**Expense Tracker Basic UML Setup**

Here is the UML in plain text format:

**ExpenseTracker**

- database: Database

+ addExpense(expense: Expense): void

+ queryExpenses(criteria: Query): List<Expense>

+ setBudget(budget: Budget): void

+ generateReport(startDate: Date, endDate: Date): Report

+ getTopSpendingCategories(): List<Category>

**Expense**

- date: Date

- amount: Float

- category: String

- tag: String

+ Expense(date, amount, category, tag)

+ getDetails(): String

**Query**

- date: Date (optional)

- category: String (optional)

- tag: String (optional)

+ Query(date, category, tag)

+ matches(expense: Expense): Boolean

**Budget**

- frequency: String (e.g., monthly, weekly)

- amount: Float

+ Budget(frequency, amount)

+ isWithinLimit(expenses: List<Expense>): Boolean

**Report**

- startDate: Date

- endDate: Date

- totalExpenses: Float

- categoryBreakdown: Map<String, Float>

+ Report(startDate, endDate, totalExpenses, categoryBreakdown)

+ generateSummary(): String

**Database**

- connection: SQLiteConnection

+ connect(): void

+ executeQuery(query: String): List<Record>

+ insertRecord(table: String, data: Map<String, Object>): void

+ close(): void

**Formatting (Rough Draft)**

**Epic Requirement**

As a user, I want to track and analyze my spending so I can manage my budget effectively.

User Stories

**Add Expenses**

As a user, I want to add expenses with details (date, amount, category, and tag) so I can log my spending.

Example: add expense 2024/10/29 15.75 'Groceries' :FOOD

**Query Expenses**

As a user, I want to query expenses by date, category, or tag so that I can analyze my spending.

Example: query :FOOD retrieves all food-related expenses.

**Set Budget**

As a user, I want to set a budget (weekly or monthly) to manage my expenses effectively.

Example: set budget monthly 500

Generate Reports

As a user, I want to generate a report of my spending over a specific date range.

**Example: report 2024/01/01 2024/12/31**

**View Spending Priorities**

As a user, I want to view the top categories where I spend the most to understand my spending patterns.

**Example: priority**

**Simple Design**

Personal Budget and Expense Tracker

**Architecture Overview**

The application follows a modular design, adhering to OOP principles. It comprises key components for managing user input, processing business logic, and interacting with the SQLite database.

**Classes and Responsibilities**

1. Expense Tracker

- Central controller for the application.

- Manages user inputs, processes commands, and communicates with other components.

- **Key Methods**

- `addExpense(expense: Expense): void`

- `queryExpenses(criteria: Query): List<Expense>`

- `setBudget(budget: Budget): void`

- `generateReport(startDate: Date, endDate: Date): Report`

**Expense**

- Represents an individual expense.

- Attributes: `date`, `amount`, `category`, `tag`

- Key Methods:

- `getDetails(): String`

**Query**

- Handles user-specified search criteria.

- Attributes: **`date`, `category`, `tag`**

- Key Methods:

- `matches(expense: Expense): Boolean`

**Budget**

- Represents budget constraints set by the user.

-Attributes: `frequency` (weekly/monthly), `amount`

- Key Methods:

- `isWithinLimit(expenses: List<Expense>): Boolean`

**Report**

- Generates reports for user-specified date ranges.

- Attributes: `startDate`, `endDate`, `totalExpenses`, `categoryBreakdown`

- Key Method\*:

- `generateSummary(): String`

**Database**

- Handles interaction with the SQLite database.

- Attributes: `connection`

- Key Methods:

- `connect()`, `executeQuery(query: String): List<Record>`, `insertRecord(table: String, data: Map<String, Object>): void`, `close()`

**Database Design**

**Expenses Table**

- `id`: Primary Key

- `date`: Date of the expense

- `amount`: Float, amount spent

- `category`: String, category of expense

- `tag`: String, optional tag for expense

**Budgets Table**

- `id`: Primary Key

- `frequency`: String (weekly, monthly)

- `amount`: Float, budget limit

**Key Design Patterns**

1. Singleton

- Used for the `Database` class to ensure only one connection is active at a time.

2. Command

- Used to process and execute user commands like `add expense`, `query`, `set budget`.

3. Observer

- Used to trigger alerts if expenses exceed a budget limit.

**Interaction Workflow**

1. **Adding an Expense**

- User enters: `add expense <date> <amount> <category> <tag>`

- `ExpenseTracker` parses input and creates an `Expense` object.

- `Database` stores the expense record in the `Expenses` table.

2. **Querying Expenses**

- User enters: `query <criteria>`

- `ExpenseTracker` creates a `Query` object and retrieves matching expenses using `Database`.

3. **Generating a Report**

- User enters: `report <startDate> <endDate>`

- `ExpenseTracker` calculates totals and breakdowns using `Report` class.