Паттерны Iterator, Command

Есилевич Александр

20 января 2012 г.

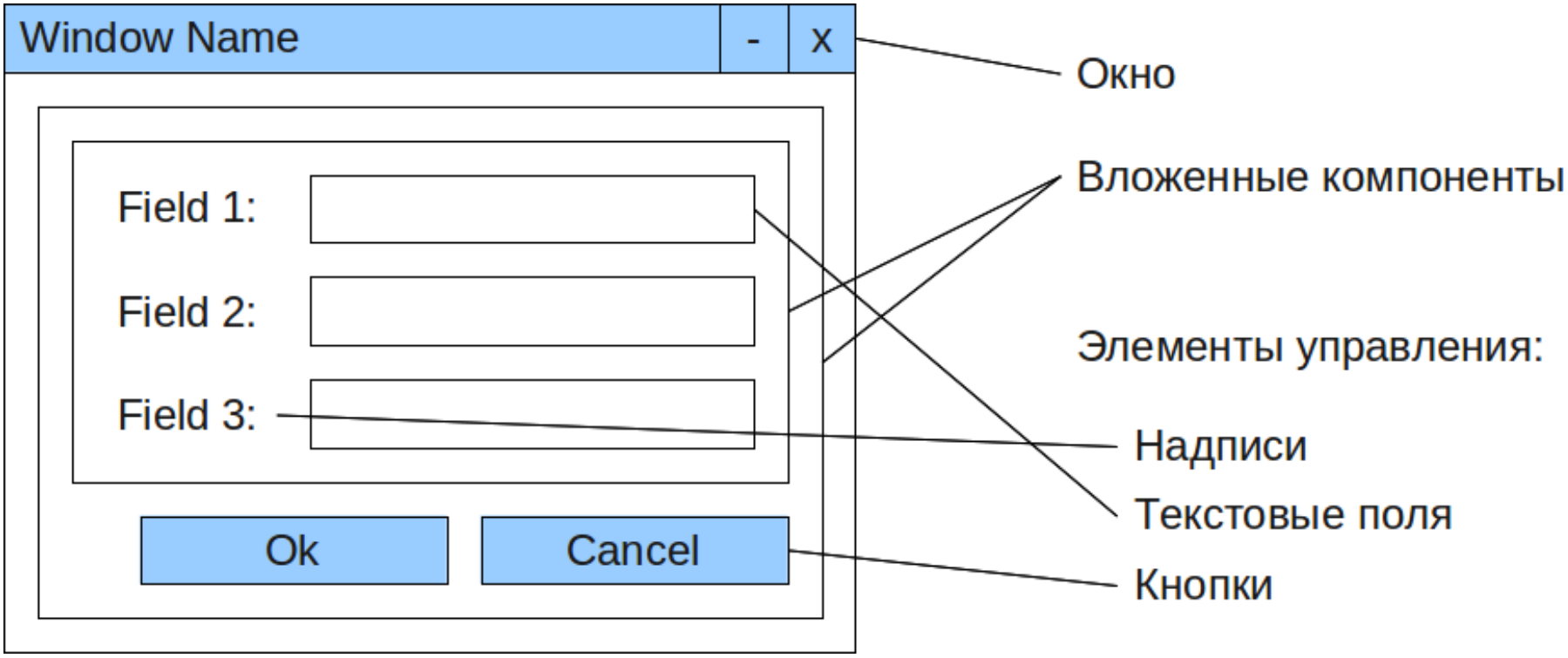
Векторный графический редактор

- Манипулирование графическими примитивами, а не точками (пикселями)
- Интерактивность — добавление и редактирование графических примитивов мышью
- Легкость расширения и портирования на разные платформы

Средства, предоставляемые графической библиотекой

- `class SystemComponent` — базовый элемент графического интерфейса
- `class SystemWindow` — окно приложения
- `class SystemCanvas` — «холст», объект для отображения содержимого компонентов
- Набор примитивных графических элементов управления (надписи, кнопки, текстовые поля, списки)

Структура графического интерфейса



Класс SystemComponent

```
public abstract class SystemComponent {

    public abstract void paint(SystemCanvas canvas);

    public void handleMousePress(int x, int y) {}
    public void handleMouseRelease(int x, int y) {}
    public void handleMouseMove(int x, int y) {}
    public void handleMouseClick(int x, int y) {}
    public void handleKeyPress(char c) {}

    // Other mouse/keyboard events
    // ...
}
```

