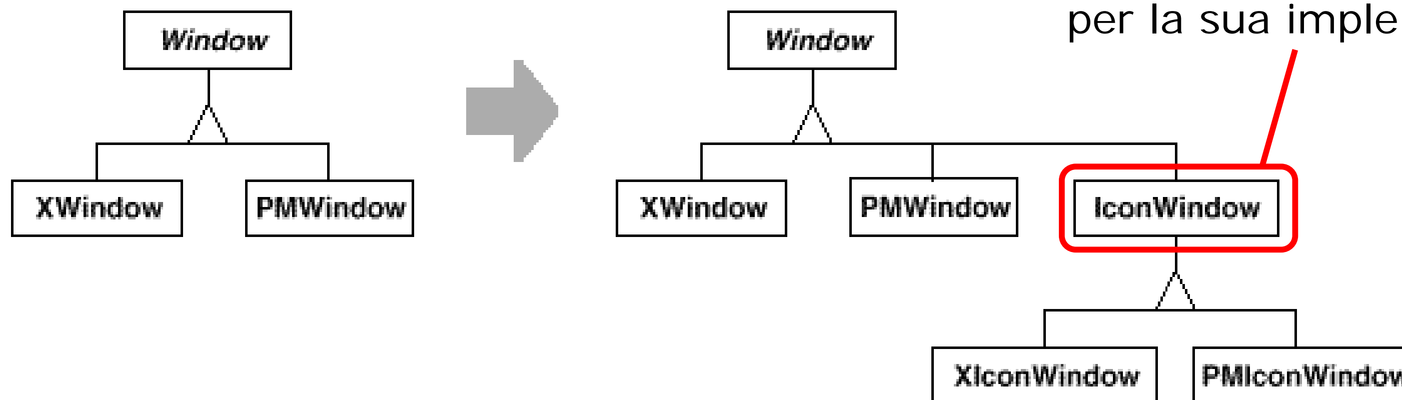
Bridge

- Problema: un'astrazione può avere una implementazione tra tante diverse fra loro
  - Tipica soluzione: **ereditarietà**
    Una classe astratta definisce l'interfaccia dell'astrazione e le sottoclassi realizzano ciascuna una singola implementazione

  Tale soluzione ha però delle limitazioni:

Per ogni tipologia di astrazione aggiuntiva devo avere 2 classi per la sua implementazione



