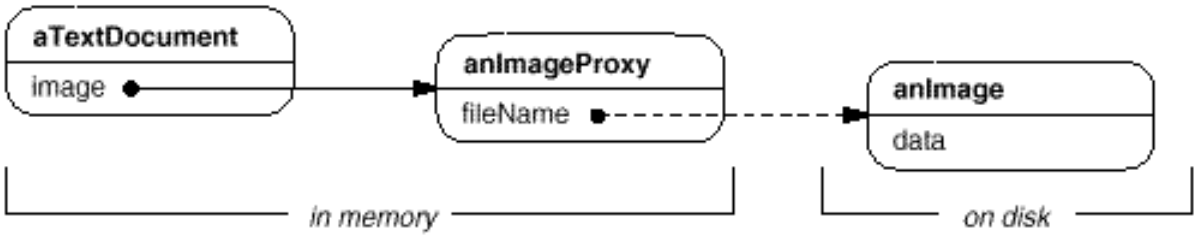
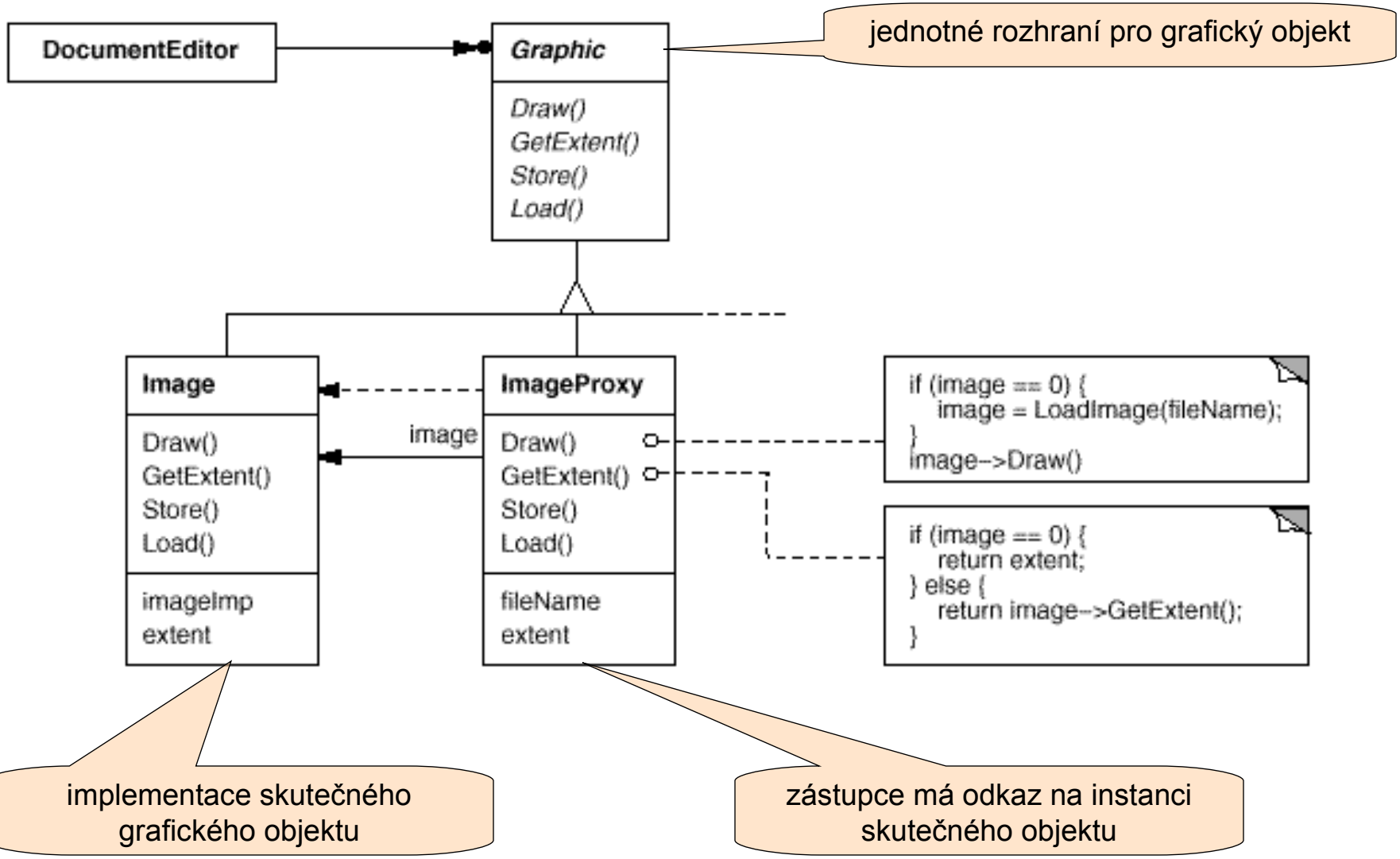
Proxy

