TALLINNA ÜLIKOOL
Informaatika osakond

PROGRAMMI ÜLESEHITUSE MUSTRID
Seminaritöö

Üliõpilane: Margo Poolak

Juhendaja: Jaagup Kippar

Tallinn 2006

Sisukord

1 Sissejuhatus

Allolev materjal on koostatud eesmärgiga laiendada programmeerija silmaringi kuidas leida häid lahendusi ja viise kuidas koodi kirjutada.
Programmeerimise juures me peame tihti mõtlema, et kuidas kirjutada hästi töötavat koodi. Tihti me ei mõtle, et kuidas ühte või teist probleemi lahendame. Me lihtsalt kirjutame programmi kood kuni see antud tingimustele tööle hakkaks ja probleemi lahendaks, mis ei ole kas kõige optimaalsem või mis on hiljem raskesti laiendatav. Programmi ülesehituse mustrid aitavad aru saada kuidas mingeid probleeme lahendada iseloomulikul viisil ning võimaldab koodi kirjutada nii, et see oleks võimalikult arusaadav ja loogiline.
Iga programm kujutab endast mingit kindlate meetodite kogumit ja loogikat sisaldavat üksust, mis täidab talle ette antud ülesandeid. Programmi ülesehituse muster on korduvalt täidetav lahendus, mis lahendab mingit korduvat probleemi programmi ülesehituses. See on programm, mis on koostatud teatud reeglite järgi, mis siis täidaks vajalike eesmärke.
Näidis koodid on toodud vastavalt keele eripärale: PHP5 skripti, C# või Javat kasutades või viidates.
Kuna palju materjali leidub internetist väga ja kõike ei ole võimeline läbi töötama, siis viitan nendele. Enamikel juhtudel on tegemist inglise keelse materjaliga. Seega olen lisanud lühendite loetelusse mõned inglise keelsete terminite tõlked, et lihtsustada inglise keelse materjali lugemist ja arusaamist.

2 Mis on programmi ülesehituse muster?

Mis asi on programmi ülesehituse muster? See on seaduspärasus. Et sellest aru saada võtame näite elust enesest. Ütleme nii, et iga päev me tegeleme mingisuguste probleemidega. Selle juures ei ole määravaks probleemi laad, vaid viis kuidas me seda lahendame. Probleemi laad ei ole määrav sellepärast, et ühte ja sama probleemi on võimalik lahendada erinevatel viisidel. Iseküsimus on see, et kuidas see probleemi lahendab. Kui see ei lahenda probleemi, siis leitakse teine lahenduskäik kuni sinnamaani, et probleem saab lahendatud. Lõpuks leitakse see üks viis kuidas probleemi lahendada võimalikult optimaalselt ja mugavalt.

Näiteks: mida teeb suudlev printsess konnaga?

Iga programm sisaldab endas n.ö. probleeme mida lahendada. Igale probleemile on mingi iseloomulik lahendus. Seda lahenduskäiku nimetatakse omakorda programmi ülesehituse mustriks. Programmi ülesehituse mustrit ei ole võimalik otse kasutada kuskil programmi koodis, vaid pigem see on kirjeldus kuidas lahendada sarnaseid probleeme erinevates olukordades.

Programmi seaduspärasus on osa programmi ülesehitusest kuidas objektid on omavahel seotud, mis moodi neid välja kutsutakse ja luuakse, kuidas nad omavahel suhtlevad ning kuidas on võimalik sarnaste probleemide korral koodi uuesti kasutada.
Seda kõike pakub programmi ülesehituse mustri seaduspärasused.

Kuidas jõuda kinnituseni, et tegemist on programmi mustriga? Järgnevalt kirjeldan ära terminid ja struktuuri, et see annaks ülevaate, mis on programmi muster ja mis mitte.

Idioom: see on viis kuidas me kirjutame koodi mingis kindlas keeles ja mis teeb mingit kindlat asja.
Konkreetne disain: lahendus, mis lahendab mingit kindlat probleemi aga ta ei ole üldistatav.

Standard disain: lahendab kõik sellelaadsed probleemid. Taaskasutust leidev kood.
Programmi muster: viis kuidas lahendada terve klass samu probleeme. See ilmneb peale mitme korra standard disaini kasutamist ja erinevate lahenduste korral püsib sarnane programmi ülesehituse muster. [1]

3 Programmi ülesehituse mustri dokumenteerimine

Dokumenteerimine peaks sisaldama endas piisavalt informatsiooni selle kohta, mis kontekstis ja kuidas konkreetne muster toimib.
Mis on konkreetse mustri eripära ja mis moodi ta toimib.
Kontekst kus seda kasutatakse ja soovituslik tulemus.

Mustri nimi ja klassifikaator: Igal mustril peaks olema unikaalne nimi, mis seda identifitseeriks ja viitaks.
Eesmärk/probleem: Määratleb ära mustri eesmärgi ja põhjuse miks seda kasutada.
Teatakse ka kui: Mustril võib olla ka teisi nimesid.
Motivatsioon: Siin tuleks kirjeldada, et mis laadi probleemides, kontekstis või millal antud mustrit võiks kasutada.
Rakendatavus: Millistest situatsioonides on konkreetne programmi ülesehituse muster kasutatav.
Struktuur: Klassi ja seos diagramm.
Osalejad: Nimekiri klassidest ja objektidest mida antud mustris kasutatakse ja mis on nende roll.
Koostöö: Kirjeldatakse kuidas klassid ja objektid mustris omavahel suhtlevad.
Tagajärjed: Siin kirjeldatakse tulemustest, kõrval efektidest ja kompromissidest kasutades seda mustrit.
Rakendamine: Kirjeldab kuidas mustrit rakendada ja kuidas see on lahendatud. Sisaldab endas tehnikat ja vihjeid kuidas seda mustrit kasutada.
Näidis kood: Näide kuidas programmi koodi kasutatakse
Kasutusjuhud: Viited reaalsetele näidetele, lahendustele kus konkreetset mustrit on kasutatud
Sarnased mustrid: Siin kirjeldatakse teisi mustreid, mis on sarnased või millegi poolest seotud. Samuti mustreid mida võib kasutada koos selle mustriga või selle asemel. Samuti tuuakse siin välja sarnaste mustrite erinevused. [2]

4 Tutvustavate programmi mustrite valik

Mustreid võib klassifitseerida erinevate kriteeriumite järgi. Enamasti liigitakse mustreid probleemide järgi mida nad lahendavad.

GoF [1] programmi ülesehituse mustrid on võetud kõigi teiste seaduspärasuste alustalaks. Nemad on oma programmi ülesehituse mustrid jaotanud kolme gruppi:
Loomise seaduspärasus (Creational patterns), Struktureeritud mustrid (Structural patterns) ja Käitumuslikud mustrid (Behavioral patterns). [3]

Minu töös käsitletakse lisaks nendele veel 4 liiki seaduspärasuste klassiga:
Fundamentaalsed mustrid (Fundamental patterns), Korduvad mustrid (Concurrency patterns), Sündmusi töötlevad mustrid (Event handling patterns) ja Arhitektuursed mustrid (Architectural patterns). [4]

4.1 *Loomise seaduspärasus (Creational patterns)*

Programmi ülesehituse seaduspärasus, mis tegeleb objekti loomisega konkreetses situatsioonis. Kui tavaliselt võib objekti loomine lisada keerukust ülesehituses, siis „loovad ülesehituse mustrid" lahendavad selle probleemi kontrollides objektide loomist.
Iga kord kui konkreetne meetod on välja kutsutud kontrollitakse mingi „lipu" seisundit. Kui see on valmis, siis tagastatakse, kui mitte siis käivitatakse vastav toiming tulemuse saamiseks.[5]

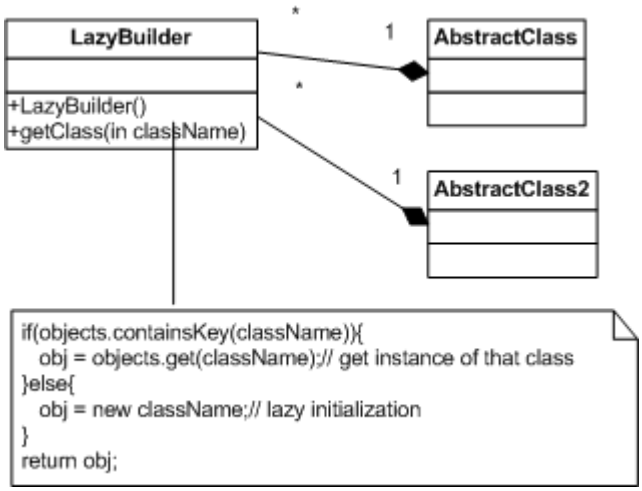
4.1.1 Lazy initialization pattern

Eesmärk/probleem: Viivitada objektide loomist seni kuni seda ei ole vaja kasutada.

Motivatsioon: Muuta süsteem paindlikumaks ressursside kasutamisel, kuna mingi objekti loomine võib olla liiga ressursi nõudlik.

Rakendatavus: Kujundaja tagab paindliku alternatiivi alamklassidele, et laiendada oma funktsionaalsust. [6]

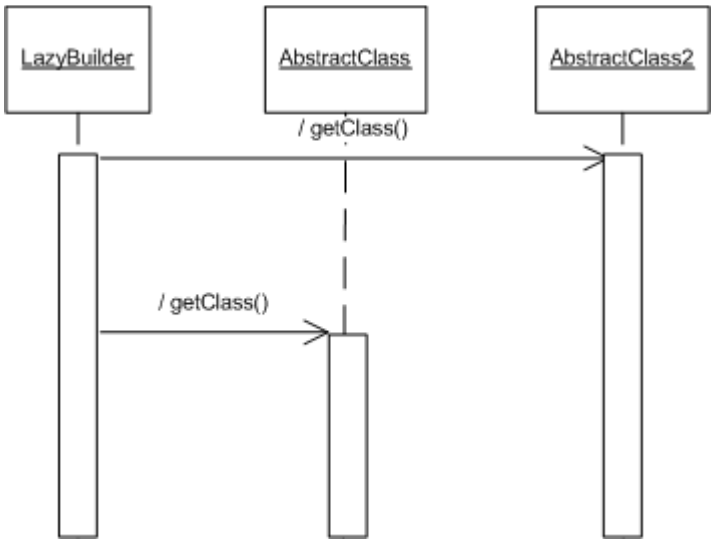
Struktuur: Klassi ja seos diagramm.



Osalejad: Klassid ja objektid mida antud mustris kasutatakse ja mis on nende roll:

- LazyBuilder
  - klass, mis haldab klasside välja kutsumist. Kui konkreetset klassi on vaja ning seda veel ei eksisteerita, siis see luuakse ja tagastatakse
- AbstractClass, AbstractClass2
  - Abstraktne klass mida välja kutsuda

Koostöö: Kirjeldatakse kuidas klassid ja objektid mustris omavahel suhtlevad.



Rakendamine: Kirjeldab kuidas mustrit rakendada ja kuidas see on lahendatud. Sisaldab endas tehnikat ja vihjeid kuidas seda mustrit kasutada.

Kasutusjuhud:

Andmebaasi päringu tegemine ja ühenduse loomine. Üsna tihti on nii, et luuakse andmebaasi ühendus ilma, et seda kasutatakse. Samas ei arvestata, et andmebaasi ühenduse loomine võib olla üsna ressursi nõudlik. Selle jaoks ongi hea kasutada konkreetset mustrit, mis looks alles siis andmebaasi ühenduse kui soovitakse teha konkreetset päringut.

```
<?php
/*
* Define the variables
*/
define('_SERVER_','localhost');// server name
define('_USER_','www');// user name
define('_PASSWORD_','tere');// password
define('_DB_','test');// database name
define('_DEBUG_',false);// do we want to debug
/*
* This is Lazy initialization pattern class for database
* It makes only then connection when the query is requested
* It uses adodb packages
*/
class db
{
    protected $db;// database connection handler
    protected $db_type; // database type
    protected $counter=0; // counts the querys
    /*
    * @string to set database type: mysql, mssql, oracle
    */
    function __construct($db_type)
    {
        $this->db_type = $db_type;
    }
    /*
    * This method makes the connection to the database
    */
    public function connect()
    {
```

```
            $this->db = &NewADOConnection($this->db_type);
            $this->db->Connect(_SERVER_, _USER_, _PASSWORD_, _DB_)
          or die("Connection to mysql database failed!! ");
            $this->db->debug = _DEBUG_;
            $this->db->SetFetchMode(ADODB_FETCH_ASSOC);
        }
        /*
        * This is method to make the database query
        * @sql is sql syntax string
        */
        public function query($sql)
        {
            if(is_object($this->db))
            {
                $rs = $this->db->Execute($sql);
                $this->counter++;
                return $rs;
            }
            else
            {
                $this->connect();
                $this->query($sql);
            }
        }
        /*
        * @returns number of querys made to the database
        */
        public function getQueryCount()
        {
            return $this->counter;
        }
}
/*
* Test case
*/
$mssql = new db('mssql');// db object set but connection not made
$mysql = new db('mysql');// db object set but connection not made

// if the connection not done then make it
$mysql->query("SET NAMES 'utf8'");
// connection exists just make the query and count it
$mysql->query("SElECT * FROM 'boo'");

echo $mssql->getQueryCount();// returns 0
echo $mysql->getQueryCount();// returns 2
?>
```

4.1.2 Anonymous subroutine objects pattern

Eesmärk: Konkreetne muster on väga platvormist kinni. Internetis leiduv materjal räägib Perl5 omapärast, et konstruktor tahab, et kõik muutujad, mis talle edasi antakse tuleb manuaalselt ise määrata. Selle jaoks tehti tsükkel, mis siis iga kord käivitati koos objekti loomisega, et parameetreid edasi anda.

