Design Patterns

Giới thiệu

- Trình bày về “design patterns”[Gam95], một cách tiếp cận dựa mẫu (pattern-based approach) có tác dụng hỗ trợ cho pha thiết kế phần mềm. (Nguyên do: Chỉ dùng các PP phân tích thiết kế vẫn chưa đủ ...)
- Liên hệ đến các tiếp cận tương tự: “analysis patterns”, “design heuristics”, “process patterns”, ...

Nội dung

1. Dẫn nhập
2. Tổng quan về mẫu thiết kế GoF
3. Kế thừa và Đa hình: cơ sở cho các mẫu GoF
4. Vài mẫu GoF tiêu biểu và ứng dụng
5. Các tiếp cận tương tự

Dẫn nhập

- Xu hướng sử dụng mẫu trong công việc chuyên môn, trong học tập:
  - Các mẫu, khuôn được dùng trong các ngành công nghiệp khác nhau (in ấn, đúc, ...)
  - Các bài tập mẫu cho học sinh, sinh viên
  - Các mẫu chương trình
  - Các mẫu hướng dẫn thiết kế giao diện với người dùng
  - ...

Dẫn nhập

- Nguồn gốc của mẫu thiết kế phần mềm
  - Ngôn ngư? mẫu của kiến trúc sư C. Alexander [Alex77]: thiết kế nhà bằng cách lắp ráp các khuôn mẫu có sẵn

    ➔ Ý tưởng cho việc sưu tầm và sử dụng "các mẫu phần mềm"
  - Các "idiom" trong lập tr?nh C++ [Cop92]

Dẫn nhập

- Thuật ngữ
  - Các từ: sample, pattern, template,...
  - “Design patterns”: Patterns in Object-oriented software design
  - “a design pattern”: an elegant solution to a specific problem in OO software design
  - Thuật ngữ tiếng Việt...

Định nghĩa

- Định nghĩa một mẫu (pattern) nói chung:

  Một mẫu là một cặp **(vấn đề, lời giải)** có thể áp dụng trong nhiều tình huống, ngữ cảnh khác nhau.

- Mỗi mẫu thường bao gồm các bộ phận sau:
  - Tên
  - Nội dung vấn đề
  - Lời giải (phải đủ tổng quát để có thể dùng trong nhiều tình huống)
  - Các hệ quả mang lại và ví dụ áp dụng

Tổng quan về mẫu thiết kế GoF

- Nguồn gốc lịch sử vàTác giả
  - Gồm 23 mẫu thiết kế của 4 tác giả: Erich Gamma, Richard Helm, Ralph Johnson, và John Vlissides;
  - Các mẫu nầy còn được gọi là mẫu GoF (Gang of Four)
- “Finding patterns is much easier than describing them”

Tổng quan về mẫu thiết kế GoF

- Nguồn gốc lịch sử vàTác giả
  - Gồm 23 mẫu thiết kế của 4 tác giả: Erich Gamma, Richard Helm, Ralph Johnson, và John Vlissides;
  - Các mẫu nầy còn được gọi là mẫu GoF (Gang of Four)
- “Finding patterns is much easier than describing them”

Tổng quan về mẫu thiết kế GoF

- Nguồn gốc lịch sử vàTác giả
  - Khoảng ½ trong của bộ mẫu nầy có nguồn gốc từ luận án tiến sĩ của Erich Gamma. Các tác giả gặp nhau tại 2 hội ngh? OOPSLA'91 và OOPSLA'92 (Object-Oriented Programming Systems, Languages, and Applications Conference);
  - Sau đó cùng làm việc để soạn lại một bộ gồm 23 mẫu và trình bày tại hội nghị ECOOP'93 (European Conference on Object-Oriented Programming).

Cấu trúc chung để mô tả mẫu GoF

- **Tên và Phân loại**
- **Mục đích, ý định**: mô tả ngắn gọn về ma?u
- **Bí danh**
- **Motivation**: trình bày một tình huống cụ thể trong thiết kế phần mềm dẫn đến việc sử dụng mẫu này để giải quyết vấn đề.
- **Khả năng ứng dụng**: gợi ý các tình huống trong thiết kế mà có thể ứng dụng mẫu nầy

Cấu trúc chung để mô tả mẫu GoF

- **Cấu trúc**: mô tả mẫu bằng các ký hiệu đồ hình thường dùng trong các phương pháp p.tích/t.kế (ký hiệu OMT, UML, ...)
- **Các thành viên**: trình bày ý nghĩa của các lớp/đối tượng tham gia vào mẫu thiết kế và trách nhiệm của chúng
- **Sự cộng tác**: các thành viên (lớp/đối tượng) của mẫu cộng tác như thế nào để thực hiện trách nhiệm của chúng
- **Các hệ quả mang lại**

Cấu trúc chung để mô tả mẫu GoF

- Chú ý liên quan đến việc cài đặt
- Mã nguồn minh họa
- Nêu ra những ví dụ về các hệ thống thực tế (đã được phát triển và đang chạy) mà có sử dụng mẫu nầy
- Các mẫu liên quan: những mẫu nào có liên hệ với mẫu nầy, những điểm quan trọng cần phải phân biệt; mẫu nầy có thể dùng phối hợp với những mẫu nào.

Danh sách 23 mẫu GoF

CAÙC MAÃU VEÀ TAÏO LAÄP LÔÙP hay ÑOÁI TÖÔÏNG	
Teân	**Muïc ñích**
Abstract Factory (Fabrique Abstraite)	Cung caáp moät interface cho vieäc taïo laäp caùc ñoái töôïng (coù lieân heävôùi nhau) maøkhoâng caàn qui ñònh lôùp khi taïo moãi ñoái töôïng
Builder (Monter)	Taùch rôøi vieäc xaây döïng (construction) moät ñoái töôïng phöùc taïp khoûi bieåu dieãn cuûa noùsao cho cuøng moät tieán trình xaây döïng coùtheåtaïo ñöôïc caùc bieåu dieãn khaùc nhau
Factory Method (Fabrication)	Ñònh nghóa moät interface ñeåtaïo laäp ñoái töôïng nhöng uûy nhieäm vieäc instanciation cho caùc lôùp con (caùc lôùp keá thöøa)
Prototype	Qui ñònh loaïi cuûa caùc ñoái töôïng caàn taïo baèng caùch duøng moät ñoái töôïng maãu, taïo môùi nhôøvaøo sao cheùp ñoái töôïng maãu naày.
Singleton	Caùch ñaët lôùp maøchæcoùmoät theåhieän (ñoái töôïng) duy nhaát

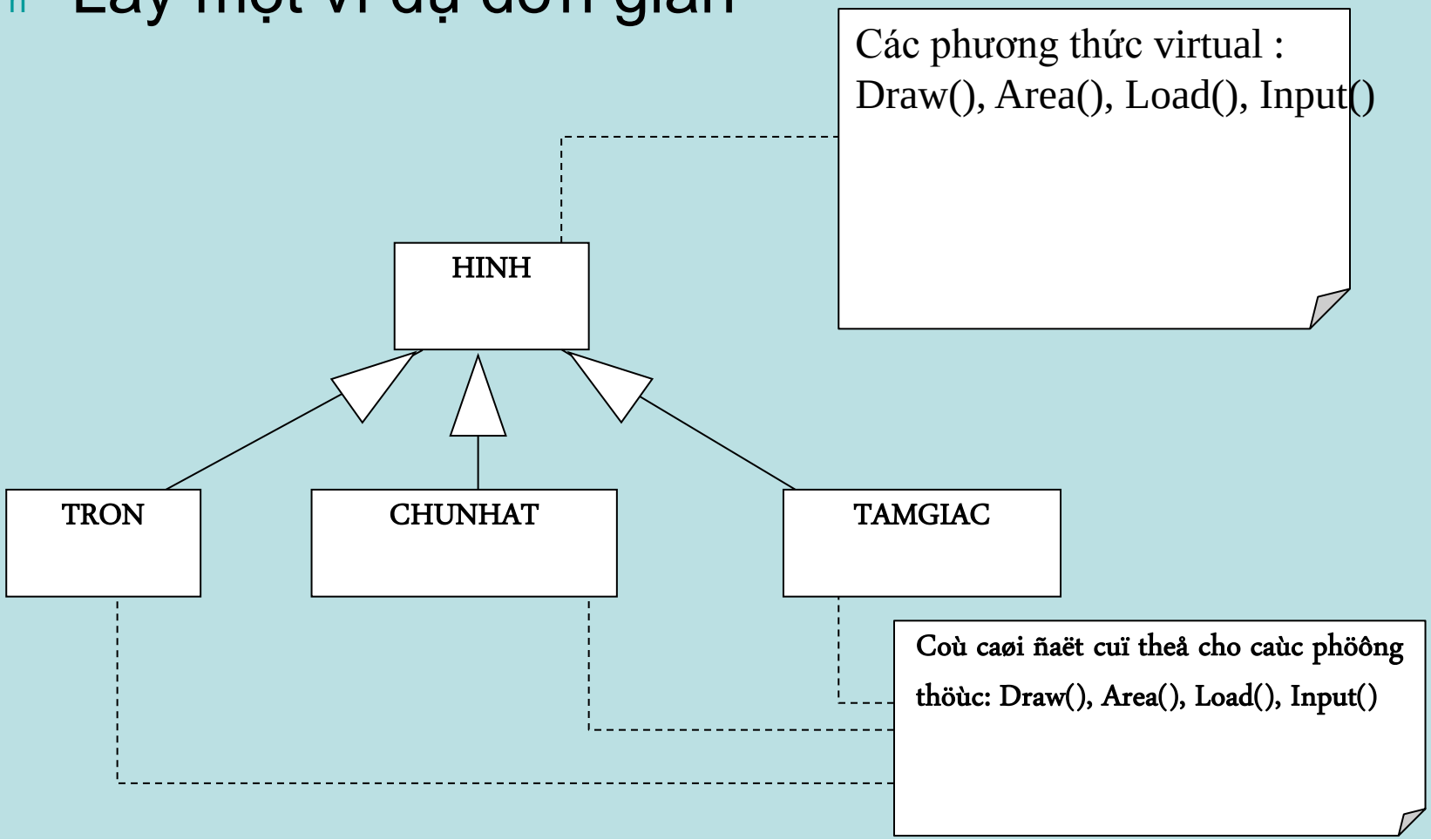
CAÙC MAÃU VEÀ CAÁU TRUÙC LÔÏP hay ÑOÁI TÖÔÏNG	
Teân	**Muïc ñích**
Adapter (adapteur)	Do vaán ñeàtöông thích, thay ñoåinterface cuûa moätlôùp thaønh moät interface khaùc phuøhôïp vôùi yeâu caàu ngöôøisöûduïng lôùp.
Bridge (Pont)	Taùch rôøingöõnghóa cuûa moätvaán ñeàkhoûi vieäc caøiñaët ; muïc ñích ñeåcaûhai boäphaän (ngöõnghóa vaøcaøiñaët) coùtheåthay ñoåiñoäc laäp nhau.
Composite	Toåchöùc caùc ñoáitöôïng theo caáu truùc phaân caáp daïng caây; Taát caû caùc ñoáitöôïng trong caáu truùc ñöôïc thao taùc theo moätcaùch thuaàn nhaátnhö nhau.
Decorator (Dekorateur)	Gaùn theâm traùch nhieäm cho ñoáitöôïng (môûroäng chöùc naêng) vaøo luùc chaïy (dynamically).
Facade (Façade)	Cung caáp moätinterface thuaàn nhaátcho moättaäp hôïp caùc interface trong moät“heäthoáng con” (subsystem).
Flyweight (Poids mouche)	Söûduïng vieäc chia seûñeåthao taùc hieäu quaûtreân moätsoálöôïng lôùn ñoáitöôïng “côûnhoû (chaúng haïn paragraph, doøng, coät, kyù töï..)
Proxy (Procuration)	Ñieàu khieån moätcaùch giaùn tieáp vieäc truy xuaátmoätñoáitöôïng thoâng qua moätñoáitöôïng ñöôïc uûy nhieäm.

CAÙC MAÃU VEÀ ÖÙNG XÖÛ CUÛA LÔÙP hay ÑOÁI TÖÔÏNG (1)	
Teân	**Muïc ñích**
Chain of Responsibility (Chaìne de responsabilitès)	Khaéc phuïc vieäc ghep caëp giöõa boägôûi vaøboänhaän thoâng ñieäp; Caùc ñoái töôïng nhaän thoâng ñieäp ñöôïc keát noái thaønh moät chuoãi vaøthoâng ñieäp ñöôïc chuyeån doïc theo chuoãi naày ñeán khi gaëp ñöôïc ñoái töôïng xöûlyùnoù
Command (Commande)	Moãi yeâu caàu (thöïc hieän moät thao taùc naøo ñoù)ñöôïc bao boïc thaønh moät ñoái töôïng. Caùc yeâu caàu seõñöôïc löu tröõvaøgôûi ñi nhö caùc ñoái töôïng.
Interpreter (Interpreteur)	Hoãtrôi vieäc ñònh nghóa bieåu dieãn vaê phaïm vaøboäthoâng dòch cho moät ngoân ngöõ
Iterator (Itèrateur)	Truy xuaát caùc phaàn töûcuûa ñoái töôïng daïng taäp hôïp tuaàn töï (list, array, ...) maøkhoâng phuï thuoäc vaøo bieåu dieãn beân trong cuûa caùc phaàn töû
Mediator (Mèdiateur)	Ñònh nghóa moät ñoái töôïng ñeåbao boïc vieäc giao tieáp giöõa moät soáñoái töôïng vôùi nhau.
Memento	Hieän chaønh vaøtraûlaïi nhö cuõtraïng thaùi beân trong cuûa ñoái töôïng maøvaãn khoâng vi phaïm vieäc bao boïc döõlieäu.

