Decorator

Motivace / Klasický návrh 1

- **Návrh GUI toolkitu**
- **`TextView` komponenta**
  - potřeba přidat okraje, skrolování textu, stín...
    - různé kombinace
- **Jedna velká třída, která všechno umí**
  - Komplikovaný návrh, špatná rozšiřitelnost

```
class TextView {
    private:
        int scroll_position;
        int border_width;

    public:
        void Draw() { ... }
        void DrawWithBorder() { ... }
        void DrawAndScroll() { ... )
        void DrawWithBorderAndScroll() { ... }
        ...
}
```

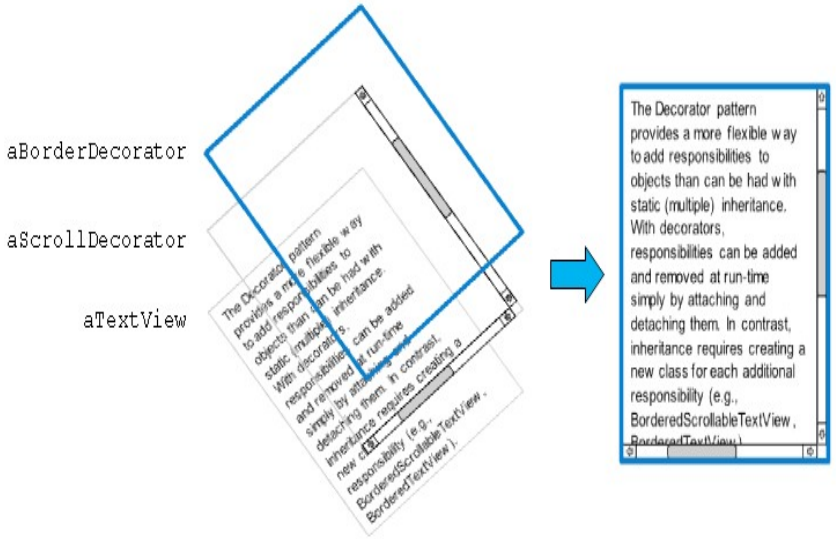
Motivace / Klasický návrh 2

- **Dědičnost: bázová třída `TextView` a odvozené třídy `ScrollableTextView`, `TextViewWithBorder`, `ScrollableTextViewWithBorder`...**
  - velké množství tříd (*exponenciálně*)

```
class TextView {
    public:
        void Draw() { ... }
};

class ScrollableTextView : public TextView {
    public:
        void Draw() { ... }
};

class TextViewWithBorder : public TextView {
    public:
        void Draw() { ... }
};

class ScrollableTextViewWithBorder : public TextView {
    public:
        void Draw() { ... }
};
```

