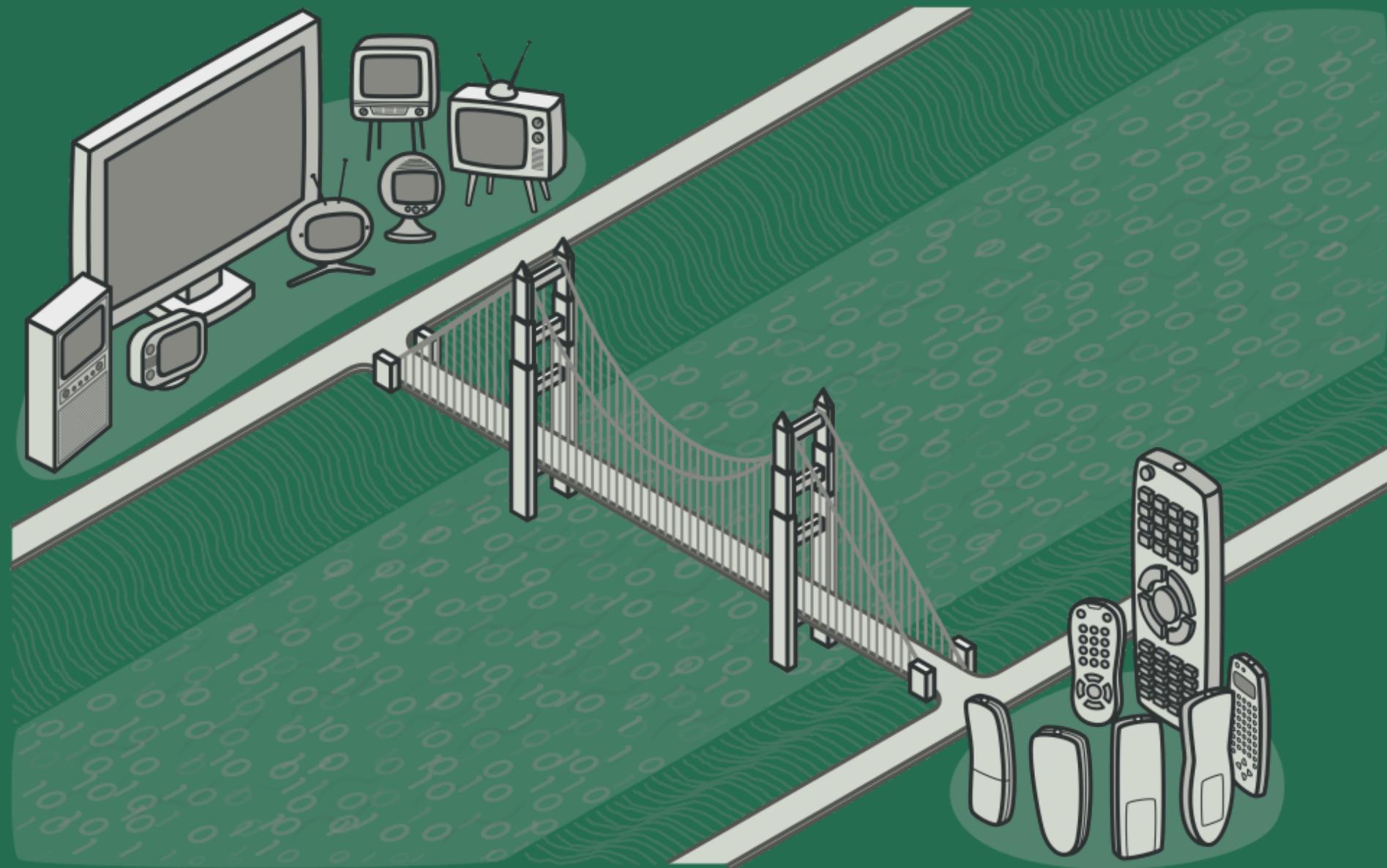


الگو BRIDGE



PUBLISHED BY : FATEMEH AHMADZADEH

معرفی

چیست؟  BRIDGE

- پترن ساختاری برای جلوگیری از کلاس‌های غول‌آسا
- جداسازی منطق سطح بالا (ABSTRACTION) از جزئیات سطح پایین (IMPLEMENTATION)

سناپریو

فرض کنید: 

- کلاس **SHAPE** داریم: **SQUARE** و **CIRCLE**:
- می خواهیم رنگ ها را هم اضافه کنیم: **BLUE** و **RED**:

راه حل اشتباه

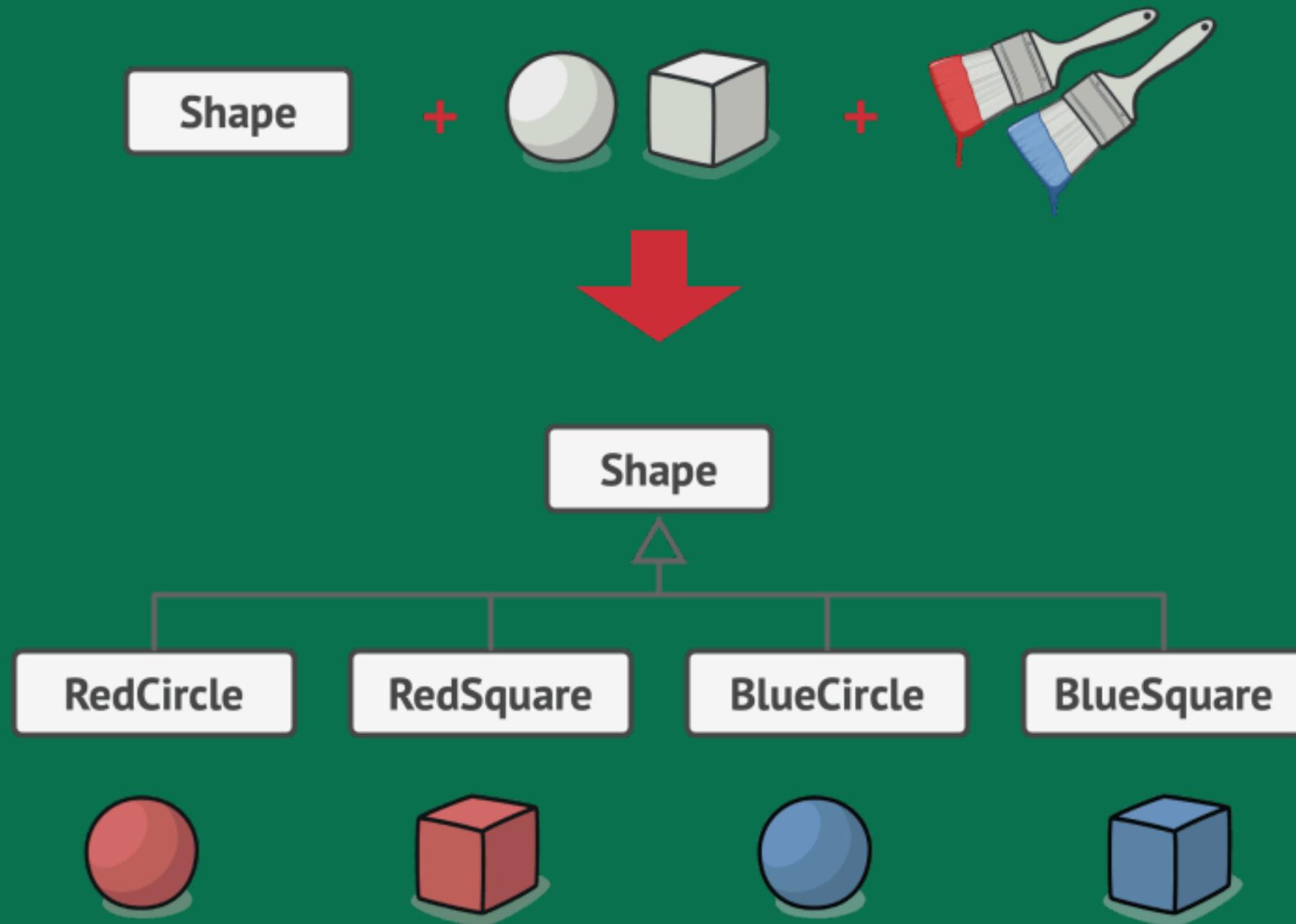
✖ ایجاد کلاس برای هر ترکیب: RED CIRCLE, BLUE CIRCLE

چرا اشتباهه:

● انفجار کلاس‌ها! وقتی رنگ یا شکل جدید اضافه می‌کنیم، تعداد

کلاس‌ها به شدت زیاد می‌شود

راه حل اشتباه



راه حل صحیح

* با BRIDGE پترن

چگونه کمک می‌کند؟

* ارث بری را با ترکیب (COMPOSITION) جایگزین می‌کنیم

* رنگ‌ها در سلسله کلاس‌جداگانه: COLOR, RED, BLUE

* هر SHAPE فقط یک COLOR REFERENCE به دارد

اشکال و رنگ‌ها مستقل از هم تغییر می‌کنند

چه تفاوتی با ABSTRACT FACTORY پ?

