Tecniche di Progettazione: Design Patterns

Esercitazione e progetti

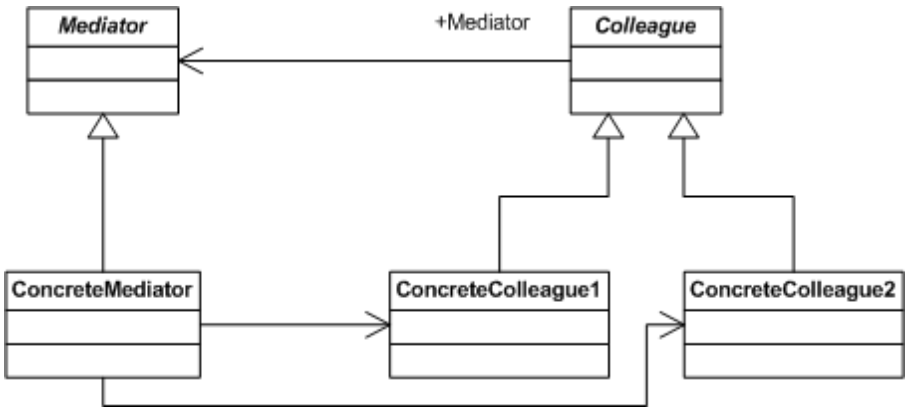
Summary

- In questa lezione alcuni esercizi e la presentazione dei progetti finali, da portare all'orale.

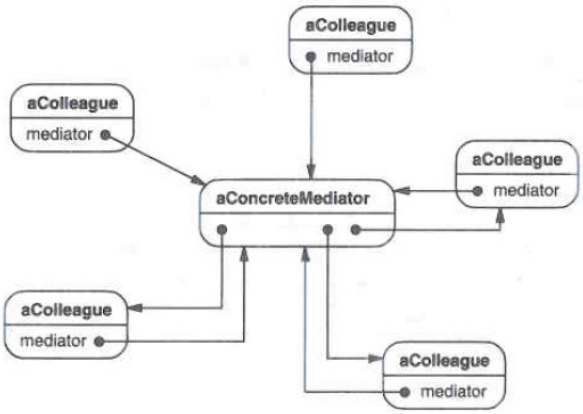
Ex: MediatorChatRoom

- This example demonstrates the Mediator pattern facilitating loosely coupled communication between different Participants registering with a Chatroom.
- The Chatroom is the central hub through which all communication takes place.
- At this point only one-to-one communication is implemented in the Chatroom, but would be trivial to change to one-to-many.

Mediator: structure



Structure

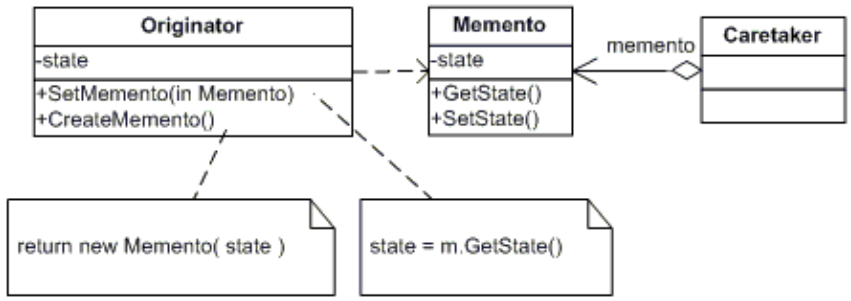


Ex: MementoSales

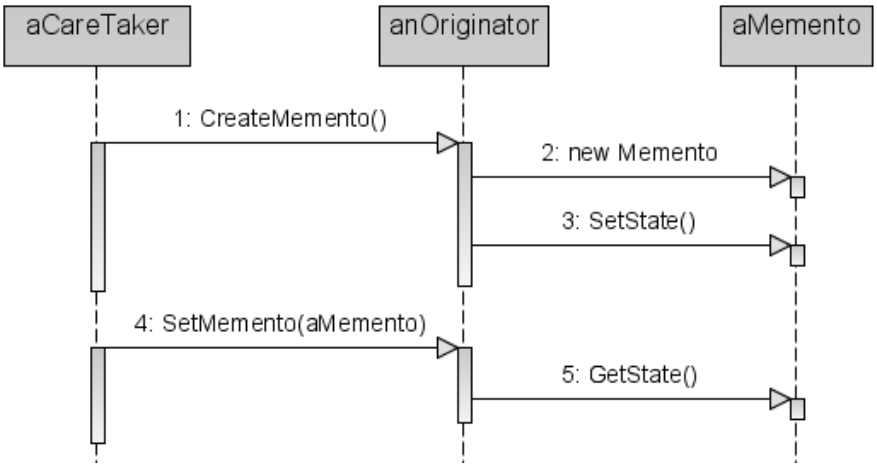
- This example demonstrates the Memento pattern which temporarily saves and then restores the SalesProspect's internal state.

