E Überblick über die 10. Übung

- Fragen zur Aufgabe 7(je nach Bedarf)
- Mehr zu CORBA-Services
- Java Security API

E.1 Aufgabe 7- FAQ

- "Communication Failure":
  - Immer die aktuelle IOR verwendet! (nicht kopieren)
  - Oder immer den Namensdienst verwenden unter `corbaloc::faui40u:4711/NameService`
- xxxPOA.java fehlt
  - vom JDK 1.4 IDL-Compiler wird standardmässig nur Client-Mapping erzeugt
  - Option `-fall` bzw. `-fserver` verwenden!
- seltsame Fehlermeldungen vom ORB?!
  - Auf jeden Fall JDK 1.4 verwenden (/local/java-1.4)!

E.2 CORBA-Services

- Von der OMG spezifizierte Standard-Dienste
- Nicht alle Dienste sind in jedem ORB implementiert

`Collection Service`	`Persistent Object Service`
`Concurrency Service`	`Property Service`
`Enhanced View of Time`	`Query Service`
`Event Service`	`Relationship Service`
`Externalization Service`	`Security Service`
`Naming Service`	`Time Service`
`Licensing Service`	`Trading Object Service`
`Life Cycle Service`	`Transaction Service`
`Notification Service`	

E.3 Naming Service (Namensdienst)

- ***Der Standard-Dienst,*** der von jedem ORB bereitgestellt wird
- Abbildung: Hierarchischer Namen => CORBA-Objekte
- siehe vorhergegangene Übung

E.4 Life Cycle Service

- Zweck:

  Dienste und Konventionen zum Erzeugen, Löschen, Kopieren und Verschieben von Objekten

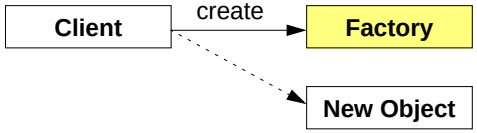
  - Standardprobleme in verteilten Systemem
  - Lösung mit einheitlichen Strategien und Diensten

E.4 Life Cycle Service (2): Standardprobleme

- Objekterzeugung:
  - Wo wird neues Objekt erzeugt, wer bestimmt das?
  - Wer erzeugt das neue Objekt? Wie findet Client diese Instanz, wie interagiert er damit?
  - Einfluss von Client auf neues Objekt (z.B. Initialisierung)?
- Objektkopie/-migration:
  - Wer bestimmt neuen Ort (Client, administrative Regeln)?
  - Wer verschiebt/kopiert das Objekt? Wie findert Client diese Instanz, wie interagiert er damit?
  - Wie behandelt man Objekt-Bäume? (deep vs shallow copy, Zyklenerhaltung, ...)

E.4 Life Cycle Service (3)

- Objekt erzeugen (aus Sicht das Client)



Aufruf einer create-Method am Factory-Objekt

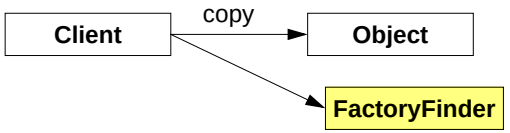
"hier" "dort"

- Factories sind keine speziellen Objekte
- Protokoll und Schnittstelle der Factory sind nicht definiert und hängen von der Art des zu erzeugenden Objekts ab
- Beispiel:

```
interface DocFactory {
    Document create();
    Document create_with_title(in string title);
    Document create_for(in natural_language nl);
};
```


E.4 Life Cycle Service (4)

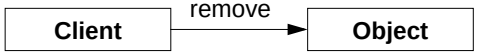
- Objekt kopieren und migrieren:



FactoryFinder bestimmt Ort

- Als Ort kann ein bestimmter Rechner, eine Gruppe von Rechnern, etc. angegeben werden

