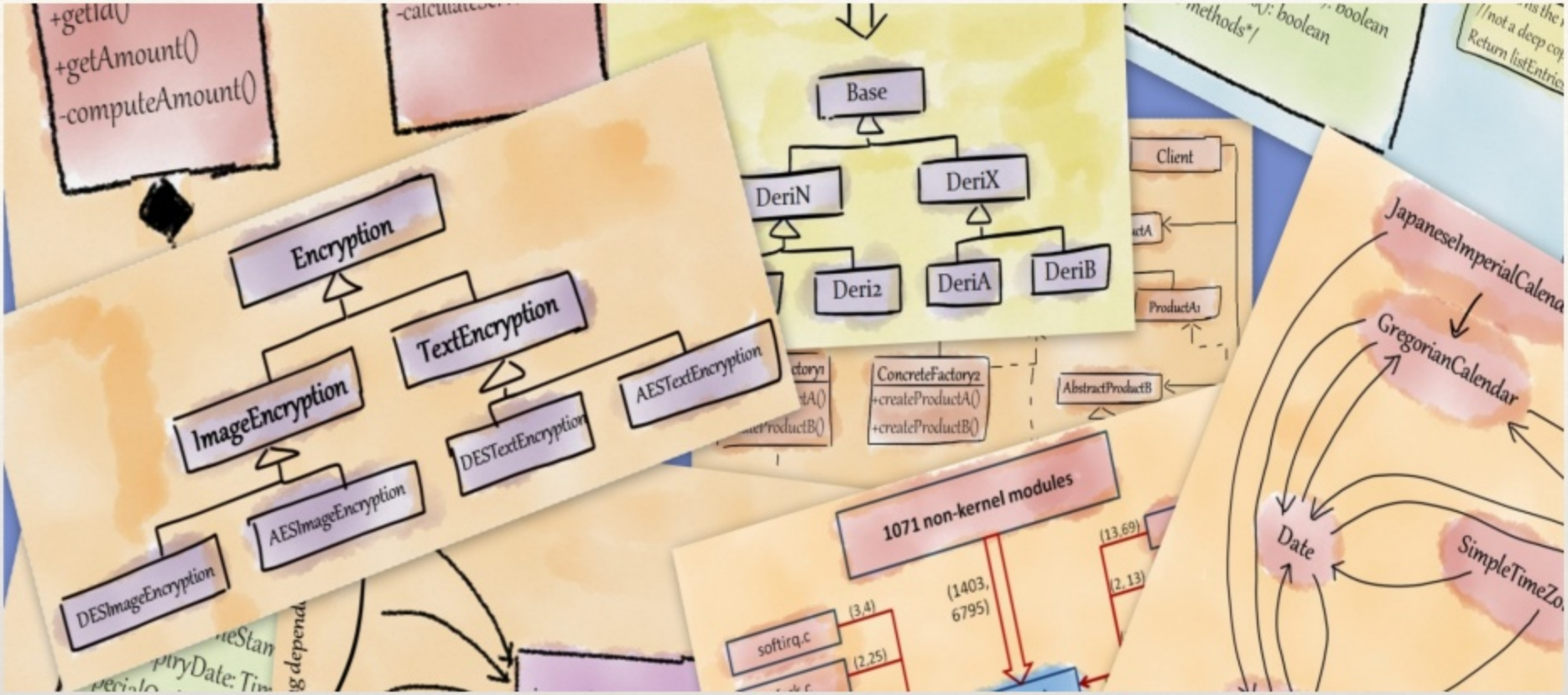


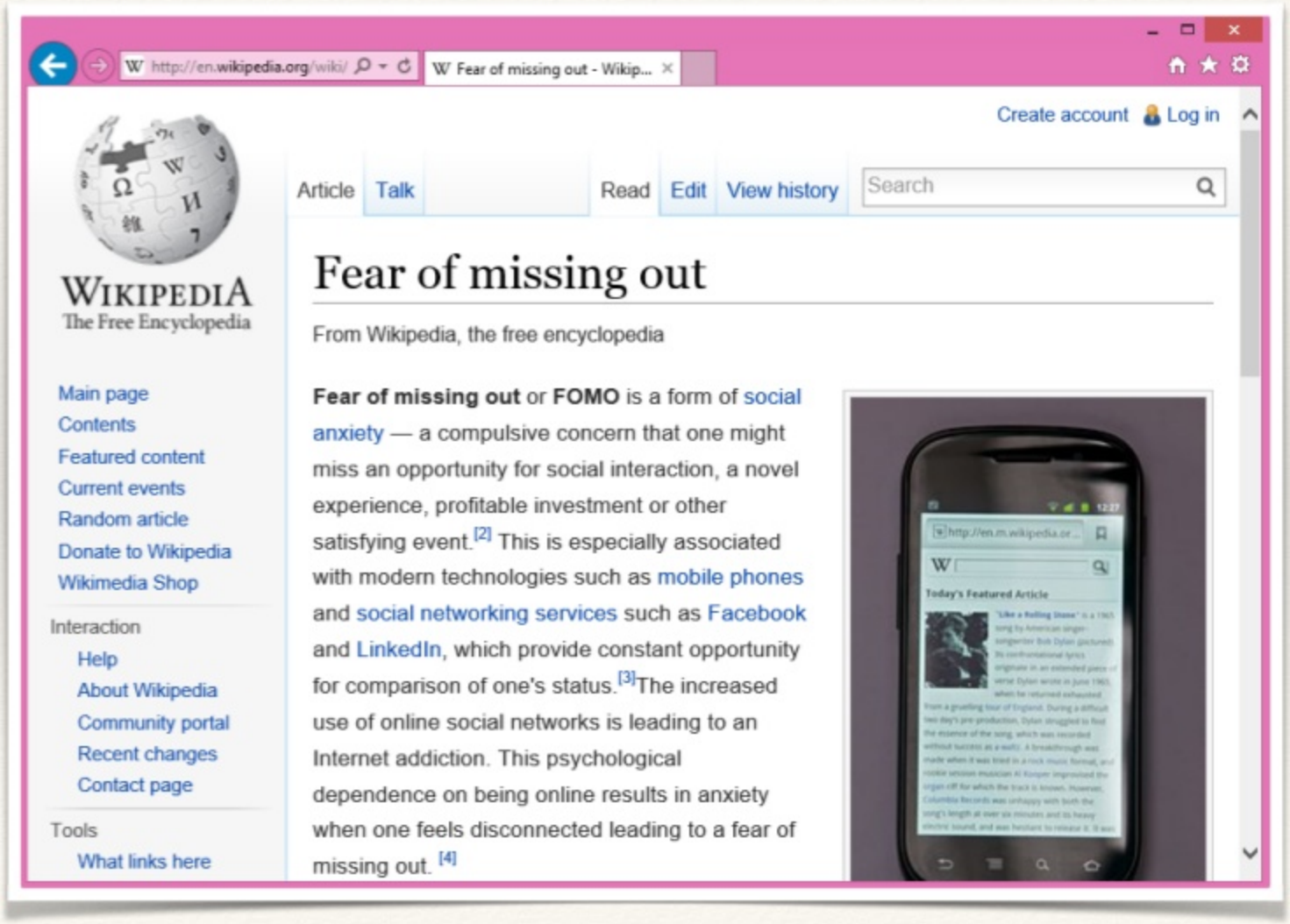
Design Patterns Through Refactoring

Ganesh Samarthyam
ganesh.samarthyam@gmail.com

"Applying design principles is the key to creating high-quality software!"

Architectural principles:
Axis, symmetry, rhythm, datum, hierarchy, transformation

Technology changes fast => FOMO



For architects: design is the key!

