Design Patterns

宋 杰

Song Jie

东北大学 软件学院

Software College, Northeastern University

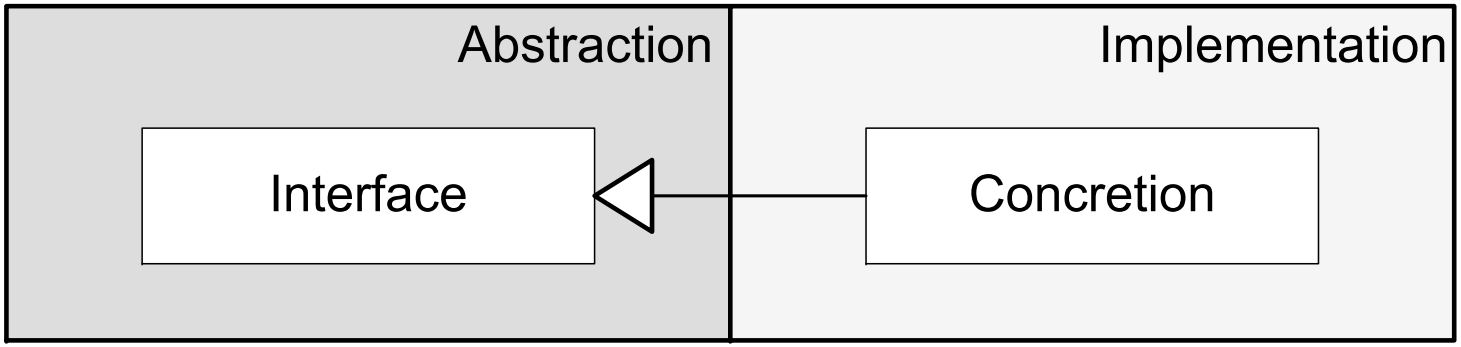
12. Bridge Pattern

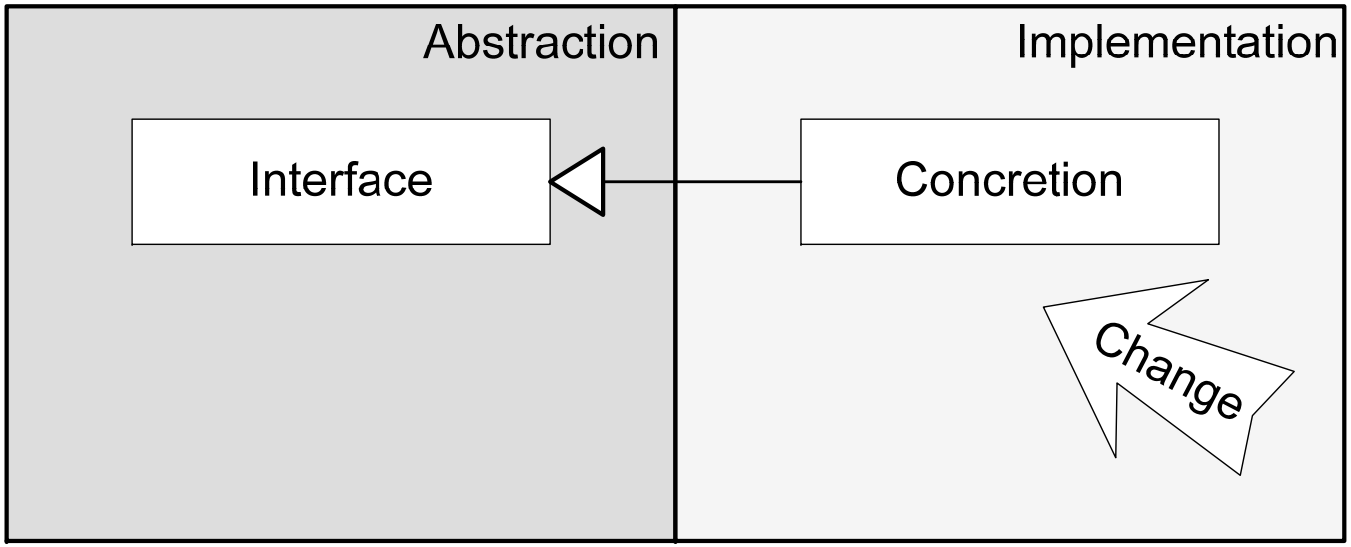
Intent

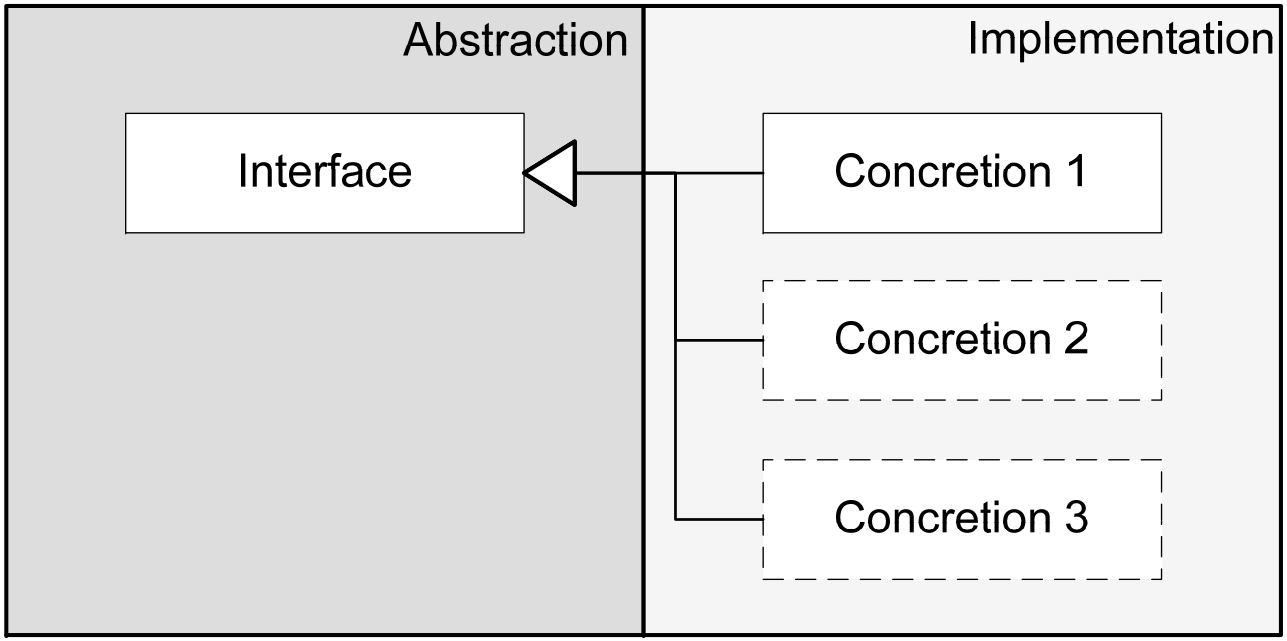
- Decouple an abstraction from its implementation so that the two can vary (change) independently.
- 桥梁模式的用意是将抽象化(Abstraction)与实现化(Implementation) 解耦，使得二者可以独立地变化。
  - 抽象化(Abstraction)不等于接口(Interface)，存在于多个实体中的共同的概念性联系，就是抽象化。接口是一种抽象化的方式。
  - 所谓强耦和，就是在编译时期已经确定的，无法在运行时期动态改变的关联；所谓弱耦和，就是可以动态地确定并且可以在运行时期动态地改变的关联。

