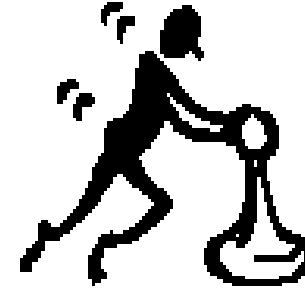


**Wie wird man ein guter
Schachspieler?**

**Wie wird man ein guter Software-
Designer?**

Wie wird man ein guter Schachspieler?



Lerne Regeln

Figurennamen und –zugmöglichkeiten, Schachbrettgeometrie ,...

Lerne Prinzipien

Relativer Figurenwert, strategischer Wert des Brettzentrums, Angriffsvermögen,...

Schließlich und vor allem:

Lerne von Spielen der „Meister“ Spielzüge und deren Anwendungssituationen:

Muster (Patterns).

Es gibt hunderte von solchen Spielzügen/Strategien

Wie wird man ein guter Software-Designer?

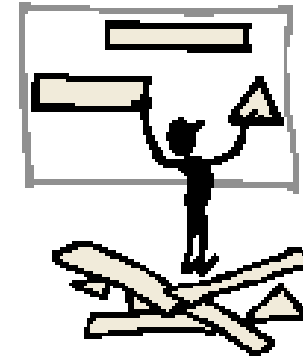
Lerne Regeln

Algorithmen, Datenstrukturen, Programmiersprachen

Lerne Techniken

Modulares Programmieren, objektorientiertes
Programmieren

Schließlich und vor allem: Lerne von SW-Designs der
„Meister“ Muster (Patterns) von allgemeinen Lösungen
immer wiederkehrender Designprobleme.



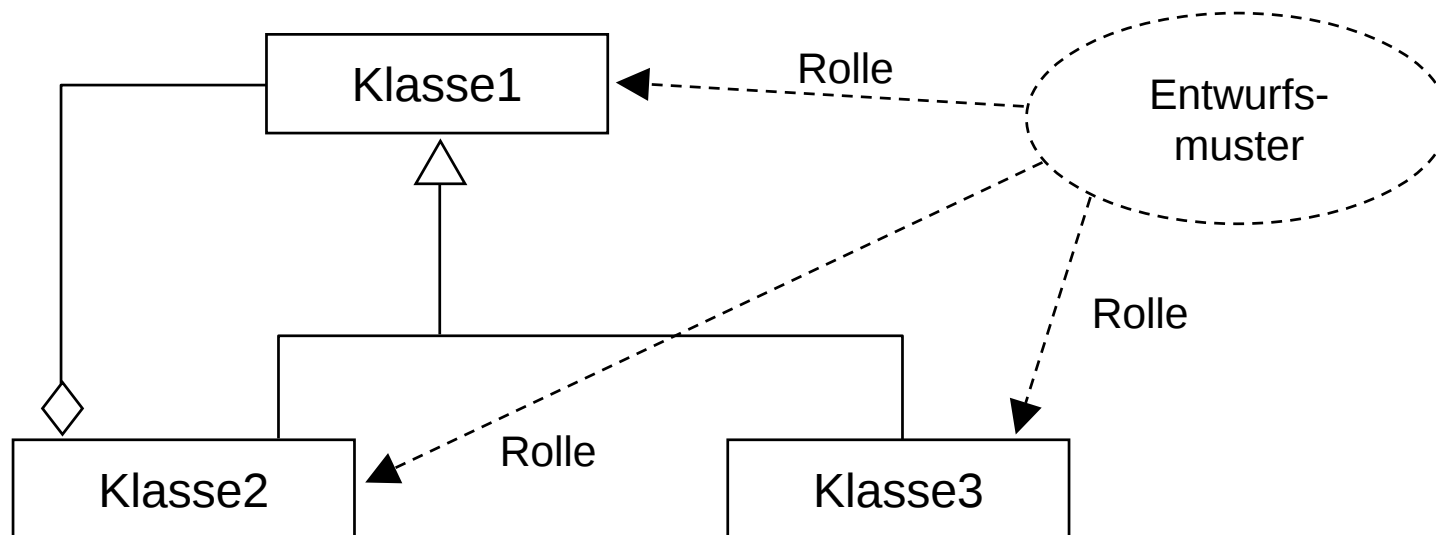
Was ist ein Entwurfsmuster (Design Pattern)?

- Ein Entwurfsmuster ist eine spezielle Komponente, die eine allgemeine, parametrisierte Lösung für ein typisches Entwurfsproblem bereitstellt.

Motivation:

- Bereitstellung bewährter, vorgefertigter Lösungen für wiederkehrende Entwurfsprobleme.
- Schaffung einer begrifflichen Basis und Terminologie für die Kommunikation über solche Probleme.

Design Pattern in der UML



Schema zur Beschreibung von Mustern:

Mustername:	Stichwort, welches das Entwurfsproblem, seine Lösungen und Auswirkungen beschreibt
Kontext:	In welchem Umfeld ist das Muster überhaupt relevant und einsetzbar?
Problem:	Welches Problem kann im gegebenen Kontext auftauchen und bedarf einer Lösung?
Lösung:	Wie kann das Problem gelöst werden? Welche Bestandteile sind nötig und welche Interaktionen finden zwischen diesen statt? Häufig begleitet von Klassendiagrammen und/oder Sequenzdiagrammen.

Schema zur Beschreibung von Mustern (Fortsetzung):

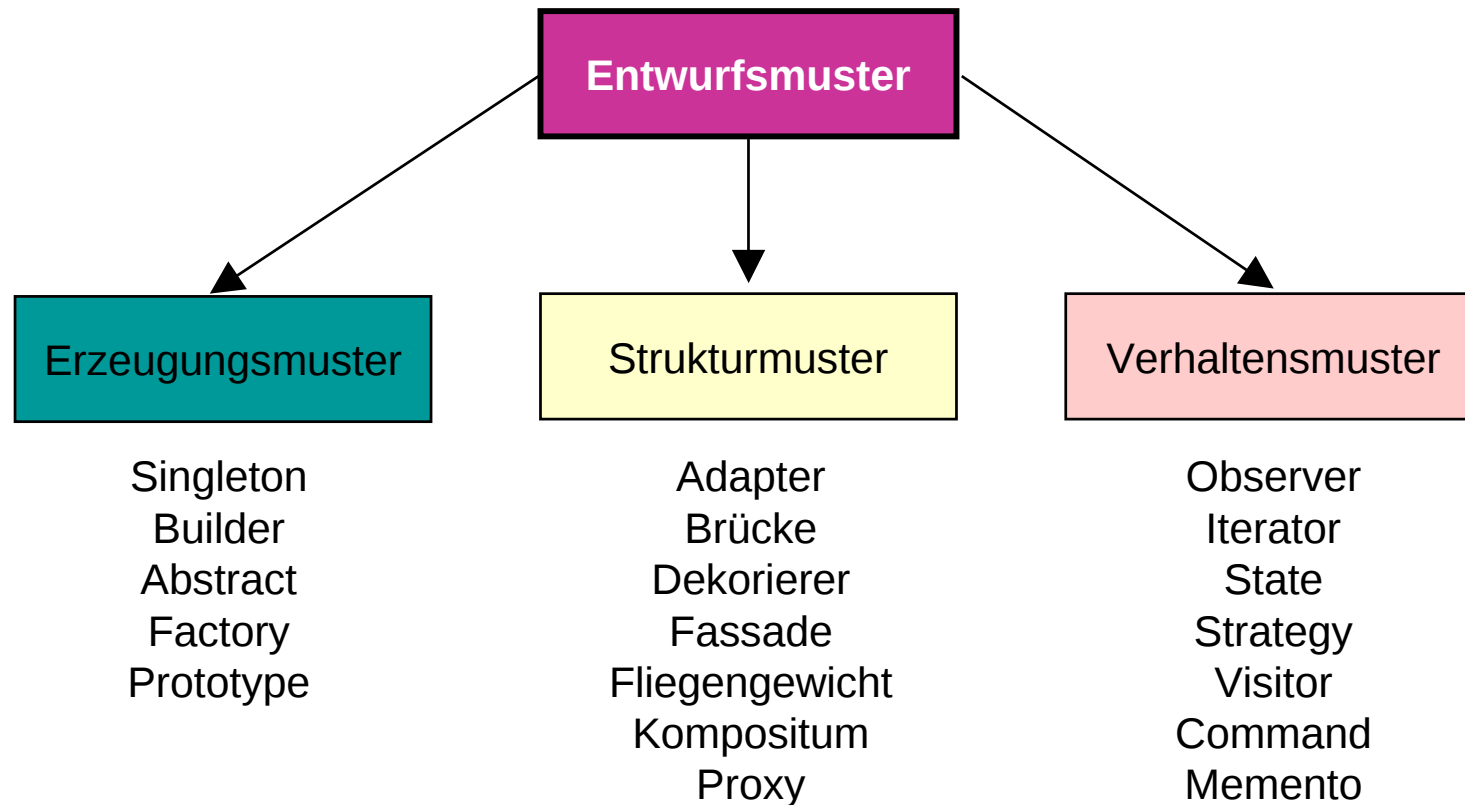
Beispiel: Ein Beispiel, bei dem Kontext und Problem gut erkennbar sind und die Lösung abgezeigt wird

Implementierungsvorschlag:

Wie kann ein Muster implementiert werden?

