Head First Design Patterns

by Eric Freeman, Elisabeth Freeman, Kathy Sierra, and Bert Bates

ch13. Better Living with Patterns

Design Pattern defined

A Pattern is a solution to a problem in a context.

All patterns in a catalog start with a name. The name is a vital part of a pattern – without a good name, a pattern can't become part of the vocabulary that you share with other developers.

The motivation gives you a concrete scenario that describes the problem and how the solution solves the problem.

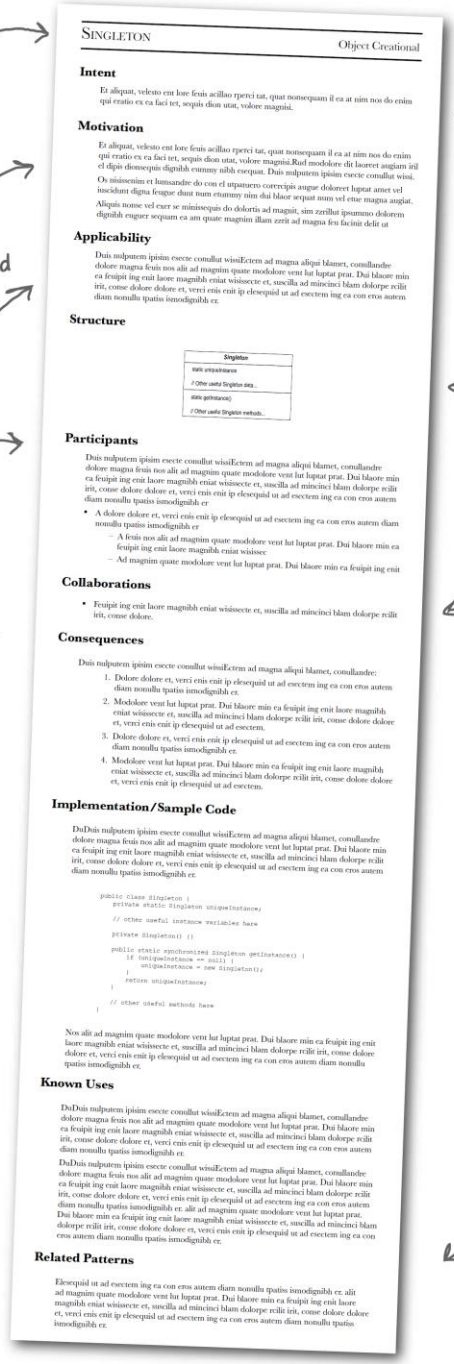
The applicability describes situations in which the pattern can be applied.

The participants are the classes and objects in the design. This section describes their responsibilities and roles in the pattern.

The consequences describe the effects that using this pattern may have: good and bad.

Implementation provides techniques you need to use when implementing this pattern, and issues you should watch out for.

Known uses describes examples of this pattern found in real systems.



SINGLETON — Object Creational

Intent

Et aliquat, velesto ent lore feuis acillao rperci tat, quat nonsequaam il ea at nim nos do enim qui eratio ex ea faci tet, sequis dion utat, volore magnisi.

Motivation

Et aliquat, velesto ent lore feuis acillao rperci tat, quat nonsequaam il ea at nim nos do enim qui eratio ex ea faci tet, sequis dion utat, volore magnisi.Rud modolore dit laoreet augiam iril el dipis dionsequis dignibh euismmy nibh esequat. Duis nulputem ipisim esecte comullut wissi.

Os nissisenim et lumsandre do con el utpatuero corercipis augue doloreet luptat amet vel tincidunt digna feugue dunt num etuimmy nim dui blaor sequat num vel etue magna augiat.

