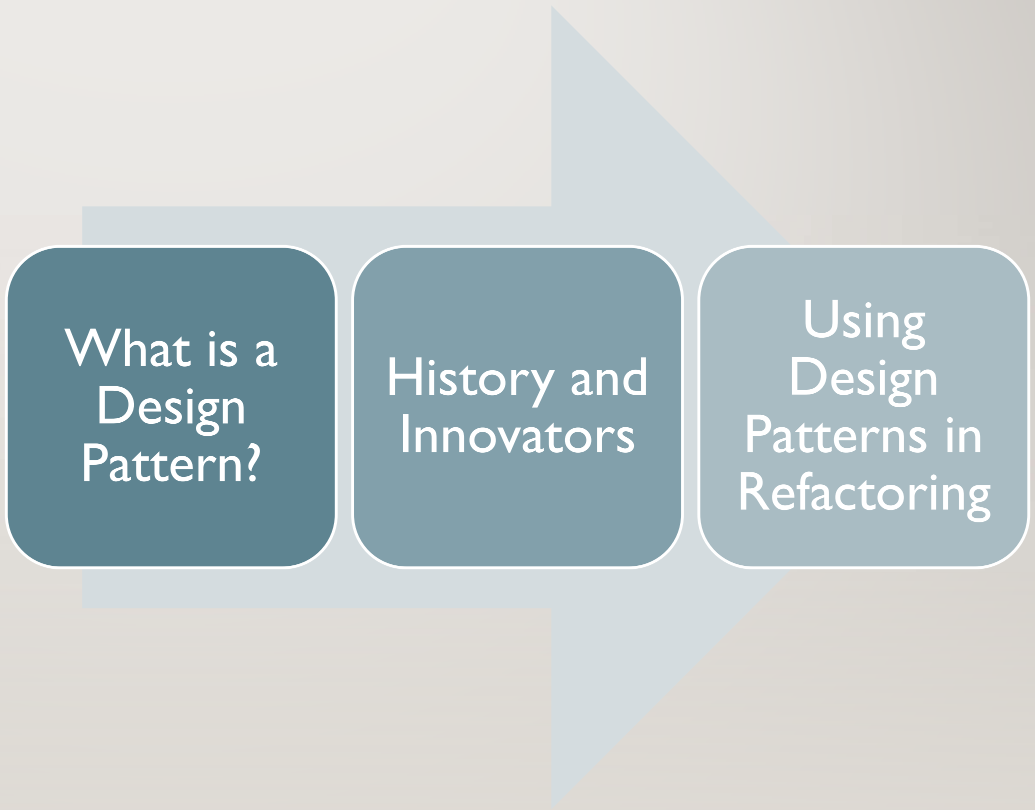
SOFTWARE DESIGN PATTERNS: REUSABLE TOOLS

OBJECT-ORIENTED ANALYSIS AND DESIGN

Thelma Looms
16 November 2017

OVERVIEW


What is a Design Pattern?
History and Innovators
Using Design Patterns in Refactoring

CHALLENGES IN OO DESIGN AND DEVELOPMENT

How do we:

- Solve problems efficiently and quickly
- Make designs reusable
- Minimize re-design
- Provide flexibility
- Communicate relationships

WHAT IS A DESIGN PATTERN?

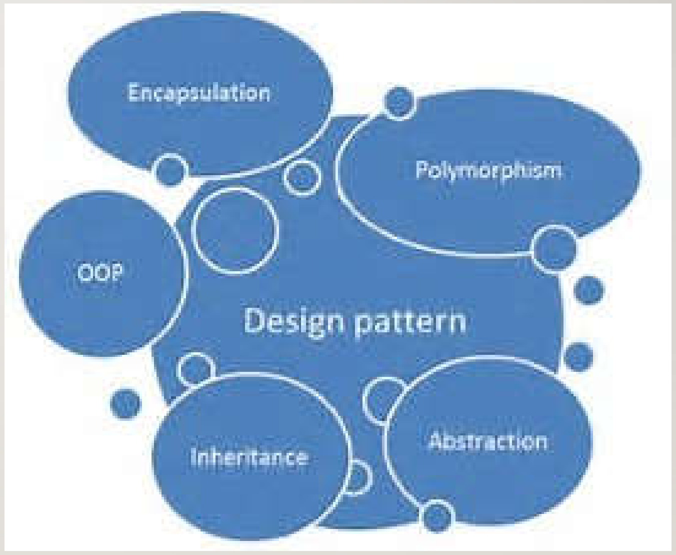
"In software engineering, a software design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. "

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to the problem in such away that you can use this solution a million times over."

"A pattern language guides a designer by providing workable solutions to all of the problems known to arise in the course of design. … written in a style and arranged in an order which leads a designer to ask (and answer) the right questions at the right time."

