Паттерн State

Есилевич Александр

18 ноября 2011 г.

Интерактивное рисование фигур

- Пользователь выбирает инструмент в панели
- Пользователь выполняет манипуляции мышью/клавиатурой на изображении
- В изображении выделяются фигуры или добавляются новые

Первая реализация

- В классе `Scene` храним ID выбранного инструмента
- Обрабатываем события мыши/клавиатуры тем или иным образом в зависмиости от ID выбранного инструмента

Первая реализация

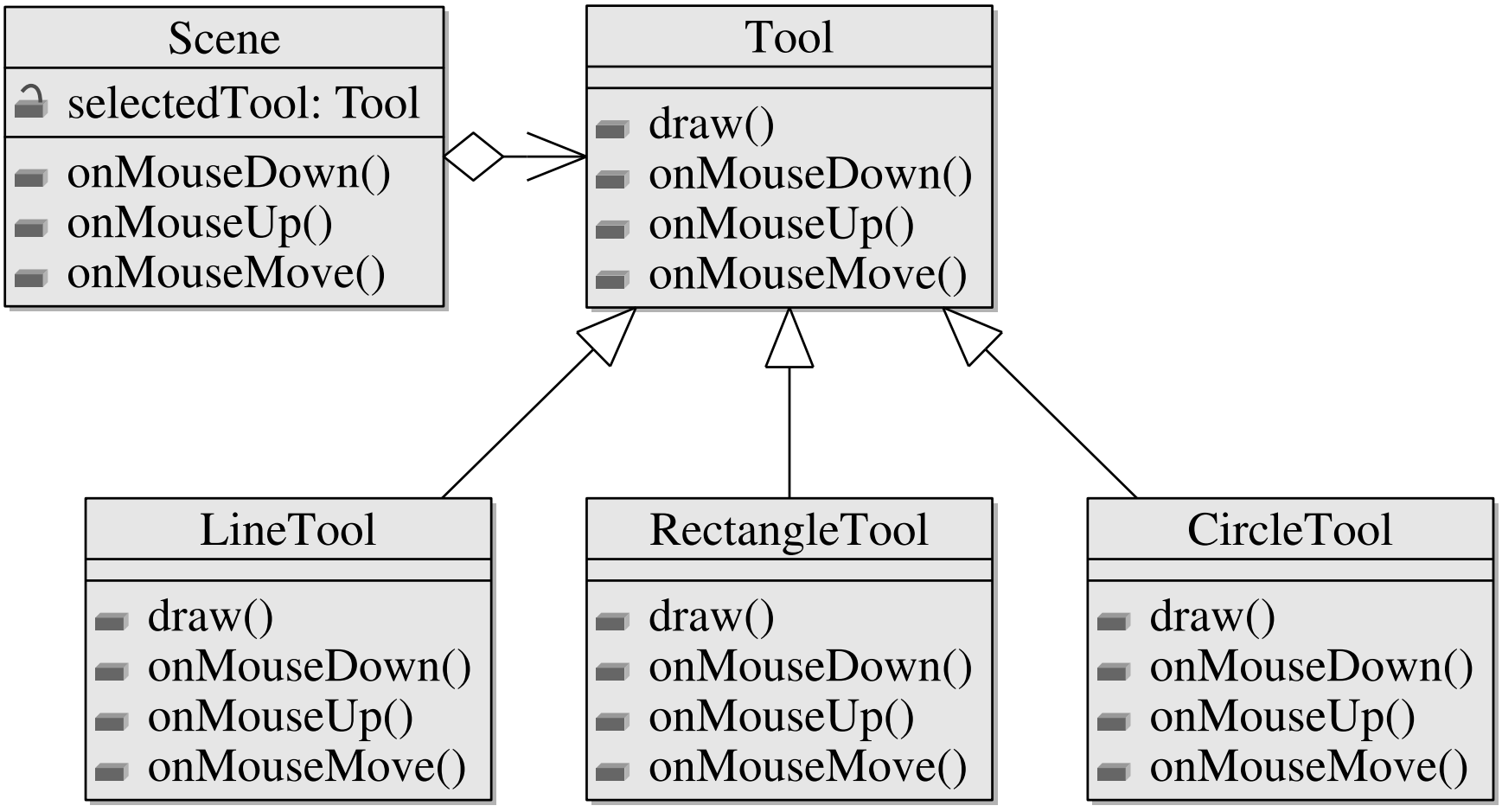
```
enum ToolId { TOOL_SELECT, TOOL_LINE,
              TOOL_RECTANGLE, TOOL_CIRCLE }
ToolId selectedToolId;

void onMouseDown(int x, int y) {
    switch(selectedToolId) {
    case TOOL_LINE:
        // line drawing logic
        break;
    case ...
    }
}

void onMouseUp(int x, int y) { ... }
void onMouseMove(int x, int y) { .. }
```