Konsequenzabschnitt:

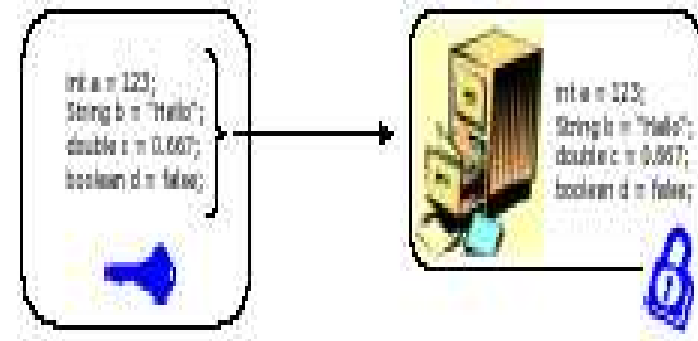
Auflistung der Vor- und Nachteile des resultierenden Entwurfs, d.h. die Konsequenzen der Musteranwendung



Das Memento – Design Pattern

Zweck:

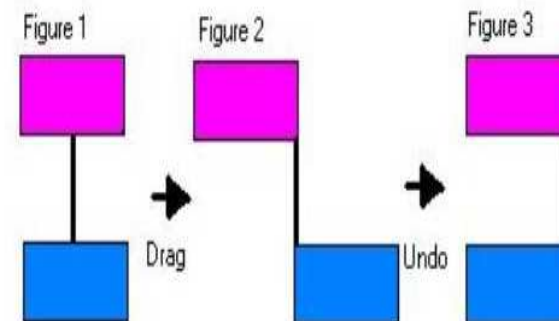
Erfasst und externalisiert den internen Zustand eines Objekts ohne Verletzung der Kapselung. Objekt kann später in diesen Zustand zurückversetzt werden.



Beispiel:

Motivation

- Wahrung der Kapselung
- Vereinfachung des Urhebers



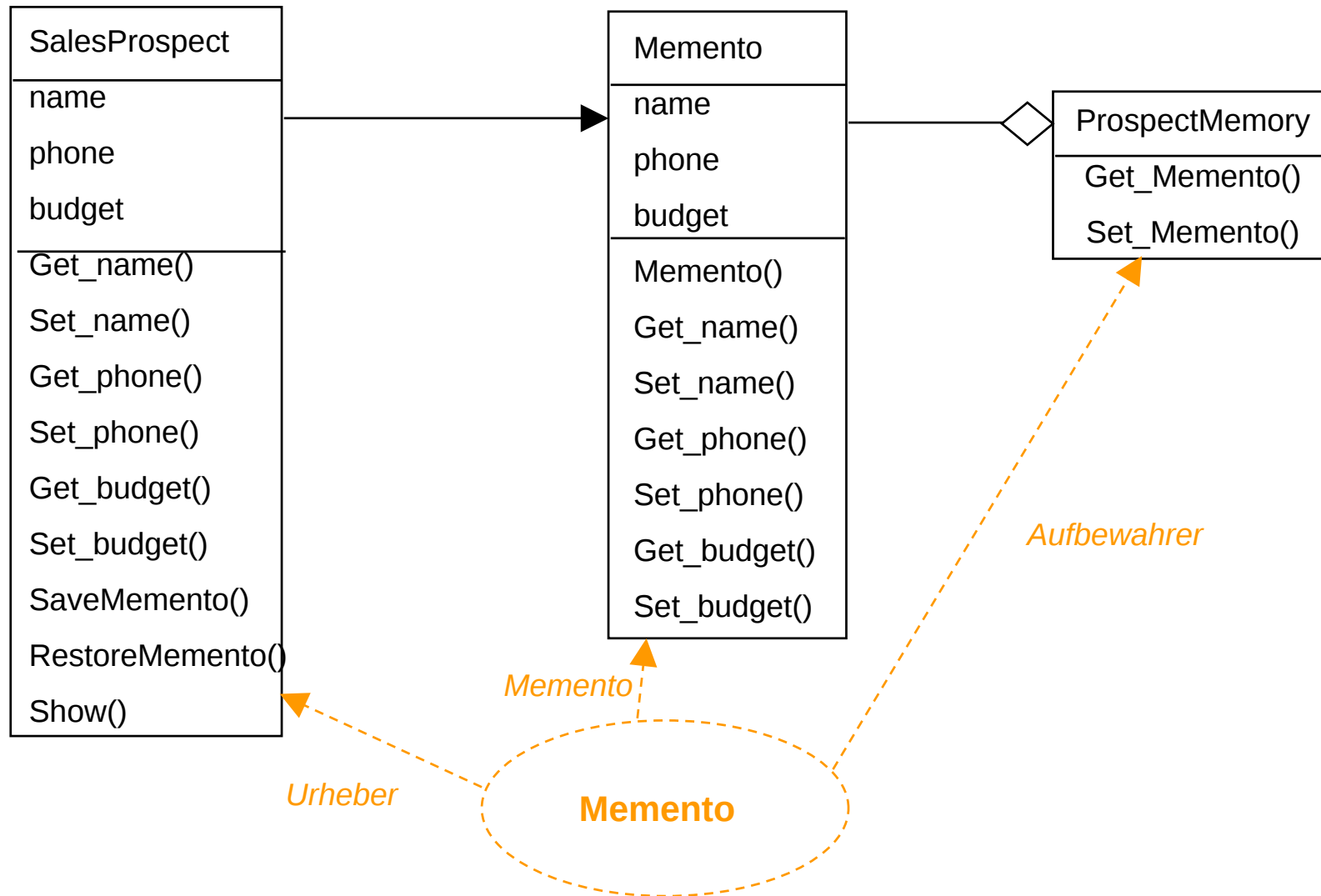
Grundgedanke

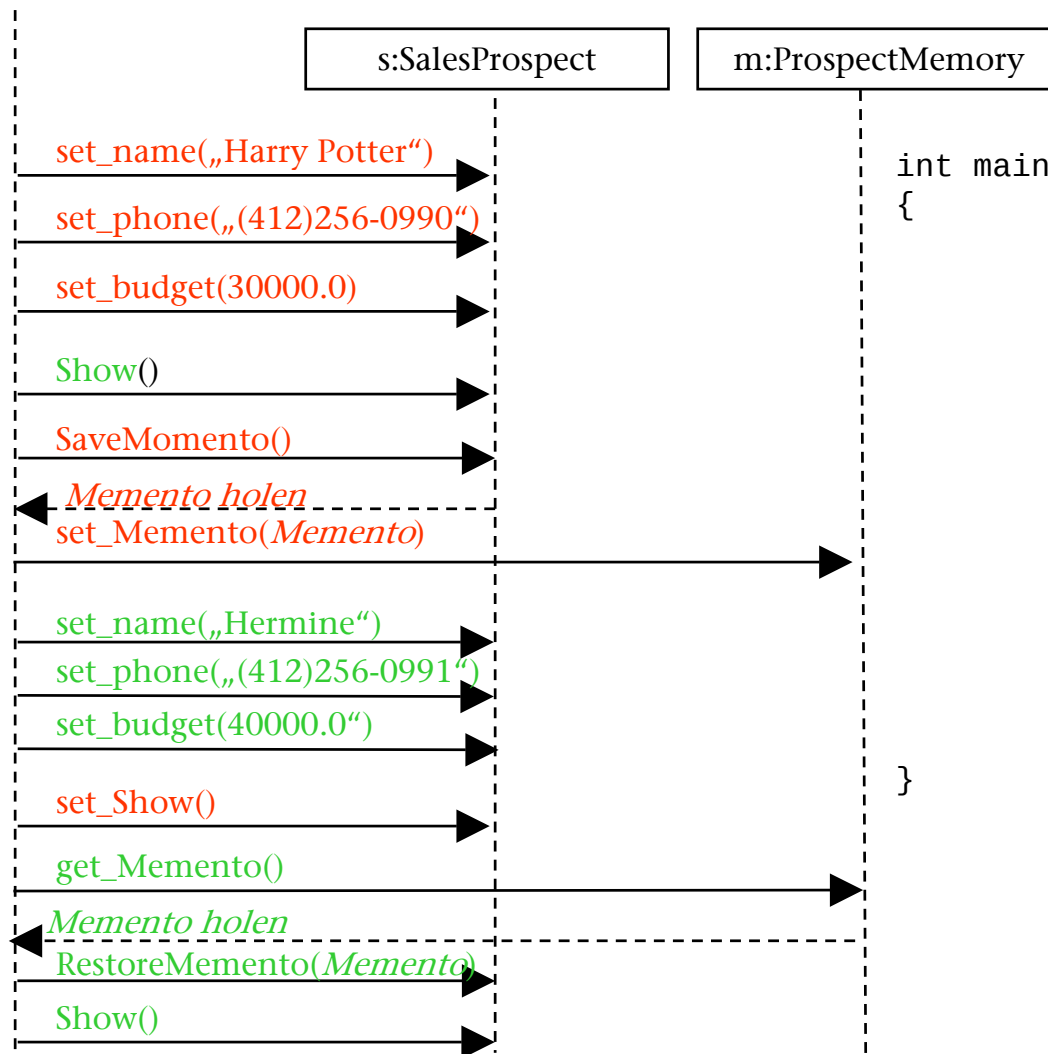
Erfasse und externalisiere den internen Zustand eines Objekts, ohne seine Kapselung zu verletzen, so dass das Objekt später in diesen Zustand zurückversetzt werden kann.



Verwenden Sie das **Memento** Design Pattern, wenn

- eine Momentaufnahme (eines Teils) des Zustands eines Objekts zwischengespeichert werden muss, so dass es zu einem späteren Zeitpunkt in diesen Zustand zurückversetzt werden kann.
- eine direkte Schnittstelle zum Ermitteln des Zustands die Implementierungsdetails offen legen und die Kapselung des Objekts aufbrechen würde.





```
int main (void)
```

```
{
```

```
SalesProspect s;
```

```
s.set_name("Harry Potter");
```

```
s.set_phone("(412) 256-0990");
```

```
s.set_budget(30000.0);
```

```
s.Show();
```

```
ProspectMemory m;
```

```
m.set_Memento(s.SaveMemento());
```

```
s.set_name("Hermine");
```

```
s.set_phone("(412) 256-0991");
```

```
s.set_budget(40000.0);
```

```
s.Show();
```

```
s.RestoreMemento(m.get_Memento());
```

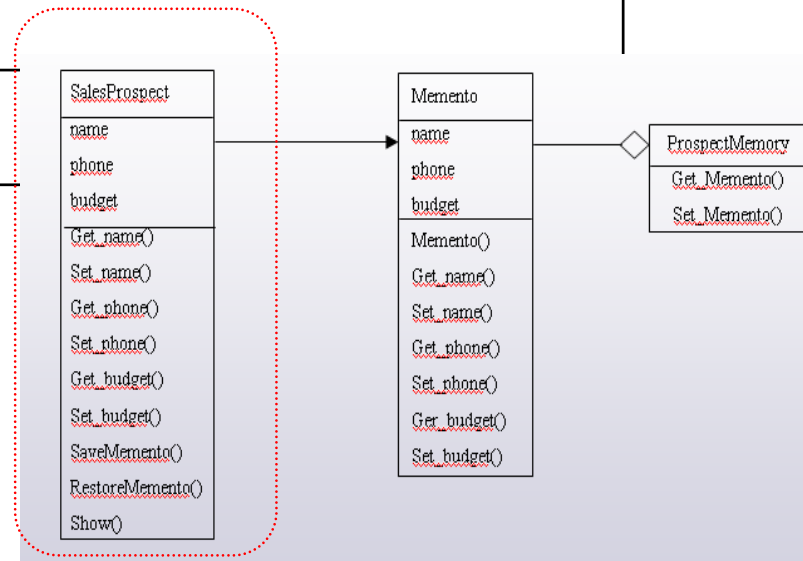
```
s.Show();
```

```
return 0;
```

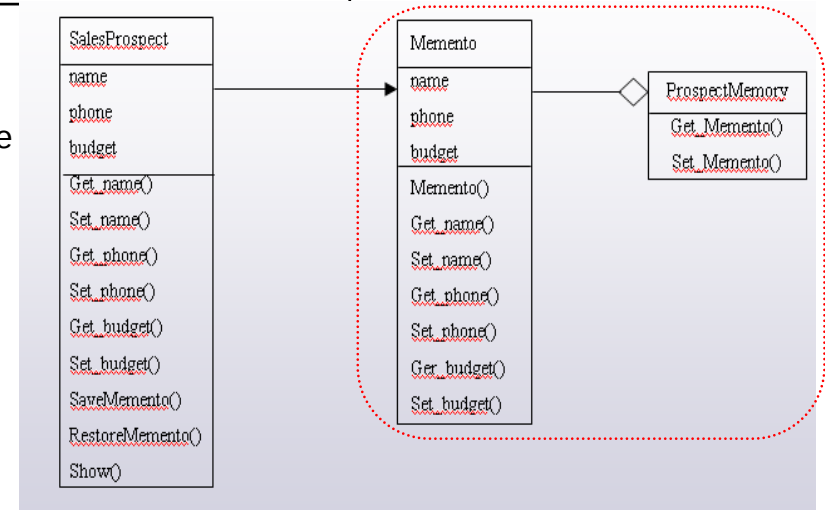
```
}
```

```

class SalesProspect
{
private:  char * name;
         char * phone;
         double budget;
public:   void set_name(char * name)
         { this->name=name;}
         void set_phone(char * phone)
         { this->phone=phone;}
         void set_budget(double budget)
         { this->budget=budget;}
         char * get_name(){return name;}
         char * get_phone(){return phone;}
         double get_budget(){return budget;}
         Memento * SaveMemento()
         {
             return new Memento(name,phone,budget);
         }
         void RestoreMemento(Memento * memento)
         {
             this->name=memento->get_name();
             this->phone=memento->get_phone();
             this->budget=memento->get_budget();
         }
         void Show()
         {
             cout<<"Sales prospect\n";
             cout<<"Name " <<this->name<<endl;
             cout<<"Phone " <<this->phone<<endl;
             cout<<"Budget " <<this->budget<<endl;
         }
};
    
```

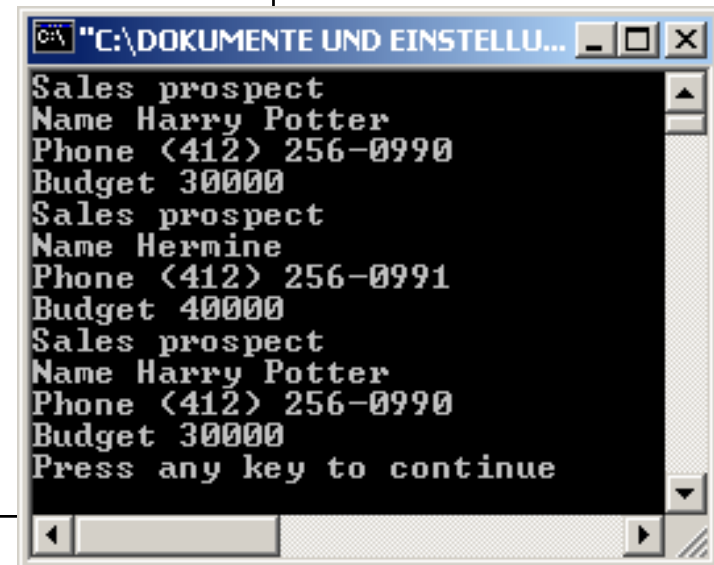


```
class Memento
{
private:  char * name;
         char * phone;
         double budget;
public:   Memento(char * name, char * phone, double
              {
                  this->name=name;
                  this->phone=phone;
                  this->budget=budget;
              }
         void set_name(char * name)
         {
             this->name=name;
         }
         void set_phone(char * phone)
         {
             this->phone=phone;
         }
         void set_budget(double budget)
         {
             this->budget=budget;
         }
         char * get_name(){return name;}
         char * get_phone(){return phone;}
         double get_budget(){return budget;}
};
```

