https://github.com/eugenp/tutorials/tree/master/patterns/design-patterns

Design Pattern – identifikuoja problema ir pateikia jau paruosta sprendima (yra zinomos problemos ir numatyti atitinkami sprendimai toms problemoms spresti). Jeigu programuotojas atpazysta koks Design Pattern’as naudojamas, jis gali lengviau perprasti application’a ir suprasti application’o struktura. Taip pat jei dirba keletas programmuotoju tai vienas is ju, kuris dirba su ta programa gali pasakyti kitiems su kokiu Design pattern ji padaryta ir kitiems bus daugiau maziau aisku programos struktura.

Pattern catalog – visu design pattern rinkinys. Turi du skyrius;

1. GOF (gang of four) Pattern catalog
2. Creational Patterns – gaires kaip sunkurti viena objekta arba objektu grupe.
3. Singleton
4. Factory
5. Abstract Factory
6. Builder
7. Prototype
8. Structural Patterns – suteikia buda nusakyti santykiams tarp klasiu.
9. Adapter
10. Bridge
11. Flyweight
12. Decorator
13. Proxy
14. Behavioral Paterns – nusako kokia turi buti komunikacija tarp klasiu ir objektu.
15. Command
16. Interpreter
17. Template Method
18. Observer
19. Visitor
20. JEE Pattern catalog

Kiekvienas JEE applicationas susideda is daugelio loginiu layeriu.

1. Data access layer – komunikacija su duombazemis.
2. Service layer – visa business logika.
3. Presentation layer – atvaizduojama informacija klientui.
4. Integration layer – komunikacija su kitais applicationais(web services).

Integration and Presentation layers uses Service layer, Service layer uses Data access layer.

Presentation layer patterns:

1. Intercepting filter
2. Front controller
3. MVC
4. Context object

Buisiness layer patters:

1. Businnes Delegate
2. Transfer object
3. Session facade
4. Servce locator

Data access layer patterns:

1. Data access object

Integration layer patterns:

1. Service activator
2. Web service broker

**GOF (gang of four) Patterns**

Singleton pattern – vienas ir tik vienas instancas ir nesvarbu kiek klasiu ji naudoja. Pvz kokio nors readerio klase kuri skaito failus, nereikia kurti daug inctancu ir sutaupai atminties. Dar vienas pavyzdys butu loggeris, vienas objektas kuris butu dalijamas per ivairias klases. Dar vienas pavysdys butu DataSource (is JDBC) kai vienas datasource objektas palaiko connection poola.

1. Declare the constructor of the class private
2. Declare a static method
3. Declare a static member of the same class type in the class

Factory pattern – paslepia objekto sukurimo procesa(pvz. Car factory, toy factory). Pvz duombazes draiveris. Connection connection = DriverManager.getConnection(String constring).

Abstract factory pattern – paslepia kitu factory sukurima.

Template method pattern – parent klase kuri turi bazinius metodus, kuriuos naudoja child klases.

Adapter pattern – adapterio klase paima parametrus is vienos klases, atlieka kzkokius veiksmus, kad galetu kreiptis i dar viena klase (adapterio klase atlieka kazkokius veiksmus kad kitos dvi klases galetu kreptis viena i kita per adapterio klase).

Flyweight pattern – strukturinis patternas atminties sutaupymui, kai vietoj naujo objekto kurimo perpanaudojami jau sukurti objektai.

1. Separate the Extrinsic state
2. Pass them as parameters
3. Create a factory class

Command pattern – behavioural paternas. Jis atskiria clienta nuo konretaus darbo. Tarkim web applikacijose gali buti naudojamas serverio puseje kad atliktu ivairius tarkim vykdytu uzsakymus, darytu pavedimus, issiustu uzsakymus. Ir priklausomai nuo veiksmo, kuri pasiuncia klientas tam tikra komanda pradeda veikti (Processorders -> execute, MakePayment -> execute, ShipOrder ->execute).

Decorator pattern – behavioural paternas, kuris prideda objektui papildomo funkcionalumo dinamiskai at runtime. Input output klases naudoja decorator patterna (new BufferedReader(new FileReader..))), siuo atveju BufferedReader yra decorator’ius. Abi klases (BufferedReader ir FileReader) implementina Reader interfeisa.

Builder pattern -

Facade pattern – paslepia nuo userio sudetingus dalykus. Tarkim automobilio uzvedimas, tai reikia paimti ora, paduoti kura, paleisti starteri, ijungti ventiliatorius, o uzgesinant automobili vel visus veiksmus atlikti. Vietoj to, tiesiog galima padaryti metodus startEngine() ir stopEngine() , kurie paslepia visa komplikuotumus.

**JEE Patterns**

Inversion of control pattern – objekto sukurimas ir inject’inimas at runtime perleidziamas frameworkui (Spring (inversion of control containeriui)). We achieve dependency injection through inversion of control.

JEE layers:

1. Presentation layer
2. Service layer
3. Data Access layer
4. Integration layer

JEE klases(C) ir interfeisai(I):

1. Model Class(C) or Entity class or domain model it all means the same.
2. Data Access Layer interface (IDAO(I)).
3. Data Access ObjectImpl(DAOImpl(I)).
4. IService(I).
5. ServiceImpl(C).
6. Controller(C).
7. Utility Class(C). Special operations across all layers.
8. Validator Class(C). To validate data from user to user.
9. Service Provider(C) or Service Consumer(C). Usually web services.
10. View(C).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Presentation Layer** | **Service Layer** | **Data Access Layer** | **Integration Layer** |
| 1. Model Class(C) | 6. Controller(C) | 4. IService(I) | 2. Data Access Layer interface (IDAO(I)) | 9. Service Provider(C) or Service Consumer |
| 7. Utility Class(C) |  |  |  |  |
| 8. Validator Class(C) |  |  |  |  |
| 10. View(C) |  | 5. ServiceImpl(C) | 3. Data Access ObjectImpl(DAOImpl(I)) |  |

Layeriu nauda:

1. Simplicity – paprastumas, lengviau suprantama, kai viskas sudeta ne i viena klase.
2. Separation of concerns - dalyku atskyrimas, kiekvienas layeris turi savo paskirti.
3. Easy maintenance – lengva prieziura, jei susiduriama su bedomis, galima lengvai nuspeti kur ieskoti problemos.

Intercepting Filter pattern – specialus procesai, kurie atliekami iki request’o patekimo pas Targeta - Request Handleri, Servleta ar Controlleri, tokie kaip uncompress request, authenticate, audit(log), decode decrypt.

Ateina requestas ir specialus filtrai atlieka specialius procesus.

Pavysdys: Tarkim kad musu application pasiektu tik is Chromo narsykles, dedama filtra, kuris patikrina ar tai Crome’as ar ne, jei taip leidziame toliau proccessinti, jei ne – neleidziame.

Model View Controller (MVC) pattern – splits web layer into three parts (Model(Java class), View(JSP), Controller(Servlet)). Model represents the current state of the application and does most of the work or business logic. View is responsible for displaying current Model to the end user. Controller acts as a glue between Model and View, it is responsible for selecting appropriate Model for the appropriate request. When the request comes from the client it goes to the Controller, the Controller will select appropriate Model (which does the all the logic), then the model return the response to the Controller and the Controller will select appropriate View to display the data (e.g HTML) to the client.

The are two main advantages for using MVC pattern:

1. Maintenance – we split responsibilities, so if there is problem we could easily find it.
2. Parallel Development – one developer could work with business logic another with views and so on