Mediator

Department of Computer Science
University of Pretoria

5 September 2023

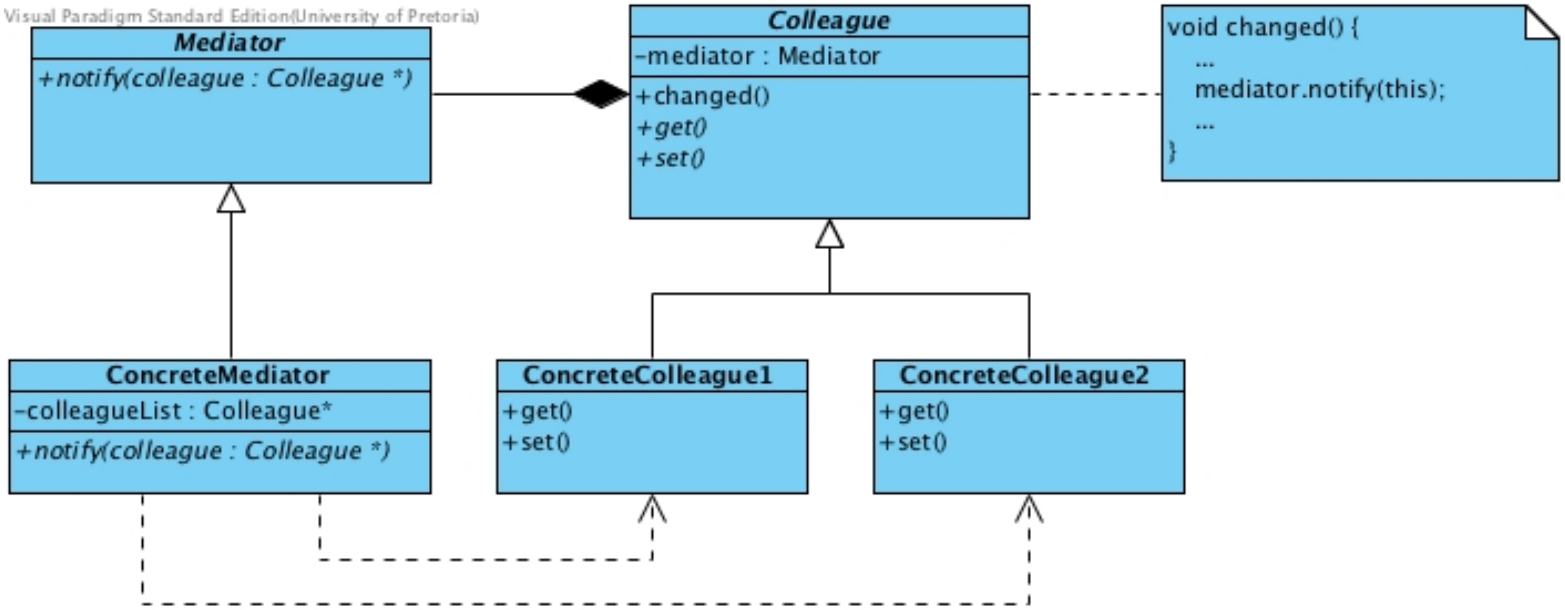
Name and Classification:

Mediator (Object Behavioural)

Intent:

"Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary the interaction independently."

GoF(273)


Visual Paradigm Standard Edition(University of Pretoria)
Mediator
+notify(colleague : Colleague *)
Colleague
-mediator : Mediator
+changed()
+get()
+set()
void changed() {
...
mediator.notify(this);
...
}
ConcreteMediator
-colleagueList : Colleague*
+notify(colleague : Colleague *)
ConcreteColleague1
+get()
+set()
ConcreteColleague2
+get()
+set()

Mediator

- defines an interface for communicating with Colleague objects

Concrete Mediator

- implements cooperative behaviour by coordinating Colleague objects
- knows and maintains colleagues