Класс SystemCanvas

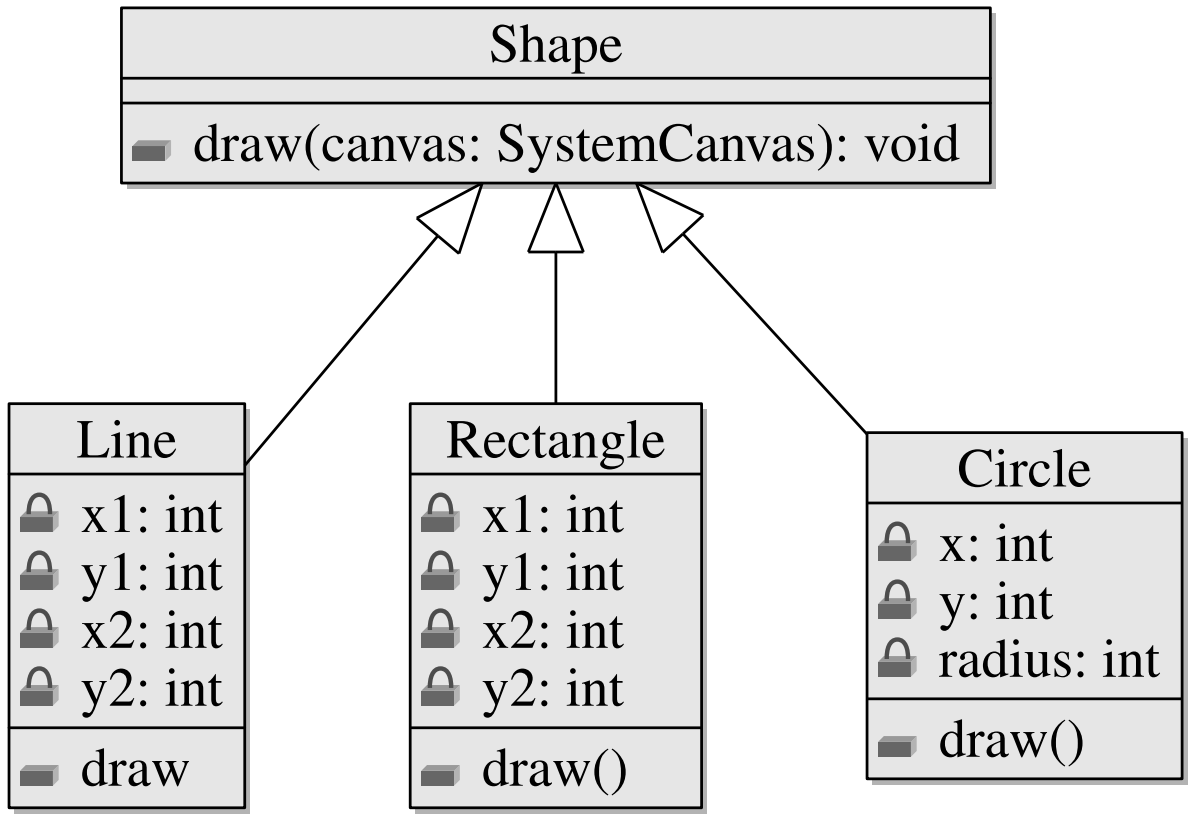
```
public class SystemCanvas {

    public void drawLine(int x1, int y1,
                         int x2, int y2);

    public void drawOval(int x, int y,
                         int width, int height);

    // Other graphics primitives
}
```

Java AWT - Abstract Window Toolkit

- `SystemWindow` = `java.awt.Frame`
- `SystemComponent` = `java.awt.Component`
- `SystemCanvas` = `java.awt.Graphics`

С чего начать?

Диаграмма классов геометрических фигур



Класс для представления фигуры

```
public abstract class Shape {
    public abstract void draw(SystemCanvas canvas);
}
```

Линия

```
public class Line extends Shape {

    public Line(int px1, int py1, int px2, int py2) {
        x1 = px1;
        y1 = py1;
        x2 = px2;
        y2 = py2;
    }

    public void draw(SystemCanvas canvas) {
        canvas.drawLine(x1, y1, x2, y2);
    }

    private int x1;
    private int y1;
    private int x2;
    private int y2;
}
```

Прямоугольник

```
public class Rectangle extends Shape {
    public Rectangle(int px1, int py1,
                     int px2, int py2) {
        x1 = px1;
        y1 = py1;
        x2 = px2;
        y2 = py2;
    }

    public void draw(SystemCanvas canvas) {
        canvas.drawLine(x1, y1, x2, y1);
        canvas.drawLine(x2, y1, x2, y2);
        canvas.drawLine(x2, y2, x1, y2);
        canvas.drawLine(x1, y2, x1, y1);
    }

    private int x1;
    private int y1;
    private int x2;
    private int y2;
}
```

Окружность

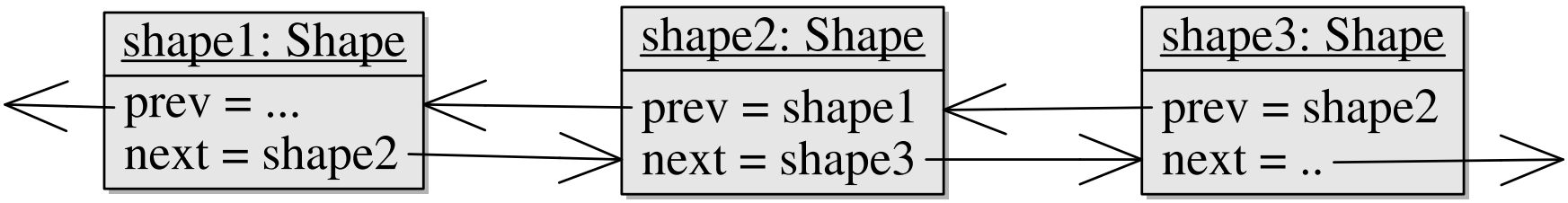
```
public class Circle extends Shape {

    public Circle(int px, int py, int pradius) {
        x = px;
        y = py;
        radius = pradius;
    }

    public void draw(SystemCanvas canvas) {
        canvas.drawOval(x - radius, y - radius,
                        x + radius, y + radius);
    }

    private int x;
    private int y;
    private int radius;
}
```

Пример использования класса Shape

```
public class MyComponent extends SystemComponent {

    public void paint(SystemCanvas canvas) {
        Shape line = new Line(10, 10, 20, 20);
        line.draw(canvas);

        Shape circle = new Circle(100, 100, 40);
        circle.draw(canvas);

        Rectangle rect = new Rectangle(30, 30, 50, 50)
        rect.draw(canvas);
    }
}
```

Как хранить коллекцию фигур?

