**Slide 1 – Title: Daily Journal App**

**"Good Morning, everyone. My name is Stefan Josevski, and I’m excited to present my final project for CSI 2300 – the Daily Journal App.**  
**This is a JavaFX-based application that lets users write journal entries, set reminders, tag their entries, and even export them.**  
**The app was developed with object-oriented principles and covers concepts from Chapters 2 through 11."**

**Slide 2 – Overview**

**"Here’s a quick overview of what I’ll be covering today. First, I’ll explain the purpose of the application and its features. Then, I’ll walk through how the project incorporates Java concepts from each chapter. After that, we’ll go through a quick demo of how the app works, and I’ll end with questions."**

**Slide 3 – Chapter 2: Data and Expressions**

**"The app heavily uses data variables and expressions, especially in the JournalEntry class.**  
**We store fields like title, content, and dateCreated using variables. For example, Date dateCreated = new Date(); automatically stores the timestamp when a new entry is created.**  
**We also use expressions to manipulate text and handle input, like converting strings to lowercase when searching for keywords in entries."**

**Slide 4 – Chapter 3: Using Classes and Objects**

**"We use objects throughout the application. For instance, a User object is created at login. Then, we create JournalEntry objects when users write new entries.**  
**Methods like addEntry() and getEntries() are used to manage those objects.**  
**This highlights how objects can interact with each other in a structured, meaningful way."**

**Slide 5 – Chapter 4: Writing Classes**

**"I wrote several custom classes: User, JournalEntry, Reminder, Tag, and ExportManager. Each one encapsulates related data and behavior.**  
**For example, the JournalEntry constructor initializes the entry’s ID, title, content, and sets the creation date.**  
**This chapter’s focus on constructors and encapsulation was especially helpful in designing the class structure."**

**Slide 6 – Chapter 5: Conditionals and Loops**

**"The login system uses conditional statements. If the entered username and password match the stored credentials, the app opens the main menu. Otherwise, it shows an error alert.**  
**This ensures the user can’t access journal functions unless they log in successfully."**

**Slide 7 – Chapter 6: More Conditionals and Loops**

**"Loops are used to display all entries. We use a for loop to iterate through the ArrayList of journal entries.**  
**Each entry is appended to a string builder and then displayed in an alert dialog.**  
**This allows users to view a readable summary of all their entries quickly."**

**Slide 8 – Chapter 7: Object-Oriented Design**

**"I used object-oriented design principles throughout the app.**  
**The User class holds journal entries and reminders, encapsulating everything a user owns. Each class handles its own responsibilities—this separation makes the app easy to extend and maintain.**  
**For example, if I wanted to add search filters or notifications later, I could do it without rewriting the whole app."**

**Slide 9 – Chapter 8: Arrays (ArrayLists)**

**"Instead of fixed-size arrays, I used ArrayLists for storing journal entries, reminders, and tags.**  
**For example: List<JournalEntry> journalEntries = new ArrayList<>(); lets the app dynamically grow and shrink these collections based on the user’s actions.**  
**ArrayLists make it easier to loop through and manage user data."**

**Slide 10 – Chapter 9: Inheritance**

**"While I didn’t use inheritance heavily in the current version, the design allows for it.**  
**For instance, we could have a generic Entry superclass that both JournalEntry and Reminder could extend, since they share properties like ID and Date.**  
**This would reduce redundancy and help support future features like notifications or calendar integration."**

**Slide 11 – Chapter 10: Polymorphism**

**"Polymorphism is shown in the export functionality. I created a class called ExportManager with methods like exportToPDF() and exportToWord().**  
**A future improvement would be to create an Exporter interface with a single export() method. Then PDF and Word exporters could implement this method, allowing us to write cleaner, more flexible code."**

**Slide 12 – Chapter 11: Exceptions**

**"I also used exception handling to make the app more user-friendly. For example, when setting a reminder, the user is asked to enter the number of days.**  
**If they enter something that’s not a number, a NumberFormatException is caught, and the app shows an error alert instead of crashing.**  
**This kind of defensive programming improves the user experience."**

**Slide 13 – Demo & Summary**

**"Let’s go through a quick demo. I’ll show how to log in, create an entry, view entries, set a reminder, and simulate exporting to PDF.**  
**As you can see, everything is handled in a clean and simple GUI built with JavaFX."**  
**"To summarize: the app uses clean object-oriented design, meaningful use of Java collections, exception handling, and a user interface built from scratch. It’s designed to be extensible and user-friendly."**

**Slide 14 – Questions**

**"That brings us to the end of the presentation. If you have any questions about the design, the code, or future improvements I’m planning, I’d love to hear them. Thanks so much for your attention!"**