Coursework Specification: Real-Time Event Ticketing System with Advanced Producer-Consumer Implementation

Additional Encouragement:
Please inform students that they are encouraged to utilize LinkedIn Learning courses to supplement their learning. Completing successfully courses related to the project's implementation, such as Object-Oriented Programming (OOP), Object-Oriented Design (OOD), Core Java, Angular, Node.js, or Spring Boot, can provide them with additional insights and skills. Moreover, they can earn up to 3 bonus points per course by successfully completing relevant LinkedIn Learning courses.

Can I use ChatGPT for this coursework?
Answer:
While tools like ChatGPT can be helpful for clarifying concepts and providing guidance, it's important to ensure that all work submitted for this coursework is your own original creation. This assignment is designed to assess your individual understanding and proficiency in object-oriented programming, multi-threading, and system design.
Guidelines for Using ChatGPT:

1. **Educational Use:**
    - **Concept Clarification:** Feel free to use ChatGPT to help understand programming concepts, algorithms, or design patterns relevant to the coursework.
    - **Problem-Solving Strategies:** You can seek general advice on how to approach a problem or structure your project.
2. **Avoiding Direct Solutions:**
    - **No Copy-Pasting Code:** Do not snippets or solutions provided by ChatGPT directly into your project.
    - **Original Implementation:** Ensure that all code, documentation, and diagrams are created by you, reflecting your own understanding.
3. **Academic Integrity:**
    - **Plagiarism Policies:** Adhere to your institution's policies on academic honesty and plagiarism.
    - **Proper Attribution:** If you incorporate ideas inspired by external sources, including AI tools, acknowledge them appropriately if required by your institution.
4. **Developing Understanding:**
    - **Deep Comprehension:** Use the assistance to enhance your understanding so you can explain and defend your work confidently.
    - **Skill Development:** Focus on building your skills in programming, problem-solving, and system design, which are the core objectives of the coursework.

Recommendations:

- **Consult Your Instructor:**
    - If you're unsure about the extent to which you can use AI tools, it's best to discuss this with your instructor or refer to the course guidelines.
- **Use Official Resources:**
    - Rely on textbooks, official documentation, and course materials as primary resources.
- **Practice Coding Independently:**
    - Write code on your own to strengthen your proficiency and identify areas where you may need further study.

Conclusion:
Using ChatGPT responsibly can aid your learning process, but the final submission should be a product of your own efforts and understanding. This approach will not only comply with academic standards but also prepare you effectively for future professional opportunities where independent problem-solving skills are crucial.

Purpose of the Coursework and Its Impact on Career Prospects

Why This Coursework Is Being Given

This coursework is designed to provide a comprehensive, real-world simulation of complex software development tasks that are highly relevant in today's technology industry. The primary objectives are:

1. **Deepen Technical Expertise:**
    - **Advanced Object-Oriented Programming (OOP):** By engaging with intricate OOP concepts such as inheritance, polymorphism, encapsulation, and design patterns, you solidify your understanding of how to build modular, scalable, and maintainable software systems.
    - **Multi-threading and Concurrency:** Implementing a system that handles multiple threads concurrently exposes you to the challenges of synchronization, race conditions, and deadlocks, which are critical considerations in high-performance applications.
    - **Producer-Consumer Pattern Mastery:** Applying this pattern in a complex environment enhances your ability to manage resources efficiently and design systems that can handle real-time data processing.
2. **Practical Application of Modern Technologies:**
    - **Frontend Frameworks:** Working with JavaFX, React.js, or Angular allows you to gain hands-on experience with tools that are widely used in the industry for building robust user interfaces.
    - **Backend Development:** Utilizing Java, Node.js, or Spring Boot helps you understand backend services, RESTful APIs, and server-side logic essential for full-stack development.
3. **Simulation of Real-World Challenges:**
    - **System Design and Architecture:** Crafting a complex system from scratch requires careful planning, architectural design, and understanding of software engineering principles.
    - **Problem-Solving Skills:** Tackling issues like deadlock detection, concurrency control, and performance optimization sharpens your analytical and critical thinking abilities.
    - **Project Management:** Managing such a comprehensive project enhances your ability to plan, organize, and execute tasks effectively.
4. **Preparation for Professional Environments:**
    - **Code Quality and Best Practices:** Emphasizing clean, readable, and maintainable code prepares you for professional coding standards.
    - **Documentation and Communication:** Creating detailed documentation and diagrams improves your ability to communicate technical information clearly, a vital skill in any tech role.
5. **Exposure to Advanced Concepts:**
    - **Integration of Advanced Features:** Optional functionalities like cloud integration, real-time analytics, and automated testing expose you to cutting-edge technologies and practices.

