

## D Überblick über die 3. Übung

D Überblick über die 3. Übung

- RPC und ORB
  - ◆ Aufrufsemantiken
- Hinweise Aufgabe 2

## D.1 Organisatorisches

D.1 Organisatorisches

- Präsentation der Übungsaufgaben 1 und 2
  - ◆ Live in der Rechnerübung am 15. oder 22. Nov.
  - ◆ Ein Gruppenmitglied ausreichend, Mitglieder müssen sich abwechseln
  - ◆ Alternativer Termin nach Rücksprache per eMail (mit Begründung)

## D.2 Object Request Brokers

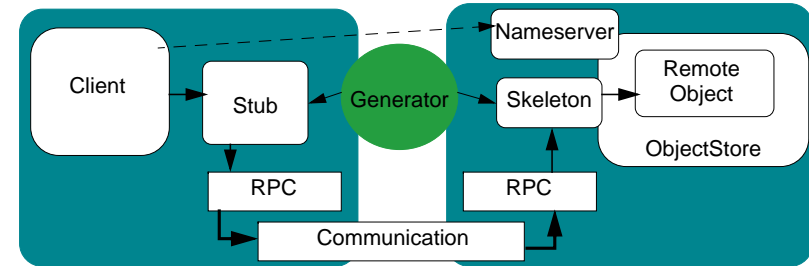
D.2 Object Request Brokers

- ermöglichen Methodenaufrufe an entfernten Objekten (Objekte in anderen JVM)
- Beispiel-ORBs: RMI, JavalDL

## 1 Komponenten eines ORBs

D.2 Object Request Brokers

- *Kommunikationsschicht*: tauscht Daten zwischen zwei Rechnern aus
- *RPC Schicht*: definiert die Aufrufsemantik und das Marshalling
- *Object Store*: verwaltet den Lebenszyklus der Objekte
- *Stub / Skeleton Generator*: erzeugt Code für die Stubs und die Skeletons
- *Namserver*: findet Objekte anhand deren Namen



Übungen zu Middleware

© Universität Erlangen-Nürnberg • Informatik 4, 2007

D-ORB.fm 2007-11-08 10.50

D.3

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Übungen zu Middleware

© Universität Erlangen-Nürnberg • Informatik 4, 2007

D-ORB.fm 2007-11-08 10.50

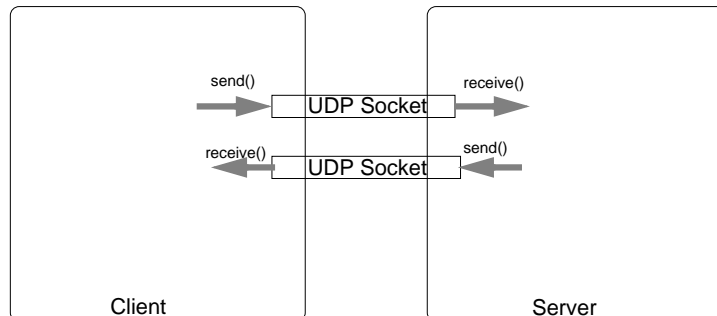
D.4

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

## 2 Kommunikationsschicht

D.2 Object Request Brokers

- zuständig für den Datenaustausch zwischen zwei Rechnern
- verwendet **DatagramSocket** (UDP)



## 3 RPC-Schicht

D.2 Object Request Brokers

- erledigt die Weiterleitung von Methodenaufrufen
- wird von den Stubs und den Skeletons verwendet
- verwendet die Kommunikationsschicht um Bytes zu versenden