Двусвязный список



Класс Scene

```
public class Scene extends SystemComponent {
    public void paint(SystemCanvas canvas) {
        Shape shape = first;
        while(shape != null) {
            shape.draw(graphics);
            shape = shape.next;
        }
    }

    public void addShape(Shape shape);
    public void addShapeBefore(Shape s, Shape before);
    public void removeShape(Shape s);

    private Shape first;
    private Shape last;
}
```

Класс Scene

```
public void addShape(Shape shape) {
    if(first == null) {
        // no items in the list
        first = shape;
        last = shape;
    } else {
        last.next = shape;
        shape.prev = last;
        last = shape;
    }
}
```

Класс Scene

```
public void addShapeBefore(Shape s, Shape before) {

    Shape shape = first;
    while(shape != null) {
        if(shape == before) {
            s.next = shape;

            if(shape.prev != null) {
                shape.prev.next = s;
                s.prev = shape.prev;
            } else {
                first = s;
            }

            shape.prev = s;
            break;
        }

        shape = shape.next;
    }
}
```

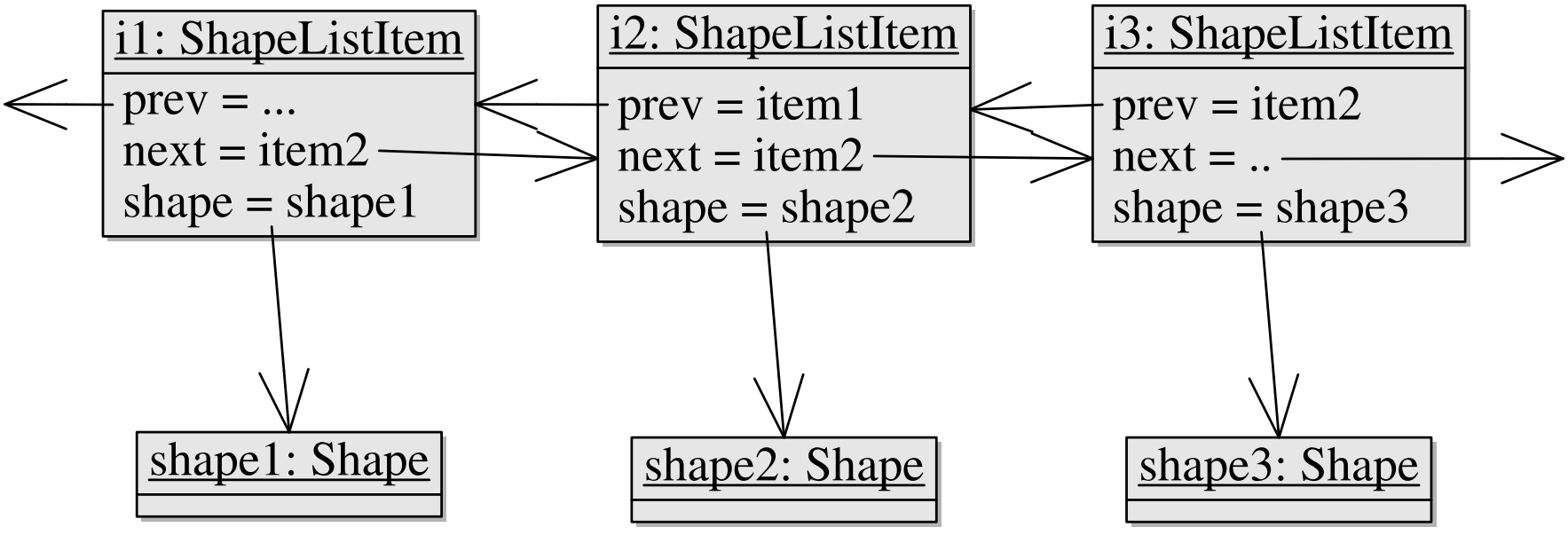
Класс Scene

```
public void removeShape(Shape s) {
    Shape shape = first;
    while(shape != null) {
        if(shape == s) {
            if(shape == first && shape == last) {
                first = null;
                last = null;
            } else if(shape == first) {
                first = shape.next;
                shape.next.prev = null;
            } else if(shape == last) {
                last = shape.prev;
                shape.prev.next = null;
            } else {
                shape.prev.next = shape.next;
                shape.next.prev = shape.prev;
            }
        }
        shape = shape.next;
    }
}
```

Недостатки

- Класс `Shape` предполагает наличие двусвязного списка, т. к. имеет поля `prev` и `next`
- Открытые (`public`) поля `prev`, `next`
- Нельзя переиспользовать операции работы со списком фигур, т. к. они привязаны к классу `Scene`
- Класс `Scene` зависит от внутренней структуры списка

Двусвязный список + ShapeListItem



Класс ShapeListItem

```
public class ShapeListItem {

    public ShapeListItem(Shape s);
    public Shape getShape();
    public void setShape(Shape s);

    ShapeListItem getNext();
    void setNext(ShapeListItem item);

    ShapeListItem getPrev();
    void setPrev(ShapeListItem item);

    private Shape shape;
    private ShapeListItem prev;
    private ShapeListItem next;
}
```

Класс Scene - модификация 1

```
public void addShape(Shape shape) {
    ShapeListItem item = new ShapeListItem(shape);
    // add item to the list
}
```

Класс Scene - модификация 1

```
public void addShapeBefore(Shape s, Shape before) {
    ShapeListItem newItem = new ShapeListItem(s);
    ShapeListItem item = first;
    while(item != null) {
        if(item.getShape() == before) {
            // add item to the list
        }
        item = item.getNext();
    }
}
```