How Excelling in This Coursework Will Help Secure Placements

Successfully completing this coursework can significantly enhance your prospects when seeking internships, job placements, or advanced academic opportunities. Here's how:

1. **Demonstration of Technical Proficiency:**
    - **Portfolio Enhancement:** A complex, functioning project showcases your ability to apply theoretical knowledge to practical problems, making your portfolio stand out.
    - **Mastery of In-Demand Skills:** Proficiency in multi-threading, concurrency, and modern frameworks is highly sought after in the tech industry.
    - **Advanced Problem-Solving:** Overcoming challenges in synchronization and deadlock resolution demonstrates your capacity to handle complex technical issues.
2. **Alignment with Industry Requirements:**
    - **Relevance to Real-World Applications:** The system you build mirrors actual industry projects, giving you experience that's directly applicable to professional roles.
    - **Understanding of Best Practices:** Adhering to coding standards, design patterns, and development methodologies shows that you're prepared for professional software development environments.
3. **Competitive Advantage in Job Applications:**
    - **Discussion Material for Interviews:** The project provides concrete examples to discuss during technical interviews, allowing you to illustrate your skills and experiences vividly.
    - **Evidence of Self-Motivation:** Taking initiative to implement advanced features signals a proactive attitude and a commitment to excellence.
4. **Development of Soft Skills:**
    - **Communication Abilities:** Documenting your work and explaining complex concepts clearly enhances your communication skills, which are essential in collaborative work environments.
    - **Time Management and Organization:** Completing a project of this scope requires effective planning and time management, skills that are highly valued by employers.
5. **Networking Opportunities:**
    - **Academic Recommendations:** Instructors and mentors who observe your dedication and skill may become valuable references or connect you with industry professionals.
    - **Peer Recognition:** Demonstrating high-quality work can earn the respect of your peers, leading to collaborative opportunities and professional networking.
6. **Adaptability and Continuous Learning:**
    - **Learning New Technologies:** Engaging with unfamiliar tools or frameworks shows that you're adaptable and eager to learn—traits that employers prize.
    - **Staying Current with Industry Trends:** Implementing features like cloud integration or real-time analytics demonstrates that you're up-to-date with the latest technological advancements.

7. **Problem-Solving Under Constraints:**
    - **Resourcefulness:** Finding solutions within the specified constraints (e.g., manual GUI coding without drag-and-drop tools) shows creativity and resourcefulness.
    - **Attention to Detail:** Meticulously handling synchronization and concurrency issues highlights your ability to focus on critical details.

Impact on Securing Placements

Employers and placement coordinators often look for candidates who not only have strong theoretical knowledge but also practical experience in solving real-world problems. Excelling in this coursework can:

1. **Enhance Your Resume:**
    - **Project Highlights:** Including this project in your resume adds significant weight, showing tangible evidence of your capabilities.
    - **Skills Listing:** You can confidently list skills like multi-threading, concurrency control, RESTful API development, and more.
2. **Provide Talking Points During Interviews:**
    - **Technical Discussions:** You can discuss the challenges you faced and how you overcame them, demonstrating your problem-solving process.
    - **Behavioral Insights:** Sharing experiences from the project can illustrate your work ethic, perseverance, and ability to learn from failures.
3. **Showcase Initiative and Drive:**
    - **Going Beyond Requirements:** Implementing advanced features indicates that you're willing to exceed expectations—a quality that employers highly value.
    - **Continuous Improvement:** Reflecting on what you could have done better shows self-awareness and a commitment to personal growth.
4. **Facilitate Strong Recommendations:**
    - **Academic Endorsements:** Professors who witness your exceptional work are more likely to provide strong letters of recommendation.
    - **Professional References:** If your project gains attention, industry professionals may take notice and offer mentorship or job opportunities.
5. **Demonstrate Industry-Relevant Experience:**
    - **Full-Stack Development:** Experience with both frontend and backend technologies makes you a versatile candidate.
    - **Understanding of Software Lifecycles:** Managing the project from conception to deployment mirrors the processes used in professional settings.

