

**Design Pattern Mini-Project**

**(14CS413)**

**Twin Pattern**

**1PI14CS129 - Vijay M Mundaragi**

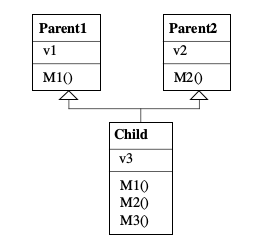
**1PI14CS087 - S Dhamodhran**

**Twin Pattern**

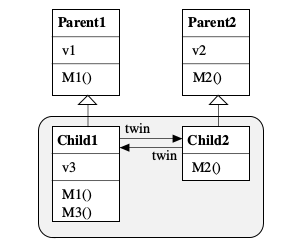
**Intent -**

Simulating multiple inheritance in programming languages that do not support this feature. The pattern avoids many of the problems of multiple inheritance while keeping most of its benefits.

**Structure -**

****

Structure of Multiple Inheritance



Structure Of Twin Pattern

Instead of having a single class that is derived from 2 different parent classes namely Parent1 and Parent2, We have two separate classes Child1 and Child2 which are derived from the Parent1 and Parent2 respectively. Child1 and Child2 are closely coupled via fields so that we can view them as twin objects having two ends. Child1 is compatible with Parent1 and Child2 is compatible with Parent2.

**Participants**

Parent1 and Parent2

* The classes from which you want to inherit.

Child1 and Child2

* The subclasses of Parent1 and Parent2. They are mutually linked via fields. Each subclass may override methods inherited from its parent. New methods and fields are usually declared just in one of the subclasses (e.g. in the figure the new method M3() is added to Child1).

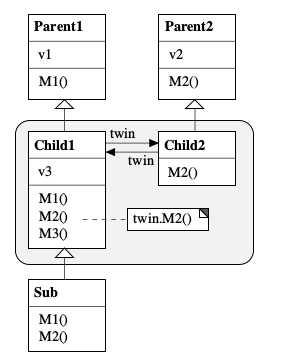
**Collaborations**

* Every child class is responsible for the protocol inherited from its parent. It handles messages from this protocol and forwards other messages to its partner class.
* Clients of the twin pattern reference one of the twin objects directly and the other via its twin field.
* Clients that rely on the protocols of Parent1 or Parent2 communicate with objects of the respective child class (Child1 or Child2).

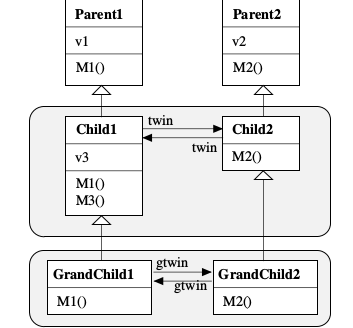
**Drawbacks -**

**Subclassing the twin Pattern**

If the twin pattern should again be subclassed, it is often sufficient to subclass just one of the partners, for example Child1. In order to pass the interface of both partner classes down to the subclass, it is convenient to collect the methods of both partners in one class. One can add the methods of Child2 also to Child1 and let them forward requests to the other partner.

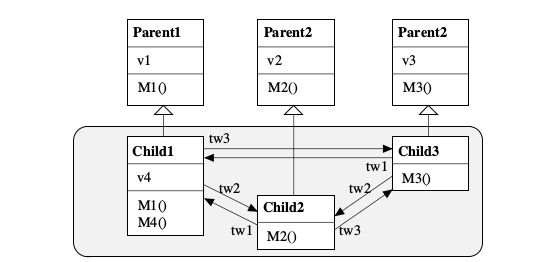


The problem with this apporach is that Sub is only compatible with Child1 but not with Child2. If one wants to make the subclass compatible with both Child1 and Child2 one has to model it according to the Twin pattern again.



**Inheriting from more than 2 parent classes -**

The Twin pattern can be extended to more than two parent classes in a straightforward way. For every parent class there must be a child class. All child classes have to be mutually linked via fields.



As the number of parent classes to be inheritance increases, it becomes more complex to maintain the classes as they are all tightly coupled. The partners of a twin class have to cooperate closely.

**Applicability -**

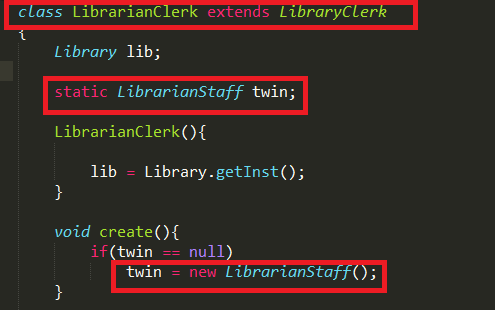
The Twin pattern can be used

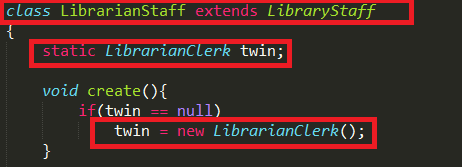
* To simulate multiple inheritance in a language that does not support this feature.
* To avoid certain problems of multiple inheritance such as name clashes.

**Sample Code -**

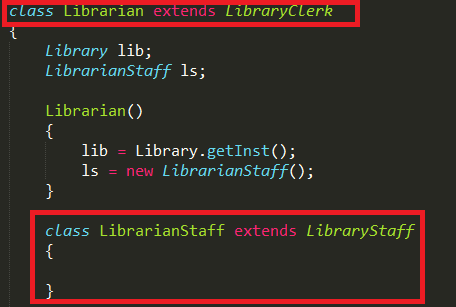
Consider a Library scenario. There are three administrative staff namely a Library Clerk, Library Staff and a Librarian. The Librarian can perform tasks that a Library Clerk or Library Staff can performs. Hence a *Librarian is both a Library Clerk and Library Staff.*

Implementation-1





Implementation-2 (Nested Classes)



Here in Implementation-2, we have one of the twin class as a nested class within the other twin class. In this implementation we don't have the dilemma - which twin class is to be used for subclassing unlike in Implementation-1.

**References**

Twin - A Design Pattern for Modeling Multiple Inheritance

http://ssw.jku.at/Research/Papers/Moe99/Paper.pdf

Twin Pattern - Wikipedia

https://en.wikipedia.org/wiki/Twin\_pattern