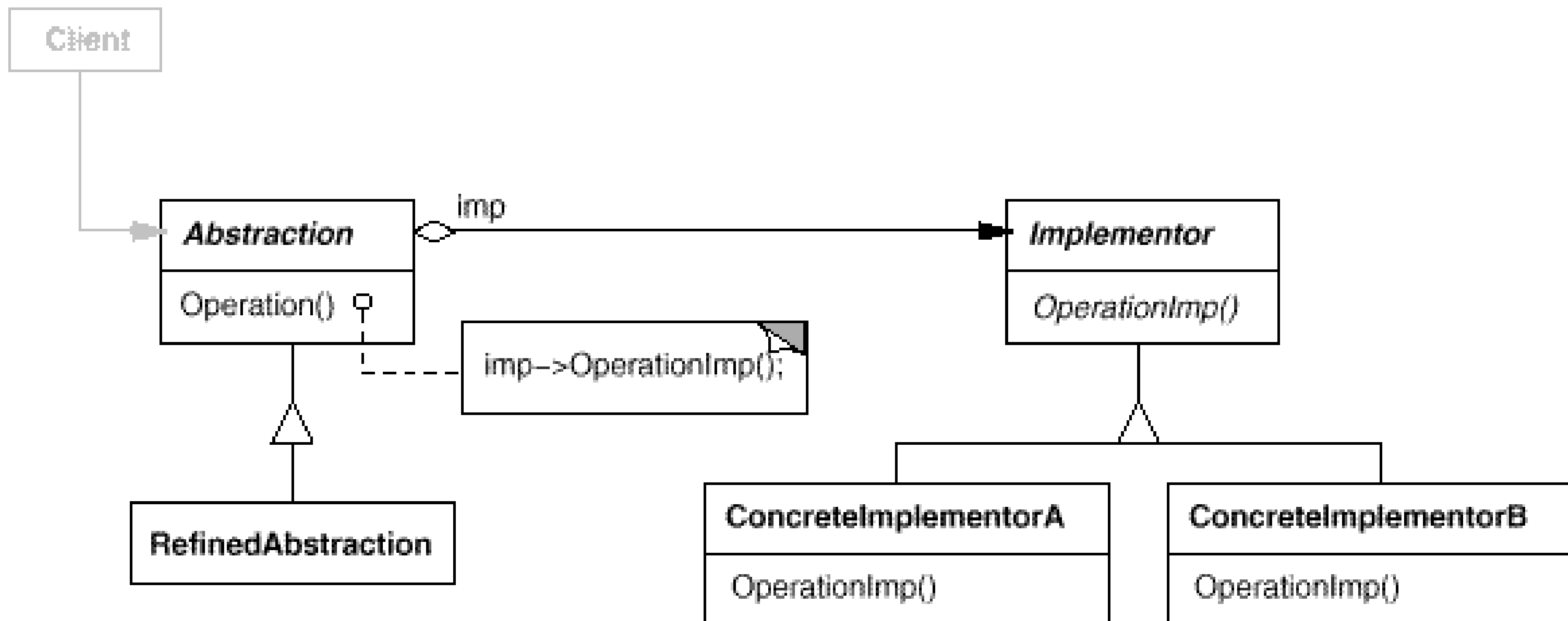
Nota: implicitamente astrazione e implementazione sono mescolate nella stessa gerarchia ...

Bridge (ii)

Limitazioni:

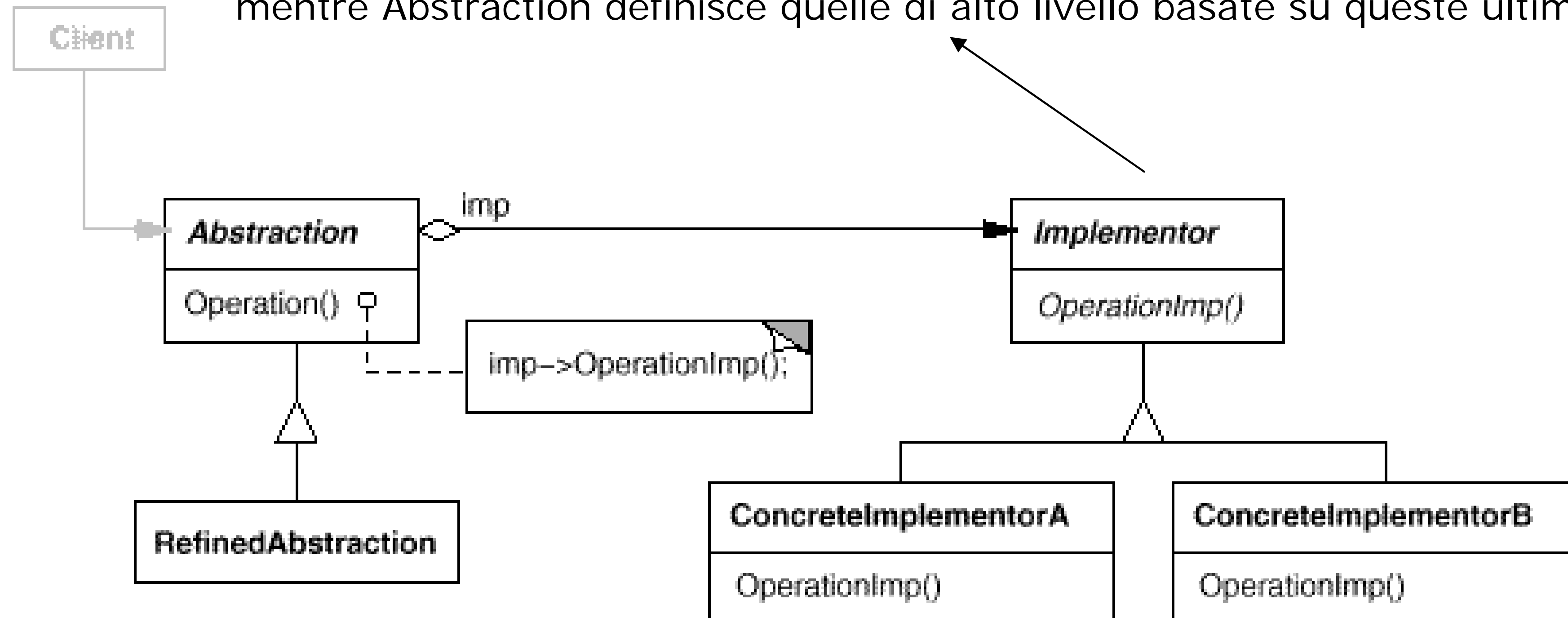
- Non è semplice estendere il codice per supportare una nuova tipologia di astrazione (v. slide precedente)
- Il codice del client è platform-dependent:
  ovunque un client richieda una astrazione, questi deve istanziare una classe concreta che ha una specifica implementazione
  (deve conoscere con quale implementazione si sta lavorando)
- un'astrazione è condizionata da un'implementazione in maniera *esplicita* e *statica*