Класс Scene - модификация 1

```
public void removeShape(Shape s) {
    ShapeListItem item = first;
    while(item != null) {
        if(item.getShape() == s) {
            // remove item from the list
        }
        item = item.getNext();
    }
}
```

Класс ShapeList

```
public class ShapeList {
    public void add(Shape shape);
    public void addBefore(Shape s, ShapeListItem before);
    public void remove(ShapeListItem item);

    public ShapeListItem getFirst();

    private ShapeListItem first;
    private ShapeListItem last;
}
```

Класс Scene - модификация 2

```
public void addShape(Shape shape) {
    shapes.add(shape);
}
```

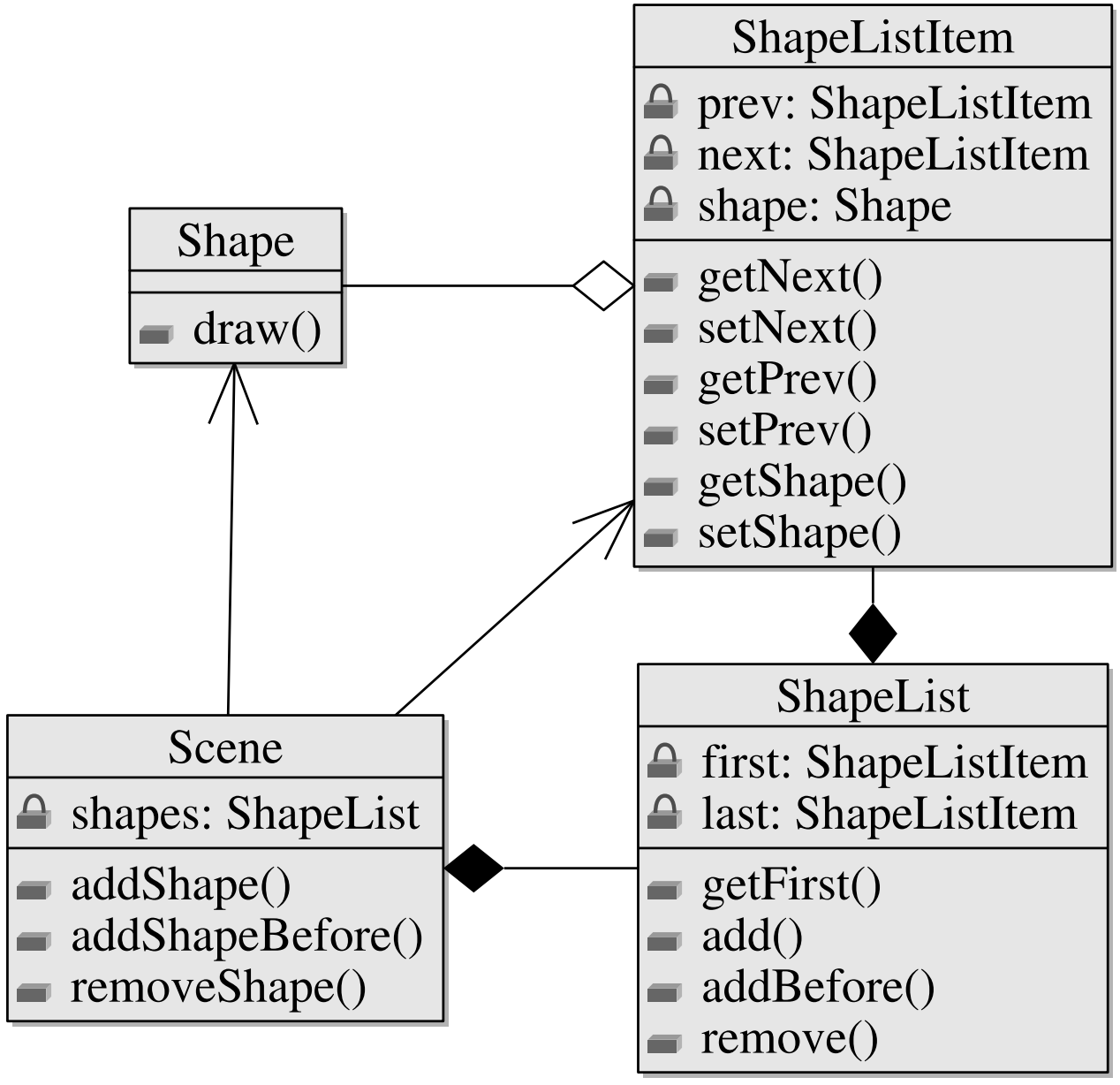
Класс Scene - модификация 2

```
public void addShapeBefore(Shape s, Shape before) {
    ShapeListItem item = shapes.getFirst();
    while(item != null) {
        if(item.getShape() == before) {
            shapes.addBefore(s, item);
            break;
        }

        item = item.getNext();
    }
}
```

Класс Scene - модификация 2

```
public void removeShape(Shape s) {
    ShapeListItem item = shapes.getFirst();
    while(item != null) {
        if(item.getShape() == before) {
            shapes.addBefore(s, item);
            break;
        }

        item = item.getNext();
    }
}
```

