1. Task Management System

Problem:
Design a task management application with the following features:

1. A `Task` class with properties: `id`, `title`, `description`, `status` (e.g., Pending, In Progress, Completed), and `dueDate`.
2. A `TaskManager` class that can:
    - Add new tasks.
    - Update task status.
    - List tasks grouped by status (Pending, In Progress, Completed).
    - Show overdue tasks (tasks with a `dueDate` in the past).
3. Use `List`, `Map`, or `Set` where appropriate.
4. Handle invalid inputs using `try-catch`.
5. Ensure null safety in properties and methods.

---

2. Online Quiz System

Problem:
Create a quiz system with the following requirements:

1. A `Question` class with attributes: `questionText`, `options` (list of options), `correctOptionIndex`, and `points`.
2. A `Quiz` class that can:
    - Add questions to the quiz.
    - Start the quiz and display each question with options.
    - Accept the user's answer and calculate the total score.
3. Allow users to retake the quiz until they achieve a passing score (use loops).
4. Use `async` to simulate fetching questions from a database.

---

3. Shopping Cart System

Problem:
Build a shopping cart system with the following functionality:

1. A `Product` class with attributes: `id`, `name`, `price`, and `stockQuantity`.
2. A `ShoppingCart` class to:
    - Add products to the cart (validate stock).

    - Remove products from the cart.
    - Calculate the total price.
3. Handle invalid inputs (e.g., adding an unavailable product) using `try-catch`.
4. Use `Map` to store cart items with quantities.
5. Simulate updating stock after checkout using a `Future` to mimic async operations.

---

4. Weather Monitoring Application

Problem:
Create an application to monitor weather conditions for multiple cities:

1. A `City` class with attributes: `name`, `temperature`, `condition` (e.g., Sunny, Rainy), and `lastUpdated` (DateTime).
2. A `WeatherMonitor` class to:
    - Add city weather data.
    - Update weather conditions for a city.
    - Display cities sorted by temperature.
    - Show cities with specific weather conditions (e.g., Rainy).
3. Use `async` to fetch new weather data (simulate using `Future.delayed`).

---

5. Employee Attendance System

Problem:
Design an attendance management system:

1. Create an `Employee` class with attributes: `id`, `name`, `designation`, and `attendance` (a map with Date as key and status as value).
2. Implement the following functionalities:
    - Mark attendance (Present/Absent).
    - View attendance history for an employee.
    - Calculate the percentage of attendance for each employee.
3. Use `Set` to ensure no duplicate employee IDs.
4. Handle invalid data or operations using `try-catch`.

---

6. Bank Transaction System

Problem:
Develop a bank application with the following features:

1. A `BankAccount` class with attributes: `accountNumber`, `accountHolderName`, `balance`, and methods:
    - `deposit(double amount)`
    - `withdraw(double amount)`
    - `checkBalance()`
2. Use inheritance to create `SavingsAccount` (with interest calculation) and `CurrentAccount` (with overdraft limit).
3. Simulate real-time transactions using async methods (e.g., delay deposit confirmation).
4. Handle invalid operations (e.g., insufficient balance) using `try-catch`.

---

7. Travel Booking System

Problem:
Build a travel booking system with the following:

1. A `Trip` class with attributes: `destination`, `price`, `availableSeats`, and `departureDate`.
2. A `Booking` class to manage:
    - Booking a trip (validate available seats).
    - Canceling a booking.
    - Displaying trip details.
3. Use a `List` to store trips and a `Map` to store user bookings.
4. Use `async` to simulate fetching trip data and validating bookings.

---

8. Multi-Layered Calculator

Problem:
Design a calculator with basic and advanced functionality:

1. Create a `BasicCalculator` class with methods: `add`, `subtract`, `multiply`, `divide`.
2. Extend the functionality with an `AdvancedCalculator` class that includes:
    - Trigonometric functions (sin, cos, tan).
    - Exponentiation and square root.
3. Handle invalid inputs (e.g., division by zero) using `try-catch`.
4. Allow users to select functions dynamically using a `switch` case.

