Mediator Pattern

I4SWD-01

Gruppe 02

201404118 Anders W. Birkelund

201271001 Rune D. Rask

201405166 Jonas R. Hartogsohn

Indhold

Introduktion

Formål

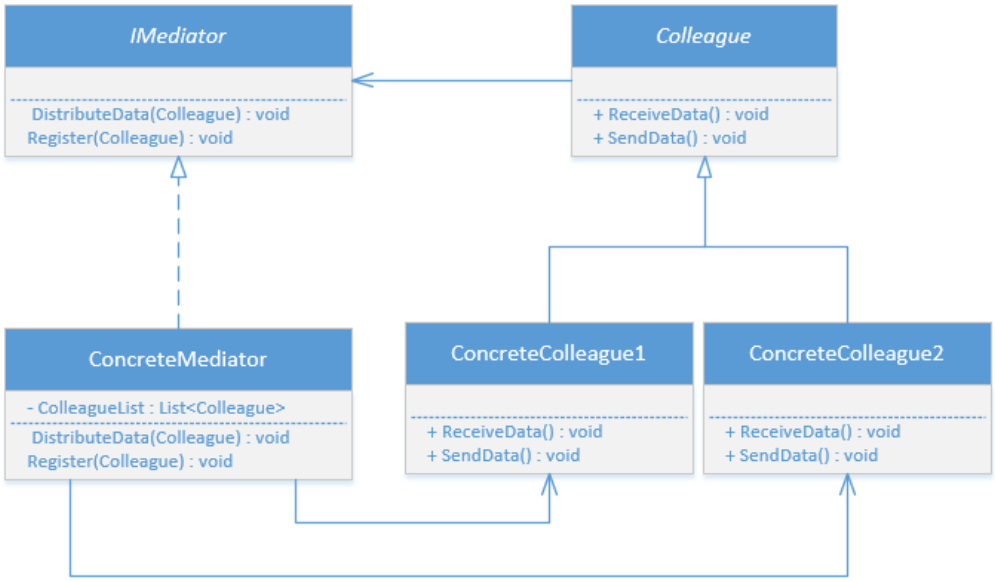
Mediator-mønstret benyttes til at nedsætte koblingen i et system, hvor mange komponenter skal kommunikere med hinanden på forskellig vis. I det givne system ville koblingen forhøjes markant ved tilføjelse af flere komponenter, og derved bliver systemet utroligt svært at vedligeholde. Et sådant komplekst kommunikationsnetværk forsimples ved at introducere mønstret, hvor al kommunikation foregår via et Mediator-objekt, der står for at facilitere kommunikationen mellem komponenterne.[1]

Gang of Four deler deres design mønstre ind i tre grupper: Creational-, Structural- og Behavioral patterns. Mediator-mønstret er et Behavioral Pattern, da det står for kommunikationen og interaktionen mellem objekter.

Struktur

Mediator-mønstret består af følgende klasse:

- **IMediator** er et interface til den konkrete Mediator-klasse. Her deklareres de generelle metoder for en Mediator.
- **Colleague** er en abstract klasse, hvori de overordnede metoder for Colleagues defineres.
- **ConcreteMediator** implementerer IMediator-interfacet og metoderne heri.
- **ConcreteColleague** nedarver fra Colleague og overrider disse metoder.



1Generel UML klasse-diagram for Mediator

ConcreteColleague registrerer sig selv i Mediatoren via Register-metoden. Herved kender Mediator alle ConcreteColleagues i form af Colleague-typen. Når en ConcreteColleague skal sende data, gøres dette via Mediatoren, der distribuerer denne data. Skal den pågældende data fordeles til bestemte Colleagues, defineres denne logik i Mediatoren.

[1] http://www.codeproject.com/Articles/186187/Mediator-Design-Pattern

Fordele

Ved at benytte Mediator-mønstret, opfylder man Open/Close-princippet fra SOLID, da man ved at nedsætte den høje kobling opnår at objekternes afhængigheder bliver forsimplet. Dette er en klar fordel, når systemet skal videreudvikles, da man herved får muligheden for at udskifte komponenterne uden det har indflydelse på de andre objekter.