Как можно улучшить?

- Логика обработки каждого инструмента разная
- Инструменты не зависят друг от друга
- Инструменты обрабатывают одни и те же действия пользователя

Вывод: можно создать абстрактный класс `Tool`, определяющий интерфейс обработки действий пользователя, и опеделить класс-наследник для каждого инструмента

Класс Tool и наследники



Реализация класса LineTool

Есть три состояния:

- `START` – рисование только начали
- `PRESSED` – нажали кнопку мыши, "тянем" линию, пока кнопку не отпустят
- `FREE_MOVE` – кнопку мыши отпустили там же, где нажали, "тянем" линию, пока кнопку ещё раз не нажмут

Класс `LineTool` хранит ссылку на список фигур, состояние рисования, начальные и конечные координаты линии.

Реализация класса LineTool

```
public class LineTool extends Tool {
    enum State { START, PRESSED, FREE_MOVE }

    private ShapeList shapes;
    private State state;
    private int startX, startY, endX, endY;

    public void onMouseDown() { ... }
    public void onMouseUp() { ... }
    public void onMouseMove() { ... }
}
```

Реализация класса LineTool

```
public void onMouseDown(int x, int y) {
    switch(state) {
    case START:
        startX = x;
        startY = y;
        endX = x;
        endY = y;
        state = State.PRESSED;
        break;
    case PRESSED:
        break;
    case FREE_MOVE:
        break;
    }
}
```

Реализация класса LineTool

```
public void onMouseMove(int x, int y) {
    switch(state) {
    case START:
        break;

    case PRESSED:
    case FREE_MOVE:
        endX = x;
        endY = y;
        break;
    }
    endX = x;
    endY = y;
}
```

Реализация класса LineTool

```
public void onMouseUp(int x, int y) {
    switch(state) {
    case START:
        break;

    case PRESSED:
        if(startX == x && startY == y) {
            state = State.FREE_MOVE;
        } else {
            shapes.add(new Line(startX, startY, x, y));
            state = State.START;
        }
        break;

    case FREE_MOVE:
        shapes.add(new Line(startX, startY, x, y));
        state = State.START;
        break;
    }
}
```