Proxy

- **Známý jako**
  - Placeholder, Surrogate
- **Účel**
  - představuje zástupce/náhradníka daného objektu
  - kontroluje a zpracovává přístupy k danému objektu
  - vytváří vrstvu abstrakce při přístupu k objektu
- **Motivace**
  - editor dokumentů, který umožňuje vkládání grafických objektů - `Graphic`
  - očekáváme rychlé otevírání dokumentů, ale načítání objektů může být pomalé
  - řešení: nahrazení původních objektů zástupci (proxy objekty) – `ImageProxy`
  - proxy zabezpečí načítání objektu na pozadí či ve chvíli, kdy je poprvé použit



Proxy - motivace



Proxy - příklad

```
class Graphic
 {
 public:
    virtual ~Graphic();
    virtual void Draw(const Point& at) = 0;
    virtual const Point& GetExtent() = 0;
protected:
    Graphic();
};
```

```
class Image : public Graphic
{
public:
    Image(const char* file);
    virtual ~Image();
    virtual void Draw(const Point& at);
    virtual const Point& GetExtent();
private: // ...
};
```

```
class ImageProxy : public Graphic
{
    public:
    ImageProxy(const char* imageFile);
    virtual ~ImageProxy();
    virtual void Draw(const Point& at);
    virtual const Point& GetExtent();
protected:
    Image* GetImage();
private:
    Image* _image; Point _extent;
    char* _fileName;
};
```

