Tab 1

1. Savings Goal Calculator using Loops and Logical Conditions

- **Task**: Create a **savings goal calculator** that calculates how many months it will take to reach a savings goal based on the monthly deposit and interest rate.
    - Input:
        - The savings goal (target amount). 10000000000000
        - The initial balance (starting amount saved). 1000
        - The monthly deposit. 5
        - The interest rate (as a percentage). 10
    - The program should calculate the total balance at the end of each month by adding the monthly deposit and applying the interest to the current balance.
    - Use a loop to continue the calculation until the savings goal is met.
    - If at any point the monthly deposit is too low (due to a very high savings goal or low interest), notify the user that the goal cannot be reached in a realistic timeframe.

Instructions:

- Use a **while loop** to calculate the balance each month, considering the interest and monthly deposit.
- Implement logic to handle cases where the savings goal might be unrealistic given the monthly deposit and interest rate.
- Print the balance at the end of each month and the number of months it took to reach the goal.

Example Output:

```
Enter savings goal: 100000
Enter initial balance: 10000
Enter monthly deposit: 4000
Enter interest rate (annual %): 4
Month 1: Total savings = 14133
Month 2: Total savings = 18300
...
Goal achieved in 20 months.
```

Tab 2

2. Vacation Fund Calculator using Loops and Logical Conditions

- **Task**: Create a **vacation fund calculator** that calculates how many months it will take to save enough money for a vacation based on the initial balance, monthly savings, and annual interest rate.
    - **Input**:
        - The vacation fund target amount (goal).
        - The initial amount saved.
        - The monthly savings contribution.
        - The annual interest rate (as a percentage).
    - The program should calculate the total fund balance at the end of each month by adding the monthly savings and applying the interest to the current balance.
    - Use a loop to continue the calculation until the vacation fund target is reached.
    - If at any point the monthly savings are too low (e.g., due to a high target amount or low interest rate), notify the user that the goal cannot realistically be met in the given conditions.

Instructions:

- Use a **while loop** to calculate the fund balance each month, factoring in interest and monthly savings.
- Implement logic to check if the savings plan is realistic given the goal, monthly savings, and interest rate.
- Print out the balance at the end of each month and the number of months it took to reach the vacation fund target.

Example Output:

```
Enter vacation fund target: 50000
Enter initial balance: 5000
Enter monthly savings: 2000
Enter interest rate (annual %): 4
Month 1: Current fund balance = 7083.33
Month 2: Current fund balance = 9170.14
...
Goal achieved in 18 months.
```

Tab 3

3. Tax Slab Calculator for Income Groups

Scenario:
Create a tax calculator for individuals. The user enters their income range (as a string) and the program calculates the applicable tax percentage.

Requirements:

- Use a `String` as the `switch` expression.
- Allow inputs like `"Low"`, `"Medium"`, `"High"`, `"Very High"`.
- Provide a `default` case for invalid inputs.

Income Categories:

- `Low` (≤ 2,50,000): Tax-free
- `Medium` (2,50,001 – 5,00,000): 5% tax
- `High` (5,00,001 – 10,00,000): 20% tax
- `Very High` (> 10,00,000): 30% tax

Expected Input:

```
Enter your income category (Low/Medium/High/Very High): Medium
```

Expected Output:

```
yaml
Income Category: Medium
Tax Percentage: 5%
```

Hints:

- Use strings like `"Low"`, `"Medium"`, `"High"`, `"Very High"` for cases.
- Implement a method to handle invalid inputs.

Tab 4

4. Access Control System

Scenario:
Build a role-based access control system where the user inputs their role (e.g., `Admin`, `Manager`, `Employee`, `Guest`), and the program displays their access rights. Additionally, handle multiple roles by accepting comma-separated input.

Requirements:

- Use a `String` as the `switch` expression.
- Implement a loop to parse comma-separated roles.
- Combine multiple roles dynamically to calculate access levels.

Details:

- `Admin`: Full access (Read, Write, Execute)
- `Manager`: Moderate access (Read, Write)
- `Employee`: Limited access (Read)
- `Guest`: View-only access

Expected Input:

```
Enter your role(s): Admin, Employee
```

Expected Output:

```
Access Rights: Full access, Limited access
```

Hints:

- Use `.split(",")` to handle multiple roles.
- Implement a nested loop to evaluate each role.

Tab 5

5. Multi-Language Greeting System

Scenario:
Create a multi-language greeting system. The user inputs their preferred language (e.g., `English`, `Spanish`, `French`, etc.) and the time of day (`Morning`, `Afternoon`, `Evening`, `Night`). The program displays a greeting in the specified language and time of day.

Requirements:

- Use `String` for both language and time of day in nested `switch` statements.
- Provide a `default` case for unsupported inputs.