Rakendamine:

PHP-s on struktuurne keel ja seetõttu on probleeme objektide loomisega. Et kood oleks loetavam paneme me need enamasti eraldi failidesse ja paneme faili nimeks sama mis klassi

nimigi, et oleks lihtsam hallata. Nüüd kui me soovime seda klassi käivitada, siis see eeldab, et see klass on sisse loetud.

Näidis kood:

PHP5

Eksisteerib fail Anonymouse.php

```
<?php
/*
* Anonymouse class
* method returns hello
*/
class Anonymouse
{
    /*
    * @return string hello
    */
    function hello()
    {
        echo"hello";
    }
}
?>
```

Et seda klassi saaks kutsuda välja mingis failis, siis tuleb see kõigepealt sisse lugeda ja seejärel käivitada.

```
<?php
require_once('Anonymouse.php');
$a = new Anonymouse();
$a->hello();
?>
```

Iseenesest ei ole seda raske teha aga kui tegeleda tuleb rohkem kui 10 või enama faili ja klassiga mida sisse lugeda, siis võib manuaalselt failide lisamine koodi tunduda väga tülikas. Samuti kaob dünaamilisus ja laiendatavus.
Selle jaoks on PHP5 loodud meetod **__autoload($class_name)** [8], mis siis saab parameetriks klassi nime mida parasjagu välja kutsutakse. See lisab tunduvalt dünaamilisust koodi kasutamisel.

Ülal toodud näidis probleemi saab lahendada järgnevalt.

Kasutusjuhud:

```
<?php
define('CLASS_EXT', '.php');// set file extention
$requiredPaths = array('.','templates');
/*
* If class is called out first time then first require a file
* If u call out new class object then it will run it here
* It calls out function using that searches the class file from predefined
directories
*/
if(!function_exists("__autoload"))
{
    function __autoload($class_name)
```

```
    {
        if(using($class_name)==0){
            exit("Could'nt load class: ".$class_name);
        }
    }
}
/*
* Requires a file using predefined direcotryes from where to include
* Searches the class file from the predefined directories
*/
function using($file)
{
    global $requiredPaths;
    if(!$requiredPaths)
        exit('Please use a <b>requiredPaths</b> array to set require files
path');
    foreach($requiredPaths as $path)
    {
        $req = $path.DIRECTORY_SEPARATOR.$file.CLASS_EXT;
        if(is_file($req)){
            require_once($req);
            return 1;
        }
    }
    return 0;
}
$a = new Anonymouse();
$a->hello();
?>
```

Nüüd on võimalik antud leheküljel välja kutsuda kõik klassid mida iganes vaja muretsemata, kas ta on laetud lehele või mitte, sest see toimub juba automaatselt klassi välja kutsumisel.

Sarnased mustrid: Lazy initialization pattern

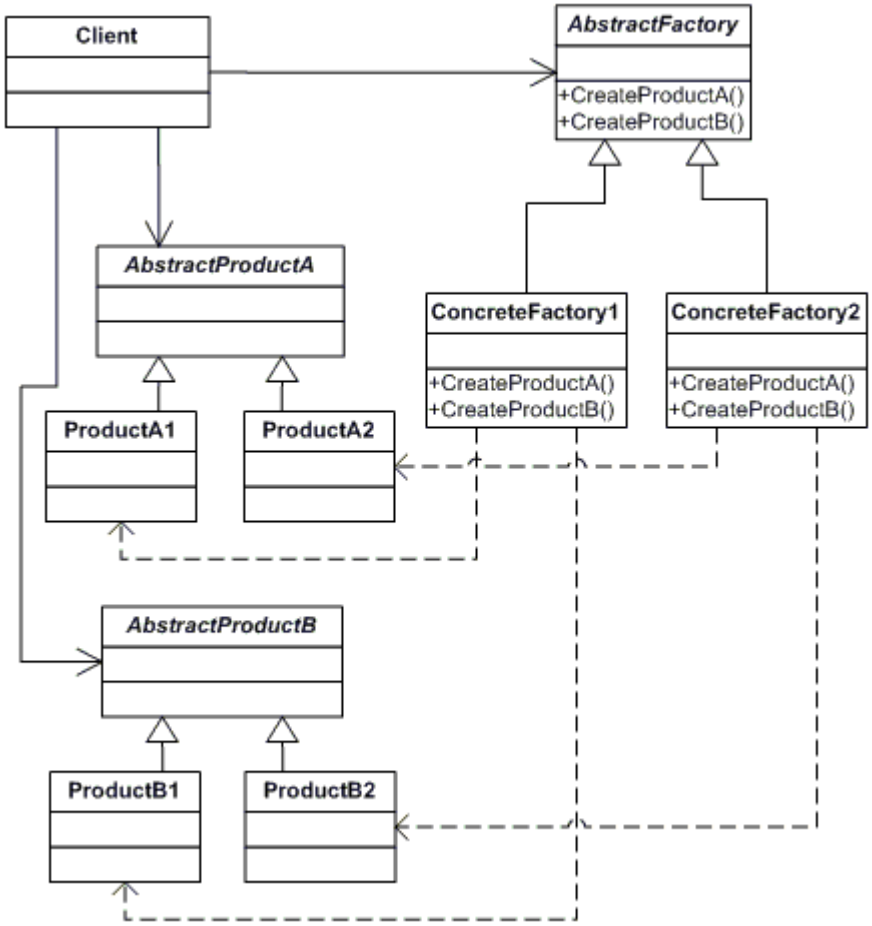
4.1.3 Abstract factory pattern

Eesmärk/probleem: Abstraktne tehase muster pakub võimalust kapseldada endasse grupi individuaalseid tehaseid millel on siis mingi ühine kujundus või ehitus.

Motivatsioon: Tavaliselt kliendi programm loob rakenduse abstraktsest tehasest ja seejärel kasutab üldist liidest, et luua konkreetset objekti, mis on osa ühisest kujundusest.
Klient ei tea ega huvita, mis konkreetne objekt kutsutakse välja nendest sisemistest tehastest, kuna ta ise kasutab ainult üldist liidest

Rakendatavus: Võimaldab luua mitmeid erinevaid liideseid, et luua ühest konkreetsest liidesest erinevad väljundid. Näiteks veebileht koosneb vaatest, vaade võib välja näha erinevad: XML, HTML, Excel jne.

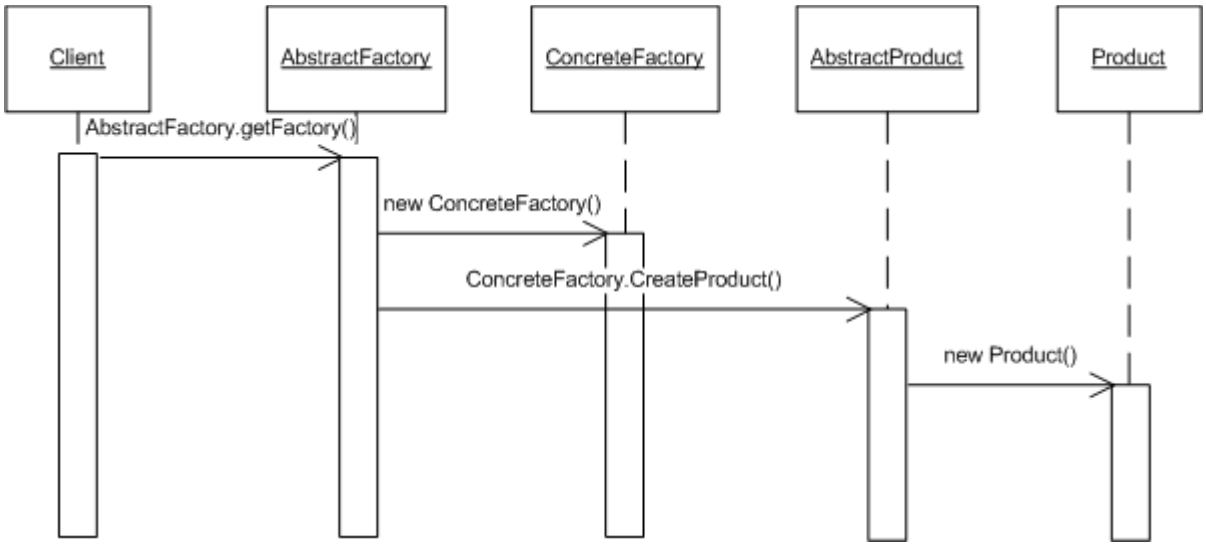
Struktuur: Klassi ja seos diagramm



Osalejad:

- Client
  - Suhtleb ainult läbi abstraktsete klassi liideste nagu abstraktne vabrik (AbstractFactory) ja abstraktne toode (AbstractProduct)
- AbstractFactory
  - Abstraktne vabrik, milles on vajalikud meetodid, mis loovad abstraktseid tooteid.
- ConcreteFactory
  - Konkreetne vabrik, milles on vajalikud meetodid loomaks konkreetseid tooteid.
- AbstractProduct
  - Abstraktne toode koos vajalike meetoditega
- Product
  - Defineerib konkreetset toote objekti, mis luuakse vastavalt konkreetse vabriku poolt.
  - Abstraktne toode rakendab selle

Koostöö: Kirjeldatakse kuidas klassid ja objektid mustris omavahel suhtlevad.



Rakendamine: Kirjeldab kuidas mustrit rakendada ja kuidas see on lahendatud. Sisaldab endas tehnikat ja vihjeid kuidas seda mustrit kasutada.

Näidis kood:

[vt. http://en.wikipedia.org/wiki/Abstract_factory]
[vt. http://www.dofactory.com/Patterns/PatternAbstract.aspx#_self1]

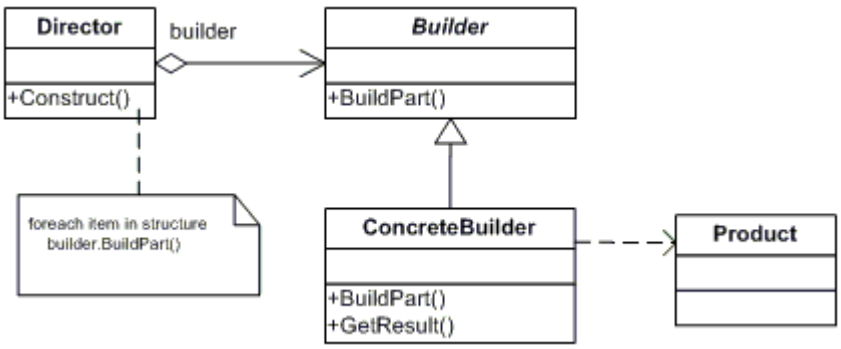
4.1.4 Builder pattern

Eesmärk/probleem: Kujundaja tagab paindliku alternatiivi alamklassidele, et laiendada oma funktsionaalsust.

Motivatsioon: Muuta süsteem paindlikumaks ressursside kasutamisel, kuna mingi objekti loomine võib olla liiga ressursi nõudlik.

Rakendatavus: Programmi ülesehituse muster kus luuakse keeruliste objektide kogum ühest alg objektist. Konkreetne objekt võib sisaldada endas erinevaid osi, mis siis osalevad iseseisvalt, et luua igasuguseid keerulisi objekte, mis on siis edasi antud läbi liidese mis ise on osa Abstraktsest ehitaja klassist.

