HAREESH RAJENDRAN

01/

DESIGN PATTERNS: EFFICIENT SOLUTION FOR SCALABLE APPLICATIONS

...

Software Developer- Highbrow Technology, Ex-Zoho, DevTuber

NEXT →

02/

Agenda

- WHAT'S A DESIGN PATTERN?
- SINGLETON 🏛️
- PROXY 🛡️
- PROTOTYPE 🧬
- OBSERVER 👁️
- MODULE 📦
- MIXIN 🧩
- MIDDLEWARE 🔄
- FLYWEIGHT 🪶
- FACTORY 🏭
- ABSTRACT FACTORY 😶‍🌫️

“What are Design Patterns ?”

DESIGN PATTERNS IN ARE **REUSABLE SOLUTIONS** TO COMMON PROBLEMS IN SOFTWARE DESIGN.

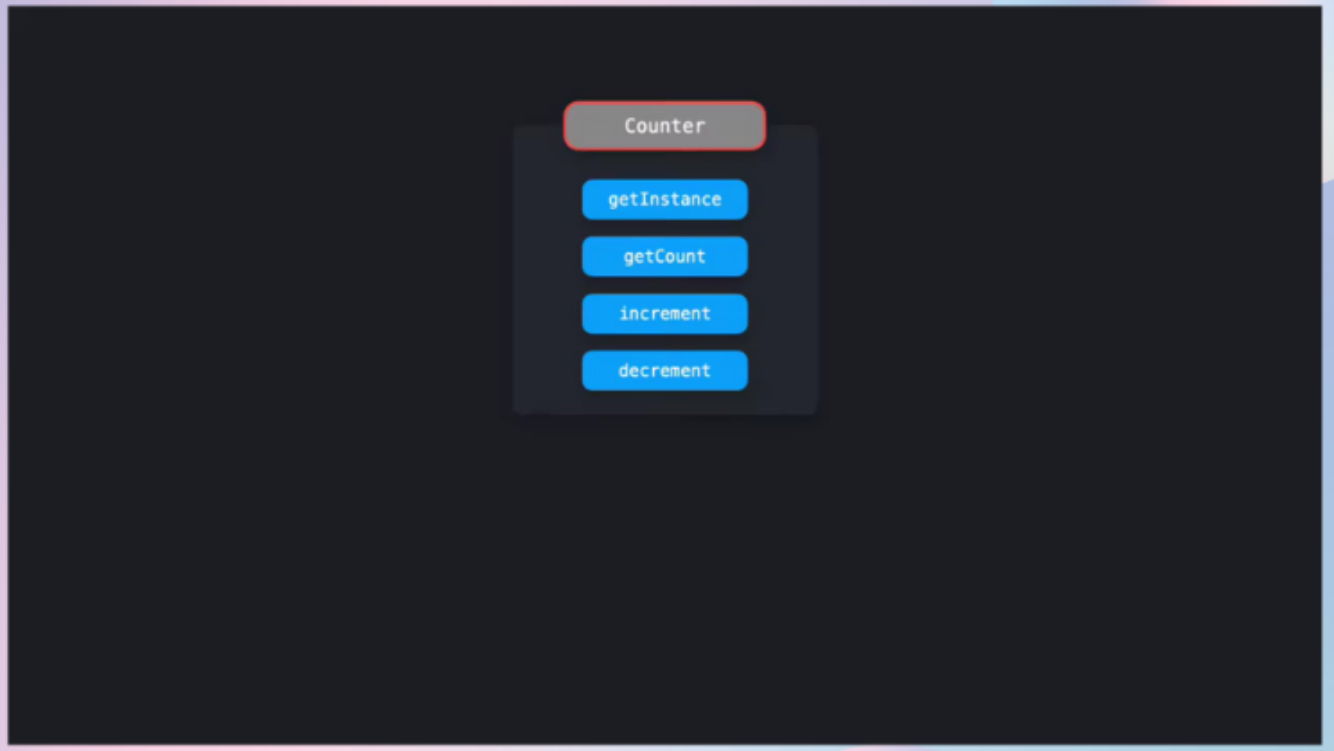
THESE PATTERNS HELP WRITE MORE STRUCTURED AND MAINTAINABLE CODE.

THEY ARE **EFFICIENT** AND **PROVEN**.

NEXT →

Singleton pattern

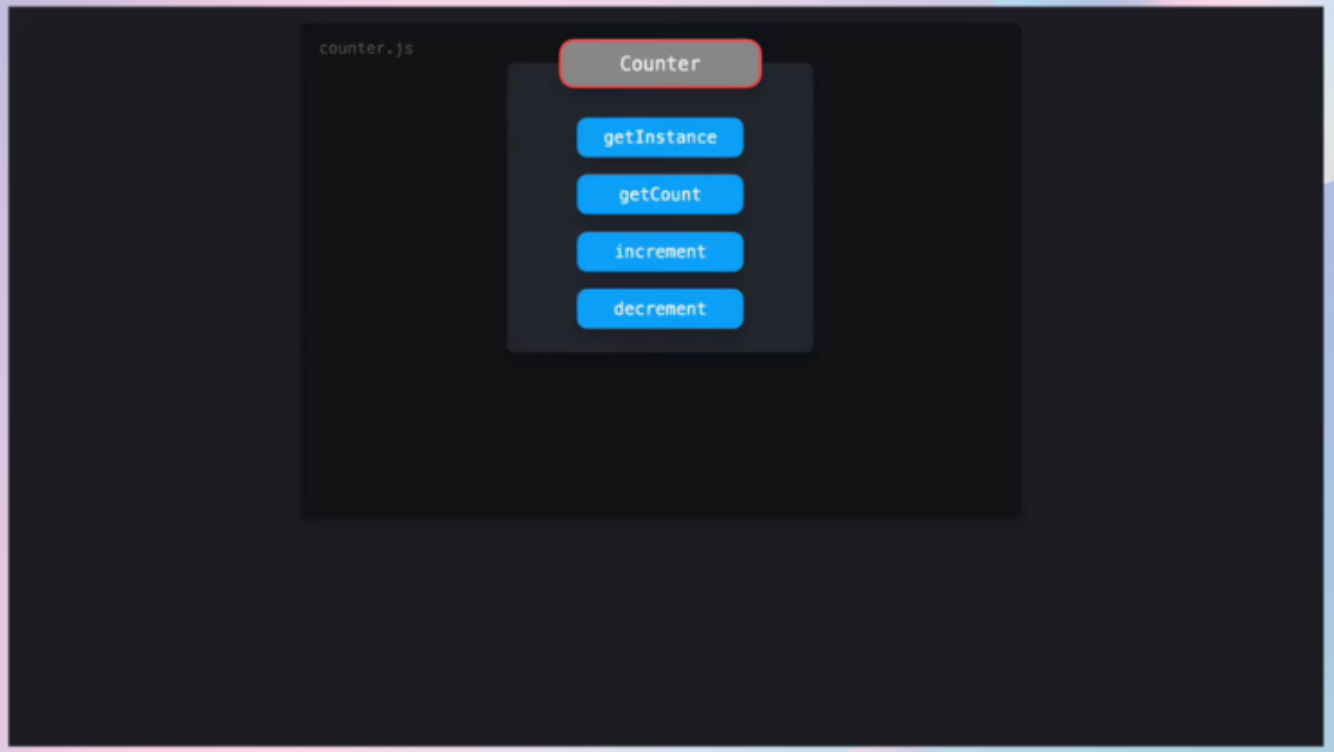
ENSURES A CLASS HAS ONLY ONE INSTANCE.



