STATE UNIVERSITY OF ZANZIBAR (SUZA)
School of Computing Communication and Media Studies (SCCMS)

Assignment 2: JavaFX GUI & Design Patterns

Course Code:	IT6003	**Programme:**	MSc. Information T
Course Name:	Advanced Programming Using Java	**Semester:**	I
Type:	Individual Assignment	**Total Marks:**	100

Submission: ZIP file named `RegNo_Assignment2.zip`. Include all source files, FXML files (if used), and a `README.txt` with compilation and run instructions.

-2 **Viva:** You must demonstrate your application and explain the design patterns used. Inability to explain your code results in **zero marks**.

Part A: JavaFX Fundamentals [25 marks]

Task 1: Scientific Calculator GUI [15 marks]

Build a **Scientific Calculator** using JavaFX:

a) Create the calculator layout using JavaFX controls:
- A `TextField` (non-editable) for displaying input and results
- A `GridPane` with buttons for digits 0–9, operators (+, −, ×, ÷), decimal point, equals, and clear
- Additional buttons for: sin, cos, tan, $\sqrt{}$, x^2, x^y, log, ln, π, parentheses
- Style buttons with different colors for numbers, operators, and functions

[5 marks]

b) Implement the calculator logic:
- Support chain operations: `5 + 3 * 2 = 11` (left-to-right evaluation)
- Handle division by zero with an alert dialog
- Support keyboard input (number keys, Enter for equals, Escape for clear)

[5 marks]

c) Add a **history panel** using a `ListView` on the right side that shows all previous calculations. Allow clicking a history item to load the result. Use a `SplitPane` to make the history panel resizable. [3 marks]

d) Apply CSS styling: create a `calculator.css` file with custom styles for the calculator (dark theme with colored buttons). [2 marks]

Task 2: Interactive Form with Validation [10 marks]

Create a **Student Registration Form** with real-time validation:

a) Build the form using FXML (not programmatic layout). Include:
- `TextField`: Full Name, Registration Number, Email
- `PasswordField`: Password, Confirm Password
- `ComboBox`: Department (5 options)
- `DatePicker`: Date of Birth
- `RadioButton` group: Gender (Male/Female)
- `CheckBox`: Agree to Terms
- `Button`: Submit, Clear

[3 marks]

b) Add **real-time validation** using property listeners:

- Name: minimum 3 characters, letters and spaces only
- Email: valid email format (use regex)
- Password: minimum 8 chars, at least one uppercase, one digit, one special character
- Confirm Password: must match Password
- Registration Number: format `SUZA-XXXX-XXXX`

Show validation errors in red `Label` below each field in real-time. [4 marks]

c) On submit, display all entered data in an `Alert` dialog. The Submit button should be disabled until all validations pass. Use `BooleanBinding` to bind the button's `disableProperty`. [3 marks]

Part B: JavaFX Data Binding and TableView [25 marks]

Task 3: Student Management Dashboard [15 marks]

Build a complete CRUD interface using `TableView`:

a) Create a `Student` model class with JavaFX properties:

```
public class Student {
    private final StringProperty regNo;
    private final StringProperty name;
    private final StringProperty department;
    private final DoubleProperty gpa;
    private final ObjectProperty<LocalDate> enrollDate;
    // constructors, getters, setters, property methods
}
```

[2 marks]

b) Create a `TableView<Student>` with columns for all fields. The table should:

- Use an `ObservableList<Student>` as the data source
- Support sorting by clicking column headers
- Allow inline editing of name and GPA (using `TextFieldTableCell`)
- Highlight rows where GPA ¡ 2.0 in red using `setRowFactory()`

[4 marks]

c) Add a form panel below the table for adding new students. Bind form fields to table selection so that clicking a row populates the form. Include Add, Update, Delete buttons. [3 marks]

d) Add a `TextField` for searching/filtering. Use `FilteredList` and `SortedList` to filter the table in real-time as the user types. Search should match against name, regNo, and department. [3 marks]

e) Add a status bar at the bottom showing: total students, average GPA, and number of students per department. Update automatically when data changes using bindings. [3 marks]

Task 4: Charts and Data Visualization [10 marks]

Extend Task 3 with a charts panel (use `TabPane` to switch between table and charts):

a) `PieChart`: Show student distribution by department. Update dynamically when students are added/removed. [2 marks]

b) `BarChart`: Show average GPA per department. [2 marks]

c) `LineChart`: Show number of enrollments over time (by month/year). [2 marks]

d) Make charts interactive: clicking a pie slice or bar should filter the table to show only students from that department. [2 marks]

e) Add an "Export to PDF" button that saves the current chart as a PNG image using `WritableImage` and `ImageIO`. [2 marks]

Part C: Design Patterns in Practice [30 marks]

Task 5: MVC Architecture [10 marks]