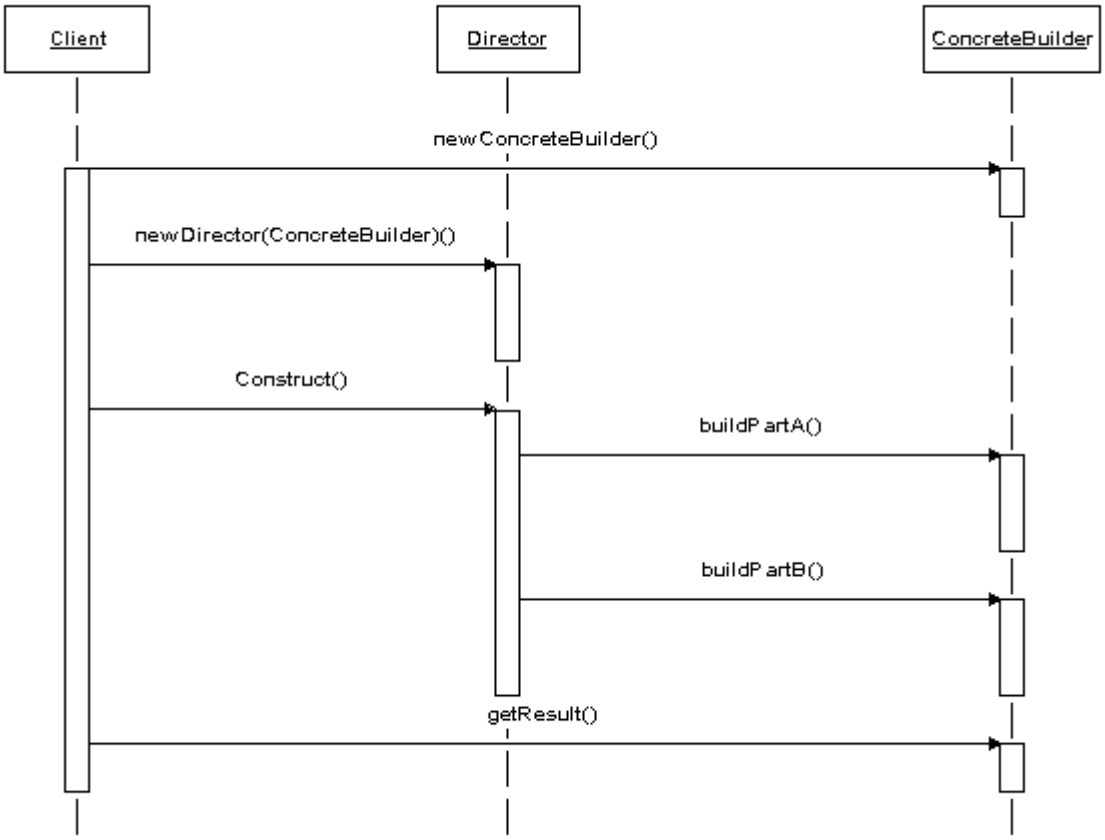
Struktuur: Klassi ja seos diagramm



Osalejad:

- Director
  - Tekitab objekti kasutades ehitaja (Builder) liidest
- Builder
  - Kirjeldab abstraktset liidest, et luua toote (Product) objekti kasutades uusi „tooteid”
- ConcreteBuilder
  - Sisaldab liidest, et pärida konkreetset „toodet”
  - Loob ja paneb kokku „produkte” kasutades ehitaja (Builder) liidest
  - Defineerib ja jälgib esitlust mida ta looma hakkab
- Product
  - Sisaldab endas konkreetseid „toodet” mida luuakse süsteemis

Koostöö: Kirjeldatakse kuidas klassid ja objektid mustris omavahel suhtlevad.



Rakendamine: Kirjeldab kuidas mustrit rakendada ja kuidas see on lahendatud. Sisaldab endas tehnikat ja vihjeid kuidas seda mustrit kasutada.

Kasutusjuhud:

PHP5

```
<?php
class Builder
{
    /*
    * Run the application
    * Build up the system
```

```
     */
     public function __construct()
     {
          $request = new TRequest;
          // requested page name Module_PageName
          $requestedPage = $request->getRequestedPage();
          /*
          * If the requestedPage variable is not empty
          */
          if(!!$requestedPage)
          {
               // activate module before page
               $module = new $request->getModule();
               // sets the request object to the module
               $module->setRequest($request);

               $view =   new $requestedPage;// activate page
               $view->setRequest($request);// sets the request to the page vie
w
               /*
               * In here we implement module to the page instance
               * This is a bridge to eachother
               */
               // implement module instance to the page
               $view->setModule($module);

               $view->execute();// execute the page
          }
     }
}
?>
```

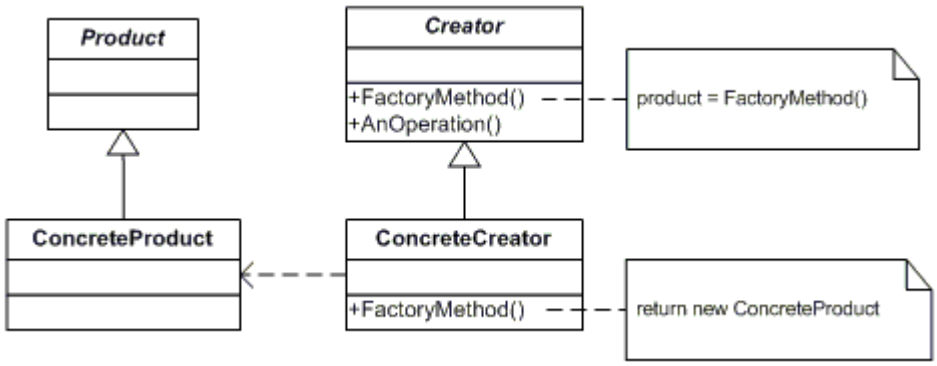
4.1.5 Factory method pattern

Eesmärk/probleem: Tegeleb samuti objektide loomisega. Selle mustri probleemi eripära on sellest, et objekti loomisel ei spetsifitseerita konkreetset klassi mida luuakse. Selle jaoks defineeritakse eraldi meetod, mis siis kutsub välja alamklassi meetodi, mis loob vajaliku objekti.

Motivatsioon: Lahendab selliseid olukordi, kus on sarnaste objektide puhul vaja kasutada erinevaid omadusi.

Rakendatavus: Kujundaja tagab paindliku alternatiivi alamklassidele, et laiendada oma funktsionaalsust.

Struktuur: Klassi ja seos diagramm



Osalejad:

- Product
  - Klass, mis defineerib konkreetse produkti omadused
- Creator
  - Loob liidese tootest ja tehasest, mis sellega tegeleb
- ConcreteProduct
  - Konkreetne toode, mis saab oma omadused vastavast toote klassist.
- ConcreteCreator
  - Konkreetne looja, mis siis rakendab mingi tegevuse

Rakendamine: Kirjeldab kuidas mustrit rakendada ja kuidas see on lahendatud. Sisaldab endas tehnikat ja vihjeid kuidas seda mustrit kasutada.

Näidis kood:

[vt. http://www.dofactory.com/Patterns/PatternFactory.aspx#_self1]

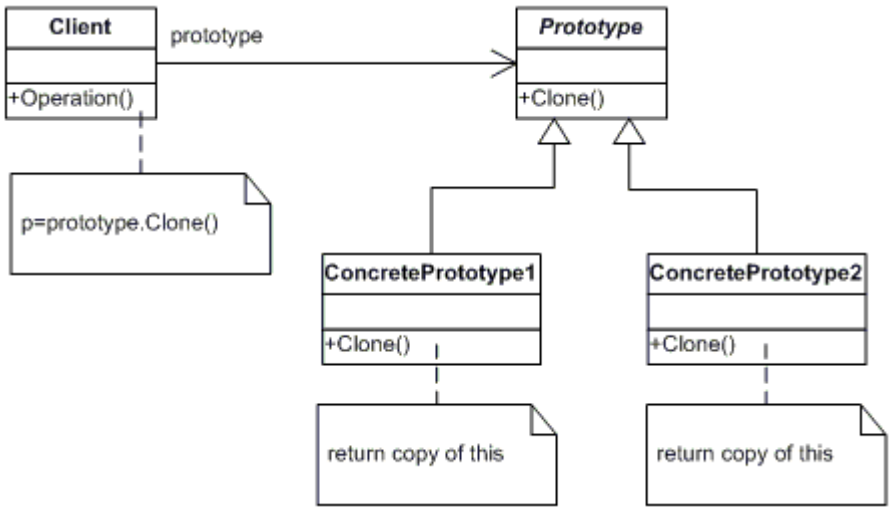
4.1.6 Prototype pattern

Eesmärk/probleem: Luua uusi objekte kloonimise teel.

Motivatsioon: Prototüübi programmi ülesehituse mustrit kasutatakse tarkvara arenduses, et uue keerulise või aega nõudva objekti loomiseks kasutatakse kindlaks tehtud prototüüp instantsi, mis on juba valmis tehtud ja millest tehakse modifitseeritud koopiaid.

Rakendatavus: Selle asemel, et kirjutada koodi, mis kutsub välja uue operaatori mingis kindla nimega klassi nimele kutsub välja kloonimise meetodi enda sees, mis siis kutsub välja vabriku meetodi koos parameetriga koos vastava klassi nimega, mida soovitakse väljastada, mis siis sisaldab endas juba teist tüüpi programmi ülesehituse mustrit.

Struktuur: Klassi ja seos diagramm



Osalejad:

- Client
  - Loob uue objekti kutsudes välja prototüübi enda klooni
- Prototype

    - Deklareerib liidese, et iseennast kloonida
- ConcretePrototype
    - Kasutab meetodit, et luua kloon iseendast

Rakendamine: Et kasutada sellist lahendust tuleb luua abstraktne virtuaalne klass, mis sisaldab endas clone() – kloonimise meetodit. Ükskõik, mis klass vajab „kuju muutvat konstruktori" oskusi, mis tuletab ennast abstraktsest peaklassist.

Näidis kood:

See kood demonstreerib prototüüp seaduspärasust milles luuakse uued objektid kopeerides juba eksisteerivaid objekte (prototüüpe) sellest samast klassist.

[vt. http://www.dofactory.com/Patterns/PatternPrototype.aspx#_self2]

Kasutusjuhud: Reaalselt kasutatakse prototüüp ülesehituse mustrit javascriptis.

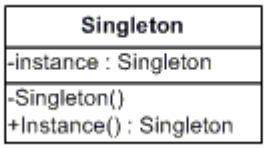
4.1.7 Singleton pattern

Eesmärk: Singleton ehk üksik objekt. See on programmi ülesehituse muster, et keelata objekti mitme kordne välja kutsumine.

Motivatsioon: Kasulik on see sellisel juhul, kui on vaja ainult ühte objekti, et süsteemi hallata. See muudab ka süsteemi kasutamise tunduvalt efektiivsemaks, kui mingit suurt objekte ei saaks uuesti ja uuesti välja kutsuda, vaid tagastatakse juba olemasolev.

Rakendatavus: Kasutatakse koos teiste programmi ülesehituse mustritega nt. Factory method pattern või prototype patterns. Kui näiteks on vaja luua mingi globaalne liides, mis siis tagastab ühte ja sama asja erinevatele programmi osadele ilma, et ta seda uuesti looma hakkaks.

Struktuur: Klassi ja seos diagramm



Osalejad:

- Singleton
    - Vastutab iseenda unikaalse objekti loomise ja haldamise üle
    - Defineerib Instance meetodi, mis lubab kliendile ligipääsu unikaalsetele klassi meetoditele.

Rakendamine:

Allolev näidis näitab kuidas kasutada Singleton pattern'i, mis tagab selle, et ainult üks instants on võimalik luua konkreetsest klassist.

Näidis kood:

[vt. http://www.dofactory.com/Patterns/PatternSingleton.aspx#_self2]

4.2 *Struktureeritud mustrid (Structural patterns)*

Struktureeritud mustrid kirjeldavad kuidas objektid ja klassid on omavahel seotud ja kuidas seda teha suures struktuuris.
Konkreetne muster peaks tagama, et süsteemi muudatus korral ei nõuta objektide omavahelise seose katkestamist.
Kuidas eristada objekti ja klassi mustreid omavahel? Objekti muster kirjeldab kuidas objekte luua ja siduda, et luua suurem ja keerulisem struktuur.
Klassi muster kirjeldab abstraktsiooni kasutades selleks pärilikust, et kuidas saab seda klassi kasutada, et tagada kasulik programmi liides.

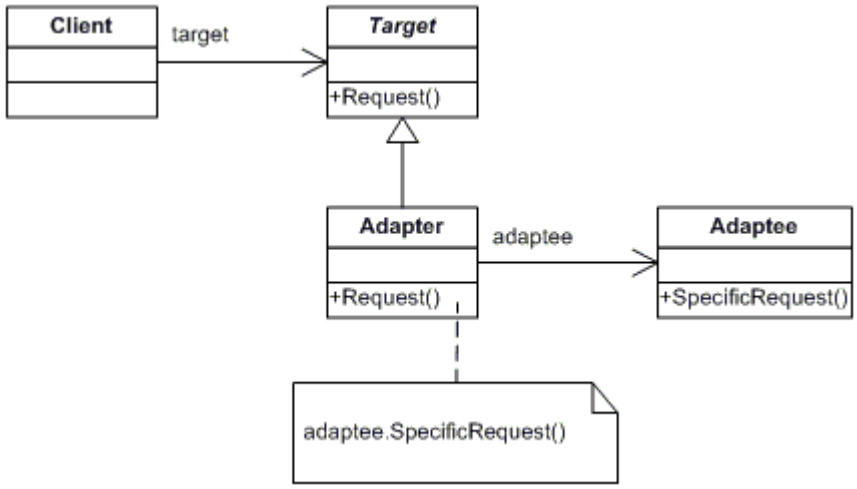
4.2.1 Adapter pattern

Eesmärk/probleem: Kasutatakse sellistel juhtudel kus on vaja, et kaks täiesti erinevat liidest saaksid suhelda omavahel. Ühenduskohta nimetakse „adapteriks".

Motivatsioon: Loob võimaluse pärida konkreetse meetodi läbi mingi teise klassi erinevat meetodit kust on oodata n.ö. õiget tulemust.