Details:
For `English`:

- `Morning`: Good Morning
- `Afternoon`: Good Afternoon
- `Evening`: Good Evening
- `Night`: Good Night

For `Spanish`:

- `Morning`: Buenos días
- `Afternoon`: Buenas tardes
- `Evening`: Buenas noches
- `Night`: Buenas noches

Expected Input:

```
Enter language: Spanish
Enter time of day: Morning
```

Expected Output:

```
Greeting: Buenos días
```

Hints:

- Handle multiple languages using nested `switch`.
- Add a `default` case for unsupported languages or times.

Tab 6

6. Discount Calculator with Membership Levels

Scenario:
Create a discount calculator for an e-commerce platform. The user inputs a product category (`Electronics`, `Clothing`, `Groceries`) and membership type (`Gold`, `Silver`, `None`). The program calculates the final discount.

Requirements:

- Use a `String` for both category and membership type.
- Handle nested discounts based on membership levels.
- Provide combined discounts for special memberships.

Details:

- `Electronics`: 10%
- `Clothing`: 20%
- `Groceries`: 5%
- Membership Discounts:
    - `Gold`: Additional 10%
    - `Silver`: Additional 5%

Expected Input:

```
Enter category: Electronics
Enter membership type: Gold
Enter price: 20000
```

Expected Output:

```
yaml
Base Discount: 10%
Membership Discount: 10%
Final Price: 16,000
```

Hints:

- Use nested `switch` statements for category and membership type.
- Calculate total discounts iteratively.

Tab 7

7. ATM Simulation with Daily Withdrawal Limit

Scenario: Build an ATM system where the user performs multiple transactions until they choose Exit. ATM enforces a daily withdrawal limit.

Input:

- Initial balance
- Daily withdrawal limit
- Menu choice repeatedly:
    - 1 Withdraw
    - 2 Deposit
    - 3 Balance
    - 0 Exit

Requirements / Instructions:

- Use `do-while` to keep showing the menu until Exit.
- Use `switch` for menu handling.
- Withdrawal rules:
    - amount must be positive
    - amount must be <= balance
    - total withdrawn today must be <= daily limit
- Print updated balance after each transaction.

Expected Input:

```
Initial balance: 5000
Daily limit: 2000
Choice: 1
Withdraw amount: 1500
Choice: 1
Withdraw amount: 700
Choice: 0
```

Expected Output:

```
Withdrawal successful. Balance: 3500
Withdrawal denied: Daily limit exceeded.
Exiting...
```

Hints:

- Maintain a variable `withdrawnToday`.
- Validate amount before updating balance.

Tab 8

8. Mobile Data Usage Monitor

Scenario: A user wants to track mobile data usage for 7 days and check whether they exceeded the weekly limit.

Input:

- Weekly data limit (GB)
- Daily usage array for 7 days (GB)

Requirements / Instructions:

- Use an array of size 7 to store daily usage.
- Use a `for` loop to compute:
    - total usage
    - average usage
    - highest usage day (index)
- If total exceeds limit, calculate extra charge:
    - extraCharge = (total - limit) * ratePerGB (fixed in code)

Expected Input:

```
Weekly limit: 10
Daily usage: 1.2 1.5 0.8 2.0 1.9 1.1 2.3
```

Expected Output:

```
Total Usage: 10.8 GB
Average Usage: 1.54 GB/day
Highest Usage Day: Day 7
Extra Charge: 40.0
```

Hints:

- Use `double` for usage.
- Track `maxUsage` and `maxDay`.

Tab 9

9. Cinema Ticket Booking with Seat Availability (Array + Loop)

Scenario: A cinema has 10 seats (1–10). Some are already booked. User tries to book seats.

Input:

- Array of seats (0 = available, 1 = booked)
- User enters seat number repeatedly
- User enters ticket type: Regular / Premium

Requirements / Instructions:

- Use `while` loop to book seats until user types `0` to stop.
- Use `switch` for ticket type pricing.
- If seat is already booked, print message and ask again.
- Print total tickets booked and total cost.

Expected Input:

```
Ticket type: Premium
Seat: 4
Seat: 4
Seat: 7
Seat: 0
```

Expected Output:

```
Seat 4 booked successfully.
Seat 4 is already booked. Choose another.
Seat 7 booked successfully.
Total tickets: 2
Total cost: 600
```

Hints:

- Convert seat number to index: seat-1.
- Use array to update seat status.

Tab 10

10. Electricity Bill Calculator (Slab + Validation)

Scenario: A utility company calculates monthly bill based on units consumed.

Input:

- Units consumed

Requirements / Instructions:

- Use `if-elseif-else` for slab billing:
    - 0–100 units: 1.5 per unit
    - 101–200 units: 2.0 per unit
    - 201–500 units: 3.0 per unit
    - 500 units: 5.0 per unit
- Add a fixed charge of 50.
- Reject invalid units (<0).