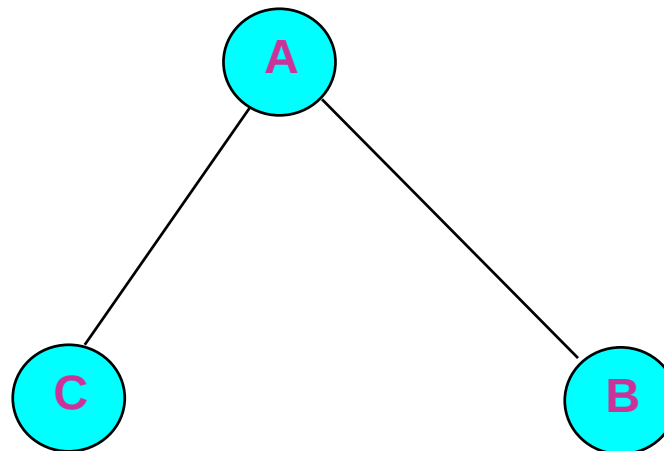


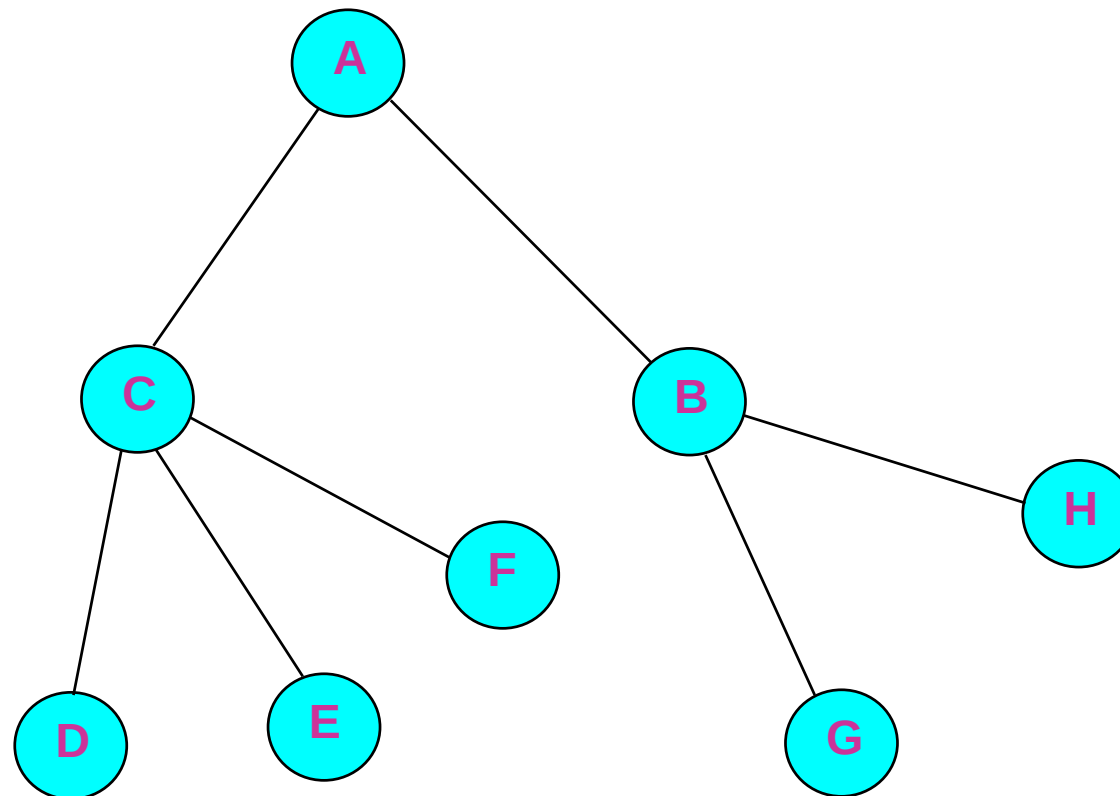
```
class ProspectMemory
{
private:  Memento * memento;
public:   void set_Memento(Memento * memento)
              {
                  this->memento=memento;
              }
         Memento * get_Memento()
         {
             return memento;
         }
};
```

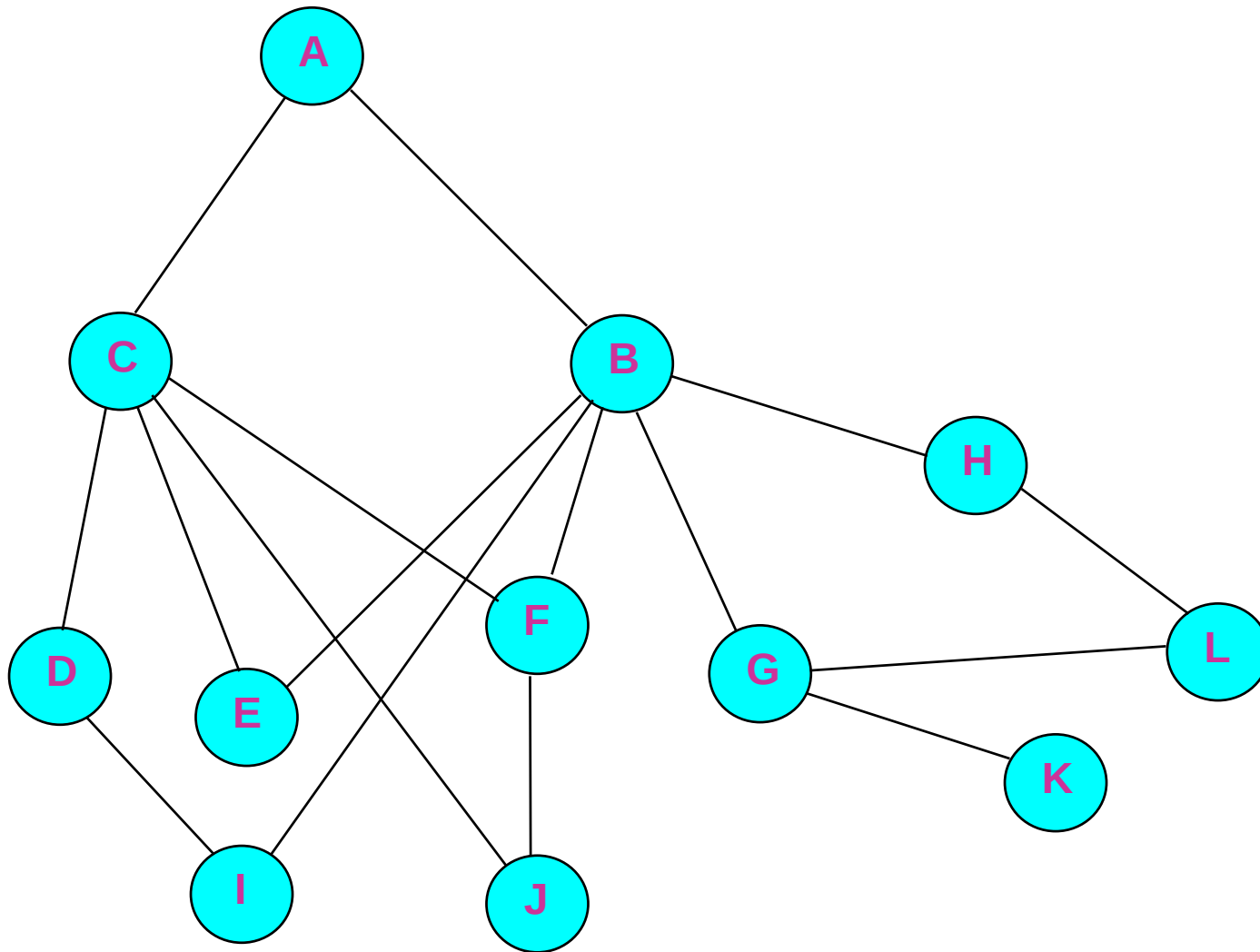
```
int main (void)
{
    SalesProspect s;
    s.set_name("Harry Potter");
    s.set_phone("(412) 256-0990");
    s.set_budget(30000.0);
    s.Show();
    ProspectMemory m;
    m.set_Memento(s.SaveMemento());
    s.set_name("Hermine");
    s.set_phone("(412) 256-0991");
    s.set_budget(40000.0);
    s.Show();
    s.RestoreMemento(m.get_Memento());
    s.Show();
    return 0;
}
```

