**CS 5800**

**FINAL**

GitHub Link: <https://github.com/gsiguenza12/CS5800-Homework/tree/main/Final>

Assume you are creating a startup. Your goal is to create an application of your own liking or in similar fashion to a rival application that is already in use today. You will need to complete the assignment in 3 parts.

Start up Application: Open Note – note taking app for college students

Proposed design patterns:  
**Structural Design Pattern: Composite Pattern**

Used for organizing notes into a hierarchy (e.g., folders containing notes and subfolders).

**Creational Design Pattern: Factory Method**

Used for creating different types of notes (e.g., text notes, checklist notes).

**Behavioral Design Pattern: Observer Pattern**

Used for notifying observers when a note is updated (e.g., a tag system or sync module that listens for changes).

**Other Design Patterns:**

**State and Memento Design Patterns**

Used for undo function in the note taking application.

(Recommended but not required)

Create Use Cases that walk though external actors interacting with your system.

**REQUIRED**

1. Create **UML Activity Diagrams** (Refer to UML 2 Lecture Notes)
2. Create **UML Sequence** of 2 of the activity diagrams (Refer to UML 2 Lecture Notes)
3. Create **UML Class Diagrams** (Refer to UML Lecture Notes)

Be ambitious! Assume this is a real company you will start. I am looking forward to seeing what you come up with.

UML SCREEN SHOTS:



A screenshot of a computer

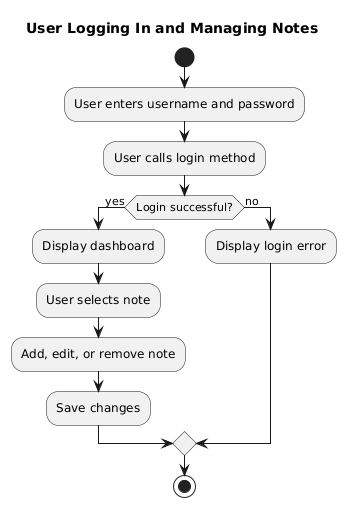
Description automatically generated

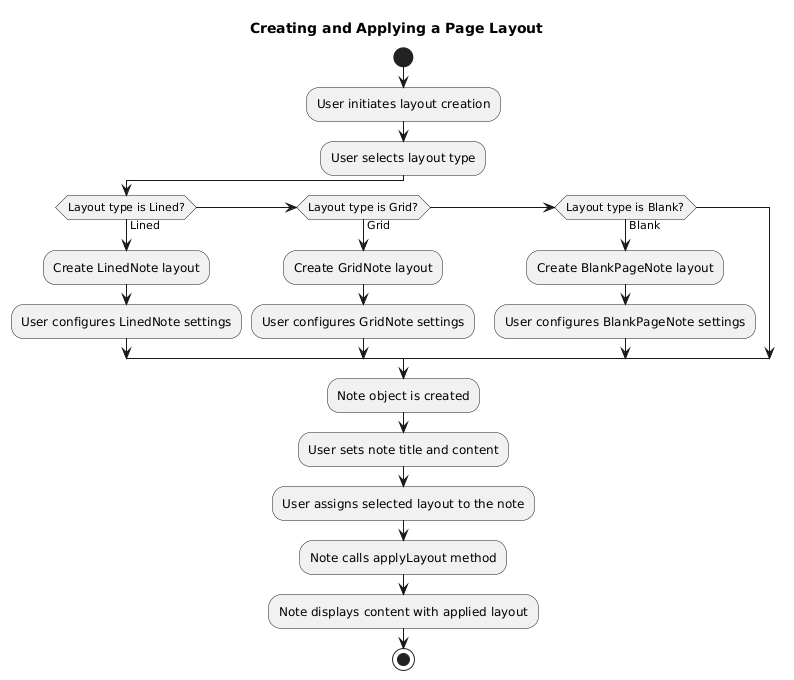
A screenshot of a computer

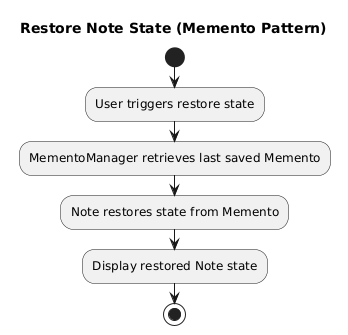
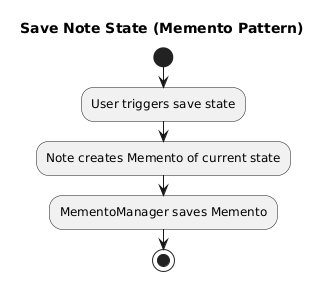
Description automatically generated