Conclusion

This coursework is more than an academic assignment; it's an opportunity to bridge the gap between education and industry. By fully engaging with the project and striving for excellence, you:

- **Equip Yourself with Essential Skills:** Gain hands-on experience with tools and concepts that are directly applicable to many technology roles.
- **Stand Out to Employers:** Differentiate yourself in a competitive job market by showcasing a robust, complex project.

- **Prepare for Real-World Challenges:** Build confidence in your ability to tackle complex problems, work under constraints, and deliver high-quality results.

In essence, excelling in this coursework can significantly enhance your employability and open doors to exciting career opportunities. It demonstrates not only your technical capabilities but also your dedication, creativity, and readiness to contribute meaningfully in a professional environment.

Elaborated Guide to Implementing the Real-Time Event Ticketing System with Producer-Consumer Pattern

Introduction

This guide aims to provide a detailed explanation of how to implement each component of the **Real-Time Event Ticketing System** as per the coursework specification. The focus is on utilizing Object-Oriented Programming (OOP) principles and the Producer-Consumer pattern to simulate a dynamic ticketing environment. The system should handle concurrent ticket releases and purchases while maintaining data integrity.

---

1. Understanding the Project Requirements

Before diving into the implementation, ensure you thoroughly understand the project requirements:

- **Objective:** Create a real-time ticketing system that handles concurrent ticket releases (by vendors) and purchases (by customers) using multi-threading and synchronization.
- **Must-Haves:**
  - Concurrency handling with multiple producers and consumers.
  - Data integrity in a multi-threaded environment.
  - Basic reporting features.
- **Technology Stack:**
  - **Frontend:** JavaFX, React.js, or Angular.
  - **Backend:** Java, Node.js, or Spring Boot.
- **Restrictions:**
  - All GUI components must be manually coded—no drag-and-drop tools or code generators.
  - Focus on core concepts suitable for second-year undergraduate students.

---

2. Choosing the Technology Stack

Frontend Choices:

- **JavaFX:** Suitable if you prefer Java and want to create desktop applications with a GUI.
- **React.js or Angular:** Ideal for web-based applications. Use if you are comfortable with JavaScript/TypeScript and web development.

Backend Choices:

- **Java:** Good for an all-Java solution, especially if using JavaFX for the frontend.
- **Node.js:** Use if you prefer JavaScript on the backend, which pairs well with React.js or Angular.
- **Spring Boot:** If you have experience with Java and want to use a framework that simplifies backend development.

Recommendation: Choose the technologies you are most comfortable with and that align with your coursework or future career interests.

---

3. System Initialization and Configuration

What to Do:

- Develop an interface (GUI or CLI) that allows users to configure the system before it starts.

- Parameters to configure:
    - **Total Number of Tickets (totalTickets)**
    - **Ticket Release Rate (ticketReleaseRate)**
    - **Customer Retrieval Rate (customerRetrievalRate)**
    - **Maximum Ticket Capacity (maxTicketCapacity)**

How to Do It:

3.1 Design a Simple Configuration Interface

For GUI:

- **Create Input Forms:**
    - Design a window or page where the user can input the configuration parameters.
    - Use text fields, dropdowns, or sliders for input.
- **Implement Validation:**
    - Ensure the values entered are valid (e.g., numbers are positive, rates are within acceptable ranges).
    - Provide feedback if invalid input is detected.

For Command-Line Interface (CLI):

- **Prompt for Input:**
    - Use console input functions to ask the user for each parameter.
- **Validate Input:**
    - Check that the inputs are valid and prompt again if they are not.