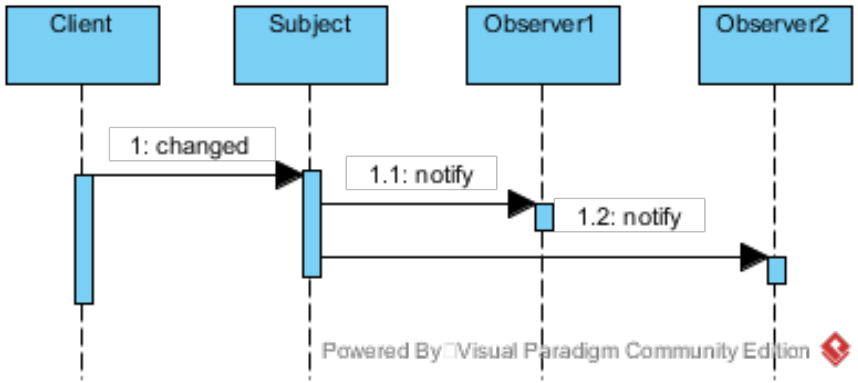
Colleagues

- each Colleague class knows its Mediator object
- each colleague communicates with its mediator whenever it would have otherwise communicated with another object

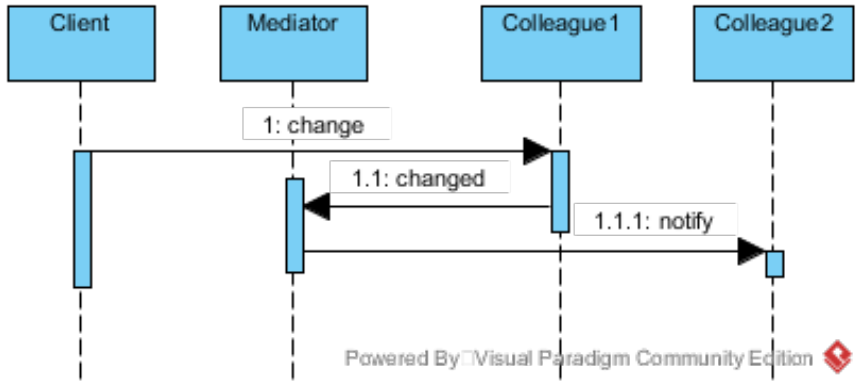
- **Façade** (105): has unidirectional behaviour - between the Facade interface to the subsystem objects, but not back. Mediator allows multidirectional communication.
- **Observer** (293): Colleagues can use the Observer pattern to communicate with the mediator.

The Mediator design pattern extends the Observer pattern.