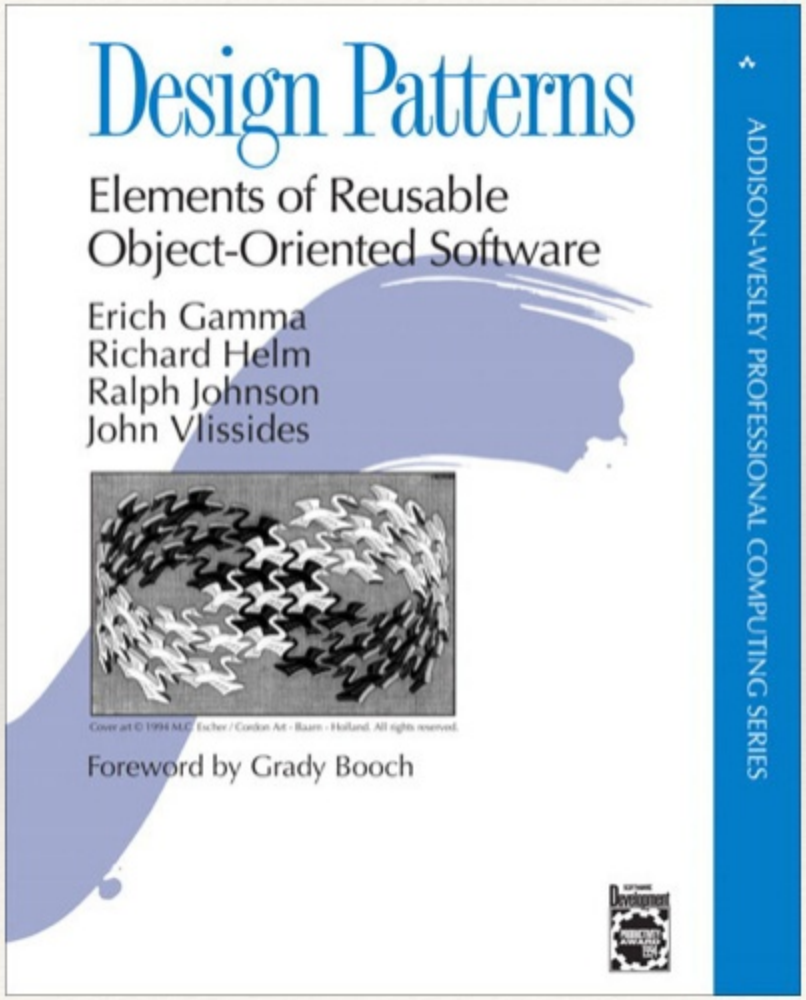
Agenda

- **Introduction**
- Design patterns though exercises
- Patterns through a case-study
- Wrap-up & key-takeaways

What are design patterns?

recurrent solutions
to common
design problems

Pattern Name
Problem
Solution
Consequences



Why care about patterns?

- Patterns capture expert knowledge in the form of proven reusable solutions
  - Better to reuse proven solutions than to "re-invent" the wheel
- When used correctly, patterns positively influence software quality
  - Creates maintainable, extensible, and reusable code

Design pattern catalog

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method	Adapter (class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Flyweight Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