Bridge (iii) (object/structural)

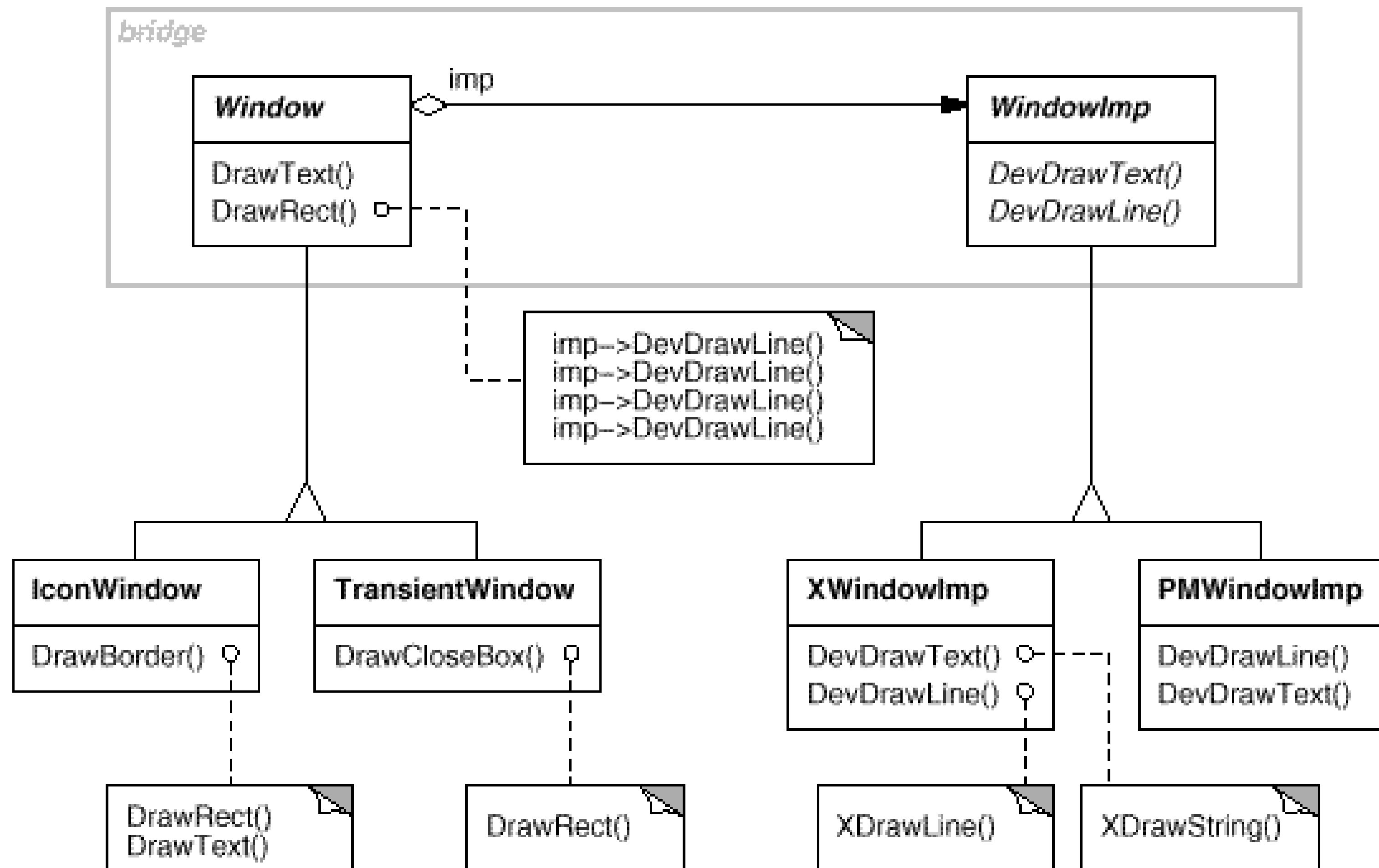


Bridge (iv)

Definisce l'interfaccia per le classi di implementazione. Questa interfaccia non deve necessariamente coincidere con l'interfaccia di *Abstraction*; anzi, tipicamente l'interfaccia di Implementor fornisce solo operazioni primitive, mentre Abstraction definisce quelle di alto livello basate su queste ultime.



Bridge – Esempio



Bridge – Codice

punta all'oggetto che implementa le funzioni di basso livello

```
class Window {
  public:
     Window(View* contents);

     // requests handled by window
     virtual void DrawContents();

     virtual void Open();
     virtual void Close();
     virtual void Iconify();
     virtual void Deiconify();

     // requests forwarded to implementation
     virtual void SetOrigin(const Point& at);
     virtual void SetExtent(const Point& extent);
     virtual void Raise();
     virtual void Lower();

     virtual void DrawLine(const Point&, const Point&);
     virtual void DrawRect(const Point&, const Point&);
     virtual void DrawPolygon(const Point[], int n);
     virtual void DrawText(const char*, const Point&);

  protected:
     WindowImp* GetWindowImp();
     View* GetView();

  private:
     WindowImp* _imp;
     View* _contents; // the window's contents
  } // end of class Window;


void Window::DrawRect (const Point& p1, const
     Point& p2) {
     WindowImp* imp = GetWindowImp();
     imp->DeviceRect(p1.X(), p1.Y(), p2.X(),
     p2.Y());
  }
```

in questo esempio le classi di tipo Window accedono all'implementazione attraverso una funzione di accesso

Bridge – Codice (ii)

ridefinizione di una funzione usando l'implementazione...

```
class IconWindow : public Window {
   public:
      // ...
      virtual void DrawContents();
   private:
      const char* _bitmapName;
   };

void IconWindow::DrawContents() {
      WindowImp* imp = GetWindowImp();
      if (imp != 0) {
         imp->DeviceBitmap(_bitmapName,
       0.0, 0.0);
      }
   }
```

```
class WindowImp {
   public:
      virtual void ImpTop() = 0;
      virtual void ImpBottom() = 0;
      virtual void ImpSetExtent(const Point&) = 0;
      virtual void ImpSetOrigin(const Point&) = 0;

      virtual void DeviceRect(Coord, Coord, Coord, Coord) = 0;
      virtual void DeviceText(const char*, Coord, Coord) = 0;
      virtual void DeviceBitmap(const char*, Coord, Coord) = 0;
      // lots more functions for drawing on windows...
   protected:
      WindowImp();
   };
```