NEXT →

Singleton pattern

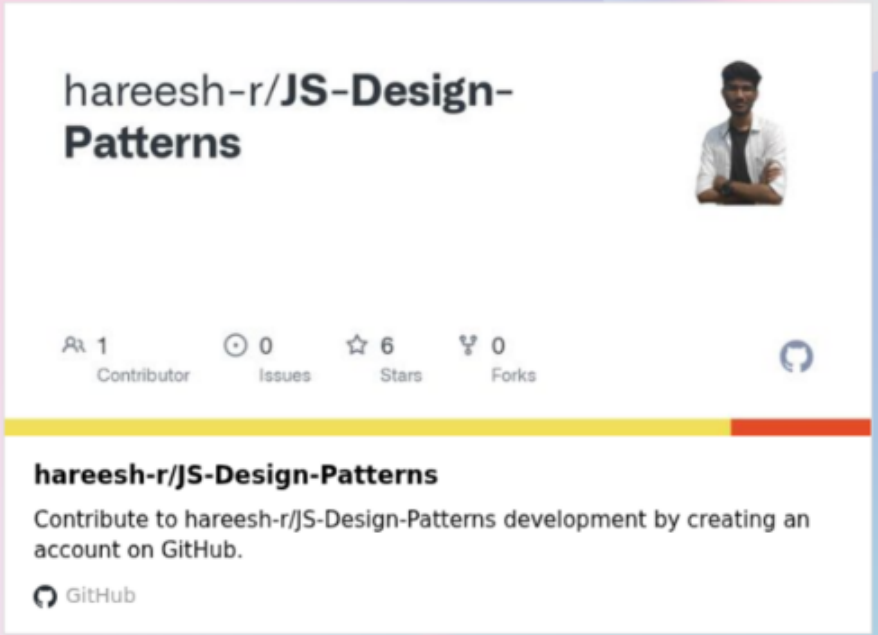
ONLY ONE INSTANCE FOR
THE ENTIRE PROJECT



NEXT →

JS Design Patterns

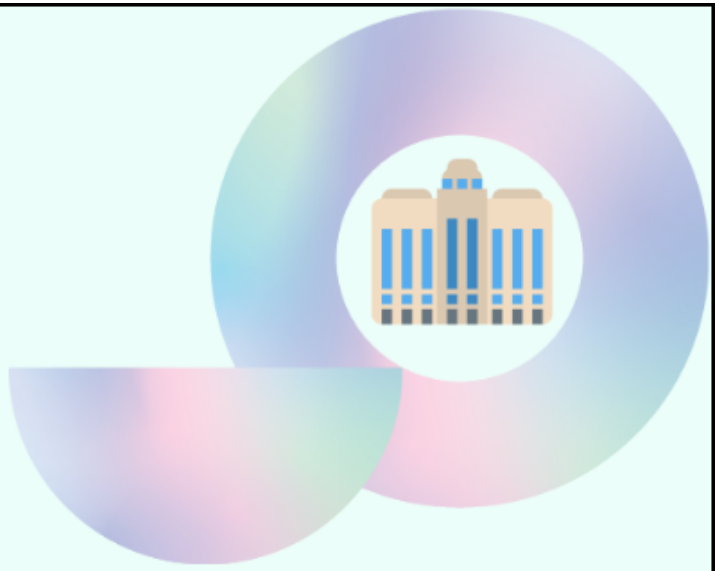
LET'S SEE A DEMO



NEXT →

07/

USECASE



- DATABASE CONNECTION INSTANCE.
- GLOBAL CONFIGURATION SETTINGS.
- LOGGER UTILITY (CENTRALIZED LOGGING).
- CACHE MANAGER FOR SHARED DATA.
- MAINTAINING A SINGLE INSTANCE OF A GLOBAL APPLICATION STATE.

NEXT →

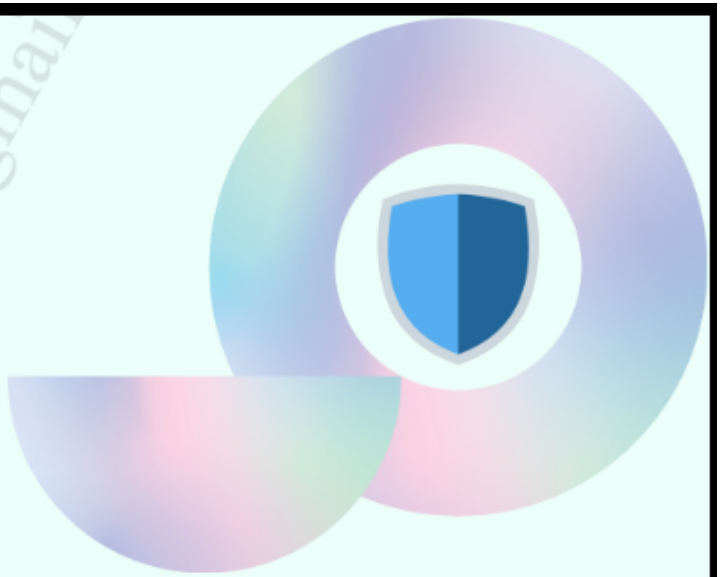
Proxy pattern

CONTROLS ACCESS TO AN OBJECT BY ACTING AS AN INTERMEDIARY.

NEXT →

09/

USECASE

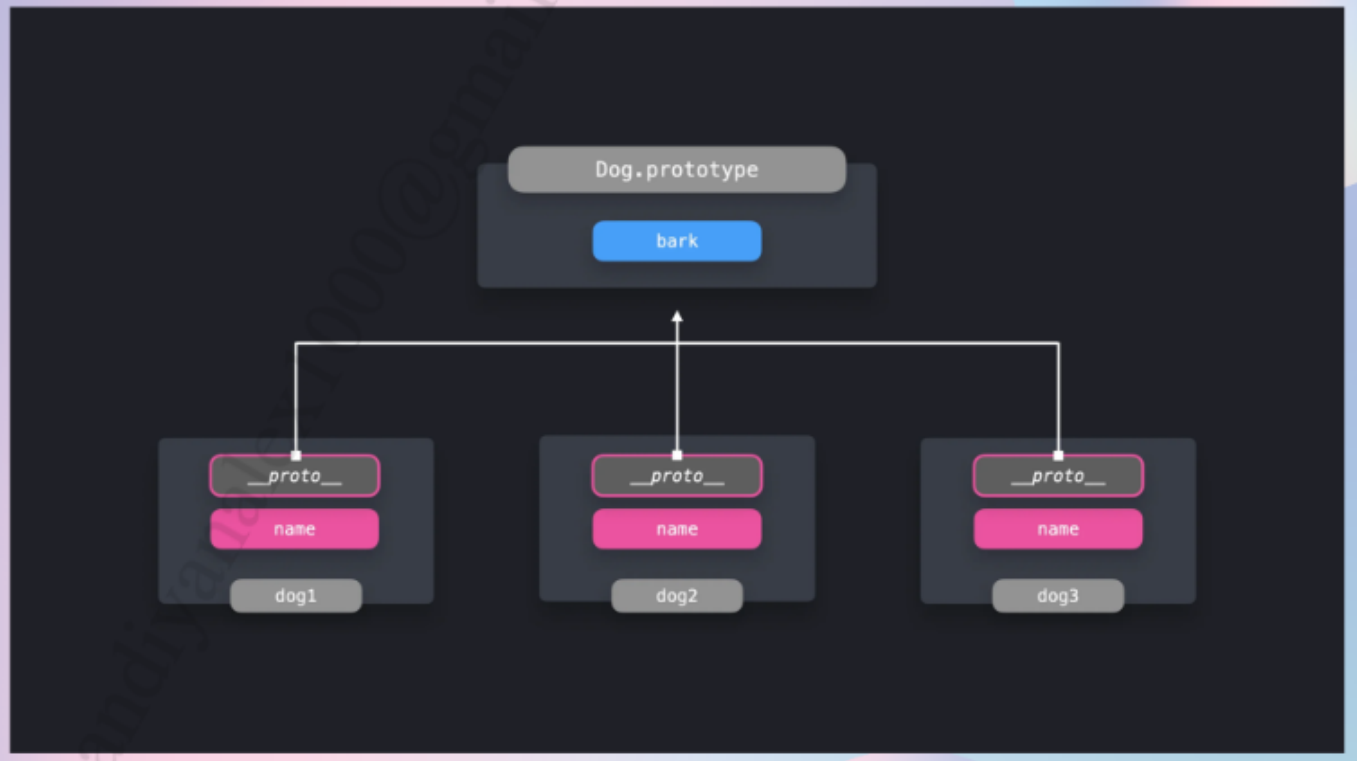


- CLONING COMPLEX OBJECTS (DEEP COPY).
- CREATING MULTIPLE INSTANCES OF OBJECTS WITH DEFAULT PROPERTIES.
- AVOIDING REINITIALIZING OBJECTS IN PERFORMANCE-CRITICAL APPS.
- COPYING OBJECTS IN GAME DEVELOPMENT (E.G., CHARACTERS).
- OBJECT POOLING FOR PERFORMANCE OPTIMIZATIONS.