9. Library Management System

Problem:
Design a system for managing a library:

1. Create a `Book` class with attributes: `id`, `title`, `author`, `isAvailable`.
2. Create a `Library` class that can:
    - Add new books.
    - Borrow books (change `isAvailable` to false).
    - Return books (change `isAvailable` to true).
    - List all available books.
    - Search for books by title or author.
3. Use `Map` to store books with their `id` as the key.
4. Handle invalid operations (e.g., borrowing unavailable books) using `try-catch`.

Topics Covered:

- OOP (classes, encapsulation)
- Map, functions, and loops
- Null safety and error handling

---

10. Expense Tracker Application

Problem:
Build an expense tracker with the following features:

1. A `Transaction` class with attributes: `id`, `title`, `amount`, `date`, and `category` (e.g., Food, Transport).
2. A `Tracker` class to:
    - Add transactions.
    - Display transactions grouped by category.
    - Calculate total expenses for a given month.
3. Use a `List` to store transactions and filter them based on date and category.
4. Use `async` to simulate fetching historical transaction data from a server.

Topics Covered:

- OOP (composition, encapsulation)
- List and Map
- Functions, loops, and conditionals
- Async programming

---

11. Online Food Ordering System

Problem:
Create a food ordering system with these requirements:

1. A `MenuItem` class with attributes: `id`, `name`, `price`, and `isAvailable`.
2. An `Order` class to:
    - Add menu items to an order (validate availability).
    - Remove items from the order.
    - Display the order summary (total price and items).
3. Use a `Set` to store ordered items.
4. Handle scenarios where unavailable items are added using `try-catch`.
5. Use `Future` to simulate order processing.

Topics Covered:

- OOP (classes, composition)
- Set and List
- Null safety and try-catch
- Async programming

---

12. Smart Home Controller

Problem:
Design a smart home controller application:

1. Create a `Device` class with attributes: `id`, `name`, `status` (On/Off).
2. A `SmartHome` class to manage devices:
    - Turn devices on/off.
    - List all devices and their statuses.
    - Filter devices based on their status.
3. Use `List` to store devices.
4. Simulate fetching device data and updating statuses using `Future` and `async`.

Topics Covered:

- OOP (classes, encapsulation)
- List and functions
- Async programming
- Loops and conditionals

---

13. Student Grading System

Problem:
Develop a student grading system:

1. A `Student` class with attributes: `id`, `name`, `marks` (Map of subjects to scores).
2. A `GradingSystem` class to:
    - Add students and their marks.
    - Calculate average marks for each student.
    - Assign grades based on the average (A, B, C, etc.).
    - List students with grades in descending order.
3. Handle invalid data (e.g., marks out of range) using `try-catch`.

---

14. Event Management System

Problem:
Create an event management system:

1. An `Event` class with attributes: `id`, `name`, `date`, `location`, and `participants` (a Set of names).
2. A `EventManager` class to:
    - Create new events.
    - Add participants to an event (avoid duplicates).
    - List upcoming events (filter by date).
3. Use `async` to simulate sending notifications to participants about their events.

---

15. Inventory Management System

Problem:
Develop an inventory management system:

1. A `Product` class with attributes: `id`, `name`, `quantity`, and `price`.
2. An `Inventory` class to:
    - Add new products to inventory.
    - Update product quantity.
    - Calculate the total value of inventory.

3. Use `Map` to store products with their `id` as the key.
4. Handle invalid operations (e.g., updating non-existent products) using `try-catch`.

---

16. Movie Booking System

Problem:
Create a movie booking application:

1. A `Movie` class with attributes: `id`, `title`, `availableSeats`, and `showTime`.
2. A `Booking` class to:
    - Book tickets for a movie (validate available seats).
    - Display booking details.
3. Use a `List` to store movies and a `Map` to store user bookings.
4. Handle errors (e.g., booking unavailable seats) using `try-catch`.