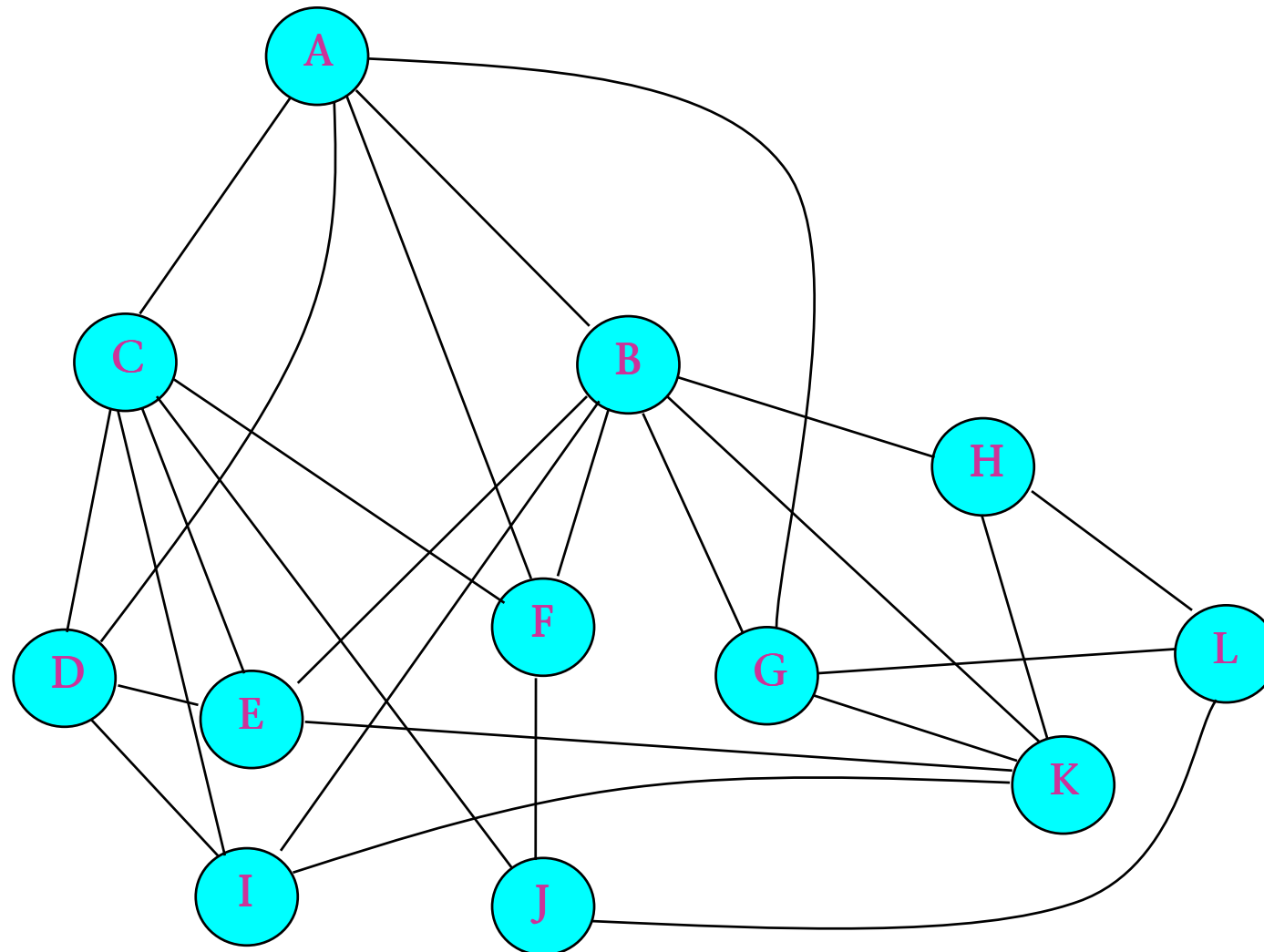


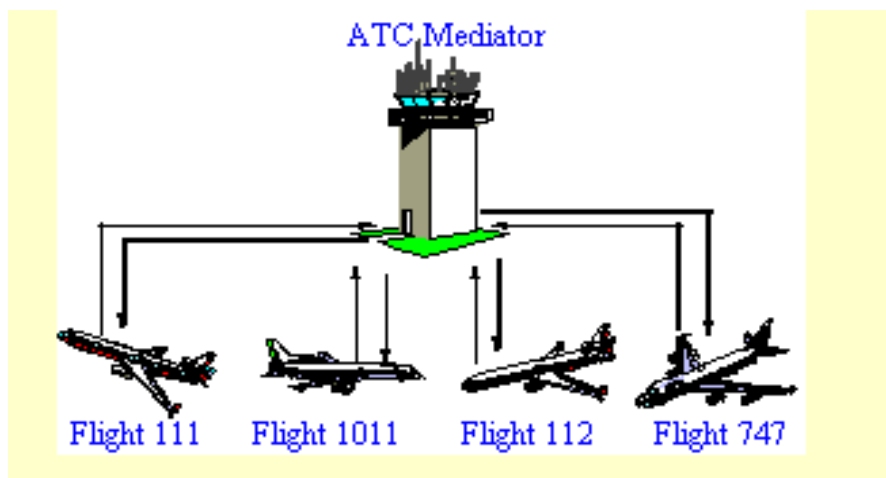
```
"C:\DOKUMENTE UND EINSTELLU...
Sales prospect
Name Harry Potter
Phone (412) 256-0990
Budget 30000
Sales prospect
Name Hermine
Phone (412) 256-0991
Budget 40000
Sales prospect
Name Harry Potter
Phone (412) 256-0990
Budget 30000
Press any key to continue
```











Chat Room

Chat Room enthält mehrere User.

Die User senden sich Nachrichten untereinander.