Rakendatavus: Konkreetne muster konverteerib ühe klassi liidese teise klient klassi liidese ootustele.

Struktuur: Klassi ja seos diagramm



Osalejad:

- Target
  - Kirjeldab oma haldusalale spetsiifilist liidest mida „Client" kasutab
- Adapter
  - Kohandab enda jaoks liidese „Adaptee", et „Target" liides oskaks seda kasutada
- Adaptee

    - Mingi olemasolev kasutajaliides millele on vaja ligipääsu ja kasutamist
- Client
    - Läbi „Target” objekti teeb koostööd vastavate objektidega

Rakendamine: Kirjeldab kuidas mustrit rakendada ja kuidas see on lahendatud. Sisaldab endas tehnikat ja vihjeid kuidas seda mustrit kasutada.

Näidis kood:

[vt. http://www.dofactory.com/Patterns/PatternAdapter.aspx#_self1]

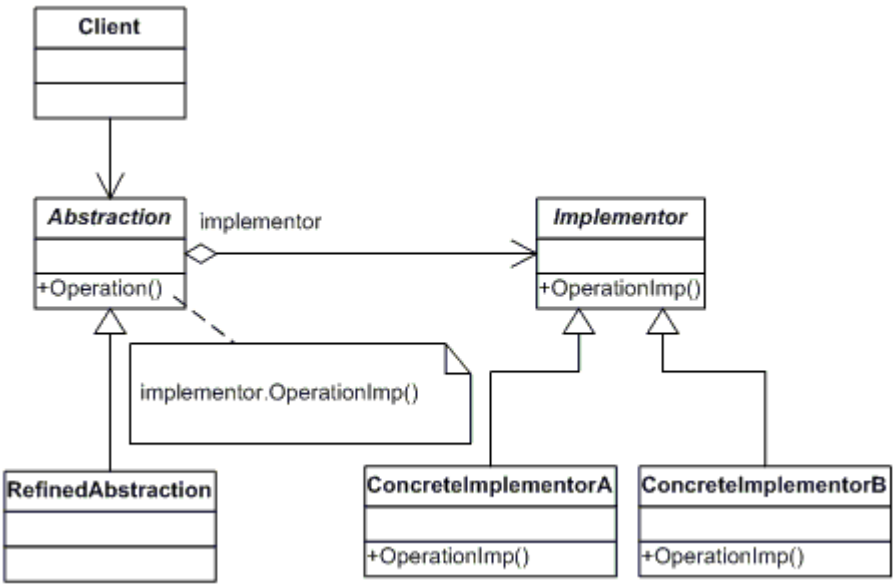
4.2.2 Bridge pattern

Eesmärk/probleem: Kasutatakse, et eraldada omavahel liidest ja rakendust.

Motivatsioon: See annab paindlikkuse, et arendada mõlemat keskkonda eraldi. Kasutatakse tihti, et siduda või pigem eraldada omavahel GUI[2] loogika andmete pärimise loogikast.

Rakendatavus: Nt. Web rakendustes kasutatakse, et eraldada omavahel back-end[3] Andmete pärimine, andmebaas) ja front-end[4] (kasutaja liides, mida kasutaja näeb HTML).

Struktuur: Klassi ja seos diagramm



Osalejad:

- Abstraction
    - Defineeritakse abstraktne kasutajaliides
    - Haldab seoseid „Implementor” tüüpi objektidele
- RefinedAbstraction
    - Laiendab abstraktsiooni (Abstraction) klassi
- Implementor
    - Sisaldab endas erinevaid klasse mille sees on mingid konkreetsed meetodid mida siis vajadusel kirjutatkse üle

- ConcreteImplementor
    - Konkreetne alamklass kus defineeritakse ära mis moodi ta mingi meetodi korral käitub

Koostöö: Kirjeldatakse kuidas klassid ja objektid mustris omavahel suhtlevad.

Rakendamine: Kirjeldab kuidas mustrit rakendada ja kuidas see on lahendatud. Sisaldab endas tehnikat ja vihjeid kuidas seda mustrit kasutada.

Näidis kood:

[vt. http://www.dofactory.com/Patterns/PatternBridge.aspx#_self1]

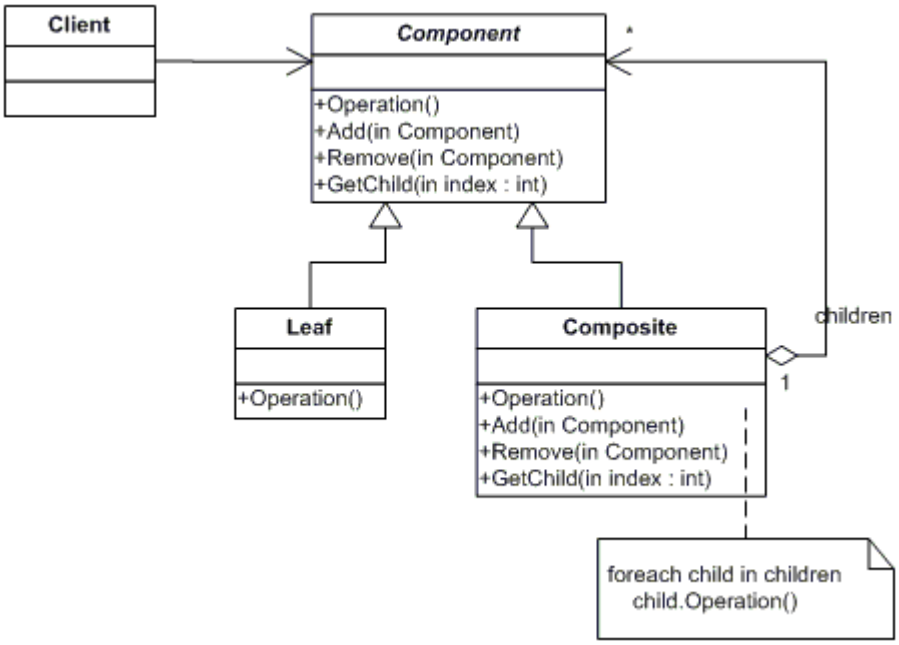
4.2.3 Composite pattern

Eesmärk/probleem: Loob hierarhilise objektipuu. Komponentiseerimine, ühe komponendi sisse pannakse teine jne. kuni siis algkomponent lõpetab alustatu. Võimaldab manipuleerida ühte liidest nii nagu oleks tegemist terve grupiga.

Motivatsioon: Näiteks muuta mingi kuju akna, et täita kogu ekraan. Samas oleks veel mugavam teha seda kõikide olemasolevate akendega.

Rakendatavus: Kui tegemist on mitme sarnase objektiga ja neil on enamvähem sarnane kood nende haldamiseks.

Struktuur: Klassi ja seos diagramm



Osalejad:

- Client
    - Töötleb objektide koosseisu läbi komponent (Component) kasutajaliidese
- Component

    - Deklareerib objektide kooseisus kasutajaliidese
- Composite
    - Defineerib järglastega komponentide käitumist
    - Salvestab järglastest komponendid
    - Rakendab komponent (Component) kasutajaliideses olevate järglaste tegevusi
- Leaf
    - Esindab lehe objekti. Lehel ei ole järglasi
    - Defineerib primitiivsete objektide kooseisu käitumist

Rakendamine: Kirjeldab kuidas mustrit rakendada ja kuidas see on lahendatud. Sisaldab endas tehnikat ja vihjeid kuidas seda mustrit kasutada.

Näidis kood:

[vt. http://www.dofactory.com/Patterns/PatternComposite.aspx#_self1]

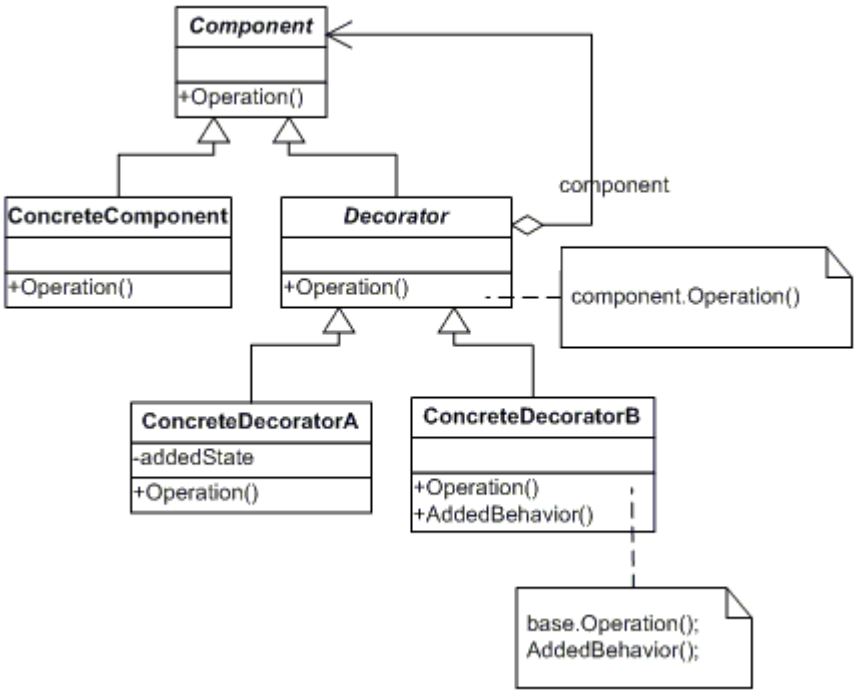
4.2.4 Decorator pattern

Eesmärk/probleem: Lisab objektidele automaatselt kohustusi

Motivatsioon: Lisada objektidele dünaamiliselt täiendavaid kohustusi.

Rakendatavus: Kujundaja tagab paindliku alternatiivi alamklassidele, et laiendada oma funktsionaalsust.

Struktuur: Klassi ja seos diagramm.



Osalejad: Klassid ja objektid mida antud mustris kasutatakse ja mis on nende roll:

- Component
  - Kasutajaliides objektidele, millel võivad olla dünaamiliselt lisatud kohustusi
- ConcreteComponent
  - Objekt millele võib lisada täiendavaid kohustusi
- Decorator
  - Haldab viiteid komponent (Component) objektile ja defineerib kasutajaliidese, mis mugandub komponent (Component) kasutajaliidesele
- ConcreteDecorator
  - Lisab komponendile kohustusi

Näidis kood: Näide kuidas programmi koodi kasutatakse

[vt. http://www.dofactory.com/Patterns/PatternDecorator.aspx#_self1]

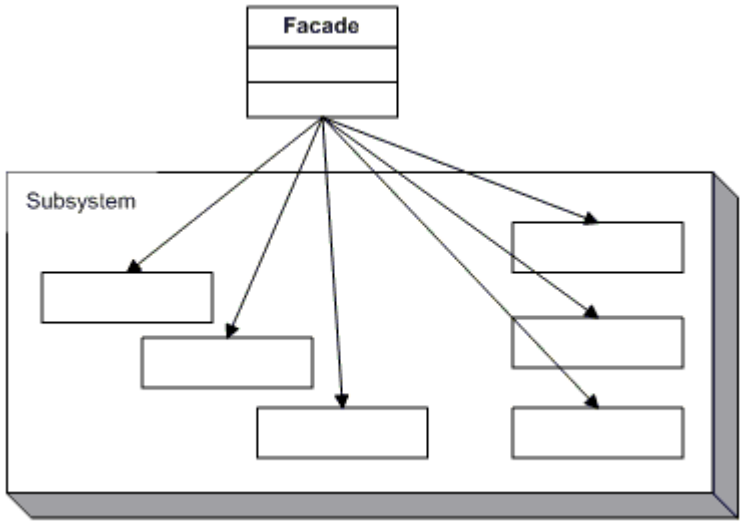
4.2.5 Facade pattern

Eesmärk/probleem: Üksik klass, mis esindab kogu alamsüsteemi

Motivatsioon: Võimaldab luua koondatud kasutajaliidese alamsüsteemidest

Rakendatavus: Fassaadi defineerib kõrgema tasemega liides, mis siis võimaldab kasutada alamsüsteeme kergemini kasutada

Struktuur: Klassi ja seos diagramm.



Osalejad: Klassid ja objektid mida antud mustris kasutatakse ja mis on nende roll:

- Facade
  - Teab millised alamklasside on vastutavad päringutele
  - Delegeerib kliendi päringuid sobiva alasüsteemi objektile
- Subsystem classes
  - Rakendavad alamsüsteemi funktsionaalsused
  - Tegelevad fassaad (Facade) poolt antud käsklustega
  - Ei ole teadlikud mitte midagi fassaadist ja ei hoia mitte mingit seost sellega.

Rakendamine: Kirjeldab kuidas mustrit rakendada ja kuidas see on lahendatud. Sisaldab endas tehnikat ja vihjeid kuidas seda mustrit kasutada.

