Exercise 1: The Temperature Logger (Basic)

Goal: Create a class WeeklyTemp that stores 7 decimal values (one for each day of the week).

- Implement an integer indexer.
- Add logic in the set accessor to ensure temperatures are within a realistic range (e.g., between **-50°C** and **60°C**). If the value is outside this range, do not update the data.

Exercise 2: The Multi-Type Indexer (Overloading)

Goal: Create a class ContactList that holds a private array of names (strings).

- Implement an indexer that takes an int to get/set a name by its position.
- **Overload** the indexer: Implement a second indexer that takes a string (a name) and returns the int index of that name if it exists, or **-1** if it doesn't.

Exercise 3: The Dictionary-Style Indexer (String Keys)

Goal: Create a class SimpleDictionary. Internally, use two arrays: string[] keys and string[] values.

- Implement an indexer that takes a string key.
- The get accessor should look for the key and return the associated value.
- The set accessor should update the value if the key exists, or add a new key/value pair if there is space.

Exercise 4: The 2D Grid (Multi-parameter Indexer)

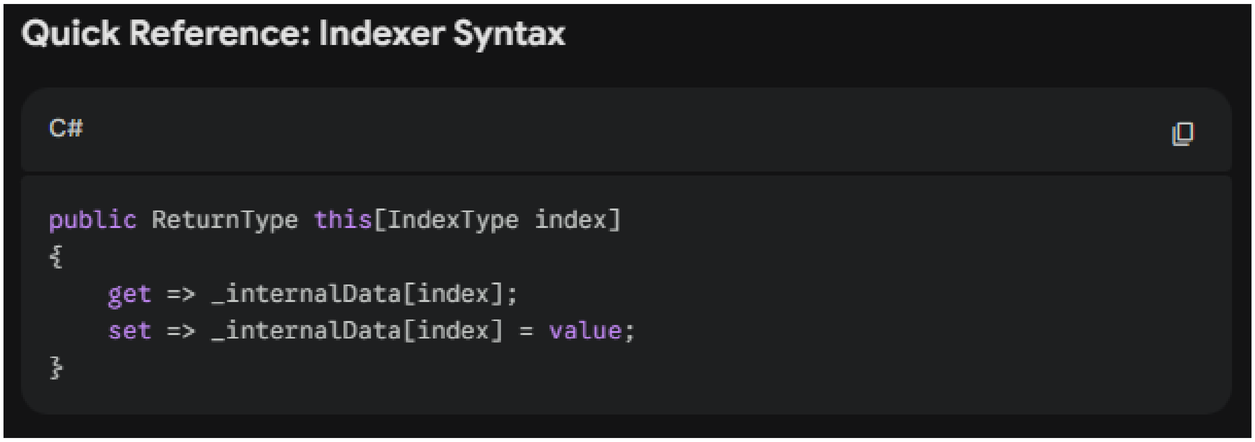
Goal: Create a class GameGrid that represents a 2D map.

- Define a private 2D array: char[,] grid = new char[10, 10];.
- Implement an indexer that takes **two** integers: this[int x, int y].
- The indexer should allow you to place a character (like 'X' or 'O') at specific coordinates: myGrid[5, 2] = 'X';.

Exercise 5: Read-Only Bit Flapper (Advanced)

Goal: Create a class BitStatus that wraps around a single int value.

- An int is 32 bits. Implement a **read-only** indexer this[int bitIndex].
- The indexer should return a bool: true if the bit at that position is 1, and false if it is 0.
- *Hint:* Use the bitwise AND operator & and the shift operator << to check the bit status.



Quick Reference: Indexer Syntax

```csharp
public ReturnType this[IndexType index]
{
    get => _internalData[index];
    set => _internalData[index] = value;
}
```