**Facade Design Pattern**

**Problem Defination**

sometimes system is very complex or difficult to understand because the system has a large number of interdependent classes or because its source code is unavailable.

**Pattern Idea**

This pattern hides the complexities of the larger system and provides a simpler interface to the client. It typically involves a single wrapper class that contains a set of members required by the client. These members access the system on behalf of the facade client and hide the implementation details.

**Usage**

The facade pattern is typically used when

* a simple interface is required to access a complex system.
* a system is very complex or difficult to understand.
* an entry point is needed to each level of layered software.
* the abstractions and implementations of a subsystem are tightly coupled.

**Pattern implementation (Java)**

*/\* Complex parts \*/*

**class** **CPU** {

**public** void freeze() { ... }

**public** void jump(long position) { ... }

**public** void execute() { ... }

}

**class** **HardDrive** {

**public** byte[] read(long lba, int size) { ... }

}

*/\* Facade \*/*

**class** **ComputerFacade** {

**private** **final** CPU processor;

**private** **final** Memory ram;

**private** **final** HardDrive hd;

**public** ComputerFacade() {

**this**.processor = **new** CPU();

**this**.ram = **new** Memory();

**this**.hd = **new** HardDrive();

}

}

*/\* Client \*/*

**class** **You** {

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

var computer = **new** ComputerFacade();

computer.start();

}

}

Drawbacks

- In Facade design pattern the subsystnem methods are connected to the Façade layer. In future development if structure of the subsystem changes then it will require subsequent change to the Façade layer and client methods.