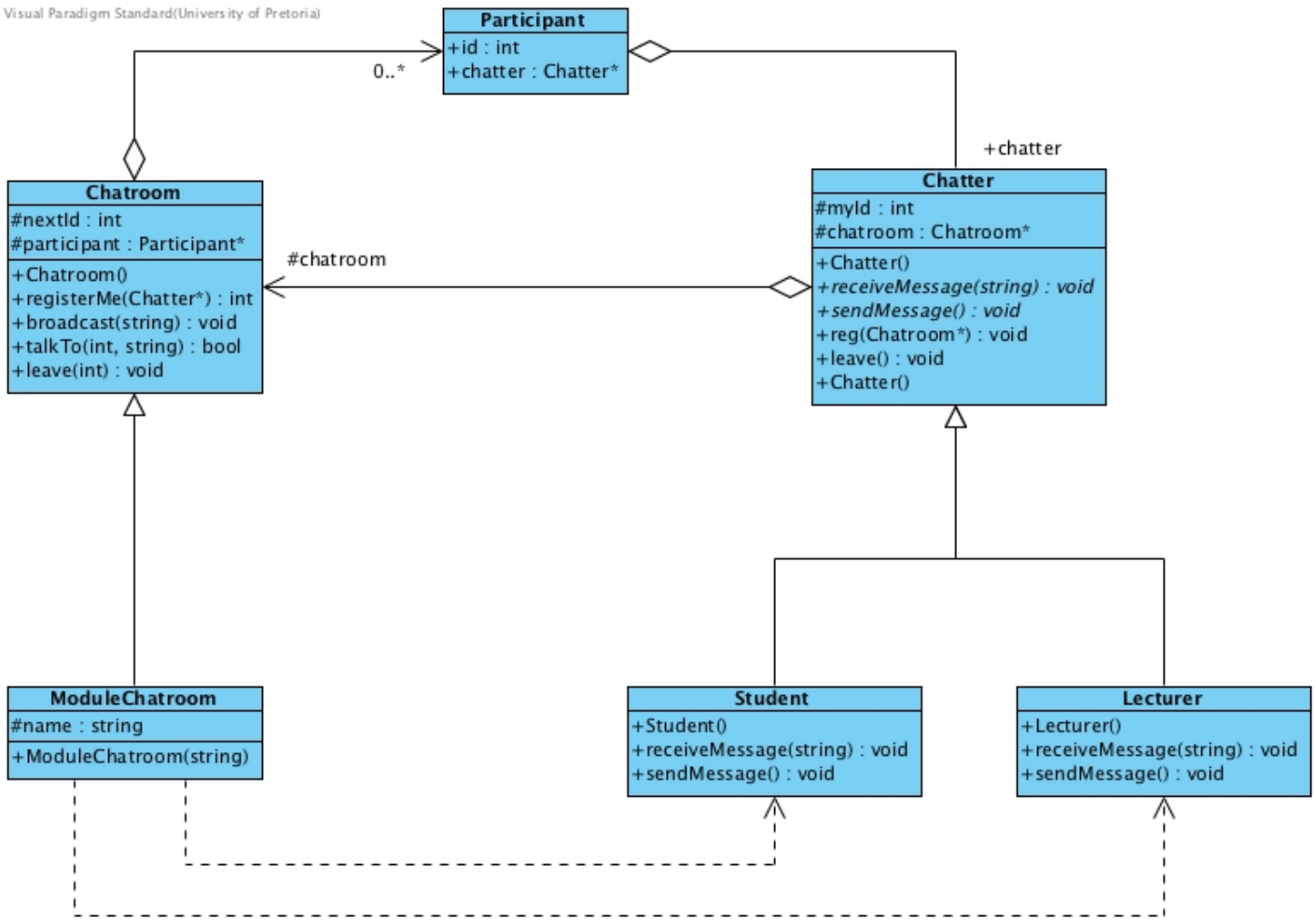
- The Observer registers observers that get updated whenever the subject changes,
- The Mediator registers colleagues that get updated whenever one of the other colleagues notifies the mediator of an update.