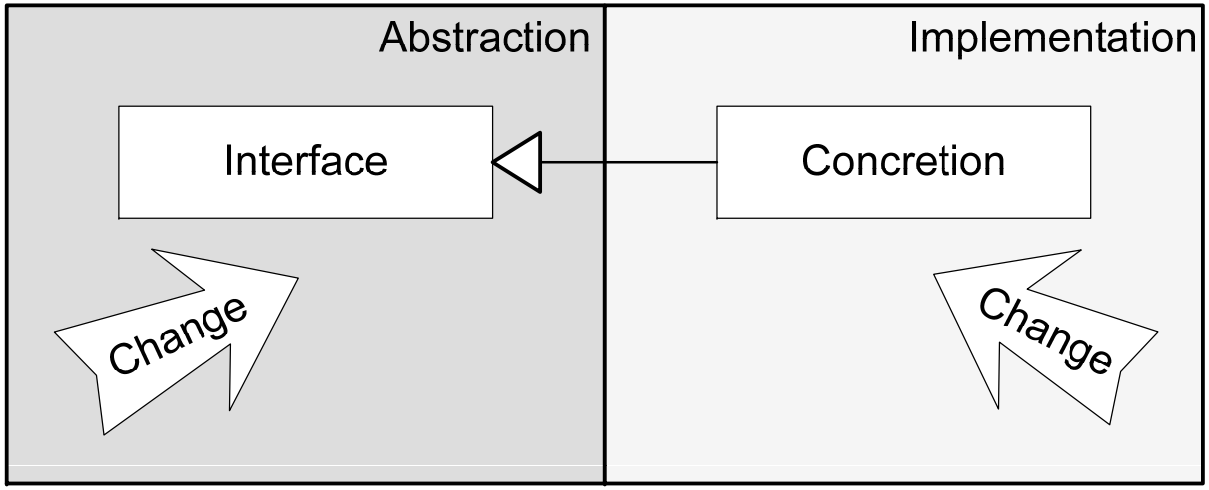
Intent

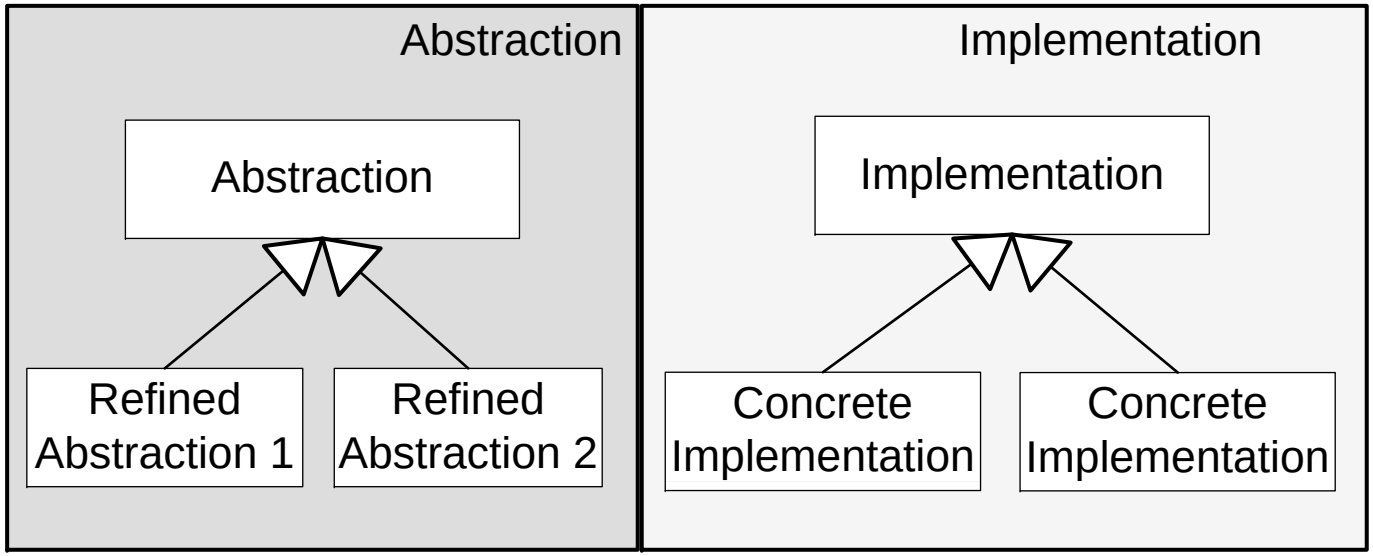
- When an abstraction can have one of several possible implementations, the usual way to accommodate them is to use inheritance.
  - An abstract class defines the interface to the abstraction, and concrete subclasses implement it in different ways.
- But this approach isn't always flexible enough. Inheritance binds an implementation to the abstraction permanently, which makes it difficult to modify, extend, and reuse abstractions and implementations independently.