CAÙC MAÃU VEÀ ÖÙNG XÖÛ CUÛA LÔÙP hay ÑOÁI TÖÔÏNG (2)	
Teân	**Muïc ñích**
Observer (Observateur)	Ñònh nghóa söï phuï thuoäc *moät-nhieàu* giöõa caùc ñoái töôïng sao cho khi moät ñoái töôïng thay ñoåi traïng thaùi thì taát caû caùc ñoái töôïng phuï thuoäc noù cuõng thay ñoåi theo.
State (Etat)	Cho pheùp thay ñoåi öùng xöû cuûa ñoái töôïng tuøy theo söï thay ñoåi traïng thaùi beân trong cuûa noù
Strategy (Strategie)	Bao boïc moät hoï caùc thuaät toaùn baèng caùc lôùp ñoái töôïng ñeå thuaät toaùn coù theå thay ñoåi ñoäc laäp ñoái vôùi chöông trình söû duïng thuaät toaùn.
Template method (Patron de methode)	Ñònh nghóa phaàn khung cuûa moät thuaät toaùn, töùc laø moät thuaät toaùn toång quaùt goïi ñeán moät soá phöông thöùc chöa ñöôïc caøi ñaët trong lôùp cô sôû, vieäc caøi ñaët caùc phöông thöùc ñöôïc uûy nhieäm cho caùc lôùp keá thöøa.
Visitor (Visiteur)	Cho pheùp ñònh nghóa theâm pheùp toaùn môùi taùc ñoäng leân caùc phaàn töû cuûa moät caáu truùc ñoái töôïng maø khoâng caàn thay ñoåi caùc lôùp ñònh nghóa caáu truùc ñoù

Kế thừa, Đa hình: cơ sở cho mẫu GoF

- Lấy một ví dụ đơn giản



Các phương thức thích hợp sẽ được gọi tùy theo kiểu của đối tượng:

```
HINH  *p;
p = new CHUNHAT;
p->Input(); // phương thức Input của CHUNHAT sẽ được gọi.
```

Các thao tác tổng quát trên kiểu HINH không nên có các chỉ thị phụ thuộc các lớp kế thừa.

```
void ThaoTac(HINH  *p) {
        // chỉ sử dụng phương thức chung.
        If(p->Load())      {
                p->Input();
                p->Draw();
                ...
        }
}
```

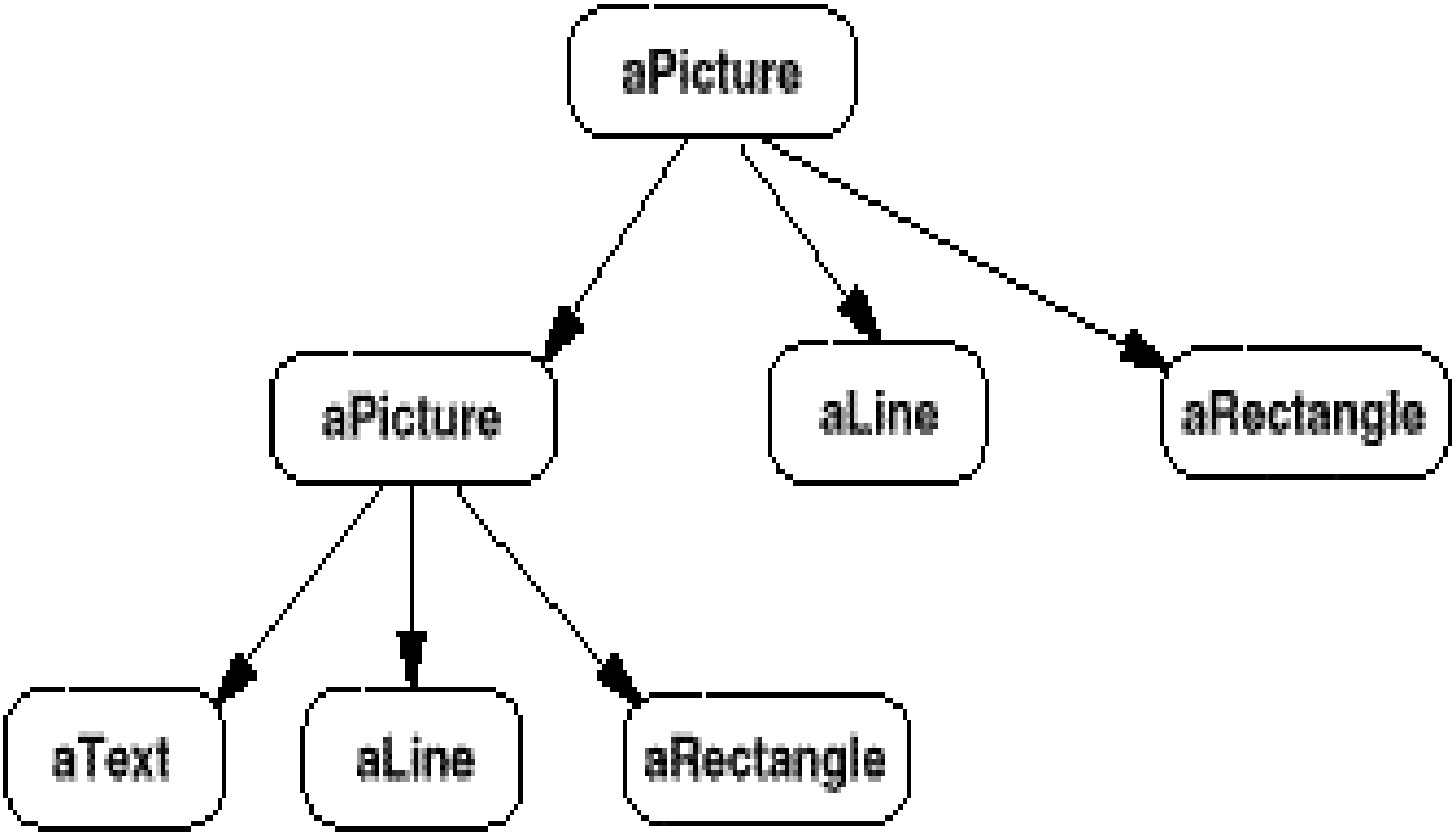
==> KHÔNG NÊN ép kiểu p, chẳng hạn thành TRON, rồi gọi các phương thức riêng của lớp TRON.

Vài mẫu GoF tiêu biểu và ứng dụng

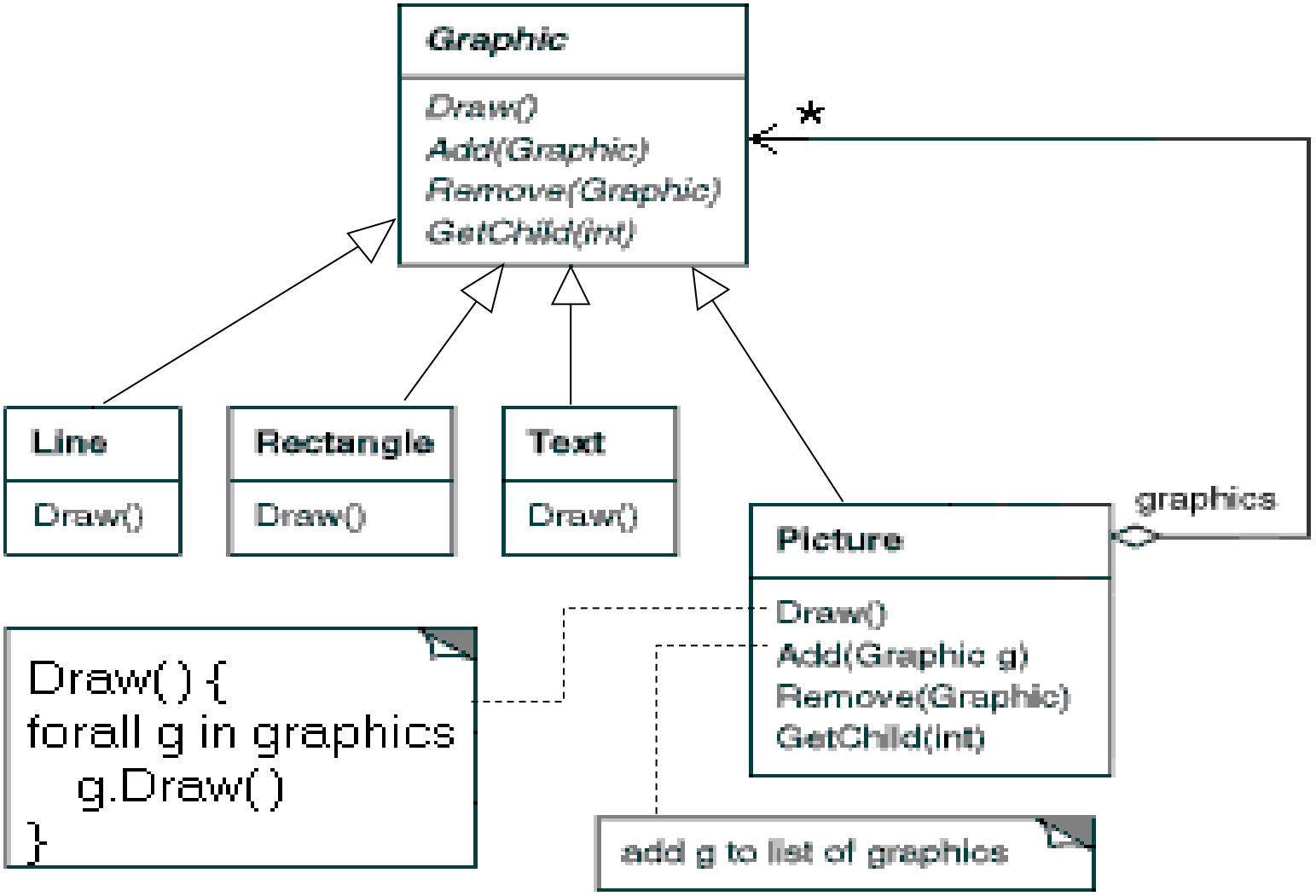
- Phần nầy sẽ trình bày các mẫu tiêu biểu sau đây:
  - Composite
  - Bridge
  - Template method
  - Observer
- Để ngắn gọn, một vài mục trong 13 mục của cấu trúc mô tả mỗi mẫu sẽ được bỏ qua. Chúng ta sẽ quan tâm nhiều đến ý nghĩa, tình huống cần dùng và ví dụ ứng dụng.

Maãu "Composite"

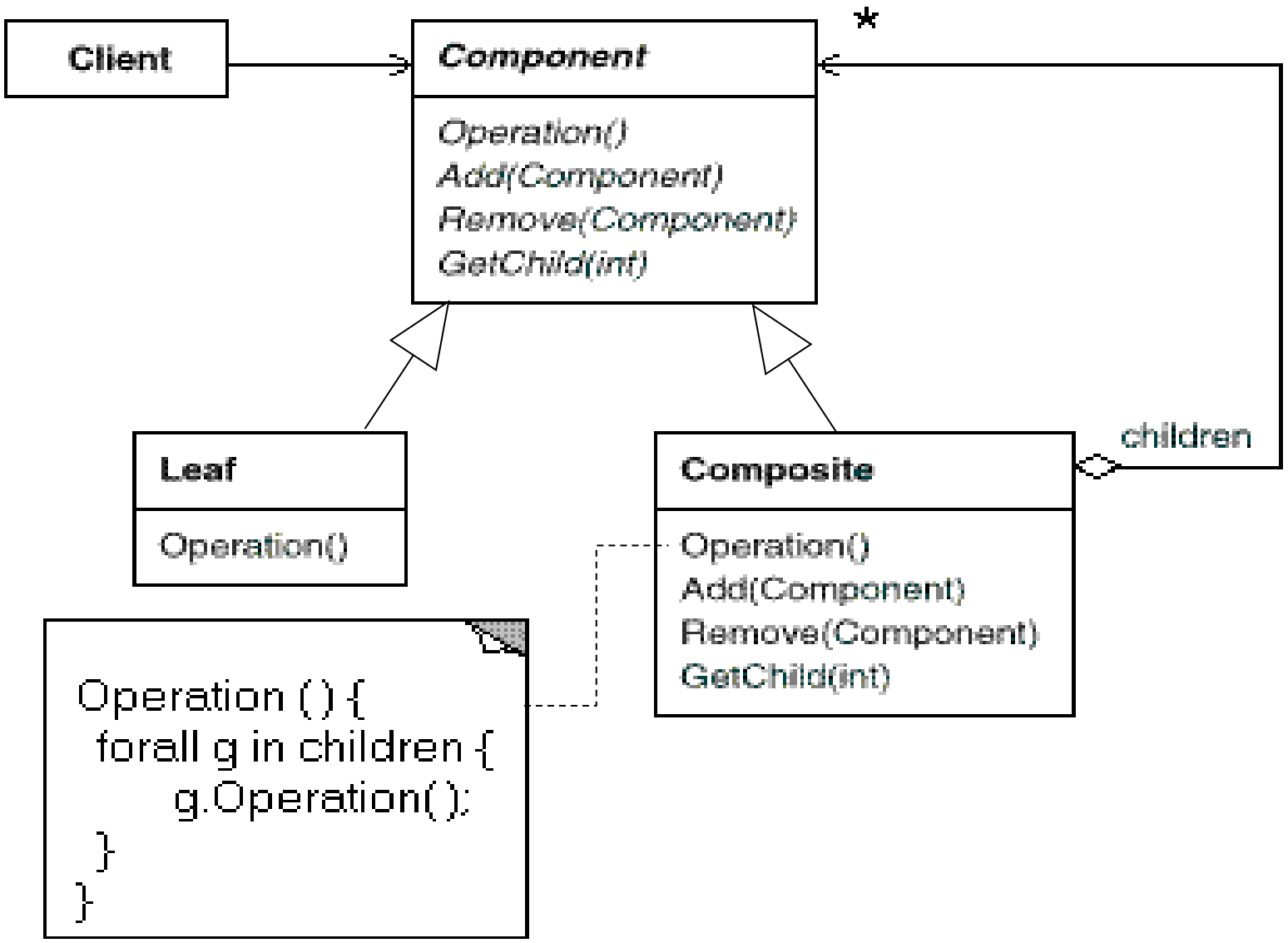
- **Tên**: "Composite", tạm dịch "Đối tượng đa hợp", thuộc lớp mẫu cấu trúc đối tượng.
- **Ý định**:Tổ chức các đối tượng theo cấu trúc phân cấp dạng cây; Tất cả các đối tượng trong cấu trúc được thao tác theo một cách thuần nhất như nhau
- **Motivation**: Các ứng dụng đồ họa thường lưu trữ các đối tượng phức hợp bao gồm nhiều đối tượng đơn giản hơn. Ví dụ như trong hình vẽ sau:


aPicture
aPicture
aLine
aRectangle
aText
aLine
aRectangle

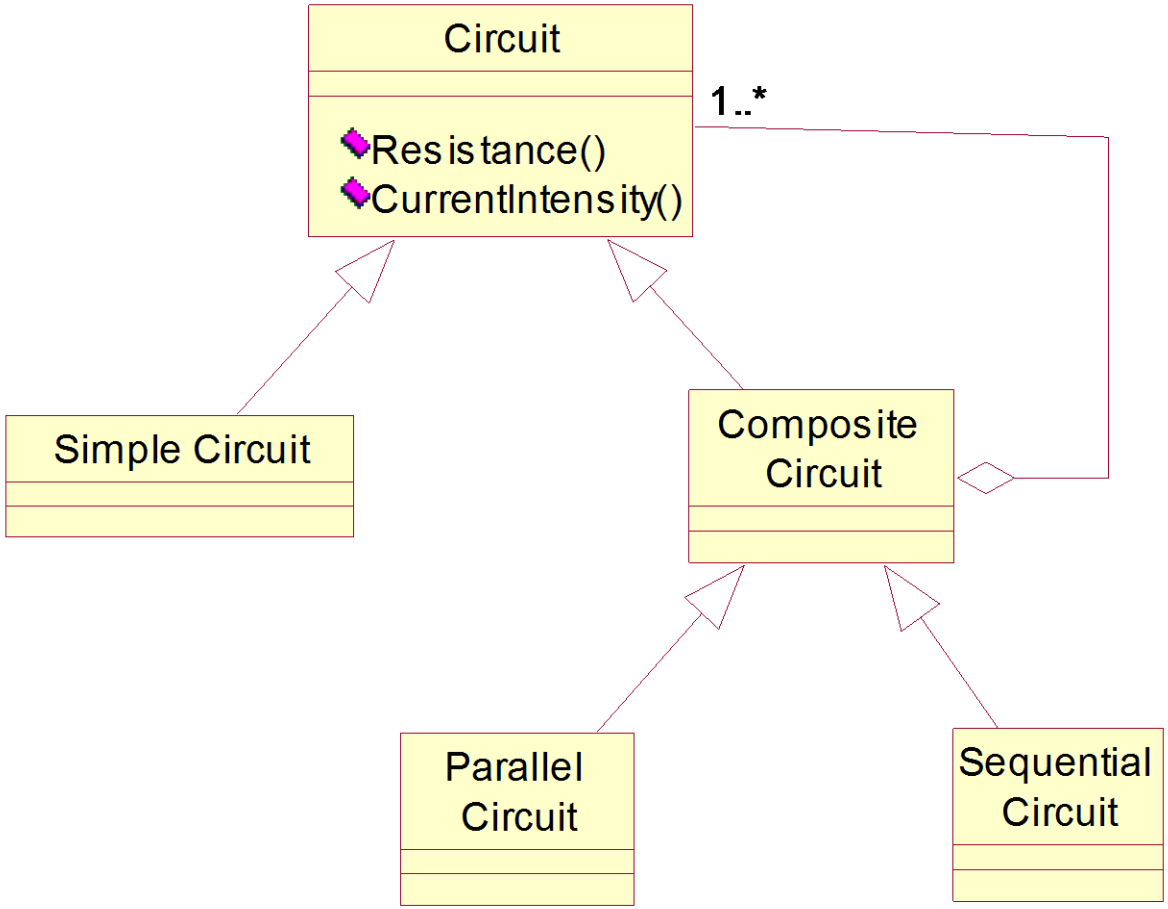
- Mẫu composite giải quyết vấn đề nầy bằng cách tổ chức tích hợp dữ liệu đệ qui như sau:



- **Cấu trúc**:

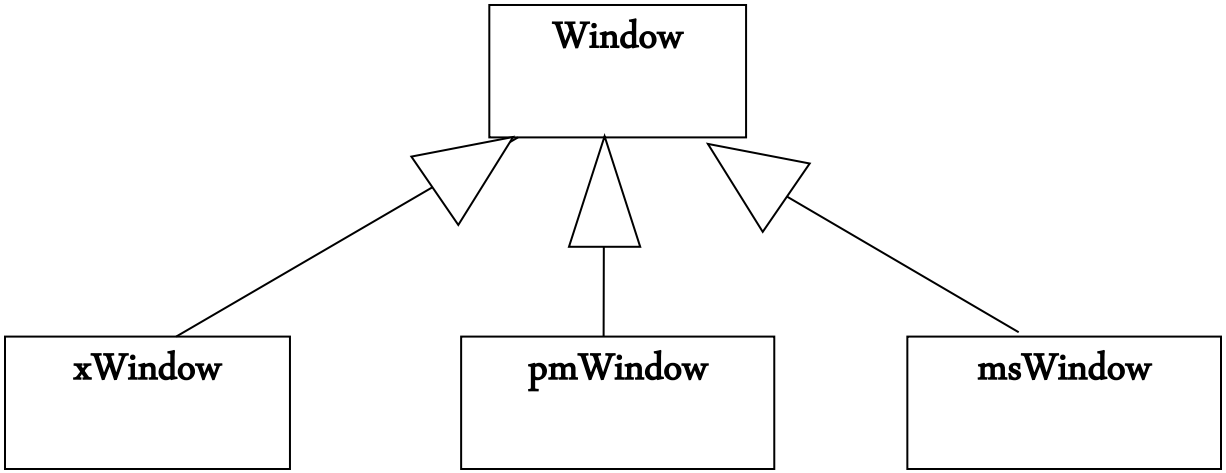

Client
Component
Operation()
Add(Component)
Remove(Component)
GetChild(int)
*
Leaf
Operation()
Composite
Operation()
Add(Component)
Remove(Component)
GetChild(int)
children
Operation ( ) {
forall g in children {
g.Operation( );
}
}

Lưu trữ mạch điện

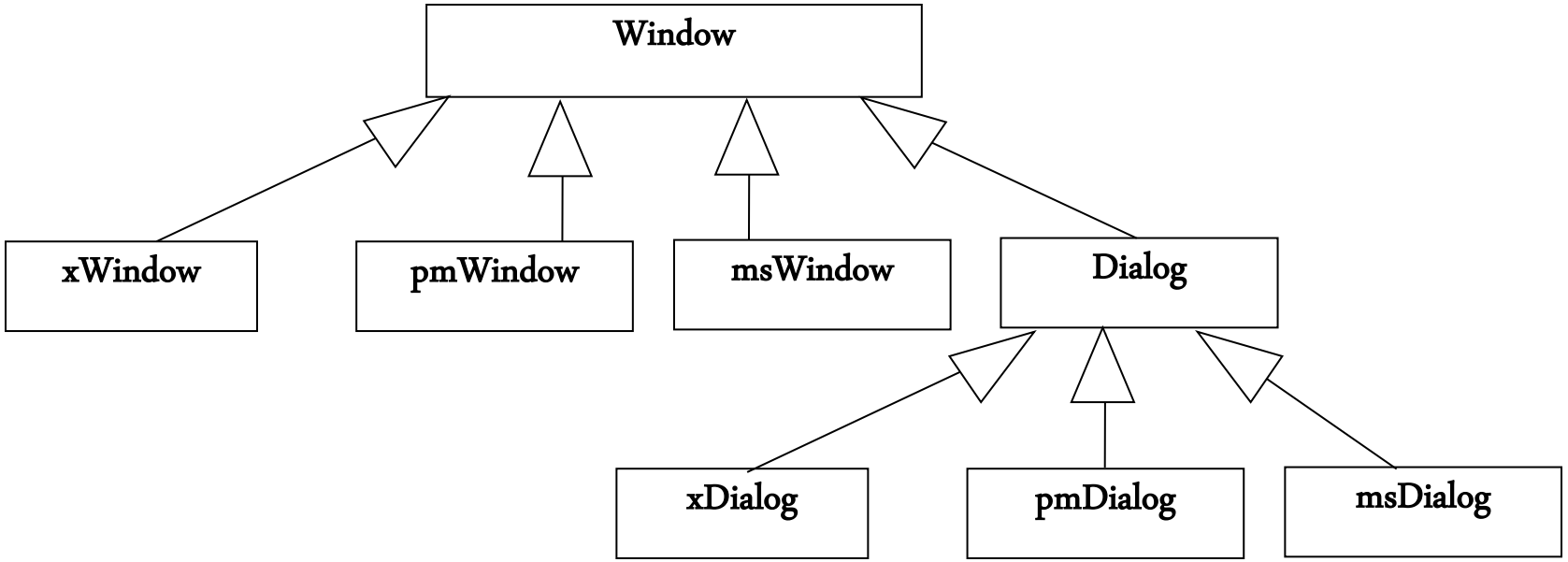


Maãu "Bridge"

- **Tên**: Bridge, tạm dịch "Cầu nối", thuộc lớp mẫu cấu trúc đối tượng.
- **Ý định**:Tách rời ngữ nghĩa của một vấn đề khỏi việc cài đặt; mục đích để cả hai bộ phận (ngữ nghĩa và cài đặt) có thể thay đổi độc lập nhau.
- **Motivation**: Giả sử phải xây dựng một lớp quản lý hệ thống "cửa sổ" của các hệ điều hành khác nhau, chẳng hạn cho 3 họ: X Window trên Unix, IBM's Presentation Manager (PM), và MS Windows của Microsoft. Cách làm tự nhiên là định nghĩa một lớp cơ sở Window và 3 lớp khác Xwindow, PMwindow, MSwindow kế thừa từ lớp Window như sau:



Giả sử có cần thêm lớp để quản lý các hộp hội thoại Dialog - một dạng Window đặc biệt. Như vậy lớp Dialog sẽ là đặc biệt hóa của lớp Window (kế thừa từ lớp Window). Kế đến lại là 3 lớp hội thoại cho 3 hệ thống cửa sổ khác nhau kế thừa từ Dialog



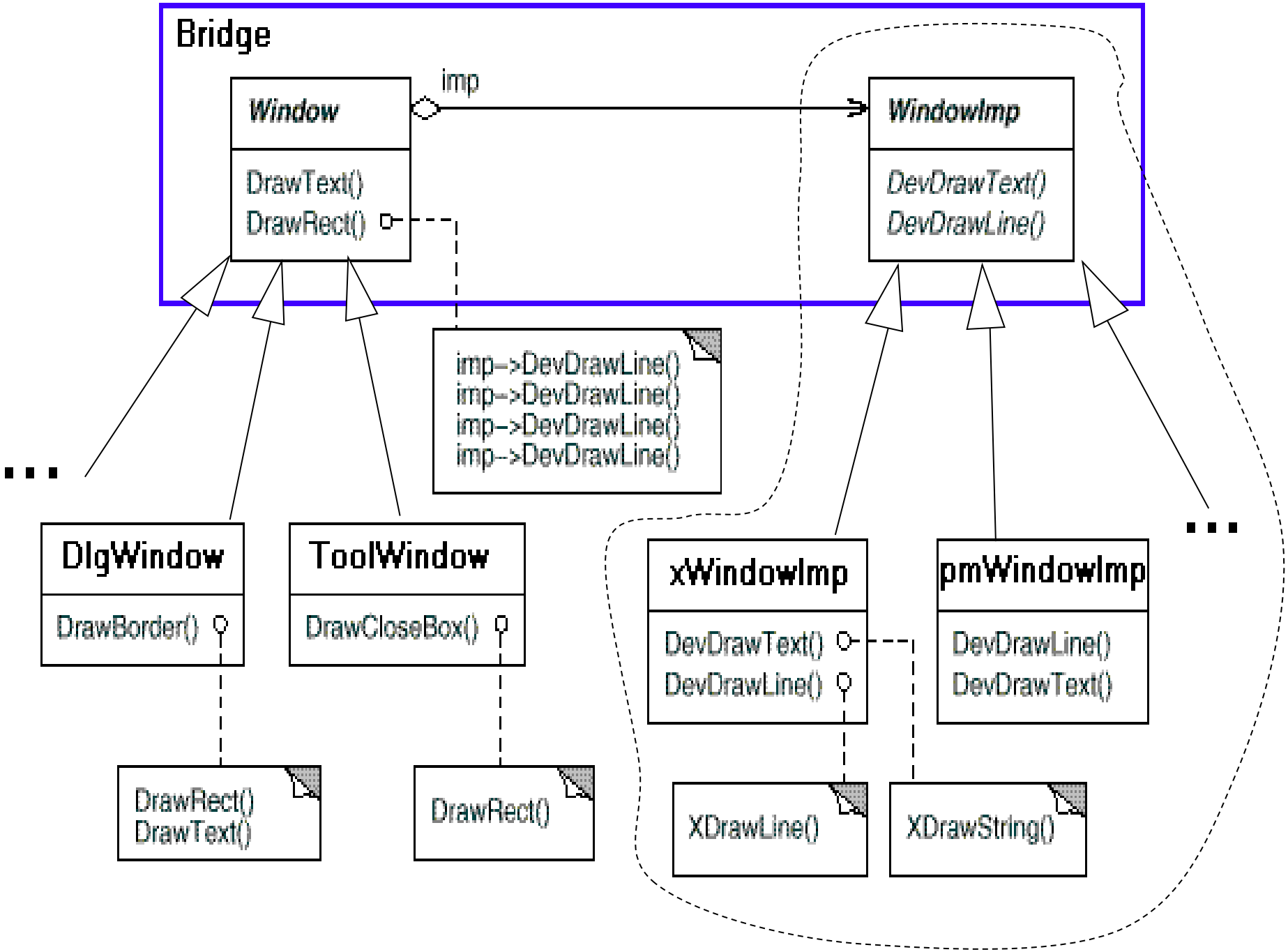
Maãu "Bridge"

- Cách tổ chức nầy bộc lộ các khuyết điểm như sau:
  - Bất tiện khi mở rộng: bổ sung hệ thống cửa sổ cho một hệ điều hành khác hay thêm các loại cửa sổ khác (chẳng hạn lớp các tool bar hay các control...).
  - Các chương trình sử dụng hệ thống lớp nầy sẽ phụ thuộc hệ điều hành. Tên các lớp phụ thuộc hệ điều hành bị rãi khắp trong cây kế thừa.

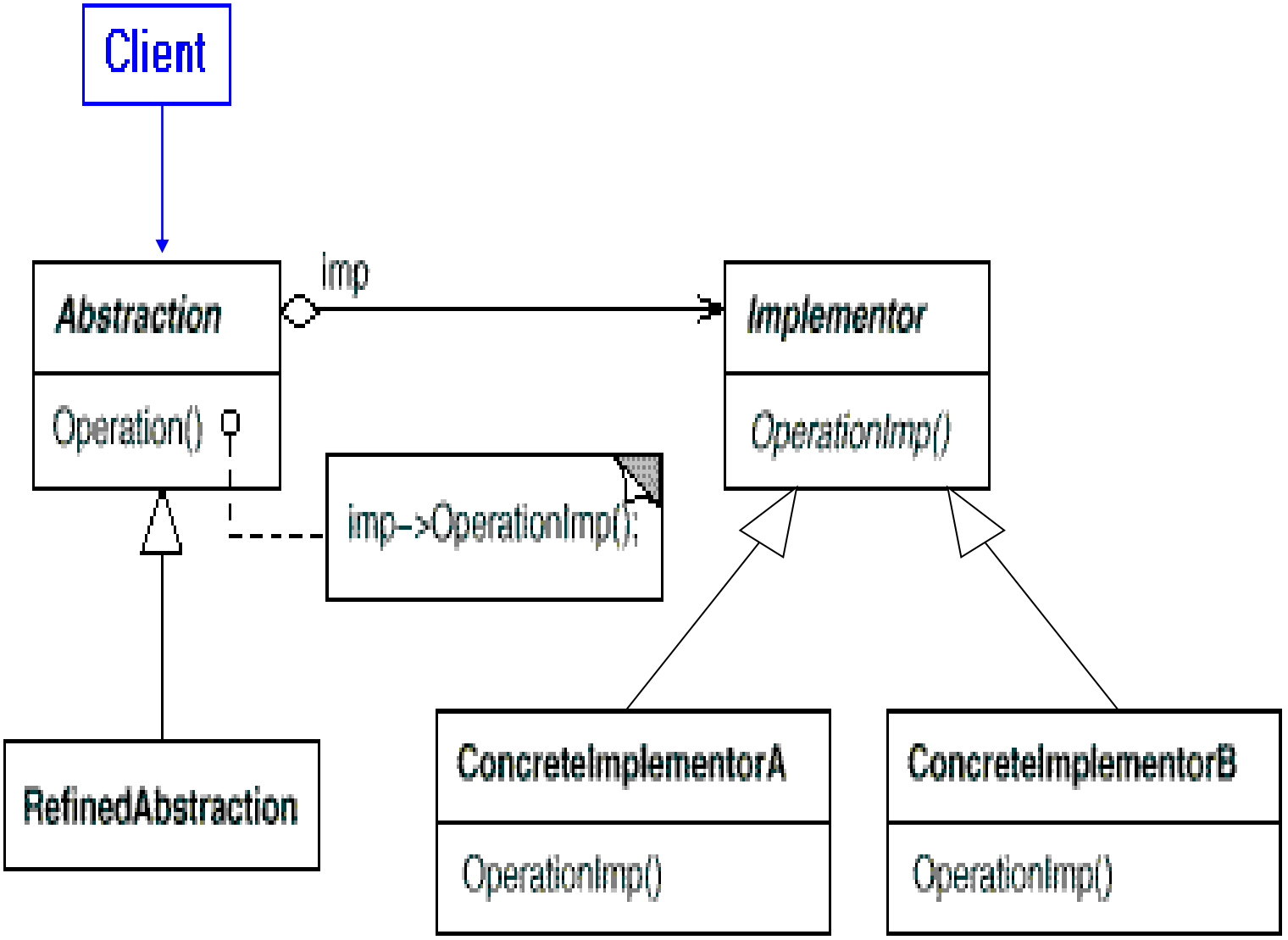
Maãu "Bridge"

- Ý tưởng của mẫu bridge là tách biệt hai phạm trù:
  - ý nghĩa của các cửa sổ (Window, Dialog, ToolBar, ...)
  - sự cài đặt phụ thuộc các hệ điều hành

thành 2 cây kế thừa riêng biệt và nối nhau bằng một "cầu"


Bridge
Window
DrawText()
DrawRect()
imp
WindowImp
DevDrawText()
DevDrawLine()
imp->DevDrawLine()
imp->DevDrawLine()
imp->DevDrawLine()
imp->DevDrawLine()
...
DlgWindow
DrawBorder()
ToolWindow
DrawCloseBox()
xWindowImp
DevDrawText()
DevDrawLine()
pmWindowImp
DevDrawLine()
DevDrawText()
...
DrawRect()
DrawText()
DrawRect()
XDrawLine()
XDrawString()

