>>> network .toCode()

Patterns and Nomenclature

Imperative

- Definition: A programming paradigm that uses statements to change a program's state.
- Real-world Analogy: Ask your friend listening to Bob Ross tell them how to paint a landscape. While good ole Bob Ross isn’t exactly commanding, he is giving them step by step directions to get the desired result.
- Characteristics:
    - Focus on "how-to" steps.
    - Explicit sequence of commands.
    - Loops, conditionals, and assignments are typical constructs.

>>> Idempotent

- Definition: An operation is idempotent if performing it multiple times yields the same result as performing it once.
- Real-world Analogy: Pushing an elevator button; whether you press it once or many times, the elevator still comes once.
- HTTP Methods: GET, PUT, and DELETE are typically idempotent in web APIs.
- Benefits:
  - Safe retries: Repeating the operation won't cause unintended side effects.
  - Predictability: Consistent outcomes regardless of repetitions.
  - System Stability: Helps in distributed systems where a request might be sent multiple times due to network hiccups.

>>> Declarative

- Definition: A programming paradigm that expresses the logic without describing its control flow.
- Real-world Analogy: Declarative Programming is like asking your friend to draw a landscape. You don’t care how they draw it, that’s up to them.
  - Imperative Programming is like your friend listening to Bob Ross tell them how to paint a landscape. While good ole Bob Ross isn’t exactly commanding, he is giving them step by step directions to get the desired result.
- Common Examples: Terraform, SALT
- Characteristics:
  - Focus on "what" the end state is vs "how" to get to that state.
  - Often more concise and readable.
  - Abstracts away the underlying processes.

DRY / WET

- Definition: A software development principle urging one to reduce repetition of code patterns.
- Real-world Analogy: If you have a base recipe used in multiple dishes (e.g., a tomato sauce), you wouldn't write it out in full for every dish. Instead, you'd reference it.
- Contrast: WET (Write Everything Twice or We Enjoy Typing) – characterized by repeated or redundant code.
- Benefits:
  - Easier maintenance: Change code in one place, not multiple.
  - Reduces errors: Fewer places to introduce bugs.
  - Enhances clarity: More readable and understandable code.

Dispatcher Method

- Definition: A method responsible for directing requests or tasks to appropriate handlers or methods based on certain criteria.
- Real-world Analogy: A receptionist at an office who directs visitors to the right department or person based on their needs.
- Contrast: A series of if, elif conditionals.
- Benefits:
    - Centralized Control: One place to manage requests, making modifications easier.
    - Flexibility: Easily add, modify, or remove handlers without disrupting the whole system.
    - Decoupling: Separation of concerns, where the dispatcher and handlers are independent.

Class Inheritance

- Definition: A mechanism where a new class derives properties and behaviors (methods) from an existing class.
- Real-world Analogy: A child inheriting traits from their parents, such as eye color or height.
- Contrast: Composition, where an object is used to achieve functionality, rather than inheriting it.
- Benefits:
    - Code Reusability: Use existing functionalities and extend or modify them.
    - Organizational: Grouping shared attributes and behaviors, promoting a logical structure.
    - Easier Maintenance: Changes in the parent (base) class can propagate to child (derived) classes.

Principle of Least Astonishment

- Definition: Design principle stating that a system should behave in a manner consistent with how users expect it to, minimizing their surprise.
  - People are part of the system. The design should match the user’s experience, expectations, and mental models.
  - A component of the principle means that a component of a system should behave in a way that most users will expect it to behave; the behavior should not astonish or surprise users.
- Real-world Analogy: Light switches in rooms; typically, the nearest switch controls the room's main light.
- Contrast: Systems or interfaces that are non-intuitive or have unexpected side effects.
- Benefits:
  - Enhanced Usability: Users find the system more intuitive and easier to navigate.
  - Reduced Errors: Predictable behavior leads to fewer mistakes.
  - Faster Learning Curve: New users can onboard more quickly.

Robustness principle (Postel's Law)

- Definition: "Be conservative in what you send, be liberal in what you accept."
- Real-world Analogy: A tolerant teacher who accepts various answers from students as long as they're close, but only teaches one standardized method.
- Contrast: Strict enforcement principles, where only a rigid set of inputs is accepted and any deviation leads to failure.
- Benefits:
  - Greater Compatibility: Systems can interoperate with a wider range of partners or components.
  - Enhanced Longevity: As standards evolve, older implementations remain usable.
  - Resilience: Systems can handle unexpected inputs without crashing.

State Machine

- Definition: A computational model designed to be in one of a finite number of states at any given time, transitioning between them based on inputs.
- Real-world Analogy: Traffic lights cycling through colors: red, green, and yellow.
- Contrast: General-purpose scripts or algorithms that don't have clearly defined states or predictable transitions.
- Benefits:
  - Predictable Behavior: Clearly defined states and transitions.
  - Easy to Model: Simplifies complex processes by breaking them down into states.
  - Enhances Debugging: Faults can often be traced to specific state transitions.

>>> network .toCode()

Thanks