Inheritance Problem Statements

🧩 1. Single Inheritance - Prime Pattern Puzzle

Scenario:
A base class `NumberSeries` generates a list of numbers based on a mathematical pattern (e.g., triangular numbers). A derived class `PrimeChecker` checks if those numbers are prime and returns a filtered list of only prime ones from the pattern.

Challenge:
Let the user input `n`, and print all prime numbers among the first `n` triangular numbers.

```
// Output for n = 5 → Triangular Numbers: 1, 3, 6, 10, 15 → Primes: 3
```

🧠 2. Multiple Inheritance - LCM & GCD Logic

Scenario:
You have two base classes: `GCDCalculator` and `LCMCalculator`. Each provides a method to compute GCD and LCM respectively.
A derived class `NumberAnalyzer` inherits from both and must find:

- All pairs `(i, j)` between two ranges such that LCM(i, j) / GCD(i, j) is a **perfect square**.

Challenge:
Given ranges: [a, b] and [c, d], find such pairs and count them.

```
// Use math to check for perfect square: sqrt(x) == floor(sqrt(x))
```

🧬 3. Multilevel Inheritance - Sequence Transformer

Scenario:

- Class `Sequence` generates a basic arithmetic sequence.
- Class `Modifier` multiplies each term by its index.
- Class `Reverser` (derived from `Modifier`) reverses the final sequence and sums the digits of each number.

Challenge:
Input: start = 1, diff = 2, terms = 5
Generate: 1 3 5 7 9
Multiply by index: 1*1, 3*2, 5*3, 7*4, 9*5 → 1 6 15 28 45
Reverse: 45 28 15 6 1
Output: Sum of digits of each → 9 10 6 6 1

🌳 4. Hierarchical Inheritance - Geometry Logic Tree

Scenario:
Base class `Shape` contains basic data like dimensions.
Derived classes:

- `Rectangle` calculates area and perimeter.
- `Triangle` calculates area using Heron's formula.
- `Circle` calculates area and circumference.

Challenge:
Let user choose shape, input dimensions, and compute area AND check if it's a **perfect number** (equal to sum of its proper divisors).

```
// e.g., Area = 28 → divisors = 1+2+4+7+14 = 28 → Perfect
```

🧪 5. Hybrid Inheritance - Matrix Analyzer

Scenario:

- Class `MatrixInput` takes a matrix from user.
- Class `RowOperations` (inherits MatrixInput) finds the row with max sum.
- Class `ColumnOperations` (inherits MatrixInput) finds column with max product.
- Class `Analyzer` inherits both and computes GCD of those two values (max row sum and max col product).

Challenge:
Input a matrix (2D vector), find the row with highest sum, column with highest product, and compute their GCD using logic from base classes.

```
// Input:
// 1 2 3
// 4 5 6
// 7 8 9
// Max row sum: 24 (row 3), Max col product: 252 (col 3) → GCD = 12
```