Proxy - příklad

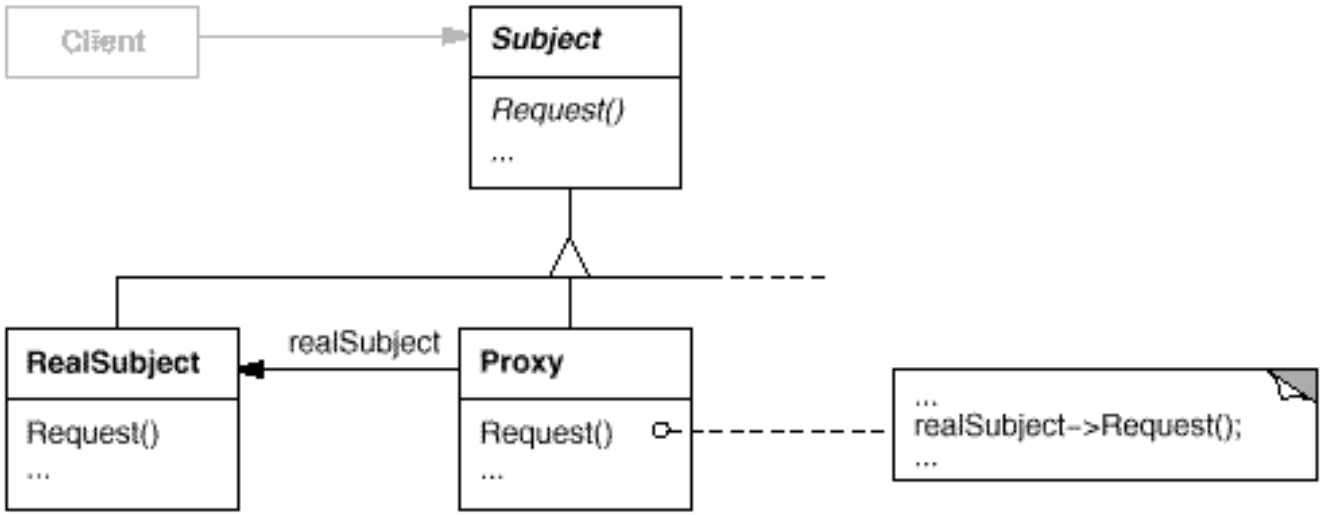
```
class Image : public Graphic
{
public:
  Image(const char* file);
  virtual ~Image();
  virtual void Draw(const Point& at);
  virtual const Point& GetExtent();
private: // ...
};
```

```
class Document
{
public:
    void Insert(Graphic*);
    // ...
};


Document * doc = new Document;
doc->Insert(
    new ImageProxy("ImgFileName")
);
```

```
ImageProxy::ImageProxy(const char * fileName)
{
    _fileName = strdup(fileName);
    _extent = Point::Zero;
    _image = 0;
}

Image * ImageProxy::GetImage()
{
    if (_image == 0)
          _image = new Image(_fileName);
    return _image;
}

void ImageProxy::Draw(const Point& at)
{
    GetImage()->Draw(at);
}

const Point& ImageProxy::GetExtent()
{
    if (_image == 0 && _extent == Point::Zero)
          //rychle nacteni _extent pouze
          //z hlavicky obrazku
    else if (_extent == Point::Zero)
          _extent = GetImage()->GetExtent();
    return _extent;
}
```

Proxy - struktura

- **Struktura**



- **Účastníci**
  - **Subject** (`Graphic`)
    - definuje společné rozhraní skutečných objektů a proxy objektů
      - C++: společná abstraktní třída
      - Java: společný interface
  - **RealSubject** (`Image`)
    - definuje skutečný objekt
  - **Proxy** (`ImageProxy`)
    - implementuje stejné rozhraní jako RealSubject (proxy jej může transparentně nahradit)
    - uchovává odkaz na skutečný subjekt, kterému deleguje provádění požadavků

Proxy - zoologie

- **Virtual proxy**
  - vytváření skutečných objektů až v případě potřeby („*load on demand*“) - viz motivace
  - rozlišení mezi vyžádáním instance objektu a skutečnou potřebou objekt použít
  - transparentní provádění optimalizací (např. **Caching proxy**)
- **Remote proxy**
  - lokální zástupce vzdáleného objektu (v jiném adresovém prostoru, na jiném počítači)
  - middleware: Java RMI, CORBA, XML/SOAP,...
- **Protection proxy**
  - transparentní kontrola či omezování přístupů ke skutečnému objektu
- **Copy-on-write proxy**
  - opožděné kopírování velkých objektů až při pokusu o jejich modifikaci
- **Synchronization proxy**
  - transparentní synchronizace vláken při přístupu k objektu
- **Smart pointers/reference**
  - náhrada běžných ukazatelů (zejm. v C++) či referencí
  - počítání referencí a automatické odalokování
  - načítání do paměti při první dereferenci

„protection proxy“

Proxy - implementace

- Abstraktní proxy pro subjekty více typů
  - pracuje-li se skutečným objektem jen pomocí rozhraní, může být typově nezávislá
  - konkrétní instanci lze přiřadit např. v konstruktoru:

```
Graphic * g = new GraphicProtectionProxy(new Image);
```

- Přetížení „->“ v C++ jako proxy
  - znemožní rozlišení mezi konkrétními požadavky, předávají se všechny
  - Virtual proxy – `ImagePtr`, Real Subject – `Image`