WHAT IS A DESIGN PATTERN?

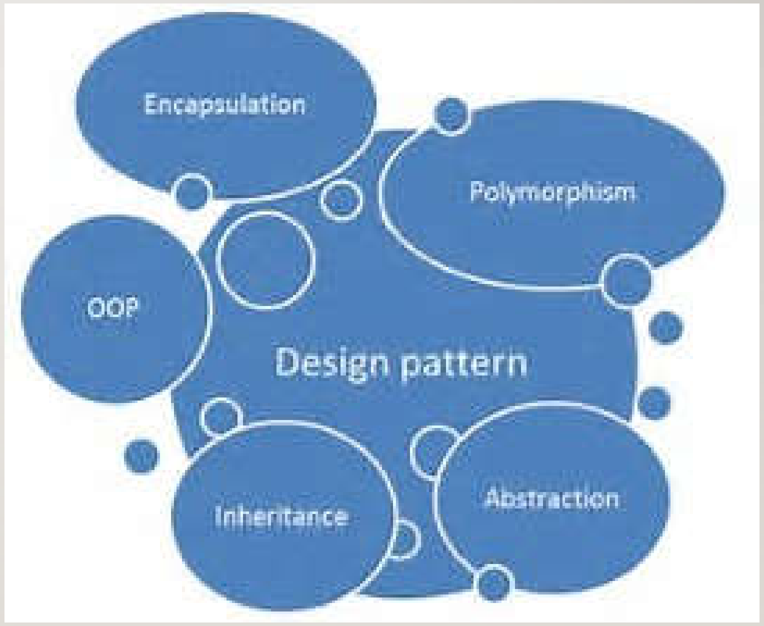
- Solution – time-tested solutions for recurring problems
- Design Vocabulary – shared language to support design
- Choose Alternatives – when it should be used and tradeoffs
- Reusable – over and over again
- Toolbox – application independent



“A Design pattern is a way of understanding OOP with scenarios.”

WHY USE DESIGN PATTERNS?

- Best Practices – found after decades of experience
- Design – from success not failure
- Cost – designing from scratch is expensive
- Accelerate – not designing concepts from scratch
- Standardized communication and documentation – by providing a design vocabulary



TIMELINE AND INNOVATORS

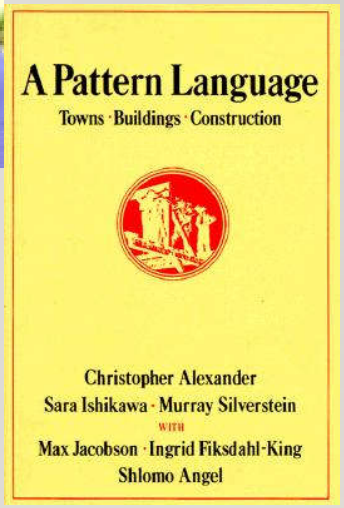
- **1977** — Christopher W. Alexander (Architecture)
- **1980** — Trygve M. H. Reenskaug (MVC)
- **1987** — Ward Cunningham and Kent Beck (Coding Patterns)
- **1994** — Portland Pattern Repository (PPR)
- **1995** — **Gang of Four**: Gamma, Helm, Johnson, Vlissides (23 Patterns)
- **1996** — **Party of Five**: Buschman, Meunier, Rohnert, Sommerlad, Stal
- **1997** — **Three Amigos:** Booch, Jacobson, Rumbaugh (UML)
- **1997** — Craig Larman (GRASP)

PATTERN LANGUAGE : ARCHITECTURE

HISTORY AND INNOVATORS

- **1977:** Christopher Alexander, *A Pattern Language, Oxford University Press, 1977*
- Introduces architectural patterns in architecture for urban design, buildings and towns
- 253 patterns each consisting of a design problem, discussion, illustration, and solution

"These tools allow anyone, and any group of people, to create beautiful, functional, meaningful places."