NEXT →

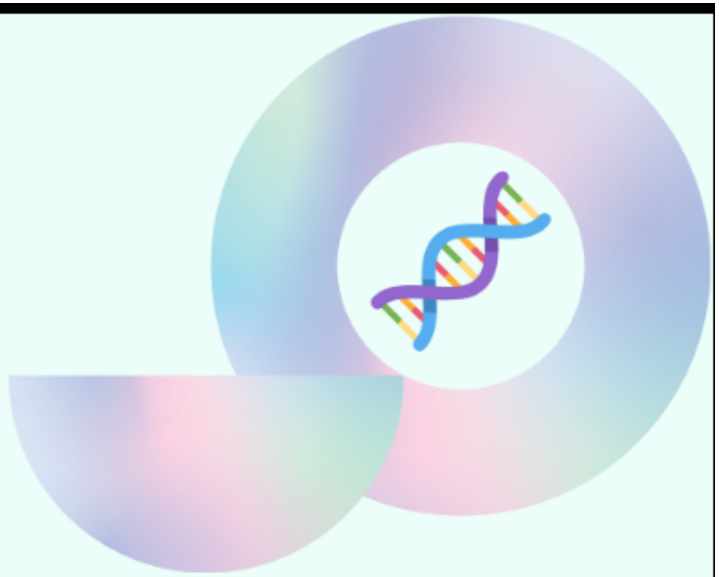
Prototype pattern

CREATES NEW OBJECTS BY CLONING EXISTING ONES.



NEXT →

USECASE

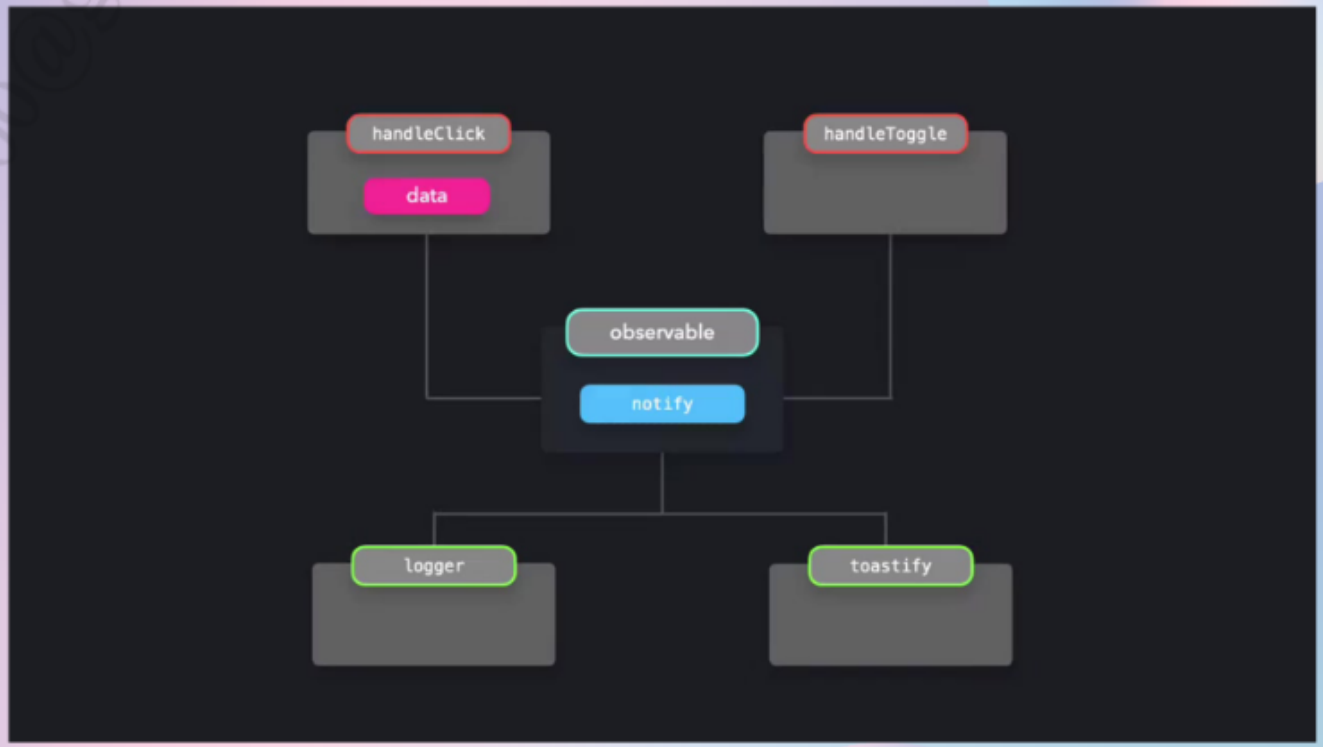


- EVENT HANDLING SYSTEMS (E.G., DOM EVENTS).
- CHAT APPLICATIONS (MESSAGE BROADCASTING).
- NOTIFICATION SYSTEMS FOR USER ACTIVITIES.
- STOCK PRICE MONITORING APPS.
- REAL-TIME DATA FEEDS IN FINANCIAL DASHBOARDS.

NEXT →

Observer pattern

NOTIFIES SUBSCRIBERS OF CHANGES IN STATE OR EVENTS.



NEXT →

Observer pattern

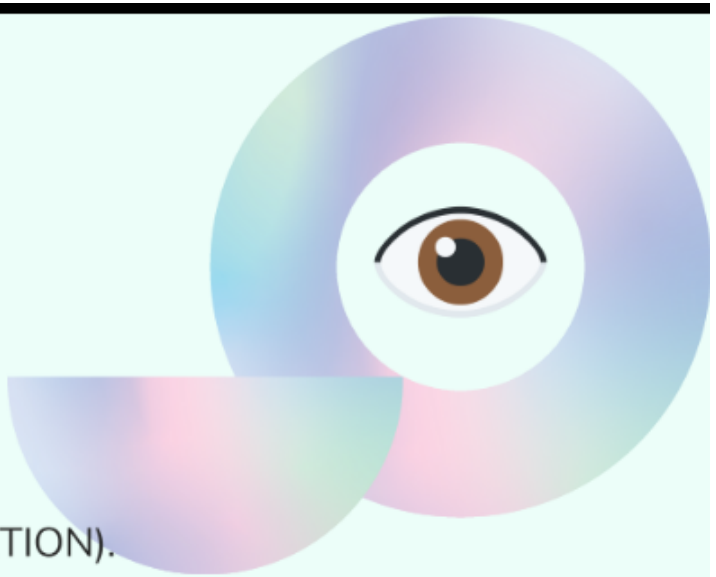
WITH THE OBSERVER PATTERN, WE CAN SUBSCRIBE CERTAIN OBJECTS,

THE OBSERVERS, TO ANOTHER OBJECT, CALLED THE **OBSERVABLE**.

WHENEVER AN EVENT OCCURS, THE OBSERVABLE NOTIFIES ALL ITS **OBSERVERS**!

NEXT →

USECASE



- ENCAPSULATING RELATED FUNCTIONS (E.G., USER AUTHENTICATION).
- ORGANIZING LARGE CODEBASES INTO REUSABLE PARTS.
- HIDING IMPLEMENTATION DETAILS WHILE EXPOSING AN API.
- CREATING A STATE MANAGEMENT MODULE.
- SETTING UP A UTILITY LIBRARY (E.G., MATH OR DATE HELPER FUNCTIONS).

