Proxy

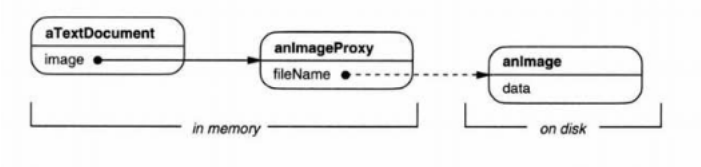
Baris Aktemur
CS 534 | Ozyegin University

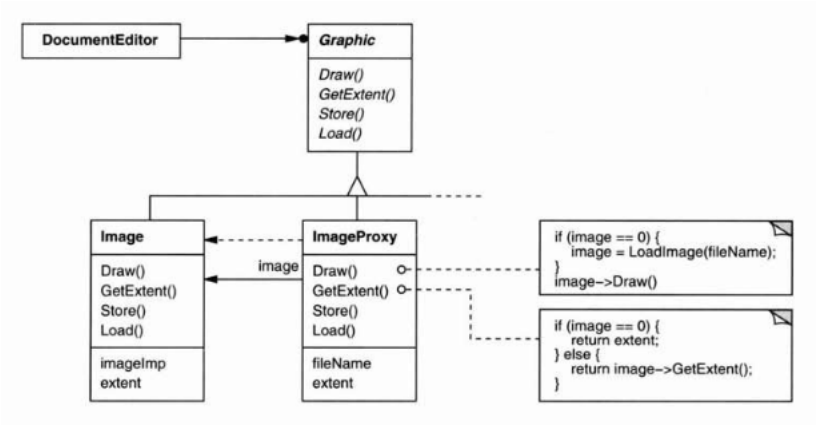
Contents are from “Design Patterns” by Gamma, Helm, Johnson, Vlissides

Proxy

Proxy

- Intent
  - Provide a surrogate or placeholder for another object to control access to it.
- Motivation
  - defer the full cost of creating and initializing an object until we actually need it.





```
class Image;
extern Image* LoadAnImageFile(const char*);
// external function
class ImagePtr {
public:
  ImagePtr(const char* imageFile);
  virtual ~ImagePtr();
  virtual Image* operator->();
  virtual Image& operator*();
private:
  Image* LoadImage();
private:
  Image* _image;
  const char* _imageFile;
};
```

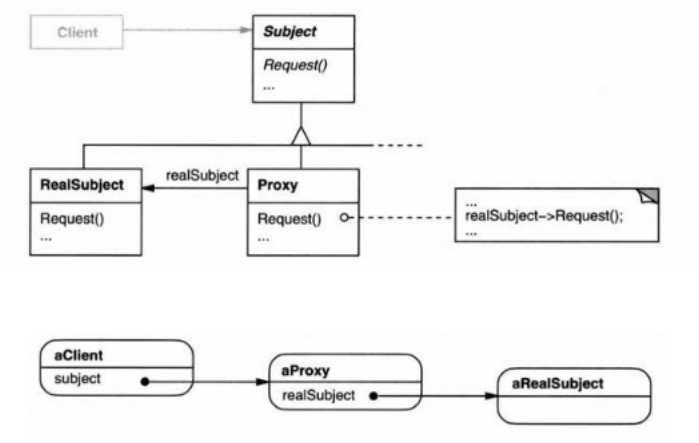


```
ImagePtr::ImagePtr(const char* theImageFile) {
  _imageFile = theImageFile;
  _image = 0;
}

Image* ImagePtr::LoadImage() {
  if (_image == 0) {
    _image = LoadAnImageFile(_imageFile);
  }
  return _image;
}

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

ImagePtr image = ImagePtr("anImageFileName");
image->Draw(Point(50, 100));
// (image.operator->())->Draw(Point(50, 100))
```

Applicability

- A **remote proxy** provides a local representative for an object in a different address space.
- A **virtual proxy** creates expensive objects on demand.
- A **protection proxy** controls access to the original object.
  - Protection proxies are useful when objects should have different access rights.
- A **smart reference** is a replacement for a bare pointer that performs additional actions when an object is accessed.
  - E.g: smart pointers, mutual exclusion

Structure



Consequences

- Optimization can be obtained for heavyweight objects, by doing **copy-on-demand** via proxies.