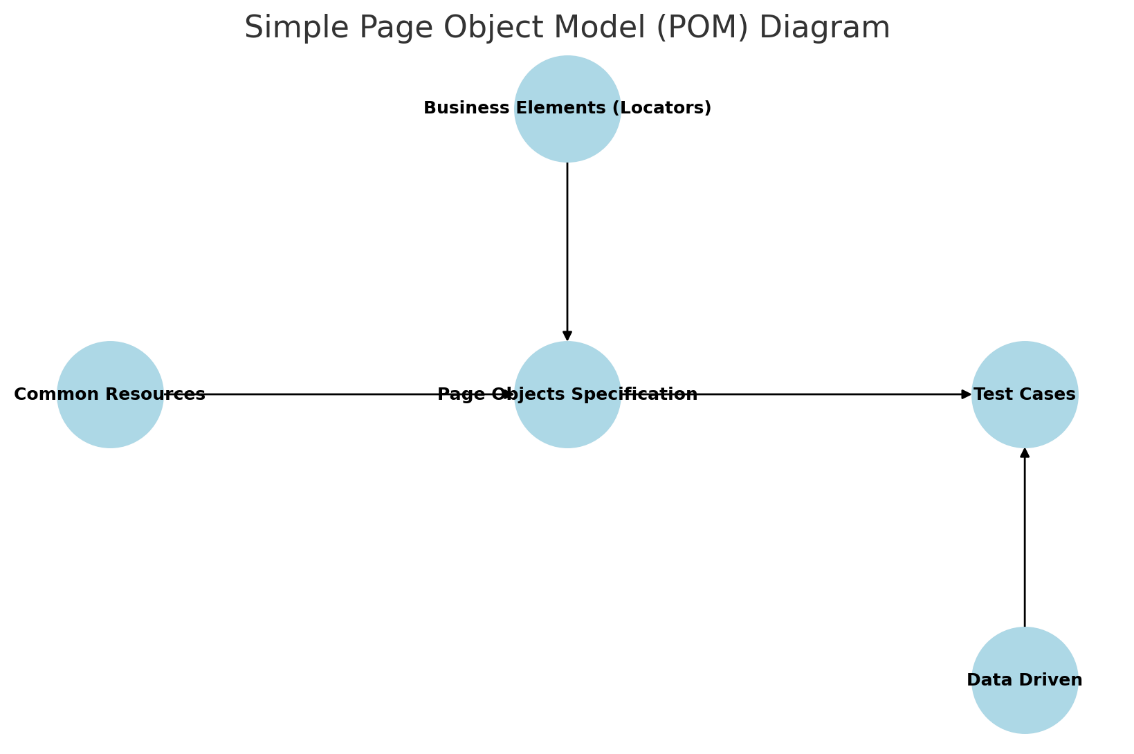
**Page Object Model (POM) Design Documentation**

**Overview**

The Page Object Model (POM) is a widely used design pattern in test automation. It helps in organizing UI test cases by separating business logic, page objects, and test execution. This ensures better maintainability, reusability, and scalability of test scripts.



A **simple Page Object Model (POM) diagram** illustrates the relationships between the different components:

* **Business Elements (Locators)** provide UI locators for interaction.
* **Common Resources** include reusable utilities.
* **Page Objects Specification** contains business logic and interactions.
* **Test Cases** use Page Objects for structured test execution.
* **Data Driven** supplies test data for automation.

**Components of POM**

**1. Business Elements (Locators)**

* Contains UI locators and identifiers.
* Helps in keeping all element locators in a structured manner.
* Example: XPath, CSS Selectors, ID, Class.

**2. Common Resources**

* Shared resources that can be used across multiple test cases.
* Example: Common login scripts, configurations, utilities.

**3. Page Objects Specification**

* Encapsulates business logic and interactions with UI components.
* Acts as an interface between business elements and test cases.
* Example: Functions/methods performing user actions like **clicking** a button, **filling forms**, etc.

**4. Test Cases**

* Contains actual test execution steps.
* Calls page objects and business elements for structured testing.
* Example: Validating login functionality, form submissions, and UI flows.

**5. Data Driven**

* Supports testing with multiple sets of data.
* Helps in parameterizing test cases for various input values.
* Example: Fetching input data from CSV, Excel, or databases.

**Benefits of Using POM**

* **Modularity:** Different components like locators, actions, and test logic are separated.
* **Reusability:** Page objects and business elements can be reused across multiple test cases.
* **Scalability:** Easy to extend test automation as UI changes.
* **Maintainability:** Changes in UI elements are reflected only in one place, reducing test script updates.

**Implementation Steps**

1. Define **locators** for all elements in a separate repository.
2. Create **Page Objects** containing all business logic and interactions.
3. Develop **Common Resources** to be shared across tests.
4. Write **Test Cases** that call page objects for execution.
5. Integrate **Data Driven** testing for various test scenarios.

**Conclusion**

The Page Object Model is an effective design pattern that enhances the efficiency and maintainability of test automation frameworks. By following this structured approach, test cases become easier to manage and update, leading to a robust automation strategy.