Bridge – Codice (iii)

```
class XWindowImp : public WindowImp {
   public:
      // ...
virtual voidDeviceRect(Coord,Coord,Coord,Coord);
//...
private:
      //XWindow specific state
   };

void XWindowImp::DeviceRect(Coord x0, Coord
        y0, Coord x1, Coord y1 ) {
int x=round(min(x0,x1));
int y=round(min(x0,x1));
int w=round(abs(x0-x1));
int h=round(abs(y0-y1));
XDrawRectangle( /*....*/ , x,y,w,h);
//un rectangle è definito come l'angolo in basso a
//sinistra + width e height
//...
}
```

```
class PMWindowImp : public WindowImp {
   public:
      // ...
virtual voidDeviceRect(Coord,Coord,Coord,Coord);
//...
private:
      //PMWindow specific state
   };

void PMWindowImp::DeviceRect(Coord x0, Coord y0,
        Coord x1, Coord y1 ) {
Coord left=min(x0,x1);
Coord right=max(x0,x1);
Coord bottom=min(y0,y1);
Coord top=max(y0,y1);
//..costruzione dei 4 vertici -> PPOINTL point[4];
//..costruizione del path ->
GpiStrokePath(/*...*/);
//per es. in questo sistema grafico non esiste una
        //primitiva per disegnare rettangoli ma spezzate}
```

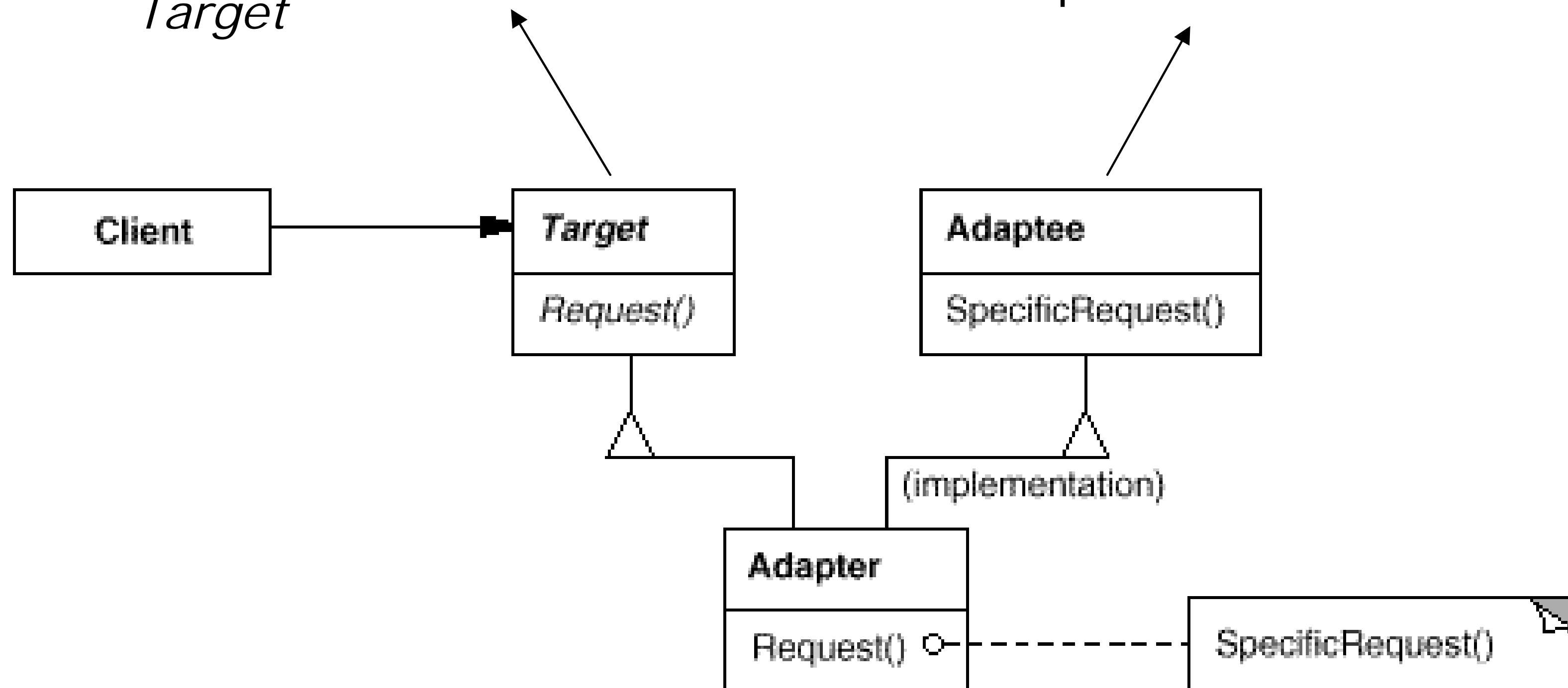
Adapter (class,object / structural)