3.2 Implement Configuration Management

- **Create a Configuration Class or Module:**
- **Save and Load Settings:**
    - **Serialization:**
        - Use serialization to save the Configuration object to a file.
    - **File Formats:**
        - **JSON:** Use a library like Gson (Java) or json module (Node.js) to read/write configuration in JSON format.
        - **Text File:** Write the settings line by line in a plain text file.

---

4. Implementing the Ticket Vendor (Producer)

What to Do:

- Simulate multiple vendors releasing tickets concurrently.
- Each vendor operates independently.

How to Do It:

4.1 Create a Vendor Class

Implement the Runnable Interface (Java):

4.2 Implement Multi-threading

- **Creating Vendor Threads:**
- **Vendor Logic in run():**

4.3 Synchronization

- Ensure that addTickets() in TicketPool is thread-safe.

---

5. Implementing the Customer (Consumer)

What to Do:

- Simulate multiple customers purchasing tickets concurrently.
- Each customer attempts to purchase tickets independently.

How to Do It:

5.1 Create a Customer Class

Implement the Runnable Interface (Java):

5.2 Implement Multi-threading

- **Creating Customer Threads:**
- **Customer Logic in run():**

5.3 Synchronization

- Ensure that removeTicket() in TicketPool is thread-safe.

---

6. Ticket Management

What to Do:

- Implement a shared ticket pool.
- Ensure safe concurrent access.

How to Do It:

6.1 Create a TicketPool Class

Using a Thread-Safe Data Structure (Java):

- **Option 1:** Use Vector
- **Option 2:** Use Collections.synchronizedList

6.2 Implement Methods

- **addTickets():** Vendors use this method to add tickets to the pool.
- **removeTicket():** Customers use this method to purchase tickets.

6.3 Synchronization

- Use synchronized methods or blocks to prevent race conditions.

---

7. Logging and Error Handling

What to Do:

- Implement basic logging of system activities.
- Handle exceptions gracefully.

How to Do It:

7.1 Logging

- **Console Logging:**
  - Use System.out.println() (Java) or console.log() (JavaScript) to output messages.
- **File Logging (Java):**
  - Use FileWriter or BufferedWriter to write logs to a file.
- **Consider Using a Logging Framework:**
  - **Java:** Use java.util.logging or Log4j for more advanced logging features.

7.2 Error Handling

- **Use Try-Catch Blocks:**
- **Provide Meaningful Error Messages:**
  - Inform the user or administrator about what went wrong and possible solutions.

---

8. Developing the User Interface (UI)

What to Do:

- Create a UI to display system status and allow interaction.
- UI should include:
  - Display of current ticket availability.
  - Options to start and stop the system.
  - Input fields for configuration settings.

How to Do It:

8.1 For JavaFX

Setting Up the JavaFX Application:

- **Main Application Class:**

Building the UI:

- **Layout:**
  - Use layouts like VBox, HBox, GridPane to organize UI components.
- **Components:**
  - **Labels:** To display text.
  - **TextFields:** For user input.
  - **Buttons:** To start and stop the system.
  - **ListView or TableView:** To display ticket availability and logs.

Handling Events:

- **Button Actions:**
- **Updating UI from Other Threads:**
  - Use Platform.runLater() to update the UI safely from other threads.

8.2 For React.js / Angular

Setting Up the Application:

Building Components:

- **Components:**
  - **ConfigurationForm:** To input configuration settings.
  - **TicketDisplay:** To show current ticket availability.
  - **ControlPanel:** Start/stop buttons.
  - **LogDisplay:** To show logs.

State Management:

- **React.js:**
  - Use useState for local state or useReducer for more complex state management.
  - Consider using a state management library like Redux for larger applications.
- **Angular:**
  - Use @Input() and @Output() decorators for component communication.
  - Consider using a service with RxJS Subject or BehaviorSubject for shared state.

Updating the UI:

- **WebSockets or Polling:**
  - Since the backend operates separately, use WebSockets or periodic polling to fetch the latest data.
- **Example with Polling:**

9. Deliverables

9.1 Source Code

- **Organize Your Project:**
  - Follow standard project structures for your chosen technology.
  - Separate code into packages or modules (e.g., models, controllers, views).
- **Documentation:**
  - Comment your code to explain complex logic.
  - Use Javadoc (Java) or similar tools to generate API documentation.

