Mittuniversitetet
MID SWEDEN UNIVERSITY

Lektion 9 – DP Proxy

Designmönster med C++

Syfte: Introduktion av designmönstret Proxy

DP Proxy

Proxy är ett DP som inte presenteras i läroboken. Det tillhör kategorin ’structural patterns’ och har flera användningsområden.

Syfte:
” *Provide a surrogate or placeholder for another object to control access to it.* ” (GoF)

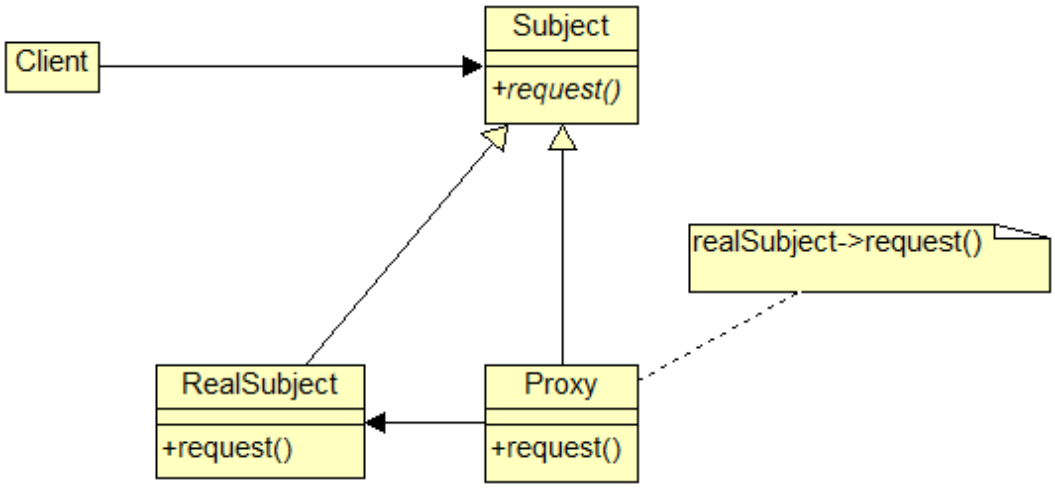
En Proxy-klass erbjuder alltså ett interface för att kommunicera med en annan klass som svarar för den egentliga implementationen.

- För applikationer som har två eller flera lager av kommunicerande mjukvara är det ofta praktiskt att införa en proxy som mellanliggande och åtskiljande lager. Exempelvis kan man i client/server-sammanhang låta klienten arbeta mot ett lokalt objekt, en *remote proxy*, som är en lokal representant för servern. Denna proxy döljer den egentliga kommunikationen med servern. Detta tillämpas i fler frameworks, t.ex. Javas RMI och Corba.

- En proxy som dessutom filtrerar kommunikationen och avgör vilken information som ska skickas, och hur den ska skickas brukar kallas en *protection proxy*.

Generellt sett så ändrar DP Proxy kommunikationen mellan klienten och mål-objektet genom att introducera ett mellanskikt.

Proxyn skickar meddelanden från klienten vidare till målobjektet:

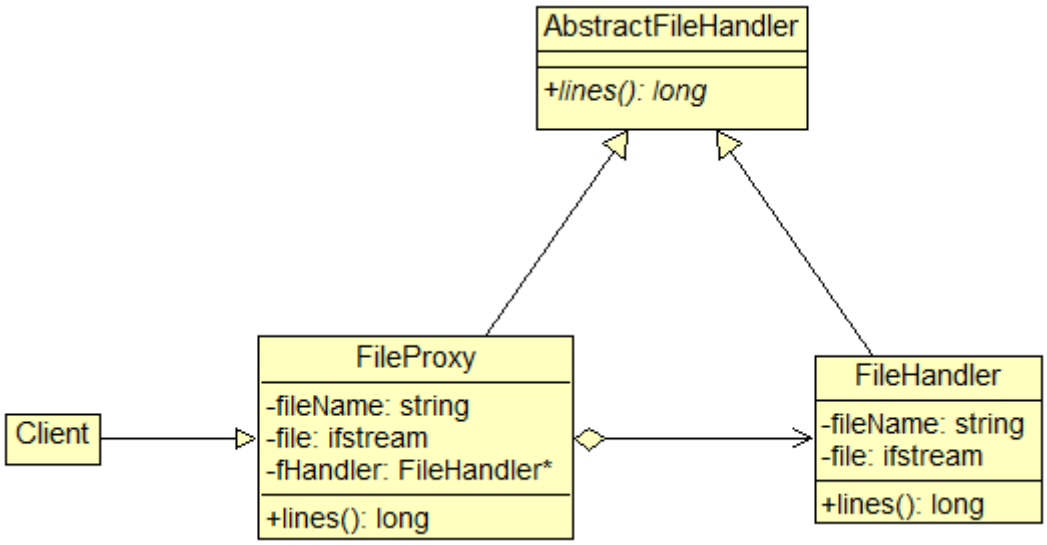


Subject definierar ett gemensamt (abstrakt) interface för RealSubject som är den egentliga implementationen av Subject, och Proxy. Klienten instansierar ett Proxy-objekt och programmerar mot det (abstrakta) gemensamma interfacet.

En *Virtual Proxy* är en annan typ av proxy-objekt som skapar andra objekt på begäran. Ett exempel kan vara en ordbehandlare och hanteringen av bild-objekt i ett dokument:

- När dokumentet öppnas skapas bara de bild-objekt som är initialt synliga. Att skapa samtliga bilder skulle kräva för mycket resurser både i tid och minne.
- Konceptet är att faktiskt instansiera endast de bilder som för tillfället är synliga i dokumentet. Detta genomförs genom att låta alla bilder representeras av ’virtual proxies’ som platshållare. Proxyn instansierar den egentliga bilden vid behov.
- Orbehandlarens klientkod arbetar mot det abstrakta interface (’Subject’ i föregående figur) som implementeras både av Proxyn och den egentliga bildklassen.

Exempel på en *on demand*-operation (exProxy):
Klassen FileHandler erbjuder en operation för att räkna antalet rader i fil. Klientkoden använder istället en proxy, FileProxy. Båda är deriverade från AbstractFileHandler.



```
long FileHandler::lines() {
	int nLine = 0;
	string s;
	while (getline(file, s))
		++nLine;
	// Reset the file
	file.clear(0);
	file.seekg(0);
	return nLine;
}
```

För att inte öppna den aktuella filen förrän det verkligen behövs skapar proxyn ett FileHandler-objekt ’on demand’:

```
long FileProxy::lines() {
	if (fHandler == 0)
		fHandler = new FileHandler(fileName);
	return fHandler->lines();
}
```

Klientkod:

```
AbstractFileHandler *f = new FileProxy("exProxy.cpp");
// doesn't open file yet
...
cout << f->lines() << endl; // open now!
```

En C++-variant på DP proxy kan användas i de fall då den abstrakta ’Subject’-klassen redan existerar och har ett interface som inte passar den tänkta proxyn, och dessutom inte kan ändras. Klienten använder Proxy som om den vore en pekare. Detta beteende kan implementeras genom överlagring av operatorerna `*` och `->` som i följande exempel (exProxy2)

```
class FileProxy { // Proxy
public:
	FileProxy(string fName)
		: fileName(fName), fHandler(nullptr)
	{ }
	// Overloaded operators
	FileHandler* operator->();
	FileHandler&  operator*();
private:
	FileHandler* openFile();
	FileHandler* fHandler; // RealSubject
	string fileName;

};
```

```
FileHandler* FileProxy::openFile() {
	if (fHandler == 0)
		fHandler = new FileHandler(fileName);
	return fHandler;
}

FileHandler* FileProxy::operator->() {
	return openFile();
}

FileHandler& FileProxy::operator*() {
	return *openFile();
}
```

Klientkod:

```
FileProxy fp("exProxy2.cpp"); // no file open yet
                                                  // ...
cout << fp->lines() << endl;  // open now
// cout << (*fp).lines() << endl; // alternative
```

’Copy-on-Write’

En annan optimering som är relaterad till att skapa objekt ’on demand’ är ’Copy-on-Write’. Att kopiera stora objekt är resurskrävande. En proxy kan erbjuda en copy-operation som istället för att skapa en kopia av ett objekt, endast ökar värdet på referensräknare och returnerar en referens till ett redan befintligt objekt.