Диаграмма классов



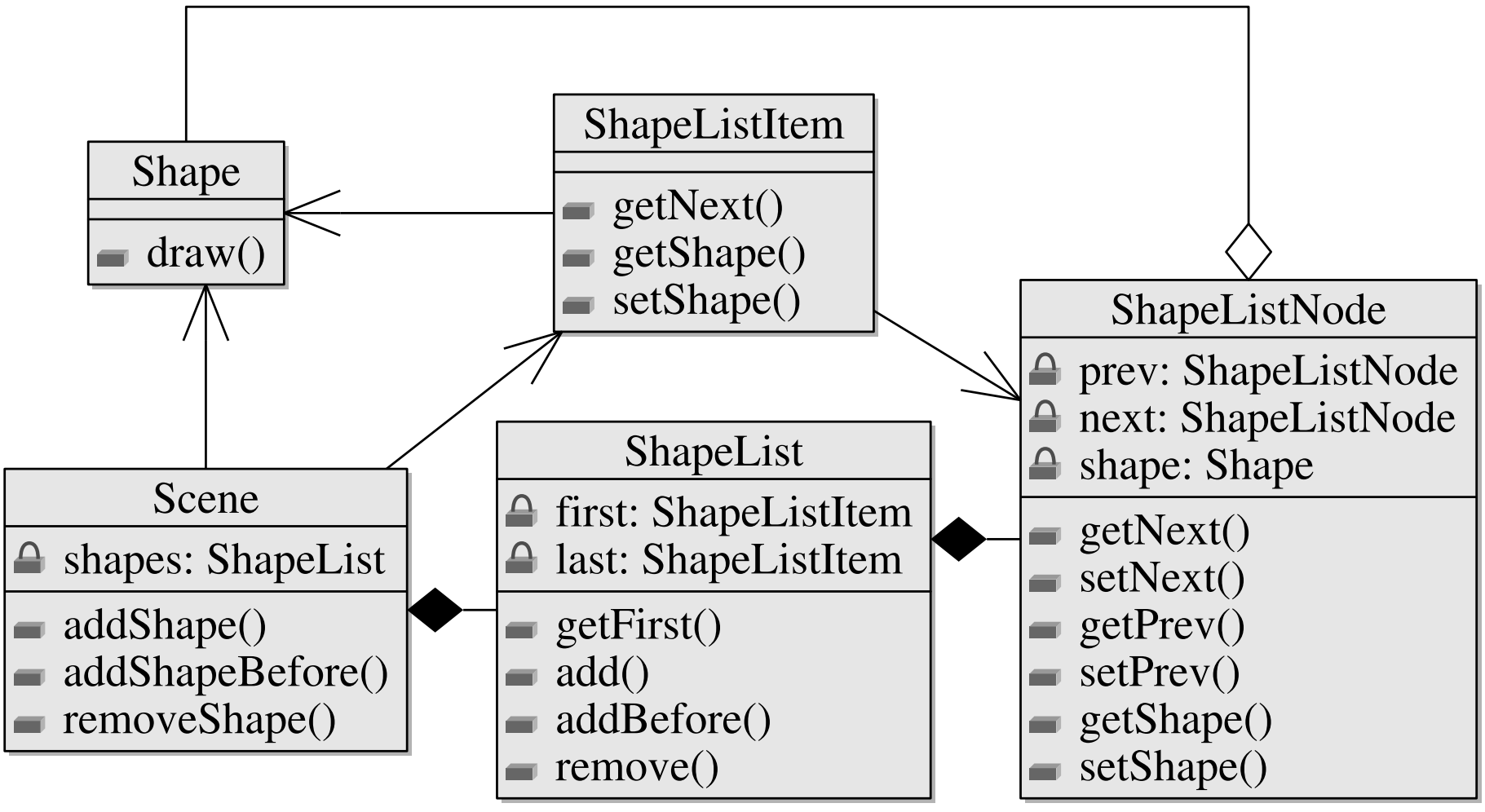
Внешний интерфейс класса ShapeListItem

```
public class ShapeListItem {
    public Shape getShape();
    public void setShape(Shape s);

    ShapeListItem getNext();
}
```

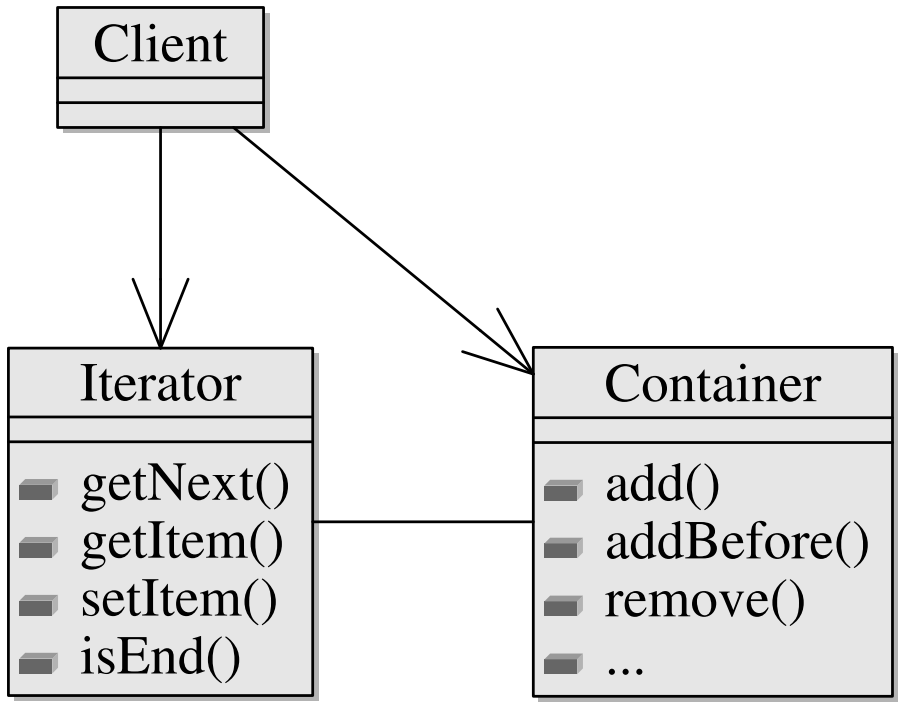
Класс ShapeListItem + ShapeListNode

```
public class ShapeListItem {
    public ShapeListItem(PrivateShapeListItem i) {
        item = i;
    }
    public Shape getShape() {
        return item.getShape();
    }
    public ShapeListItem getNext() {
        if(item.getNext() == null)
            return null;

        return new ShapeListItem(item.getNext());
    }
    // for internal use only
    public PrivateShapeListItem getPrivateItem() {
        return item;
    }
    private ShapeListNode node;
}
```