Übungen zu Middleware

© Universität Erlangen-Nürnberg • Informatik 4, 2007

D-ORB.fm 2007-11-08 10.50

D.5

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Übungen zu Middleware

© Universität Erlangen-Nürnberg • Informatik 4, 2007

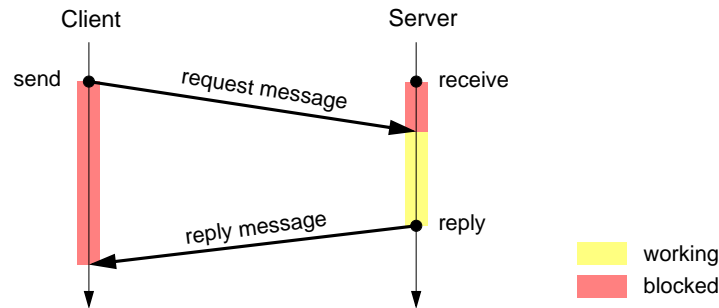
D-ORB.fm 2007-11-08 10.50

D.6

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

### 3 RPC

- einfachster RPC ist ein primitives (synchrones) request/reply-Protokoll:



### 3 Marshalling

- RPC-Schicht verpackt die Parameter in ein Anfragepaket und den Rückgabewert in ein Antwortpaket == Marshalling
- kopiere alle Parameter zum Server (besser wäre es, eine Referenz zu verschicken wenn ein Parameter ein *remote Interface* implementiert)

- verwende ObjectStreams/ByteArrayStreams/Datagramme um Objekte zu verschicken:

```

ByteArrayOutputStream stream = new ByteArrayOutputStream();
ObjectOutputStream out = new ObjectOutputStream(stream);
out.writeObject(...);
byte[] buf = stream.toByteArray();
DatagramPacket packet = new DatagramPacket(buf, ... );
  
```

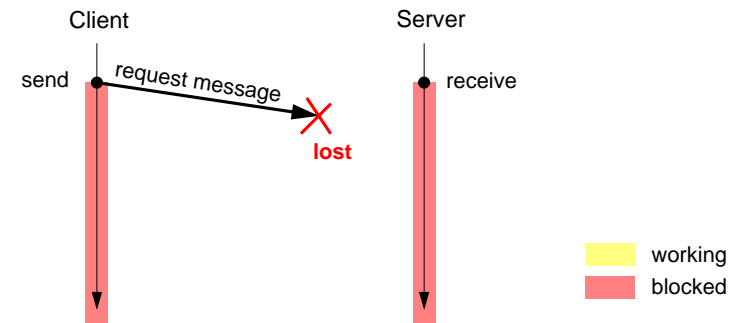
- Annahme für Aufgabe 2: alle Parameter passen in ein **DatagramPacket**

### 3 Fehlerbehandlung

- Implementierung einer bestimmten Aufrufsemantik
  - ◆ exactly once
  - ◆ at least once
  - ◆ at most once
  - ◆ last of many
  - ◆ ...
- abhängig von der Servicequalität der Kommunikationsschicht
- muss mit Kommunikationsfehlern umgehen können:
  - ◆ verlorene Pakete
  - ◆ veränderte Reihenfolge (non-FIFO)
  - ◆ duplizierte Pakete
  - ◆ veränderte Pakete

### 3 Fehlerbehandlung - Problem 1

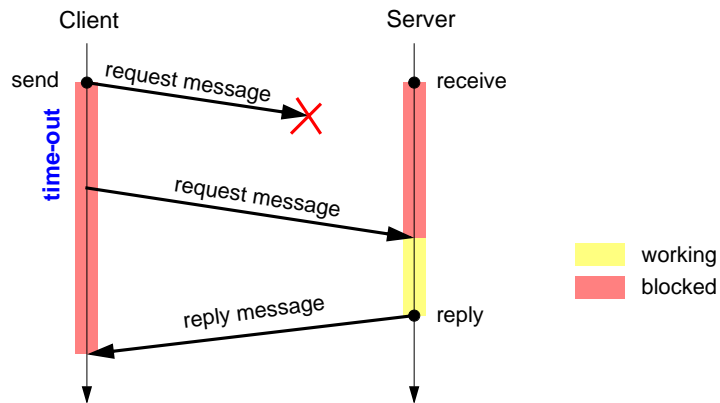
- Kommunikationsfehler: Verlust einer Anfragennachricht