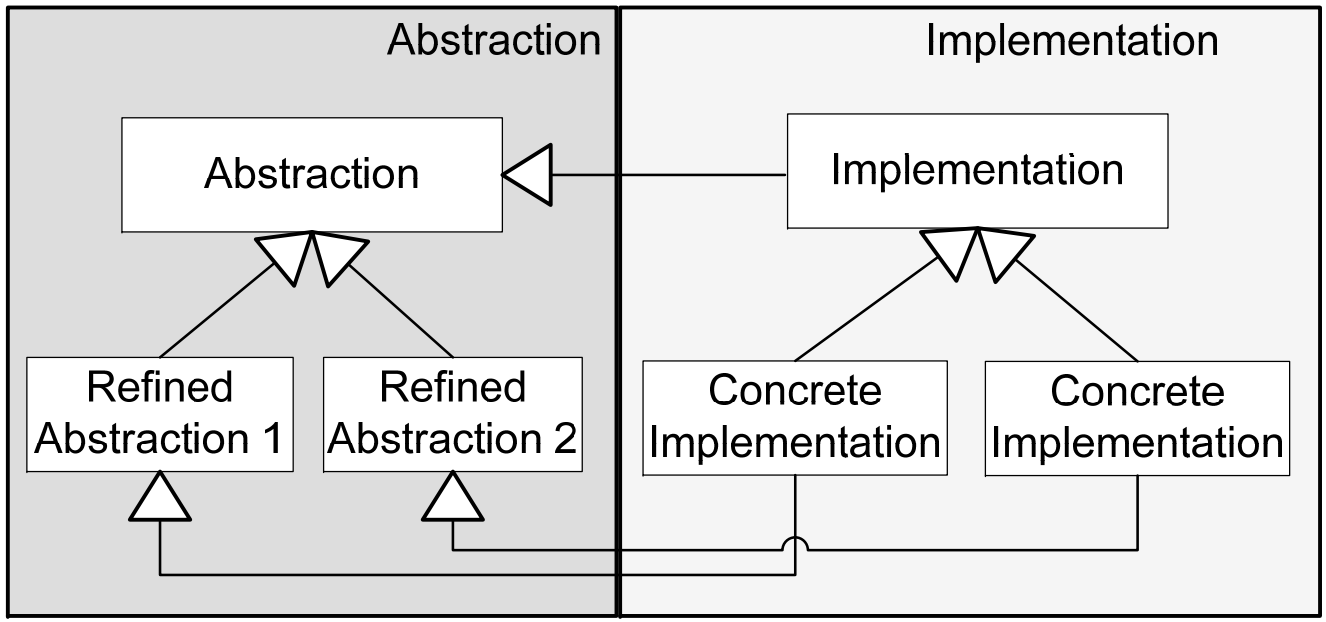

Abstraction
Implementation
Interface
Concretion

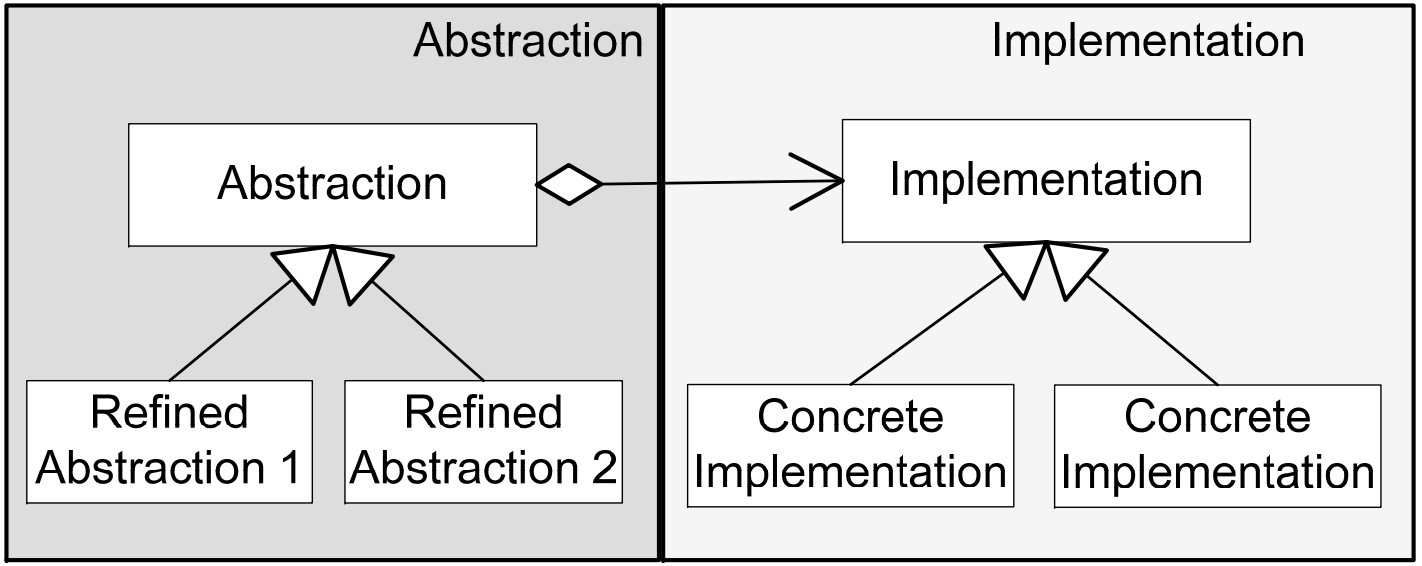

Abstraction
Implementation
Interface
Concretion
Change


Abstraction
Implementation
Interface
Concretion 1
Concretion 2
Concretion 3

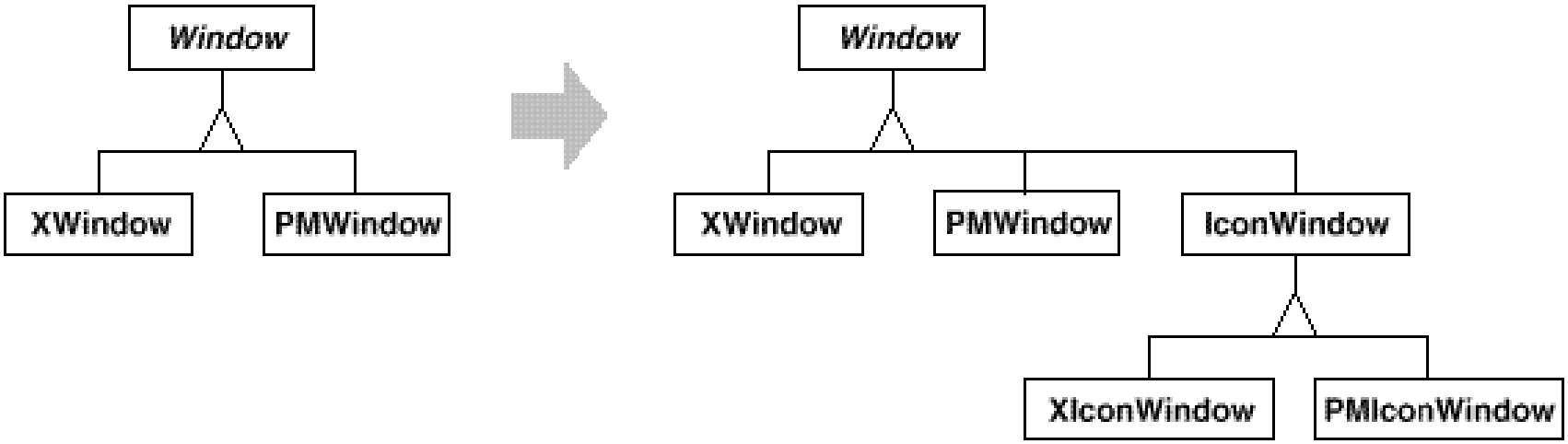

Abstraction
Implementation
Interface
Concretion
Change
Change


Abstraction
Abstraction
Refined Abstraction 1
Refined Abstraction 2
Implementation
Implementation
Concrete Implementation
Concrete Implementation