Диаграмма классов с ShapeListNode



Паттерн Iterator



Массив вместо списка

```
public class ShapeListIterator {
    public Shape getShape() {
        assert(!isEnd());
        return array[index];
    }
    public ShapeListIterator getNext() {
        assert(!isEnd());
        return new ShapeListIterator(array, index + 1);
    }
    public boolean isEnd() {
        return index >= array.length;
    }

    Shape [] array;
    private int index;
}
```

Массив вместо списка

```
public class ShapeList {

    public ShapeListIterator getFirst() {
        return new ShapeListIterator(array, 0);
    }

    private Shape [] array;
};
```

Виды итераторов

- Создаётся контейнером или клиентом?
- Для доступа к элементу нужно вызвать метод итератора или контейнера?
- Может ли итератор изменять элементы?
- Для изменения элемента нужно вызвать метод итератора или контейнера?
- Отделена ли итерация от доступа к элементу?

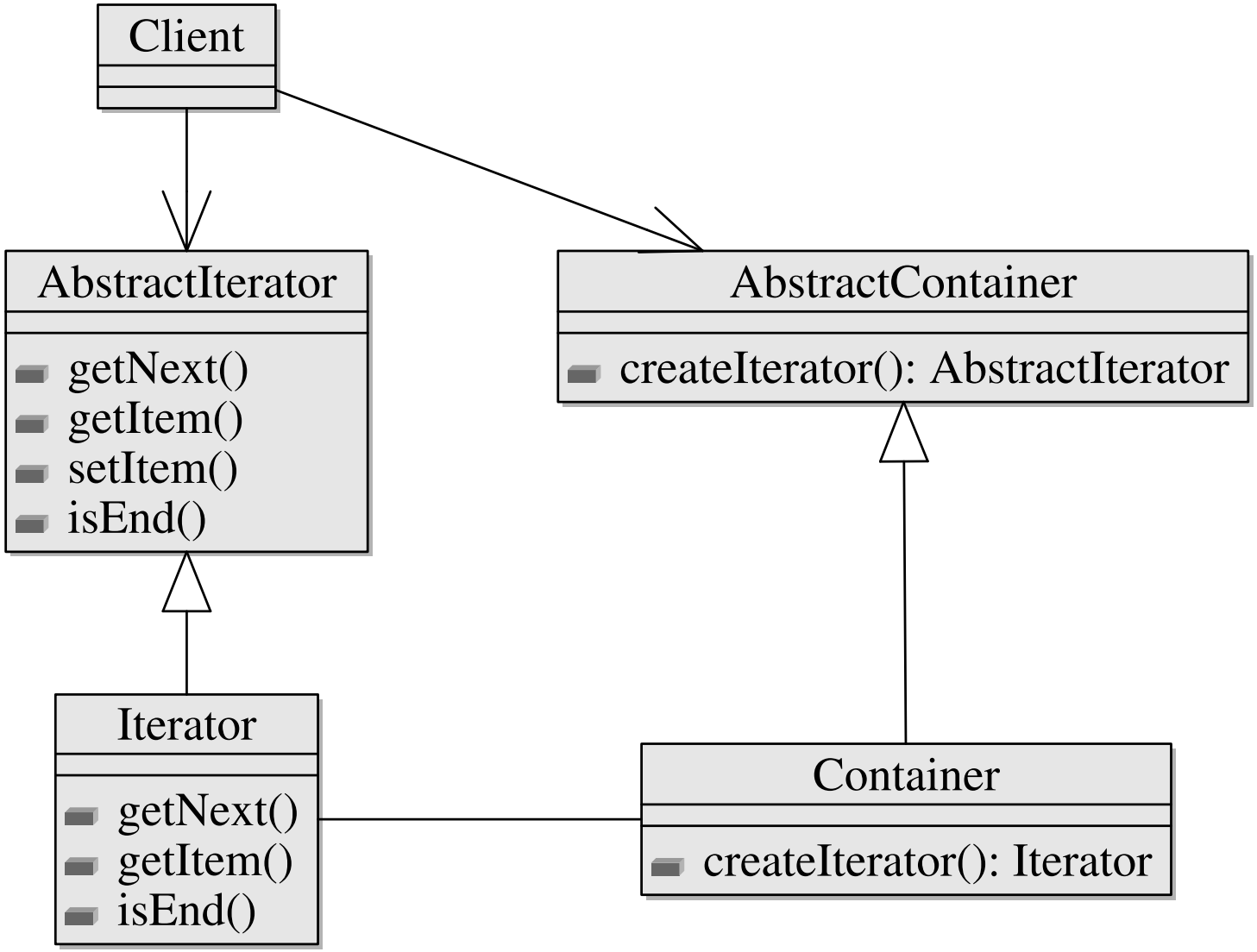
C++ Iterator

```
public class ShapeIterator {
    public ShapeListIterator getNext();
    public Shape getShape();
};
```