Refactor Task 3 into a proper **Model-View-Controller** architecture:

a) **Model:** `Student.java` (data class), `StudentModel.java` (manages ObservableList, provides add/remove/update/search methods, handles file persistence). [3 marks]

b) **View:** `StudentView.fxml` (FXML layout) and `student-style.css` (styling). No logic in the view — only layout and style. [3 marks]

c) **Controller:** `StudentController.java` (handles user interactions, updates model, initializes bindings). Use `@FXML` annotations for all injected controls. [2 marks]

d) `App.java`: Main application class that loads FXML and launches the application. [2 marks]

Task 6: Implement Design Patterns [20 marks]

Integrate the following design patterns into a unified **Task Manager** application:

a) **Singleton — Application Configuration** [4 marks]
Create `AppConfig` that:
- Loads settings from a properties file (theme, language, window size)
- Provides global access via `AppConfig.getInstance()`
- Is thread-safe (lazy initialization with double-checked locking)
- Provides methods: `getString(key)`, `getInt(key)`, `setProperty(key, value)`, `save()`

b) **Observer — Event Notification System** [4 marks]
Implement an event system where:
- `TaskManager` is the `Observable` (subject)
- When a task is added, completed, or deleted, notify all observers
- Observers: `LogObserver` (writes to log file), `StatsObserver` (updates statistics), `UIObserver` (refreshes the GUI)
- Support adding/removing observers dynamically

c) **Strategy — Task Sorting Strategies** [4 marks]
Define a `SortStrategy` interface with method `List<Task> sort(List<Task> tasks)`. Implement:
- `SortByPriority` (HIGH, MEDIUM, LOW)
- `SortByDeadline` (earliest first)
- `SortByCreationDate` (newest first)
- `SortByName` (alphabetical)

Allow the user to switch sorting strategy via a `ComboBox` in the GUI.

d) **Builder — Task Builder** [4 marks]
Implement a `TaskBuilder` for constructing `Task` objects:

```
Task task = new TaskBuilder("Complete Assignment")
    .withDescription("Finish the Java assignment")
    .withPriority(Priority.HIGH)
    .withDeadline(LocalDate.of(2025, 6, 15))
    .withCategory("Academic")
    .withTags("java", "assignment", "urgent")
    .build();
```

Validate that required fields (title) are set. Throw `IllegalStateException` if `build()` is called without a title.

e) **Command — Undo/Redo System** [4 marks]
Implement the Command pattern for undo/redo:
- `Command` interface with `execute()` and `undo()` methods
- `AddTaskCommand`, `DeleteTaskCommand`, `EditTaskCommand`, `ToggleCompleteCommand`
- `CommandHistory` class with two stacks (undo/redo)
- Add Undo (Ctrl+Z) and Redo (Ctrl+Y) buttons/shortcuts to the GUI

- Support at least 20 levels of undo

Part D: Integration — Complete Task Manager Application [20 marks]

Task 7: Putting It All Together [20 marks]

Combine all patterns from Part C into a polished **Task Manager** application:

a) **Main Window Layout:** [5 marks]
- `MenuBar`: File (New, Open, Save, Exit), Edit (Undo, Redo), View (Sort By submenu), Help (About)
- `ToolBar`: Add Task, Delete, Mark Complete, Undo, Redo buttons with icons
- Center: `TableView` showing tasks with columns: Title, Priority, Deadline, Category, Status
- Right panel: Task details form (shown when a task is selected)
- Bottom: Status bar showing task counts (total, completed, pending, overdue)

b) **Task Features:** [5 marks]
- Create, edit, delete tasks using the Builder pattern
- Mark tasks as complete/incomplete
- Color-code by priority (red=HIGH, orange=MEDIUM, green=LOW)
- Highlight overdue tasks (deadline passed and not completed)
- Filter tasks: All, Active, Completed, Overdue

c) **Persistence:** [4 marks]
- Save tasks to a JSON-formatted text file
- Load tasks on application start
- Auto-save when the application closes (use `stage.setOnCloseRequest()`)

d) **Keyboard Shortcuts:** [2 marks]
- Ctrl+N: New task, Ctrl+S: Save, Ctrl+Z: Undo, Ctrl+Y: Redo
- Delete key: Delete selected task
- Space: Toggle complete on selected task

e) **Polish:** [4 marks]
- Apply a consistent CSS theme
- Show confirmation dialogs before delete
- Handle all edge cases (empty selection, no tasks, etc.)
- Write a brief user guide in `README.txt`

Grading Rubric

Part	Marks	Weight
Part A: JavaFX Fundamentals	25	25%
Part B: Data Binding & TableView	25	25%
Part C: Design Patterns	30	30%
Part D: Integration	20	20%
Total	**100**	**100%**

Viva Questions (be prepared):

1. Explain the JavaFX application lifecycle (`init()`, `start()`, `stop()`).
2. What is the difference between JavaFX properties and standard Java fields?
3. Explain the Observer pattern and how JavaFX uses it internally.
4. What is the advantage of the Builder pattern over a constructor with many parameters?
5. How does the Command pattern enable undo/redo?
6. Modify your Task Manager to add a new feature on the spot (live coding).

End of Assignment 2