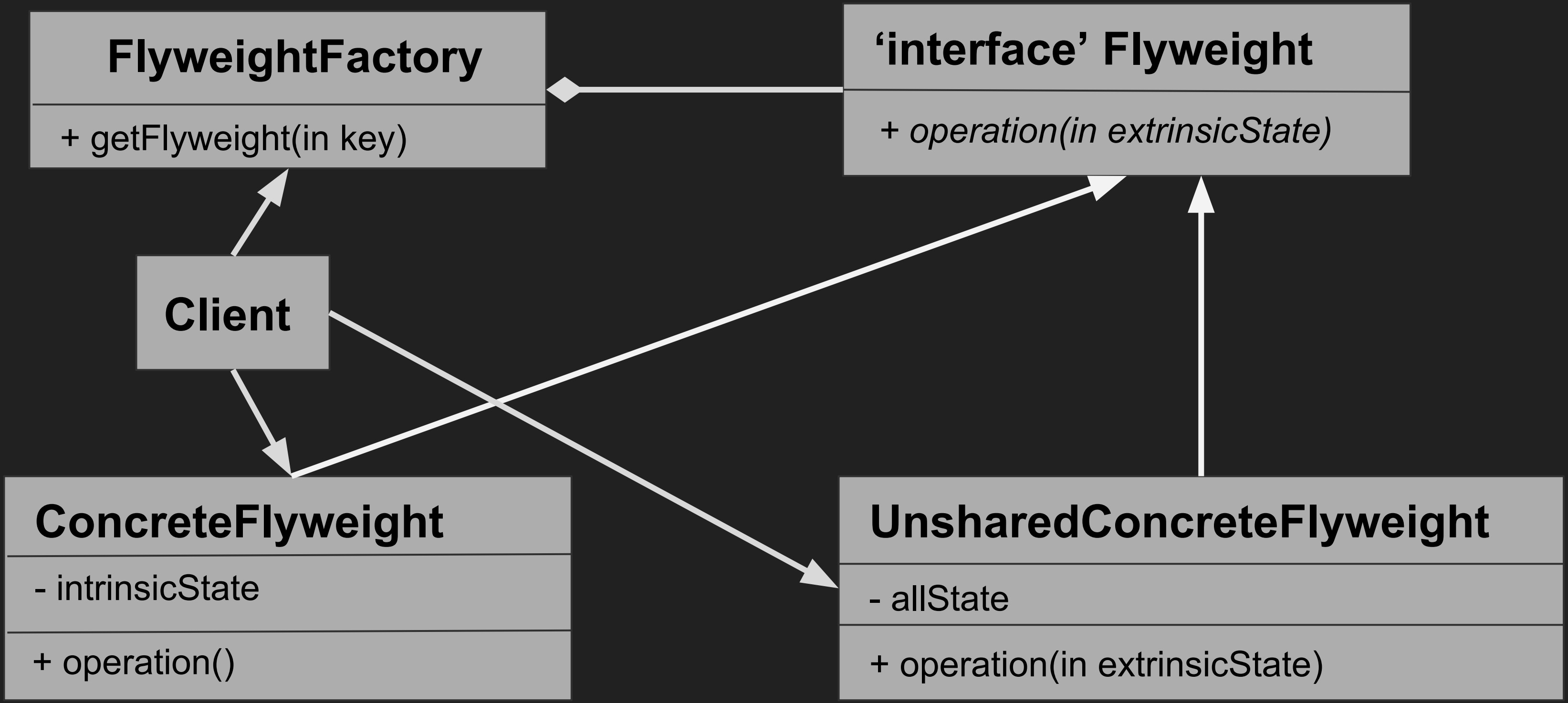
Технологии программирования

Структурные паттерны. Flyweight, Decorator, Proxy.
Использование Doxygen для документирования кода