A Pattern Language

Towns · Buildings · Construction

Christopher Alexander

Sara Ishikawa · Murray Silverstein

WITH

Max Jacobson · Ingrid Fiksdahl-King

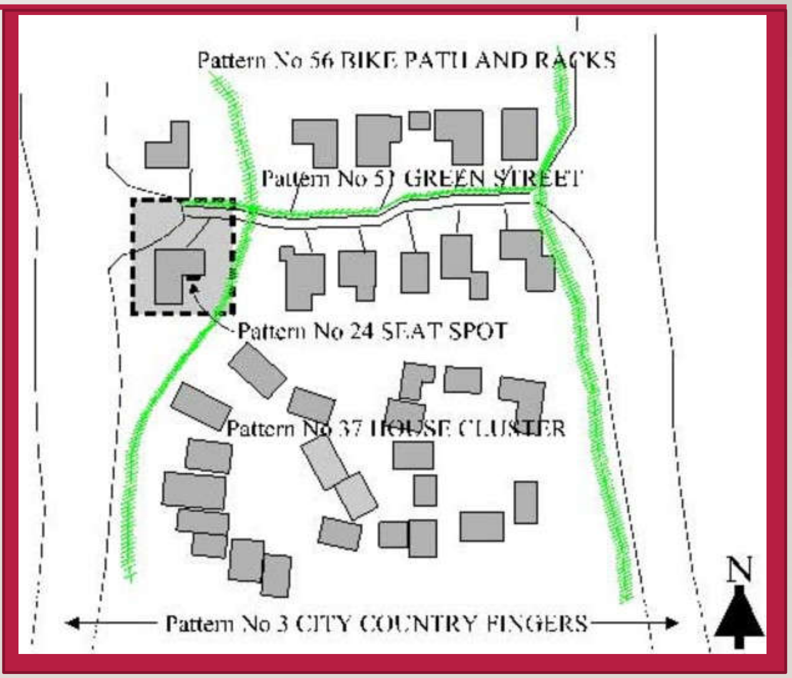
Shlomo Angel

TIMELINE AND INNOVATORS

PATTERN LANGUAGE : ARCHITECTURE

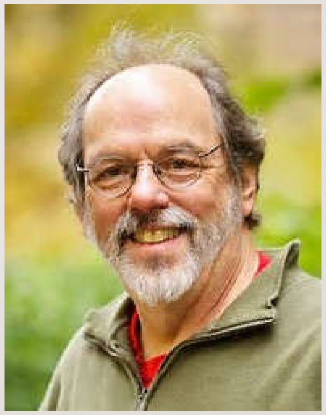
- Patterns 21 – 27 “Local Environment”
- Patterns 35 – 40 “Housing Clusters”
- Patterns 49 – 56 “Local Networks”
- Patterns 87 – 94 “Gathering Places and Shops”

http://www.jacana.plus.com/pattern/P0.htm.

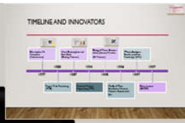

Pattern No 56 BIKE PATH AND RACKS
Pattern No 51 GREEN STREET
Pattern No 24 SEAT SPOT
Pattern No 37 HOUSE CLUSTER
Pattern No 3 CITY COUNTRY FINGERS
N

HISTORY AND INNOVATORS

- **1987:** Wade Cunningham and Kent Beck, *Using Pattern Languages for Object-Oriented Programs*, OOPSLA-87
- Used concepts from Christopher Alexander
- Pattern language for describing graphical user interface design
- Beck: Coding Patterns
- **C**lass-**R**esponsibility-**C**ollaborator brainstorming tool to describe:
  - class of objects
  - behaviors
  - interactions



"*Our initial success using a pattern language for user interface design has left us quite enthusiastic about the possibilities for computer users designing and programming their own applications.*" http://c2.com/doc/oopsla87.html



KENT BECK: CODING PATTERNS, 1995.

- Domain: 89 Coding Patterns for Smalltalk Language
- Recipe of Programming Solutions
- Goal: Able to develop code faster and with less risk that is easier to maintain and reuse

Smalltalk Best Practice Patterns Volume 1: Coding, Kent Beck, 1995, 1996.

Problem: *How do you provide access to an instance variable? How can we change the value?*

Solution: *Provide a method that returns or sets the value of the variable.*

Pattern: “Getting and Setting Method”

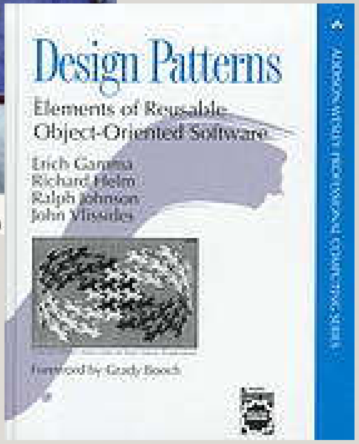
Example: C# property implementation

```
private int _x;
public int x {
 get { return _x; }
 set { _x = value; }
}
someObject.x = 50;
```

GANG OF FOUR (GOF)

- **1995:** Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.
- Birds of a feather (BoF) session at OOPSLA '90.
- *Design Patterns: Elements of Resuable Object-Oriented Software, Addison-Wesley, 1995.*