Implementace:

```
Image* ImagePtr::LoadImage ()
{ if (_image == 0)
    _image = LoadFile(_imageFile);

  return _image;
}

Image* ImagePtr::operator-> ()
{ return LoadImage(); }

Image& ImagePtr::operator* ()
{ return *LoadImage(); }
```

Použití:

```
ImagePtr image = ImagePtr("ImgFileName");
image->Draw(Point(0, 0));
// (image.operator->())->Draw(Point(0, 0))
```

Protection proxy - Java

- Řešení problému absence *const* v Javě:
  - omezená rozhraní
  - *protection proxies* (wrappers)

```
public interface java.util.Collection<E> {
     boolean contains(Object o);
     boolean add(E o);
     //....
}
```

```
public class UnmodifiableCollection<E> implements Collection<E> {
     private Collection<? extends E> c;

     public UnmodifiableCollection(Collection<? extends E> c) { this.c = c; }
     public boolean contains(Object o) { return c.contains(o); }
     public boolean add(E o) { throw new UnsupportedOperationException(); }
     //...
}

public static Collection<E> unmodifiableCollection(Collection<E> c) {
     return new UnmodifiableCollection<E>(c);
}
```

```
/ vytvoření seznamu
List lst = new ArrayList();
// příprava dat apod.
// nyní se vytvoří neměnná kolekce
Collection c = Collections.unmodifiableCollection(lst);
```

Synchronization proxy - Java

- Standardní kolekce nejsou v Javě thread-safe
  - umožňuje to vyšší rychlost
- Volitelné řešení: *synchronization proxies* (wrappers)
  - myšlenkově identické s předchozí `unmodifiableCollection`

```
public class SynchronizedCollection<E> implements Collection<E> {
     private Collection<? extends E > c;

     public SynchronizedCollection(Collection<E> c) { this.c = c; }
     public boolean contains(Object o) {
         synchronized(this) {return c.contains(o);}
     }
     public boolean add(E o) {
         synchronized(this) {return c.add(o);}
     }
     //...
}

public static Collection<E> synchronizedCollection(Collection<E> c) {
     return new SynchronizedCollection<E>(c);
}
```

```
// typický příklad - nepotřebujeme nesynchronizovanou instanci
List l = Collections.synchronizedList(new LinkedList());
```

Proxy +/-

- **Výhody**
  - zjednodušení cílového kódu díky zapouzdření optimalizací a přistupování do proxy
  - třídy lze vylepšovat bez použití přímé dědičnosti
  - transparentní NV
- **Nevýhody**
  - jakmile je skutečný objekt vytvořen, použití proxy znamená výpočetní zátěž
  - tvorba sady proxy pro větší subsystém může být zdlouhavá
    - řešení: automatizace pomocí např. Java Reflection
    - lepší řešení: vytvoření *Fasády* pro subsystém a jediné proxy (možno sloučit)
  - může generovat výjimky, které by skutečný objekt nikdy neprodukoval
    - ze síťového prostředí (*remote proxy*) či porušení ochrany (*protection proxy*)
    - cílový kód by na ně měl být připraven
- **Poznámka**
  - virtuální proxy přesouvá výpočetní složitost z inicializace do fáze běhu
    - může být problém pro již zinicializované real-time systémy

Proxy - související NV

- **Adapter**
  - též tvoří mezivrstvu
  - adaptuje jedno rozhraní na druhé odlišné
    - proxy implementuje identické rozhraní jako skutečný objekt

- **Decorator**
  - podporuje dynamické i rekurzivní přiřazení skutečného objektu
  - strukturálně velmi podobný vzor, avšak jiný použitím - přidává funkčnost
  - musí po celou dobu běhu držet fyzickou instanci skutečného objektu
    - u proxy ještě nemusí existovat či může být např. na jiném počítači