- sistemare in modo che sia il caretaker a ordinare i set e get menento

Structure



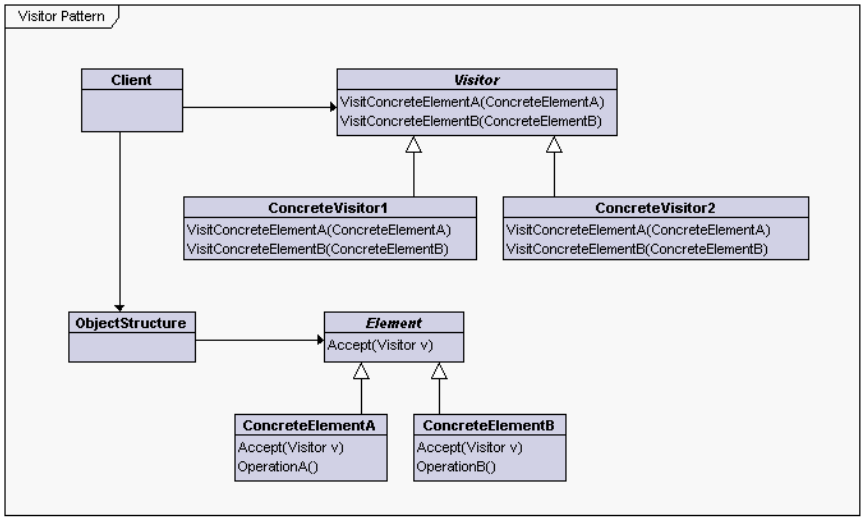
Collaboration



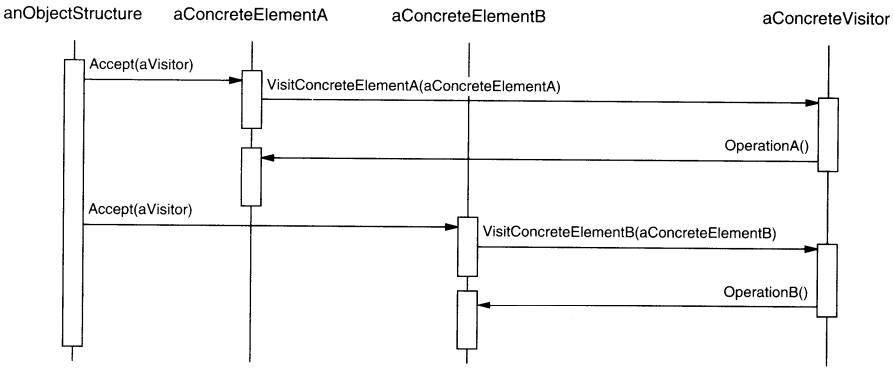
Ex: VisitorEmployee

- Two objects traverse a list of Employees and performs the same operation on each Employee.
- The two visitor objects define different operations -- one adjusts vacation days and the other income