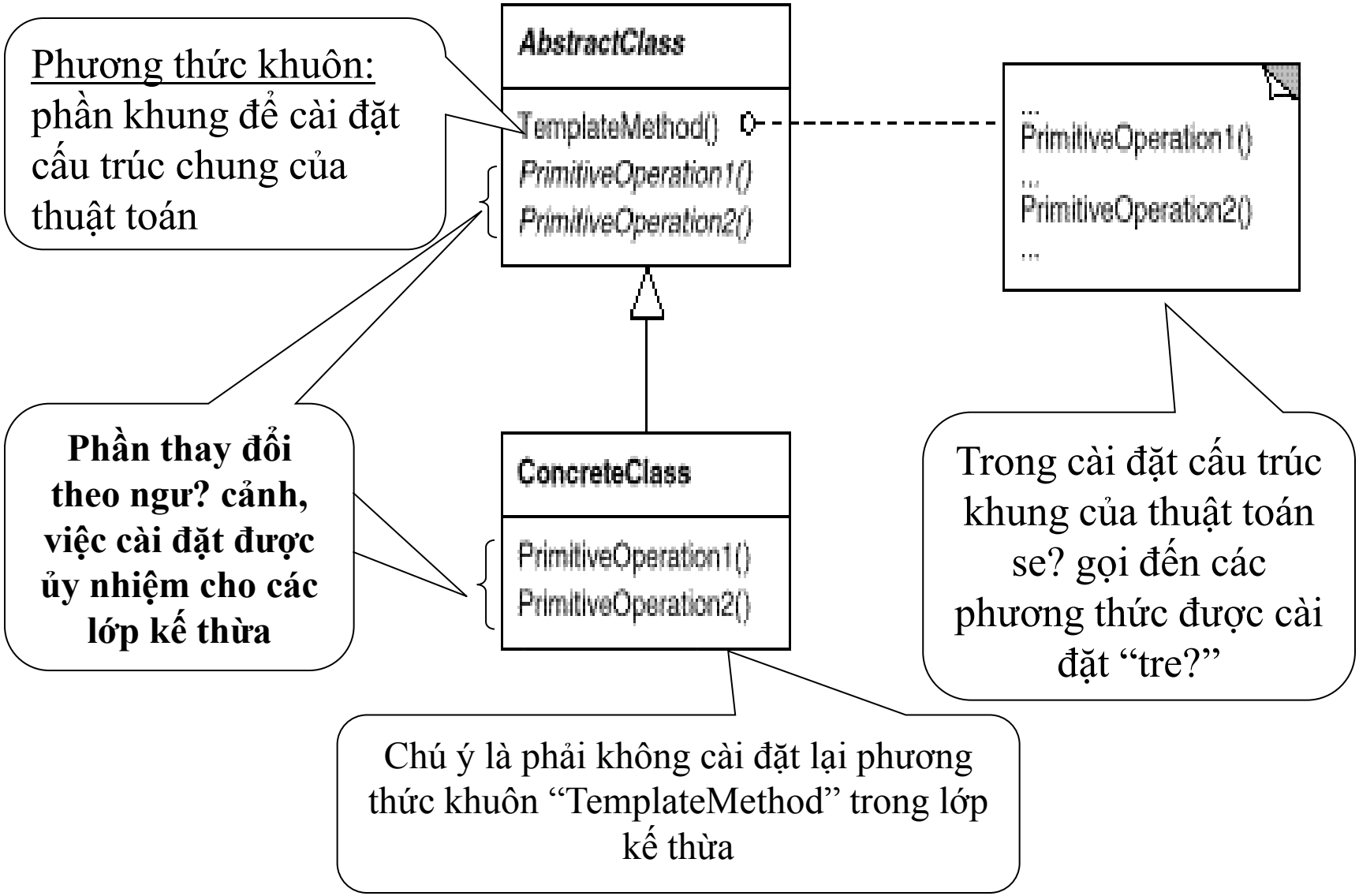
- Cấu trúc:


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

Mẫu “Template method”

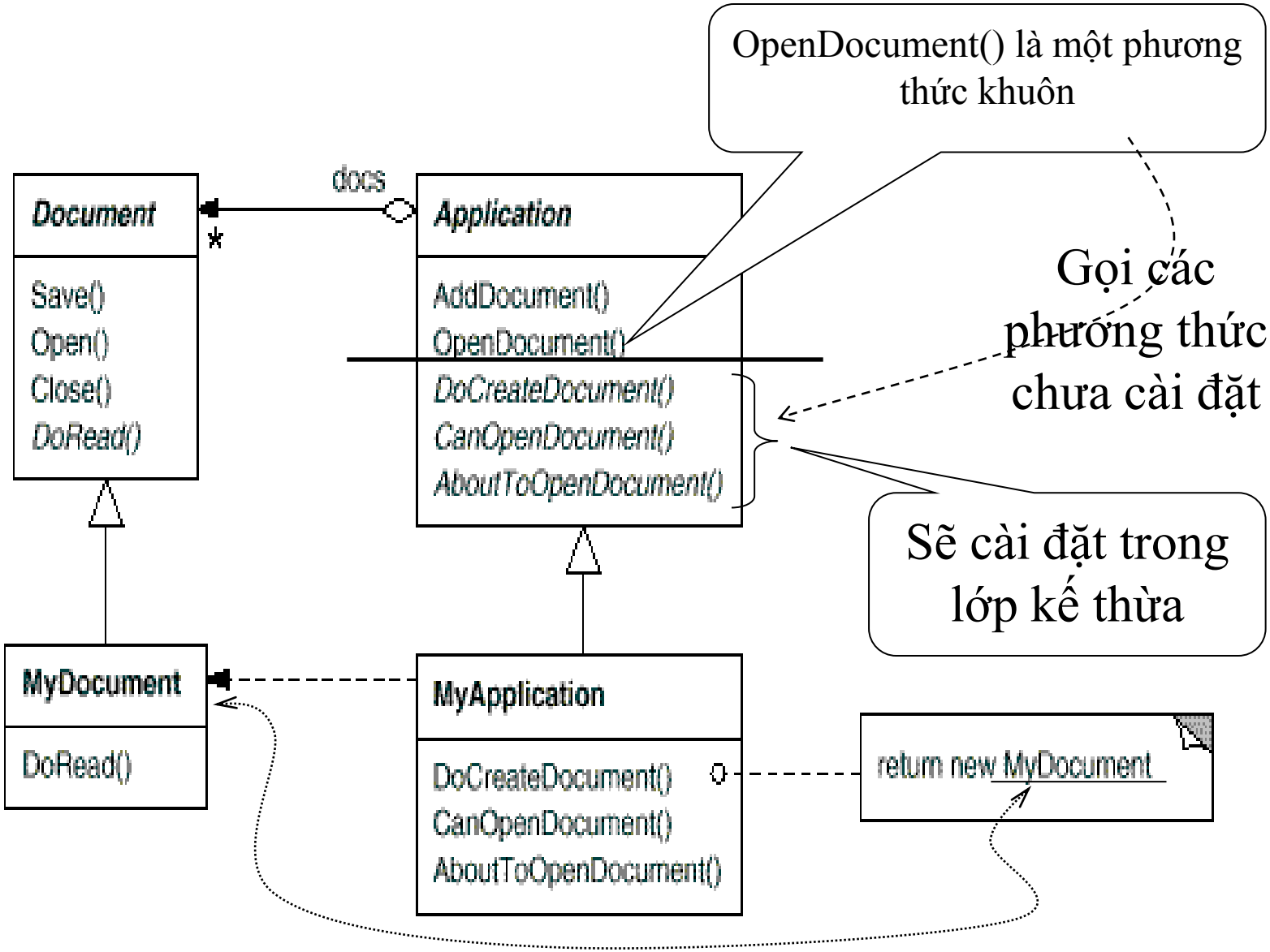
- **Tên**: Template method, tạm dịch “Phương thức khuôn”, thuộc lớp mẫu về ứng xử của lớp
- **Ý định**: Định nghĩa phần khung (**phần bất biến**) của một thuật toán, tức là một thuật toán tổng quát gọi đến một số phương thức (phần thay đổi tùy vào ngữ cảnh) chưa được cài đặt trong lớp cơ sở. Việc cài đặt của các phương thức nầy được ủy nhiệm cho các lớp kế thừa. Thuật toán chạy được cho các lớp kế thừa mà cấu trúc của thuật toán vẫn không thay đổi.

- Cấu trúc


Phương thức khuôn:
phần khung để cài đặt cấu trúc chung của thuật toán
AbstractClass
TemplateMethod()
PrimitiveOperation1()
PrimitiveOperation2()
...
PrimitiveOperation1()
...
PrimitiveOperation2()
...
Phần thay đổi theo ngư? cảnh, việc cài đặt được ủy nhiệm cho các lớp kế thừa
ConcreteClass
PrimitiveOperation1()
PrimitiveOperation2()
Trong cài đặt cấu trúc khung của thuật toán se? gọi đến các phương thức được cài đặt "tre?"
Chú ý là phải không cài đặt lại phương thức khuôn "TemplateMethod" trong lớp kế thừa

Mẫu “Template method”

- Cơ sở để cài đặt: dựa vào sự đa hình cho các phương thức được gọi bên trong phương thức khuôn. Xem ví dụ đơn giản viết bằng C++ sau đây


OpenDocument() là một phương thức khuôn
Gọi các phương thức chưa cài đặt
Sẽ cài đặt trong lớp kế thừa
docs
Document
Save()
Open()
Close()
DoRead()
Application
AddDocument()
OpenDocument()
DoCreateDocument()
CanOpenDocument()
AboutToOpenDocument()
MyDocument
DoRead()
MyApplication
DoCreateDocument()
CanOpenDocument()
AboutToOpenDocument()
return new MyDocument

```
void Application::OpenDocument (const cha
{
   if (!CanOpenDocument(name))
   {
        // cannot handle this document …
        return;
   }
   Document* doc = DoCreateDocument();
   if (doc != NULL)
   {
        _docs->AddDocument(doc);
        AboutToOpenDocument(doc);
        doc->Open();
        doc->DoRead();
   }
 }
```

Chú ý rằng các p. thức nầy sẽ được cài đặt lại trong các lớp kế thừa. Chẳng hạn như trong lớp MyApplication thì p.thức DoCreateDocument trả về một đối tượng kiểu MyDocument. Nhờ vậy, đối tượng doc có kiểu là MyDocument và lệnh gọi doc->DoRead sẽ gọi phương thức DoRead của lớp MyDocument.

Ví dụ: Thuật toán cây khung

Spanning Tree Algorithm

Böôùc 1. Choïn tuøy yù v ∈ X vaø khôûi taïo V := { v }; T := ∅

Böôùc 2. Choïn y ∈ X \ V sao cho coù moät caïnh e naøo ñoù cuûa G noái y vôùi moät ñænh x trong V

Böôùc 3. Gaùn V := V∪{y} vaø T:=T∪{e}

Böôùc 4. Neáu T ñuû n-1 phaàn töû thì döøng, ngöôïc laïi laøm tieáp tuïc böôùc 2.

PRIM Algorithm

Böôùc 1. Choïn tuøy yù v ∈ X vaø khôûi taïo V := { v }; T := ∅

Böôùc 2. Choïn caïnh e coù troïng soá nhoû nhaát noái ñænh x∈X vôùi ñænh y ∈ X \ V

Böôùc 3. Gaùn V := V∪{y} vaø T:=T∪{e}

Böôùc 4. Neáu T ñuû n-1 phaàn töû thì döøng, ngöôïc laïi laøm tieáp tuïc böôùc 2.

```
class ARC {
  // Some members...
  // ...
};

class SpanningTree {
  int n;
  int nT;
  // Some other members...
  // ...

  void initialize();
  void add_Vertex_to_V(int y);
  void add_Arc_to_T(ARC e);
  virtual int search_Arc(ARC& e, int& x, int& y);
  int SpanningTreeAlgorithm();
};
ï
```

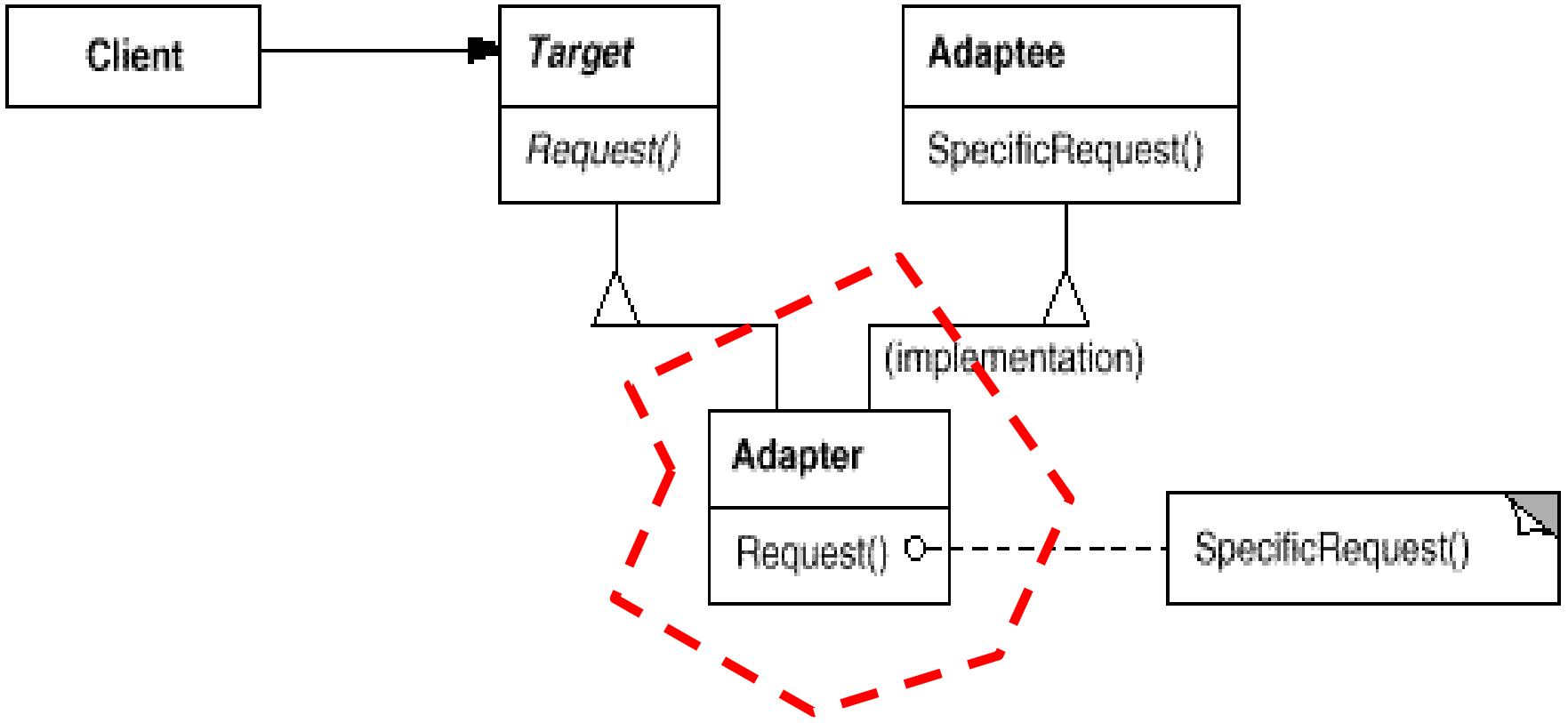
```
int SpanningTree::SpanningTreeAlgorithm() {
  initialize();
  while(nT < n-1){
                ARC e;
                int x, y;
                if(!search_Arc(e, x, y))
                   return 0;
                add_Vertex_to_V(y);
                add_Arc_to_T(e);
  }
  return 1;
}
```

```
class PrimSpanningTree: public SpanningTree{
   int search_Arc(ARC& e, int& x, int& y) {
                // to find the minimum arc..
                // ...
                return 1;
   };
};
```

