Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Design Patterns and SOLID with TypeScript

Class Duration

21 hours of live training delivered over 3 days.

Student Prerequisites

- Professional experience writing TypeScript
- Comfort with functions, classes, and modules in TypeScript
- Familiarity with React or Node.js is helpful but not required
- No prior design patterns knowledge required

Target Audience

This course is designed for working TypeScript developers who want to make better design decisions in a language that blends object-oriented and functional styles. It suits engineers maintaining growing front-end or Node.js codebases, developers who learned patterns from class-heavy material and want a treatment that fits TypeScript's structural type system, and team leads who review code — including code produced by AI assistants — and need a shared vocabulary for evaluating structure. Participants should write TypeScript professionally; no prior patterns background is assumed.

Description

TypeScript sits between two design traditions: the class-and-interface world the Gang of Four catalog was written for, and the functional, module-oriented world of modern JavaScript. This course teaches SOLID and the GoF catalog the way they should be taught in TypeScript 6.x: selectively and honestly, covering the roughly fifteen patterns that still earn their keep in contemporary code — split across creational, structural, and behavioral families — and showing where structural typing, discriminated unions, first-class functions, and ES modules change or eliminate the classic implementation.

The course is built around refactoring rather than recitation. Each pattern is motivated by a real code smell, introduced as the refactoring that resolves it, and weighed against simpler alternatives so participants learn when a pattern is overengineering. TypeScript-specific design gets equal billing with the classics: structural typing's impact on interface-based patterns, union

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

and discriminated-union alternatives to class hierarchies, generics and conditional types in pattern design, modules versus Singleton, functional composition versus inheritance, Stage 3 decorators, and the dependency injection landscape in the TS ecosystem. Patterns are grounded in the places TypeScript actually runs, with worked examples from React and Node.js codebases. The final day addresses design in the age of AI-generated code: recognizing pattern use (and misuse) in agent output, reviewing generated designs against SOLID, and steering coding assistants toward structures your team can maintain.

Learning Outcomes

- Apply each SOLID principle to real TypeScript code and recognize when a principle is being over-applied
- Identify the design problem each major GoF pattern solves and select among creational, structural, and behavioral options
- Explain how structural typing changes interface-based patterns and exploit it for lightweight seams
- Replace class hierarchies with discriminated unions and exhaustive switches where they are clearer
- Use generics, conditional types, and mapped types deliberately in pattern design without descending into type golf
- Use ES modules in place of Singleton and evaluate module-level state honestly
- Apply functional composition — higher-order functions, closures, and pipelines — as an alternative to inheritance
- Use Stage 3 decorators appropriately and distinguish them from the legacy experimental decorators
- Apply dependency injection in TypeScript, from manual constructor injection to container libraries, and choose when each fits
- Recognize and apply patterns in React (hooks, providers, render composition) and Node.js (middleware, repositories, adapters) contexts
- Refactor toward patterns incrementally from concrete code smells, and recognize anti-patterns and overengineering
- Review AI-generated TypeScript for structural quality, naming the patterns it uses and the SOLID violations it hides

Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Training Materials

Comprehensive courseware is distributed online at the start of class. All students receive a downloadable MP4 recording of the training.

Software Requirements

- Node.js 24 LTS or later
- TypeScript (latest stable, installed per-project via npm)
- Text editor or IDE (VS Code recommended)
- Git for version control
- Access to an AI coding assistant for the code review exercises (any major tool)

Training Topics

Design Principles Foundations

- What design patterns are and what they are not
- Coupling, cohesion, and the cost of change
- Reading the GoF catalog critically in 2026
- Why TypeScript changes the catalog
- The vocabulary value of patterns on a team

SOLID Principles, Applied Honestly

- Single Responsibility: reasons to change, not line counts
- Open/Closed via interfaces, unions, and composition
- Liskov Substitution in a structural type system
- Interface Segregation and small object shapes
- Dependency Inversion without ceremony
- When SOLID is over-applied: speculative abstraction

TypeScript's Type System as a Design Tool

- Structural typing and its impact on interface patterns
- Type aliases vs. interfaces in pattern code
- Discriminated unions vs. class hierarchies
- Exhaustiveness checking with `never`
- Generics and conditional types in pattern design
- Branded types and making illegal states unrepresentable

Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Creational Patterns

- Factory functions vs. Factory Method classes
- Abstract Factory and dependency seams
- Builder vs. object literals and fluent APIs
- Singleton vs. ES modules and module-level state
- Lazy initialization and memoization

Structural Patterns

- Adapter for integrating third-party code
- Facade for taming complex subsystems
- Decorator pattern, wrappers, and higher-order functions
- Stage 3 decorators vs. legacy experimental decorators
- Proxy and the JavaScript `Proxy` object
- Composite for tree structures

Behavioral Patterns

- Strategy via first-class functions and object maps
- Command, closures, and undo stacks
- Observer vs. event emitters and reactive streams
- Template Method vs. composition and callbacks
- State machines: unions, objects, and libraries
- Visitor vs. discriminated unions and pattern matching
- Iterators, generators, and async iteration

Composition and Dependency Injection

- Composition over inheritance in practice
- Functional composition: pipelines and higher-order functions
- Manual constructor injection and factory wiring
- DI containers in the TS ecosystem: costs and benefits
- Wiring at the application edge: the composition root
- Testability as a design feedback signal

Patterns in React and Node Contexts

- Hooks as Strategy and Observer in disguise
- Provider pattern and context as injection
- Component composition vs. inheritance
- Middleware chains in Node frameworks
- Repository and adapter patterns at service boundaries

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Refactoring, Anti-Patterns, and Overengineering

- Code smells that motivate each pattern family
- Incremental refactoring with tests as a safety net
- Case study: replacing conditional sprawl with Strategy
- Pattern abuse: class ceremony transplanted into TypeScript
- Type-level overengineering and unreadable generics
- Premature abstraction, YAGNI, and the rule of three

Patterns in the Age of AI-Generated Code

- Structural habits of AI coding assistants
- Recognizing pattern use and misuse in agent output
- Reviewing generated code against SOLID
- Steering agents with design-aware prompts and conventions
- Keeping architectural ownership when agents write the code