Aliquis nonse vel exer se minissequis do dolortis ad magnit, sim zzrillut ipsummo dolorem dignibh euguer sequam ea am quate magnim illam zzrit ad magna feu faciniit delit ut

Applicability

Duis nulputem ipisim esecte comullut wissiEctem ad magna aliqui blamet, comullandre dolore magna feuis nos alit ad magnim quate modolore vent lut luptat prat. Dui blaore min ea feuipit ing enit laore magnibh eniat wisissecte et, suscilla ad mincinci blam dolorpe rcilit irit, conse dolore dolore et, verci enis enit ip elesequisl ut ad esectem ing ea con eros autem diam nonullu tpatiss ismodignibh ee

Structure

Participants

Duis nulputem ipisim esecte comullut wissiEctem ad magna aliqui blamet, comullandre dolore magna feuis nos alit ad magnim quate modolore vent lut luptat prat. Dui blaore min ea feuipit ing enit laore magnibh eniat wisissecte et, suscilla ad mincinci blam dolorpe rcilit irit, conse dolore dolore et, verci enis enit ip elesequisl ut ad esectem ing ea con eros autem diam nonullu tpatiss ismodignibh ee

- A dolore dolore et, verci enis enit ip elesequisl ut ad esectem ing ea con eros autem diam nonullu tpatiss ismodignibh ee
  - A feuis nos alit ad magnim quate modolore vent lut luptat prat. Dui blaore min ea feuipit ing enit laore magnibh eniat wisissec
  - Ad magnim quate modolore vent lut luptat prat. Dui blaore min ea feuipit ing enit

Collaborations

- Feuipit ing enit laore magnibh eniat wisissecte et, suscilla ad mincinci blam dolorpe rcilit irit, conse dolore.

Consequences

Duis nulputem ipisim esecte comullut wissiEctem ad magna aliqui blamet, comullandre:

1. Dolore dolore et, verci enis enit ip elesequisl ut ad esectem ing ea con eros autem diam nonullu tpatiss ismodignibh et.
2. Modolore vent lut luptat prat. Dui blaore min ea feuipit ing enit laore magnibh eniat wisissecte et, suscilla ad mincinci blam dolorpe rcilit irit, conse dolore dolore et, verci enis enit ip elesequisl ut ad esectem.
3. Dolore dolore et, verci enis enit ip elesequisl ut ad esectem ing ea con eros autem diam nonullu tpatiss ismodignibh ee.
4. Modolore vent lut luptat prat. Dui blaore min ea feuipit ing enit laore magnibh eniat wisissecte et, suscilla ad mincinci blam dolorpe rcilit irit, conse dolore dolore et, verci enis enit ip elesequisl ut ad esectem.

Implementation/Sample Code

DuDuis nulputem ipisim esecte comullut wissiEctem ad magna aliqui blamet, comullandre dolore magna feuis nos alit ad magnim quate modolore vent lut luptat prat. Dui blaore min ea feuipit ing enit laore magnibh eniat wisissecte et, suscilla ad mincinci blam dolorpe rcilit irit, conse dolore dolore et, verci enis enit ip elesequisl ut ad esectem ing ea con eros autem diam nonullu tpatiss ismodignibh ee

```
public class Singleton {
   private static Singleton uniqueInstance;

   // other useful instance variables here

   private Singleton() {}

   public static synchronized Singleton getInstance() {
      if (uniqueInstance == null) {
         uniqueInstance = new Singleton();
      }
      return uniqueInstance;
   }

   // other useful methods here
}
```

Nos alit ad magnim quate modolore vent lut luptat prat. Dui blaore min ea feuipit ing enit laore magnibh eniat wisissecte et, suscilla ad mincinci blam dolorpe rcilit irit, conse dolore dolore et, verci enis enit ip elesequisl ut ad esectem ing ea con eros autem diam nonullu tpatiss ismodignibh ee.

Known Uses

DuDuis nulputem ipisim esecte comullut wissiEctem ad magna aliqui blamet, comullandre dolore magna feuis nos alit ad magnim quate modolore vent lut luptat prat. Dui blaore min ea feuipit ing enit laore magnibh eniat wisissecte et, suscilla ad mincinci blam dolorpe rcilit irit, conse dolore dolore et, verci enis enit ip elesequisl ut ad esectem ing ea con eros autem diam nonullu tpatiss ismodignibh et.