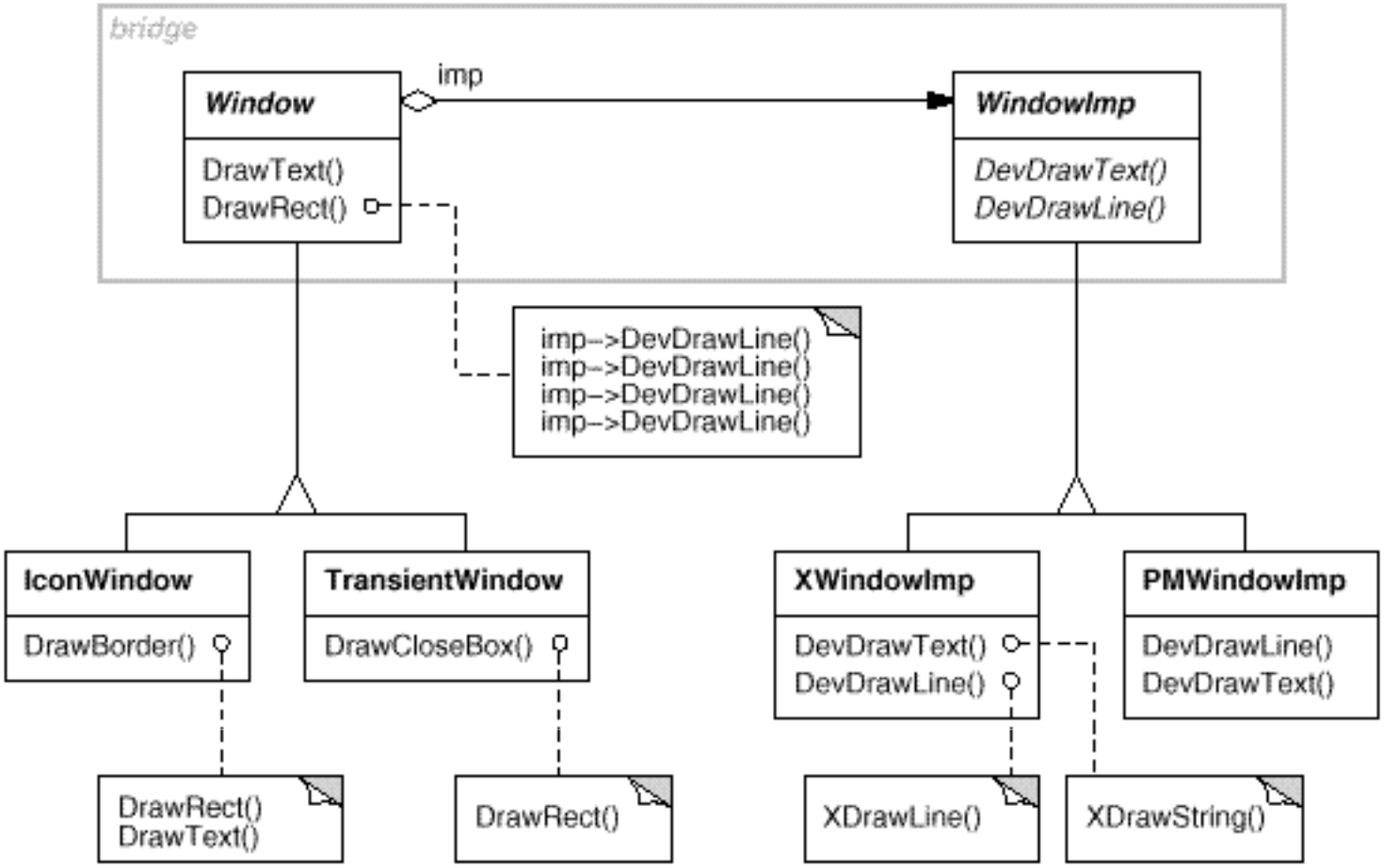

Abstraction
Implementation
Abstraction
Implementation
Refined Abstraction 1
Refined Abstraction 2
Concrete Implementation
Concrete Implementation


Abstraction
Implementation
Abstraction
Implementation
Refined Abstraction 1
Refined Abstraction 2
Concrete Implementation
Concrete Implementation

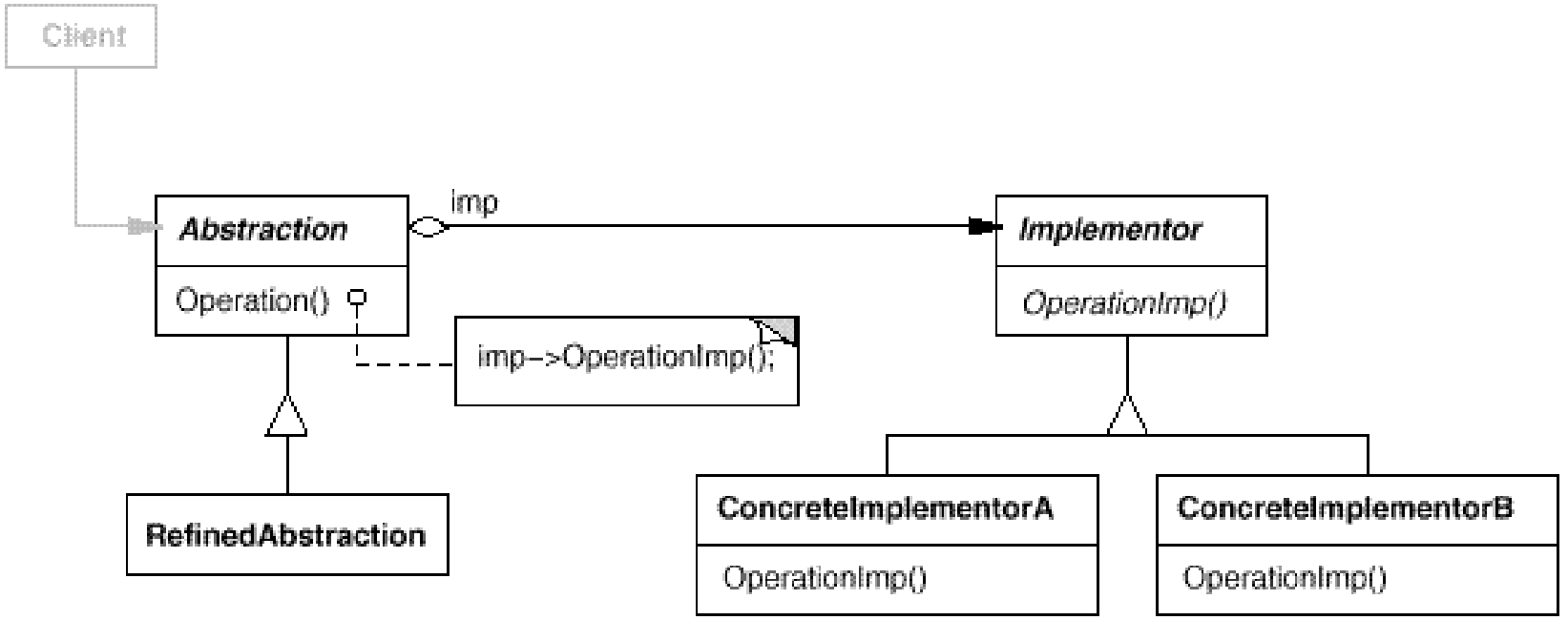
Example


Window
XWindow
PMWindow
Window
XWindow
PMWindow
IconWindow
XIconWindow
PMIconWindow