Näidis kood: Näide kuidas programmi koodi kasutatakse

[vt. http://www.dofactory.com/Patterns/PatternFacade.aspx#_self1]

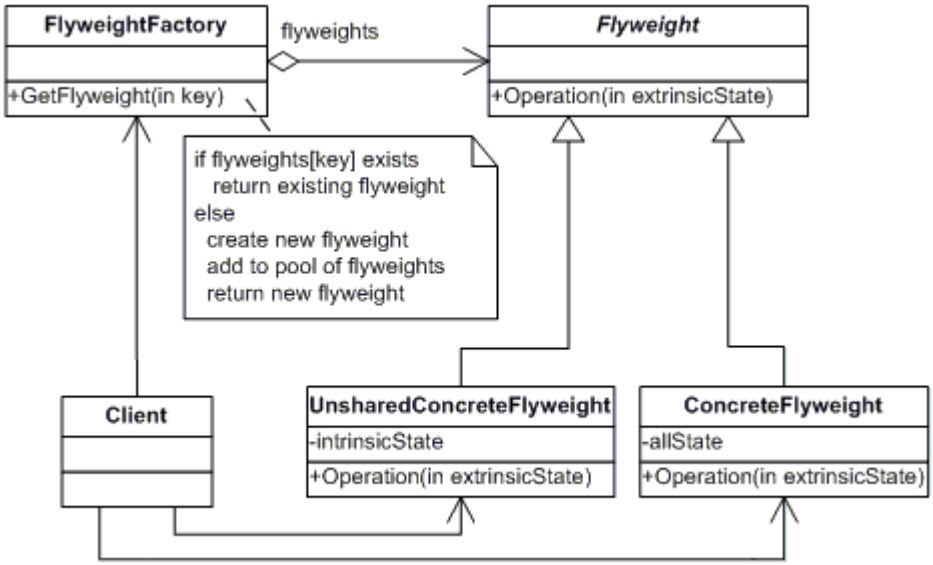
4.2.6 Flyweight pattern

Eesmärk/probleem: Kui on vaja manipuleerida mitme objektiga ja neile ei ole võimalik rakendada lisa infot.

Motivatsioon: Võimaldab suurendada tulemuslikust lisades

Rakendatavus: Mingi selline situatsioon kus on hulk andmeid mida on vaja töödelda. Iga infoväli on mingi objekt, mis siis sisaldab mingeid omadusi.
Nt. XML struktuuri töötlemine.

Struktuur: Klassi ja seos diagramm.



Osalejad: Klassid ja objektid mida antud mustris kasutatakse ja mis on nende roll:

- Flyweight
  - Kutsub välja liidese läbi mille on võimalik siis rakendada vastavaid tegevusi
- ConcreteFlyweight
  - Võtab kasutusele kärbeskaal (Flyweight) liidese ja lisab oleku salvestamise võimaluse. Peab olema jagatav.
- UnsharedConcreteFlyweight
  - Mitte kõik kärbeskaal (Flyweight) alamklassid ei ole jagatavad. Kärbeskaal (Flyweight) liides võimaldab jagamist, kuid ei nõua seda. Seega peab olema võimalus selliste järglaste jaoks, mis sinna ei kuulu ning mis siis ainult salvestatakse struktuuri osana (rida ja veerg).
- FlyweightFactory
  - Loob ja haldab kärbeskaal (Flyweight) objekti

    - Tagab, et kärbeskaal (Flyweight) on jagatud nõuetekohaselt. Kui klient pärib kärbeskaal (Flyweight), siis kärbeskaal vabrik (FlyweightFacotory) objekt pakub olemasoleva liidese või tekitab selle.
- Client
    - Haldab viiteid kärbeskaal (flyweight(s)).
    - Arvutab või salvestab jooksvaid kärbeskaal (flyweight) seisundeid

Rakendamine: Kirjeldab kuidas mustrit rakendada ja kuidas see on lahendatud. Sisaldab endas tehnikat ja vihjeid kuidas seda mustrit kasutada.

Näidis kood: Näide kuidas programmi koodi kasutatakse

[vt. http://www.dofactory.com/Patterns/PatternFlyweight.aspx#_self1]

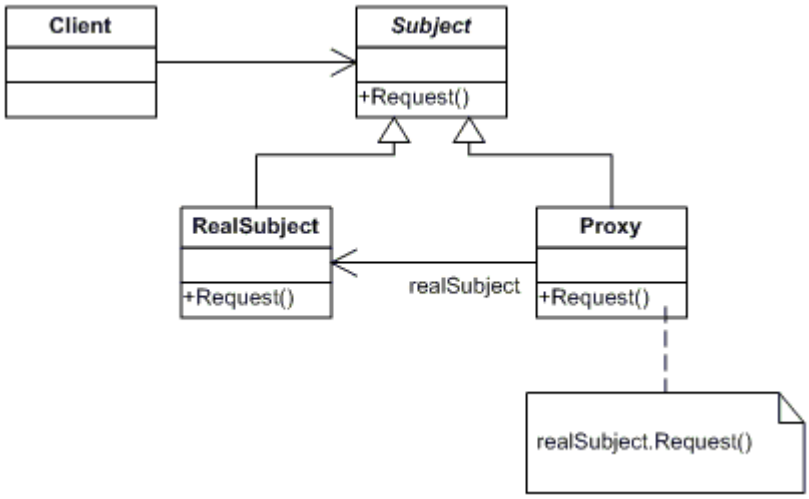
4.2.7 Proxy pattern

Eesmärk/probleem: Võimaldab kasutada muutujat või võõrast objekti, et tagada ligipääsu mingile teisele objektile.

Motivatsioon: Kui on vaja ligipääsu mingile objektile pakkudes või kasutades selle funktsionaalsust, et siis neid muudatusi sisse viia.

Rakendatavus: Nt. vahendaja tüübid on Remote Proxy, Virtual Proxy, Cache Proxy, Firewall Proxy.

Struktuur: Klassi ja seos diagramm.



Osalejad: Klassid ja objektid mida antud mustris kasutatakse ja mis on nende roll:

- Client
    - Kustub välja konkreetse subjekti klassi
- Subject
    - On üldine liides reaalne subjekt (RealSubject) ja vahendaja(Proxy) vahel, et vahendajat (Proxy) oleks võimalik kasutada igal pool kus eeldatakse, et RealSubject eksisteerib
- Proxy

    - Haldab viiteid, et vahendajal oleks võimalus ligi pääseda reaalsele subjektile. Vahendaja (Proxy) võib viidata subjektile (Subject), kui reaalse subjekti (Real Subject) ja subjekti (Subject) liidesed on ühesugused.
    - Kontrollib ligipääsu reaalse subjekti üle ja võib vastutada selle loomise ja kustutamise eest.
- RealSubject
    - Kujutab endast reaalset objekti, mida

Näidis kood: Näide kuidas programmi koodi kasutatakse

[vt. http://www.dofactory.com/Patterns/PatternProxy.aspx#_self1]

4.3 *Käitumuslikud mustrid (Behavioral patterns)*

Kirjeldab üldist kommunikatsiooni programmi ülesehituse mustrit tuginedes objektide vahelistele seostele. See lisab paindlikust andmevahetusse.

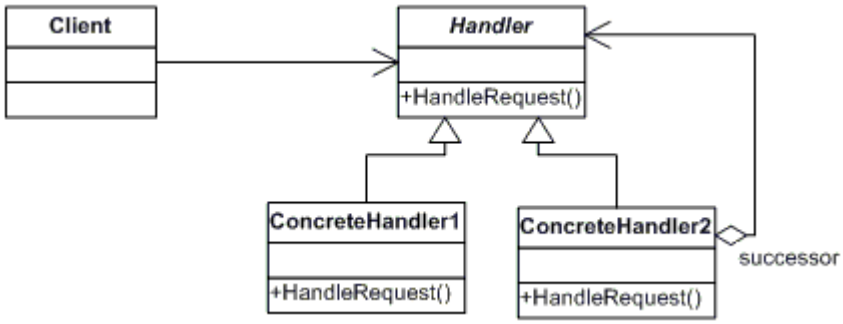
4.3.1 Chain of responsibility pattern

Eesmärk/probleem: Kui on vaja, et mingi konkreetne päring sooritada mingi konkreetse päringu haldaja poolt. Sisaldab endas viisi kuidas anda edasi päringuid mööda objektide ahelat.

Motivatsioon: Päringu seose ahel, mis vahendaks päringuid ja määraks, kes selle päringuga tegelema peaks.

Rakendatavus: Annab mööda objektide ahelat päringut edasi kuni konkreetse objektini, mis sellega tegelema hakkab.

Struktuur: Klassi ja seos diagramm.



Osalejad: Klassid ja objektid mida antud mustris kasutatakse ja mis on nende roll:

- Client
    - Teeb päringu konkreetse treener (ConcreteHandler) ahelas olevale objektile
- Handler
    - Liides, mis defineerib päringu haldamise
    - (vajadusel) sisaldab linki oma järglasele
- ConcreteHandler(ConcreteHandler1, ConreteHandler2)
    - Tegeleb määratud päringutega
    - Võimalus ligi pääseda oma järglasele

- Kui päring vastab tingimustele, siis ta täidab selle, muidu saadab päringu edasi oma järglasele

Näidis kood: Näide kuidas programmi koodi kasutatakse

[vt. http://www.dofactory.com/Patterns/PatternChain.aspx#_self1]

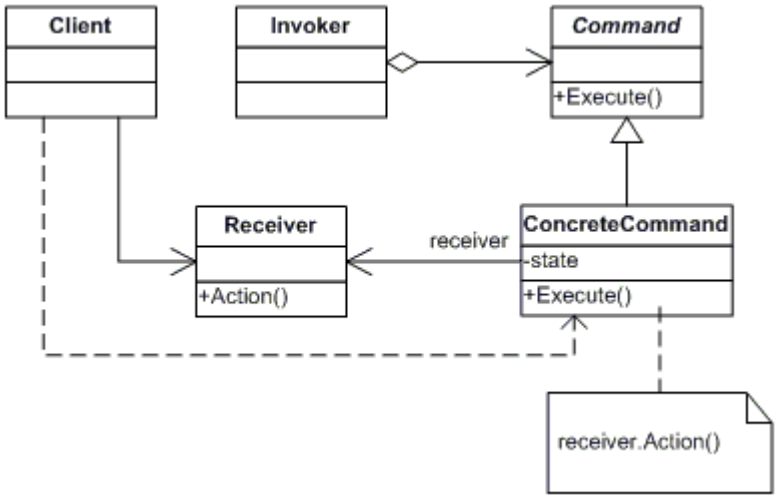
4.3.2 Command pattern

Eesmärk/probleem: Kapseldab objekti kujul käsu päringu

Motivatsioon: Võimaldab anda klientidele parameetreid erinevatele päringutele, järjekorda seada, päringuid logida ja toetada tagasi võetavaid operatsioone.

Rakendatavus: Objektid sisaldavad endas tegevust ja parameetreid

Struktuur: Klassi ja seos diagramm.



Osalejad: Klassid ja objektid mida antud mustris kasutatakse ja mis on nende roll:

- Command
  - Deklareerib liidese, et käivitada käsk
- ConcreteCommand
  - Deklareerib seose vastuvõtja (Receiver) objekti ja tegevuse vahel
  - Rakendab käivituse (Execute) kutsudes välja vastavad tegevused vastuvõtja (Receiver) objektis
- Client
  - Loob konkreetse käsu (ConcreteCommand) objekti ja seadistab selle vastuvõtja
- Invoker
  - Palub käsul läbi viia päring
- Receiver
  - Teab mis tegevusi tuleb teha, et läbi viia päring

Näidis kood: Näide kuidas programmi koodi kasutatakse

[vt. http://www.dofactory.com/Patterns/PatternCommand.aspx#self_1]

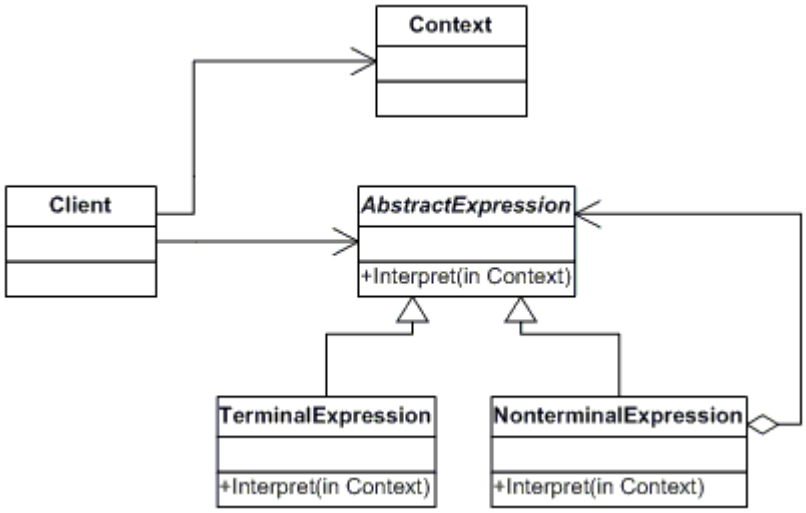
4.3.3 Interpreter pattern

Eesmärk/probleem: Seaduspärasus kuidas lisada programmi sees endale keele elemente.