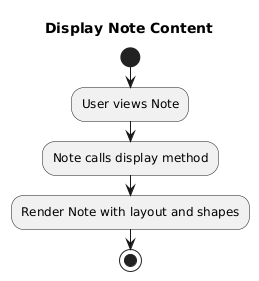
A screenshot of a computer

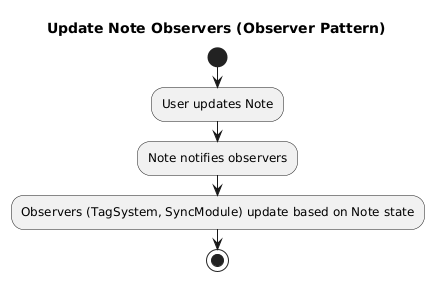
Description automatically generated

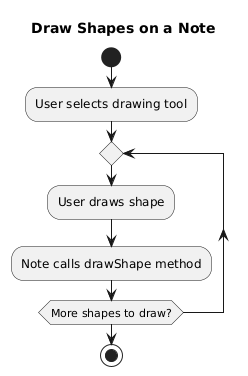


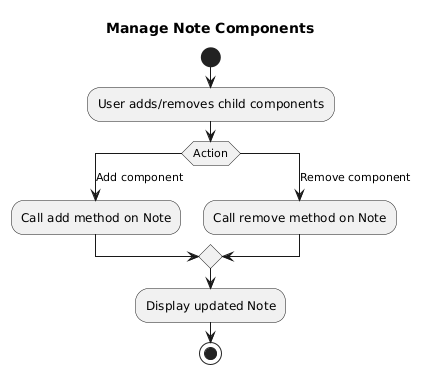


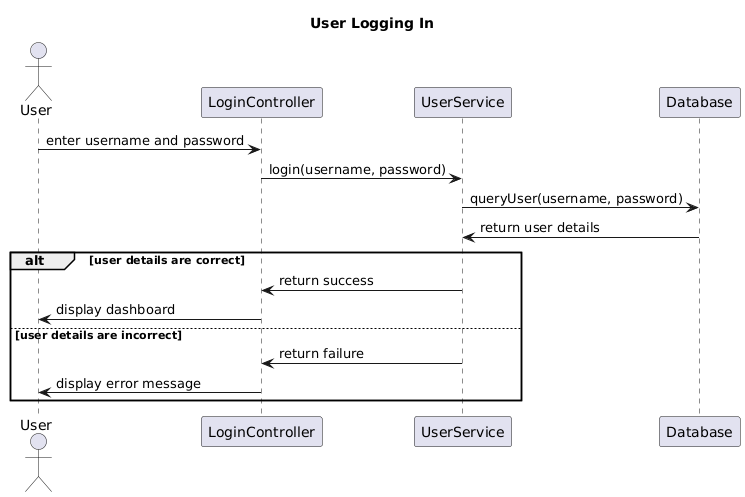


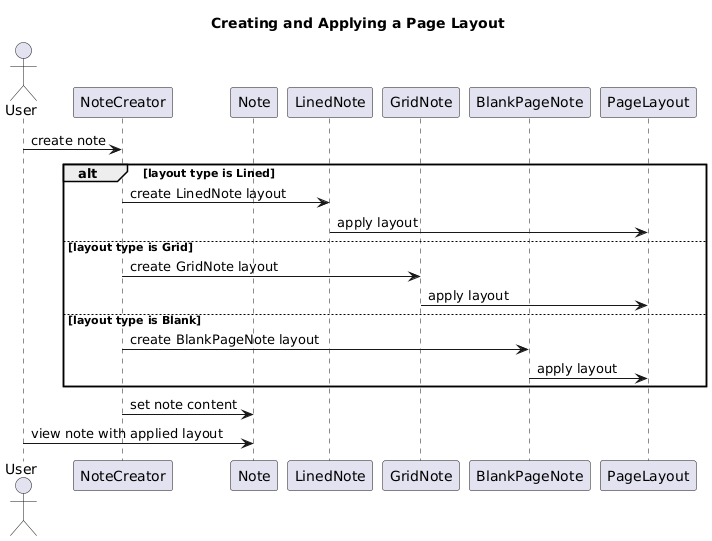












**PUML CODE FOR CLASS UML DIAGRAMS:**

' ==============================

' Open Note Application Plant UML Code

' ==============================

@startuml

title Open Note App

' ==============================

' Classes and Interfaces

' ==============================

class User {

- username: String

- password: String

+ getUsername(): String

+ setUsername(String username)

+ setPassword(String password)

+ login(username: String, password: String): boolean

}

interface NoteComponent {

+ add(NoteComponent component)

+ remove(NoteComponent component)

+ getChild(int index): NoteComponent

+ display()