Реализация класса LineTool

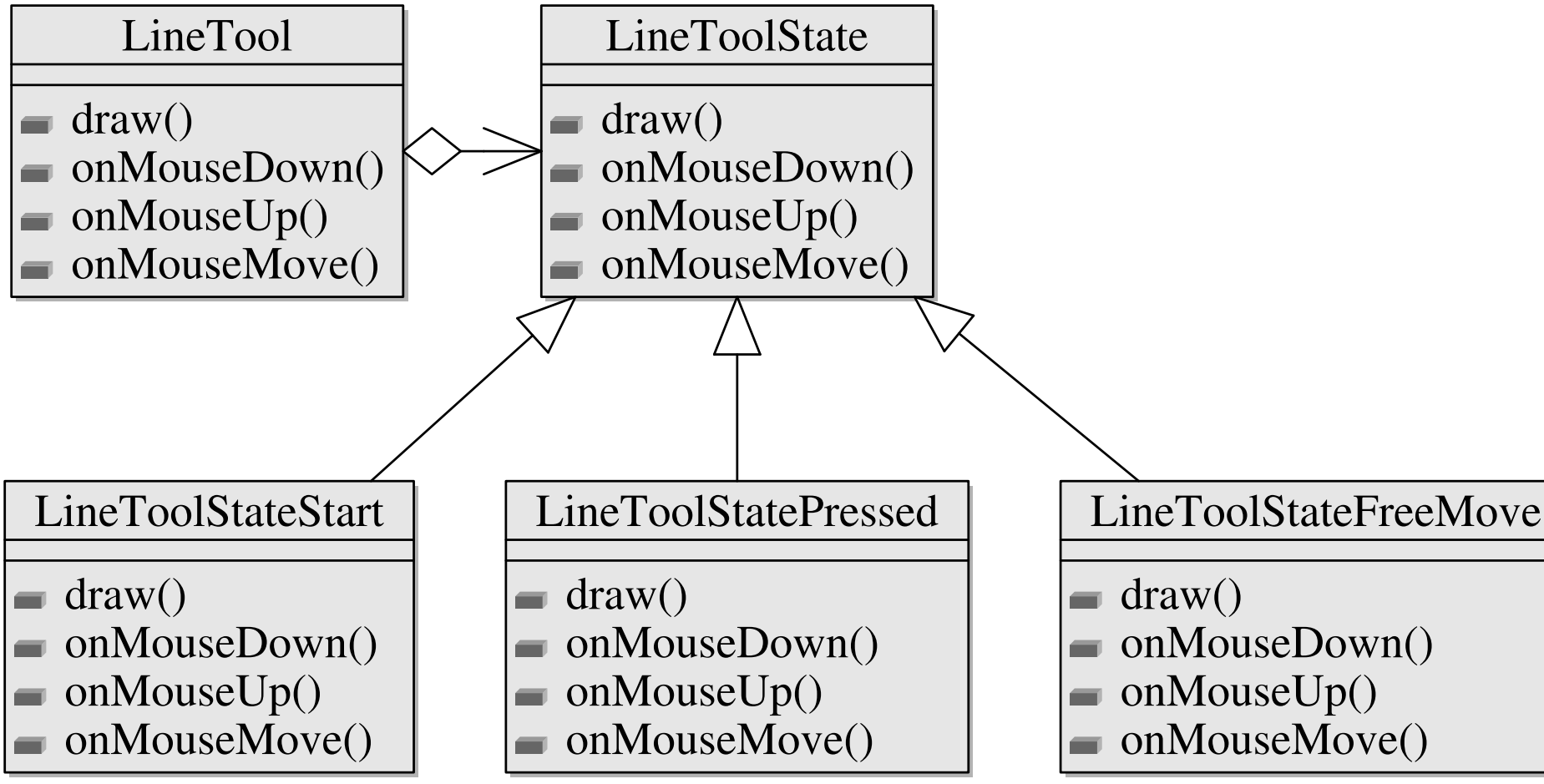
```
public void draw(SystemCanvas canvas) {

    if(state == State.START) {
        return;
    }

    (new Line(startX, startY, endX, endY)).draw(canvas);
}
```

Недостатки реализации LineTool

- Присутствуют большие операторы `switch`
- Логика перехода между состояниями явно не выделена
- Код сложен для понимания и добавления новой функциональности

Решение: ввести отдельный класс для каждого состояния

Диаграмма классов LineToolState



Кто будет изменять состояние?