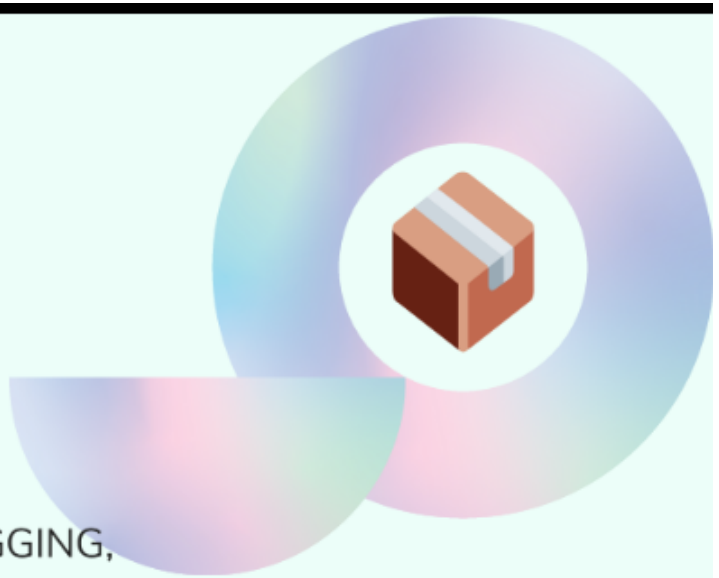
NEXT →

Module pattern

ENCAPSULATES CODE INTO SELF-CONTAINED UNITS WITH PUBLIC AND PRIVATE MEMBERS.

NEXT →

USECASE

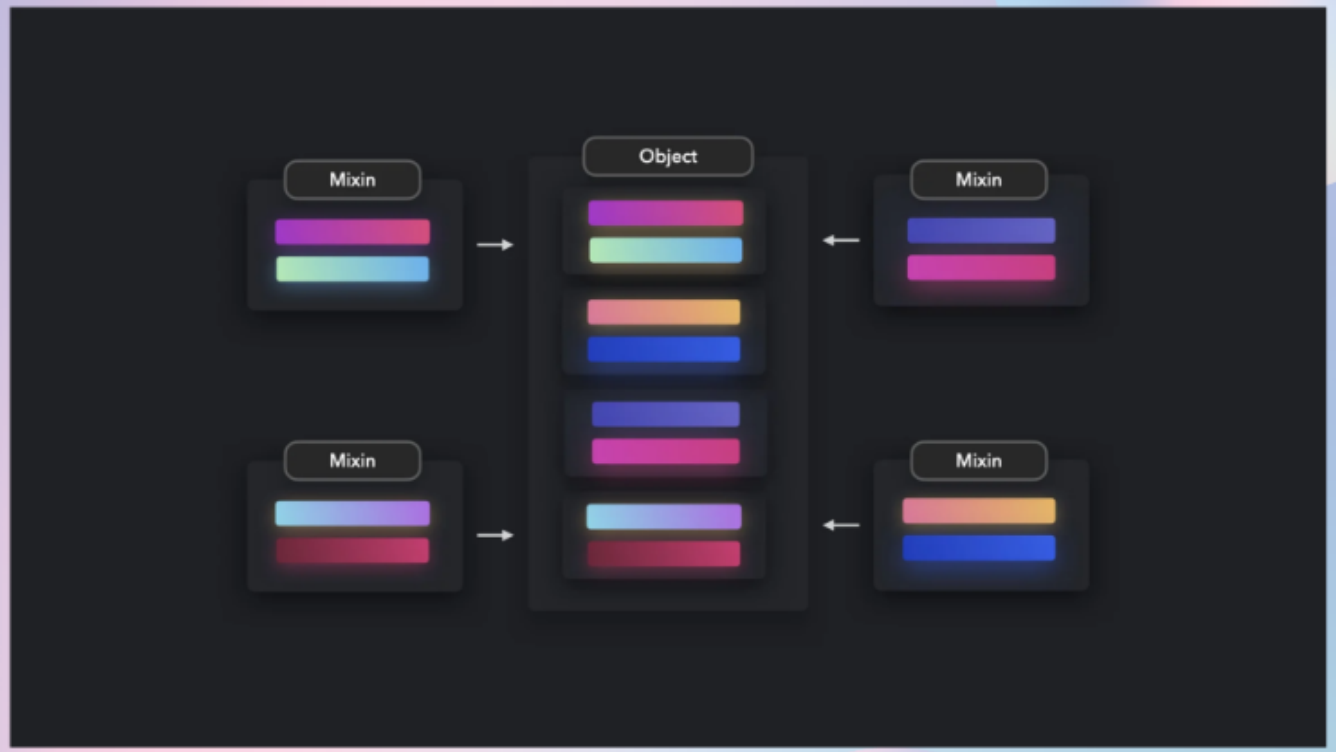


- ADDING REUSABLE METHODS TO DIFFERENT CLASSES (E.G., LOGGING, EVENT HANDLING).
- CROSS-CUTTING CONCERNS LIKE CACHING OR VALIDATION LOGIC.
- EXTENDING CLASS FUNCTIONALITY WITHOUT INHERITANCE.
- ADDING BEHAVIOR TO OBJECTS DYNAMICALLY IN UI FRAMEWORKS.
- SHARING FUNCTIONALITY BETWEEN DIFFERENT UI COMPONENTS.

NEXT →

Mixin pattern

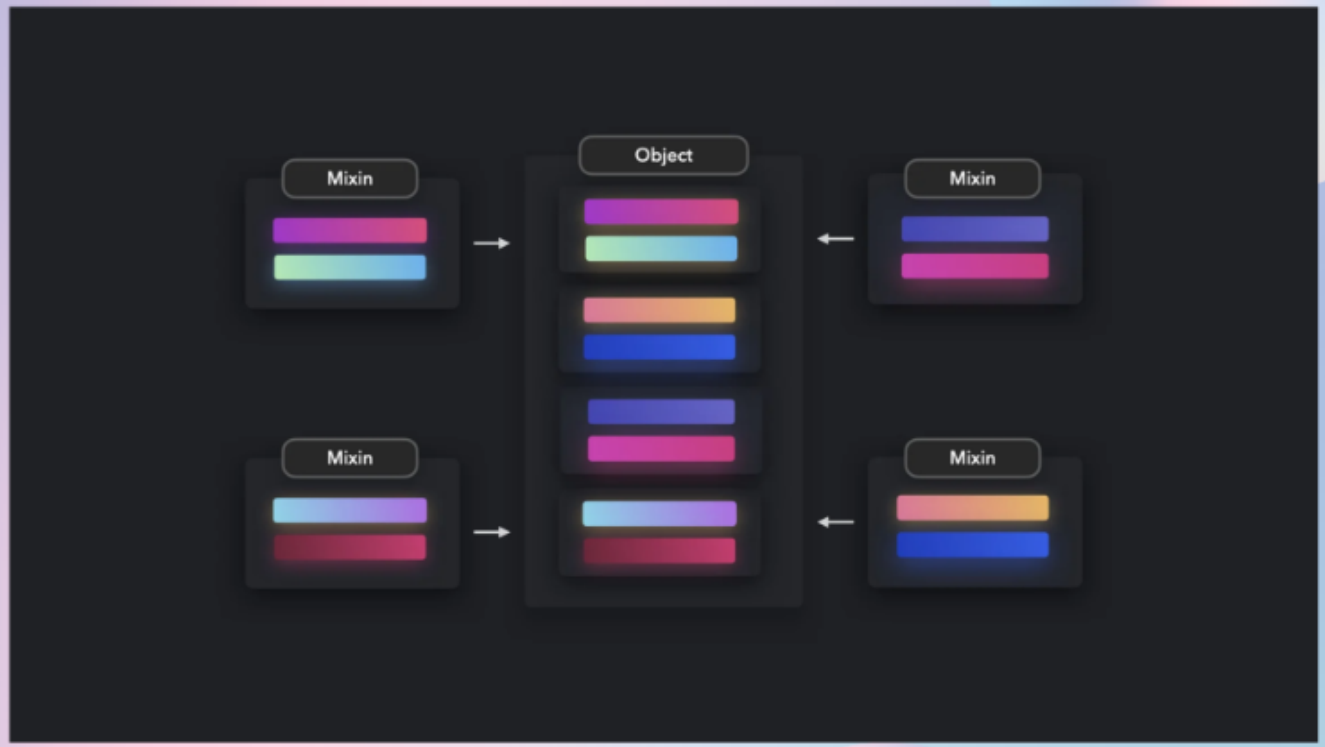
SHARES REUSABLE METHODS ACROSS MULTIPLE CLASSES WITHOUT INHERITANCE.