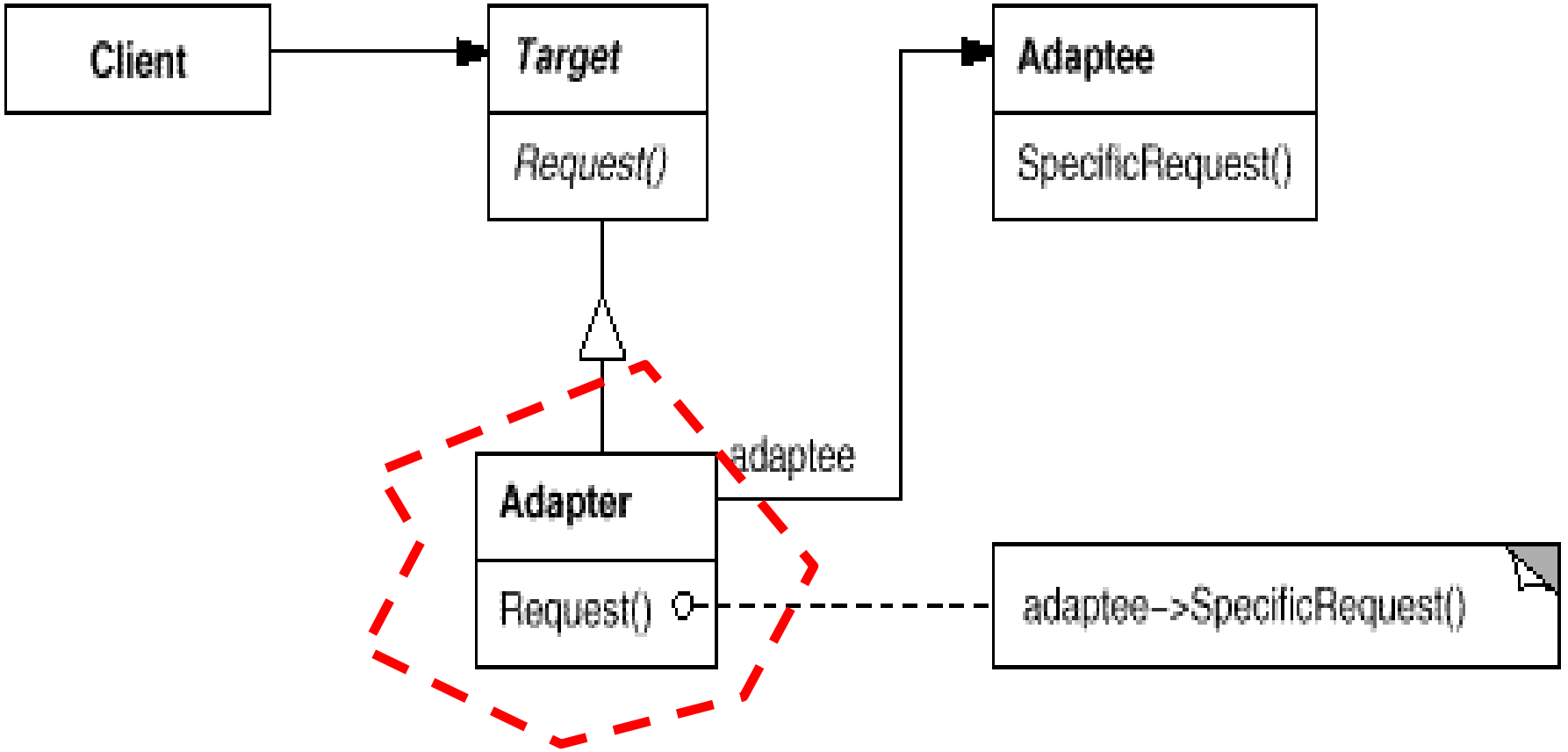
Mẫu Adapter

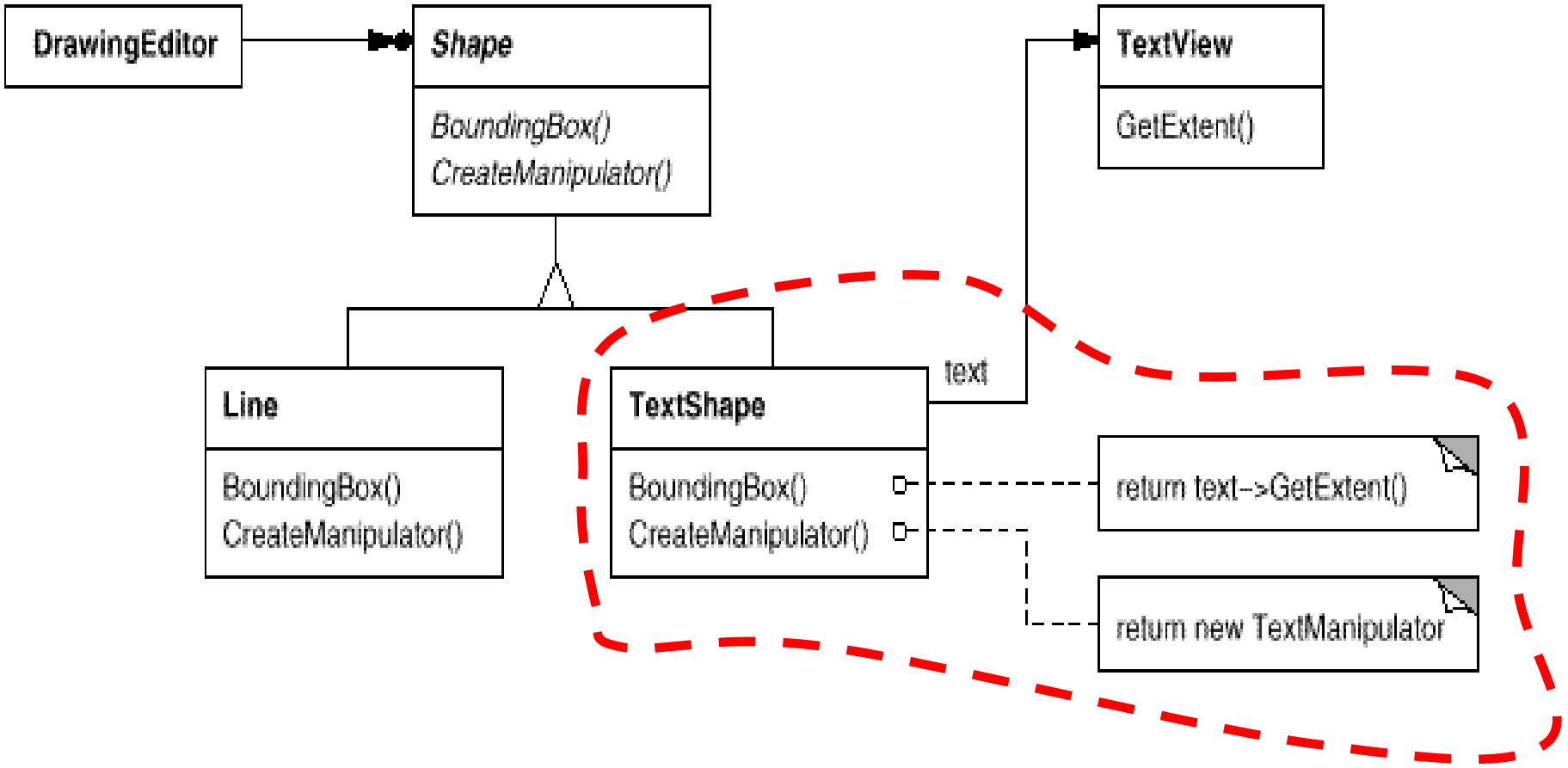
- **Mục đích**: thay giao tiếp của một lớp bởi một giao tiếp khác phù hợp với yêu cầu người sử dụng lớp, nhằm giải quyết bài toán tương thích.
- Cấu trúc của mẫu Adapter: có 2 dạng
  - Multiple composition
  - Object composition

Adapter structure (Multiple inheritance)

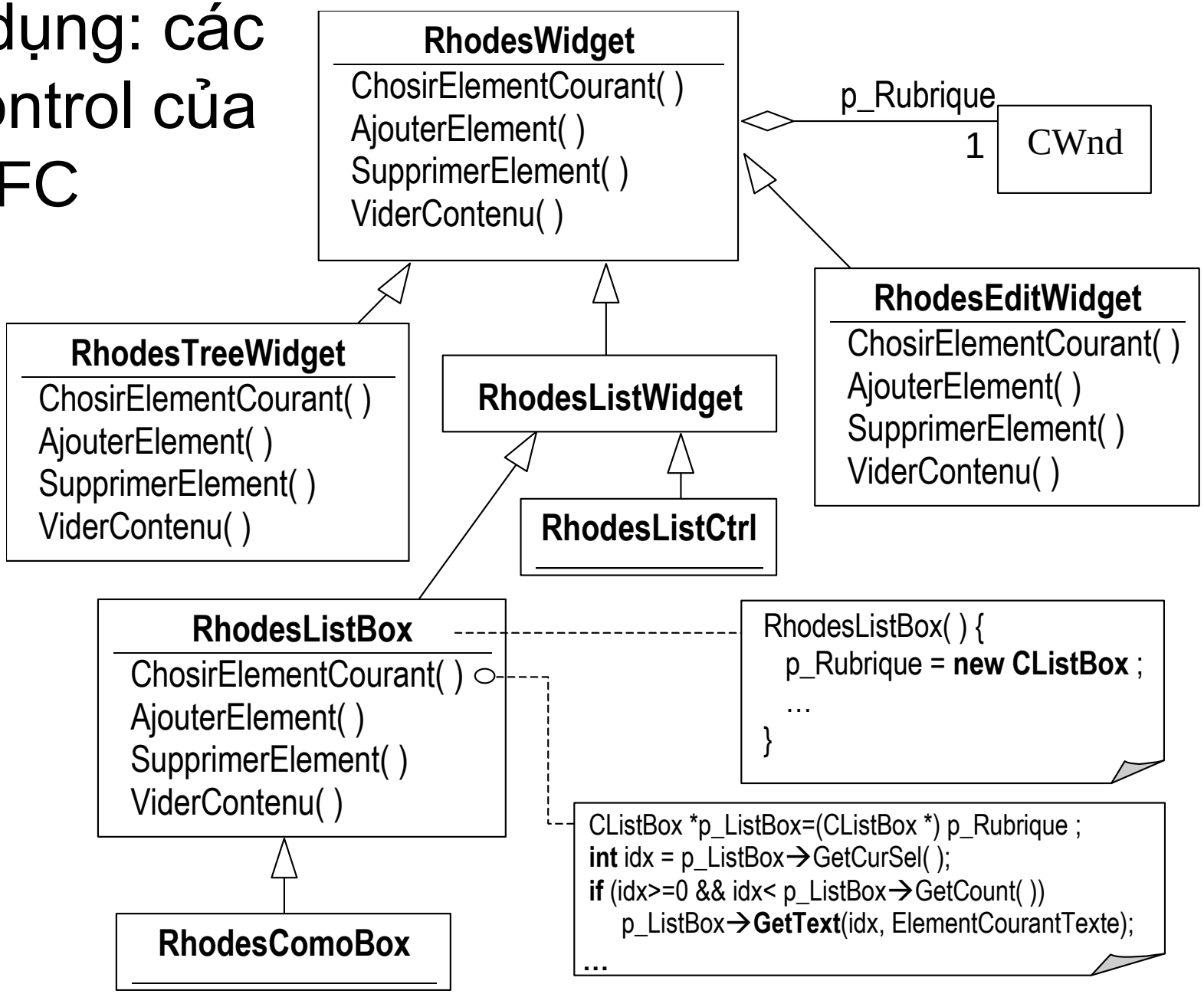


Adapter structure (Object composition)




DrawingEditor
Shape
BoundingBox()
CreateManipulator()
TextView
GetExtent()
Line
BoundingBox()
CreateManipulator()
TextShape
BoundingBox()
CreateManipulator()
text
return text->GetExtent()
return new TextManipulator