DuDuis nulputem ipisim esecte comullut wissiEctem ad magna aliqui blamet, comullandre dolore magna feuis nos alit ad magnim quate modolore vent lut luptat prat. Dui blaore min ea feuipit ing enit laore magnibh eniat wisissecte et, suscilla ad mincinci blam dolorpe rcilit irit, conse dolore dolore et, verci enis enit ip elesequisl ut ad esectem ing ea con eros autem diam nonullu tpatiss ismodignibh ee. alit ad magnim quate modolore vent lut luptat prat. Dui blaore min ea feuipit ing enit laore magnibh eniat wisissecte et, suscilla ad mincinci blam dolorpe rcilit irit, conse dolore dolore et, verci enis enit ip elesequisl ut ad esectem ing ea con eros autem diam nonullu tpatiss ismodignibh ee.

Related Patterns

Elesequisl ut ad esectem ing ea con eros autem diam nonullu tpatiss ismodignibh ee. alit ad magnim quate modolore vent lut luptat prat. Dui blaore min ea feuipit ing enit laore magnibh eniat wisissecte et, suscilla ad mincinci blam dolorpe rcilit irit, conse dolore dolore et, verci enis enit ip elesequisl ut ad esectem ing ea con eros autem diam nonullu tpatiss ismodignibh ee.

This is the pattern's classification or category. We'll talk about these in a few pages.

The intent describes what the pattern does in a short statement. You can also think of this as the pattern's definition (just like we've been using in this book).

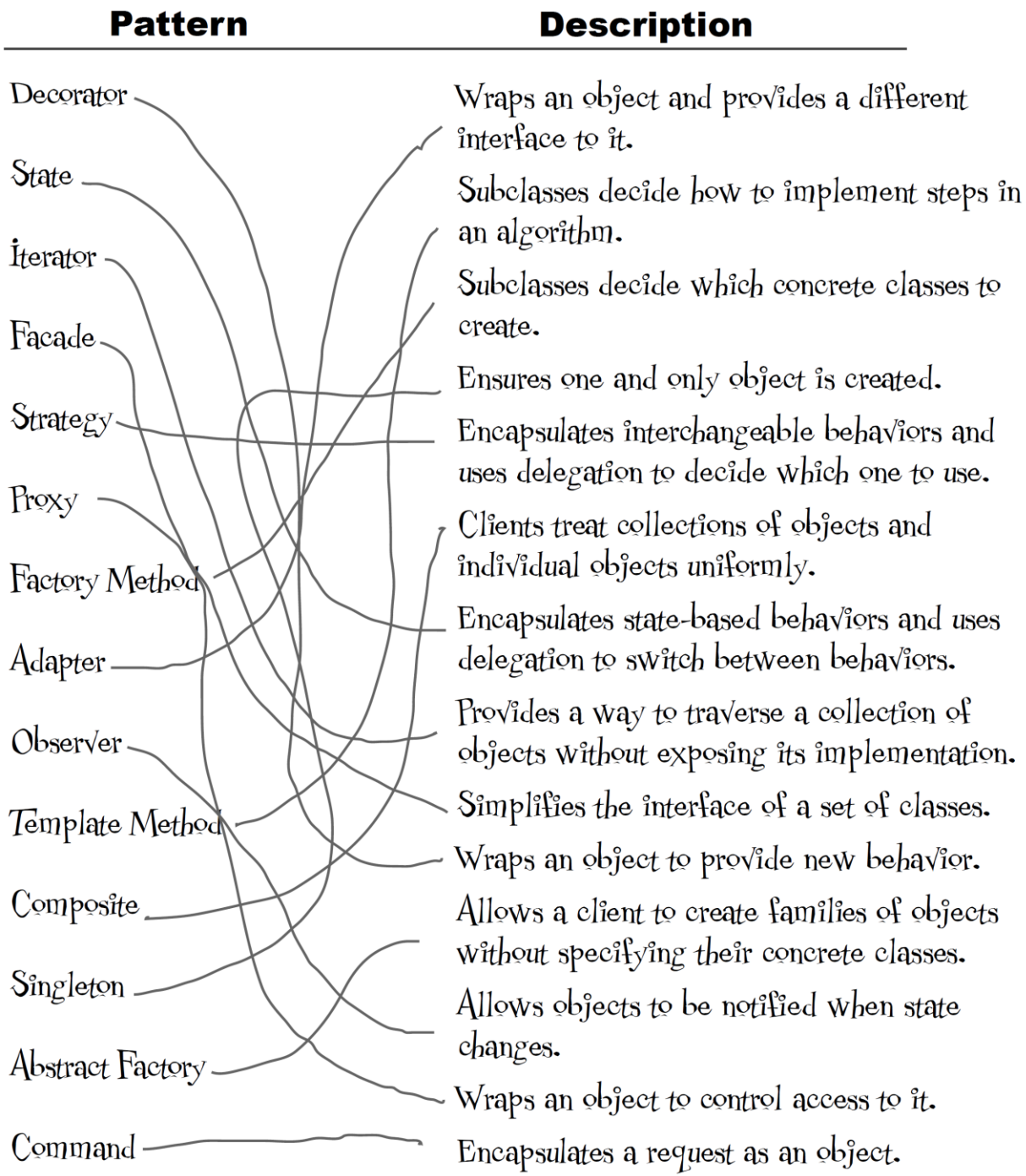
The structure provides a diagram illustrating the relationships among the classes that participate in the pattern.