NEXT →

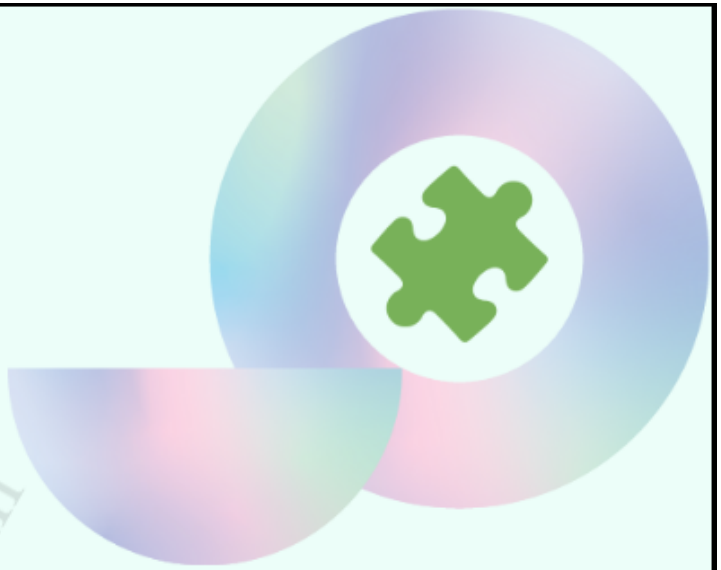
Mixin pattern

MINI CODES THAT CAN RUN INDEPENDENTLY AND MIGHT BE REQUIRED ANYWHERE IN YOUR CODE BUT NOT ALL THE TIME .... UMMM LIKE AN HELPER FUNCTOIN



NEXT →

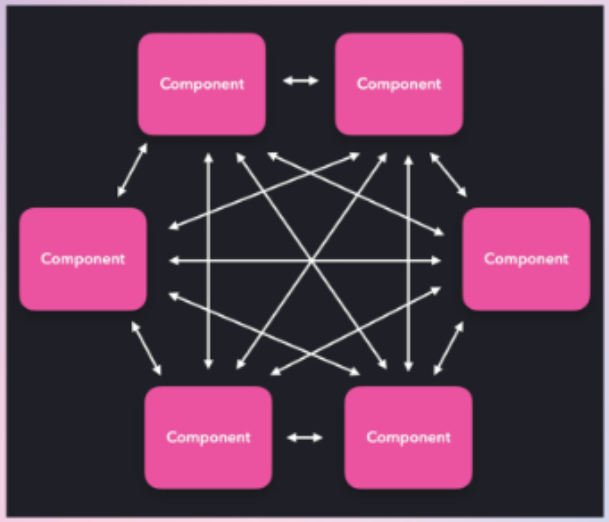
USECASE

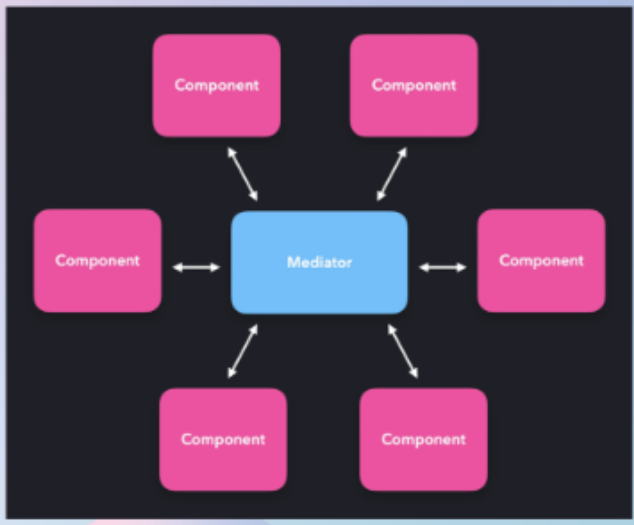


- DRAG AND DROP: REUSABLE DRAG BEHAVIOR.
- FORM VALIDATION: SHARED VALIDATION LOGIC.
- THEME SWITCHING: LIGHT/DARK MODE TOGGLING.
- TOOLTIP DISPLAY: REUSABLE HOVER TOOLTIPS.
- MODAL HANDLING: CENTRALIZED MODAL MANAGEMENT.

Middleware pattern

SEQUENTIALLY PROCESSES REQUESTS/RESPONSES IN A PIPELINE.