Java Iterator

```
public class ShapeIterator {
    public Shape getNext();
    public bool isEnd();
};
```

Абстрактный итератор



Результаты

- Клиент не зависит от внутренней структуры контейнера
- Клиент не зависит от алгоритма обхода элементов в контейнере
- Клиент поддерживает несколько алгоритмов обхода
- Одновременно для контейнера может быть активно несколько обходов

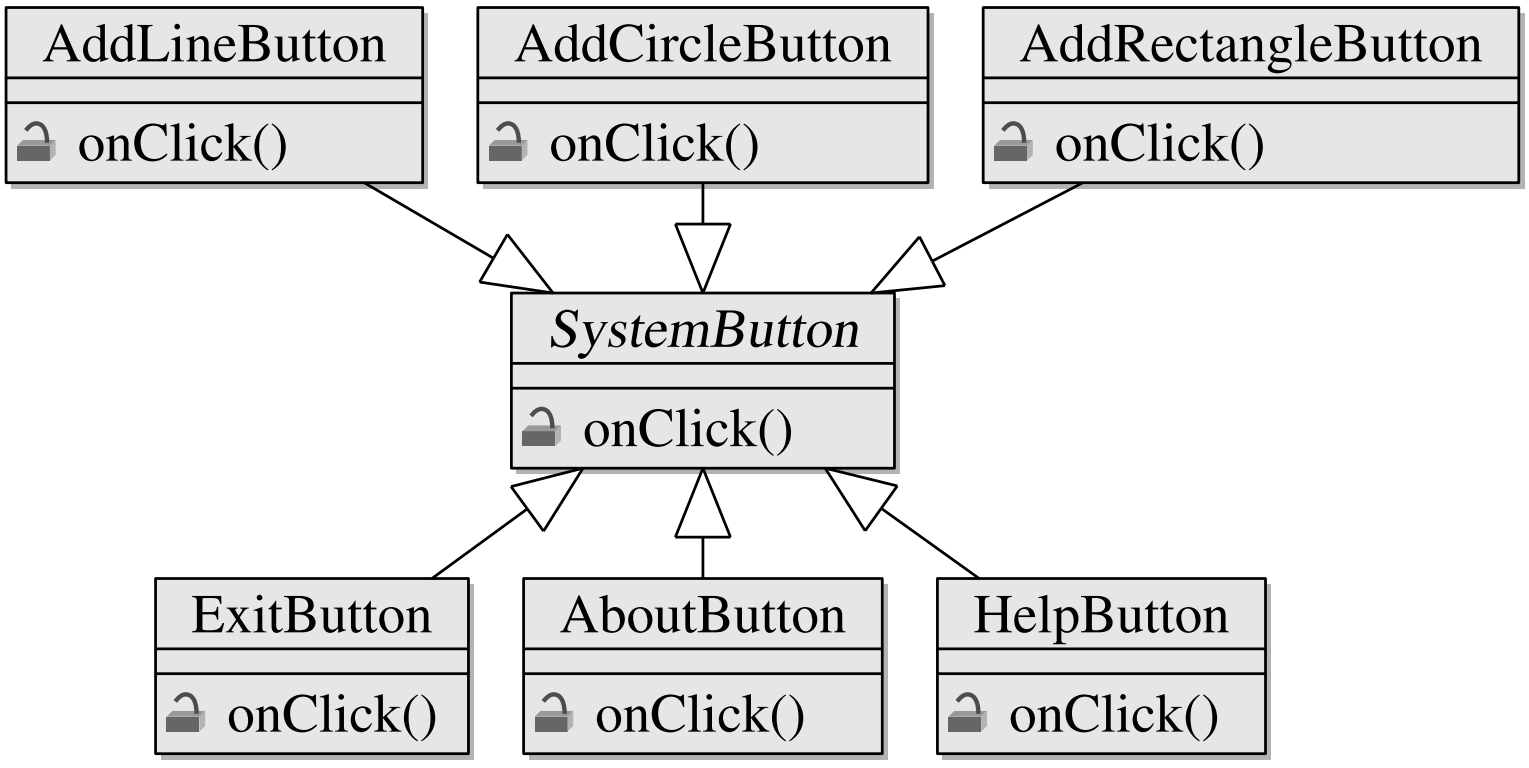
Класс SystemButton

```
public class SystemButton extends SystemComponent {
    public SystemButton(String text);

    protected void onClick();
}
```

Кнопка добаления геометрической фигуры

```
public class AddLineButton extends SystemButton {
    public AddLineButton(Scene scene) {
        super("Add Line");
    }

    protected void onClick() }
        Line line = new Line(100, 100, 200, 200);
        scene.addShape(line);
    }

    private Scene scene;
}
```