9.2 README File

- **Include:**
  - **Introduction:** Brief overview of the system.
  - **Setup Instructions:**
    - Prerequisites (e.g., Java version, Node.js version).
    - How to build and run the application.
  - **Usage Instructions:**
    - How to configure and start the system.
    - Explanation of UI controls.

9.3 Diagrams

- **Class Diagram:**
  - Show main classes (Vendor, Customer, TicketPool, etc.) and relationships.
- **Sequence Diagram:**
  - Illustrate the interaction flow:
    - Vendor releases tickets to TicketPool.
    - Customer retrieves tickets from TicketPool.
- **Tools:**
  - Use diagramming tools like Lucidchart, draw.io, or Visio.

9.4 Testing Report

- **Test Cases:**
  - Describe scenarios you tested (e.g., multiple customers purchasing tickets, ticket pool reaching maximum capacity).
- **Results:**
  - Document whether each test case passed or failed.
  - Include any issues found and how they were resolved.

9.5 Demonstration Video

- **Content:**
  - Start with a brief introduction.
  - Show the configuration process.
  - Demonstrate the system running with multiple vendors and customers.
  - Highlight concurrent ticket additions and purchases.
  - Conclude with any key observations or findings.
- **Tips:**
  - Keep the video between 5-10 minutes.
  - Ensure audio is clear if you are providing narration.
  - Use screen recording software like OBS Studio or Camtasia.

10. Optional Advanced Functionalities (Bonus Marks)

10.1 Priority Customers

- **Implement VIP Customers:**
  - Modify the Customer class to include a priority attribute.
  - Use a priority queue or synchronization mechanism to give VIP customers preference.

10.2 Dynamic Vendor/Customer Management

- **Add/Remove Vendors and Customers at Runtime:**
  - Implement functionality in the UI or CLI to start or stop vendor/customer threads dynamically.

10.3 Real-Time Analytics Dashboard

- **Visual Charts:**
  - Use charting libraries to display ticket sales over time.
  - Update charts in real-time as transactions occur.

10.4 Advanced Synchronization

- **Locks and Semaphores:**
  - Use ReentrantLock or Semaphore (Java) for more control over synchronization.
  - Ensure proper acquisition and release of locks to prevent deadlocks.

10.5 Persistence

- **Save Data to a Database:**
  - Use a simple database like SQLite or an embedded database.
  - Store ticket sales, customer data, and vendor data.
  - Implement basic CRUD operations.

---

11. Guidelines

11.1 Object-Oriented Programming

- **Principles:**
  - **Encapsulation:** Keep data private within classes and expose it through public methods.
  - **Inheritance:** Use to extend classes and reuse code.
  - **Polymorphism:** Allow objects to be treated as instances of their parent class.
- **Best Practices:**
  - Use interfaces or abstract classes where appropriate.
  - Keep classes focused on a single responsibility.

11.2 Multi-threading and Synchronization

- **Thread Safety:**
  - Protect shared resources with synchronization mechanisms.
  - Be cautious of deadlocks and race conditions.
- **Best Practices:**
  - Keep synchronized blocks as short as possible.
  - Avoid synchronizing on objects that can be accessed externally.

11.3 Error Handling

- **User-Friendly Messages:**
  - Inform users of errors in a clear and concise manner.

- **Prevent Crashes:**
    - Handle exceptions to prevent the application from crashing unexpectedly.

11.4 Code Quality

- **Readability:**
    - Use meaningful variable and method names.
    - Follow consistent naming conventions.
- **Maintainability:**
    - Write modular code.
    - Avoid duplication.

11.5 Documentation

- **Comments:**
    - Explain why certain decisions were made, not just what the code does.
- **Instructions:**
    - Provide clear setup and usage instructions in the README.