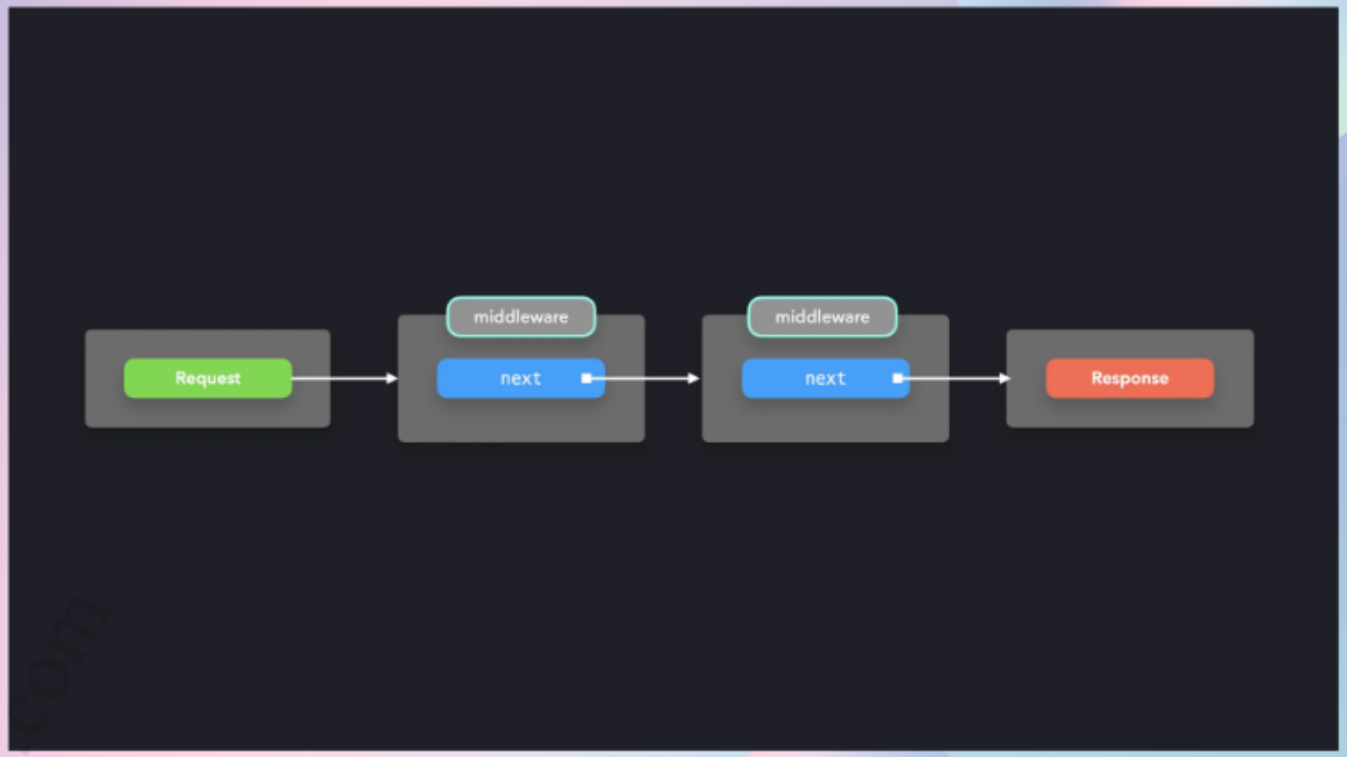




NEXT →

Middleware pattern

MOST OF THEM KNOW THIS AS APP.USE() FUNCTION IN NODE.JS



NEXT →

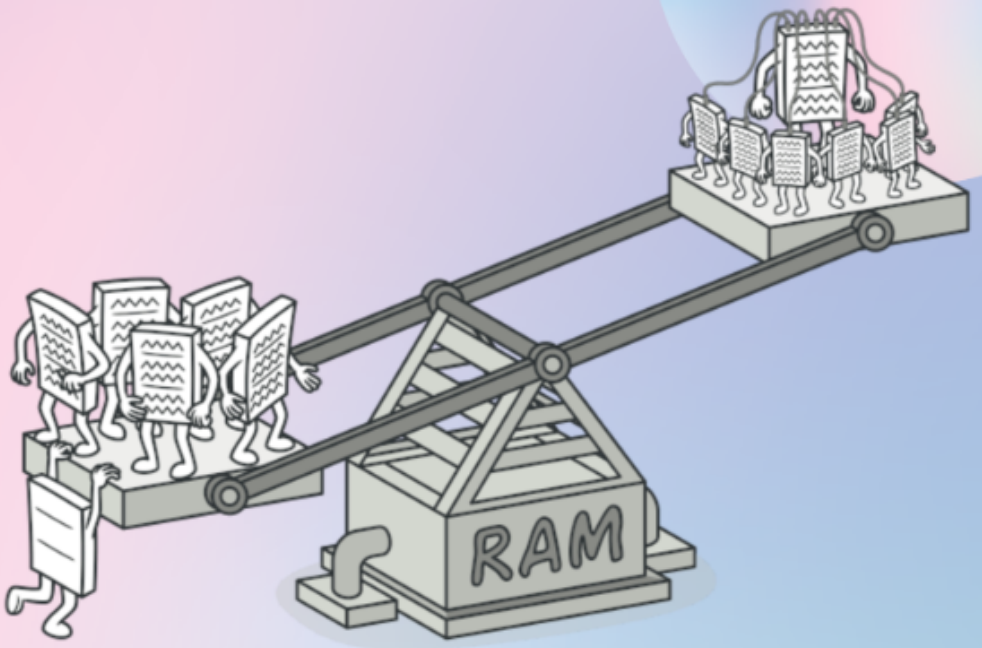
USECASE

- HANDLING REQUESTS IN EXPRESS.JS BEFORE REACHING THE SERVER.
- LOGGING HTTP REQUESTS.
- AUTHENTICATION AND AUTHORIZATION IN WEB APPLICATIONS.
- ERROR HANDLING AND VALIDATION PIPELINES.
- ADDING CUSTOM LOGIC IN API RESPONSE PROCESSING.

NEXT →

Flyweight pattern

MINIMIZES MEMORY USAGE BY SHARING COMMON OBJECT DATA.



NEXT →

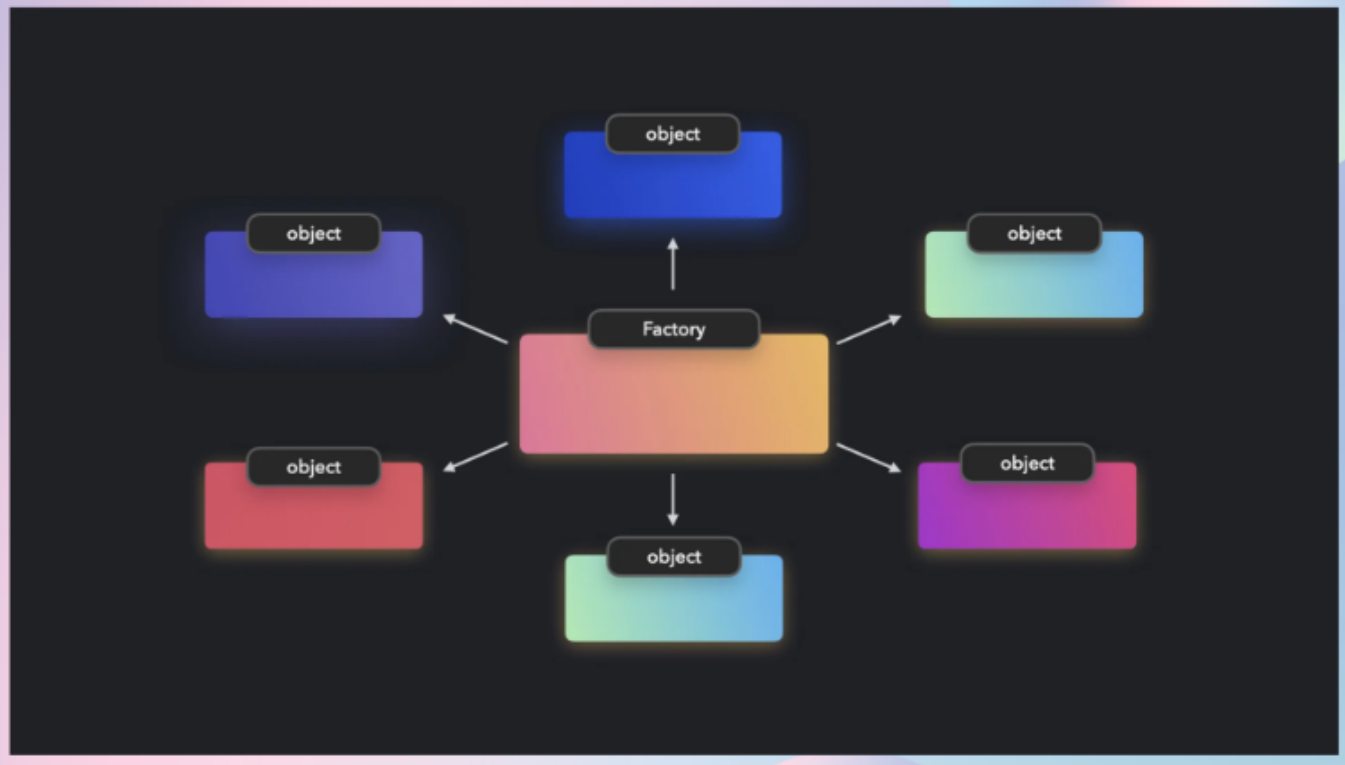
USECASE

- OPTIMIZING MEMORY USAGE IN GAME DEVELOPMENT (SHARED TEXTURES).
- CACHING UI COMPONENTS TO AVOID REDUNDANT RENDERING.
- EFFICIENTLY HANDLING LARGE NUMBERS OF DOM ELEMENTS.
- FONT MANAGEMENT IN GRAPHIC EDITORS.
- STRING INTERNING TO REDUCE MEMORY OVERHEAD.