- Converte l'interfaccia di una classe in un'altra interfaccia che i client si aspettano
- Adapter permette a due classi con interfacce incompatibili di cooperare

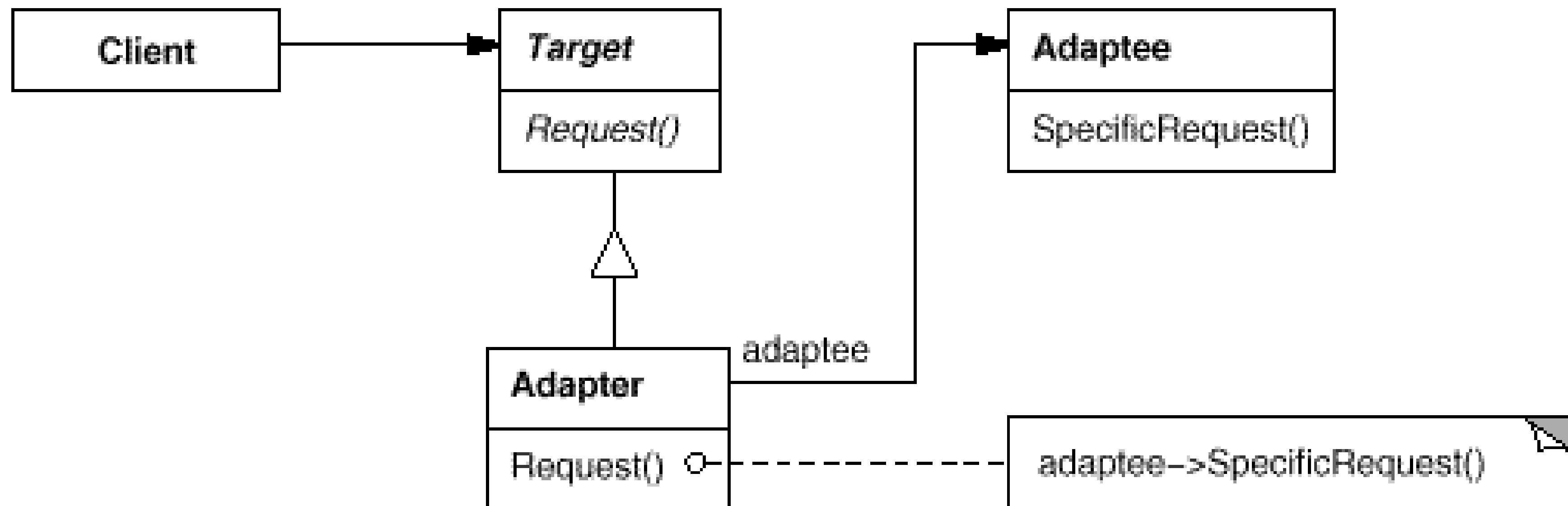
- Due diversi approcci:
  1. Basato su EREDITARIETA' MULTIPLA
  2. Basato su COMPOSIZIONE