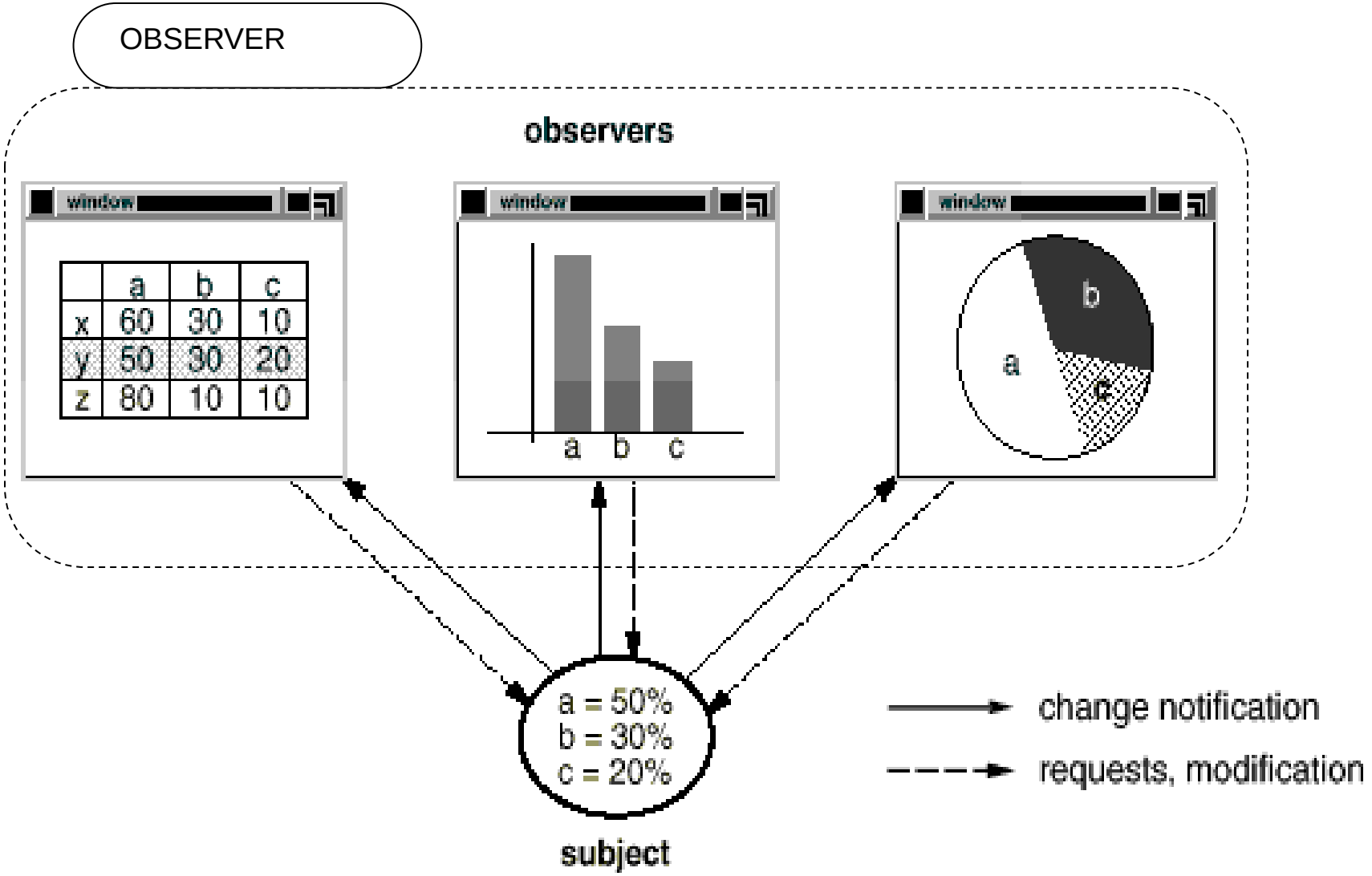
Áp dụng: các control của MFC



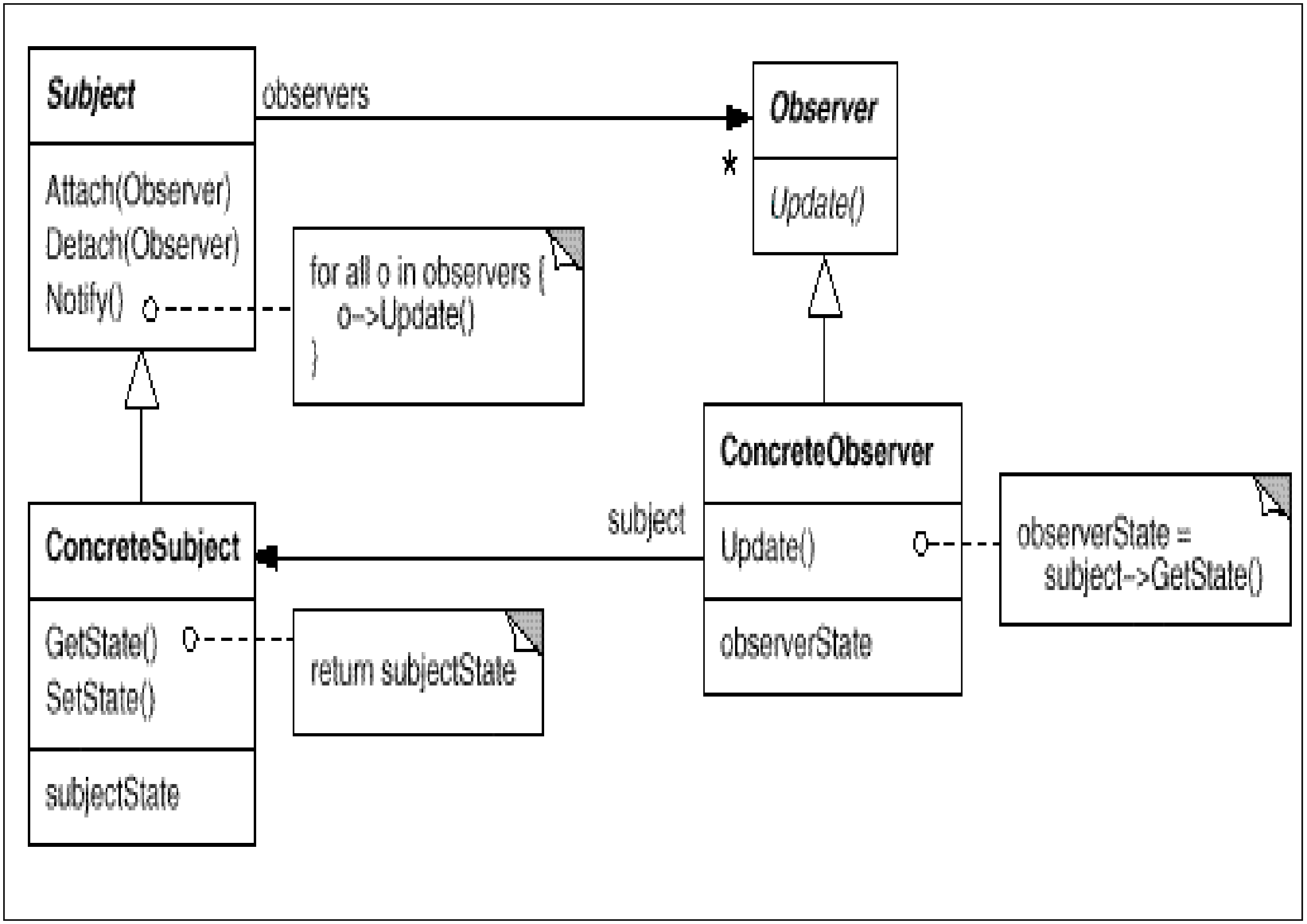
Mẫu “Observer”

- **Tên**: Observer, tạm dịch “Quan sát viên”, thuộc lớp mẫu về ứng xử của đối tượng
- **Ý định**:định nghĩa quan hệ phụ thuộc một-nhiều giữa các đối tượng. Khi một đối tượng thay đổi trạng thái, tất cả các đối tượng phụ thuộc được thông báo và cập nhật trạng thái.
- **Motivation**: Trong các phần mềm có giao diện đồ họa với người sử dụng, thông thường có nhiều biểu diễn đồ họa khác nhau cho cùng một lô dữ liệu, khi dữ liệu thay đổi thì tất cả các biểu diễn đồ họa phải được cập nhật

Hình vẽ sau minh họa trường hợp của các bảng tính điện tử (Excel, Lotus 1-2-3..):

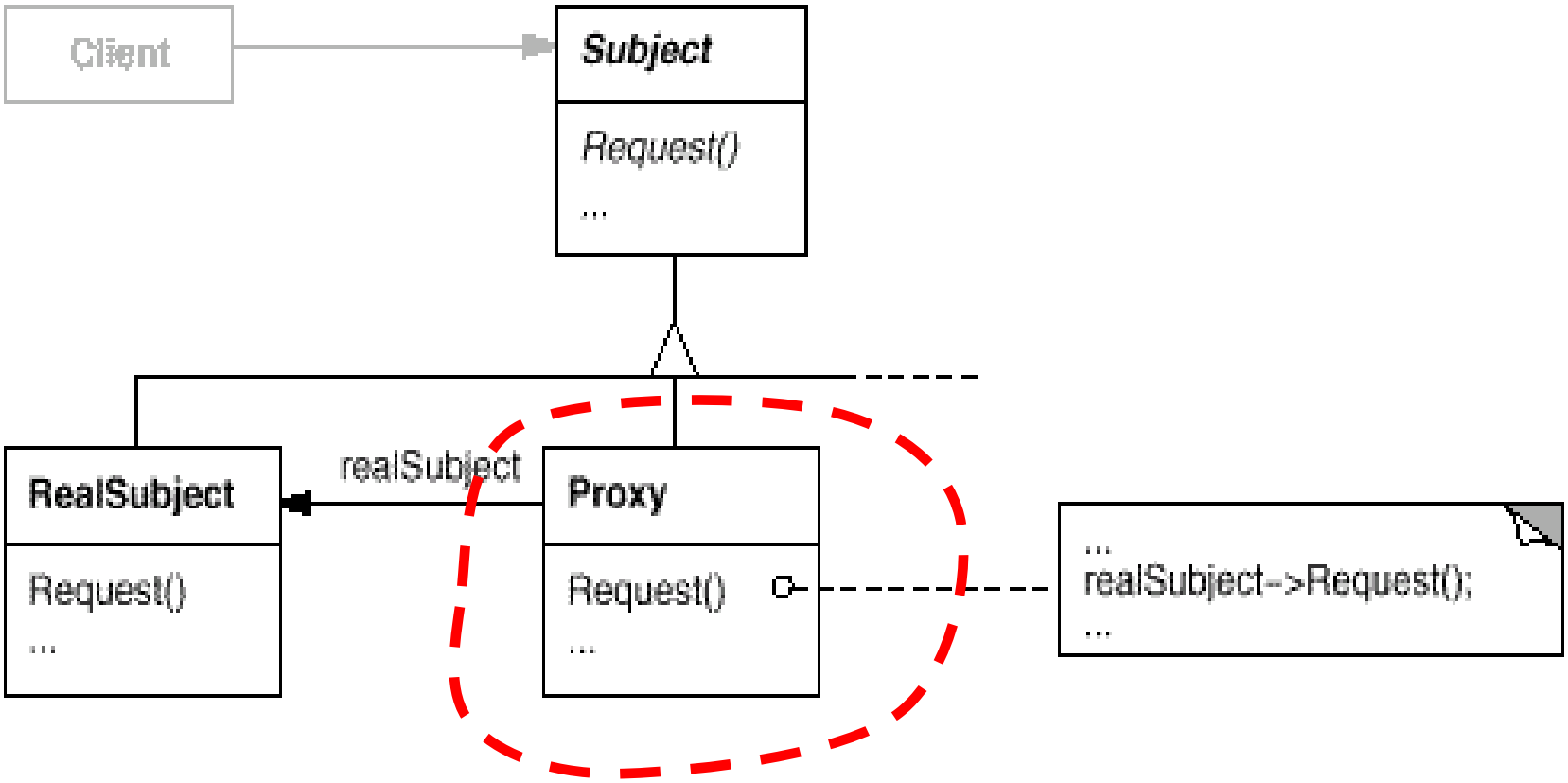


Cấu trúc


Subject
observers
Observer
Attach(Observer)
Detach(Observer)
Notify()
*
Update()
for all o in observers {
o->Update()
}
ConcreteObserver
subject
ConcreteSubject
Update()
observerState =
subject->GetState()
GetState()
SetState()
return subjectState
observerState
subjectState

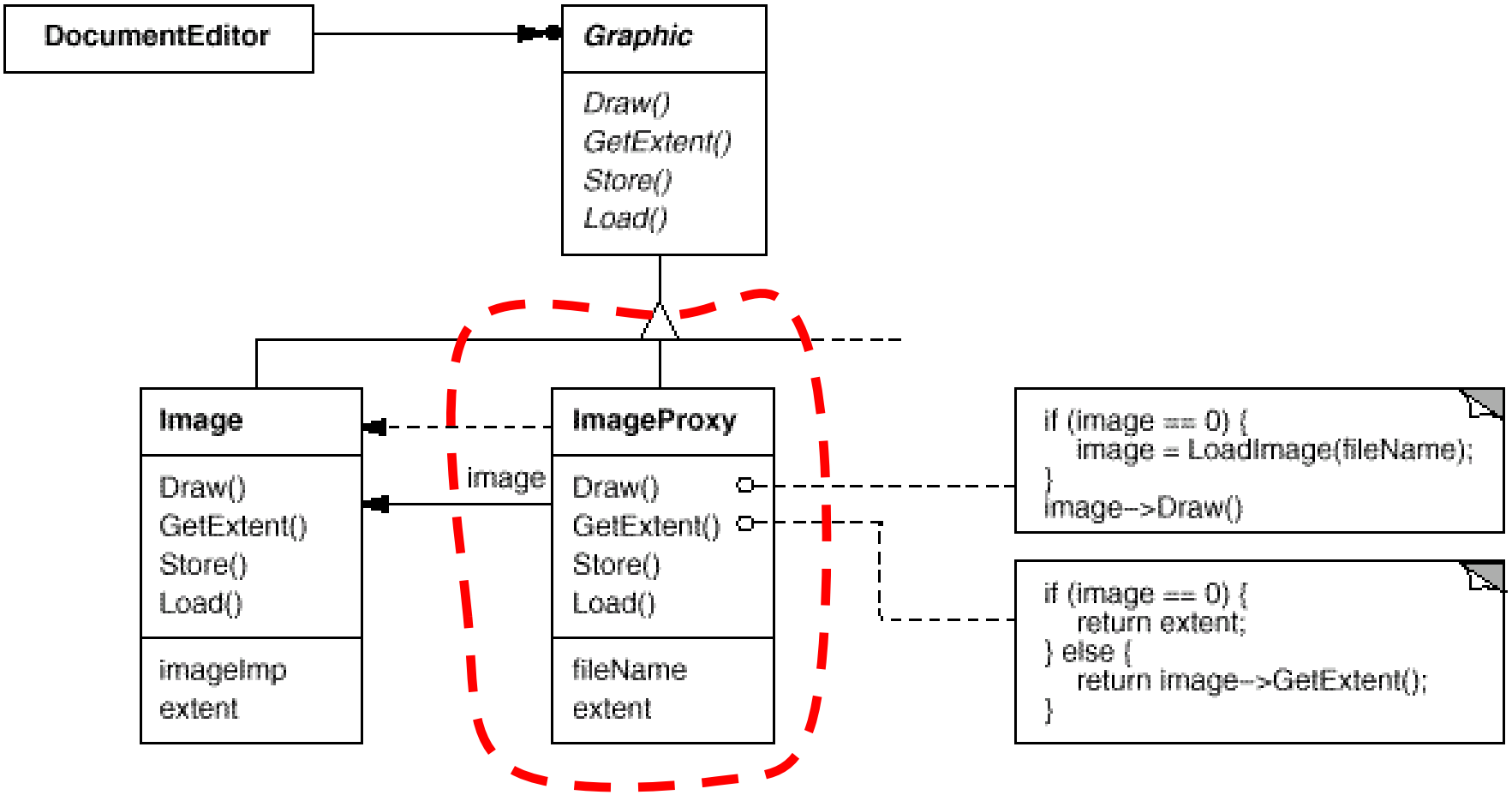
Mẫu proxy

- **Mục đích**: Truy xuất một đối tượng thông qua một đối tượng được ủy nhiệm



Dùng trong các trường hợp

- Remote object
- Expensive cost of creation and initialization of the object
- To protect the original object
- A smart reference
  - Count number of references to the real object: to free automatically
  - Loading a persistent object into memory when it's first referenced
  - Checking that the real object is locked


DocumentEditor
Graphic
Draw()
GetExtent()
Store()
Load()
Image
Draw()
GetExtent()
Store()
Load()
imageImp
extent
image
ImageProxy
Draw()
GetExtent()
Store()
Load()
fileName
extent
if (image == 0) {
image = LoadImage(fileName);
}
image->Draw()
if (image == 0) {
return extent;
} else {
return image->GetExtent();
}

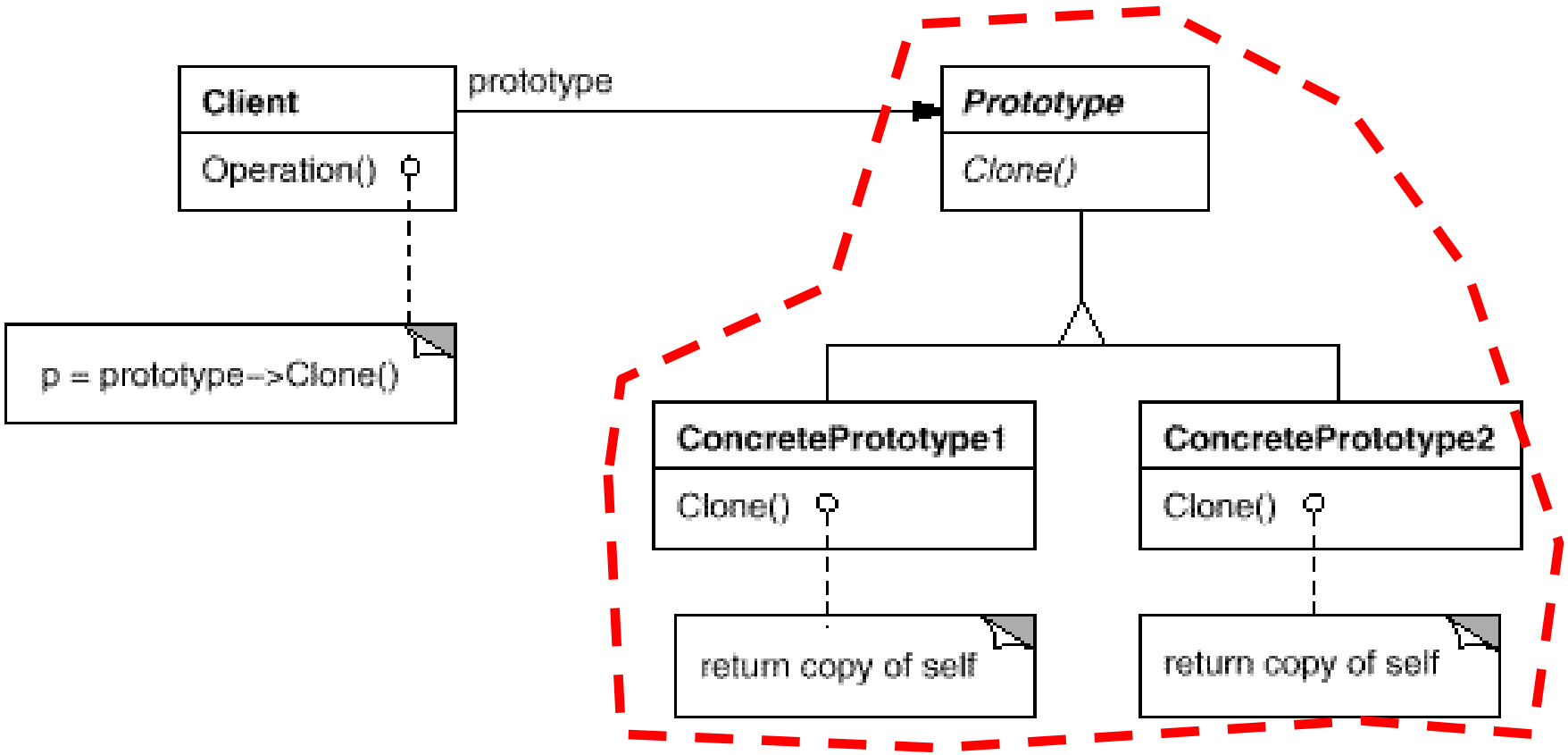
Code example (Proxy)

```
class Graphic {
   public:
        …
};
class Image : public Graphic {
   public:
        Image(const char* file);
        // loads image from a file
        …
};
class ImageProxy : public Graphic {
    public:
        ...
        ImageProxy(const char*
        imageFile);
        Image* GetImage();
    private:
         Image* _image;
         Point _extent;
         char* _fileName;
};
```