Collaborations tells us how the participants work together in the pattern.

Sample code provides code fragments that might help with your implementation.

Related patterns describes the relationship between this pattern and others.

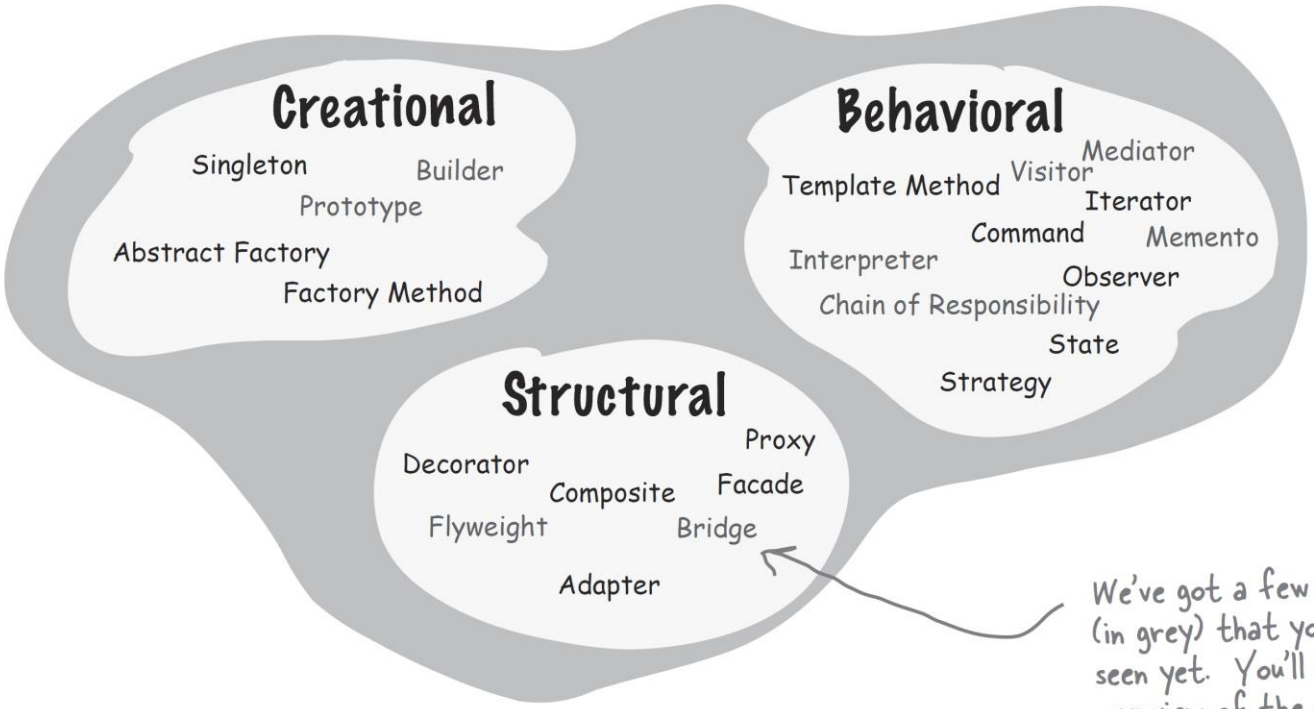
Pattern	Description
Decorator	Wraps an object and provides a different interface to it.
State	Subclasses decide how to implement steps in an algorithm.
Iterator	Subclasses decide which concrete classes to create.
Facade	Ensures one and only object is created.
Strategy	Encapsulates interchangeable behaviors and uses delegation to decide which one to use.
Proxy	Clients treat collections of objects and individual objects uniformly.
Factory Method	Encapsulates state-based behaviors and uses delegation to switch between behaviors.
Adapter	Provides a way to traverse a collection of objects without exposing its implementation.
Observer	Simplifies the interface of a set of classes.
Template Method	Wraps an object to provide new behavior.
Composite	Allows a client to create families of objects without specifying their concrete classes.
Singleton	Allows objects to be notified when state changes.
Abstract Factory	Wraps an object to control access to it.
Command	Encapsulates a request as an object.