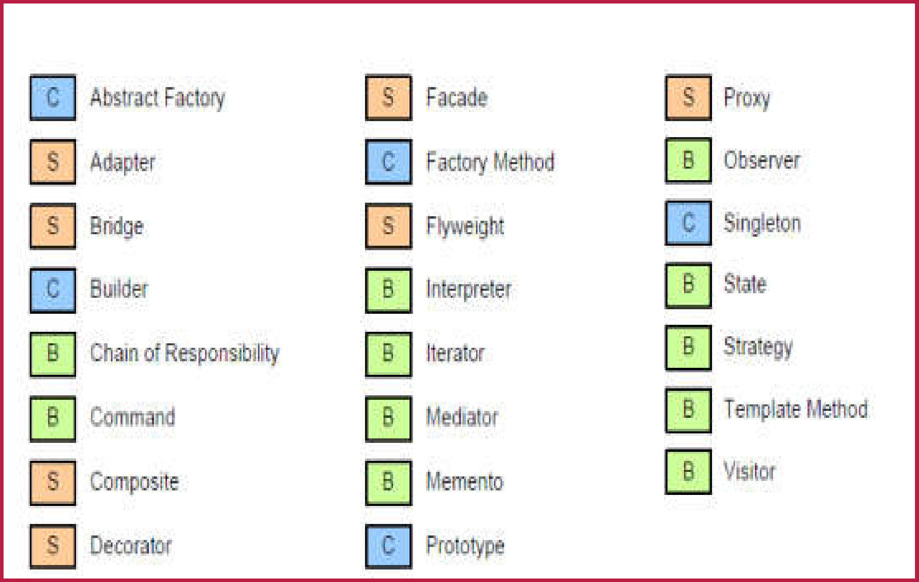
Erich Gamma, Richard Helm, Ralph Johnson and John


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

"The design patterns in this book are description of communicating objects and classes that are customized to solve a general design problem."

GOF PATTERN CLASSIFICATION

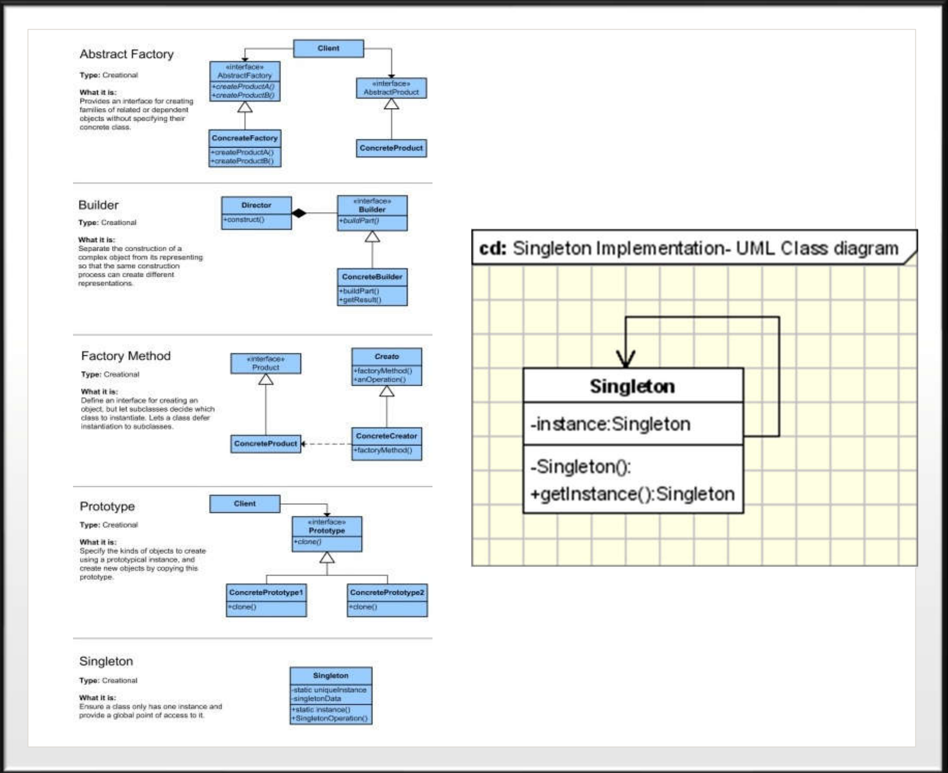
- **Creational Patterns:**
  “object creation”
- **Structural Patterns**
  “composition to form larger and new structures”
- **Behavior Patterns**
  “interaction”
  “communication”



C	Abstract Factory	S	Facade	S	Proxy
S	Adapter	C	Factory Method	B	Observer
S	Bridge	S	Flyweight	C	Singleton
C	Builder	B	Interpreter	B	State
B	Chain of Responsibility	B	Iterator	B	Strategy
B	Command	B	Mediator	B	Template Method
S	Composite	B	Memento	B	Visitor
S	Decorator	C	Prototype		

UML: SINGLETON METHOD PATTERN

- One instance for class, for example a window manager, file system, print spooler
- Provide a global access point to the object



Abstract Factory

Type: Creational

What it is:
Provides an interface for creating families of related or dependent objects without specifying their concrete class.