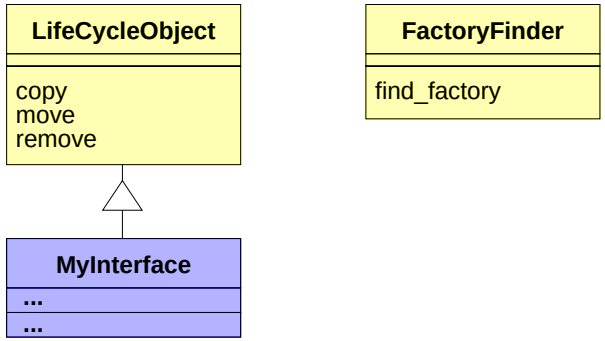
◆ Löschen:



- Mit dem Aufruf von **`remove`** wird das Objekt gelöscht

E.4 Life Cycle Service (5)

- Objekte müssen das LifeCycleObject-Interface implementieren



  - **copy** und **move** benötigen ein **FactoryFinder**-Objekt, um ein **Factory**-Objekt zu finden, das die Erzeugung der Kopie bzw. die Migration übernimmt

E.4 Life Cycle Service (6) -IDL

```
mode CosLifeCycle {
   typedef CosNaming::Name Key;
   typedef Object Factory;
   typedef sequence<Factory> Factories;
   typedef struct NVP {
      CosNaming::Istring name;
      any value;
   } NameValuePair;
   typedef sequence <NameValuePair> Criteria;

   exception NoFactory { Key search_key; };
   exception NotCopyable { string reason; };
   exception NotMovable { string reason; };
   exception NotRemovable { string reason; };
   exception InvalidCriteria{ Criteria invalid_criteria; };
   exception CannotMeetCriteria {Criteria unmet_criteria;};

   interface FactoryFinder {
      Factories find_factories(in Key factory_key)
         raises(NoFactory);
   };                              // Fortsetzung folgt ...
```


E.4 Life Cycle Service (7)

```
   interface LifeCycleObject {
      LifeCycleObject copy(in FactoryFinder there,
                           in Criteria the_criteria)
            raises(NoFactory, NotCopyable,
                   InvalidCriteria, CannotMeetCriteria);
      void move(in FactoryFinder there,
                in Criteria  the_criteria)
            raises(NoFactory, NotMovable, InvalidCriteria,
                   CannotMeetCriteria);
      void remove()
            raises(NotRemovable);
   };

   interface Generic Factory {
      Object create_object (in Key k,
                            in Criteria the_criteria)
            raises (NoFactory, InvalidCriteria,
                    CannotMeetCriteria);
   };
};
```


E.4 Life Cycle Service (8)

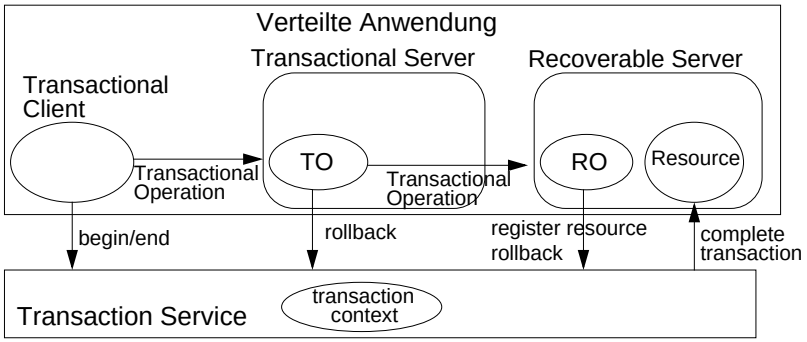
- Beispiel: Implementierung der **copy**-Methode
  - Client ruft **copy** auf und übergibt eine **FactoryFinder**-Referenz
  - Die **copy**-Method wird vom Entwickler **oder** von der CORBA-Implementierung bereitgestellt
  - **copy** wählt über **FactoryFinder** eine geeignete **Factory**
  - Neues Objekt wird mit Daten des bestehenden Objekts initialisiert
- Nach aussen: klar definierte Schnittstelle
- Nach innen: offen und implementierungsabhängig, z.B. Protokoll zwischen **copy** und **Factory**