- Spesso l'*Adapter* fornisce anche funzionalità che la classe adattata non è in grado di supportare

Adapter – Ereditarietà multipla



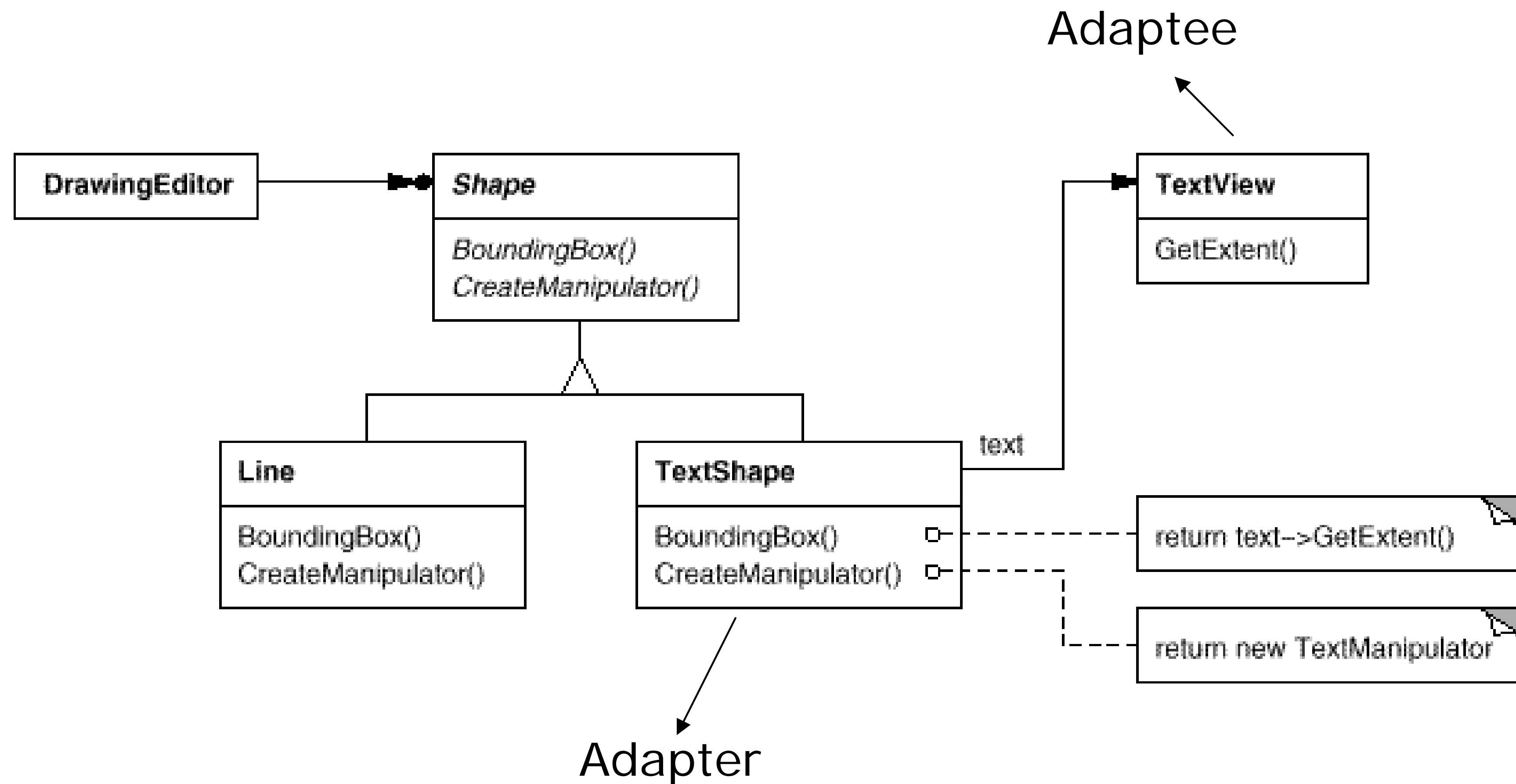
Adapter – Composizione



Adapter (ii)