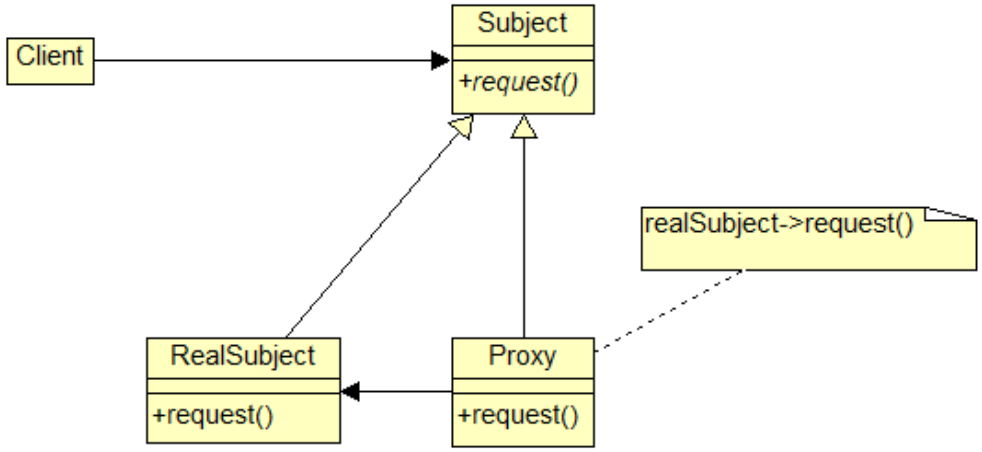
- Om objektet inte kommer att ändras så har en kostsam kopieringsoperation sparats in.
- Först när (och om) objektet ändras utför proxyn den fysiska kopieringen och referensräknaren minskas. När referensräknaren får värdet noll deallokeras objektet.

’Smart Reference’

En proxy som har en referensräknare till RealSubject och som skapar objektet ’on demand’ och deallokerar det när referensräknare är noll kan ersätta en vanlig pekare. En proxy med dessa egenskaper brukar kallas en *’Smart reference’*.

Sammanfattning DP Proxy

- Struktur



- Syfte
  - Att erbjuda en ’stand-in’ för ett annat objekt i syfte att kontrollera access till det.

- Deltagare
  - *Proxy*
  - Innehåller en referens som tillåter Proxy att accessa det verkliga subjektet, RealSubject.
  - Erbjuder samma interface som Subject så att Proxy kan ersätta RealSubject.
  - Kontrollerar access till RealSubject och kan också ansvara för att skapa och förstöra det.
  - En *remote proxy* ansvarar för att skicka vidare ett anrop med argument till RealSubject som kan vara i en annan adressrymd.
  - En *virtual proxy* skapar RealSubject ’on demand’ och kan också lagra information om det, t.ex. storleken på en bild.
  - En *protection proxy* kontrollerar att rätt behörighet finns för anropet.
  - *Subject*
  - Definierar det gemensamma interfacet för RealSubject och Proxy så att Proxy kan användas var som helst där ett RealSubject förväntas.
  - *RealSubject*
  - Definierar det verkliga objektet som Proxy representerar.

- Konsekvenser
  Proxy DP inför ett mellanliggande skikt i kommunikationen när en klient accessar ett objekt. Detta extra skikt kan utnyttjas på flera sätt:
  - En *remote proxy* bidrar till att gör accessen till ett objekt i en annan adressrymd transparent för klienten.
  - En *virtual proxy* kan bidra med optimeringar som t.ex. att skapa ett objekt ’on demand’.
  - En *protection proxy* utför accesskontroller.
  - ’*Copy-on-Write*’ kan minska resursförbrukningen vid kopiering av stora objekt.
  - En *’Smart Reference*’ kan ersätta explicita allokera- / deallokera operationer.

- Använd DP Proxy när…
  - du vill ha ett lokalt objekt som representerar ett objekt i en annan adressrymd.
  - du vill skapa objekt ’on demand’.
  - du vill kontrollera att en klient har rätt behörighet vid access till ett objekt.
  - du vill utföra kompletterande ’actions’ i samband med access till ett objekt.