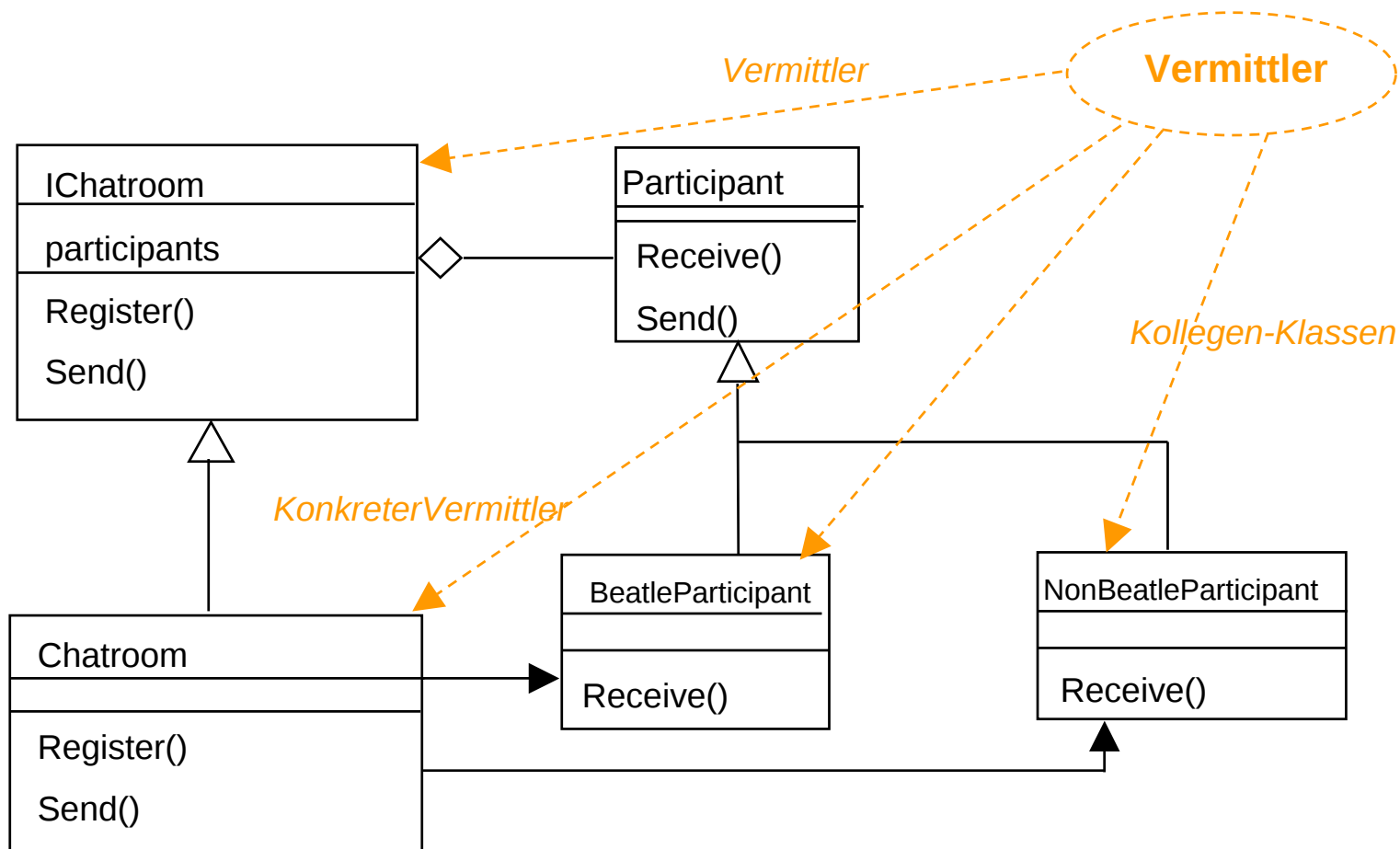
Die User können Nachrichten von anderen Usern empfangen.

Grundgedanke

Definiere ein Objekt, welches das Zusammenspiel einer Menge von Objekten in sich kapselt. Vermittler fördern lose Kopplung, indem sie Objekte davon abhalten, aufeinander explizit Bezug zu nehmen. Sie ermöglichen es ihnen, das Zusammenspiel von ihnen unabhängig zu variieren.

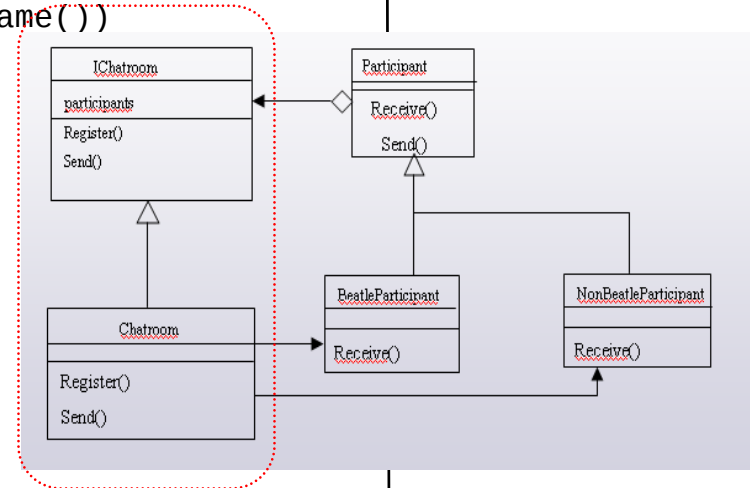


Verwenden Sie das Vermittler Design Pattern, wenn sie eine Menge von Objekten vorliegen haben, die in wohldefinierter, aber komplexer Weise miteinander zusammenarbeiten. Die sich ergebenden Abhängigkeiten sind unstrukturiert und schwer zu verstehen.



```

class Chatroom:public IChatroom
{
private:  std::list<Participant *>participants;
public:   void Register(Participant * participant)
        {   std::list<Participant *>::iterator pos;
            for(pos=participants.begin();
                pos!=participants.end();++pos)
            { if ((*pos)->get_name()==participant->get_name())
              {(*pos)->set_chatroom(this);
                return;
              }
            }
            participants.push_back(participant);
            participant->set_chatroom(this);
        }
        void Send(char * from,char * to, char * message)
        {   std::list<Participant *>::iterator pos;
            for (pos=participants.begin();
                pos!=participants.end();++pos)
            {   if((*pos)->get_name()==to)
                {   (*pos)->Receive(from,message);
                    return;
                }
            }
        }
};
    
```

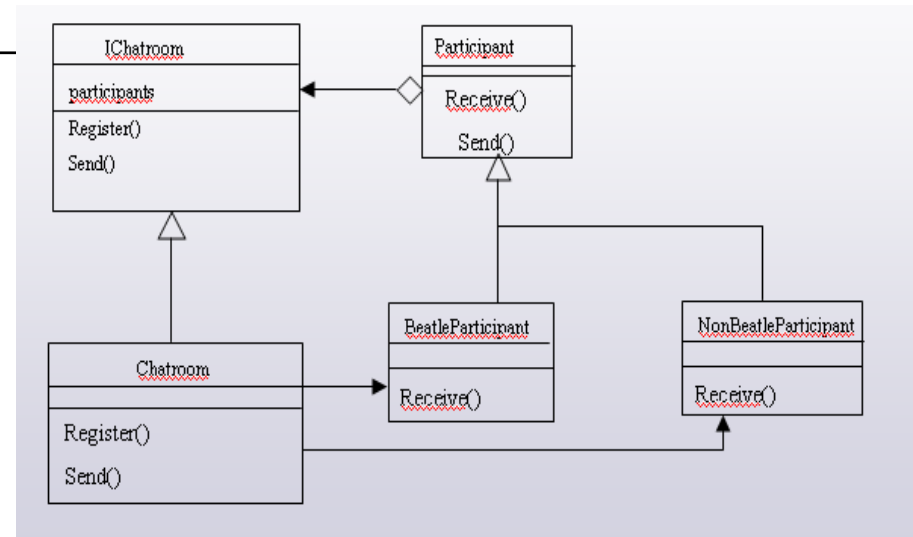


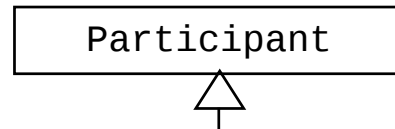
```

class IChatroom
{public:
    virtual void Register(Participant
                        *participant)=0;
    virtual void Send(char * frm,char * to,
                    char * message)=0;
};
    
```

```

class Participant
{
private: IChatroom * chatroom;
        char * name;
public:  Participant(char *name)
        {
            this->name=name;
        }
        char * get_name()
        {
            return name;}
        IChatroom * get_chatroom()
        {
            return chatroom;}
        void set_chatroom(IChatroom * chatroom)
        {
            this->chatroom=chatroom;}
        void Send(char * to,char * message)
        {
            chatroom->Send(name,to,message);
        }
        void Receive(char * from,char *message)
        {
            cout<<from<<" to "<<this->name<<": "<<message<<endl;
        }
};
    
```