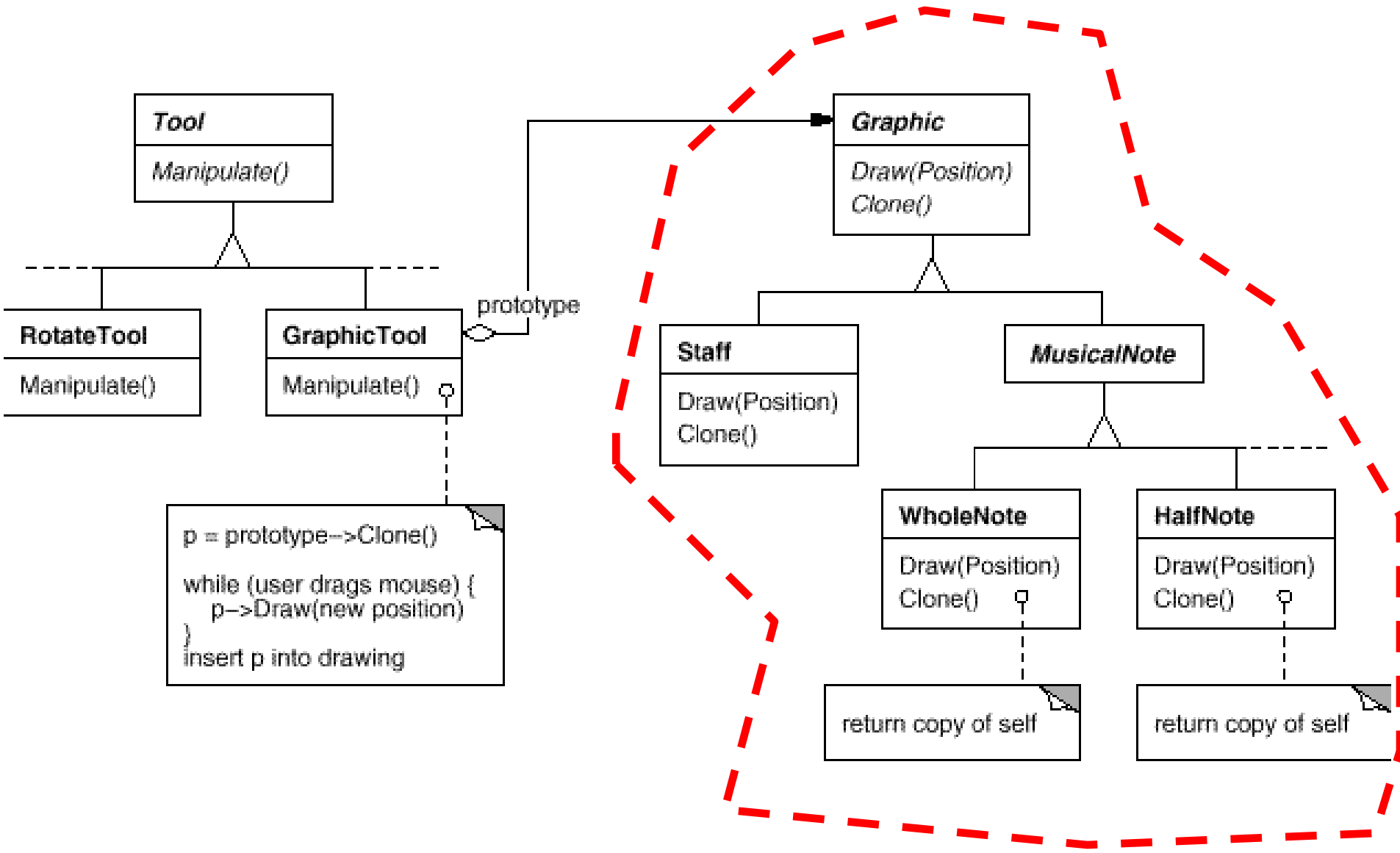
```
ImageProxy::ImageProxy (const char*
        fileName)  {
         _fileName = strdup(fileName);
         _extent = Point::Zero;  // don't
        know extent yet
         _image = 0;
}

Image* ImageProxy::GetImage() {
         if (_image == 0) {
              _image = new
        Image(_fileName);
         }
         return _image;
}
const Point& ImageProxy::GetExtent () {
         if (_extent == Point::Zero) {
              _extent = GetImage()-
        >GetExtent();
         }
         return _extent;
}
```

Mẫu prototype

- Muïc ñích: Qui ñònh loaïi cuûa caùc ñoái töôïng caàn taïo baèng caùch duøng moät ñoái töôïng maãu, taïo môùi nhôø vaøo sao cheùp ñoái töôïng maãu naày.



Mẫu prototype: ví dụ



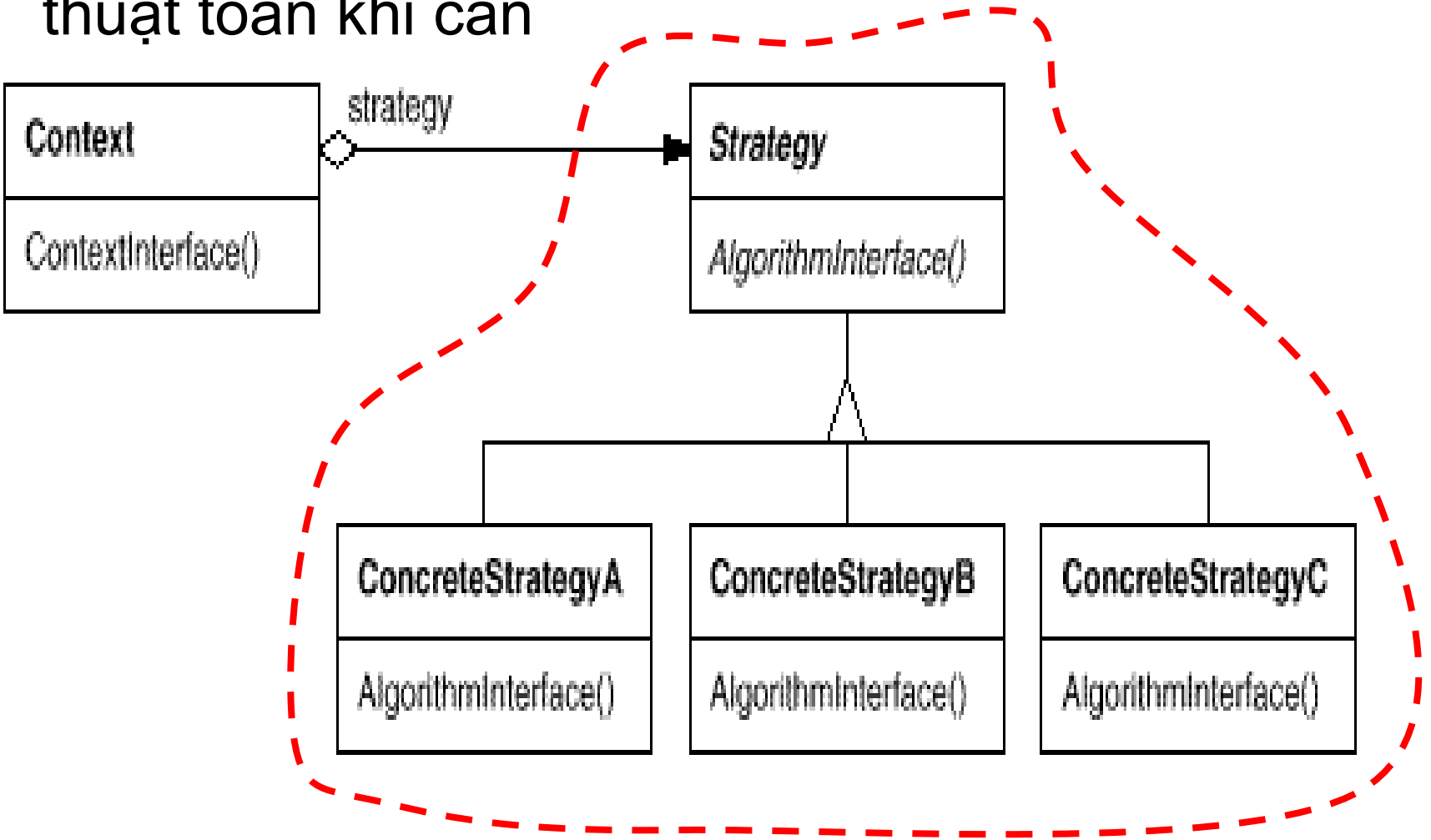
Maãu prototype: code example

```
class Staff : public Graphic {
    public:
        Staff();
        Staff(const Staff&);

        virtual Staff* Clone();
        ...
};

Staff::Staff (const Staff& other) {
        // copying members...
}

Staff* Staff::Clone () {
        return new Staff(*this);
}
```

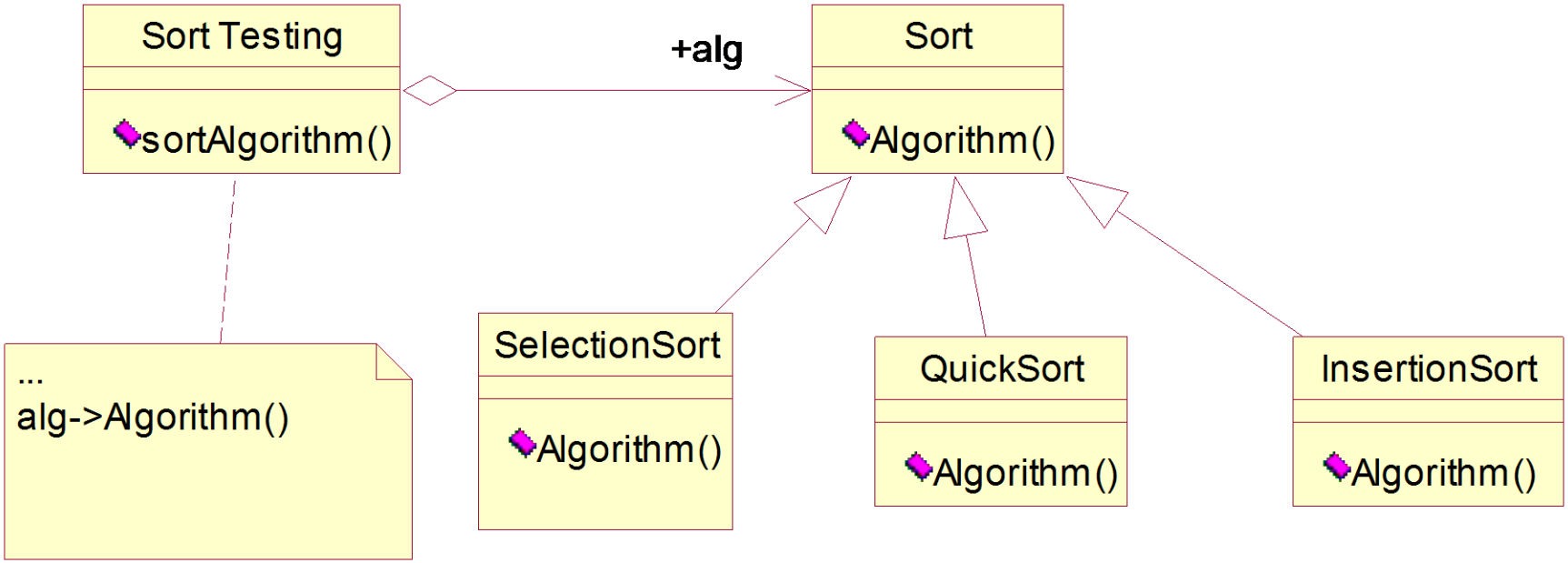
```
struct Mapping {
   char *ObjectType;  Graphic* Object;
};
Mapping sampleobjects[]={
    {“Staff”, new Staff(…)},
    {“WholeNote”, new WholeNote(…)},
    {“HalfNote”, new HalfNote(…)}
};
Graphic* sampleSearch(char*
strType){
     …
}
Graphic* objCreate(char* strType){
    Graphic*
sObj=sampleSearch(strType);
    if(sObj!=NULL)
            return sObj->Clone();
   else  return NULL;
}
```

Mẫu strategy

- **Mục đích**: bao bọc một họ các thuật toán bằng các lớp đối tượng để ứng dụng có thể chọn lựa thuật toán khi cần