ABSTRACT FACTORY ◆

• هدف: تولید خانواده‌های مرتبط 

• ساختار:  FACTORY → PRODUCTS

• مثال:  RED CIRCLE, BLUE SQUARE

• مشکل: انفجار کلاس‌ها 

چه تفاوتی با ABSTRACT FACTORY پ?

BRIDGE ◆

- هدف: جداسازی IMPLEMENTATION و ABSTRACTION
- ساختار: SHAPE → HAS-A COLOR
- مثال: CIRCLE(RED), SQUARE(BLUE)
- مزیت: انعطاف و سادهسازی

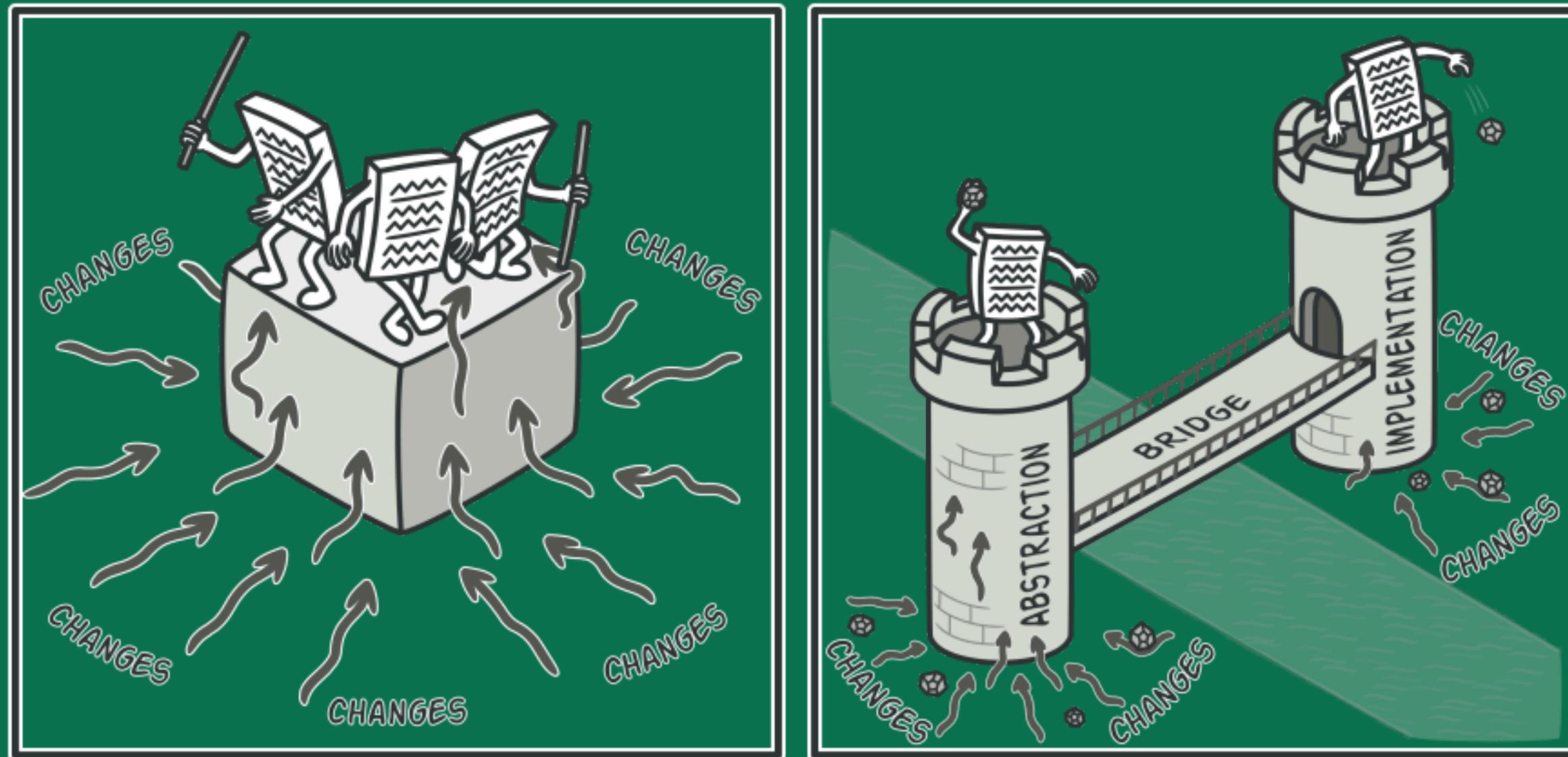
فرمول ساده برای درک این مقایسه

فرمول ساده:

💡 ABSTRACT FACTORY = SHAPE × COLOR → SUBCLASS EXPLOS

💡 BRIDGE = SHAPE + COLOR → COMPOSITION

مقایسه با ABSTRACT FACTORY



ساختار کلی

: نقش‌ها در 

DEVICE  → IMPLEMENTATION .