Александр Михалевич

Flyweight


FlyweightFactory
+ getFlyweight(in key)
'interface' Flyweight
+ *operation(in extrinsicState)*
Client
ConcreteFlyweight
- intrinsicState
+ operation()
UnsharedConcreteFlyweight
- allState
+ operation(in extrinsicState)

Flyweight example

```
class CParticle {
private:
        CColor m_color;
        CSprite m_sprite;
        CVector m_speed;
        CPoint m_coordinates;
        /* ... */
```

```
class CGame {
private:
        vector<CParticle> m_particles;
        /* ... */
```

m_color	4B
m_coordinates	6B
m_speed	6B
m_sprite	20048B
particle ~ 20064B * 1’000’000 = ~ 18.68 GB	

Flyweight example

```
class CParticle {
private:
        CColor m_color;
        CSprite m_sprite;
        /* ... */
```

```
class CGame {
private:
        vector<CMovingParticle>
m_mps;
        CParticle m_particles;
        /* ... */
```

```
class CMovingParticle {
private:
        CVector m_speed;
        CPoint m_coordinates;
        /* ... */
```

m_color	4B
m_coordinates	6B
m_speed	6B
m_sprite	20048B
particle ~ 20052B * 1 + 12B * 1’000’000 = ~ 11.5 MB	