Builder

Type: Creational

What it is:
Separate the construction of a complex object from its representing so that the same construction process can create different representations.

Factory Method

Type: Creational

What it is:
Define an interface for creating an object, but let subclasses decide which class to instantiate. Lets a class defer instantiation to subclasses.

Prototype

Type: Creational

What it is:
Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Singleton

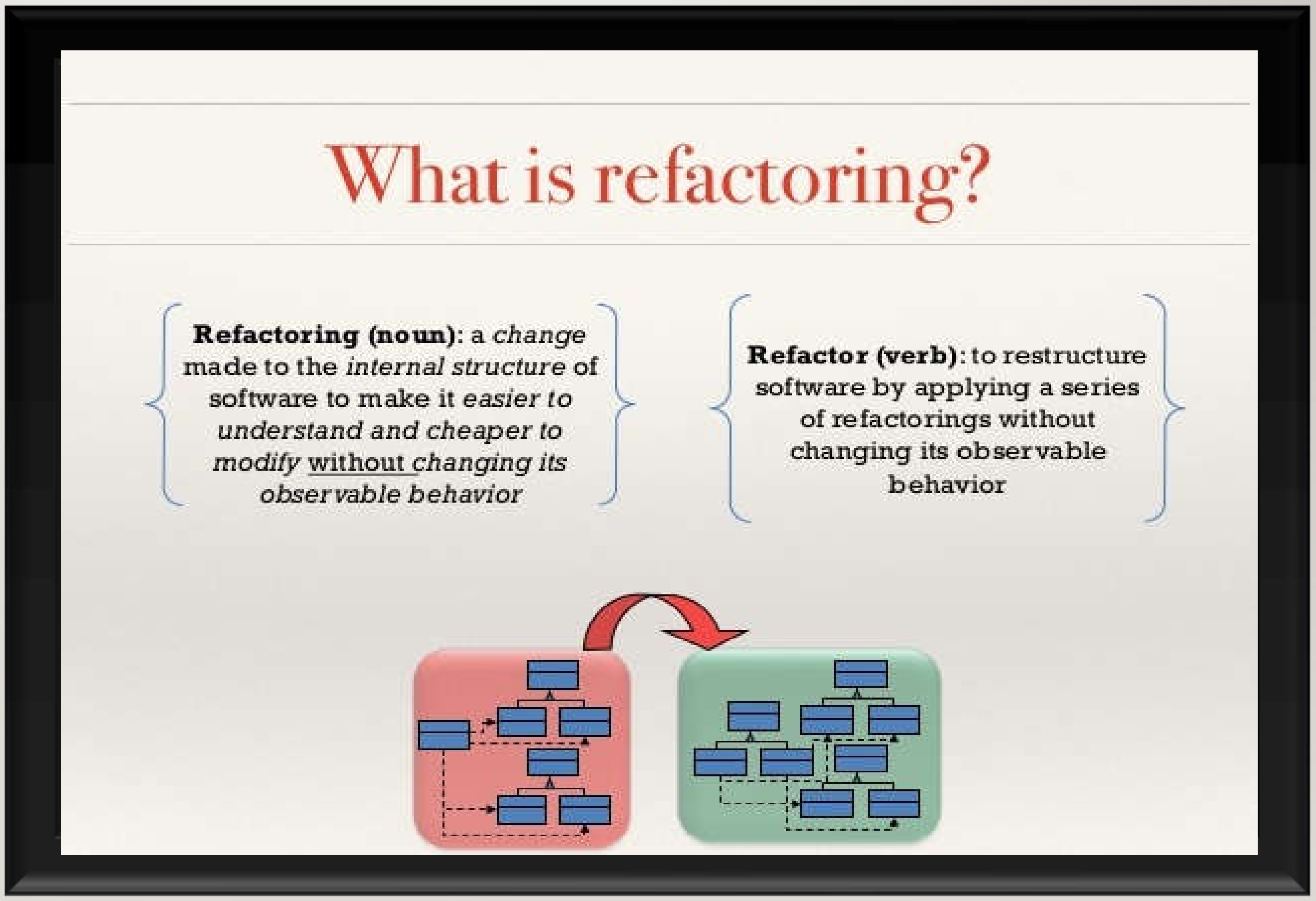
Type: Creational

What it is:
Ensure a class only has one instance and provide a global point of access to it.