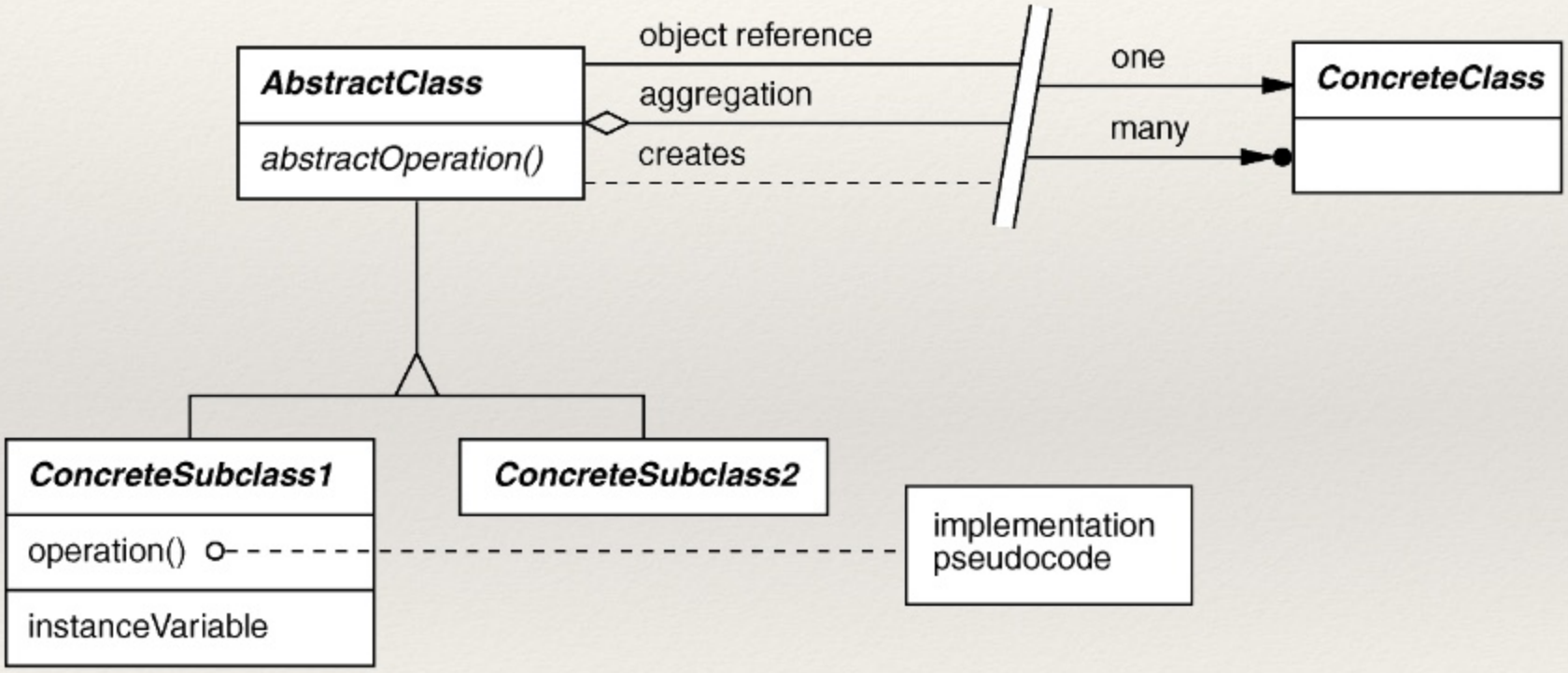
Source: "Design Patterns: Elements of Reusable Object-Oriented Software", Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Addison-Wesley, 1994