Konsekvenser

Når man benytter sig af Mediator-mønstret, skal man være opmærksom på ikke at lave en gudeklasse, med kendskab til alle objekter i systemet og med høj kompleksitet. I scenarier hvor flere forskellige typer subklasser, som skal benytte sig af Mediatoren, vil det være nødvendigt at have logik, som håndterer de individuelle typer. Dette kan medfølge kompleks og uoverskuelig kode. I sådanne tilfælde kan det være nødvendigt oprette flere typer Mediatorer, som håndterer kommunikation mellem hver de forskellige typer klasser. [2]

Sammenligning

Observer

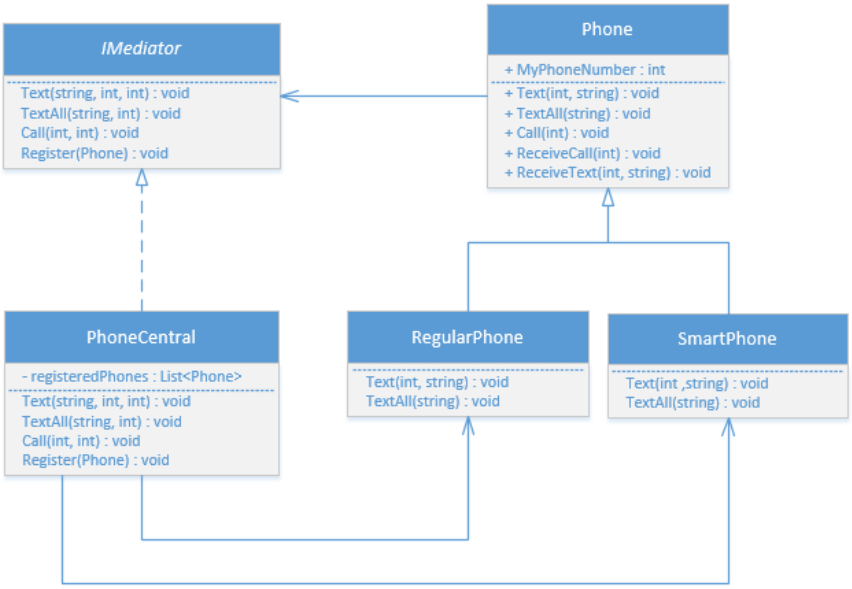
Til forskel fra Mediator, som faciliterer kommunikationen mellem eksisterende objekter i et system, introducerer Observer-mønstret Observer- og Subject-klasser til systemet. I dette mønster går kommunikationen kun én vej – fra Subject til Observer (alle Observere), hvorimod kommunikationen i Mediator kan være mere kompleks. Begge mønstre giver lav kobling i et givent system, da de benyttes til at simplificere afhængighederne.

Implementering

I vores egen implementering af Mediator-mønstret, har vi taget udgangspunkt i et system, hvor telefoner skal kommunikere med hinanden uden at være direkte forbundet. Her implementeres en PhoneCentral, som fungerer som vores Mediator. Telefonerne kan her sende beskeder og ringe til hinanden via PhoneCentral. Der oprettet to typer af telefoner, som hver overrider de nedarvede metoder fra super-klassen 'Phone'. Når en telefon skal sende en besked til en anden, bliver denne kommunikation dirigeret af logikken i Mediator.

Nedenfor ses et UML klasse-diagram over systemet.

[2] https://app.pluralsight.com/player?course=design-patterns-java-behavioral&author=bryan-hansen&name=design-patterns-java-behavioral-m6&clip=0&mode=live



2UML klasse-diagram over egen implementering

Konklusion

Når man står som softwareudvikler, og man skal udvikle et program med intern kommunikation imellem objekter, kan Mediator-mønstret være et godt mønster til at sikre lav kobling. Man skal dog overveje, hvordan objekterne skal kommunikere mellem hinanden, og om objekterne er ens.

I en løsning hvor kommunikationen skal være meget specifik, og objekterne er ens, er Mediator-mønstret perfekt. Skal man derimod lave et program, hvor kommunikationen er meget simpel, og objekterne varierer i funktionalitet, kan mønstre som observer være et bedre valg.