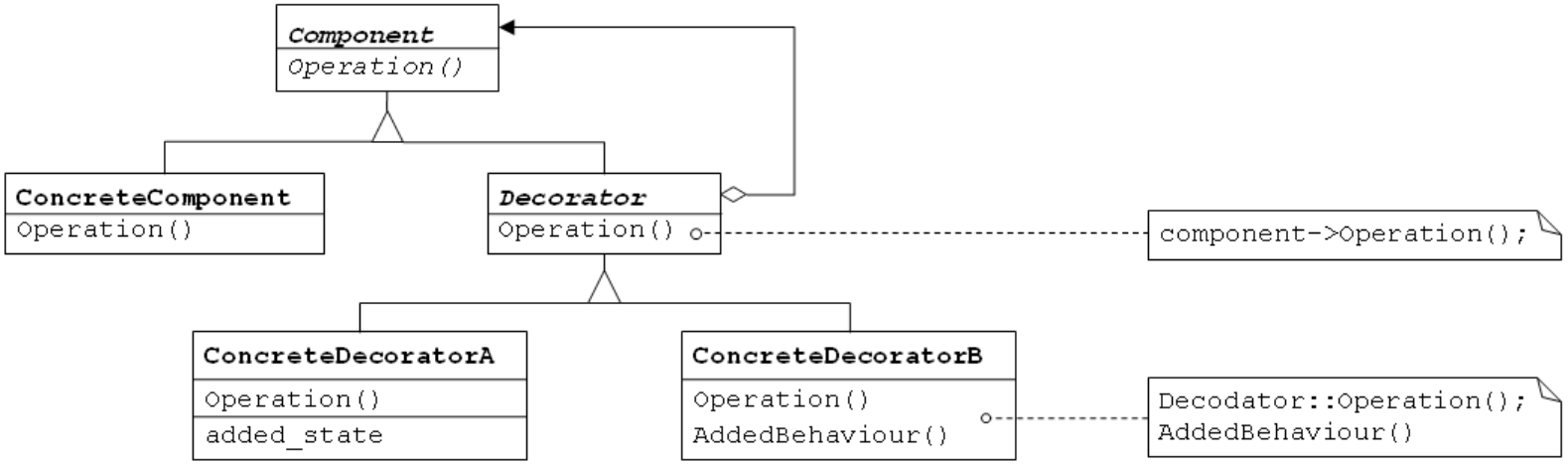
Motivace / Decorator

- **Decorator**
  - Malé objekty přidávají dekorace
  - `TextView` neví o dekoracích
  - Dekorace jsou navzájem nezávislé
  - Dekorace lze kombinovat
- **Známý jako**
  - Wrapper
- **Kategorie**
  - strukturální (*structural*)
- **Účel**
  - rozšíření objektu o nové vlastnosti
  - rozšiřuje konkrétní objekt, ne třídu
  - dynamický (dědičnost je statická)
  - transparentní: pro uživatele i pro rozšiřovaný objekt