Example


bridge
Window
DrawText()
DrawRect()
imp
WindowImp
DevDrawText()
DevDrawLine()
imp->DevDrawLine()
imp->DevDrawLine()
imp->DevDrawLine()
imp->DevDrawLine()
IconWindow
DrawBorder()
TransientWindow
DrawCloseBox()
XWindowImp
DevDrawText()
DevDrawLine()
PMWindowImp
DevDrawLine()
DevDrawText()
DrawRect()
DrawText()
DrawRect()
XDrawLine()
XDrawString()

Structure


Client
Abstraction
Operation()
imp
Implementor
OperationImp()
imp->OperationImp();
RefinedAbstraction
ConcreteImplementorA
OperationImp()
ConcreteImplementorB
OperationImp()

Participants

- Abstraction
  - Defines the abstraction's interface.
  - Maintains a reference to an object of type Implementor.
- RefinedAbstraction
  - Extends the interface defined by Abstraction.
- Implementor
  - Defines the interface for implementation classes.
- ConcreteImplementor
  - implements the Implementor interface and defines its concrete implementation.

Implementation's interface and Abstraction's interface

- Abstraction is interface of Abstraction
- Implementor is interface of implementation
  - Implementor doesn't have to correspond exactly to Abstraction;
  - Two interfaces can be quite different;
  - Implementor is defined by Abstraction (DIP);
  - Typically Implementor provides only primitive operations;
  - Typically Abstraction defines higher-level operations based on these primitives.

Consequences

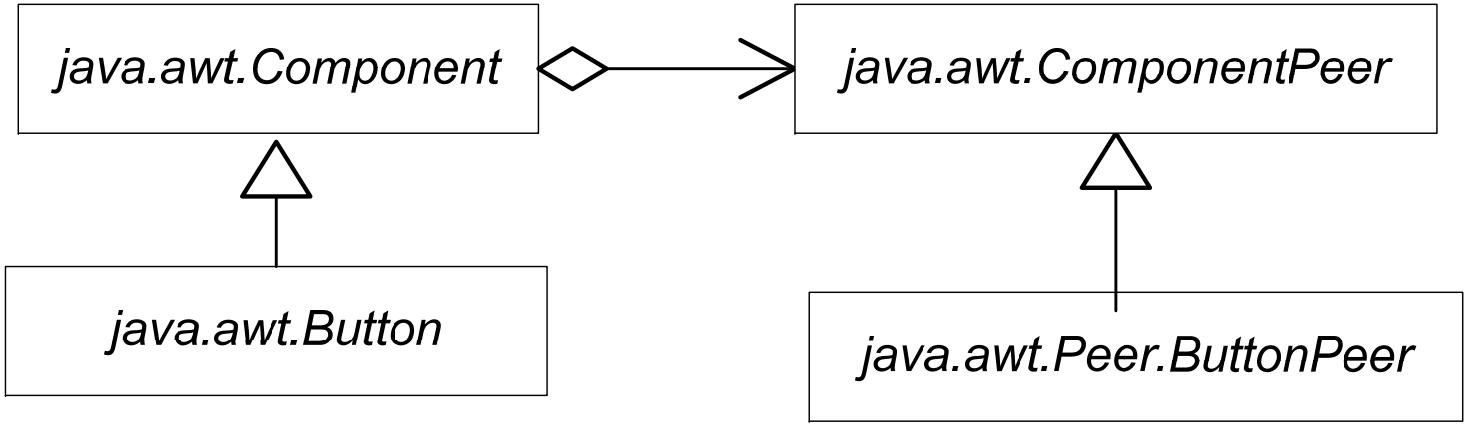
- Decoupling interface and implementation.
  - An implementation is not bound permanently to an interface.
  - The implementation of an abstraction can be configured at run-time. It's possible for an object to change its implementation at run-time.
  - Eliminates compile-time dependencies on the implementation. Changing an implementation class doesn't require recompiling the Abstraction class and its clients.
  - Encourage layering that can lead to a better-structured system. The high-level part of a system only has to know about Abstraction and Implementor.
- Improved extensibility.
- Hiding implementation details from clients.