THEORY INTO PRACTICE

- Using Patterns in Refactoring
- Case Study

REFACTORING WITH DESIGN PATTERNS



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*

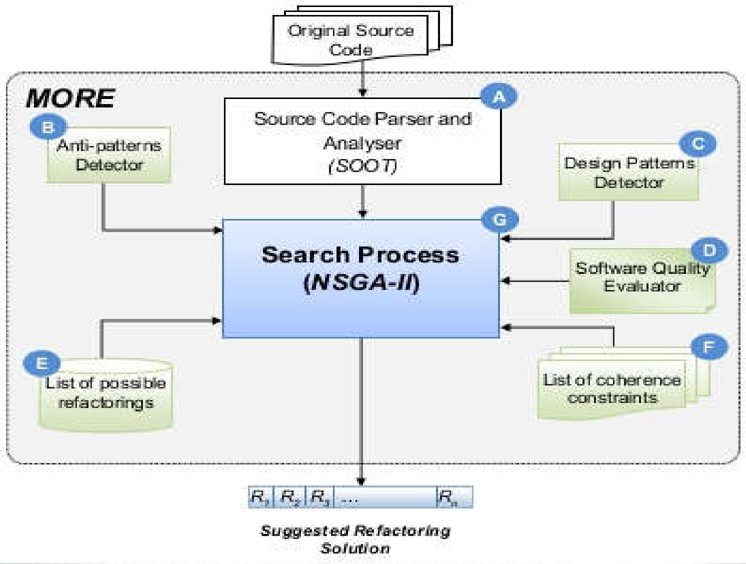
Refactor (verb): to restructure software by applying a series of refactorings without changing its observable behavior

REFACTORING WITH DESIGN PATTERNS

"A Multi-Objective Refactoring Approach to Introduce Design Patterns and Fix Anti-Patterns," Ouni *et al.*

"Automatic Design Pattern Detection," Heuzeroth *et al.*

Approach



Original Source Code
MORE
A
Source Code Parser and Analyser (SOOT)
B
Anti-patterns Detector
C
Design Patterns Detector
G
Search Process (NSGA-II)
D
Software Quality Evaluator
E
List of possible refactorings
F
List of coherence constraints
R_1 R_2 R_3 ... R_n
Suggested Refactoring Solution

Software Engineering Laboratory

7

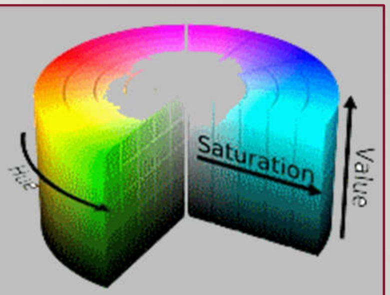
Department of Computer Science, Graduate School of Information Science and Technology, Osaka University

MORE (Multi-Objective Refactoring REcommendation)

CASE STUDY:

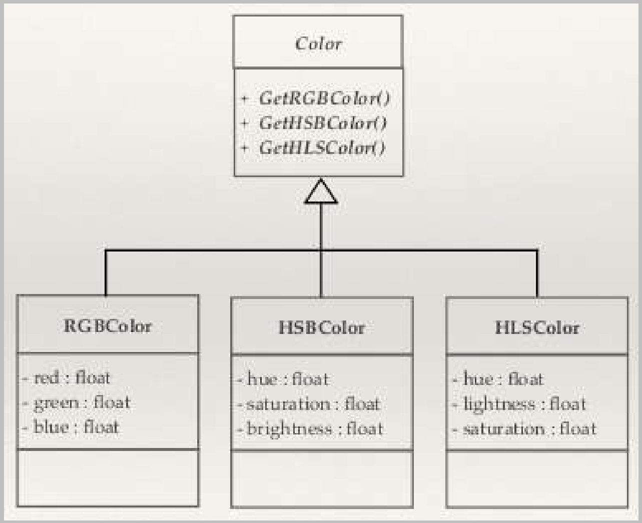
You've just started a new job and your team lead says:
"Here's some code in our graphic system that handles different color schemes. Could you come up with a better design?"

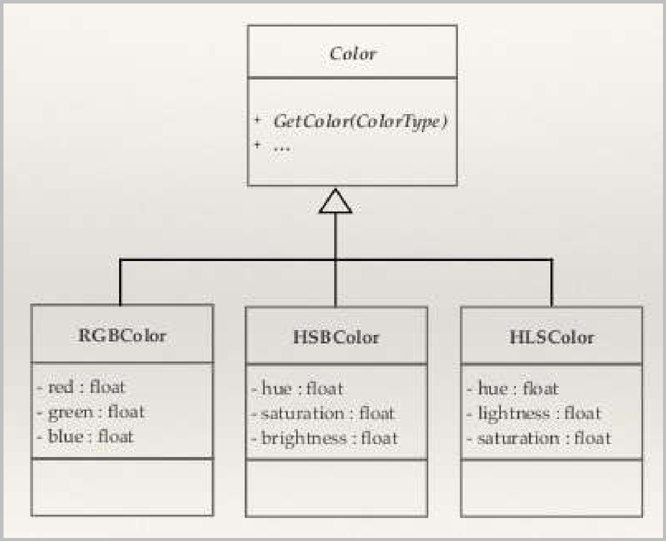
```
enum ColorScheme { RGB, HSB, HLS };
public class Color
{
    private float red, green, blue;      // RGB scheme
    private float hue1, sat1, bright1;  // HSB scheme
    private float hue2, light2, sat2;   // HLS scheme

     public Color(float arg1, float arg2, float arg3, ColorScheme cs)
     {
          switch (cs)
          {
               // init arg1, arg2, and arg3 based on ColorScheme
          }
     }
}
```