---

12. Frequently Asked Questions (FAQ)

1. **Can I use external libraries for threading and synchronization?**
    - **Answer:** Yes, you may use standard libraries provided by your programming language. However, avoid using third-party libraries that abstract away core functionalities you are expected to implement.
2. **Is collaboration allowed?**
    - **Answer:** This is an individual assignment. Discussing concepts is acceptable, but sharing code is not permitted.
3. **Can I use design patterns not mentioned in the specification?**
    - **Answer:** Yes, using appropriate design patterns like Singleton, Observer, or Factory is encouraged if it enhances your design.
4. **What if I have difficulties with multi-threading?**
    - **Answer:** Review your course materials, official documentation, and seek help from instructors or teaching assistants.
5. **Do I need to implement security features?**
    - **Answer:** Basic validation and error handling are required. Implementing advanced security features is optional and can earn bonus marks.

---

13. Conclusion

This project is an excellent opportunity to apply your knowledge of OOP, multi-threading, and synchronization in a practical context. Start by planning your implementation carefully, choose the technologies you are most comfortable with, and build your system incrementally.

Remember to test your application thoroughly and ensure all requirements are met. Good luck with your project!

Additional Tips:

- **Start Early:** Give yourself ample time to work on each component.
- **Version Control:** Use Git or another version control system to track changes.
- **Backup Your Work:** Regularly back up your project to avoid data loss.
- **Seek Feedback:** Don't hesitate to ask for help or clarification when needed.

Weekly Plan to Complete the Real-Time Event Ticketing System by November 30, 2024

Overview

This plan outlines a week-by-week approach to help you complete the Real-Time Event Ticketing System coursework by November 30, 2024. Starting from October 12, 2024, the plan spans over 8 weeks, breaking down the project into manageable tasks to ensure timely completion while allowing for thorough development, testing, and documentation.

Week 1: October 12 - October 18

Objectives:

- **Understand the Project Requirements**
- **Set Up the Development Environment**
- **Begin System Design**

Tasks:

1. **Review the Coursework Specification Thoroughly**
    - **Action:**
        - Read all sections of the specification carefully.
        - Identify and note down key requirements and deliverables.
        - Highlight any areas that are unclear and prepare questions for your instructor if needed.
2. **Choose the Technology Stack**
    - **Frontend:**
        - Decide between **JavaFX**, **React.js**, or **Angular** based on your proficiency.
    - **Backend:**
        - Choose between **Java**, **Node.js**, or **Spring Boot**.
    - **Action:**
        - Install necessary software and tools for your chosen technologies.
        - Ensure all environments are properly configured.
3. **Set Up Version Control**
    - **Action:**
        - Initialize a Git repository for your project.
        - Create a .gitignore file to exclude unnecessary files.
        - Make an initial commit with the basic project structure.
4. **Initial System Design**
    - **Action:**
        - Sketch out high-level diagrams (e.g., use case diagrams, class diagrams).
        - Define the main classes (Vendor, Customer, TicketPool, Configuration).
        - Decide on the relationships between classes and how they will interact.
5. **Create a Project Plan**
    - **Action:**
        - Outline the main milestones and deadlines.
        - Allocate time for development, testing, and documentation.

        - Set personal goals for each week.

Deliverables by End of Week 1:

- Basic project repository initialized.
- High-level system design diagrams.
- A detailed project plan with milestones.

Week 2: October 19 - October 25

Objectives:

- **Develop System Initialization and Configuration Module**
- **Implement Core Classes for Vendors and Customers**

Tasks:

1. **Implement Configuration Module**
    - **Action:**
        - Create a Configuration class to hold all configurable parameters.
        - Develop methods to save and load configurations (e.g., using JSON files).
        - Implement validation to ensure input values are within acceptable ranges.
2. **Design and Implement Core Classes**
    - **Vendor Class:**
        - Define attributes: vendorId, ticketsPerRelease, releaseInterval.
        - Implement the Runnable interface (or equivalent) for threading.
    - **Customer Class:**
        - Define attributes: customerId, retrievalInterval.
        - Implement the Runnable interface for threading.
3. **Set Up the TicketPool Class**
    - **Action:**
        - Create the TicketPool class using a thread-safe data structure.
        - Implement addTickets() and removeTicket() methods with basic synchronization.
4. **Begin Implementing Multi-threading**
    - **Action:**
        - Write basic run() methods for Vendor and Customer classes.
        - Start testing with a few threads to simulate vendors and customers.

