Design Pattern in JAVA

Design pattern adalah best practice bagi developer software berorientasi object untuk mengatasi permasalahan-permasalahan umum yang akan dihadapi pada proses pengambagan software. Design pattern dikategorikan menjadi 3 jenis yaitu Creational, Structural, dan Behavioural.

Creational design pattern adalah proses membuat object dengan menyembunyikan logic pembuatan object tersebut. Hal ini memudahkan program untuk lebih flexible dan dapat memilih object mana yang perlu dibuat untuk skenario tertentu. Creational design pattern terdiri dari

1. Abstract Factory: Memanggil object-object yang saling berhubungan tanpa perlu menspesifikasikan class konkrit mereka.
2. Builder: Proses membangun sebuah object kompleks dengan beberapa langkah. Hal ini membuat anda dapat membangun tipe-tipe yang berbeda dari sebuah object menggunakan konstruktor yang sama.
3. Factory Method: Menyediakan interface untuk membuat object dalam superclass, namun memperbolehkan subclass untuk mengubah tipe object yang akan dibuat.
4. Prototype: Memperbolehkan anda untuk mengcopy sebuah object tampa membuat kode anda bergantung pada class tersebut.
5. Singleton: Memastikan bahwa sebuah class hanya memiliki satu instance dan menyediakan akses global ke instance tersebut.

Structural design pattern berfokus pada komposisi object dan class. Konsep inheritance digunakan untuk membangun interface dan mendefinisikan cara untuk membangun object dengan fungsi-fungsi baru

1. Adapter: Mengubungkan object-object yang mengimplementasikan interface yang berbeda.
2. Bridge: Membuat anda dapat memecah sebuah class besar menjadi class-class kecil yang memiliki hirarki yang berbeda dan dapat dikembangkan secara independent satu sama lain.
3. Composite: Menyusun object dalam bentuk tree structure dan membuat anda dapat bekerja pada struktur-struktur ini seakan-akan mereka adalah struktur independent.
4. Decorator: Memperbolehkan anda untuk menambahkan sifat baru pada object dengan menempatkan object tersebut pada sebuah wrapper object yang menyimpan sifat tersebut.
5. Facade: Menyederhanakan interface untuk library, framework, atau class-class complex.
6. Flyweight: Membuat anda dapat menyimpan object kedalam RAM secara lebih efisien dengan membagikan bagian-bagian yang mirip dari object-object tersebut satu sama lain.
7. Proxy: Object pengganti dari sebuah object asli. Object pengganti ini membuat anda dapat melakukan aksi sebelum atau sesudah sebuah request diterima oleh object asli.

Behavioural berfokus pada komunikasi antar object

1. Chain of Responsibility: Membuat anda dapat mengoper sebuah request pada method-method yang berurutan sesuai kriteria yang sudah ditetapkan sebelumnya untuk menentukan apakah request tersebut memenuhi kriteria yang ada.
2. Iterator: Mengakses sebuah element pada sebuah collection tanpa perlu mengekspose representasi object tersebut (list, stack, tree, etc)
3. Command: Mengubah sebuah request menjadi stand-alone object yang berisi informasi dari request tersebut.
4. Mediator: Meringankan ketergantungan sebuah object pada object lain dengan membangun sebuah mediator untuk menengahi semua komunikasi antar object.
5. Memento: Menyimpan dan mengembalikan kondisi sebuah object tanpa menunjukan detail implementasi object tersebut.
6. Observer: Sebuah fungsi untuk memberikan notifikasi pada object-object tertentu saat sebuah event terjadi pada object yang diawasi.
7. State: Membuat object dapat mengubah sifatnya saat terdapat perubahan kondisi internal. Seakan-akan object tersebut berganti class.
8. Strategy: Membangun algoritme-algoritme yang berhubungan dan menempatkanya pada class-class yang berbeda sehingga object-object didalamnya dapat ditukar satu sama lain.
9. Template: Membuat sebuah algoritme fondasi pada superclass lalu memperbolehkan subclass mengoverride tahap spesifik pada algoritme tersebut tanpa mengubah strukturnya.
10. Visitor: Memisahkan algoritme dari object yang mereka operasikan.