- Approccio basato su **ereditarietà**:
  - L’interfaccia di Adaptee è adattata all’interfaccia Target appoggiandosi ad una classe Adaptee concreta
    - Di conseguenza, una classe Adapter non può essere utilizzata quando si vuole adattare una classe e tutte le sue sottoclassi
  - Consente alla classe Adapter di sovrascrivere parte del comportamento di Adaptee, visto che Adapter è una sottoclasse di Adaptee
  - Introduce soltanto un oggetto, e non occorrono ulteriori indirezioni per ottenere un riferimento all’oggetto adattato (design più semplice e intuitivo)
- Approccio basato su **composizione**:
  - Permette a un'unica classe Adapter di lavorare con Adaptee e tutte le sue sottoclassi
    - L’Adapter può anche aggiungere delle funzionalità a tutti gli Adaptee contemporaneamente
  - Introduce un oggetto ulteriore complicando il design

Adapter – Esempio



Manipulator: definisce oggetti per animare una "Shape"->drag and drop

Adapter – Codice

adaptee

```
class Shape{
public:
 Shape();
 virtual void BoundingBox(Point& bottomLeft, Point& topRigth) const;
virtual Manipulator* CreateManipulator()const;
}
```

```
class TextView{
public:
 TextView();
void GetOrigin(Coord&x, Coord&y)const;
void GetExtent(Coord&width,Coord&heigth)const;
virtual bool IsEmpty()const;
}
```

```
class TextShape:public Shape,private TextView{
public:
 TextShape();
 virtual void BoundingBox(Point& bottomLeft, Point& topRigth) const;
virtual Manipulator* CreateManipulator()const;
virtual bool IsEmpty()const;
}
```

eredita l'interfaccia di Shape e l'implementazione di TextView

adapter (di che tipo?)

Adapter – Codice

```
void TextShape::BoundingBox(Point& bottomLeft, Point& topRight)const{
Coord bottom,left,width,heigth;
GetOrigin(bottom,left);
GetExtent(width,height);
bottomLeft=Point(bottom,left);
topRigth=Point(bottom+heigth,left+width);
}
bool TextShape::IsEmpty()const{
return TextView::IsEmpty();
}
Manipulator* TextShape::CreateManipulator()const{
return new TextManipulator(this);
}
```

→ implementazione dell'interfaccia di Shape utilizzando l'interfaccia di TextView

→ inoltro diretto di una richiesta all'adaptee

→ implementazione di un metodo dell'interfaccia di Shape che non usa nessuna delle funzionalità di TextView

Adapter – Codice: object comp.

```
class TextShape :: public Shape {
public:
  TextShape(TextView*);
  virtual void BoundingBox(Point& bottomLeft,Point& topRigth) const;
  virtual bool IsEmpty ()const;
  virtual Manipulator* CreateManipulator()const;
private:
  TextView* _text;
}

TextShape::TextShape(TextView* t){
_text=t;
}
```

maggiore overhead

maggiore flessibilità

Adapter – Codice: object comp.

```
void TextShape::BoundingBox(Point& bottomLeft, Point& topRight)const{
Coord bottom,left,width,heigth;
_text->GetOrigin(bottom,left);
_text->GetExtent(width,height);
bottomLeft=Point(bottom,left);
topRigth=Point(bottom+heigth,left+width);
}
```

→ implementazione dell'interfaccia di Shape utilizzando l'interfaccia di TextView

```
bool TextShape::IsEmpty()const{
return _text->IsEmpty();
}
```

→ inoltro diretto di una richiesta all'adaptee

```
Manipulator* TextShape::CreateManipulator()const{
return new TextManipulator(this);
}
```

→ in questo caso non è cambiato nulla: il metodo non utilizza l'interfaccia di TextView