Design pattern catalog

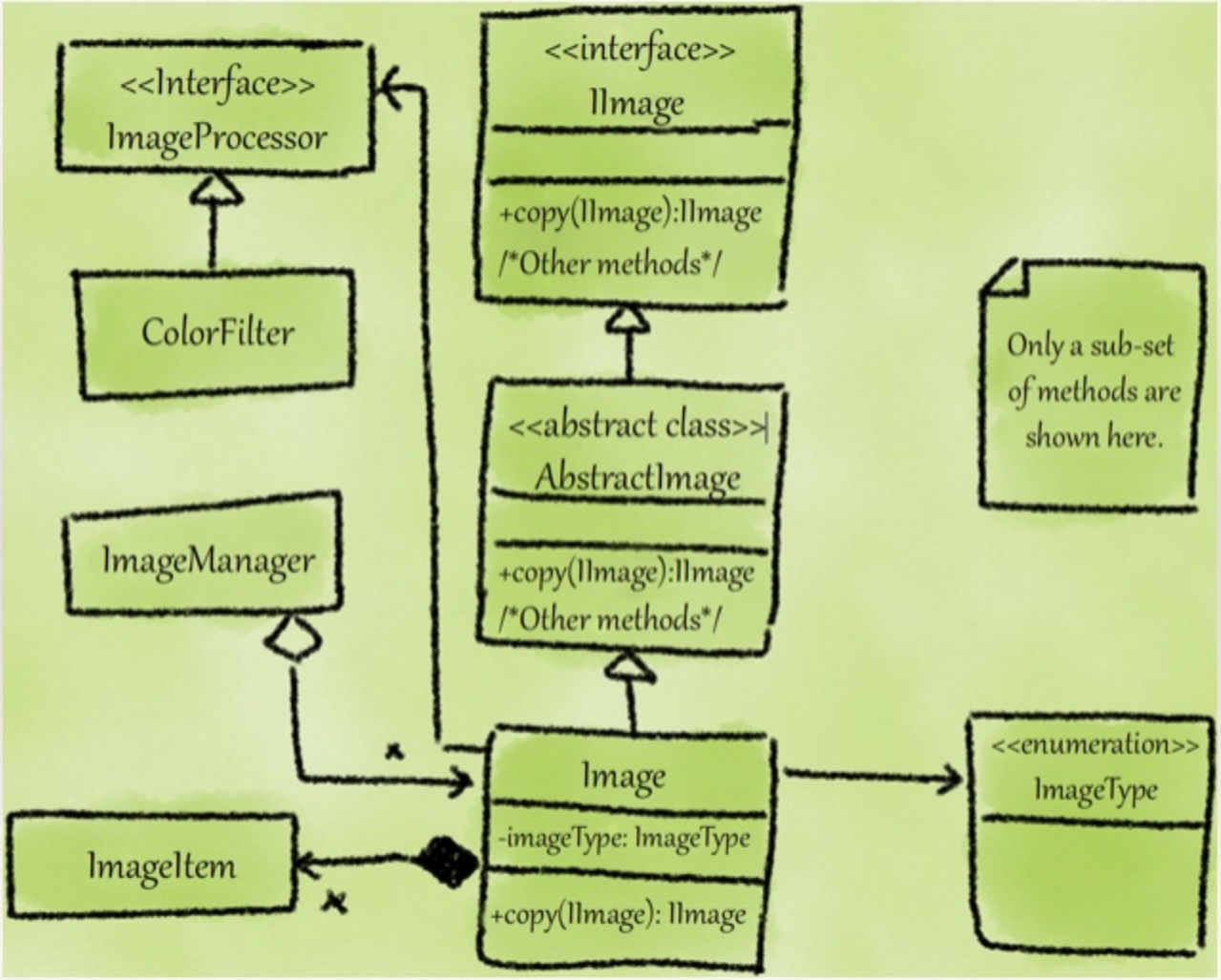
Creational	Deals with controlled object creation	*Factory method*, for example
Structural	Deals with composition of classes or objects	*Composite*, for example
Behavioral	Deals with interaction between objects / classes and distribution of responsibility	*Strategy*, for example

Source: "Design Patterns: Elements of Reusable Object-Oriented Software", Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Addison-Wesley, 1994

5 minutes intro to notation



An example



Source: "Refactoring for Software Design Smells: Managing Technical Debt", Girish Suryanarayana, Ganesh Samarthyam, Tushar Sharma, Morgan Kaufmann/Elsevier, 2014

3 principles behind patterns

Program to an interface, not to an implementation

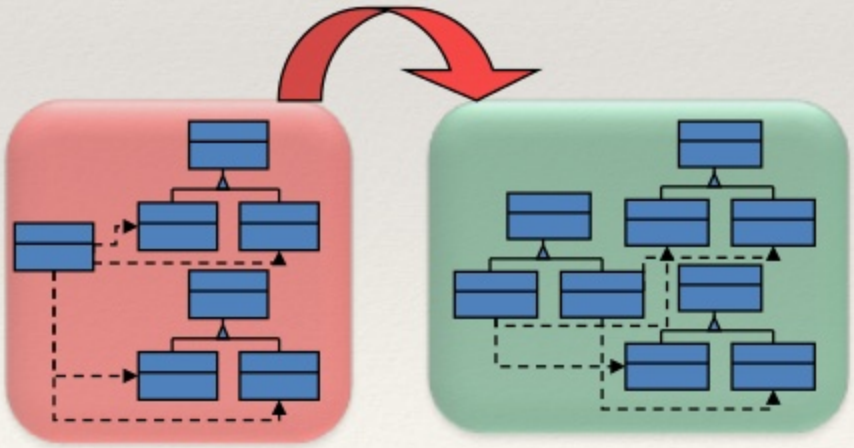
Favor object composition over inheritance

Encapsulate what varies

What is refactoring?