Observer

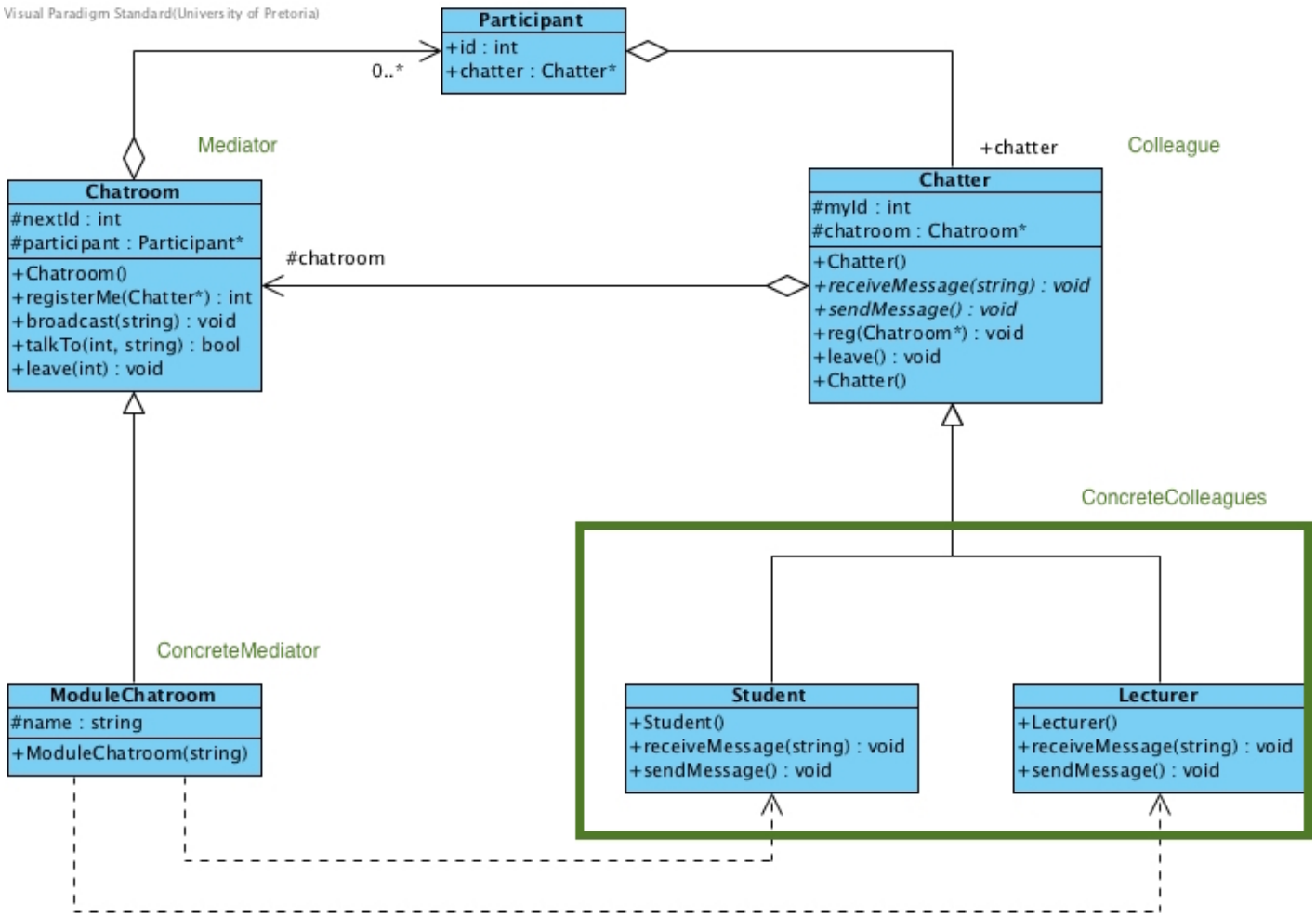


Mediator


Visual Paradigm Standard(University of Pretoria)
Participant
+id : int
+chatter : Chatter*
0..*
+chatter
Chatroom
#nextId : int
#participant : Participant*
+Chatroom()
+registerMe(Chatter*) : int
+broadcast(string) : void
+talkTo(int, string) : bool
+leave(int) : void
#chatroom
Chatter
#myId : int
#chatroom : Chatroom*
+Chatter()
+receiveMessage(string) : void
+sendMessage() : void
+reg(Chatroom*) : void
+leave() : void
+Chatter()
ModuleChatroom
#name : string
+ModuleChatroom(string)
Student
+Student()
+receiveMessage(string) : void
+sendMessage() : void
Lecturer
+Lecturer()
+receiveMessage(string) : void
+sendMessage() : void