Pattern
Description
Decorator
State
Iterator
Facade
Strategy
Proxy
Factory Method
Adapter
Observer
Template Method
Composite
Singleton
Abstract Factory
Command
Wraps an object and provides a different interface to it.
Subclasses decide how to implement steps in an algorithm.
Subclasses decide which concrete classes to create.
Ensures one and only object is created.
Encapsulates interchangeable behaviors and uses delegation to decide which one to use.
Clients treat collections of objects and individual objects uniformly.
Encapsulates state-based behaviors and uses delegation to switch between behaviors.
Provides a way to traverse a collection of objects without exposing its implementation.
Simplifies the interface of a set of classes.
Wraps an object to provide new behavior.
Allows a client to create families of objects without specifying their concrete classes.
Allows objects to be notified when state changes.
Wraps an object to control access to it.
Encapsulates a request as an object.

Organizing Design Patterns Pattern Categories

Creational patterns involve object instantiation and all provide a way to decouple a client from the objects it needs to instantiate.

Any pattern that is a **Behavioral Pattern** is concerned with how classes and objects interact and distribute responsibility.

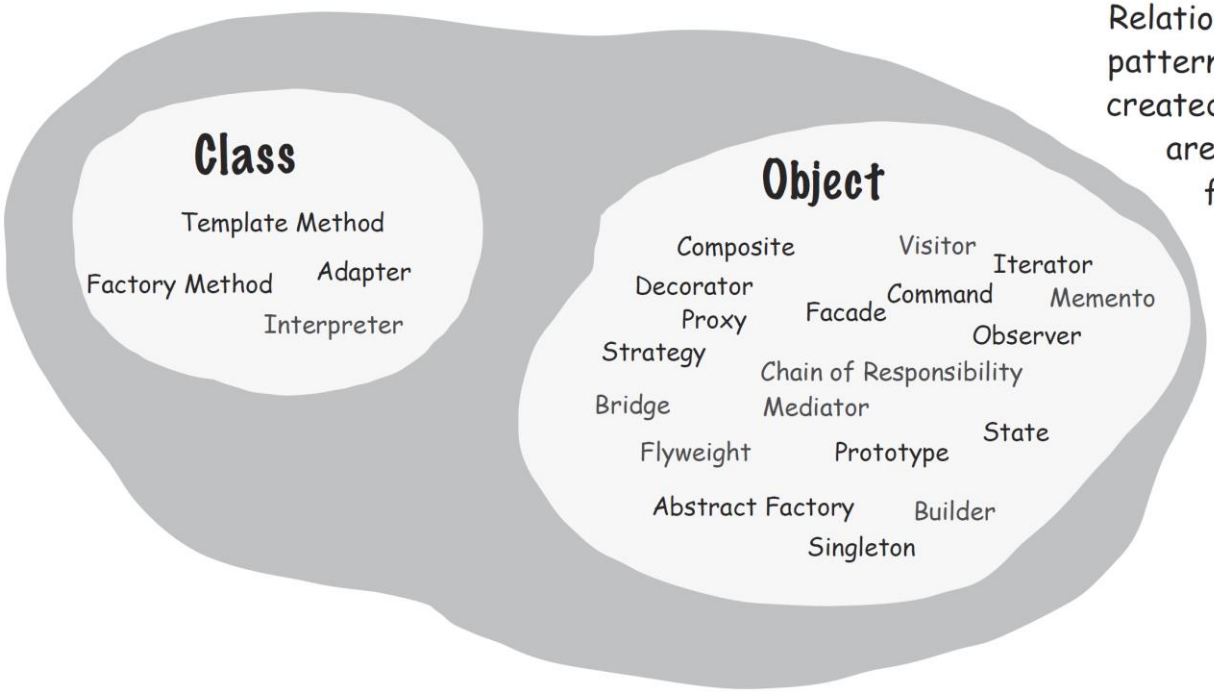


We've got a few patterns (in grey) that you haven't seen yet. You'll find an overview of the these patterns in the appendix.

