Nghia Tran

1. Observer pattern:

Intent:

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

Example:

**public** **class** Subject {

**private** ArrayList<Observer> observers = **new** ArrayList();

**public** **void** registerObservers(Observer aStudent) {

observers.add(aStudent);

}

**public** **void** removeObservers(Observer aStudent) {

observers.remove(observers.indexOf(aStudent));

}

**public** **void** notifyObservers() {

System.***out***.println("Teacher said: \"Hello\" ");

**for** (**int** i = 0; i < observers.size(); i++) {

observers.get(i).update();

}

}

}

**public** **class** Observer {

**public** **void** update() {

System.***out***.println("Student replied: \"Good morning\"");

}

}

2. Strategy pattern:

Intent: Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

**public** **interface** Strategy {

**public** **void** greeting();

}

**public** **class** StrategyOne **implements** Strategy {

@Override

**public** **void** greeting() {

System.***out***.println("Nice to meet you");

}

}

**public** **class** StrategyTwo **implements** Strategy {

@Override

**public** **void** greeting() {

System.***out***.println("Not so nice to meet you");

}

}

**public** **class** Person {

**private** Strategy greetingStrategy;

**public** Person(Object object) {

**this**. greetingStrategy = (Strategy) object;

}

**public** **void** greet() {

**this**. GreetingStrategy.greeting();

**}**

}

3. Facade Pattern

Intent: Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.

**public** **class** FacadePattern {

**public** **class** ClassA {

**public** **static void** methodA() {

/\*\*

\* Do something

\*/

}

**}**

**public** **class** ClassB {

**public** **static void** methodB() {

/\*\*

\* Do something

\*/

}

**}**

**public** **static** **void** main(String[] args) {

ClassA.methodA();

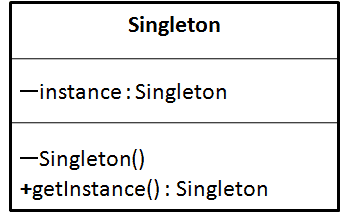
ClassB.methodB();

}

}

4. Singleton Pattern

Intent: Ensure a class has only one instance, and provide a global point of access to it.



**public** **class** Singleton {

**private** **static** Singleton *instance* = **null**;

**private** String string;

**private** Singleton(){

string = "Hello";

}

**public** **static** Singleton getInstance(){

**if**( *instance* == **null**)

{

*instance* = **new** Singleton();

}

**return** *instance*;

}

**public** String getString(){

**return** **this**.string ;

}

**public** **void** setString(String value){

string = value;

}

}

5. Adapter Pattern

Intent: Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

**public** **class** French {

**public** **void** sayHi() {

System.***out***.println("Bonjour!!!");

}

}

**public** **interface** Person {

**public** **void** greeting();

}

**public** **class** American **implements** Person {

**public** French frenchLanguage = **new** French();

**public** **void** greetingAFrench() {

frenchLanguage.sayHi();

}

}

6. Decorator Pattern

Intent: Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

**public** **class** Person {

**public** String language;

**public** String getLanguage(){

**return** "I speak Spanish";

}

}

**public** **abstract** **class** Language **extends** Person{

**abstract** **public** String getLanguage();

}

**public** **class** Vietnamese **extends** Language{

**public** Person person;

**public** Vietnamese(Person person){

**this**.person = person;

}

@Override

**public** String getLanguage() {

**return** person.getLanguage() + ", Vietnamese";

}

}

**public** **class** English **extends** Language{

**public** Person person;

**public** English(Person person){

**this**.person = person;

}

@Override

**public** String getLanguage() {

**return** person.getLanguage() + ", English";

}

}

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Person me = **new** Person();

me = **new** Vietnamese(me);

me = **new** English(me);

System.***out***.println(me.getLanguage());

}

}