Motivatsioon:

Rakendatavus: Vastavalt keelele, defineeritakse taasesitus tuginedes grammatikale koos interpretaatoriga, mis siis kasutab keele interpreet lausendeid.

Struktuur: Klassi ja seos diagramm.



Osalejad: Klassid ja objektid mida antud mustris kasutatakse ja mis on nende roll:

- Client
  - Kutsub välja Interpreet operatsiooni
- Context
  - Sisaldab informatsiooni, mis on globaalne interpretaatorile
- AbstractExpression
  - Deklareerib liidese, et käivitada tegevus
- TerminalExpression
  - Iga lause terminaalse sümboli jaoks on oma juhtum
  - Vastavalt terminaalsele sümbolile rakendatakse vastav Interpreet operatsioon
- NonterminalExpression
  - Tavaliselt kutsub rekursiivselt iseennast välja vastavalt sümboli muutujatele R_1 kuni R_n
  - Sisaldab Interpreet operatsiooni grammatikas esinevate mitte terminaalsete sümbolite jaoks

Näidis kood: Näide kuidas programmi koodi kasutatakse

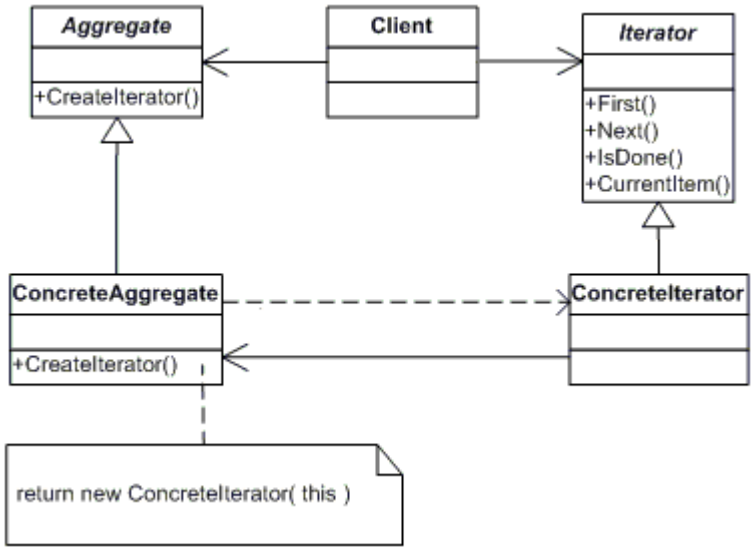
[vt. http://www.dofactory.com/Patterns/PatternInterpreter.aspx#_self1]

4.3.4 Iterator pattern

Eesmärk/probleem: Vaja pidevalt ligipääsu elementide hulgale ilma, et seaks hädaohtu aluseks olevate esitlust.

Motivatsioon: Võimaldab ligipääsu elementidele tervikuna.

Struktuur: Klassi ja seos diagramm.



Osalejad: Klassid ja objektid mida antud mustris kasutatakse ja mis on nende roll:

- Iterator
  - Loob liidese pääsemaks ligi elementidele ja nende läbimiseks
- ConcreteIterator
  - Realiseerib *Iterator* liidese
  - Jälgib hetke positsiooni elementide läbimisel
- Aggregate
  - Loob liidese, et luua *Iterator* objekt
- ConcreteAggregate
  - Realiseerib *Iterator* loomise liidese, et tagastada konkreetne *ConcreteIterator*

Näidis kood: Näide kuidas programmi koodi kasutatakse

[vt. http://www.dofactory.com/Patterns/PatternIterator.aspx#_self1]

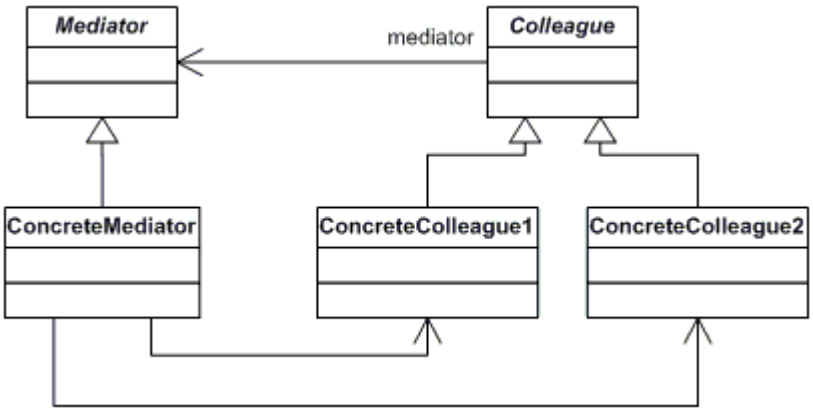
4.3.5 Mediator pattern

Eesmärk/probleem: Defineerib lihtsustatud klasside vahelist kommunikatsiooni

Motivatsioon: Võimaldab luua objekti, mis siis

Rakendatavus:

Struktuur: Klassi ja seos diagramm.



Osalejad: Klassid ja objektid mida antud mustris kasutatakse ja mis on nende roll:

- Mediator
  - Liides, et suhelda Colleaque objektidega
- ConcreteMediator
  - Kordineerides Colleague objekti rakendab ta koostöö tegevused
  - Teab ja haldab oma colleagues*
- Colleague classes(ConcreteColleague1, ConcreteColleague2)

Rakendamine: Kirjeldab kuidas mustrit rakendada ja kuidas see on lahendatud. Sisaldab endas tehnikat ja vihjeid kuidas seda mustrit kasutada.

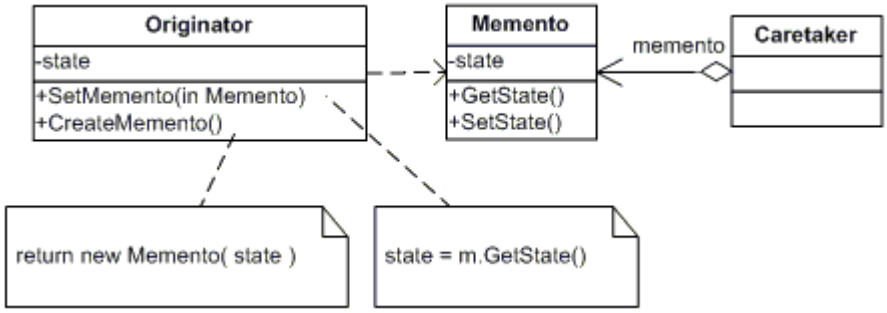
Näidis kood: Näide kuidas programmi koodi kasutatakse

[vt. http://www.dofactory.com/Patterns/PatternMediator.aspx]

4.3.6 Memento pattern

Eesmärk/probleem: Võimaldab rikkumata kapsel kinni püüda ja väljendada objekti sisemist seisundit nii, et hiljem oleks võimalik konkreetne objekt taastada oma eelnevasse seisundisse.

Struktuur: Klassi ja seos diagramm.



Osalejad: Klassid ja objektid mida antud mustris kasutatakse ja mis on nende roll:

- Memento
  - Salvestab algataja (Originator) seisundi.
- Originator
  - Loob eseme, mis sisaldab vaate oma hetke seisust
  - Kasutab eset, et salvestada oma hetke seisu

- Caretaker
    - On vastutav eseme mingi hetke salvestamise eest
    - Ei jälgi ega opereeri eseme sisuga

Rakendamine: Kirjeldab kuidas mustrit rakendada ja kuidas see on lahendatud. Sisaldab endas tehnikat ja vihjeid kuidas seda mustrit kasutada.

Näidis kood: Näide kuidas programmi koodi kasutatakse

[vt. http://www.dofactory.com/patterns/PatternMemento.aspx#_self1]

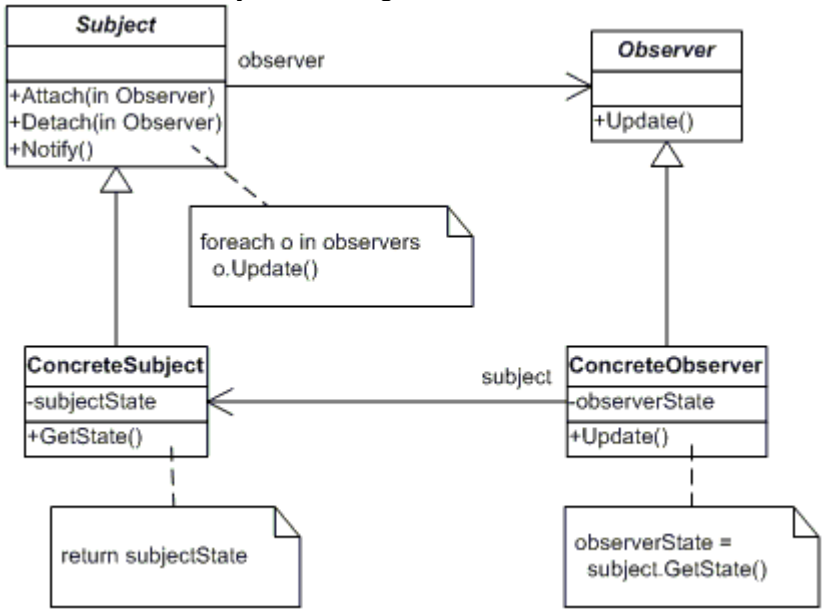
4.3.7 Observer pattern

Eesmärk/probleem: Lisab objektidele automaatselt kohustusi

Motivatsioon: Lisada objektidele dünaamiliselt täiendavaid kohustusi.

Rakendatavus: Kujundaja tagab paindliku alternatiivi alamklassidele, et laiendada oma funktsionaalsust.

Struktuur: Klassi ja seos diagramm.



Osalejad: Klassid ja objektid mida antud mustris kasutatakse ja mis on nende roll:

- Subject (Stock)
    - Sisaldab liidest millega saab ühendada ja lahti ühendada jälgija (Observer) objekti
    - Teab oma jälgijaid (Observers). Ükskõik milline Observeri objekt võib jälgida mingit alam objekti
- Observer (IBM)
    - Kujutab endast uuendamise liidest objektidele keda tuleks teavitada muudatustest alam objektides
- ConcreteSubject (Investor)
    - Salvestab mingi seisundi ConcreteObserveris
- ConcreteObserver (Investor)

Koostöö: Kirjeldatakse kuidas klassid ja objektid mustris omavahel suhtlevad.

Rakendamine: Kirjeldab kuidas mustrit rakendada ja kuidas see on lahendatud. Sisaldab endas tehnikat ja vihjeid kuidas seda mustrit kasutada.

Näidis kood:

[vt. http://www.dofactory.com/Patterns/PatternObserver.aspx#_self2]

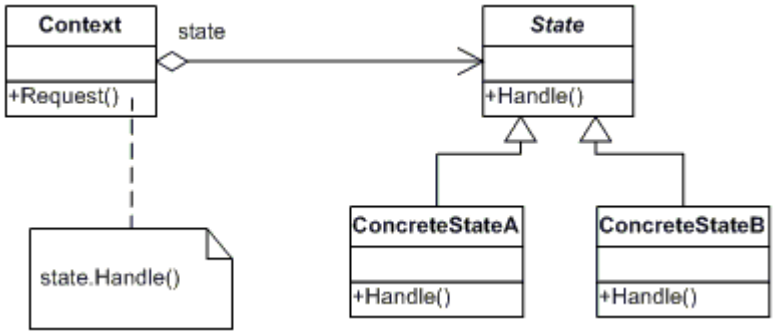
4.3.8 State pattern

Eesmärk/probleem: Lubab objektil muuta oma seisundit ning võimalusel korral delegeerib seda ka teistele objektidele.

Motivatsioon: Lubab objektidel vahendada oma käitumist kui tema sisemine seisund muutub.

Rakendatavus: Näiteks on vaja jälgida mingeid seisundeid ja vastavalt nendele seisunditele, siis aktiveerida mingi tegevus ka teistes alamobjektides.

Struktuur: Klassi ja seos diagramm.



Osalejad: Klassid ja objektid mida antud mustris kasutatakse ja mis on nende roll:

- Context (Konktekst)
  - Kirjeldab kasutaja poolt defineeritud vajadusi konktekstina
  - Haldab ConcreteState alamklasse, mis siis defineerib hetke seisundeid.
- State (Seisund)
  - Konkreetse konteksti jaoks rakendatav seisund
- ConcreteState (ConcreteStateA, ConcreteStateB)(Konkreetne seisund)
  - Iga alamklass rakendab mingi käitumismaali vastavalt konteksti seisundile.

Koodi näide:

[vt. http://www.dofactory.com/Patterns/PatternState.aspx#_self2]

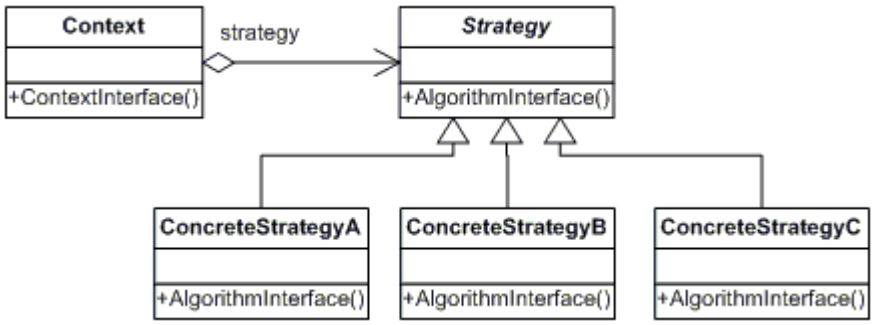
4.3.9 Strategy pattern

Eesmärk/probleem: Kapseldab algoritme klassi sisse

Motivatsioon: Võimaldab kapseldada hulga algoritme nii, et neid ei oleks võimalik enam vahetada.