Structural patterns let you compose classes or objects into larger structures.

Class patterns describe how relationships between classes are defined via inheritance. Relationships in class patterns are established at compile time.

Object patterns describe relationships between objects and are primarily defined by composition. Relationships in object patterns are typically created at runtime and are more dynamic and flexible.



Notice there's a lot more object patterns than class patterns!

Thinking in Patterns

Design Patterns aren't a magic bullet; in fact they're not even a bullet!

Refactoring time is Patterns time!

Take out what you don't really need. Don't be afraid to remove a Design Pattern from your design.

If you don't need it now, don't do it now.

Your Mind on Patterns

BEGINNER MIND

"I need a pattern for Hello World."

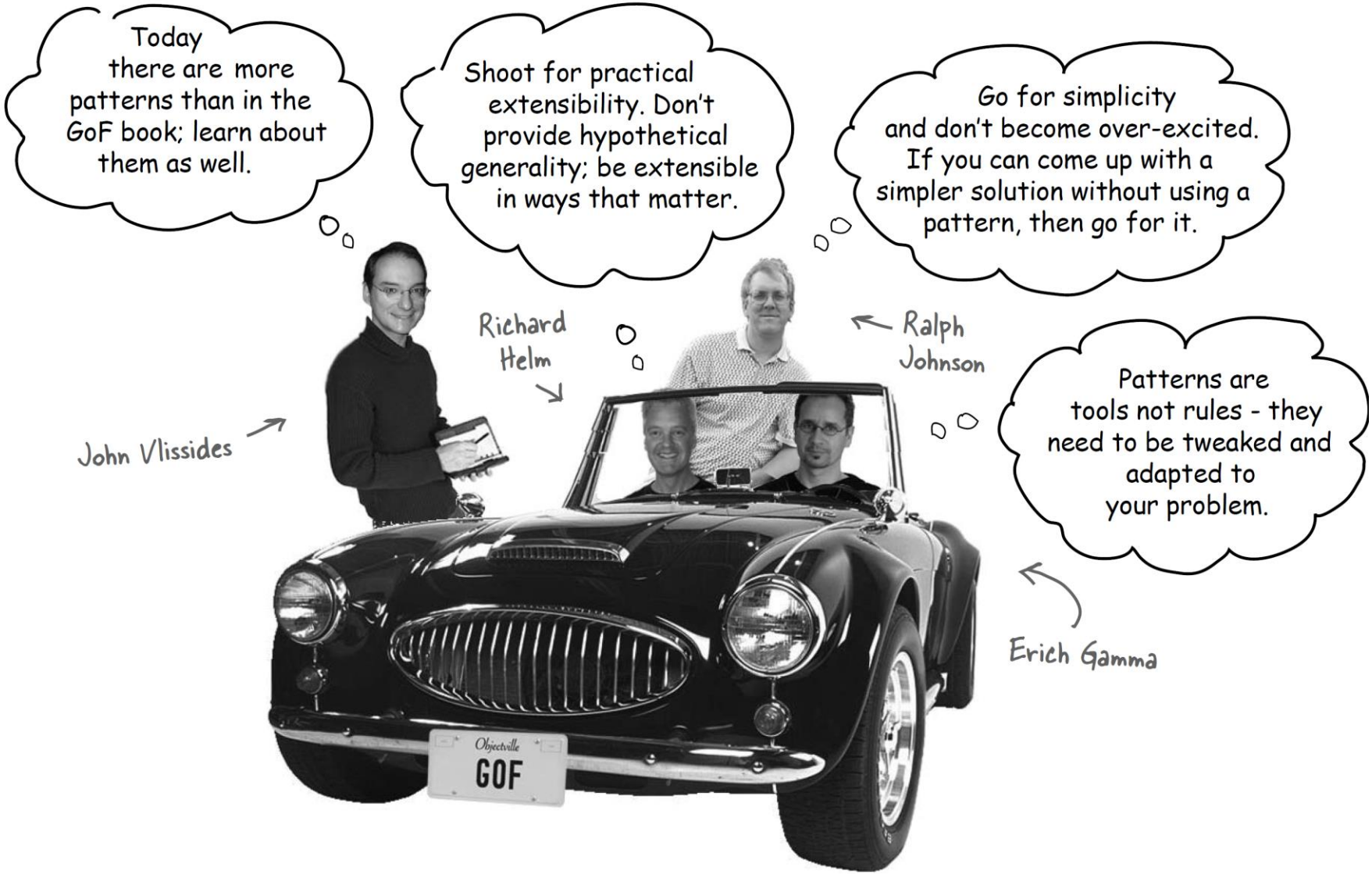
INTERMEDIATE MIND

"Maybe I need a Singleton here."

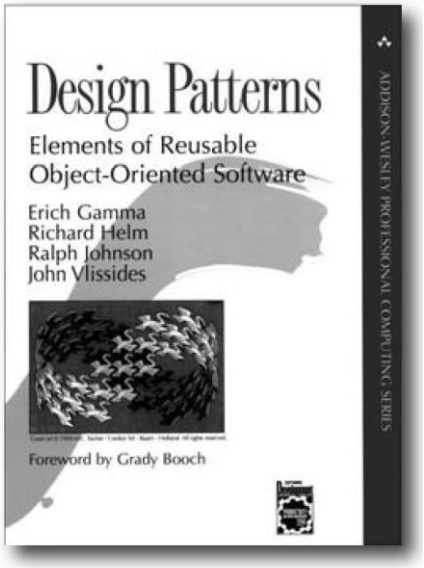
ZEN MIND

"This is a natural place for Decorator."

Don't forget the power of the shared vocabulary


Today there are more patterns than in the GoF book; learn about them as well.
Shoot for practical extensibility. Don't provide hypothetical generality; be extensible in ways that matter.
Go for simplicity and don't become over-excited. If you can come up with a simpler solution without using a pattern, then go for it.
Richard Helm
Ralph Johnson
John Vlissides
Patterns are tools not rules - they need to be tweaked and adapted to your problem.
Erich Gamma
Objectville
GOF

Your journey has just begun...



The definitive Design Patterns text

The authors of Design Patterns are affectionately known as the "Gang of Four" or GoF for short.

The Patterns Zoo

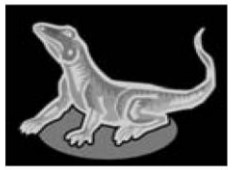