E.5 Object Transaction Service (OTS)

- Transaktionen: "ACID":
  - atomic (alles oder nichts)
  - consistent (Neuer Zustand erfüllt Konsistenzbedingungen)
  - isolated (Isolierung: keine Zwischenzustände sichtbar)
  - durable (Persistenz)
- Eine Transaktion => mehrere Objekte, mehreren Requests: Bindung an einen *"Transaction context"*
  - wird normalerweise *implizit* an alle Objekte weitergereicht
  - auch explizite Weitergabe durch Client möglich (Spez. in IDL)
- Typischer Ablauf:
  - Beginn der Transaktion erzeugt *Transaction context* (an den Client-Thread gebunden)
  - Ausführung von Methoden (implizit an die Transaktion gebunden)
  - Schliesslich: Client beendet die Transaktion (commit/roll back)

E.5 Object Transaction Service (OTS)

- Bestandteile einer Anwendung, die vom OTS unterstützt wird:
  - Transactional Client
  - Transactional Objects (TO)
  - Recoverable Objects (RO)
  - Transactional Servers
  - Recoverable Servers



E.5 Object Transaction Service (OTS)

- Zugriff auf Transaction Service über **`Current`**-Objekt
  - **`orb.resolve_initial_reference("TransactionCurrent");`**

```
interface Current : CORBA::Current {
   void begin() raises(SubtransactionsUnavailable);
   void commit(in boolean report_heuristics)
      raises(NoTransaction,HeuristicMixed,HeuristicHazard);
   void rollback() raises(NoTransaction);
   void rollback_only() raises(NoTransaction);

   Status get_status();
   string get_transaction_name();
   void set_timeout(in unsigned long seconds);

   Control get_control();
   Control suspend();
   void resume(in Control which) raises(InvalidControl);
};
```


E.5 Object Transaction Service (OTS)

- *Transaction Context*: **`Control`**-Objekt

```
interface TransactionFactory {
   Control create(in unsigned long time_out);
   Control recreate(in PropagationContext ctx);
};

interface Control {
   Terminator get_terminator() raises(Unavailable);
   Coordinator get_coordinator() raises(Unavailable);
};

interface Terminator {
   void commit(in boolean report_heuristics) raises(...);
   void rollback();
};
```

E.5 CORBA Services - Zusammenfassung

- Von OMG standardisierte Spezifikationen von Dienste für Probleme, die in verteilten Systemen häufig auftreten
- Spezifikationen legen fest:
  - Immer: Einheitliche Schnittstelle (IDL)
  - Meistens: generelle Vorgehensweise bei der Problemlösung
  - Manchmal: konkrete Strategien (oft auch offen gelassen bzw. implementierungsabhängig)
- Weitere Dokumentation/Informationen:
  - http://www.omg.org/technology/documents/formal/corbaservices.htm
  - http://developer.java.sun.com/developer/onlineTraining/corba/corba.html

E.6 Java Security API

- Ab JDK 1.1: **`java.security.*`**
  - Keine Ver- und Entschlüsselung wegen US-Exportbeschränkungen!
  - JCE: Java Cryptography Extension (nicht exportierbar...)
- Ab JDK 1.4:
  - Ver- und Entschlüsselungsroutinen im JDK integriert
  - Paket **`javax.crypto.*`**

E.6 Java Security API

- Was bietet die API (**`java.security`**):
  - Digitale Signaturen (SHA-1 + DSA; MD2/MD5/SHA-1 + RSA)
  - Message Digests (kryptographische Hashfunkionen: SHA-1, MD2, MD5)
  - Schlüsselverwaltung
  - Sichere Zufallszahlen
- Was bietet die API (**`javax.crypto`**):
  - Ver- und Entschlüsselung
    - symmetrische Algorithmen (DES, 3DES, Blowfish)
    - asymmetrische Algorithmen (RSA)
    - Diffie-Hellman-Schlüsselerzeugung
    - Message Authentication Codes
- Einfach erweiterbar durch Registrierung von eigenen Implementierungen anderer Algorithmen ("Cryptographic Service Provider")
  - Standard: SunJCE