}

' Abstract Note class that holds content and page layout

abstract class Note implements NoteComponent {

- title: String

- content: String

- pageLayout: PageLayout

- drawingTool: Drawing

- state: NoteState

- memento: NoteMemento

+ add(NoteComponent component)

+ remove(NoteComponent component)

+ getChild(int index): NoteComponent

+ display()

+ setContent(String content)

+ getContent(): String

+ setPageLayout(PageLayout layout)

+ getPageLayout(): PageLayout

+ setDrawingTool(Drawing tool)

+ getDrawingTool(): Drawing

+ draw(Shape shape, String color, float thickness)

+ changeState(NoteState state)

+ createMemento(): NoteMemento

+ restoreMemento(NoteMemento memento)

}

class NoteMemento {

- state: String

+ getState(): String

+ setState(String state)

}

class MementoManager {

- mementos: List<NoteMemento>

+ saveMemento(NoteMemento memento)

+ getMemento(): NoteMemento

}

interface NoteState {

+ applyState(Note note)

}

class ContentState implements NoteState {

+ applyState(Note note)

}

class LayoutState implements NoteState {

+ applyState(Note note)

}

' PageLayout interface that defines the template for note pages

interface PageLayout {

+ applyLayout(Note note)

}

' Concrete PageLayouts (Templates)

class LinedNote implements PageLayout {

+ applyLayout(Note note)

}

class GridNote implements PageLayout {

+ applyLayout(Note note)

}

class BlankPageNote implements PageLayout {

+ applyLayout(Note note)

}

' Subject class for composite behavior

class Subject implements NoteComponent {

- name: String

- children: List<NoteComponent>

+ add(NoteComponent component)

+ remove(NoteComponent component)

+ getChild(int index): NoteComponent

+ display()

}

abstract class NoteFactory {

+ createNote(): Note

}

' Factories for creating different note types with page layouts

class GridNoteFactory extends NoteFactory {

+ createNote(): Note

}

class LinedNoteFactory extends NoteFactory {

+ createNote(): Note

}

class BlankPageNoteFactory extends NoteFactory {

+ createNote(): Note

}

interface NoteObserver {

+ update(note: Note)

}

class TagSystem implements NoteObserver {

+ update(note: Note)

}

class SyncModule implements NoteObserver {

+ update(note: Note)

}

class NoteObservable {

- observers: List<NoteObserver>

+ addObserver(NoteObserver observer)

+ removeObserver(NoteObserver observer)

+ notifyObservers()

}

interface Drawing {

+ drawShape(Shape shape, String color, float thickness)

}

' Abstract Shape class

abstract class Shape <<abstract>> {

+ draw(Drawing drawing, String color, float thickness)

}

' Specific shapes

class Circle extends Shape {

+ draw(Drawing drawing, String color, float thickness)

}

class Rectangle extends Shape {

+ draw(Drawing drawing, String color, float thickness)

}

class Square extends Shape {

+ draw(Drawing drawing, String color, float thickness)

}

class CurvedLine extends Shape {

+ draw(Drawing drawing, String color, float thickness)

}

class DashedLine extends Shape {

+ draw(Drawing drawing, String color, float thickness)

}

class Arrow extends Shape {

+ draw(Drawing drawing, String color, float thickness)

}

class Pen implements Drawing {

- color: String

- thickness: float

+ drawShape(Shape shape, String color, float thickness)

+ setColor(String color)

+ getColor(): String

+ setThickness(float thickness)

+ getThickness(): float

}

class Highlighter implements Drawing {

- opacity: float

+ drawShape(Shape shape, String color, float thickness)

+ setOpacity(float opacity)

+ getOpacity(): float

}

class Pencil implements Drawing {

- color: String

- thickness: float

+ drawShape(Shape shape, String color, float thickness)

+ setColor(String color)

+ getColor(): String

+ setThickness(float thickness)

+ getThickness(): float

}

class Brush implements Drawing {

- size: float

- texture: String

+ drawShape(Shape shape, String color, float thickness)

+ setSize(float size)

+ getSize(): float

+ setTexture(String texture)

+ getTexture(): String

}

class Marker implements Drawing {

- color: String

- thickness: float

+ drawShape(Shape shape, String color, float thickness)

+ setColor(String color)

+ getColor(): String

+ setThickness(float thickness)

+ getThickness(): float

}

class Eraser implements Drawing {

- size: float

+ drawShape(Shape shape, String color, float thickness)

+ setSize(float size)

+ getSize(): float

}

class LineTool implements Drawing {

- color: String

- thickness: float

+ drawShape(Shape shape, String color, float thickness)

+ setColor(String color)

+ getColor(): String

+ setThickness(float thickness)

+ getThickness(): float

}

class ShapeTool implements Drawing {

- color: String

- thickness: float

+ drawShape(Shape shape, String color, float thickness)

+ setColor(String color)