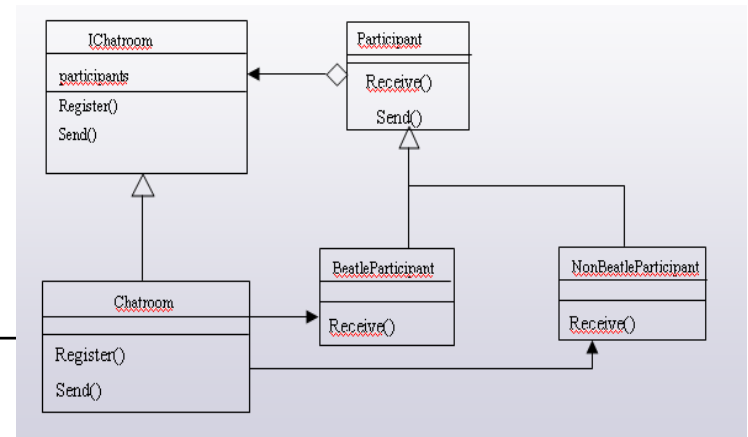


```

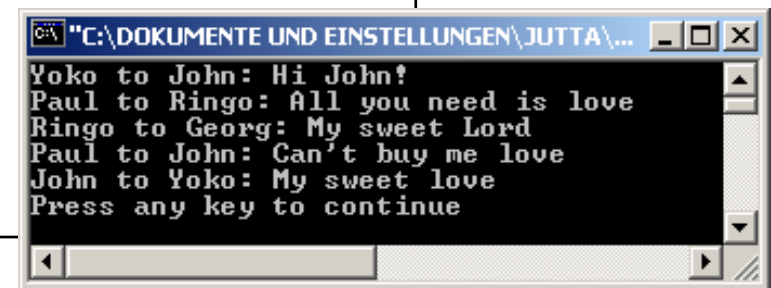
class BeatleParticipant:public Participant
{
public: BeatleParticipant(char * name):Participant(name){}
      void Reiceive(char * from,char * message)
      {
          cout<<"To a Beatle: ";
          Participant::Receive(from,message);
      }
};
    
```

```

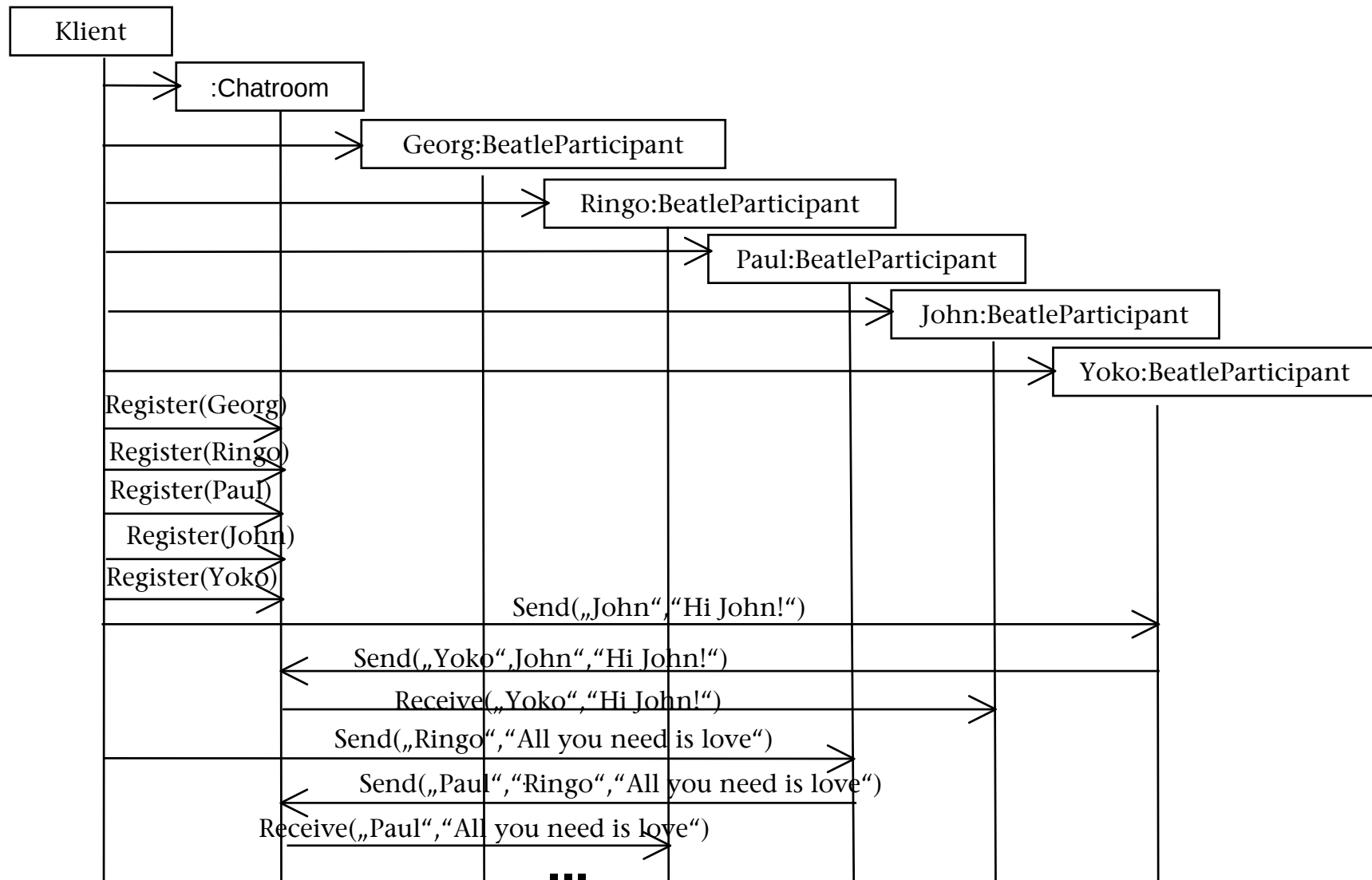
class NonBeatleParticipant:public Participant
{
public: NonBeatleParticipant(char * name):Participant(name){}
      void Reiceive(char * from,char * message)
      {
          cout<<"To a non-Beatle: ";
          Participant::Receive(from,message);
      }
}
    
```

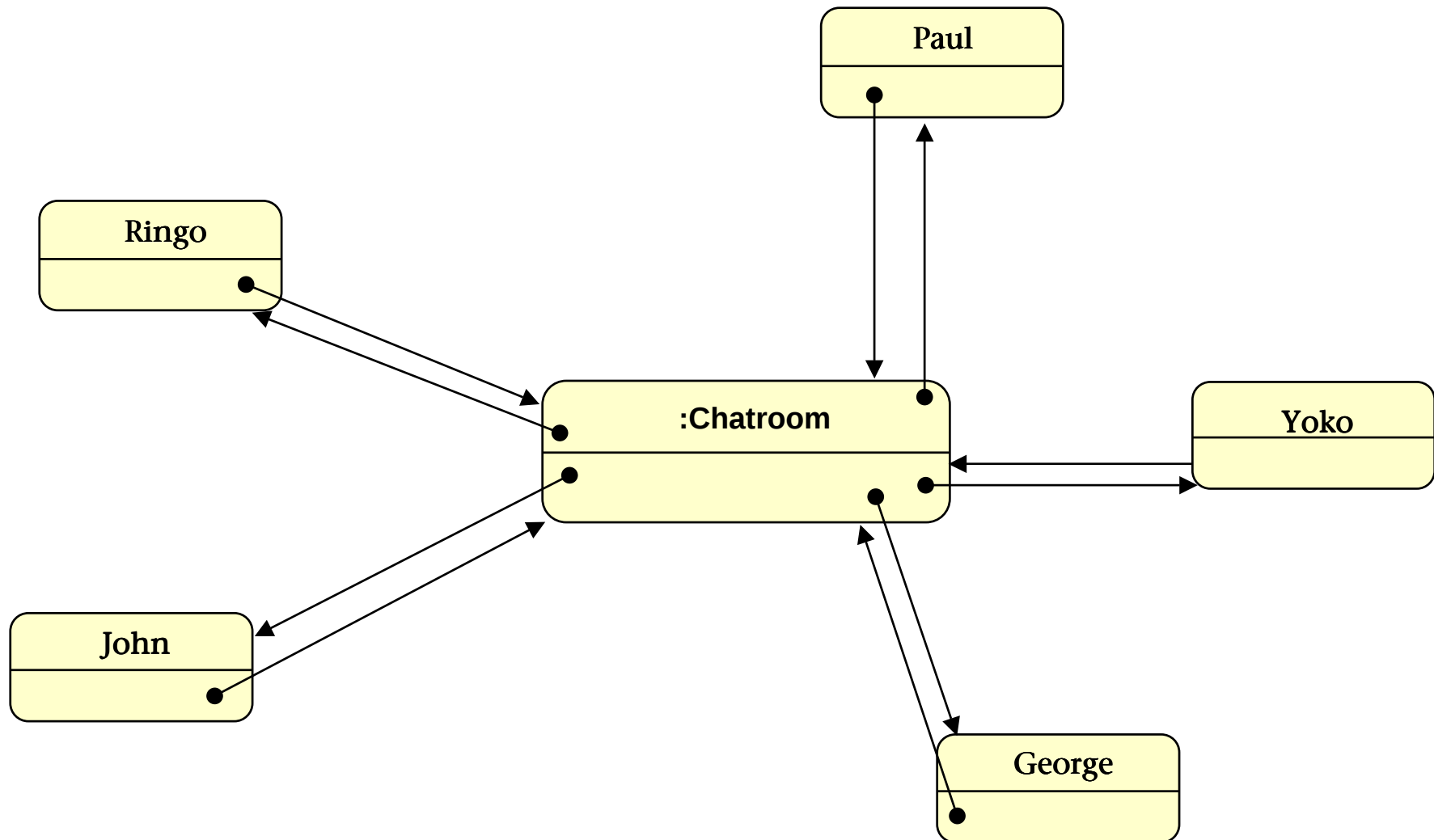


```
int main (void)
{
    IChatroom *c=new Chatroom();
    Participant * George=new BeatleParticipant("Georg");
    Participant * Ringo=new BeatleParticipant("Ringo");
    Participant * Paul=new BeatleParticipant("Paul");
    Participant * John=new BeatleParticipant("John");
    Participant * Yoko=new NonBeatleParticipant("Yoko");
    c->Register(George);
    c->Register(Ringo);
    c->Register(Paul);
    c->Register(John);
    c->Register(Yoko);
    Yoko->Send("John", "Hi John!");
    Paul->Send("Ringo", "All you need is love");
    Ringo->Send("Georg", "My sweet Lord");
    Paul->Send("John", "Can't buy me love");
    John->Send("Yoko", "My sweet love");
    return 0;
}
```