Participants

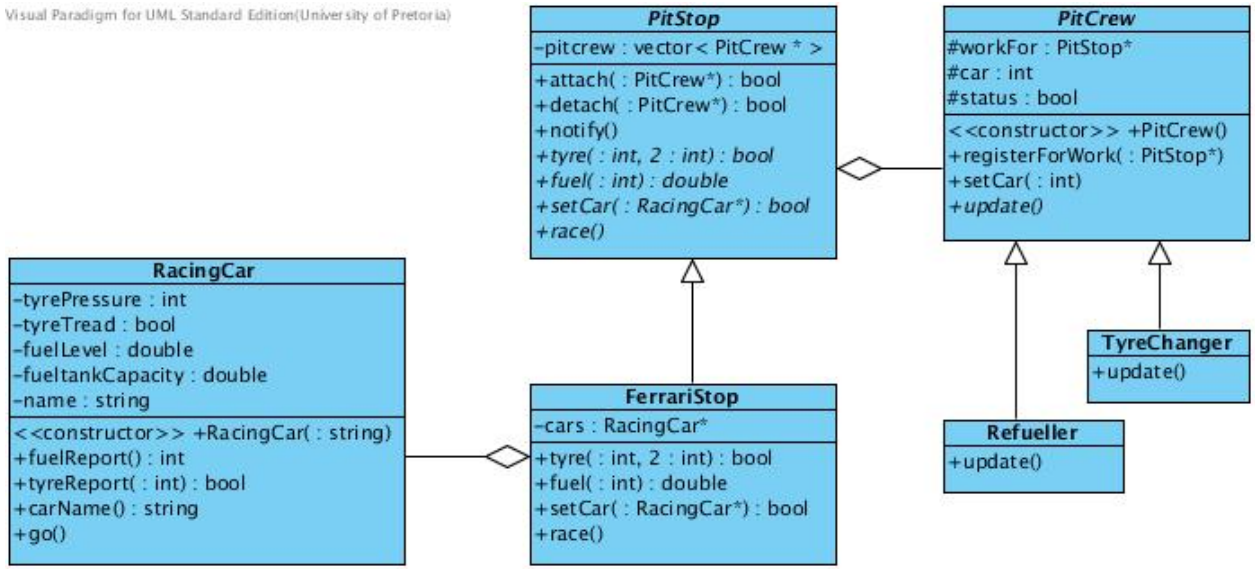
- Mediator - Chatroom
- ConcreteMediator - ModuleChatroom
- Colleague - Chatter
- ConcreteColleague - Student, Lecturer


Visual Paradigm Standard(University of Pretoria)
Participant
+id : int
+chatter : Chatter*
0..*
Mediator
+chatter
Colleague
Chatroom
#nextId : int
#participant : Participant*
+Chatroom()
+registerMe(Chatter*) : int
+broadcast(string) : void
+talkTo(int, string) : bool
+leave(int) : void
#chatroom
Chatter
#myId : int
#chatroom : Chatroom*
+Chatter()
+receiveMessage(string) : void
+sendMessage() : void
+reg(Chatroom*) : void
+leave() : void
+Chatter()
ConcreteColleagues
ConcreteMediator
ModuleChatroom
#name : string
+ModuleChatroom(string)
Student
+Student()
+receiveMessage(string) : void
+sendMessage() : void
Lecturer
+Lecturer()
+receiveMessage(string) : void
+sendMessage() : void

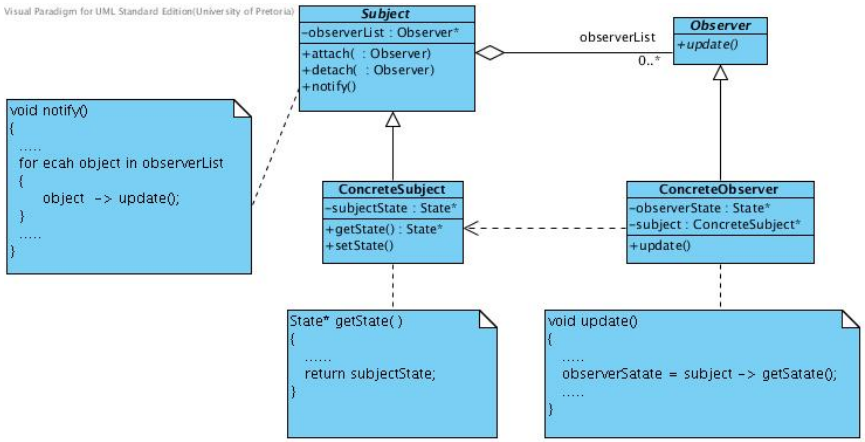
Draw a UML Sequence diagram for the following main program

```
int main() {
  Chatroom* cr = new ModuleChatroom("COS121");
  Chatter* student[2];

  Chatter* lecturer = new Lecturer();
  student[0] = new Student();  student[1] = new Student();

  lecturer->reg(cr);  student[0]->reg(cr);  student[1]->reg(cr);

  student[0]->sendMessage();  student[0]->sendMessage();
  student[1]->sendMessage();  lecturer->sendMessage();

  student[1]->leave();  lecturer->leave();
  lecturer->sendMessage();

  delete lecturer;
  for (int i = 0; i < 2; i++)  delete student[i];
  delete cr;

  return 0;
}
```