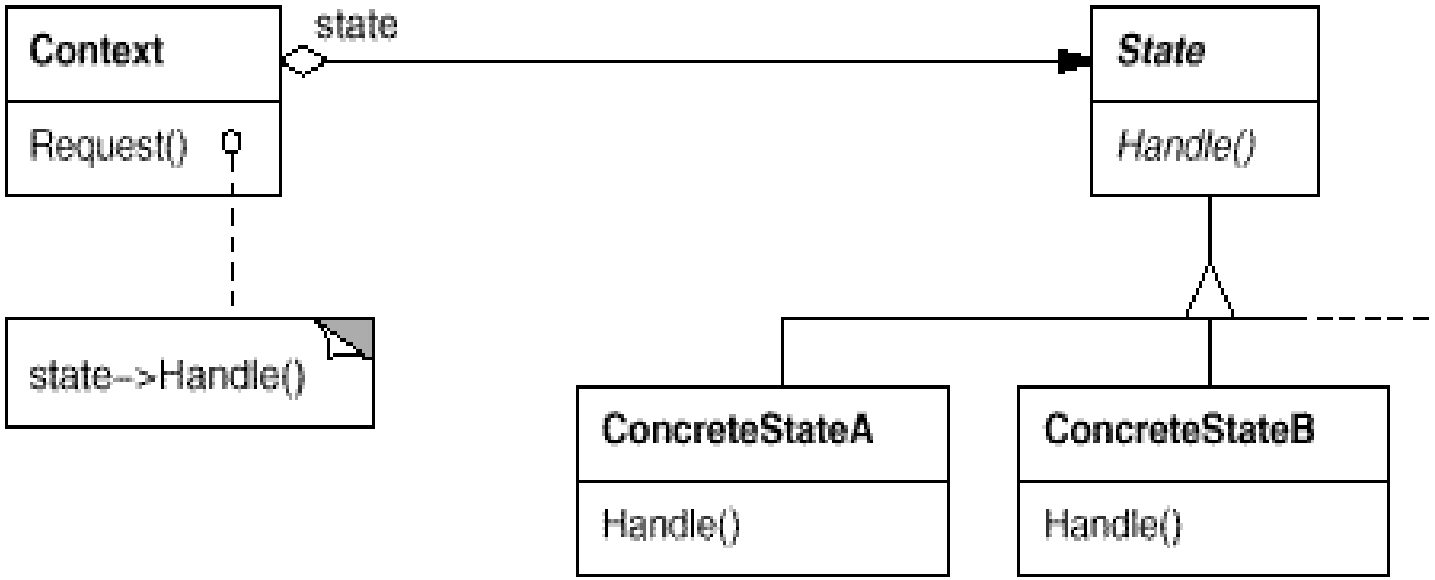
Mẫu strategy: ví dụ



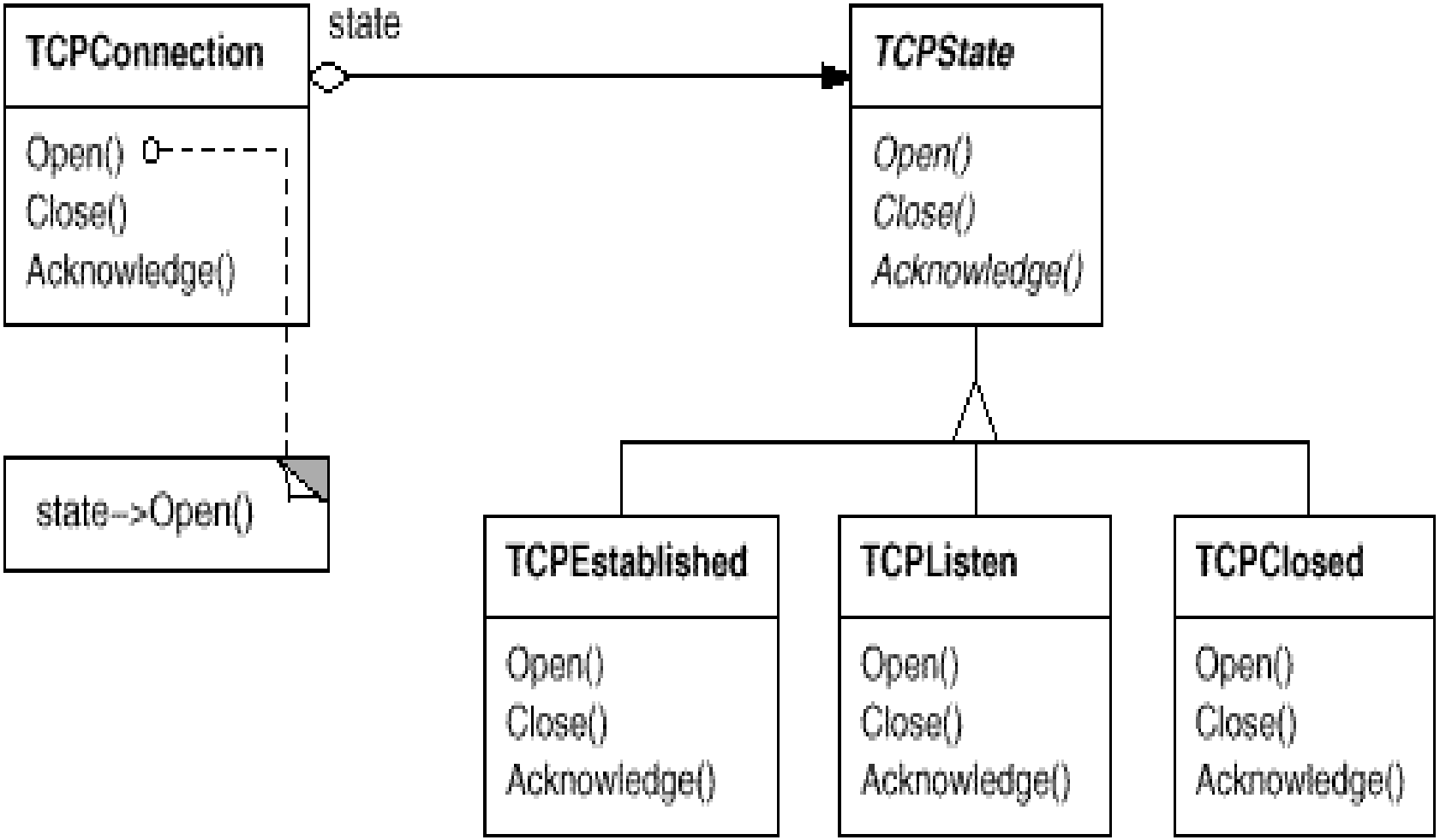

```
SortTesting*  qsortTest= new SortTesting(new QuickSort);
// Data input…
qsortTest->sortAlgorithm(…);
```

Mẫu state

- Mục đích: cho phép thay đổi ứng xử của đối tượng tùy theo sự thay đổi trạng thái bên trong của nó
- Mẫu nầy rất giống với mẫu strategy về mặt hình thức, tuy nhiên khác về ý nghĩa



Mẫu state: ví dụ



Các tiếp cận tương tự

- Các tiếp cận dùng lại theo kiểu pattern:
  - Design patterns [Gam95]
  - GRASP pattern [Lar98]
  - Analysis patterns [Fow97]
  - Process patterns [Amb99]
- Các kinh nghiệm phương pháp luận để định hướng cho quá trình thiết kế
  - Design heuristics [Rie96]
  - GRASP: General Responsibility Assignment Software Patterns.

Các tiếp cận tương tự

- Việc hỗ trợ dùng lại cho quá trình phát triển phần mềm
  - Frameworks
  - Software components

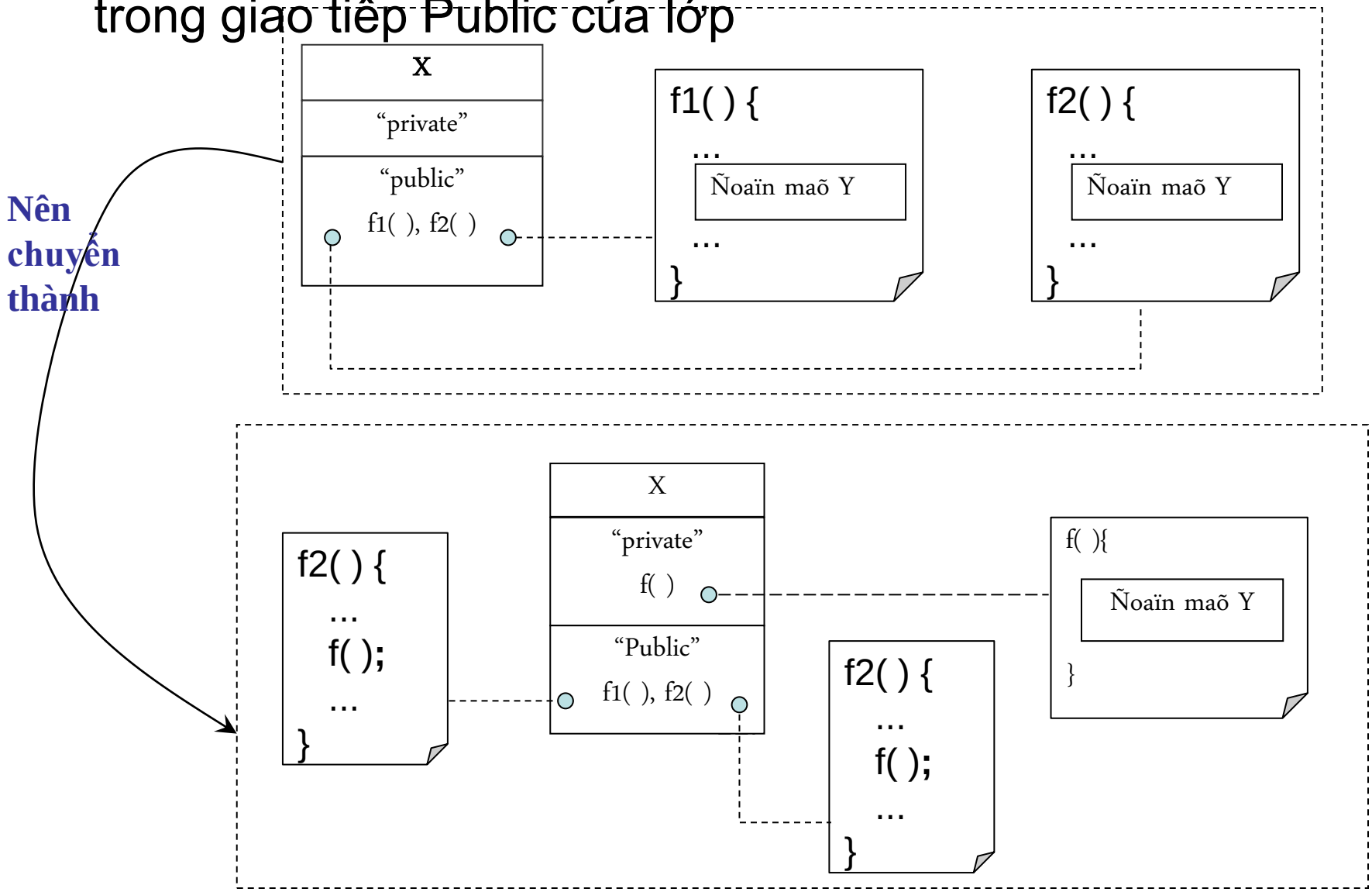
Design heuristics

- **Khái niệm**: Mỗi heuristic thiết kế là một hướng dẫn súc tích về mặt phương pháp luận cho một vấn đề nhỏ trong thiết kế phần mềm.
- **Nguồn gốc**: T. giả Arthur J. Riel [Rie96] đã thu thập và hệ thống hóa 61 heuristic liên quan đến thiết kế phần mềm hướng đối tượng.
- Mã số của các heuristic sau đây được lấy theo tài liệu gốc của tác gia

Vài heuristic chung về lớp và đối tượng

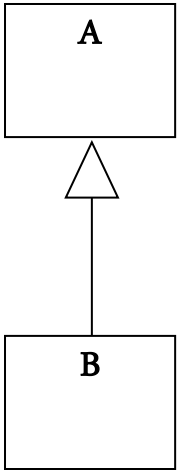
- **Heuristic 2.1**: Dữ liệu của lớp nên được che dấu bên trong lớp đó (Việc thao tác dữ liệu của lớp nên thực hiện gián tiếp qua các phương thức)
- **Heuristic 2.7**: Một lớp chỉ nên dùng các phương thức, các phép toán trong phần giao tiếp Public của lớp khác.
  - Hệ quả: giảm sự coupling giữa các lớp.
  - Như vậy không nên dùng cơ chế “friend” như của C++ để truy xuất vào các phần cài đặt bên trong lớp khác.

- Heuristic 2.5: Không nên trình bày cài đặt chi tiết của các phương thức khác nhau có đoạn mã nguồn giống nhau trong giao tiếp Public của lớp



- **Heuristic 5.2**: Các lớp kế thừa phải có tri thức về lớp cơ sở của chúng (do quan hệ kế thừa), nhưng lớp cơ sở không nên biết bất kỳ điều gì về lớp kế thừa của nó.

  Lý do: Vì nếu lớp cơ sở có chứa tri thức về các lớp kế thừa thì khi có thêm lớp kế thừa mới từ lớp cơ sở đó, mã nguồn của lớp cơ sở có khả năng bị thay đổi.



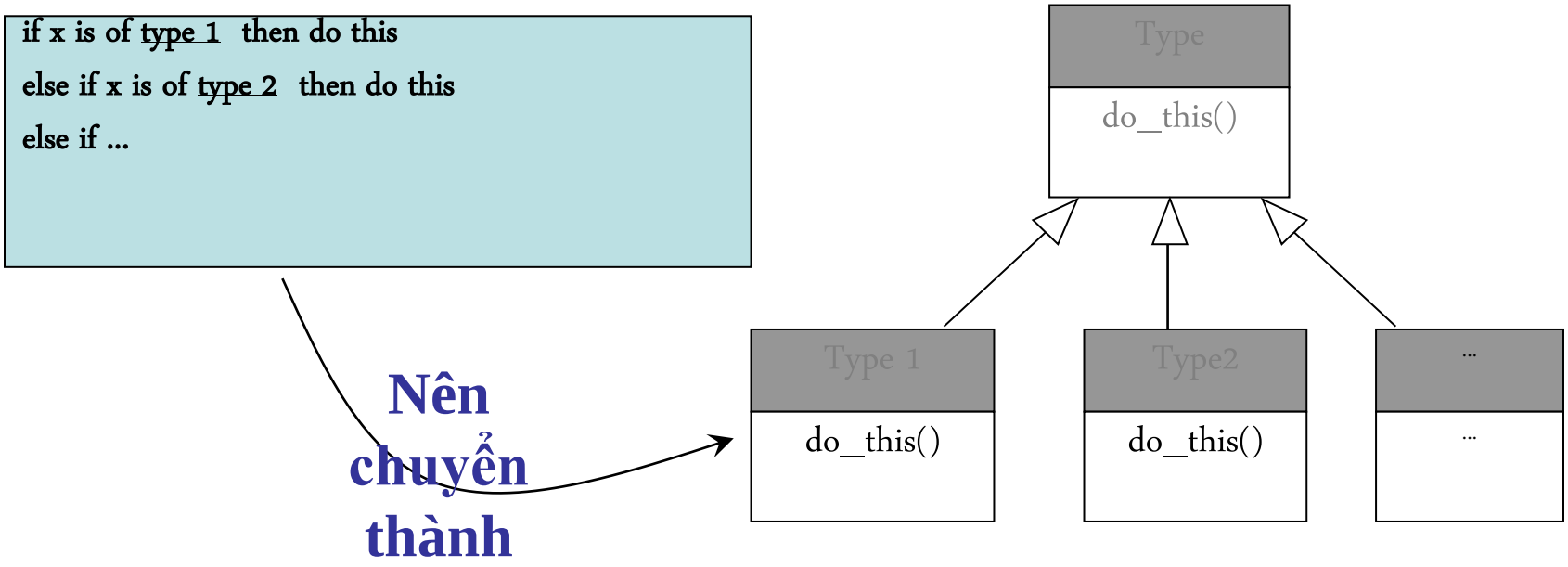
A không nên biết thông tin gì về B, nếu A có tri thức về B thì A sẽ mất tính tổng quát

Vài heuristic chung

- **Heuristic 5.4 và 5.5**: Về mặt lý thuyết, sự phân cấp các lớp kế thừa càng mịn (nhiều tầng) càng tốt. Về mặt thực hành, số tầng phân cấp không nên vượt quá một số trung bình mà một người thông thường có thể theo dõi tốt các lớp trong cây kế thừa. Con số trung bình này không nên vượt quá 6 tầng.
- **Heuristic 5.8**: Các thuộc tính và phương thức chung cho nhiều lớp nên ở tầng cao nhất có thể được trong phân cấp kế thừa.

- **Heuristic 5.9 và 5.10:**
  - Nếu hai hay nhiều lớp chỉ chia sẻ chung phần dữ liệu (không có phương thức chung) thì phần dữ liệu chung đó nên là một thuộc tính có kiểu là một lớp mới bao bọc phần dữ liệu chung đó.
  - Nếu hai hay nhiều lớp chia sẻ chung dữ liệu và các phương thức thì các lớp đó nên kế thừa từ một lớp cơ sở chung mà bao gồm dữ liệu và các phương thức đó.

- **Heuristic 5.12**: Không nên dùng kỹ thuật kiểm tra kiểu của đối tượng. Trong hầu hết các trường hợp, có thể dùng đa hình để giải quyết vấn đề



- **Heuristic 5.13**:Trường hợp cần kiểm tra giá trị thuộc tính của đối tượng của lớp để thực hiện các ứng xử hoàn toàn khác nhau thì lớp đó nên phân rã thành nhiều lớp, mỗi giá trị của thuộc tính sẽ là một lớp kế thừa từ lớp cơ sở.
- **Heuristic 5.14**: Không chuyển các đối tượng của một lớp thành các lớp kế thừa, phải cẩn thận bởi vì nhiều khi lớp kế thừa chỉ là một thể hiện (một đối tượng) của lớp cơ sở.