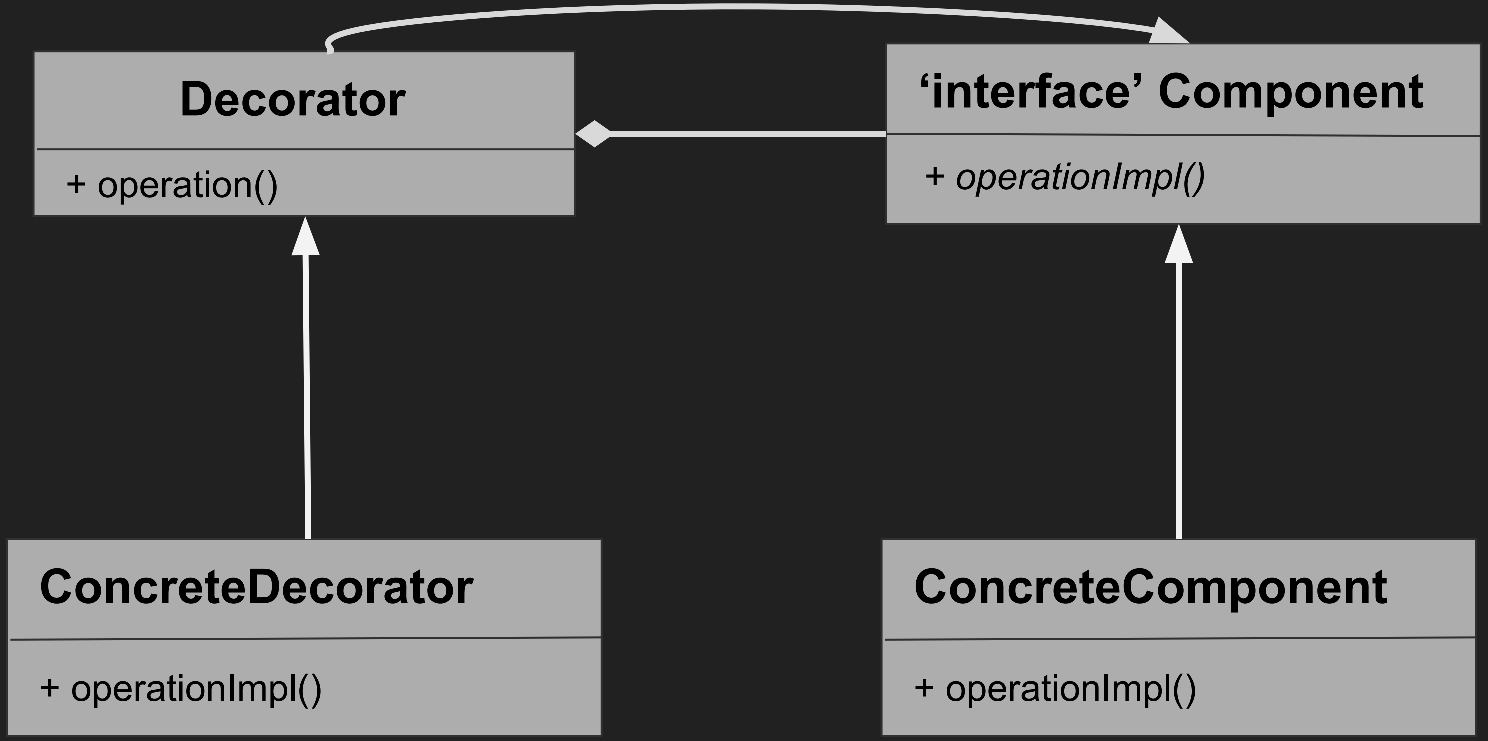
Flyweight in real life

Java Integer class

Relations

flyweight vs singleton

Decorator


Decorator
+ operation()
'interface' Component
+ *operationImpl()*
ConcreteDecorator
+ operationImpl()
ConcreteComponent
+ operationImpl()

Decorator example

```
class IHTTPServer {
public:
    virtual result_t get(const data_t& data, data_t& response) = 0;
    virtual result_t post(const data_t& data, data_t& response) = 0;
};
```

```
class CHTTPServerImpl : public IHTTPServer {
public:
    result_t get(const data_t& data, data_t& response) {
        // implementation
    }
    result_t post(const data_t& data, data_t& response) {
        // implementation
    }
};
```

```
class CHTTPServerStub : public IHTTPServer {
public:
    result_t get(const data_t& data, data_t& response) {
        _set_test_response();
        return sOk;
    }
    result_t post(const data_t& data, data_t& response) {
        _set_test_response();
        return sOk;
    }
};
```

Decorator example

client code

```
…
std::shared_ptr<IHTTPServer> server_instance =
          CServerFactory::get_standard_server();
if (server_instance == nullptr) {
    _trace("Failed to create server instance!");
    return eFail;
}
auto status = server_instance->get(my_data, response);
if (FAILED(status)) {
    _trace("Failed to get data, status code: ", status);
    return status;
}
...
```

test code

```
…
TEST_F(ResponseProcessor, BasicTest)
{
std::shared_ptr<IHTTPServer> server =
          CServerFactory::get_server_stub();
ASSERT_NE(server_instance, nullptr);
auto status =
          m_processor->from_server(query, server,output);
EXPECT_EQ(status, sOk);
}
...
```

Decorator example

```
class IHTTPServer {
public:
    virtual result_t get(const data_t& data, data_t& response) = 0;
    virtual result_t post(const data_t& data, data_t& response) = 0;
};
```

```
class CHTTPServerEncryptionDecorator : public IHTTPServer {
public:
    CHTTPServerEncryptionDecorator(std::shared_ptr<IHTTPServer> server)  ← НЕ управляет временем жизни
            : m_server(server) {}
    result_t get(const data_t& data, data_t& response) {
        return m_server->get(CEncryptor::encrypt(data), response);
    }
    result_t post(const data_t& data, data_t& response) {
        return m_server->post(CEncryptor::encrypt(data), response);
    }
private:
    std::shared_ptr<IHTTPServer> m_server;
};
```

Decorator example

client code

```
…
std::shared_ptr<IHTTPServer> server_instance = CServerFactory::get_standard_server();
if (server_instance == nullptr) {
    _trace("Failed to create server instance!");
    return eFail;
}
CHTTPServerEncryptionDecorator ecrypt_decorator(server_instance);
auto status = ecrypt_decorator->get(my_data, response);
if (FAILED(status)) {
    _trace("Failed to get data, status code: ", status);
    return status;
}
...
```