Rakendatavus:

Struktuur: Klassi ja seos diagramm.



Osalejad: Klassid ja objektid mida antud mustris kasutatakse ja mis on nende roll:

- Strategy
  - Sisaldab endas liidest, mis on üldine kõikidele toetatud algoritmidele. Konktekst (Context) kasutab seda liidest, et kutsuda välja Konkreetse strateegia (ConcreteStrategy) poolt defineeritud algoritmi.
- ConcreteStrategy
  - Rakendab algoritmi kasutades strateegilist (Strategy) liidest
- Context
  - Konkreetse strateegia poolt konfigureeritav objekt
  - Haldan viidet strateegia (Strategy) objektile.
  - Võib luua liidese, mis laseb strateegial (Strategy) infot kasutada.

Näidis kood: Näide kuidas programmi koodi kasutatakse

[vt. http://www.dofactory.com/Patterns/PatternStrategy.aspx#_self1]

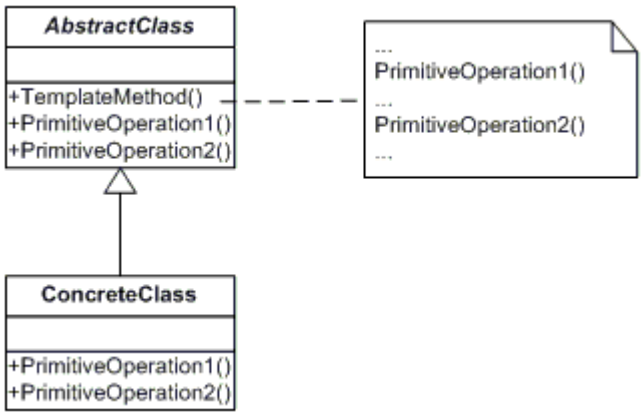
4.3.10 Template method pattern

Eesmärk/probleem: Kui on vaja luua skelett mida siis konkreetsed alamprogrammi hakkasid kasutama.

Motivatsioon: Kui on vaja luua algoritmi skelett ning vajadusel kutsuda välja alamklassi meetodeid.

Rakendatavus: On üks enam levinud programmi ülesehituse mustreid. Seda kasutatakse, et luua skelett või meetodite kogum mida ülejäänud programmi osad kasutavad oma struktuuri juures.

Struktuur: Klassi ja seos diagramm.



Osalejad: Klassid ja objektid mida antud mustris kasutatakse ja mis on nende roll:

- AbstractClass
  - Sisaldab primitiivseid tegevusi mida konkreetsed alamklassid peaksid kasutama oma rakenduse.
  - Sisaldab template meetodeid defineerides algoritmi skeleti. Template meetod kutsub välja primitiivseid tegevusi samuti AbstractClass's klassi seest.
- ConcreteClass
  - Sisaldab algelisi operatsioone, et täita alamklassile iseloomulike algoritmi samme.

Rakendamine: Kirjeldab kuidas mustrit rakendada ja kuidas see on lahendatud. Sisaldab endas tehnikat ja vihjeid kuidas seda mustrit kasutada.

Toon näitena abstraktse template lehe kuvamise klassi. Kõik kes seda klassi endale laiendavad saavad kaasa omadused, mis siis viivad selleni, et kuvatakse HTML leht. Tal on meetodid, mis tegelevad muutujate seadmisega lehele ja Pea lehe seadmisega. Leht koosneb kahest seisundist onInit ja onLoad, mis on siis meetoditeks jagatud. onInit käivitatakse kõige alguses, et oleks võimalik muuta lehe seadeid. onLoad on üldine lehe laadimine ja seejärel kuvamine.

Näidis kood: Näide kuidas programmi koodi kasutatakse

```
<?php
/**
 * TPage
 *
 * @date 10.02.2006
 * @name TPage
 * @author Margo Poolak <margo 'dot' poolak 'at' mail 'dot' ee>
 * @copyright Copyright &copy;2006 Margo Poolak. All Rights Reserved.
 * @version 1.1
 * @package Base module
 * @category
 *
 * Main template page class
 * All displayed web is going through this class. It uses a smarty template
engine to display and render the display
 * It includes itself TRequest, TAja, TUser and TT classes to have more fun
```

```
ctionality
 *
 * Changes in 1.1:
 * Page is faster because smarty will be called out only once and Component
s will be returned to the page
 * $this->set() : now improved set variables to the page. It looks when Mas
terPage is accesible object then it will be set all the values to the Maste
rPage otherwise it sets them to itself
 * Possible to set page visible or not setVisible() | default true
 */
abstract class TPage
{
    protected $s;
    protected $file;
    public $Module;
    public $MasterPage='';
    public $masterPageName='';
    public $Request;

    public function __construct()
    {
        $this->s = new Smarty();
        $this->Request = new TRequest();// get request object
        $this->file = get_class($this);
    }
    /*
    * Sets smarty variables
    * can set a array with keys and values
    */
    public function set($val,$key='')
    {
        if(is_object($this->MasterPage))
        {
            #echo "Set <b>$val</b> To master page";
            if(is_array($val))
                $this->MasterPage->s->assign($val);
            else
                $this->MasterPage->s->assign($val,$key);
        }
        else
        {
            if(is_array($val))
                $this->s->assign($val);
            else
                $this->s->assign($val,$key);
        }
    }
    /*
    * To set master page object
    */
    public function setMasterPage($value)
    {
        $this->MasterPage = $value;
    }
    /*
    * To get master page object
    */
    public function getMasterPage()
    {
        return $this->MasterPage;
    }
```

```
    /*
    * To set module object
    */
    public function setModule($value)
    {
        $this->Module = $value;
    }
    /*
    * returns module object
    */
    public function getModule()
    {
        return $this->Module;
    }
    /*
    * To set master page name
    */
    public function setMasterPageName($value)
    {
        $this->masterPageName = $value;
    }
    /*
    * To get master page name
    */
    public function getMasterPageName()
    {
        return $this->masterPageName;
    }
    /*
    * Executes the page by accessing the load and then renders the page to
display it
    */
    public function execute()
    {
        // page is first to run
        // can change master page name
        if(method_exists($this,'onInit'))
            $this->onInit();// load initialization state
            if($this->masterPageName!=$this->file AND !!($this-
>masterPageName))
            {
                $this->MasterPage = new $this->masterPageName;// calls out
the master page
                // get master page initialization state
                if(method_exists($this->MasterPage,'onInit'))
                    $this->MasterPage->onInit();// load initialization stat
e
            }
            // check for the page states
            if(method_exists($this,'onLoad'))
                $this->onLoad();// load page content
            if(method_exists($this->MasterPage,'onLoad'))
                $this->MasterPage->onLoad();// load page content
            $this->render();// display page
    }
    /*
    * Renders the page and then displays
    */
    public function render()
    {
        if(!($this->MasterPage)){
```

```
            $this->s->display($this->file.TEMPLATE_EXT);
        }else{
            $this->MasterPage->set('page',$this->file);// set sub page to t
he master page
            $this->MasterPage->s->display($this-
>masterPageName.TEMPLATE_EXT);// displays it trough master page
        }
    }
}
?>
```

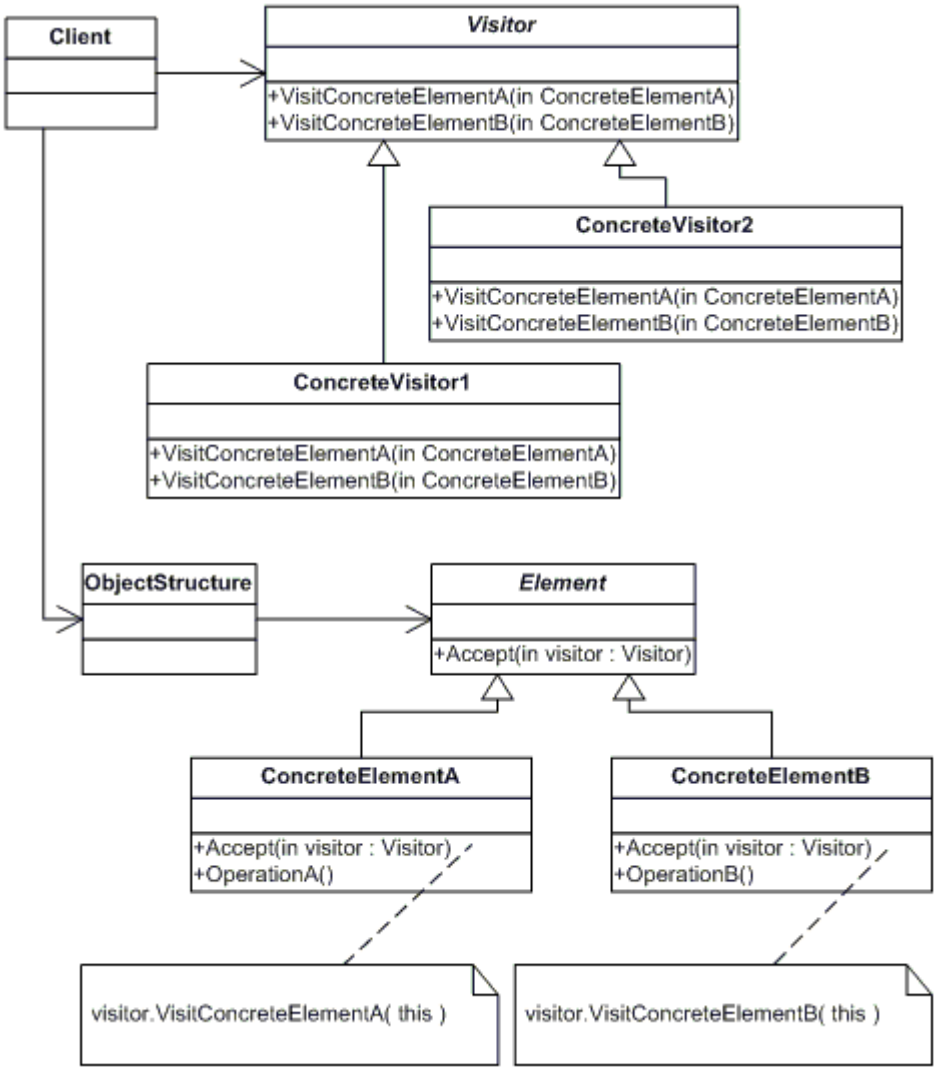
Kasutusjuhud: Viited reaalsetele näidetele, lahendustele kus konkreetset mustrit on kasutatud

```
<?php
class Main_LoginPage extends TPage
{
    function onInit()
    {
        $this->setMasterPageName('Plain_MasterPage');
    }
    function onLoad()
    {
        $page = $this->Request->getRequestedPage();
        $this->MasterPage->setTitle($page);
        $this->MasterPage->setCharset('UTF-8');
        $this->MasterPage->setCSS('default.css');

        $this->set('color','green');

        if($this->Request->isPostBack())
        {
            $valid = $this->Module->Login($this->Request-
>getPOST(USER),$this->Request->getPOST(PASS));
            if(!$valid)
                $this->set('color','red');
            else
                redirect(URL);
        }
    }
}
?>
```

4.3.11 **Visitor pattern**

Eesmärk/probleem: Esitleb operatsioone mida on kasutatud objekti struktuuri elementidele. Külastaja (Visitor) lubab defineerida uue tegevuse ilma, et muudaks elementide klassi milles ta tegutseb.

Struktuur: Klassi ja seos diagramm.



Osalejad: Klassid ja objektid mida antud mustris kasutatakse ja mis on nende roll:

- Visitor
  - Defineeritakse Visit tegevused iga ConcreteElement klassi jaoks objekti struktuuris. Tegevuse nimi ja signatuur identifitseerib klassi kes saatis Visit päringu visitor(külalisele).
- ConcreteVisitor
  -
- Element
  - Defineerib Accept tegevuse, kus külastaja on kui argumenti
- ConcreteElement
  - Rakendab Accept tegevuse kasutades külastaja argumenti
- ObjectStructure
  - Nummerdab oma elemendid.
  - Pakub kõrgema tasemega liidest, et lubada külastajal külastada oma elemente
  - Võib olla Composite (pattern) või kogum sellise nimekirja jaoks.

Näidis kood: Näide kuidas programmi koodi kasutatakse

[vt. http://www.dofactory.com/Patterns/PatternVisitor.aspx#_self1]

4.4 *Fundamentaalsed mustrid (Fundamental patterns)*

4.4.1 Delegation pattern

[vt. http://en.wikipedia.org/wiki/Delegation_pattern]

4.4.2 Functional design

[vt. http://en.wikipedia.org/wiki/Functional_design]

4.4.3 Interface pattern

Ei ole päris programmi ülesehituse muster nagu teised.