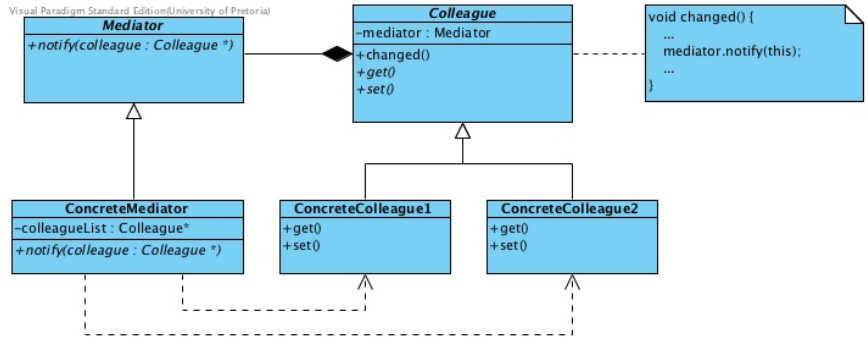
What do we have?



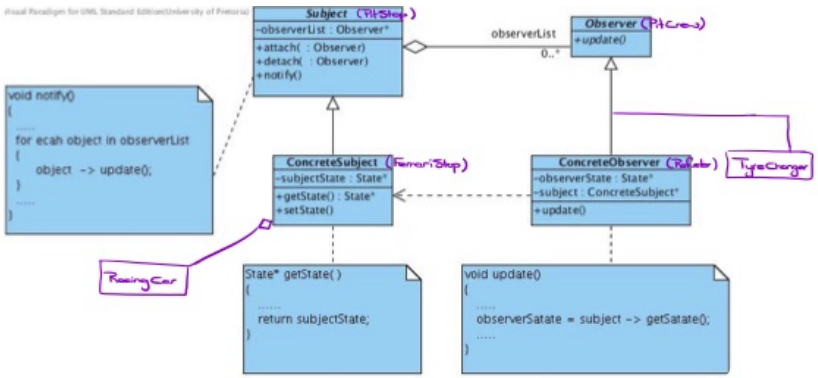
What do we have?



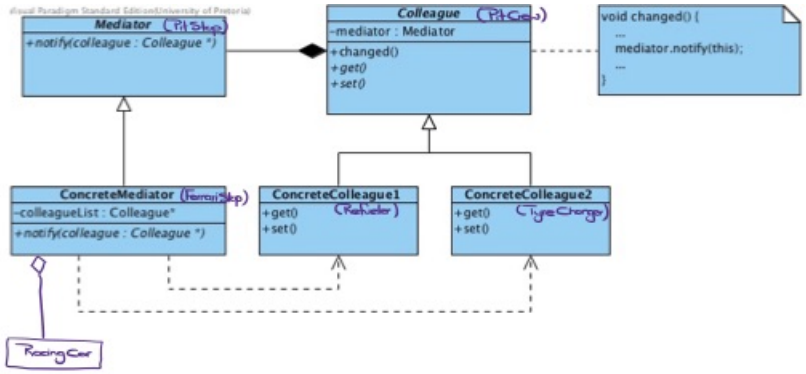
Observer



Mediator



Observer



Mediator