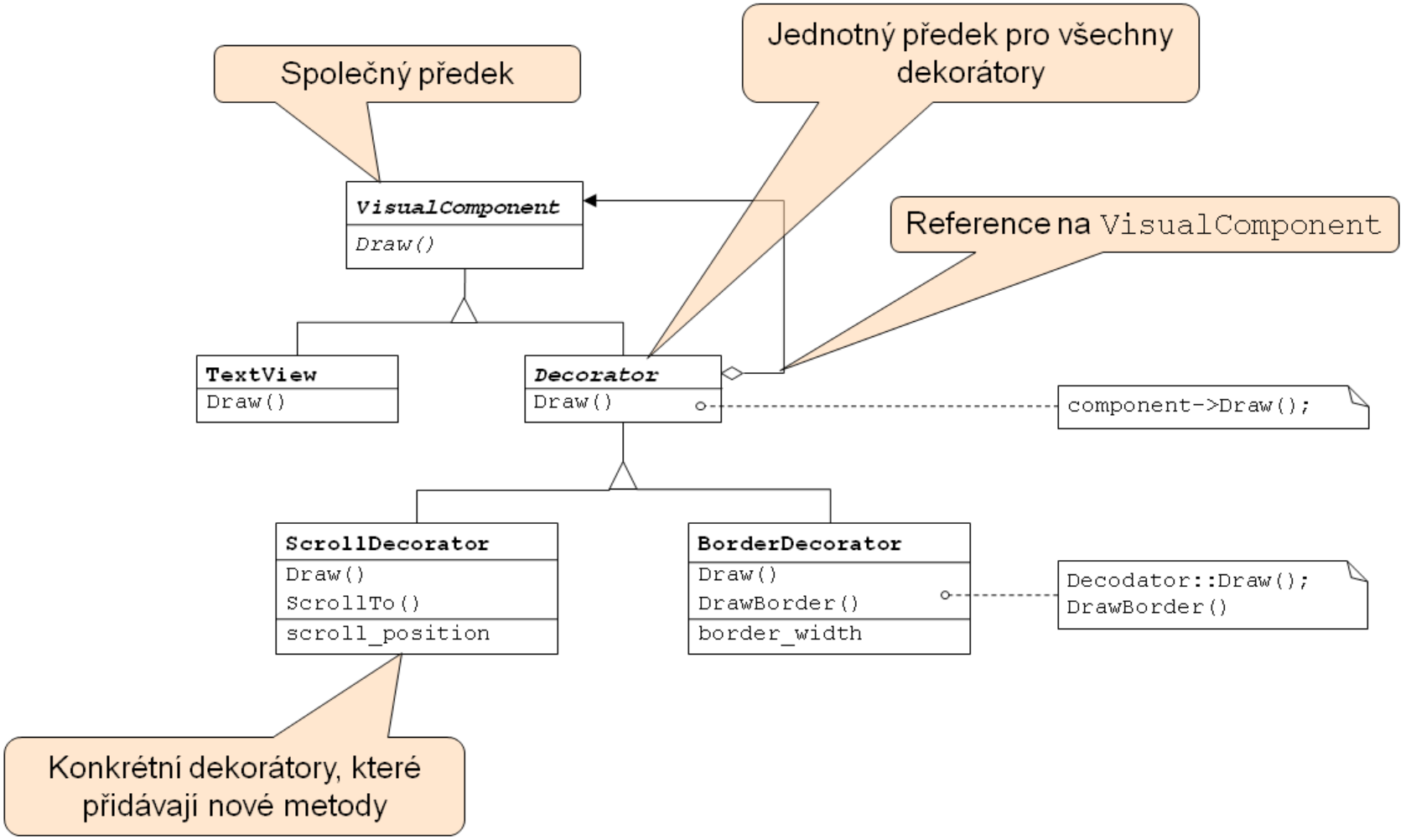
Struktura

- **Struktura**



- **Účastníci**
  - `Component` (`VisualComponent`)
    - Definuje rozhraní pro objekty, ku kterým se dynamicky přidávají další funkce
  - `ConcreteComponent` (`TextView`)
    - konkrétní rozšiřovaný objekt
  - `Decorator`
    - předek všech dekorujících tříd, definuje jejich rozhraní
    - obsahuje refenci na objekt, který rozšiřuje
    - všechny voláni deleguje na zapouzdřený objekt
  - `ConcreteDecorator` (`BorderDecorator`, `ScrollDecorator`)
    - Přidává konkrétní nové funkce

Struktura: motivační příklad



Implementace: motivační příklad

```
class VisualComponent {
   public: virtual void Draw() = 0;
};

class TextView : public VisualComponent {
   public: void Draw() {}
};

class Decorator : public VisualComponent {
        VisualComponent * vc;
   public:
         Decorator( VisualComponent *v ) : vc(v) {}
         void Draw() { vc->draw(); }
};

class ScrollDecorator : public Decorator {
         int scroll_position;
   public:
         ScrollDecorator(VisualComponent *v) : Decorator(v) {}
         void ScrollTo( int pos ) { scroll_position = pos; }
         void Draw() {
                 Decorator::Draw(); // Delegate to base class
                 cout << "Scroll Decorator" << endl;
     }
};

class BorderDecorator : public Decoratror { ... };
```

Použití

- **potřeba přidání nových funkcí**
  - dynamicky, transparentně
- **potřeba funkce odebrat**
  - nebo omezit
- **když dědičnost nestačí**
  - nemáme kód předka nebo je předek špatně navržen
  - příliš mnoho tříd

Výhody/nevýhody

- **Výhody**
  - dynamičnost
  - „chaining“
    - není problém přidat více dekorátorů pro jeden objekt
  - „pay-as-you-go“
  - není potřeba předvídat úplně všechny potřeby klienta
    - přidání nového dekorátoru je jednoduché

- **Nevýhody**
  - dekorátor není dekorovaný objekt
    - nelze volat „rozšířené“ metody z dekorovaného objektu
  - mnoho podobných (malých) objektů
    - horší orientace v kódu
  - rychlost
    - potřeba virtualních metod
    - postupné delegace volání