Structure



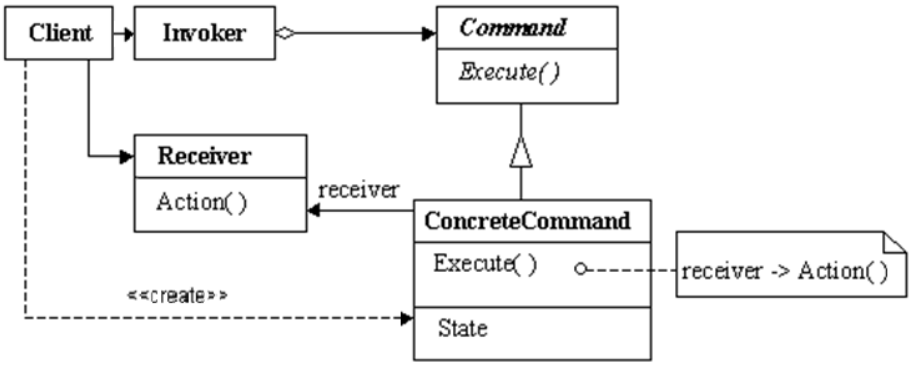
Visitor Pattern: Collaborations



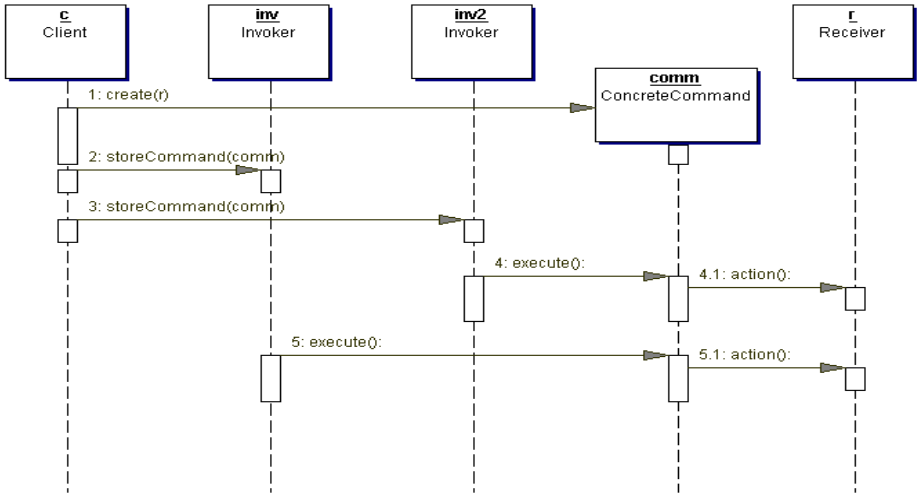
Ex: CommandDaCombinareConMemento

- pattern used in a simple calculator with unlimited number of undo's and redo's

The Command Pattern structure



Command: collaboration (with two invokers for a command)



Progetto 1: Game of Life

- The Game of Life is not your typical computer game. It is a 'cellular automaton', and was invented by Cambridge mathematician John Conway.
- This game became widely known when it was mentioned in an article published by Scientific American in 1970. It consists of a collection of cells which, based on a few mathematical rules, can live, die or multiply. Depending on the initial conditions, the cells form various patterns throughout the course of the game.
- http://www.bitstorm.org/gameoflife/

Rules

- The universe of the Game of Life is an infinite two-dimensional orthogonal grid of square *cells*, each of which is in one of two possible states, *live* or *dead*. Every cell interacts with its eight *neighbours*, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:
  1. Any live cell with fewer than two live neighbours dies, as if caused by under-population.
  2. Any live cell with two or three live neighbours lives on to the next generation.
  3. Any live cell with more than three live neighbours dies, as if by overcrowding.
  4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.
- The initial pattern constitutes the *seed* of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed—births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a *tick* (in other words, each generation is a pure function of the preceding one). The rules continue to be applied repeatedly to create further generations.

Progetto 2: RMI Auction Server Project

- In this project, you will be adding some functionality to the auction server you did in the midterm Project and also implementing a true client-server system using RMI and the Proxy pattern.
- As in the first project, the server will be used to maintain a list of items available for auction purchase. Clients will be allowed to make bids on available items or put new items up for auction. Clients can also be notified when the current bid on a particular item changes. In addition, the client will be able to specify different automatic bidding strategies. This application will require that both the client and server have remote objects.

Progetto 2: RMI Auction Server Project

- The server has a remote object which implements the following interface:
- public interface IAuctionServer extends Remote {

```
public void placeItemForBid(String ownerName, String itemName,
    String itemDesc, double startBid, int auctionTime) throws
    RemoteException;

public void bidOnItem(String bidderName, String itemName, double
    bid) throws RemoteException; public Item[] getItems() throws
    RemoteException;

public void registerListener(IAuctionListener al, String itemName)
    throws RemoteException
}
```

Progetto 2: RMI Auction Server Project

These methods do the following:

- public void placeItemForBid(String ownerName, String itemName, String itemDesc, double startBid, int auctionTime)

- Puts a new item up for auction by the owner with name ownerName. The itemName argument uniquely identifies the new item to be auctioned. If an item by that name already is up for auction in the server, a RemoteException is thrown. A description of the item is given by the itemDesc argument. The starting (minimum) bid is given by the startBid argument. The item will be available for auction for the number of seconds given by the auctionTime argument.

Progetto 2: RMI Auction Server Project

- public void bidOnItem(String bidderName, String itemName, double bid)

- The bidder with name bidderName makes a new bid on the item specified by the itemName argument. The bid amount is specified by the bid argument. For the bid to be accepted it must be higher than the current bid on the specified item, else a RemoteException is thrown.

Progetto 2: RMI Auction Server Project

- public Item[] getItems()
- Returns an array of items available for auction. Each Item object consists of the owner's name, item name, item description, current bid, current bidder's name and time remaining on the auction period for the item.

Progetto 2: RMI Auction Server Project

- public void registerListener(IAuctionListener al, String itemName)
- Registers a listener with the auction server for changes in the item specified by the itemName argument. Whenever the current bid on the specified item changes (or its auction period expires), the IAuctionListener is notified via its update() method. Note that the IAuctionListener object is a remote object!

Progetto 2: RMI Auction Server Project

- Any client object which desires to be notified of changes in the bid status of a specific item must implement the following interface:
- public interface IAuctionListener extends Remote { public void update(Item item) throws RemoteException; }

- The update() method of this interface does the following:
- public void update(Item item) Invoked by the auction server for each IAuctionListener which has registered to be notified of changes in the bid status of the specified item.