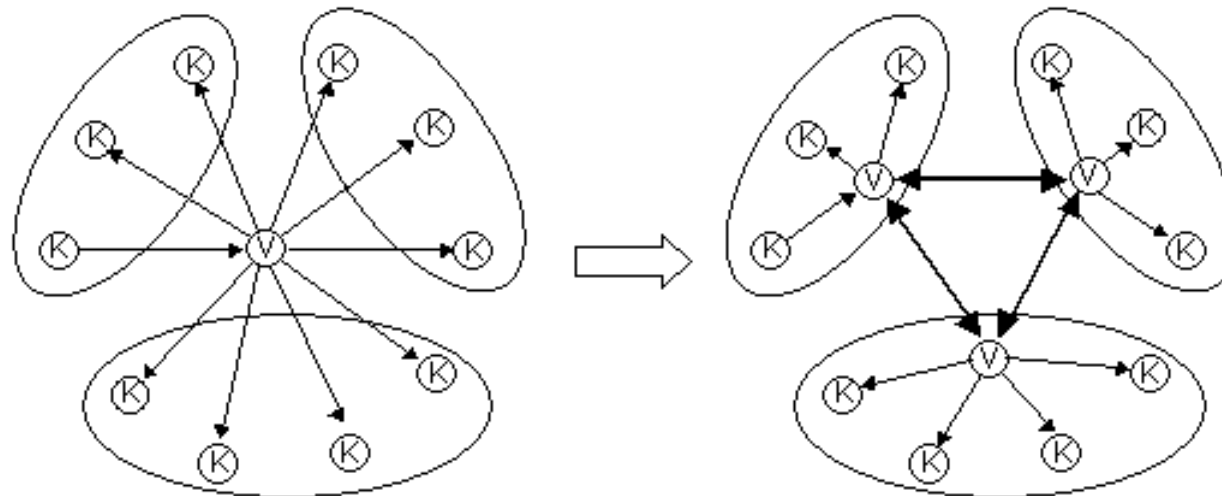
```
"C:\DOKUMENTE UND EINSTELLUNGEN\JUTTA\...
Yoko to John: Hi John!
Paul to Ringo: All you need is love
Ringo to Georg: My sweet Lord
Paul to John: Can't buy me love
John to Yoko: My sweet love
Press any key to continue
```





Tipp:

Befinden sich Kollegen und Vermittler auf unterschiedlichen Stationen eines Netzwerkes, ist der durch die Protokolle erzeugte Kommunikationsaufwand besonders zu beachten. Besteht die Möglichkeit einer Clusterbildung von Kollegen, welche sich auf derselben Station befinden (wie in der nachfolgenden Grafik illustriert), lässt sich der Kommunikationsaufwand reduzieren, indem man den Vermittler in Instanzen aufsplittet und jedem solchem Cluster eine Instanz zuordnet:



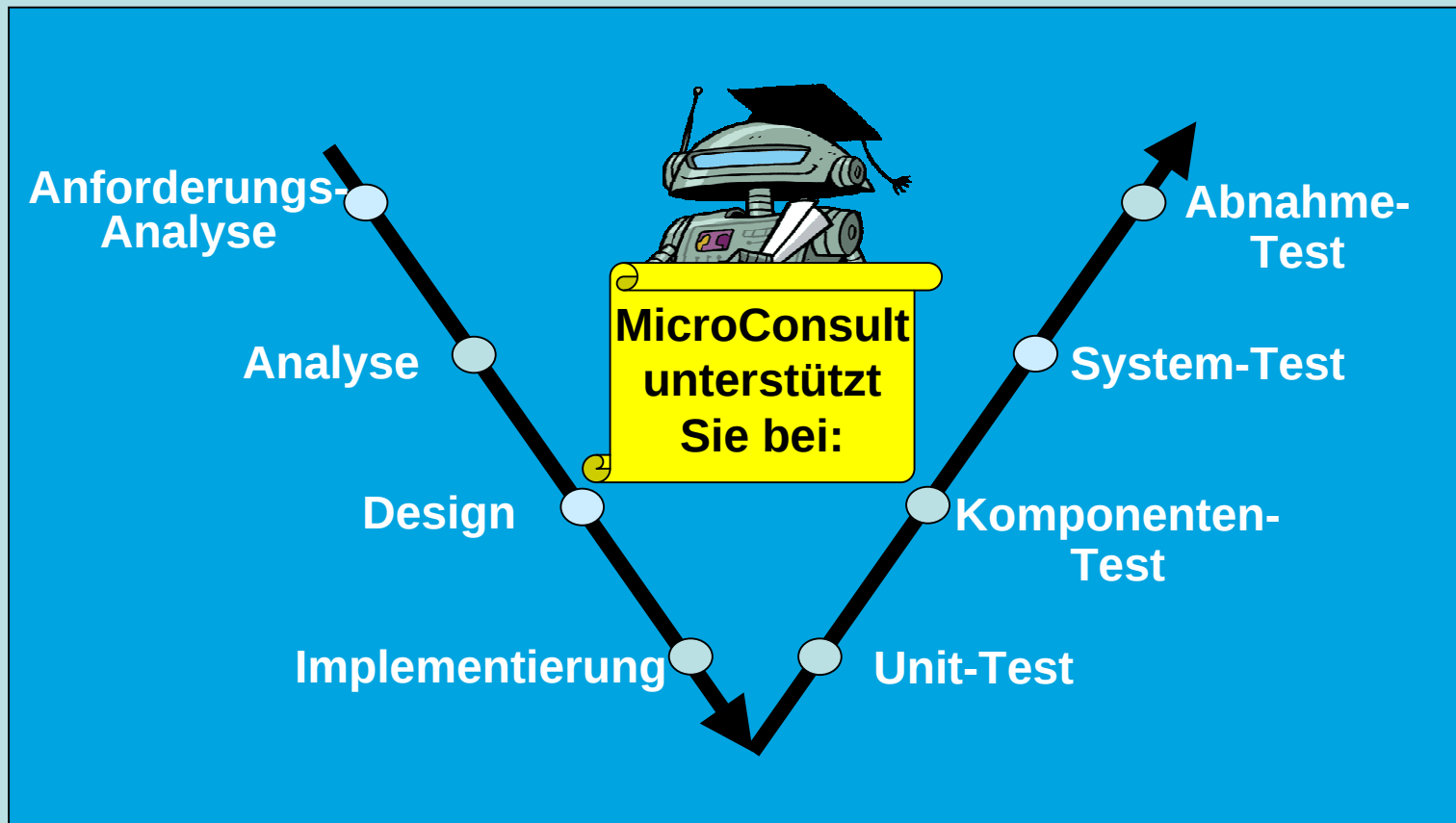
Entwurfsmuster

Addison-Wesley

E.Gamma/R.Helm/R.Johnson/J.Vlissides

ISBN 3-89319-950-0

Training, Coaching, Engineering



HW-/SW-Technologien, Tools, Methoden, Prozess, Team