Refactoring (noun): a *change* made to the *internal structure* of software to make it *easier to understand and cheaper to modify* without *changing its observable behavior*

Refactor (verb): to restructure software by applying a series of refactorings without changing its observable behavior



Cover key patterns through examples

It is not about the number of patterns you know, but how well you understand “why, when, where, and how” to apply them effectively

Smells approach to learn patterns

Agenda

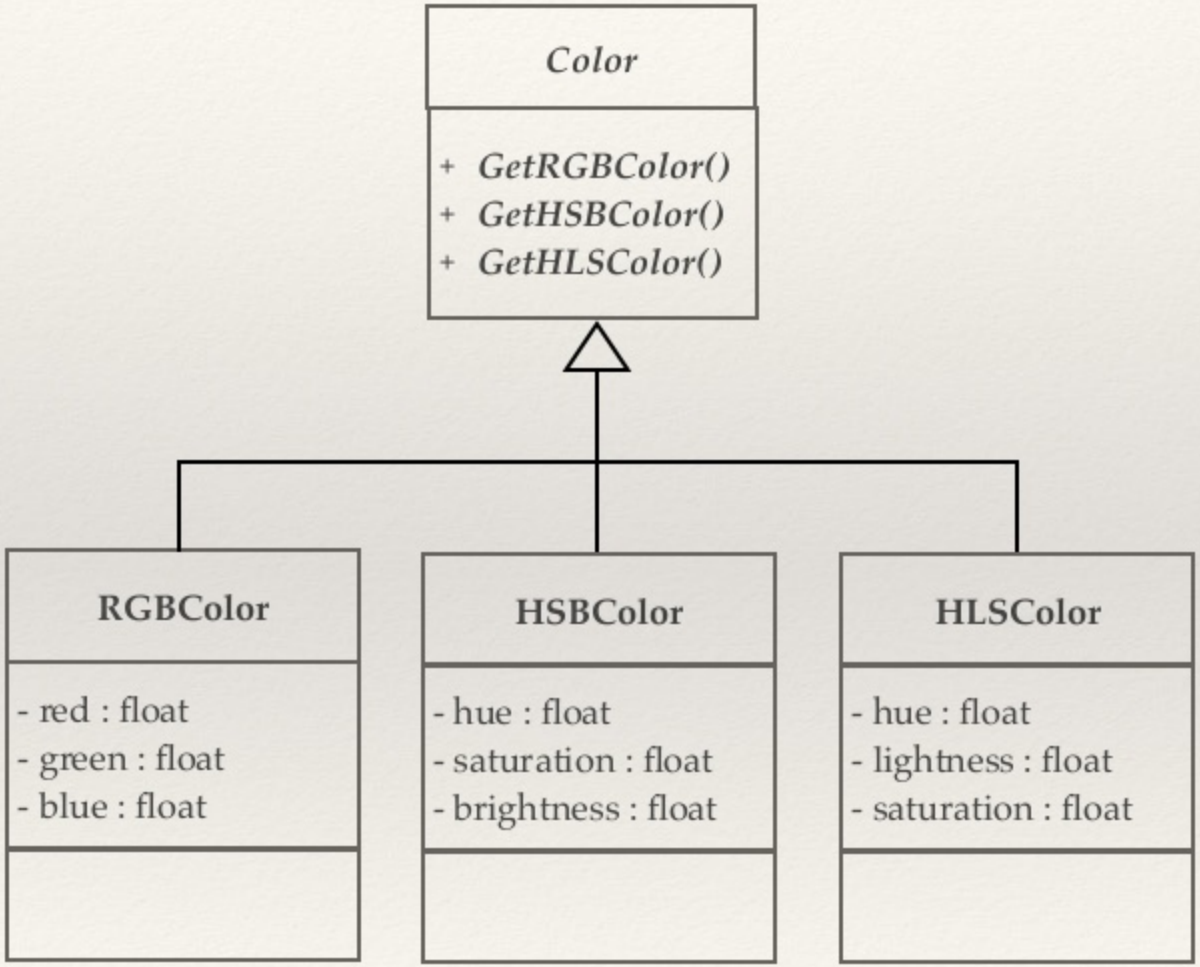
- Introduction
- **Design patterns though exercises**
- Patterns through a case-study
- Wrap-up & key-takeaways