- класс `LineTool`
- каждое состояние будет само изменять состояние
  - классы состояний меют доступ к `LineTool`
  - обработчики пользовательского ввода будут возвращают новое состояние

Модифицированный класс LineTool

```
public class LineTool extends Tool {
    LineTool(ShapeList list) {
        shapes = list;
        state = new LineToolStateStart(shapes);
    }
    public void draw(SystemCanvas canvas) {
        state.draw(canvas); }
    public void reset() {
        state = new LineToolStateStart(shapes) ]
    }
    public void onMouseDown(int x, int y) {
        state = state.onMouseDown(x, y);
    }
    public void onMouseUp(int x, int y) {
        state = state.onMouseUp(x, y);
    }
    public void onMouseMove(int x, int y) {
        state = state.onMouseMove(x, y);
    }
    ShapeList shapes;
    LineToolState state;
}
```

Класс LineToolStateStart

```
class LineToolStateStart extends LineToolState {
    public LineToolStateStart(ShapeList list) {
        shapes = list;
    }

    public void draw(java.awt.Graphics canvas) {}

    public LineToolState onMouseDown(int x, int y) {
        return new LineToolStatePressed(shapes, x, y);
    }

    public LineToolState onMouseUp(int x, int y) {
     return this;
    }

    public LineToolState onMouseMove(int x, int y) {
        return this;
    }

    private ShapeList shapes;
};
```

Класс LineToolStatePressed

```
private ShapeList shapes;
private int startX, startY, endX, endY;

public LineToolStatePressed(ShapeList list,
                            int x, int y) {
    shapes = list;
    startX = x;
    startY = y;
    endX = x;
    endY = y;
}

public void draw(java.awt.Graphics canvas) {
    (new Line(startX, startY, endX, endY))
        .draw(canvas);
}

public LineToolState onMouseDown(int x, int y) {
    return this;
}
```

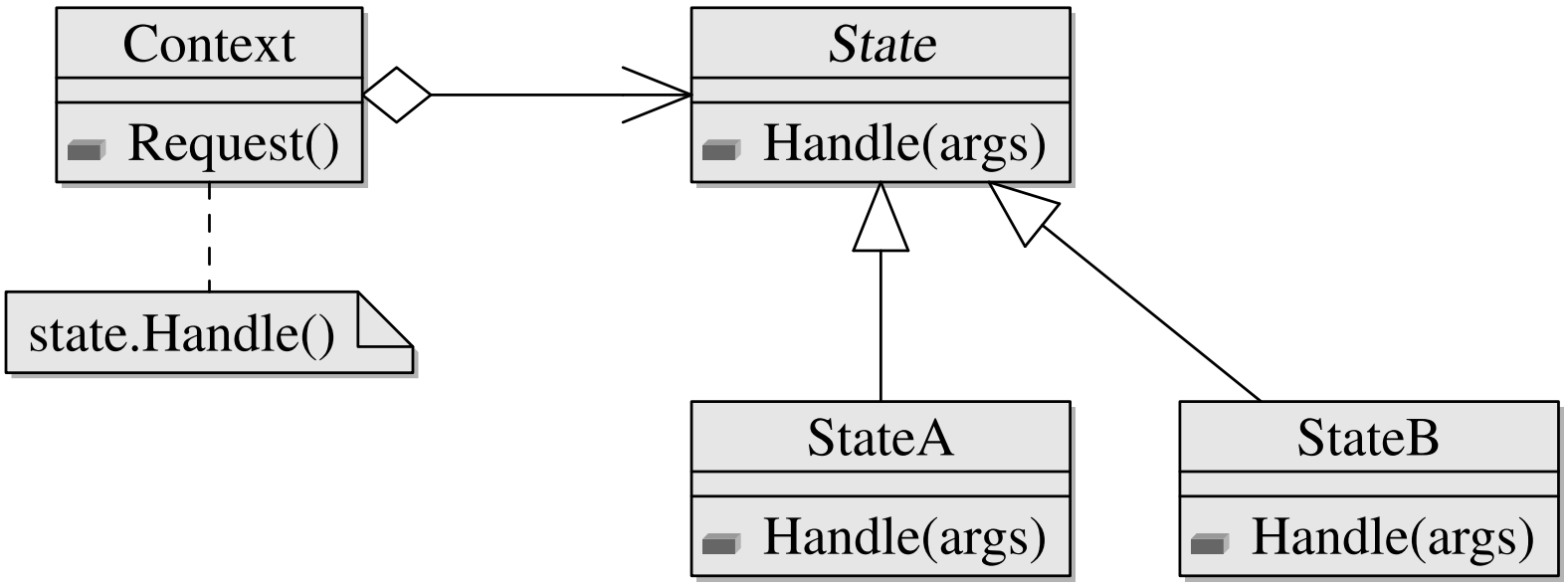
Класс LineToolStatePressed

```
public LineToolState onMouseUp(int x, int y) {
    if(x == startX && y == startY) {
        return new LineToolStateFreeMove(shapes, x, y);
    } else {
        shapes.add(new Line(startX, startY, x, y));
        return new LineToolStateStart(shapes);
    }
}

public LineToolState onMouseMove(int x, int y) {
    endX = x; endY = y; return this;
}
```