### 3 Fehlerbehandlung - Lösung 1

D.2 Object Request Brokers

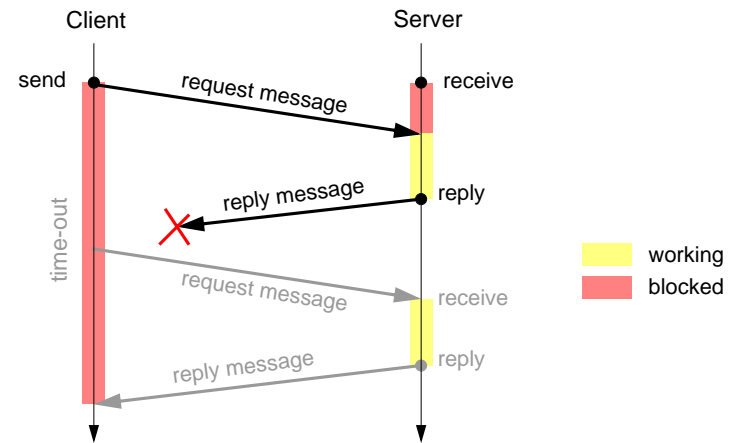
- Verlust einer Anfragenachricht:
  - ◆ mögliche Lösung: Anfrage erneut schicken



### 3 Fehlerbehandlung - Problem 2

D.2 Object Request Brokers

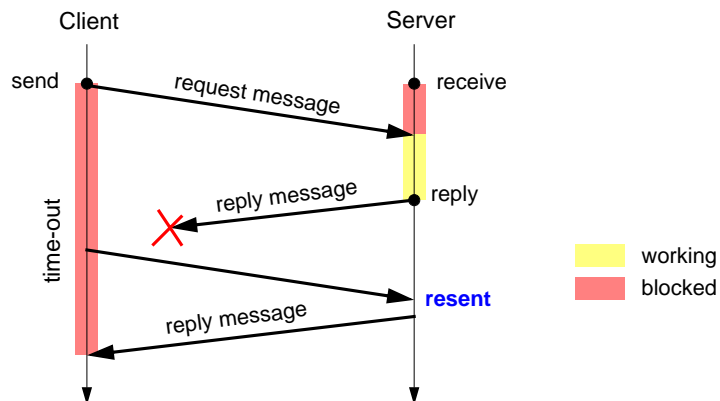
- Kommunikationsfehler: Verlust einer Antwortnachricht
  - ◆ bei Wiederholung wird die Methode mehrmals ausgeführt



### 3 Fehlerbehandlung - Lösung 2

D.2 Object Request Brokers

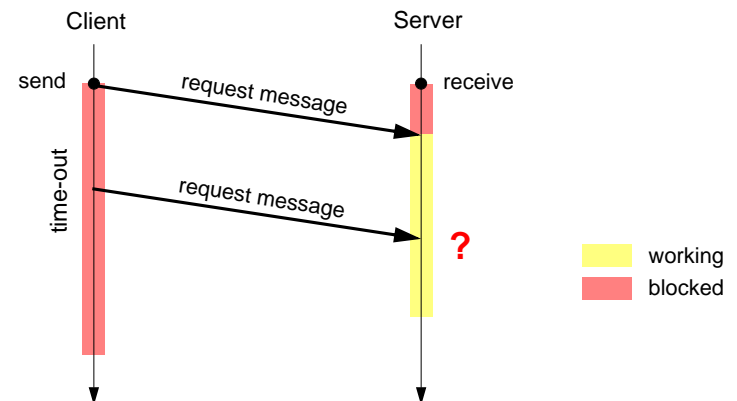
- Verlust einer Antwortnachricht
  - ◆ Server hebt Antwort auf und wiederholt sie