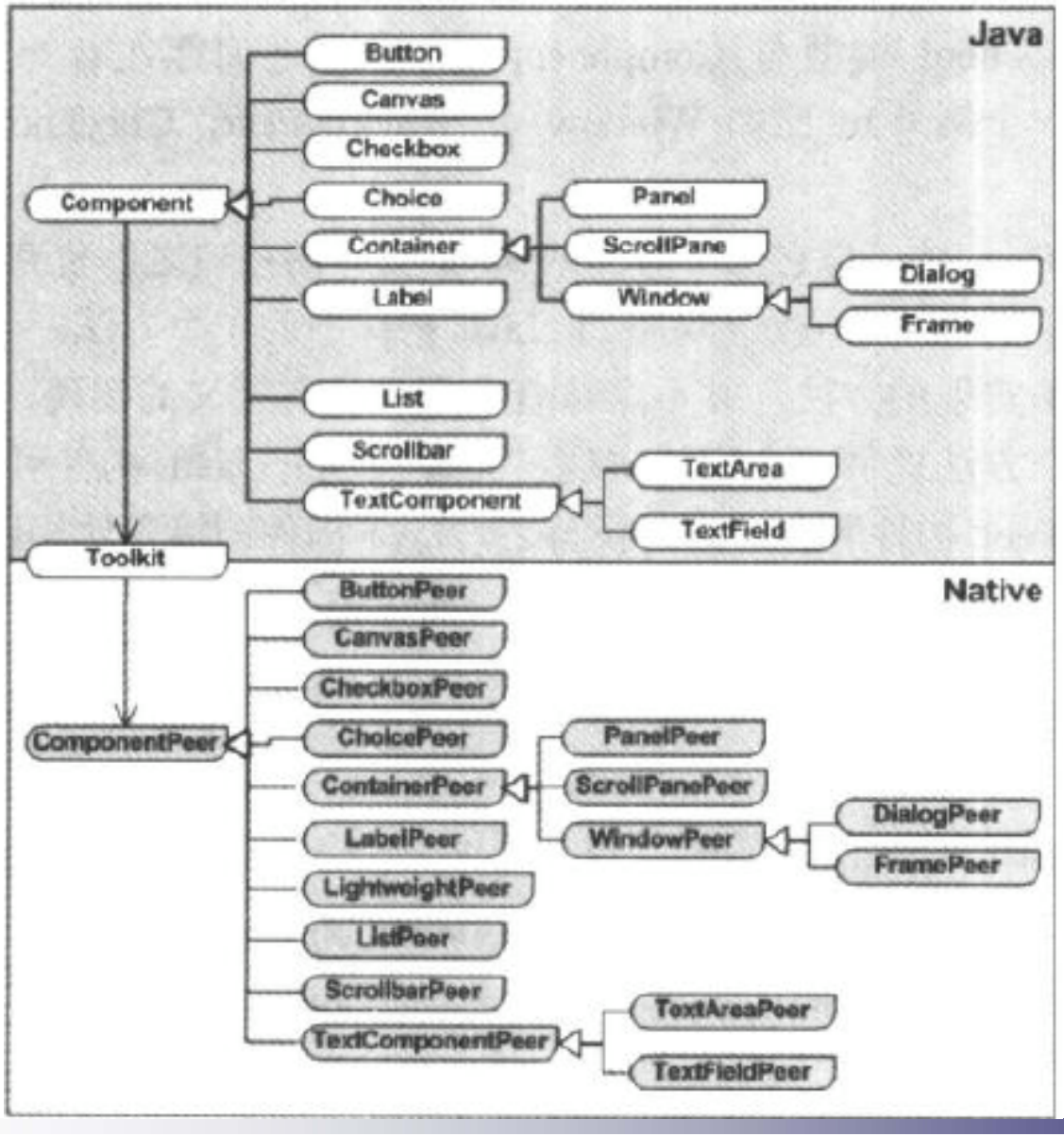
Applicability

- You want to avoid a permanent binding between an abstraction and its implementation.
- Both the abstractions and their implementations should be extensible by subclassing.
- Changes in the implementation of an abstraction should have no impact on clients;

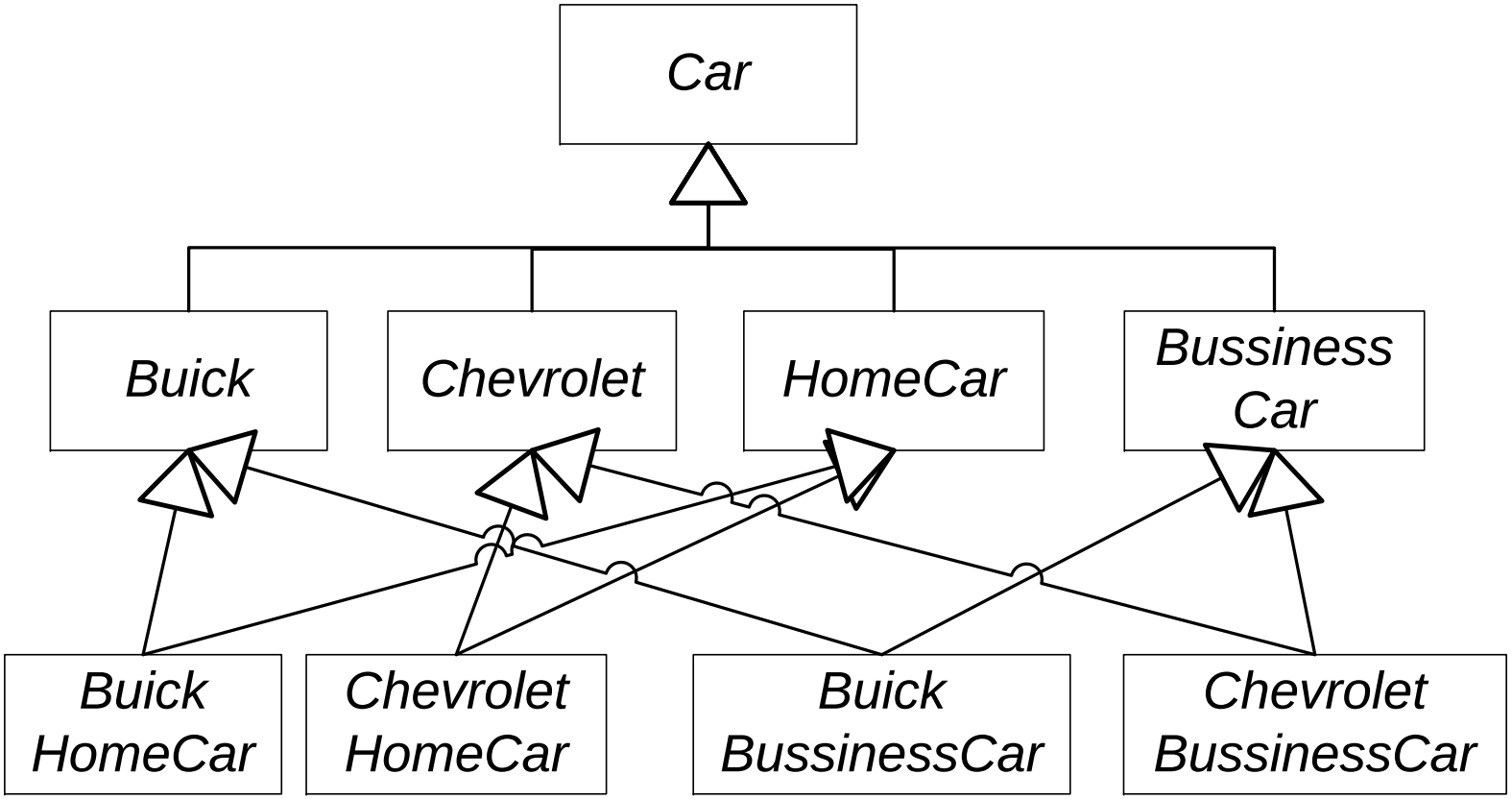
Example: Peer

- The interfaces in the java.awt.peer package define the native GUI capabilities that are required by the heavyweight AWT components of the java.awt package.