Назначение

Паттерн State позволяет объекту изменять своё поведение в зависимости от внутреннего состояния. Извне создаётся впечатление, что изменился класс объекта.

Диаграмма классов



Результаты

Преимущества:

- Локализует зависящее от состояния поведение и делит его на части, соответствующие состояниям
- Избавляет от необходимости использования больших операторов `if` и `switch`
- Делает явными переходы между состояниями
- Объекты состояния можно разделять

Недостатки:

- Увеличивает число классов

Класс Tool тоже может быть паттерном State

- Определяем один класс `ToolState` для всех инструментов
- Задаём начальное состояние при выборе инструмента
- В качестве контекста в данном случае выступает класс `Scene`

Пример: TcpConnection

```
public class TcpConnection {
    public bool connect(String addr, int port);
    public void close();
    public void send(byte [] data);
    public byte [] recv();
    public void listen(int port);
    public void accept();
}
```

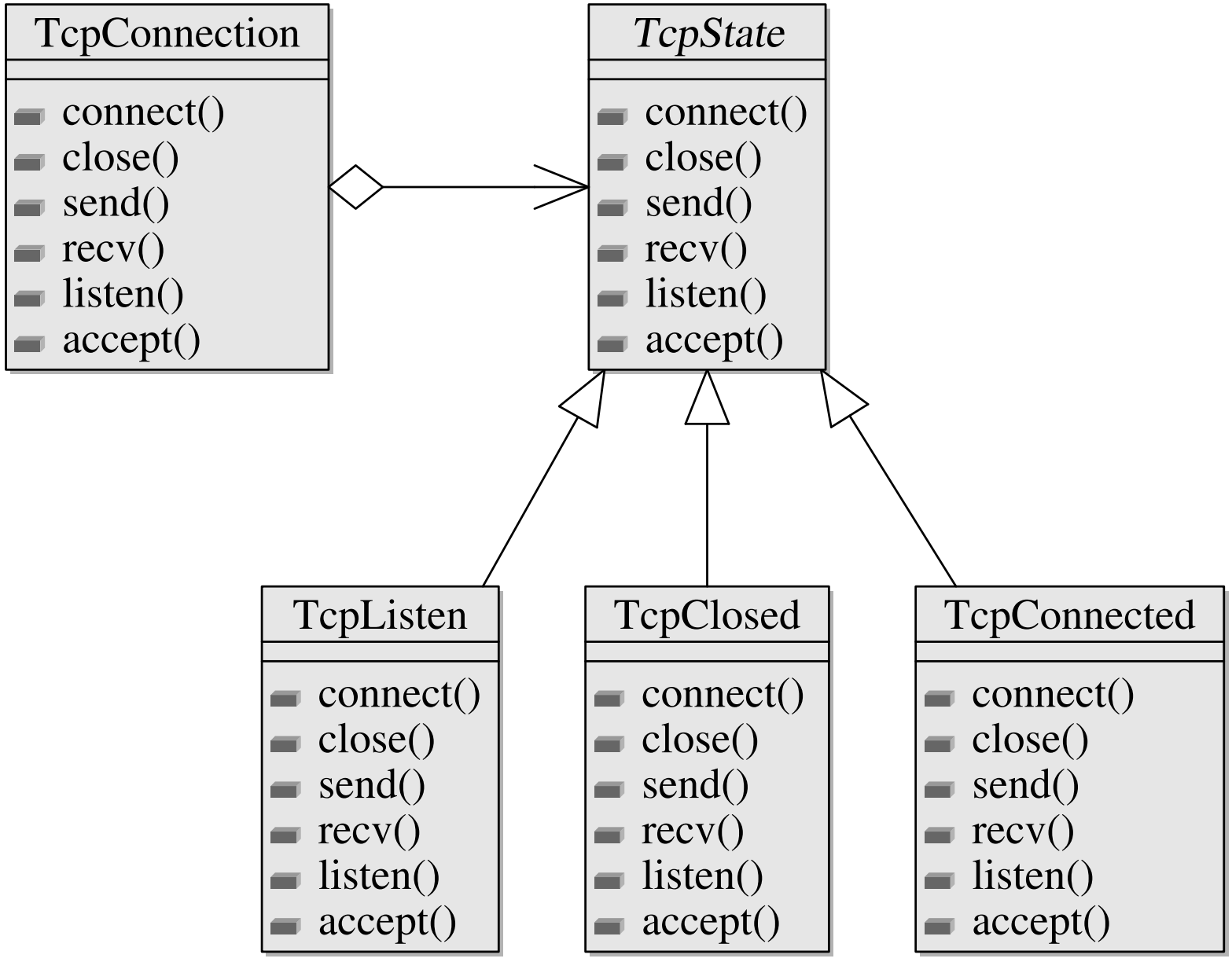
Пример: TcpConnection

```
public class TcpConnection {
    private enum State { CLOSED, CONNECTED, LISTEN }
    private State state;

    TcpConnection() {
        state = CLOSED;
    }

    public bool connect(String addr, int port) {
        if (state == CONNECTED || state == LISTEN) {
            // throw exception
        }

        if (connectToRemoteHost(addr, port)) {
            state = CONNECTED;
            return true;
        }
        return false;
    }
}
```

Пример: TcpConnection

```
public class TcpConnection {
    public void send(byte [] data) {
        if (state != CONNECTED) {
            // throw exception
        }

        // send data
    }