Scenario

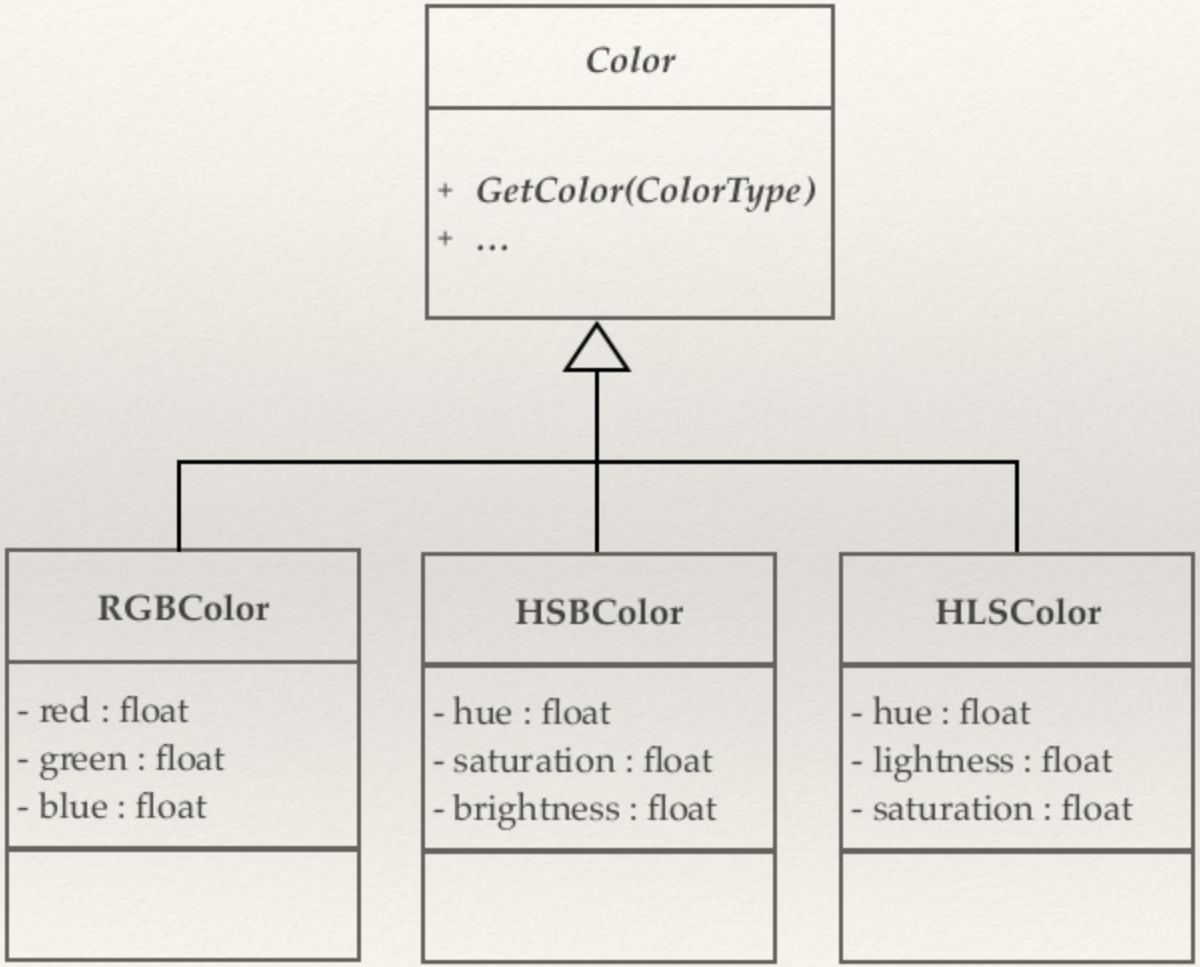
- Assume that you need to support different Color schemes in your software
  - RGB (Red, Green, Blue), HSB (Hue, Saturation, Brightness), and HLS (Hue, Lightness, and Saturation) schemes
- Overloading constructors and differentiating them using enums can become confusing
- What could be a better design?

```
enum ColorScheme { RGB, HSB, HLS, CMYK }
class Color {
        private float red, green, blue;                         // for supporting RGB scheme
        private float hue1, saturation1, brightness1;       // for supporting HSB scheme
        private float hue2, lightness2, saturation2;        // for supporting HLS scheme
        public Color(float arg1, float arg2, float arg3, ColorScheme  cs) {
                switch (cs) {
                        // initialize arg1, arg2, and arg3 based on ColorScheme value
                }
        }
}
```