---

17. Quiz Leaderboard System

Problem:
Build a quiz leaderboard with the following features:

1. A `Player` class with attributes: `id`, `name`, and `score`.
2. A `Leaderboard` class to:
    - Add players and their scores.
    - Display the top 3 players.
    - Allow players to update their scores.
3. Use `List` to store players and sort them by scores.

18. Online Shopping Cart

Problem:
Design an online shopping cart system:

1. A `Product` class with attributes: `id`, `name`, `price`, and `quantity`.
2. A `Cart` class to:

    - Add products to the cart.
    - Remove products from the cart.
    - Calculate the total price of the cart.
    - Apply a discount code if provided (e.g., "SAVE10" gives 10% off).
3. Use a `List` to store products in the cart.
4. Handle scenarios where invalid discount codes are used using `try-catch`.

---

19. Weather Application

Problem:
Create a weather application with these features:

1. A `WeatherData` class with attributes: `city`, `temperature`, `condition` (e.g., Sunny, Rainy).
2. A `WeatherApp` class to:
    - Fetch weather data asynchronously for a given city.
    - Display the current weather.
    - Allow users to add multiple cities to a favorite list (use `Set` to avoid duplicates).
3. Use `Future` to simulate fetching weather data from an API.

---

20. Banking System

Problem:
Develop a banking system:

1. An `Account` class with attributes: `id`, `name`, `balance`.
2. A `Bank` class to:
    - Create accounts.
    - Deposit and withdraw money (validate sufficient balance).
    - Transfer money between accounts.
3. Use `Map` to store accounts with their `id` as the key.
4. Handle errors such as invalid accounts or insufficient balance using `try-catch`.

---

21. To-Do Application

Problem:
Design a to-do application:

1. A `Task` class with attributes: `id`, `description`, `isCompleted`.
2. A `ToDoList` class to:
    - Add tasks.
    - Mark tasks as completed.
    - List all pending tasks.
    - Delete tasks.
3. Use a `List` to store tasks and filter them based on their completion status.
4. Simulate saving tasks to a file using `Future`.

---

22. Airline Reservation System

Problem:
Build an airline reservation system:

1. A `Flight` class with attributes: `id`, `destination`, `departureTime`, and `availableSeats`.
2. A `Reservation` class to:
    - Book a flight (validate seat availability).
    - Display reservation details.
3. Use a `List` to store flights and a `Map` to store reservations.
4. Simulate fetching flight data from an API using `Future`.

---

23. Quiz Game

Problem:
Create a quiz game with these features:

1. A `Question` class with attributes: `id`, `questionText`, `options` (List), and `correctAnswer`.
2. A `QuizGame` class to:
    - Load questions asynchronously.

    - Allow users to answer questions and keep track of their score.
    - Display the final score at the end of the quiz.
3. Use `List` to store questions and filter incorrect answers.

---

24. E-Learning Platform

Problem:
Develop an e-learning platform:

1. A `Course` class with attributes: `id`, `title`, `description`, and `price`.
2. A `Student` class to:
    - Enroll in courses.
    - View enrolled courses.
    - Calculate the total cost of all enrolled courses.
3. Use a `Set` to store enrolled courses to avoid duplicates.
4. Simulate course purchase confirmation with `Future`.

---

25. Hospital Management System

Problem:
Design a hospital management system:

1. A `Patient` class with attributes: `id`, `name`, `age`, `disease`.
2. A `Doctor` class with attributes: `id`, `name`, `specialty`, and `patients` (List of `Patient`).
3. A `Hospital` class to:
    - Assign patients to doctors based on their specialty.
    - List all patients of a specific doctor.
    - Display all available doctors.
4. Use `Map` to store doctors with their `id` as the key.

---

26. Expense Management App

Problem:
Create an expense management application:

1. An `Expense` class with attributes: `id`, `category`, `amount`, and `date`.
2. An `ExpenseManager` class to:
    - Add new expenses.
    - Group expenses by category.
    - Calculate total expenses for a given month.
3. Use a `Map` to group expenses by category and a `List` for the overall expense list.
4. Use async to simulate fetching expenses from a database.