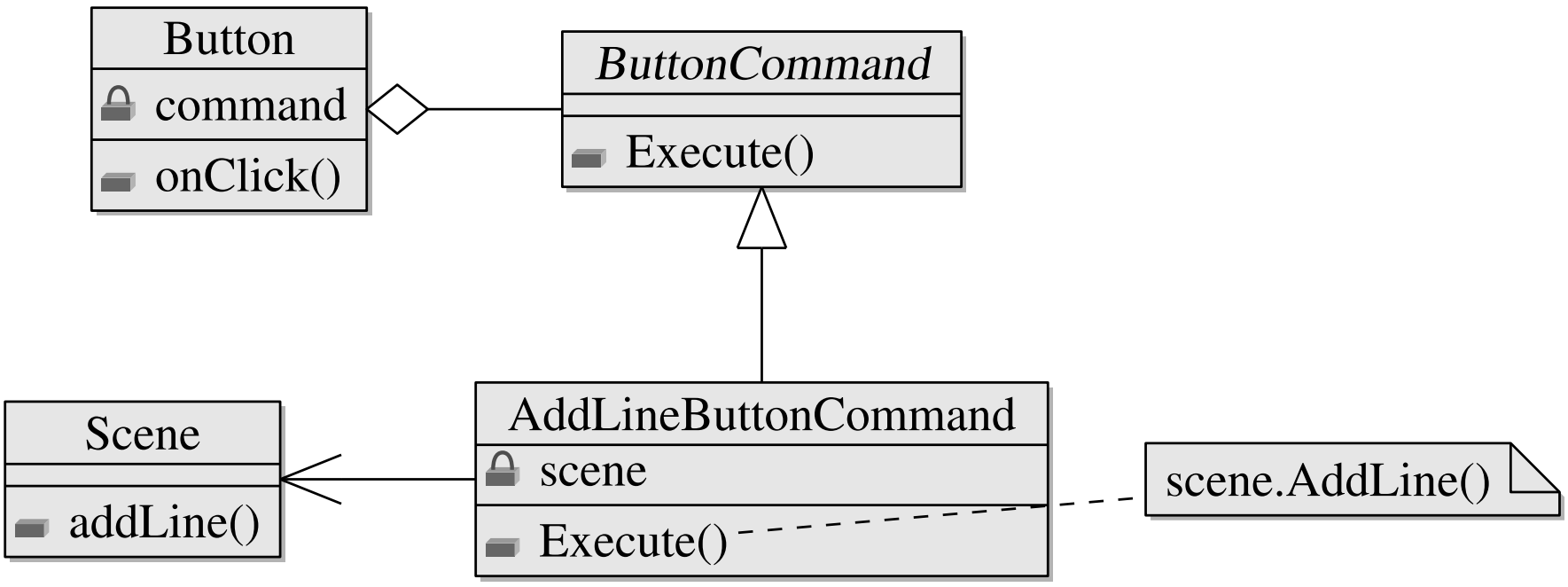
Иерархия классов кнопок



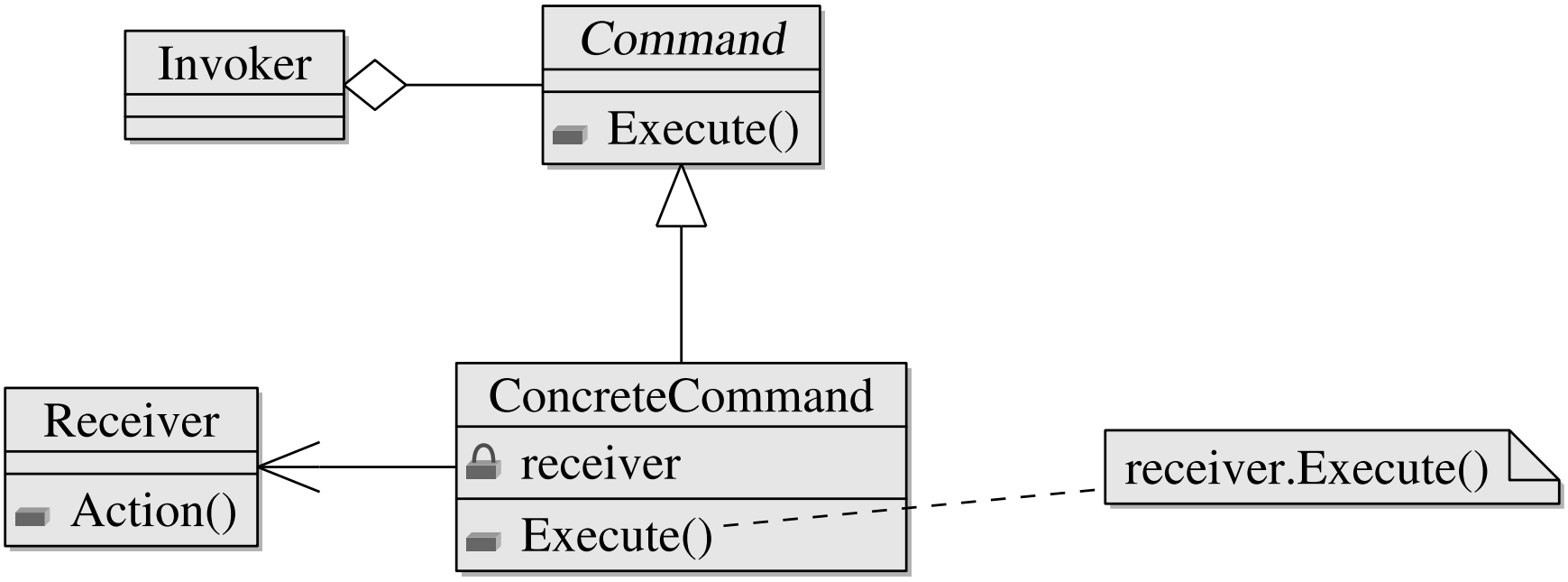
Недостатки

- Иерархия классов кнопок разрастается слишком сильно
- Обработчик нажатия кнопки нельзя переиспользовать в другом элементе управления
- Код обработки события жестко связан с кодом графического интерфейса

Отедльный класс, инкапсулирующий операцию



Паттерн Command



Результаты

- Возможность переиспользования обработчиков событий в разных элементах управления
- Независимость логики приложения от графического интерфейса
- Возможность уменьшения количества классов за счёт использования языковых средств (анонимные классы, функторы)
- Возможность легкого добавления новой функциональности:
  - Протоколирование/отмена операций
  - Группировка операций