NEXT →

Factory pattern

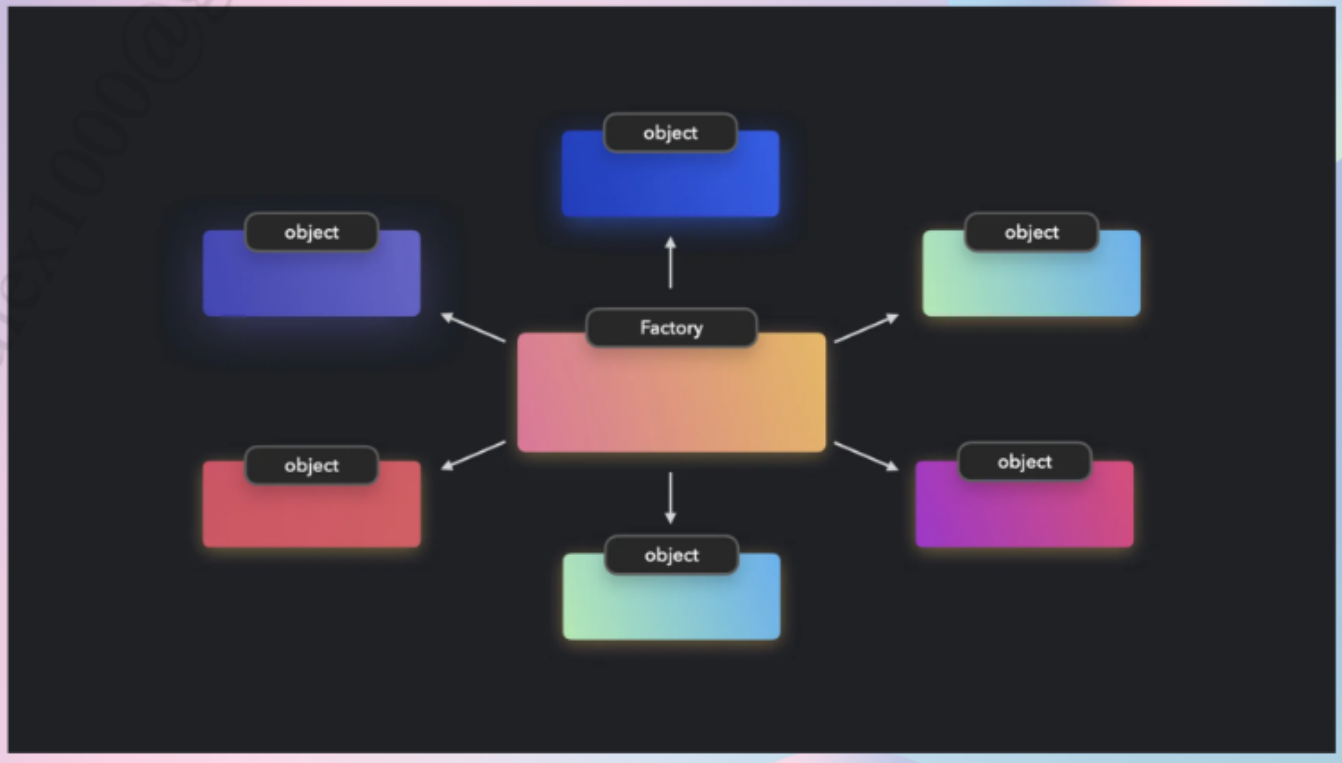
CREATES OBJECTS WITHOUT SPECIFYING THE EXACT CLASS.



NEXT →

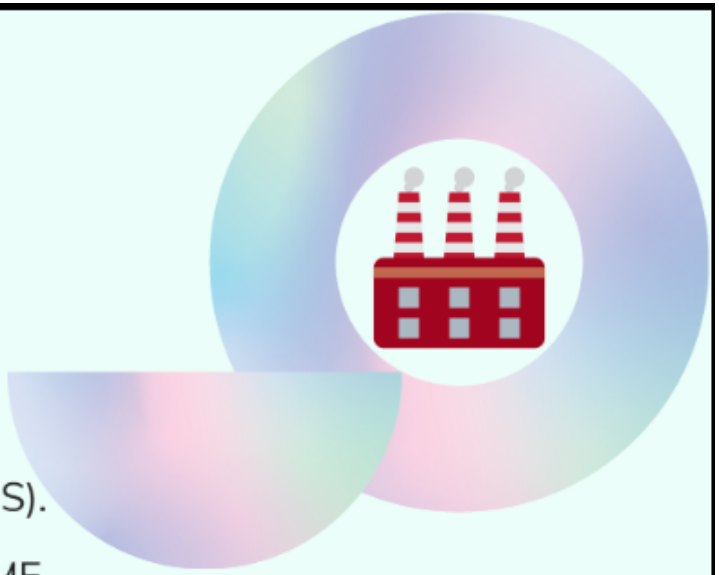
Factory pattern

A FUNCTION IS A FACTORY FUNCTION WHEN IT RETURNS A NEW OBJECT WITHOUT THE USE OF THE ***NEW*** KEYWORD!



NEXT →

USECASE

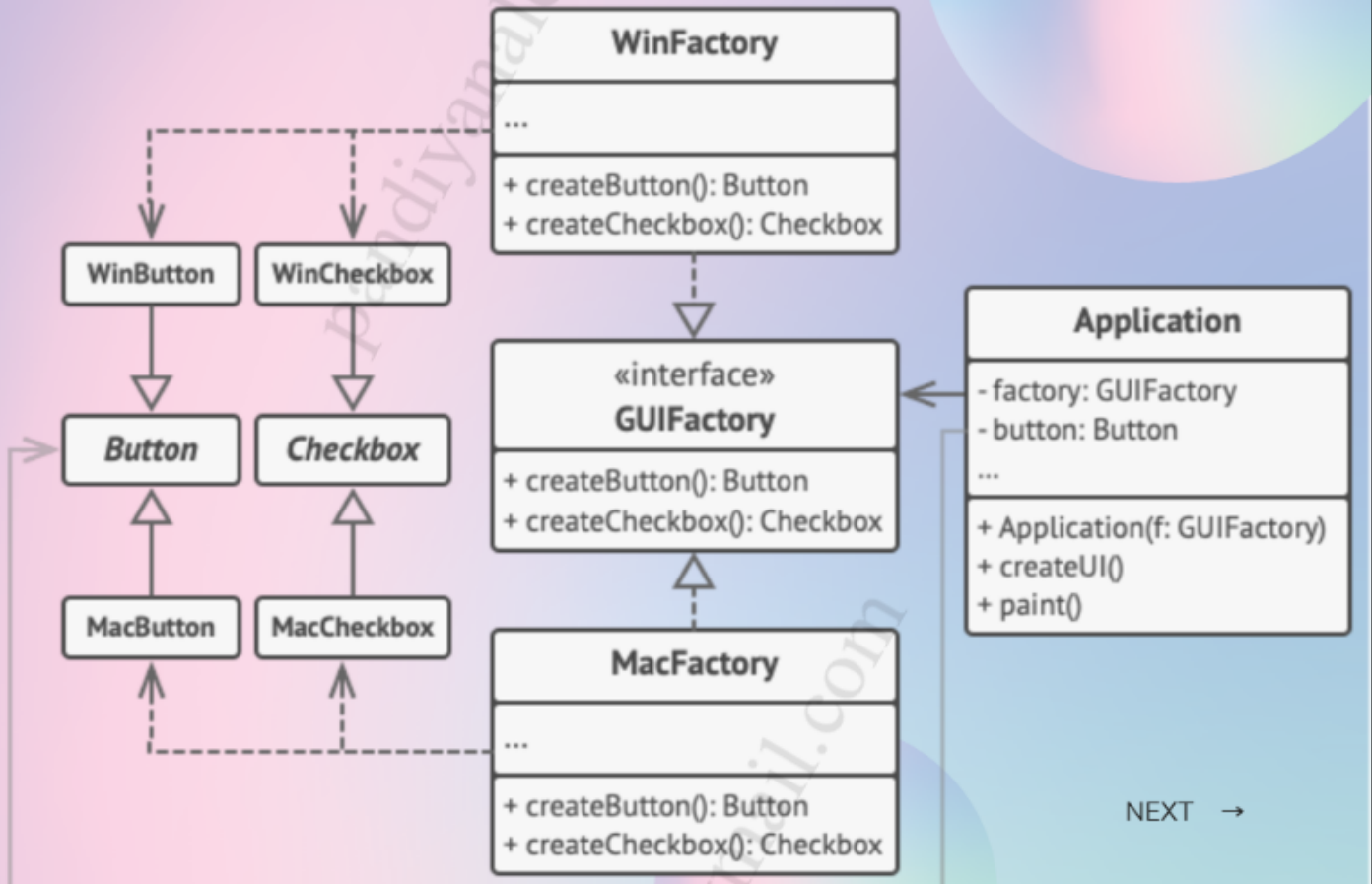


- GENERATING UI ELEMENTS DYNAMICALLY (E.G., BUTTONS, FORMS).
- CREATING DIFFERENT TYPES OF VEHICLES IN A SIMULATION GAME.
- OBJECT CREATION IN ORM FRAMEWORKS.
- INSTANTIATING DIFFERENT PAYMENT METHODS (E.G., CREDIT CARD, PAYPAL).
- GENERATING CONFIGURATION OBJECTS BASED ON ENVIRONMENT VARIABLES.