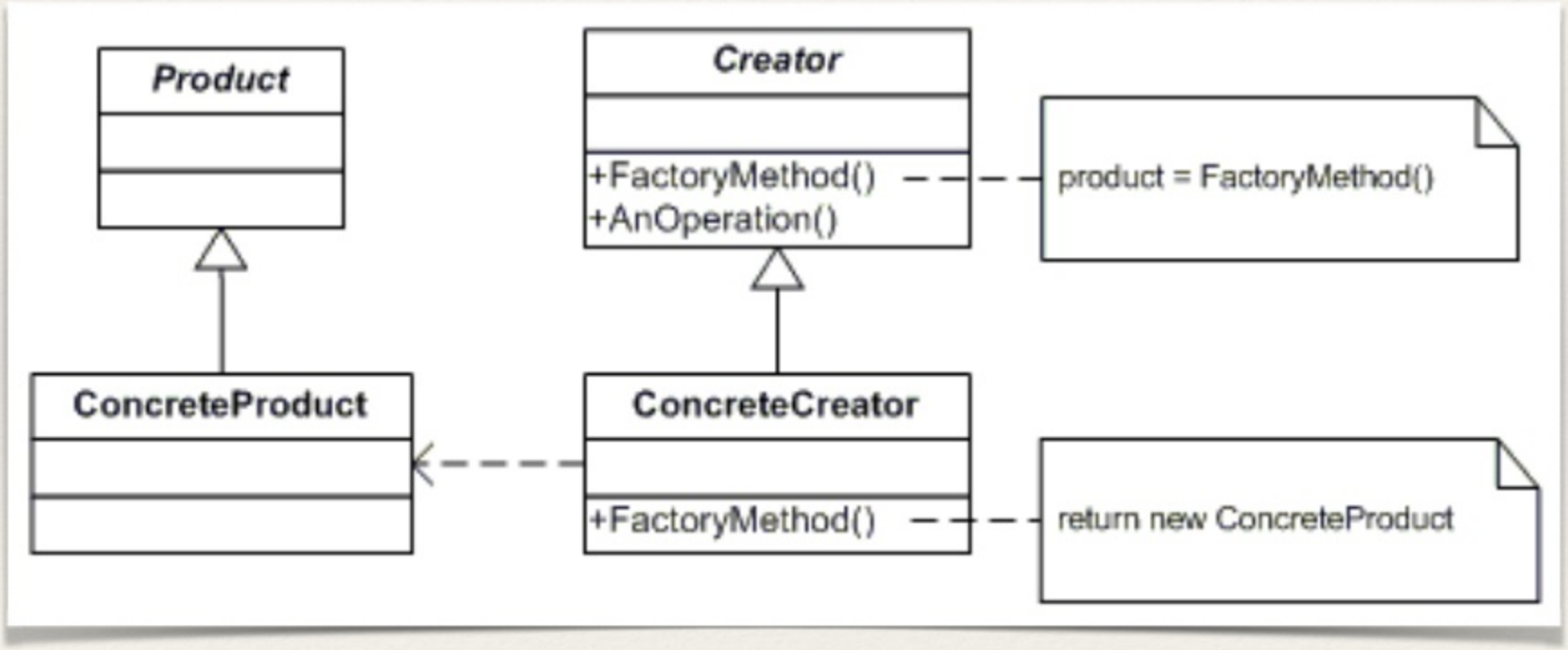
A solution using factory method pattern



A solution using factory method pattern



Factory method pattern: Structure



Source: "Design Patterns: Elements of Reusable Object-Oriented Software", Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Addison-Wesley, 1994