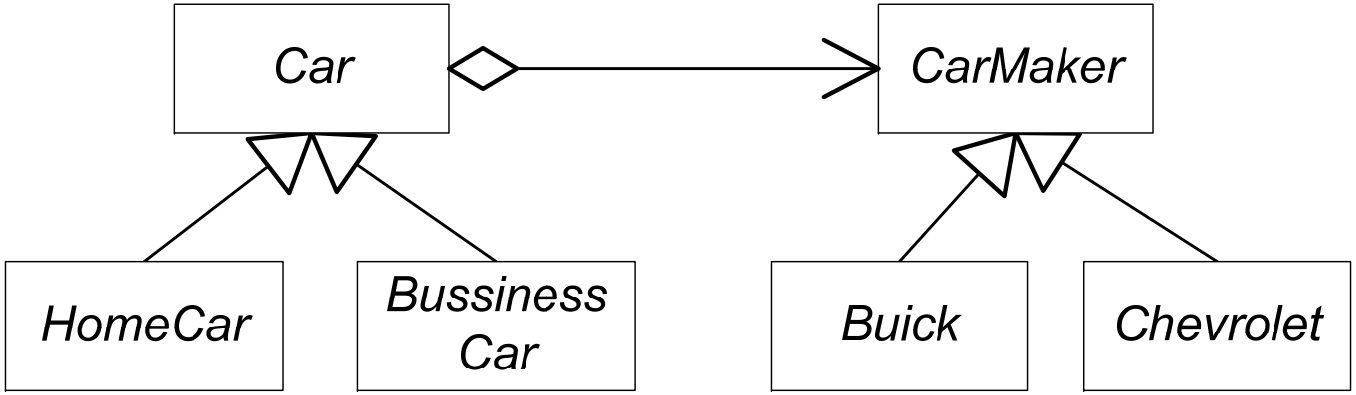



Java
Button
Canvas
Checkbox
Component
Choice
Container
Label
Panel
ScrollPane
Window
Dialog
Frame
List
Scrollbar
TextComponent
TextArea
TextField
Toolkit
Native
ButtonPeer
CanvasPeer
CheckboxPeer
ComponentPeer
ChoicePeer
ContainerPeer
LabelPeer
LightweightPeer
ListPeer
ScrollbarPeer
TextComponentPeer
PanelPeer
ScrollPanePeer
WindowPeer
DialogPeer
FramePeer
TextAreaPeer
TextFieldPeer