Deliverables by End of Week 2:

- Functional Configuration module.
- Core classes (Vendor, Customer, TicketPool) implemented.
- Basic multi-threading setup with sample threads running.

Week 3: October 26 - November 1

Objectives:

- **Enhance Multi-threading and Synchronization**
- **Implement Vendor and Customer Logic**
- **Implement Logging and Error Handling**

Tasks:

1. **Enhance Multi-threading Implementation**

- **Action:**
  - Ensure thread safety when accessing the ticket pool.
  - Use synchronization techniques to prevent race conditions.

2. **Develop Vendor Logic**
   - **Action:**
     - Implement the logic in the run() method for vendors to release tickets at specified intervals.
     - Ensure vendors stop adding tickets when the maximum capacity is reached.
3. **Develop Customer Logic**
   - **Action:**
     - Implement the logic in the run() method for customers to attempt ticket purchases at specified intervals.
     - Handle scenarios where no tickets are available.
4. **Implement Logging**
   - **Action:**
     - Set up a simple logging mechanism to record events (e.g., ticket additions, purchases).
     - Log messages to the console or a file.
5. **Implement Basic Error Handling**
   - **Action:**
     - Use try-catch blocks to handle potential exceptions.
     - Provide meaningful error messages to aid in debugging.

Deliverables by End of Week 3:

- Enhanced multi-threading with proper synchronization.
- Functional Vendor and Customer classes with implemented logic.
- Basic logging system operational.
- Error handling mechanisms in place.

Week 4: November 2 - November 8
Objectives:

- **Develop the User Interface (UI)**
- **Integrate UI with Backend Logic**
- **Continue Testing and Debugging**

Tasks:

1. **Design the UI**
   - **Action:**
     - For **JavaFX**:
       - Create the main window with input forms for configuration.
       - Add labels and buttons to display system status and control the system.
     - For **React.js/Angular**:
       - Build components for configuration, status display, and control buttons.
2. **Implement UI Functionality**
   - **Action:**
     - Implement event handlers for UI controls (e.g., start/stop buttons).

        - Update the UI to reflect changes in the system (e.g., ticket availability).
3. **Integrate UI with Backend**
    - **Action:**
        - Connect UI inputs to the Configuration module.
        - Ensure real-time updates from the backend are reflected in the UI.
4. **Testing and Debugging**
    - **Action:**
        - Test the UI thoroughly to ensure it responds correctly.
        - Fix any issues related to event handling and data display.

Deliverables by End of Week 4:

- Functional UI with basic features.
- UI integrated with backend logic.
- Initial testing completed, and identified bugs fixed.

Week 5: November 9 - November 15

Objectives:

- **Implement Ticket Management Enhancements**
- **Expand Logging and Error Handling**
- **Begin Documentation**

Tasks:

1. **Enhance Ticket Management**
    - **Action:**
        - Refine the TicketPool class to handle edge cases.
        - Implement maximum capacity checks and proper notifications.
2. **Improve Logging System**
    - **Action:**
        - Enhance log messages with timestamps and detailed information.
        - Optionally, use a logging framework for better management.
3. **Strengthen Error Handling**
    - **Action:**
        - Handle specific exceptions (e.g., InterruptedException, custom exceptions).
        - Ensure the system remains stable under error conditions.
4. **Begin Writing Documentation**
    - **Action:**
        - Start drafting the **README** file with setup and usage instructions.
        - Comment your code thoroughly, explaining complex logic.
5. **Continue Testing**
    - **Action:**
        - Perform more comprehensive tests, including stress testing with multiple threads.
        - Document test cases and results.

Deliverables by End of Week 5:

- Enhanced ticket management with robust handling.
- Improved logging and error handling mechanisms.
- Draft of the README file and code documentation.

- Updated testing report with new test cases.