E.6 Java Security API

- Abstrakte Klassen mit Factory-Methoden ("engine class")
  - Provider-Implementierung durch **`getInstance`**(algorithm[, provider])
- In **`java.security:`**
  - Message Digest: **`MessageDigest`**
  - Signaturen: **`Signature`**
  - Zufallszahlen: **`SecureRandom`**
  - Schlüsselerzeugung: **`KeyFactory, KeyPairGenerator`**
  - Zertifikaterzeugung: **`CertificateFactory`**
- In **`javax.crypto:`**
  - **`Cipher`**
  - **`CipherInputStream/CipherOutputStream`**
  - KeyGenerator, KeyAgreement
  - Mac

E.6 Java Security API - Verschlüsselung

- Beispiel:

```
import javax.crypto.*;
import java.security.AlgorithmParameters;

// Cipher-Objekt für Password-basierte Verschlüsselung
Cipher c = Cipher.getInstance("PBEWithMD5AndDES");

// Initialisierung für Verschlüsselung. myKey enthält
// einen bereits generierten Schlüssel
c.init(Cipher.ENCRYPT_MODE, myKey);

// Verschlüsseln
byte[] cipherText = c.doFinal("Ein Beispiel".getBytes());

// interne Parameter der Cipher-Implementierung
// (automatisch generiert, wenn bei init nicht angegeben)
AlgorithmParameters algParams = c.getParameters();
byte[] encodedAlgParams = algParams.getEncoded();
```


E.6 Java Security API - Entschlüsselung

- Beispiel:

```
import javax.crypto.*;
import java.security.AlgorithmParameters;

// Parameter-Objekt erzeugen und initialisieren
AlgorithmParameters algParams =
    AlgorithmParameters.getInstance("PBEWithMD5AndDES");
algParams.init(encodedAlgParams);

// Cipher-Objekt erzeugenen und initialisieren
Cipher c = Cipher.getInstance("PBEWithMD5AndDES");
c.init(Cipher.DECRYPT_MODE, myKey, algParams);

// Entschlüsseln
byte[] originalText = c.doFinal(cipherText);
```


E.6 Java Security API - Digitale Signatur

```
import java.security.*;

// Schlüsselpaar erzeugen
KeyPairGenerator keyGen =
                    KeyPairGenerator.getInstance("DSA");
keyGen.initialize(256);
KeyPair pair = keyGen.generateKeyPair();

// Signatur-Objekt erzeugen und initialisieren
Signature dsa = Signature.getInstance("SHA/DSA");
dsa.initSign(pair.getPrivate());

// Datei lesen und Signatur berechnen
FileInputStream fis = new FileInputStream(argv[0]);
byte[] b=new byte[fis.available()];
fis.read(b);
dsa.update(b);

byte[] signature = dsa.sign();
PublicKey pubkey = pair.getPublic();
```


E.6 Java Security API - Digitale Signatur

- Digitale Signatur überprüfen

```
import java.security.*;

// Signatur-Objekt erzeugen und initialisieren
Signature dsa = Signature.getInstance("SHA/DSA");
dsa.initVerify(pubkey);

// Datei lesen und Signatur überprüfen
FileInputStream fis = new FileInputStream(argv[0]);
byte[] b=new byte[fis.available()];
fis.read(b);
dsa.update(b);

boolean verifies = dsa.verify(signature);
```

E.6 Java Security API - Zusammenfassung

- API sieht "engine"-Klassen für verschiedene kryptographische Funktionen vor
- Einheitliche Vorgehensweise:
  - konkrete Instanz erzeugen (bestimmter Algorithmus und Provider)
  - Initialisieren
  - Verwenden

E.7 Ausblick: Aufgabe 9

- **`AuthenticatedNameService`** (8.7.-15.7., 15 Punkte)
- API angelehnt an Standard-CORBA-Namensdienst
- Zugriff (bind/unbind) nur mit Authentisierung