    public void close() {
        if (state == LISTEN) {
            // cancel listening
        } else if(state == CONNECTED) {
            // send disconnect packet
        } else {
            // throw exception
        }

        state = CLOSED;
    }
}
```

Пример: TcpConnection



Пример: TcpConnection

```
public class TcpClosed {
    public bool connect(String addr, int port) {
        if (connectToRemoteHost(addr, port)) {
            changeState(new TcpConnected());
            return true;
        }
        return false;
    }

    public void listen(int port) {
        changeState(new TcpListen());
    }

    public void send() { /* throw exception */ }
    public void recv() { /* throw exception */ }
    public void close() { /* throw exception */ }
    public void accept() { /* throw exception */ }
}
```

Пример: TcpConnection

```
public class TcpListen {
    public void close() {
        // stop listening
        changeState(new TcpClosed());
    }

    public void accept() {
        // accept connection
        changeState(new TcpConnected());
    }

    public bool connect() { /* throw exception */ }
    public void send() { /* throw exception */ }
    public void recv() { /* throw exception */ }
    public void listen() { /* throw exception */ }
}
```

Пример: TcpConnection

```
public class TcpConnected {
    public void close() {
        // close connection
        changeState(new TcpClose());
    }

    public void send(byte [] data) {
        // send data
    }

    public byte [] recv() {
        // receive data
    }

    public bool connect() { /* throw exception */ }
    public void listen() { /* throw exception */ }
    public void accept() { /* throw exception */ }
}
```