Expected Input:

- `Units: 275`

Expected Output:

- `Units: 275`
- `Bill Amount: 725.0`

Hints:

- Use incremental slab calculation (not one-rate for all).
    -

Tab 11

11. Student Result Card (Marks Array + Grade + Pass/Fail)

Scenario: Student enters marks for 5 subjects and gets a result card.

Input:

- Marks array of 5 subjects

Requirements / Instructions:

- Use array + loop to compute total and percentage.
- Pass rule: each subject >= 35.
- Grade based on percentage:
    - =90 A
    - =80 B
    - =70 C
    - =60 D
    - else F
- Print subject-wise marks + total + percentage + grade + PASS/FAIL.

Expected Output:

- `Total: 378`
- `Percentage: 75.6`
- `Result: PASS`
- `Grade: C`

Hints:

- Use boolean flag `allPassed`.
    -

Tab 12

12. Order Processing with Shipping Charges (Category + Weight)

Scenario: A delivery system calculates shipping based on category and package weight.

Input:

- Category: Standard / Express / SameDay
- Weight in kg

Requirements / Instructions:

- Use `switch` on category to set base rate per kg.
- Use `if-else` to add surcharge:
    - weight > 10kg adds additional flat fee
- Validate weight >0.

Expected Output:

- `Category: Express`
- `Weight: 12`
- `Shipping Cost: 540`

Hints:

- Keep baseRate fixed per kg for each category.
-

Tab 13

13. Monthly Budget Tracker (Expense Categories Array)

Scenario: Track expenses across categories and detect overspending.

Input:

- Budget amount
- Expenses for 6 categories (array): Rent, Food, Travel, Shopping, Bills, Other

Requirements / Instructions:

- Use array to store expenses.
- Compute total expense and remaining budget.
- Print category that has maximum expense.
- If overspent, print overspent amount.

Expected Output:

```
Total Expense: 32000
Remaining: -2000
Overspent by: 2000
Max Expense Category: Rent
```

Hints:

- Keep category names in a parallel string array.

Tab 14

14. Password Retry System (Do-While + Attempts)

Scenario: User must enter correct PIN within 3 attempts.

Input:

- Hardcode a PIN (e.g., 4321)
- User enters PIN up to 3 times

Requirements / Instructions:

- Use `do-while` loop with attempt counter.
- If correct → "Access Granted"
- If 3 wrong attempts → "Account Locked"

Expected Output:

```
Attempt 1: Wrong
Attempt 2: Wrong
Attempt 3: Wrong
Account Locked
```

Hints:

- Stop early if correct.

Tab 15

15. Bus Ticket Fare System (Age + Trip Type + Switch)

Scenario: Fare depends on route type and passenger age.

Input:

- Route type: City / Intercity / Night
- Age
- Base fare

Requirements / Instructions:

- Use `switch` to apply route multiplier:
    - City: 1.0
    - Intercity: 1.5
    - Night: 2.0
- Use `if-elseif` for age discount:
    - <12: 50% off
    - 12–59: no discount
    - =60: 30% off
- Print final fare.

Hints:

- Apply multiplier first then discount.

Tab 16

16. Inventory Reorder Alert (Array + Threshold + Report)

Scenario: Store tracks stock of 8 products and wants a reorder report.

Input:

- Stock array (8 integers)
- Reorder threshold

Requirements / Instructions:

- Use loop to list items below threshold.
- Count how many require reorder.
- Print “All Good” if none.

Expected Output:

```
Items to reorder: 3
Product 2: Stock=4
Product 6: Stock=1
Product 8: Stock=3
```

Hints:

- Products can be numbered 1–8.

Tab 17

17. Exam Slot Booking (Switch + Array of Slots)

Scenario: Student chooses an exam slot. Each slot has limited seats.

Input:

- Slots: Morning, Afternoon, Evening (array seats: 5 each)
- User chooses slot repeatedly until exit

Requirements / Instructions:

- Use `switch` to map choice to slot index.
- If slot full, print “Slot Full”.
- Use loop until user enters Exit.

Expected Output:

```
Booked Morning. Remaining seats: 4
Booked Morning. Remaining seats: 3
...
Morning slot is full.
```

Hints:

- Store seats in an int array: `int[] seats = {5,5,5}`.

Tab 18

18. Vote Counter System (Array + Winner + Tie)

Scenario: A club election has 4 candidates. Collect votes and announce result.

Input:

- Number of votes `n`
- Each vote is candidate number 1–4 (invalid allowed)

Requirements / Instructions:

- Use an array `counts[4]`.
- Count invalid votes separately.
- Print votes per candidate.
- Print winner or tie.

Expected Output:

```
Candidate 1: 3
Candidate 2: 5
Candidate 3: 5
Candidate 4: 2
Invalid: 1
Result: Tie between Candidate 2 and Candidate 3
```

Hints:

- Track max count and tie detection using loop.