Implementace

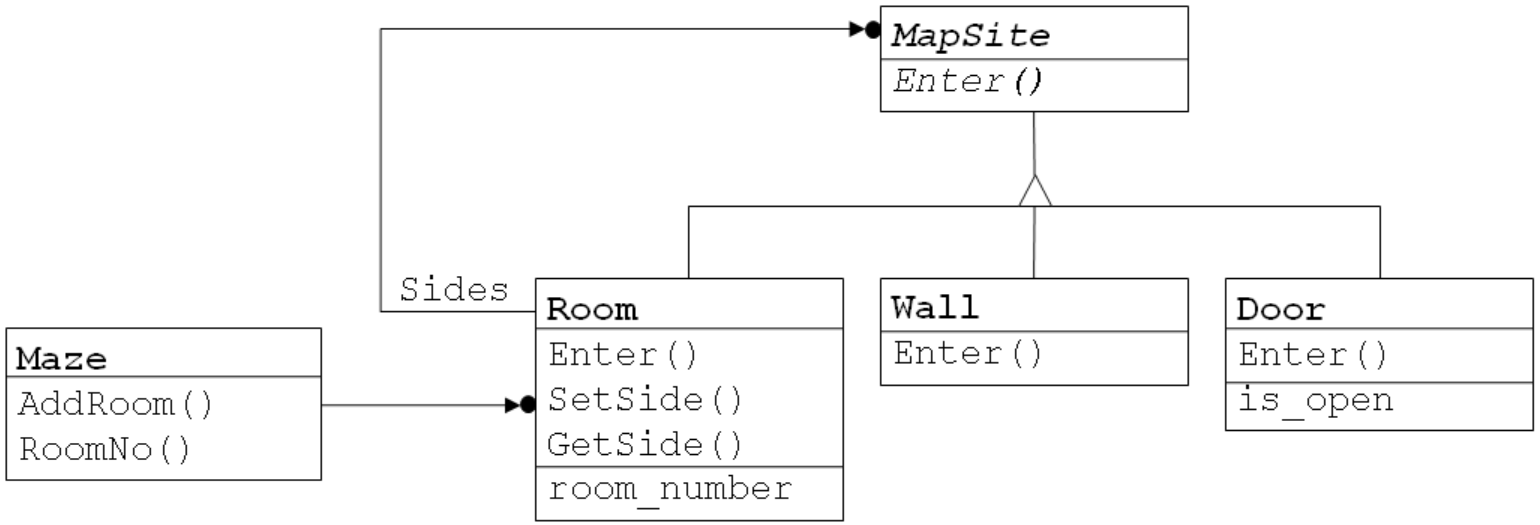
- **Implementace**
  - dekorátor i dekorovaný objekt musí mít společného předka
  - bázové třídy (pro `Component` i `Decorator`) – light-weight
    - interface, abstraktní třída
    - datová reprezentace a funkcionalita v potomcích
  - abstraktní předek `Decorator` není potřeba máme-li jen jeden dekorátor
  - dekorace objektu
    - typicky pomocí konstruktoru
    - jednoduché (přehledné) řetězení

```
Component c = new ConcreteDecoratorA(
    new ConcreteDecoratorB(
        new ConcreteComponent(...)
    )
);
```

```
VisualComponent * vc = new BorderDecorator(
    new ScrollDecorator(
        new TextView()
    )
);
vc->Draw();
```

Príklad: bludiště

- **Vytvoření bludiště**
  - seznam místností a jejich sousedů (zdi, dveře, jiná místnost)



- **Rozšíření možností objektů ve hře**
  - `BombedRoom`
  - `WeakDoor`

Bludiště: implementace

```
Maze *MazeGame::CreateMaze() {
  Maze *maze = new Maze;
  Room *room_a = new BombedRoom(
    new Room(1));
  Room *room_b = new BombedRoom(
    new Room(2));
  Door *door = new WeakDoor(
    new Door(room_a, room_b));

  maze->AddRoom(room_a);
  maze->AddRoom(room_b);

  room_a->SetSide(North, new Wall);
  room_a->SetSide(East, door);
  room_a->SetSide(South, new Wall);
  room_a->SetSide(West, new Wall);

  room_b->SetSide(North, new Wall);
  room_b->SetSide(East, new Wall);
  room_b->SetSide(South, new Wall);
  room_b->SetSide(West, door);

  return maze;
}
```

```
class BombedRoom : public Room {
  public:
    BombedRoom(Room *room)
      { _room = room; _has_bomb = true; }
    virtual void Enter()
      { _room->Enter(); }
    /* ... */
  private:
    Room  *_room;
    bool  _has_bomb;
};
```

```
class WeakDoor : public Door {
  public:
    WeakDoor(Door *door)
      { _door = door; }
    virtual void Enter()
      { _door->Enter(); }
    /* ... */
    virtual void Kick()
      { _door->SetOpen(true); }
  private:
    Door *_door;
};
```

Známe použití a související návrhové vzory

- **Známé použití dekorátorů**
  - grafické toolkity (XWindow, Swing)
  - smart-pointer (boost, loki, ...)
  - Java I/O
    - abstraktní třídy `InputStream`, `OutputStream`
    - odvozené dekorátory `FilterInputStream`, `FilterOutputStream`
    - z nich odvozené např. `FileInputStream`, `StringBufferInputStream`, ...
    - množství tříd: kompilované pro začátečníky
      - bez dekorátorů: (mnohem) více tříd anebo méně vlastností

- **Související vzory**
  - Adapter
    - mění rozhraní objektu
  - Strategy
    - mění chování objektu
  - Composite
    - agregace objektů, dekorátor objekt rozšiřuje