CASE STUDY:

Define an interface for creating an object, but let subclasses decide which class to instantiate



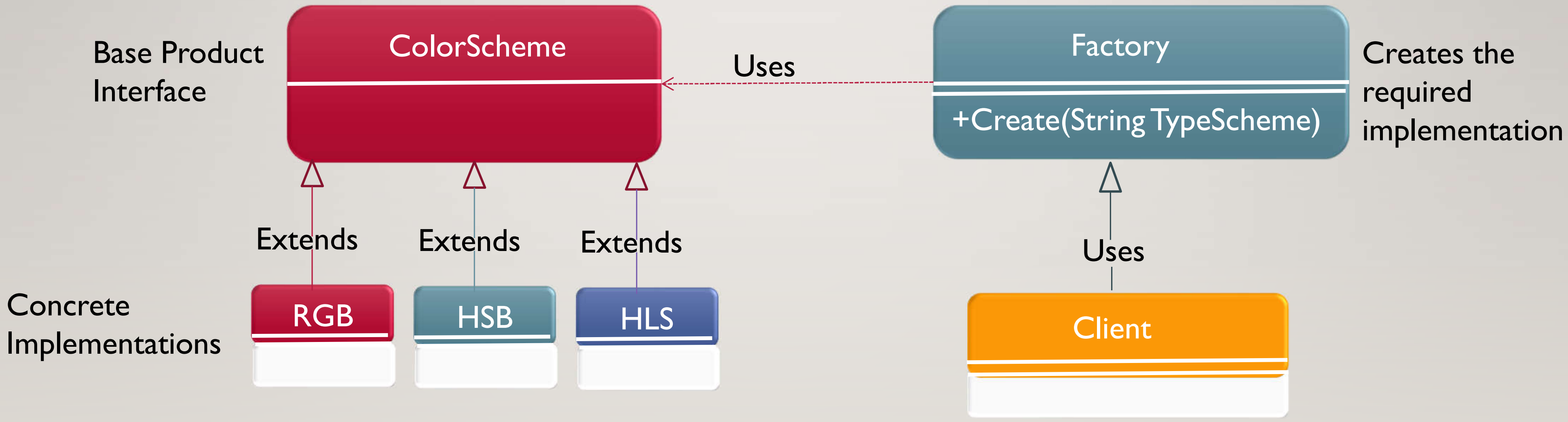


REFACTORING WITH THE FACTORY PATTERN

Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

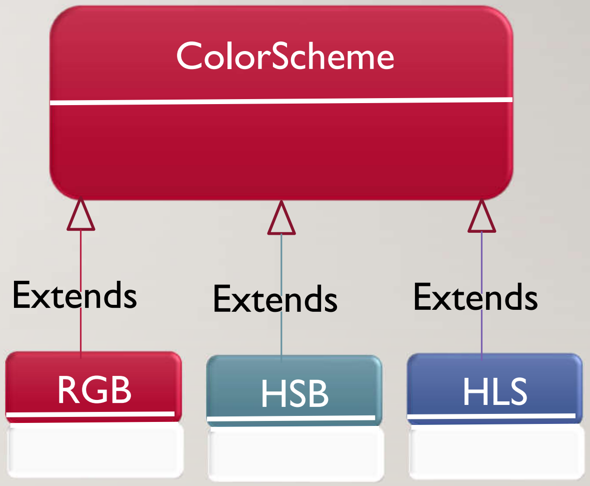
- Factory method pattern is a Creational Pattern.
- There is a base factory interface/base class which defines common methods for creating objects of subclasses.
- The logic for creating different types is contained in the subclass.
- Uses polymorphism in that what is returned is an object of type base class at run time.