◦ جزئیات واقعی کاری که انجام می‌دهد

◦ مثال: تلویزیون، دستگاه صوتی

REMOTE  → ABSTRACTION .

◦ رابط سطح بالا برای کنترل دستگاه

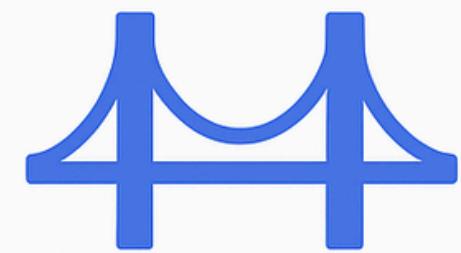
◦ مثال: ریموت کنترل تلویزیون

ساختار کلی



Device

Implementation



Bridge



Remote

Abstraction

یادآوری: IMPLEMENTATION و ABSTRACTION = جدا کردن BRIDGE



کی از BRIDGE استفاده می‌کنیم؟

■ جدا کردن منطق سطح بالا از جزئیات

وقتی نمی‌خوای همه‌چیز توی یک کلاس غول‌پیکر جمع بشه.

■ جلوگیری از پیچیدگی بیش از حد

وقتی کلاس‌ها دارن سنگین می‌شن و به سختی می‌شه نگهشون داشت.

■ پشتیبانی از حالت‌های مختلف

وقتی چند مدل ABSTRACTION و چند مدل IMPLEMENTATION داری.

کی از BRIDGE استفاده می‌کنیم؟

توسعه‌پذیری بدون دردسر 

وقتی می‌خوای هر بخش جداگانه رشد کنه بدون اینکه به بقیه آسیب بزنه.

تغییر رفتار بدون تغییر ساختار 

وقتی فقط می‌خوای یکی از بخش‌ها رو تغییر بدی، نه کل برنامه رو.

نیاز به انعطاف بلندمدت 

وقتی می‌خوای پروژه‌ت در آینده راحت مقیاس‌پذیر بشه.

ارتبط با الگو ADAPTER

= حل مشکل سازگاری بعد از بروز مشکل 

زمانی استفاده می‌شود که:

- دو سیستم آماده هست
- اما با هم سازگار نیستند
- و اکنون مجبوریم بدون تغییر زیاد، آنها را به هم وصل کنیم

ارتباط با الگو ADAPTER

= طراحی پیشگیرانه برای انعطاف 

زمانی استفاده می‌شود که از ابتدا می‌دانیم سیستم قرار است:

- تغییرپذیر باشد
- قابل توسعه باشد
- پیاده‌سازی و ABSTRACTION از هم مستقل رشد کند

ارتباط با الگو BUILDER

چطور BUILDER با BRIDGE مرتبط است؟ 

پترن BUILDER دو بخش اصلی دارد:

مشخص منند چی ساخته منشود → DIRECTOR = ABSTRACTION .

مشخص منند چی ساخته بناشود → BUILDER = IMPLEMENTATION .

منشود

ارتباط با الگو BUILDER

این دقیقاً همان ایده‌ی پترن BRIDGE است:

- در BRIDGE همیشه ABSTRACTION از IMPLEMENTATION جدا می‌شود.
- یعنی «چیزی که کاربر با آن سروکار دارد» از «چگونگی انجام کار» مستقل می‌شود.

* نتیجه این ترکیب چیست؟

= جداسازی کامل چی ساخته می‌شود از چگونگی ساخت آن

ارتباط با الگو BUILDER

این دقیقاً همان ایده‌ی پترن BRIDGE است:

- در BRIDGE همیشه ABSTRACTION از IMPLEMENTATION جدا می‌شود.
- یعنی «چیزی که کاربر با آن سروکار دارد» از «چگونگی انجام کار» مستقل می‌شود.

* نتیجه این ترکیب چیست؟

= جداسازی کامل چی ساخته می‌شود از چگونگی ساخت آن

جمع‌بندی

با این الگو می‌توانید

جلوگیری از انفجار کلاس‌ها 

انعطاف‌پذیری بالا 

نگهداری و توسعه راحت‌تر 

نکته دوستانه: 

با BRIDGE، کلاس‌هایتان نفس می‌کشند، نه اینکه غول‌آسا شوند! 😊