+ getColor(): String

+ setThickness(float thickness)

+ getThickness(): float

}

class TextTool implements Drawing {

- font: String

- size: float

+ drawShape(Shape shape, String color, float thickness)

+ setFont(String font)

+ getFont(): String

+ setSize(float size)

+ getSize(): float

}

' ==============================

' Relationships

' ==============================

Note ..|> NoteObservable

NoteObservable o-- NoteObserver

'NoteObserver <|.. TagSystem'

'NoteObserver <|.. SyncModule'

Subject o-- NoteComponent : contains

User "1" o-- "\*" Note : manages notes

Note o-- Drawing : uses drawing tools

Note o-- NoteState : manages state

Note o-- NoteMemento : creates and restores mementos

MementoManager o-- NoteMemento : manages mementos

MementoManager o-- Note : interacts with notes

Note o-- PageLayout : applies layout

Note ..|> NoteFactory : created by

Note -> Shape : uses shapes for drawing

Drawing --|> Shape : shapes are drawable

@enduml

' ==============================

' User Logging in Activity UML

' ==============================

@startuml

title User Logging In and Managing Notes

start

:User enters username and password;

:User calls login method;

if (Login successful?) then (yes)

:Display dashboard;

:User selects note;

:Add, edit, or remove note;

:Save changes;

else (no)

:Display login error;

endif

stop

@enduml

' ==============================

' Applying Page Layout Activity UML

' ==============================

@startuml

title Creating and Applying a Page Layout

start

:User initiates layout creation;

:User selects layout type;

if (Layout type is Lined?) then (Lined)

:Create LinedNote layout;

:User configures LinedNote settings;

elseif (Layout type is Grid?) then (Grid)

:Create GridNote layout;

:User configures GridNote settings;

elseif (Layout type is Blank?) then (Blank)

:Create BlankPageNote layout;

:User configures BlankPageNote settings;

endif

:Note object is created;

:User sets note title and content;

:User assigns selected layout to the note;

:Note calls applyLayout method;

:Note displays content with applied layout;

stop

@enduml

' ==============================

' Save Note State Activity UML

' ==============================

@startuml

title Save Note State (Memento Pattern)

start

:User triggers save state;

:Note creates Memento of current state;

:MementoManager saves Memento;

stop

@enduml

' ==============================

' Restore Note State Activity UML

' ==============================

@startuml

title Restore Note State (Memento Pattern)

start

:User triggers restore state;

:MementoManager retrieves last saved Memento;

:Note restores state from Memento;

:Display restored Note state;

stop

@enduml

' ==============================

' Update Note Observers Activity UML

' ==============================

@startuml

title Update Note Observers (Observer Pattern)

start

:User updates Note;

:Note notifies observers;

:Observers (TagSystem, SyncModule) update based on Note state;

stop

@enduml

' ==============================

' Display Note Content Activity UML

' ==============================

@startuml

title Display Note Content

start

:User views Note;

:Note calls display method;

:Render Note with layout and shapes;

stop

@enduml

' ==============================

' Draw Shapes Activity UML

' ==============================

@startuml

title Draw Shapes on a Note

start

:User selects drawing tool;

repeat

:User draws shape;

:Note calls drawShape method;

repeat while (More shapes to draw?)

stop

@enduml

' ==============================

' Manage Note Components Activity UML

' ==============================

@startuml

title Manage Note Components

start

:User adds/removes child components;

switch (Action)

case (Add component)

:Call add method on Note;

case (Remove component)

:Call remove method on Note;

endswitch

:Display updated Note;

stop

@enduml

' ==============================

' User Logging in Sequence UML

' ==============================

@startuml

title User Logging In

actor User

participant "LoginController" as LC

participant "UserService" as US

participant "Database" as DB

User -> LC: enter username and password

LC -> US: login(username, password)

US -> DB: queryUser(username, password)

DB -> US: return user details

alt user details are correct

US -> LC: return success

LC -> User: display dashboard

else user details are incorrect

US -> LC: return failure

LC -> User: display error message

end

@enduml

' ==============================

' Applying a Page Layout Sequence UML

' ==============================

@startuml

title User Logging In

actor User

title Creating and Applying a Page Layout

actor User

participant "NoteCreator" as NC

participant "Note" as N

participant "LinedNote" as LN

participant "GridNote" as GN

participant "BlankPageNote" as BN

participant "PageLayout" as PL

User -> NC: create note

alt layout type is Lined

NC -> LN: create LinedNote layout

LN -> PL: apply layout

else layout type is Grid

NC -> GN: create GridNote layout

GN -> PL: apply layout

else layout type is Blank

NC -> BN: create BlankPageNote layout

BN -> PL: apply layout

end

NC -> N: set note content

User -> N: view note with applied layout

@enduml