Eesmärk/probleem: Üldine meetod, et struktureerida programme, et nendest oleks võimalik paremini aru saada.

Motivatsioon: Annab võimaluse programmeerijal lihtsamini hallata ja pääseda ligi teistele klassidele.

Rakendatavus: Kasutajaliides võib sisaldab objektide kogumikku. Pakub programmeerijale lihtsamaid ja kõrgemal tasemel funktsioone. Annab võimaluse kasutada keerulisi klasse tunduvalt selgemal kujul.

Sarnased mustrid: delegation pattern, composite pattern, and bridge pattern.

[vt. http://en.wikipedia.org/wiki/Interface_pattern]

4.4.4 Proxy pattern

[vt. http://en.wikipedia.org/wiki/Proxy_pattern]

4.4.5 Immutable pattern

[vt. http://en.wikipedia.org/wiki/Immutable_pattern]

4.4.6 Marker interface pattern

[vt. http://en.wikipedia.org/wiki/Marker_interface_pattern]

4.5 *Korduvad mustrid (Concurrency patterns)*

[vt. http://72.14.207.104/search?q=cache:wESOIZ5CCPYJ:www.cs.wustl.edu/~schmidt/patterns-ace.html+concurrency+pattern&hl=en&ct=clnk&cd=1&client=firefox-a]

4.5.1 Action at a distance pattern

[vt. http://en.wikipedia.org/wiki/Action_at_a_distance]
[vt. http://www.hasslberger.com/phy/phy_12.htm]

4.5.2 Active object pattern

[vt. http://www.titu.jyu.fi/modpa/Patterns/pattern-ActiveObject.html]

4.5.3 Balking pattern

[vt. http://en.wikipedia.org/wiki/Balking_pattern]
[vt. http://www.mindspring.com/~mgrand/5pattern-talk/tsld043.htm]

4.5.4 Double checked locking pattern

[vt. http://www-128.ibm.com/developerworks/java/library/j-dcl.html]
[vt. http://www.mindspring.com/~mgrand/double-checked_locking_implementation.html]

4.5.5 Guarded suspension pattern

[vt.
http://www.cs.pdx.edu/~antoy/Courses/Patterns/w01/assignments/GuardedSuspension.html]
[vt. http://www.mindspring.com/~mgrand/pattern_synopses.htm]

4.5.6 Half-Sync/Half-Async pattern

[vt. http://www.cs.wustl.edu/~schmidt/PDF/PLoP-95.pdf]

4.5.7 Leaders/followers pattern

[vt. http://deuce.doc.wustl.edu/doc/pspdfs/lf.pdf]

4.5.8 Monitor object pattern

[vt .
http://publib.boulder.ibm.com/infocenter/txen/index.jsp?topic=/com.ibm.txseries510.doc/aetga30019.htm]

4.5.9 Read write lock pattern

[vt. http://en.wikipedia.org/wiki/Read_write_lock_pattern]

4.5.10 Scheduler pattern

[vt. http://en.wikipedia.org/wiki/Scheduler_pattern]

4.5.11 Thread pool pattern

[vt. http://en.wikipedia.org/wiki/Thread_pool_pattern]

4.5.12 Thread-specific storage pattern

[vt. http://www.cs.wustl.edu/~schmidt/POSA/pattern-abstracts.html#tss]

4.6 *Sündmusi töötlevad mustrid (Event handling patterns)*

Event-driven architectures are becoming pervasive in networked software applications. The four patterns in this chapter help to simplify the development of flexible and efficient event-driven applications. The first pattern can be applied to develop synchronous service providers [15]*:*

Seda laadi programmi mustrid töötlevad sündmuste tsüklit. Pidevalt käib mingi seisundi, tegevuse või aja jälgimine ja seejärel käivitatakse mingi funktsiooni (trigger function), mis töötleb seda infot edasi. [16]

4.6.1 Reactor pattern

The Reactor pattern has been introduced in [Schmidt95] as a general architecture for event-driven systems. It explains how to register handlers for particular event types, and how to activate handlers when events occur, even when events come from multiple sources, in a single-threaded environment. In other words, the reactor allows for the combination of multiple event-loops, without introducing additional threads. [9]

Eesmärk/probleem:.

- Aktiveerib käsitleja kui mingi tegevus toimub
- Suudab käsitleda mitmest kohast tuleva tegevusega
- Ühe sõlmelises protsessis

Motivatsioon:

Rakendatavus: Põhimõte on *Hollywood Principle -- `Don't call us, we'll call you'. Hollywoodi printsiip – ‚Ära helista meile, meie helistame teile' [10].* Programm annab ise teada, et millal mingit tegevust rakendada.

[vt. http://en.wikipedia.org/wiki/Reactor_Pattern]

4.6.2 Proactor pattern

[vt. http://www.cs.wustl.edu/~schmidt/patterns-ace.html]

4.6.3 Asynchronous Completion Token design pattern

Eesmärk/probleem: Viib edukalt täide tegevuse sisaldades kliendi vastust, et lõpetada a-sünkroonne operatsioon, mis on käivitatud kasutaja poolt.

Teatakse ka kui: Active Demultiplexing

Motivatsioon: Muudab aplikatsioonide haldamise dünaamilisemaks.

Rakendatavus: Teavitamaks mingite uutest seisunditest või muudatustest kuskil eemal olevates aplikatsioonides.

4.6.4 Acceptor-Connector design pattern

[vt. http://www.cs.wustl.edu/~schmidt/POSA/pattern-abstracts.html]

4.7 *Arhitektuursed mustrid (Architectural patterns)*

„Väljendades programmi süsteemis fundamentaalseid struktuure organisatsiooni skeemidest. See pakub hulga eeldefineeritud alamsüsteeme, spetsifitseerib nende vastust ja lisab reegleid ja juhiseid, et organiseerida seoseid nende vahel.” [19]

Konkreetsed mustrid luuakse juba programmi spetsiifikat arvestades. Taolisi

5 Analüüs ja kokkuvõte

Töö osutus tunduvalt mahukamaks kui algul ette kujutasin. Materjali on antud teema kohta väga palju, kuid sellegi poolest väga raskesti hallatav. Iseenesest annab programmi ülesehituse mustritest lugemine väga palju juurde ja avardab ideeliselt programmeerimise võtteid.

Mõned arvavad, et kui võtta kasutusele „programmi ülesehituse mustrid”, siis ühel hetkel viib see kokku probleemiga, et väga raske on laiendada või edasi arendada oma programmi.

Selline praktika ei ole ainult tavaline, vaid institutsiooniline. Näiteks OO maailmas kuuled tihti kui head on „programmi ülesehituse mustrid”. Vahetevahel mõtlen, et kas need ei ole mitte tõendusmaterjalid selle kohta, et inimesed on selle taga. Kui ma näen programmi ülesehituse mustrit oma programmis, siis ma näen selles pigem probleeme. Programmi kuju peaks peegeldama ainult seda, mis probleemi ta peab lahendama. Ülejäänud regulaarsused koodis on märgis sellele, et ma pean tegelema laiendustega, mis ei ole piisavalt võimsad – tihti siis, kui mul on vaja luua käsitsi laiendust mõne macro kirjutamise korral. [20]

Isiklikult mina arvan, et jah iga probleem on omaette probleem aga kui need on korduvad probleemid, siis aja kokku hoiu ja töö lihtsustamise mõttes on kõige parem viis need kuidagi kapseldada ühtse struktuuri peale ehk siis paratamatult kalduda programmi ülesehituse mustrite maailma.

Üldiselt programmi kirjutades me ei mõtle, et kuidas ja mis mustrit kuskil kasutada, vaid pigem tugineme oma intuitsioonile. Nõnda võime isegi enese teadmata kirjutada valmis taolise koodi, mis kuulub mingite programmi ülesehituse mustrite valdkonda.

Seega kindlasti kasutada programmi ülesehituse mustrit, kui on vaja lahendada sarnaseid probleeme üha uuesti. Samuti on kasulik kasutada taolist lähenemist, et luua ühtne skelett struktuur või lähenemise viis, et lihtsustada iseenda ja teiste tööd.
Programmi ülesehituse muster peab olema võimalikult lihtne ja arusaadav.

Programmi üleehituse mustrid on meie igapäevane nähtus programmeerimise juures ja seetõttu annab taoline lähenemine väga palju juurde programmile ja selle tegijale.

6 Kasutatud kirjandus

[1] Kuidas jõuda programmi mustri mõisteni:
http://minitorn.tpu.ee/~jaagup/kool/java/abiinfo/tip/html/TIPatterns.htm
[2] Kuidas dokumenteerida programmi ülesehituse mustrit:
http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29
[3] Nimekiri GoF programmi ülesehituse mustritest:
http://en.wikipedia.org/wiki/Design_Patterns
[4] Nimekiri ülejäänud mustrite liigist
http://www.cs.wustl.edu/~schmidt/patterns-ace.html
[5] Üldiselt loomise mustrist
http://en.wikipedia.org/wiki/Creational_pattern
[6] Veidi lähemalt Lazy initalization pattern'ist
http://en.wikipedia.org/wiki/Lazy_initialization_pattern
[7] Anonymouse subroutine
http://en.wikipedia.org/wiki/Anonymous_subroutine_objects_pattern
[8] Kuidas toimib __autoload() PHP 5:
http://ee2.php.net/manual/en/language.oop5.autoload.php
[9] Abstract factory pattern
http://en.wikipedia.org/wiki/Abstract_factory_pattern
[10] Abstract factory pattern klassi ja seos diagramm
http://www.dofactory.com/Patterns/PatternAbstract.aspx
[11] Mis on reactor pattern:
http://www.cs.vu.nl/~eliens/online/oo/l/2/reactor.html
[12] Reactor pattern toimimise printsiip:
http://www.cs.wustl.edu/~schmidt/POSA/event-patterns.html
[13] http://en.wikipedia.org/wiki/Builder_pattern
[14] http://www.dofactory.com/Patterns/PatternPrototype.aspx
[15] http://www.cs.wustl.edu/~schmidt/POSA/event-patterns.html
[16] http://en.wikipedia.org/wiki/Event-driven_programming
[17] http://en.wikipedia.org/wiki/Architectural_pattern
[18] http://www.martinfowler.com/eaaDev/
[19] Arhitektuursete ülesehituse mustrite mõiste:
http://www.crystalclearsoftware.com/cgi-bin/arch_patterns/wiki.pl?Definition_Of_Architectural_Pattern
[20] Programmi mustrid ei ole piisavalt laiendatavad, pigem piiravad
http://www.paulgraham.com/icad.html
[21] http://www.opengroup.org/architecture/togaf7-doc/arch/p4/patterns/patterns.htm
[22] Asynchronouse Completion Token, Douglas C. Schmidt 1998,1999 – sain teada, mis on teine nimi ja mis moodi töötab ja kus kasutatakse. Lk 1.

Lisad

Lühendite loetelu

1. GoF **– Gang of Four** (Nelja inimese grupp)koosneb neljast mehest Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, kes panid aluse programmi ülesehituse mustritele andes välja raamatu „Design Patterns”
2. GUI (Grafical User Interface) _ graafiline kasutaja liides
3. back-end – tagakülg, süsteemi poolne komponent
4. front-end - eessüsteem, esikomponent (kasutajapoolne liides)
5. adaptee - adapteeritu
6. appropriate - sobiv, volitatud, omastama
7. additional - täiendavaid, lisa
8. alter – muutma, vahendama
9. avoid - vältima
10. aggregate – tervik
11. behavioral - käitumuslik
12. binding – siduv
13. Client – klient
14. composition - koosseis
15. conforms - kohanduma, muganduma, vastavuses olema
16. ConcreteFlyweight – konkreetne kärbeskaal
17. colleagues – kolleeg
18. corresponding - vastu võtma
19. defines – defineerib
20. dispatches - lähetatud, teelesaadetud
21. declare – deklareerima
22. decorator - kujundaja
23. enumerate - loetlema, üle lugema
24. encapsulate – kapseldama
25. Factory – vabrik, tehas
26. Flyweight – kärbeskaal
27. FlyweightFactory – kärbeskaalu vabrik
28. implements - rakendama, kasutusele võtma
29. interface – liides, kasutajaliides
30. instance – juhtum, aste
31. invoke – välja kutsuma
32. operation - tegevus
33. provide - andma, tagama, varustama
34. placeholder - muutuja, asetäitja, esindaja
35. proxy - vahendaja, volinik ; vahendatud
36. reference - osutus, viide
37. Real World example – reaalne näide
38. represents - esindama, esitab
39. responsibilities – kohustused
40. Structural example – struktuurne näide
41. subroutine - alamprogramm
42. successor - järglane
43. sequentially – järjestikku
44. target – sihtmärk, suunama
45. traversing – läbi käima
46. manipulated – manipuleerima ja käsitsi töötlema
47. memento - ese
48. unified – ühendatud, ühtne, koond
49. UnsharedFlyweight – jagamatu kärbeskaal
50. queue – järjekord, rivi, saba