### 3 Fehlerbehandlung - Problem 3

D.2 Object Request Brokers

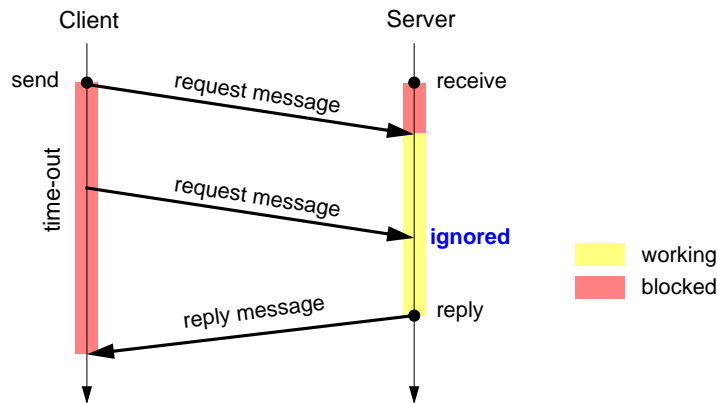
- Bearbeitung dauert länger als Timeout bei Client
  - ◆ Bearbeitung ist noch nicht fertig



### 3 Fehlerbehandlung - Lösung 3

D.2 Object Request Brokers

- Bearbeitung ist noch nicht fertig
  - ◆ wiederholte Anfrage wird ignoriert



Übungen zu Middleware  
© Universität Erlangen-Nürnberg • Informatik 4, 2007

D-ORB.fm 2007-11-08 10.50

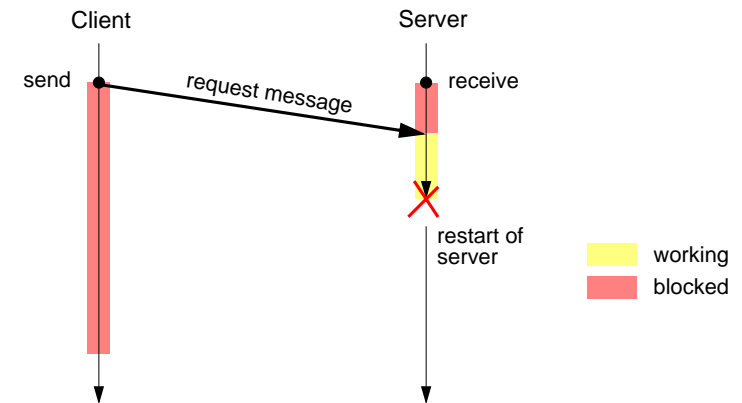
D.15

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

### 3 Fehlerbehandlung - Problem 4

D.2 Object Request Brokers

- Serverfehler
  - ◆ Fehler während des Ausführens der Methode (Server stürzt ab)



Übungen zu Middleware  
© Universität Erlangen-Nürnberg • Informatik 4, 2007

D-ORB.fm 2007-11-08 10.50

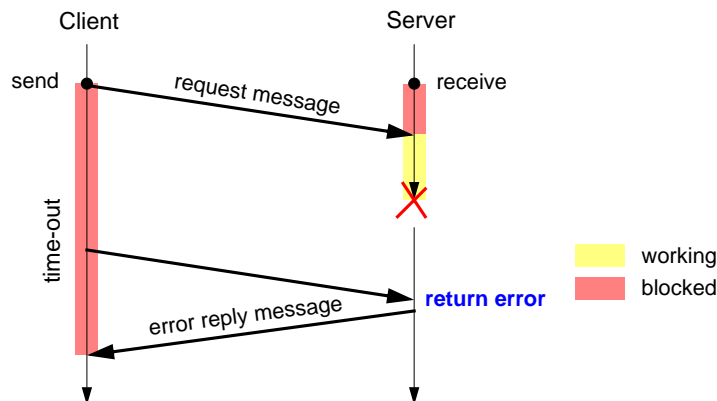
D.16

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

### 3 Fehlerbehandlung - Lösung 4

D.2 Object Request Brokers

- Server stürzt ab:
  - ◆ Server erkennt alte Anfragen (alte Generationsnummer) und liefert Fehler