Architectural Patterns are used to create the living, vibrant architecture of buildings, towns, and cities. This is where patterns got their start.

Habitat: found in buildings you like to live in, look at and visit.

Habitat: seen hanging around 3-tier architectures, client-server systems and the web.

Application Patterns are patterns for creating system level architecture. Many multi-tier architectures fall into this category.

Field note: MVC has been known to pass for an application pattern.

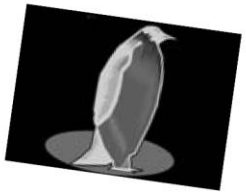


Domain-Specific Patterns are patterns that concern problems in specific domains, like concurrent systems or real-time systems.

Business Process Patterns describe the interaction between businesses, customers and data, and can be applied to problems such as how to effectively make and communicate decisions.

Seen hanging around corporate boardrooms and project management meetings.

Organizational Patterns describe the structures and practices of human organizations. Most efforts to date have focused on organizations that produce and/or support software.



User Interface Design Patterns address the problems of how to design interactive software programs.

Habitat: seen in the vicinity of video game designers, GUI builders, and producers.

Annihilating evil with Anti-Patterns

An **Anti-Pattern** tells you how to go from a problem to a BAD solution.

An anti-pattern always looks like a good solution, but then turns out to be a bad solution when it is applied.

By documenting anti-patterns we help others to recognize bad solutions before they implement them.

Like patterns, there are many types of anti-patterns including development, OO, organizational, and domain specific anti-patterns.