18316 --- [blog] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2025-04-05T19:33:45.626-06:00  INFO 18316 --- [blog] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 0 ms
0000001743903225657?f?0000000000000000000?f?Fourth Message!!?f?Actually, this is the fifth message
```

This is a sample console log of the code running, along with test messages being printed to the console. Note that, although the data is being stored to a CSV, it uses a non-standard

delimitator, to ensure that the user can type commas.

```java
package com.InternalAssessment.blog;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

import com.InternalAssessment.blog.Messages.*;
You, 21 hours ago | 1 author (You)
/**
 * Creates a tree of messages present. Note that, under nominal circumstances, a parent will never come after its child while parsing the database. This fact handles errors associated with building the tree.
 * In addition, the tree is only edited when going through the program. Therefore, all error checking is handled much before any data reaches this tree, and as such, is very robust and not prone to error.
 */
public class MessageTreeNode {
    private Message message;
    //Uses a list to manage the children of a node. This list is dynamic and features error handling if the child being added or looked up is null
    private ArrayList<MessageTreeNode> children =new ArrayList<>();
    public MessageTreeNode(Message message){
        this.message = message;
    }
    /**
     * Uses Breadth First Search to look through the children of a node to find a matching ID
     * @param id
     * @return
     */
    public MessageTreeNode findMessageBFS(long id){
        //Uses a dynamically implemented linkedlist as a queue, to search through the children. Such a method is one of the fastest ways of searching through such a tree/
        Queue<MessageTreeNode> mainQueue = new LinkedList<>();
        mainQueue.add(this);
        while(mainQueue.peek() != null){
            MessageTreeNode curNode = mainQueue.poll();
            if(curNode.getMessage().getId() == id){
                return curNode;
            }
            for(MessageTreeNode childNode : curNode.children){
                mainQueue.add(childNode);
            }
        }
        return null;
    }
    public MessageTreeNode findMessageDFS(long id){
        //Implements Depth First Search to locate a message. This is optimal if the user knows that the intended message is far away from the root node
        Stack<MessageTreeNode> mainQueue = new Stack<>();
        mainQueue.add(this);
        while(mainQueue.peek() != null){
            MessageTreeNode curNode = mainQueue.pop();
            if(curNode.getMessage().getId() == id){
                return curNode;
            }
            for(MessageTreeNode childNode : curNode.children){
                mainQueue.push(childNode);
            }
        }
        return null;
    }
    public MessageTreeNode findMessageRecursive(long id, MessageTreeNode curMessage){
        //Uses Depth First Search to recursively locate the message.
        if(curMessage.getMessage().getId() == id){
            return curMessage;
        }
        else {
            for(var i : curMessage.getChildren()){
                return findMessageRecursive(id, i);
            }
        }
        return null;
    }
    public void addNode(Message otherMessage){
        MessageTreeNode parent = findMessageBFS(otherMessage.getParent());
        parent.addChild(new MessageTreeNode(otherMessage));
    }
    //Note that this function is private. This use of encapsulation makes it so that only the class can utilize this method, reducing the possiblity of errors. This method has a rather generic name, which makes this encapsulation even more necessary.
    private void addChild(MessageTreeNode child){
        children.add(child);
    }
    public Message getMessage(){
        return message;
    }
    public ArrayList<MessageTreeNode> getChildren(){
        return children;
    }
}
```

This is an implementation of a tree. It features the message, along with storing a reference to each of the message's children. In it, key features such as the message and the children are marked as private – using encapsulation to prevent other classes from directly manipulating the tree itself and potentially causing errors. It features an ArrayList to manage children, and proper methods for manipulating the ArrayList

In addition, the tree allows for the merging of two trees, by matching a parent and adding a 'subtree' to the parent tree.

```
0000000000000000000–ƒ–0000000000000000000–ƒ–This is the top index–ƒ–Please do not delete this!
0000001743795346949–ƒ–0000000000000000000–ƒ–First post–ƒ–First!
0000001743831827621–ƒ–0000000000000000000–ƒ–Second!–ƒ–Yup.
0000001743834787134–ƒ–0000000000000000000–ƒ–Third Message–ƒ–Test. Lorem Impsum
0000001743835070059–ƒ–0000001743834787134–ƒ–Fourth Message–ƒ–This is a new message!
0000001743835146426–ƒ–0000001743835070059–ƒ–Fifth Message–ƒ–New message!
0000001743887463872–ƒ–0000000000000000000–ƒ–Fourth Message–ƒ–New message!
0000001743903225657–ƒ–0000000000000000000–ƒ–Fourth Message!!–ƒ–Actually, this is the fifth message
```

This is a snippet from the data CSV itself. Note the use of the Em-Dashes and the stylized ‘f’, which is a combination rare enough that an end user would not be able to type it in

```java
/**
 * This class creates a more specific type of message, used for the top-level message. As such, it allows for more flexibility and better readability
 */
public class TopMessage extends Message{
    public TopMessage(long id, String title, String content){
        super(id, parent:0, title, content);
    }
    public void forceMessage(){
        this.setTitle("TOP: " + this.getTitle());
    }
```

This uses polymorphism and inheritance to provide more specific behavior to special messages; in this case, the top message.

```java
package com.InternalAssessment.blog.Messages;
import com.InternalAssessment.blog.Util;
You, 10 seconds ago | 1 author (You)
/**
 * Implements a message framework. This serves as a hierarchical composite data structure by allowing multiple values to be stored inside a single class. The parent, in this case, is stored as the ID of the parent. This makes the record easier to understand without loss of data._
 */
public class Message {
    protected long id;
    protected long parent;
    protected String title;
    protected String content;
    public Message(long id, long parent, String title, String content) {
        this.id = id;
        this.parent = parent;
        this.title = title;
        this.content = content;
    }
    public Message() {
        //TODO Auto-generated constructor stub
    }
    public Message getMessage(){
        return this;
    }
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content = content;
    }
    public String toCsv(){
        return String.format(format:"%s,%s,%s,%s", Util.padId(id), Util.padId(parent), title, content);
    }
    public String getDate(){
        return Util.getDateTime(id);
    }
    public long getParent() {
        return parent;
    }
    public void setParent(long parent) {
        this.parent = parent;
    }
```

This is an implementation of a hierarchical composite data structure. Note that the Parent ID functions as a message unto itself, as error checking and a robust searching system allow the program to easily match the ID with the actual message. Such this fail to suffice,

the TreeNode class functions as a hierarchical composite data structure.

```
public MessageTreeNode findMessageRecursive(long id, MessageTreeNode curMessage){
    //Uses Depth First Search to recursively locate the message.
    if(curMessage.getMessage().getId() == id){
        return curMessage;
    }
    else {
        for(var i : curMessage.getChildren()){
            var temp = findMessageRecursive(id, i);
            if(temp != null){
                return temp;
            }
        }
    }
    return null;
}
```

This is an example of using recursion to search using Depth First Search, which provides for an easy-to-understand and fast way of searching through the tree.