Week 6: November 16 - November 22
Objectives:

- **Implement Optional Advanced Functionalities (Bonus Marks)**
- **Finalize All Core Features**
- **Prepare Diagrams**

Tasks:

1. **Choose Optional Advanced Features**
    - **Options:**
        - Priority Customers (VIP customers)
        - Dynamic Vendor/Customer Management
        - Real-Time Analytics Dashboard
        - Advanced Synchronization
        - Persistence (saving data to a database)
    - **Action:**
        - Select features that interest you and are feasible within the time frame.
        - Plan how to integrate them into your existing system.
2. **Implement Chosen Advanced Features**
    - **Action:**
        - Develop the selected features, ensuring they are well-integrated.
        - Test each feature thoroughly before moving on.
3. **Finalize Core Features**
    - **Action:**
        - Review all core functionalities to ensure they meet the requirements.
        - Refactor code for better performance and readability.
4. **Prepare Required Diagrams**
    - **Action:**
        - Create the class diagram showing main classes and relationships.
        - Develop a sequence diagram illustrating the interaction between vendors, customers, and the ticket pool.
        - Use tools like Lucidchart, draw.io, or similar.

Deliverables by End of Week 6:

- Implemented optional advanced features.
- All core features finalized and functioning correctly.
- Class diagram and sequence diagram completed.
- Updated testing report including tests for new features.

Week 7: November 23 - November 29
Objectives:

- **Finalize Documentation**
- **Record Demonstration Video**
- **Perform Final Testing and Code Review**

Tasks:

1. **Complete Documentation**

- **Action:**
  - Finalize the README file with all necessary details.
  - Ensure code comments are complete and helpful.
  - Compile the testing report with all test cases and results.

2. **Record the Demonstration Video**
   - **Action:**
     - Plan the video content to cover all required aspects.
     - Use screen recording software to capture the system in action.
     - Demonstrate concurrent ticket additions and purchases.
     - Keep the video within the 5-10 minute guideline.
3. **Perform Final Testing**
   - **Action:**
     - Conduct exhaustive testing, including edge cases.
     - Verify that all features work as expected under various conditions.
4. **Code Review and Cleanup**
   - **Action:**
     - Review your code for any potential issues.
     - Remove any unnecessary code or comments.
     - Ensure code follows standard conventions.

Deliverables by End of Week 7:

- All documentation finalized.
- Demonstration video recorded and ready for submission.
- Final testing completed with a comprehensive report.
- Cleaned and reviewed codebase.

Week 8: November 30

Objectives:

- **Prepare for Submission**
- **Submit the Project**

Tasks:

1. **Prepare All Deliverables**
   - **Action:**
     - Ensure all files are organized and named appropriately.
     - Double-check that all required deliverables are included:
       - Source code
       - README file
       - Diagrams
       - Testing report
       - Demonstration video
2. **Final Verification**
   - **Action:**
     - Run the application one last time to ensure everything works.
     - Verify that all instructions in the README are accurate.
3. **Submit the Project**
   - **Action:**
     - Follow the submission guidelines provided by your instructor.

        - Upload all necessary files to the designated platform before the deadline.

Deliverables by End of Week 8:

- Project submitted successfully with all required components.

Additional Tips

- **Time Management:**
    - Allocate specific hours each day for coursework.
    - Use tools like calendars or planners to keep track of tasks.
- **Stay Organized:**
    - Keep your project files well-organized.
    - Use meaningful names for variables, methods, and classes.
- **Seek Help When Needed:**
    - If you encounter difficulties, consult your instructor or peers.
    - Use online resources and documentation for additional guidance.
- **Backup Your Work:**
    - Regularly push changes to your Git repository.
    - Keep backups in case of unforeseen issues.
- **Maintain Code Quality:**
    - Follow coding standards and best practices.
    - Write modular, reusable code.
- **Test Frequently:**
    - Don’t wait until the end to test your code.
    - Regular testing helps catch issues early.
- **Take Care of Yourself:**
    - Get adequate rest and take breaks to avoid burnout.
    - Maintain a healthy balance between work and leisure.

Conclusion

By following this weekly plan, you'll be able to manage your time effectively and ensure that you complete the **Real-Time Event Ticketing System** coursework by the deadline of November 28th, 2024. Stay disciplined, focus on one task at a time, and don't hesitate to reach out for help if needed.

Good Luck with Your Project!