NEXT →

Abstract Factory pattern

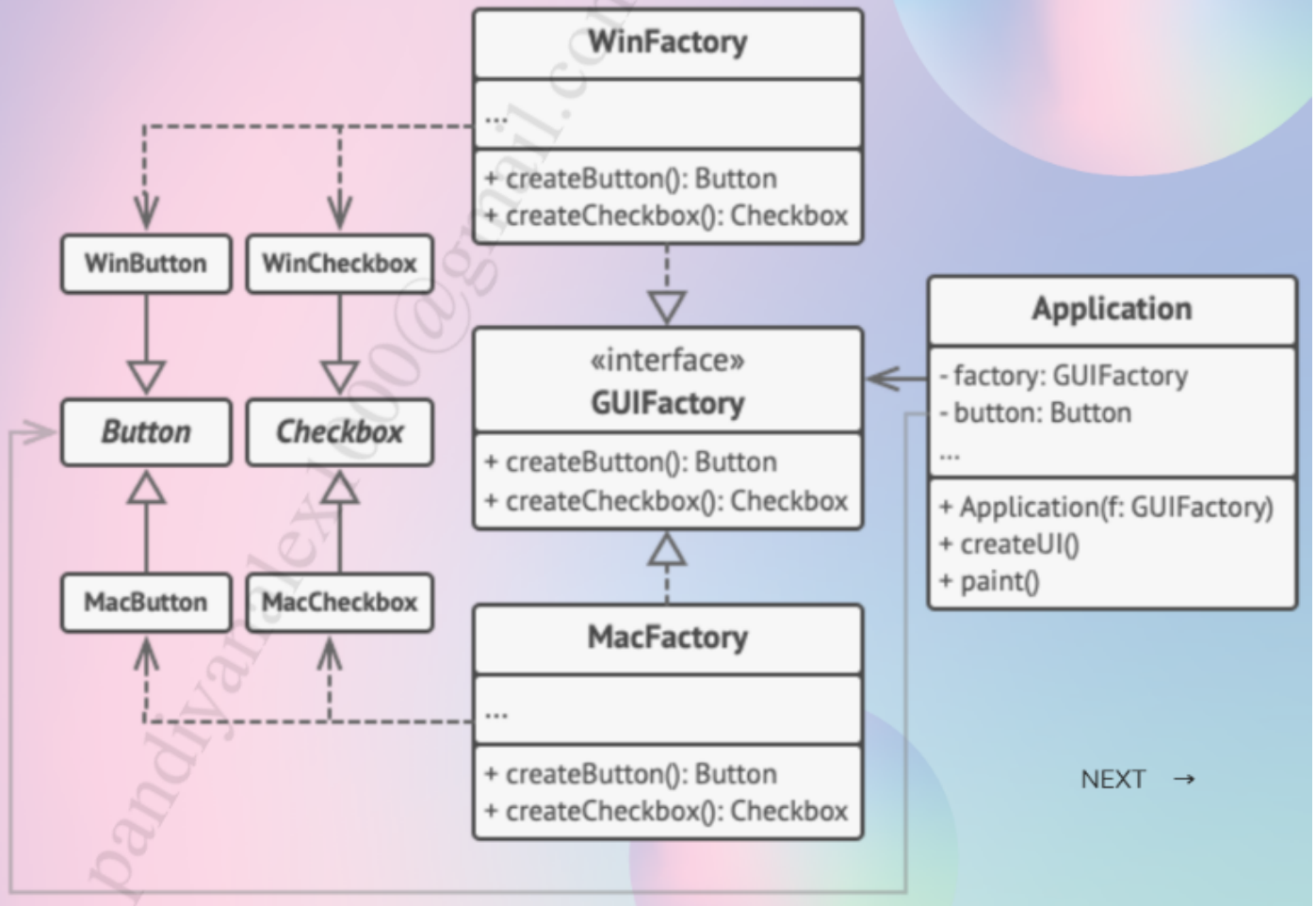
PROVIDES AN INTERFACE FOR CREATING FAMILIES OF RELATED OR DEPENDENT OBJECTS WITHOUT SPECIFYING THEIR CONCRETE CLASSES.



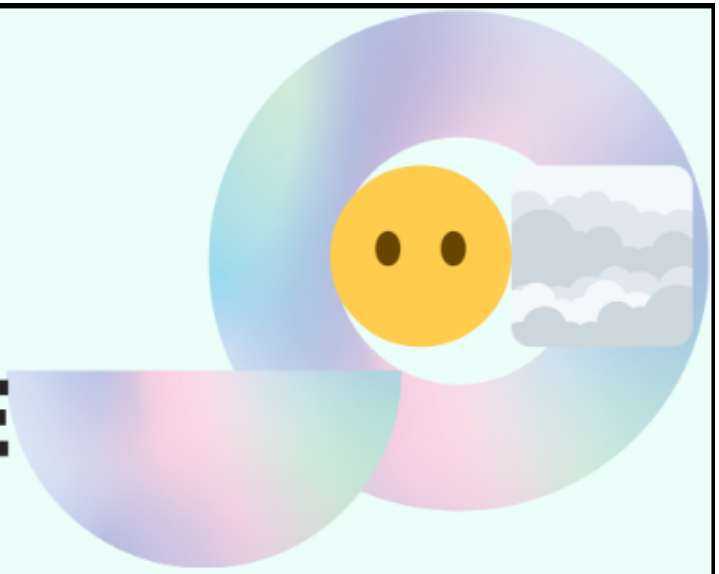
NEXT →

Abstract Factory pattern

CREATES RELATED OBJECTS (LIKE UI COMPONENTS) WITHOUT KNOWING THEIR EXACT CLASSES.



NEXT →



USECASE

- 🖥️ CREATING THEMED UI KITS (BUTTONS, CHECKBOXES, TEXT FIELDS) — MAC VS WINDOWS.
- 🚗 BUILDING VEHICLE PARTS BY BRAND — TESLAFACTORY VS TOYOTAFACTORY.
- 💳 PAYMENT PROVIDERS — STRIPEFACTORY VS PAYPALFACTORY (BUTTON, VALIDATOR, RECEIPT).
- 🌐 LOADING SERVICES BY ENVIRONMENT — DEVFACTORY VS PRODFACTORY (LOGGER, DB, CACHE).

NEXT →

FAQS

- WHEN TO USE STRATEGY VS. STATE PATTERN?
- WHAT IS THE DIFFERENCE BETWEEN BUILDER AND PROTOTYPE PATTERN?
- HOW IS DEPENDENCY INJECTION A DESIGN PATTERN?
- EXPLAIN THE REPOSITORY PATTERN AND ITS BENEFITS.
- EXPLAIN CIRCUIT BREAKER AND RETRY PATTERNS IN DISTRIBUTED SYSTEMS.
- WHAT IS THE TEMPLATE METHOD PATTERN AND WHERE TO USE IT?
- HOW WOULD YOU IMPLEMENT A COMMAND PATTERN FOR A TASK QUEUE?
- DIFFERENCE BETWEEN MVC AND MVVM PATTERNS.
- EXPLAIN CHAIN OF RESPONSIBILITY WITH LOGGING/MIDDLEWARE USE CASE.
- HOW DO DESIGN PATTERNS RELATE TO SOLID PRINCIPLES?

NEXT →

QUESTION & ANSWERS

LET'S MAKE THIS AS COLLABORATIVE AS POSSIBLE