Übungen zu Middleware  
© Universität Erlangen-Nürnberg • Informatik 4, 2007

D-ORB.fm 2007-11-08 10.50

D.17

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

### 3 Fehlerbehandlung - mögliche Implementierung

D.2 Object Request Brokers

- Annäherung an *exactly once*:
  - ◆ Um verlorene Anfragenachrichten entgegenzuwirken werden die Pakete nach einem Timeout erneut gesendet.
  - ◆ Zum Vermeiden von mehrfachen Ausführungen bei doppelt gesendeten Paketen werden IDs verwendet, welche bei einem erneuten Versenden der gleichen Anfrage nicht verändert werden.
  - ◆ Der Server soll abgesendete Antworten puffern, für den Fall, dass eine Antwort verloren geht und um eine erneute Ausführung der Methode zu vermeiden.
  - ◆ Den Absturz eines Servers zu verkraften ist schwieriger ..... daher ignorieren wir das Problem.

Übungen zu Middleware  
© Universität Erlangen-Nürnberg • Informatik 4, 2007

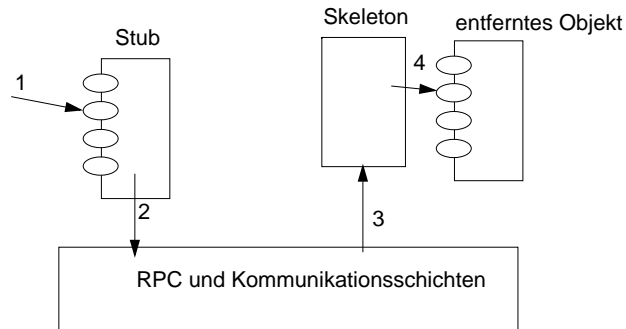
D-ORB.fm 2007-11-08 10.50

D.18

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

## 4 Stubs und Skeletons

- Stub: Stellvertreter (Proxy) des entfernten Objekts.
- Skeleton: Ruft die Methoden am entfernten Objekt auf



## 4 Stub

- implementiert den gleichen Typen wie das entfernte Objekt (gleiches Interface)
- verpackt einen Methodenaufruf in ein Request-Objekt:
  - ◆ Objekt ID, Methoden ID, Parameter
- verwendet die RPC-Schicht um eine Anforderung zu versenden
- transformiert das Rückgabeobjekt in den entsprechenden Typ

## 4 Stub Beispiel

```
package test;
import orb.*;
public class AccountImpl_Stub implements test.Account {
    int oid;
    ClientContext ctx;
    public AccountImpl_Stub(int oid, ClientContext ctx) {
        this.oid = oid;
        this.ctx = ctx;
    }

    // erzeugte Methoden (Bsp ohne Exceptionbehandlung)
    public int deposit(int param0) {
        Object[] parameters = new Object[1];
        parameters[0] = new Integer(param0);
        Request req = new Request(ctx, oid, 9, parameters);
        Object ret = req.send();
        return ((Integer)ret).intValue();
    }
    // weitere erzeugte Methoden ...
}
```

## 4 Skeleton

- ruft Methoden am "echten" Objekt auf
- notwendige Informationen:
  - ◆ Objektreferenz
  - ◆ Methoden ID
  - ◆ Parameter

## 4 Skeleton Beispiel

D.2 Object Request Brokers

```
package test;
import orb.*;
public class AccountImpl_Skel implements Skeleton {
    AccountImpl object;
    public AccountImpl_Skel() { /* ... */ }
    public void init(int oid, Object object) {
        this.oid = oid;
        this.object = (AccountImpl) object;
    }
    public Object invoke(int mid, Object[] parameters)
        throws Exception {
        switch(mid) {
            ...
            case 9: {
                int p1 = ((Integer)parameters[0]).intValue();
                int result = object.deposit(p1);
                return new Integer(result);
            }
        }
    }
}
```