CLASS DIAGRAM: FACTORY METHOD PATTERN



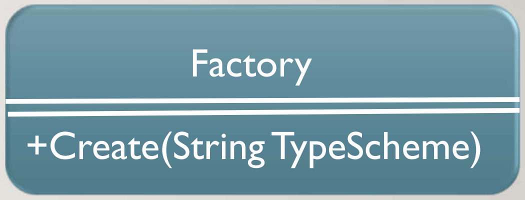
RE-IMPLEMENTATION USING FACTORY PATTERN

```
public class ColorScheme {
        // base class
}

public class RBGScheme : ColorScheme {
        // code for overriding ColorScheme methods
}

public class HSBScheme : ColorScheme {
        // code for overriding ColorScheme methods

}

public class HLSScheme : ColorScheme {
        // code for overriding ColorScheme methods
}
```



RE-IMPLEMENTATION USING FACTORY PATTERN

```
public static class Factory
{
        public static Dictionary<string, ColorScheme> scheme =
                new Dictionary <string, ColorScheme>();
        static Factory()
        {
                scheme.Add( "RGB", new RGBScheme());
                scheme.Add( "HSB", new HSBScheme());
                scheme.Add( "HLS", new HLSScheme());
        }
        public static ColorScheme Create(string TypeScheme)
        {
                return scheme[TypeScheme];
        }
}
```

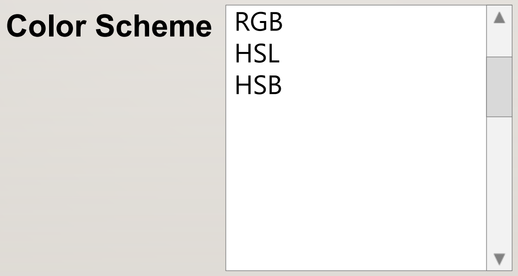


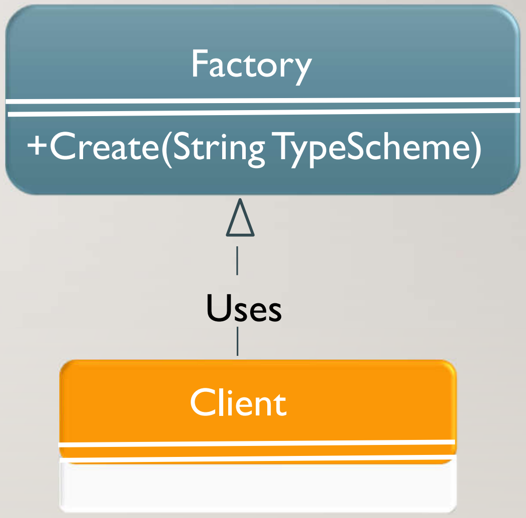
Factory has a **Create** method with an input string that identifies the color scheme type.

Returns an object of type ColorScheme

RE-IMPLEMENTATION USING FACTORY PATTERN

```
public class Client
{
        private ColorScheme selection = null;

        // create a new scheme based on selection

        selection = Factory.Create ( “RGB” );

}
```





CASE STUDY

You've just started a new job and your team lead says:
"Here's some code in our graphic system that handles different color schemes. Could you come up with a better design?"

NEW IMPLEMENTATION

- Uses Base and Concrete Classes
- Factory Pattern to create objects
- RIP (Replace IF with Polymorphism)
- Modular and easily modified

OLD IMPLEMENTATION

- Use Enums and variables
- Constructor with multiple arguments
- *switch* and *if* statements
- Non-modular

SUMMARY

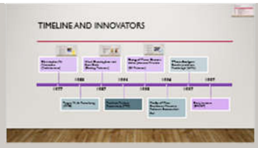
- Patterns are the recurring solutions to the problems of design.
- People learn patterns by seeing them and recall them when need be without a lot of effort.
- Patterns link together in the mind so that one pattern leads to another and another until familiar problems are solved.

http://c2.com/ppr/

Portland Pattern Repository

- Repository of solutions to recurring problems found in computer programming
- Pattern Language Catalogue
- People, Projects & Patterns

THANK YOU



LESSON ACROYNM

Acronym	Definition
CRC	Class-Responsibility-Collaboration
GRASP	General Responsibility Assignment Software
OOAD	Object Oriented Analysis and Design
OOP	Object Oriented Programming
OOPSLA	Object oriented Programming, Systems, Language and Architecture
MVC	Model-View-Controller
NSGA-II	Non-dominated Sorting Genetic Algorithm II
UML	Unified Modeling Language

WORKS CITED

- *A Multi-Objective Refactoring Approach to Introduce Design Patterns and Fix Anti-Patterns:* *http://csserver.ucd.ie/~meloc/papers/nasbase_2015b_final.pdf*
- *Design Patterns Through Refactoring, Ganesh Samarthyam, 2015.*
- *OODesign.com : http://www.oodesign.com/*
- *Table Summary for A Pattern Language:* *http://www.jacana.plus.com/pattern/P0.htm*.
- *.NET Interview Preparation videos*