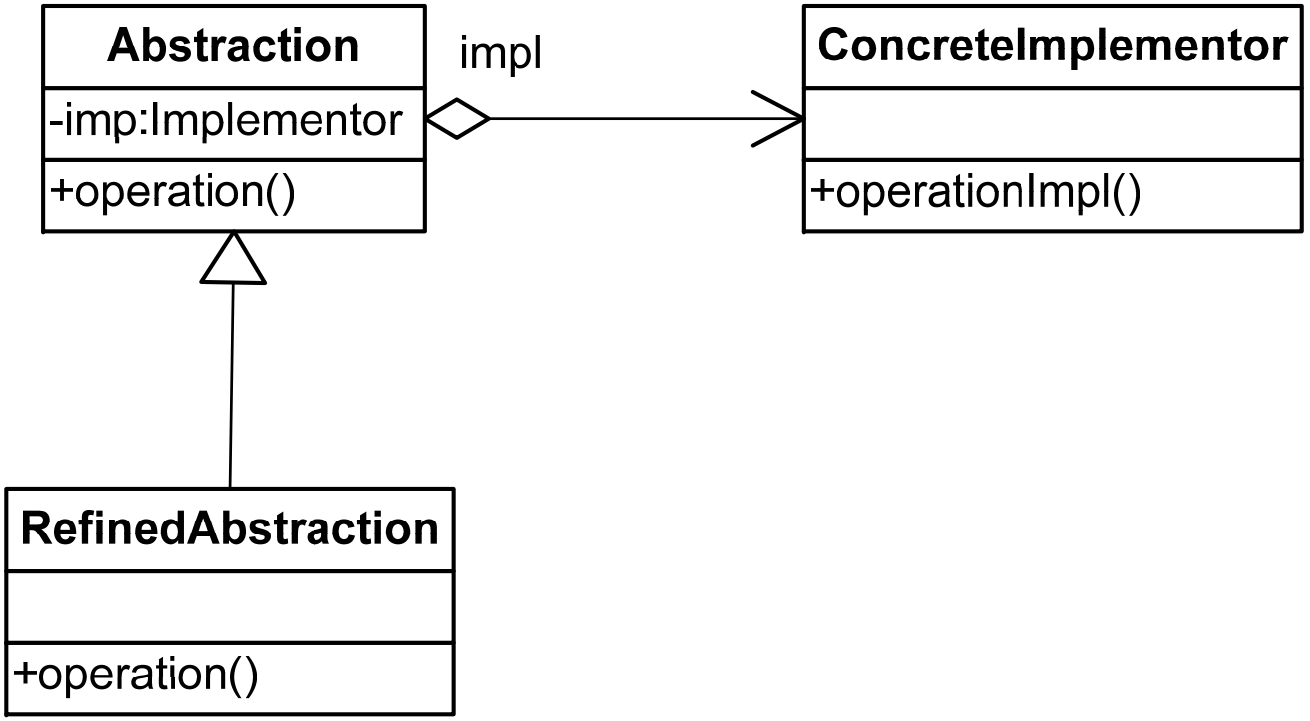
Example: Car Factory



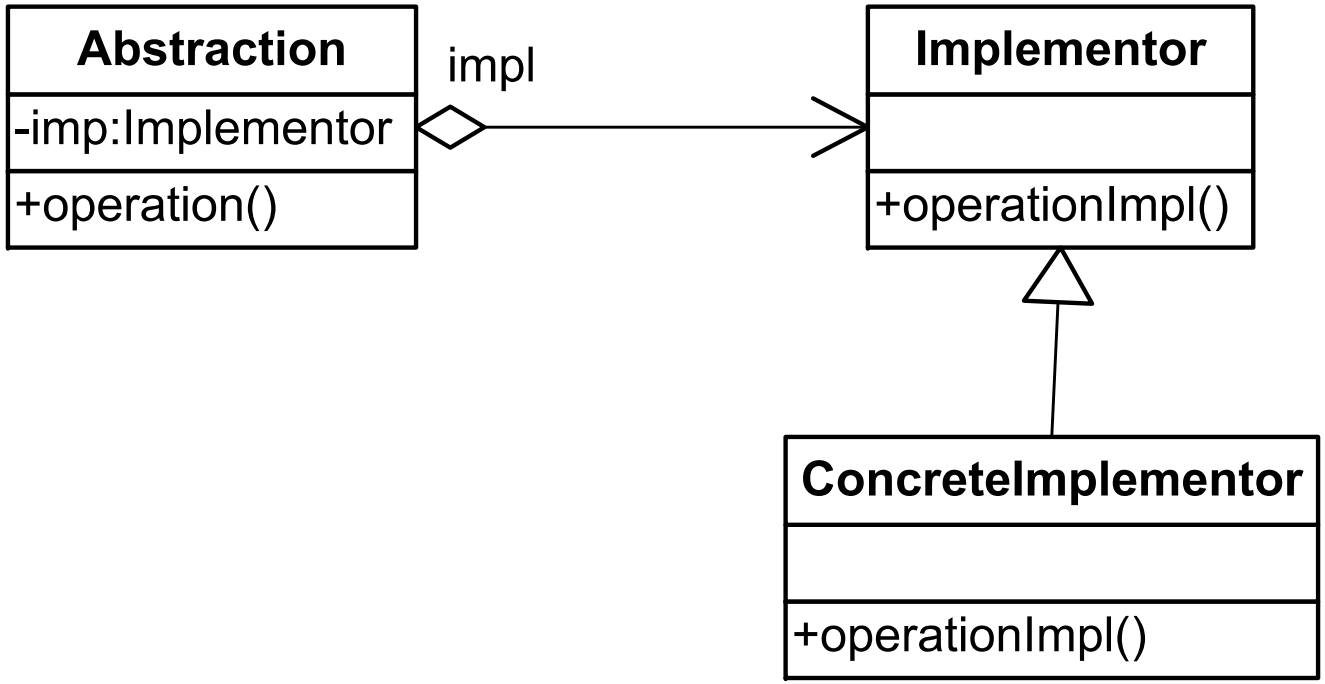
Example: Car Factory



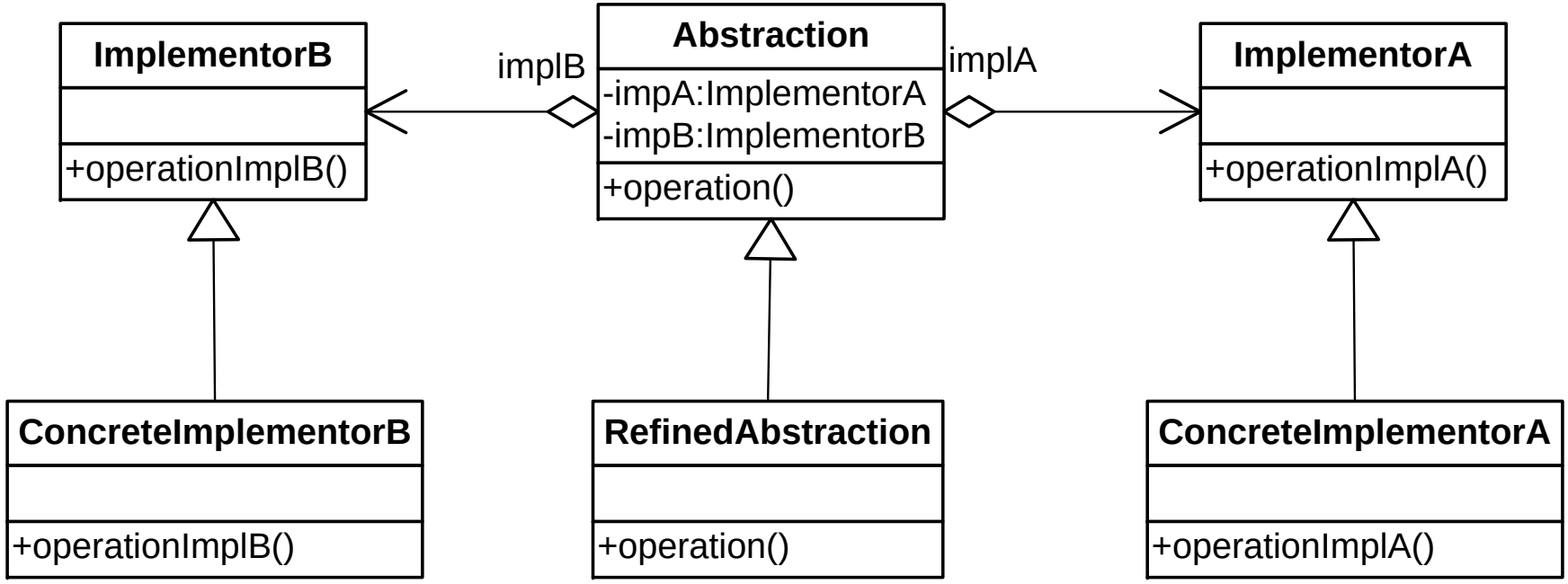
Variation 1: Implementor is omitted

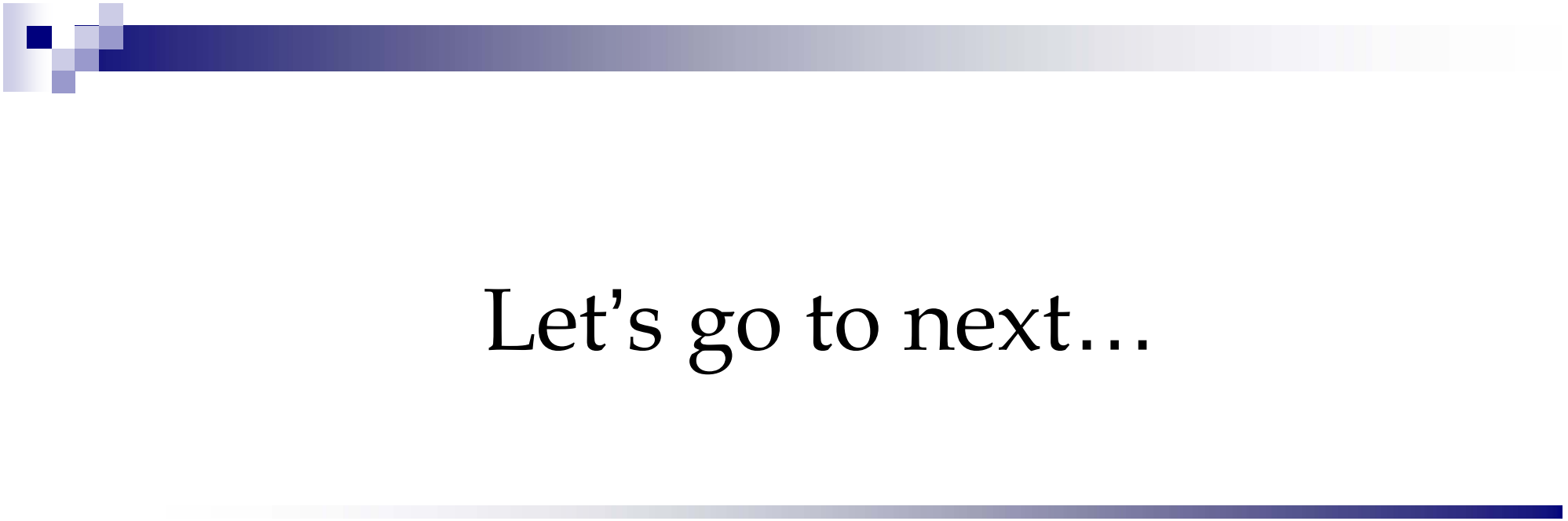


Variation 2: Refined Abstraction is omitted



Variation 3: Sharing Implementors





Let’s go to next…