Übungen zu Middleware

© Universität Erlangen-Nürnberg • Informatik 4, 2007

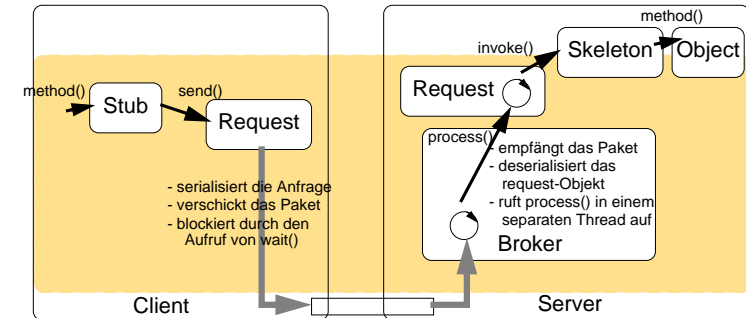
D-ORB.fm 2007-11-08 10.50

D.23

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

## 4 Request

D.2 Object Request Brokers



Übungen zu Middleware

© Universität Erlangen-Nürnberg • Informatik 4, 2007

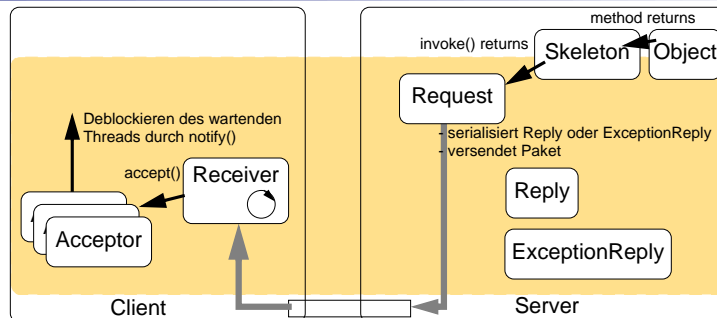
D-ORB.fm 2007-11-08 10.50

D.24

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

## 4 Reply

D.2 Object Request Brokers



Übungen zu Middleware

© Universität Erlangen-Nürnberg • Informatik 4, 2007

D-ORB.fm 2007-11-08 10.50

D.25

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

## 5 Object Store

D.2 Object Request Brokers

### Interface:

- ◆ **int registerObject(Object obj)**  
gibt eine eindeutige Objekt-ID zurück.
- ◆ **Object lookupByID(int oid)**  
liefert das Objekt zur gegebenen Objekt-ID

### Implementationstechnik:

- ◆ **java.util.Hashtable** kann verwendet werden um die Abbildung zwischen Objekt-IDs und Objekten vorzunehmen (Hinweis: die Hashtabelle kann nur Objekte speichern, die **OID** sollte daher in ein **Integer**-Objekt gekapselt werden.)

Übungen zu Middleware

© Universität Erlangen-Nürnberg • Informatik 4, 2007

D-ORB.fm 2007-11-08 10.50

D.26

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

## 6 Nameserver

D.2 Object Request Brokers

- ist aufgeteilt in einen ID-Finder und einen Proxy-Erzeuger
- ID Finder
  - ◆ bildet Namen auf OIDs ab:
    - `void bind(String name, int oid)`
    - `int lookupID(String name)`
  - ◆ verwendet `Hashtable`
  - ◆ ist selbst als entferntes Objekt mit der OID 1 implementiert
- ein realistischer Nameserver würde Referenzen auf Stub-Objekte zurückliefern anstatt OIDs
- das erfordert, dass zusammen mit der OID der Klassenname des Stubs zurückgegeben wird.

## 7 Nameserver

D.2 Object Request Brokers

- Stub-Erzeuger
  - ◆ sucht die ID (und den Klassennamen); erzeugt daraufhin ein Stub-Objekt
    - `Object lookup(String name)`
  - ◆ der Nameserver lädt die Klasse und erzeugt eine Instanz der Klasse