Decorator in real life: ThreadWeaver

```
class THREADWEAVER_EXPORT QObjectDecorator : public QObject, public IdDecorator {
        Q_OBJECT
public:
        explicit QObjectDecorator(JobInterface *decoratee, QObject *parent = nullptr);
        explicit QObjectDecorator(JobInterface *decoratee, bool autoDelete, QObject *parent = nullptr);

Q_SIGNALS:
        void started(ThreadWeaver::JobPointer);
        void done(ThreadWeaver::JobPointer);
        void failed(ThreadWeaver::JobPointer);

protected:
      void defaultBegin(const JobPointer& job, Thread *thread) Q_DECL_OVERRIDE;
      void defaultEnd(const JobPointer& job, Thread *thread) Q_DECL_OVERRIDE;
};

typedef QSharedPointer<QObjectDecorator> QJobPointer;
```

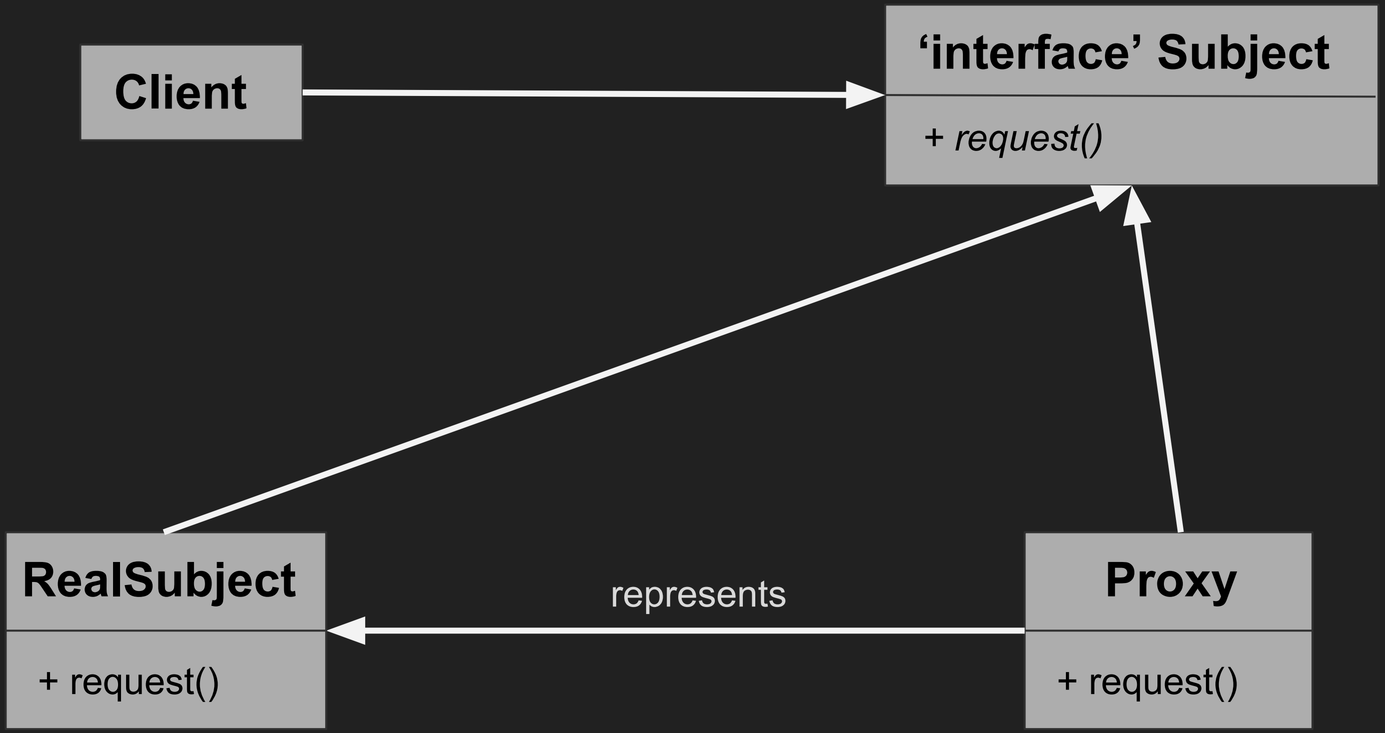
Relations

adapter vs proxy vs decorator

adapter vs decorator

composite & decorator & prototype

Proxy


Client
‘interface’ Subject
+ *request()*
RealSubject
+ request()
represents
Proxy
+ request()

Proxy example

```
class IVideoReceiver {
public:
    virtual result_t get(const url_t& url, video_t& video) = 0;
};
```

```
class CVideoReceiverImpl : public IVideoReceiver {
public:
    result_t get(const url_t& url, video_t& video) {
        // implementation
    }
};
```

Proxy example

```
class IVideoReceiver {
public:
    virtual result_t get(const url_t& url, video_t& video) = 0;
};
```

```
class CVideoReceiverCacheProxy : public IVideoReceiver {
public:
    CVideoReceiverCacheProxy() {
        m_receiver =
          std::shared_ptr<IVideoReceiver>(new CVideoReceiverImpl);
    }

    result_t get(const url_t& url, video_t& video) {
        if (CCache::find(url))
            return CCache::get(url, video);
        return m_receiver->get(url, video);
    }

private:
    std::shared_ptr<IVideoReceiver> m_receiver;
};
```

← управляет временем жизни

Decorator vs Proxy

Decorator	Proxy
получает ссылку на делегируемый объект извне	создает объект самостоятельно
всегда удерживает ссылку на делегируемый объект	может не инстанцировать объект вовсе
предоставляет такой же или расширенный интерфейс	предоставляет такой же интерфейс, как и делегируемый объект
“указывает” на свой базовый класс	может “указывать” на другой производный класс (при этом имеет с ним общий интерфейс)

Proxy in real life: Qt

```
class Q_WIDGETS_EXPORT QProxyStyle : public QCommonStyle {
    Q_OBJECT
public:
    QProxyStyle(QStyle *style = Q_NULLPTR);
    QProxyStyle(const QString &key);
    ~QProxyStyle();
    QStyle *baseStyle() const;
    void setBaseStyle(QStyle *style);
    void drawPrimitive(PrimitiveElement element, const QStyleOption *option, QPainter *painter,
                                    const QWidget *widget = Q_NULLPTR) const Q_DECL_OVERRIDE;
…
protected:
    bool event(QEvent *e) Q_DECL_OVERRIDE;
private:
    Q_DISABLE_COPY(QProxyStyle)
    Q_DECLARE_PRIVATE(QProxyStyle)
};
```

Proxy in real life: Okular (nontrivial use)

```
class OKULARCORE_EXPORT AnnotationProxy {
    public:
        enum Capability
        {
            Addition,       ///< Generator can create native annotations
            Modification,   ///< Generator can edit native annotations
            Removal         ///< Generator can remove native annotations
        };
        virtual ~AnnotationProxy();
        virtual bool supports(Capability capability) const = 0;
        virtual void notifyAddition(Annotation *annotation, int page) = 0;
        virtual void notifyModification(const Annotation *annotation, int page, bool appearanceChanged) = 0;
        virtual void notifyRemoval(Annotation *annotation, int page ) = 0;
};
```

Relations

adapter vs proxy vs decorator

facade vs proxy

decorator vs proxy

Doxygen

```
class CAlgoLMDirichlet {
public:
        /**
         * \brief       Executes Language Model algorithm using Dirichlet evaluation.
         * \details     This LM implementation uses unigram language model for the given corpus.
         * \param[in]       query       the given query
         * \param[in]       text        the given document
         * \param[out]      score       the result Dirichlet similarity
         * \return                      sOk if succedeed,
         *                              the appropriate error code otherwise
         */
        static result_t execute(const text_t& query, const text_t& text, double& score);
….
};
```