Übungsblatt 1: Grundlagen komponentenbasierter Systeme

1. Die Konzepte komponentenbasierter Systeme, wie Datenstrukturen und Operationen, sind z.T. direkt in der Java-Syntax abgebildet. Manche Konzepte werden implizit genutzt, manche gar nicht. Nennen Sie für die folgenden Konzepte die Umsetzung in Java:

Konzept	Umsetzung in Java
Komposition	Klassen (als komplexe Datentypen) mit Attributen und Referenzen zu anderen Klassen
Klassen und Objekte	Klassen und Objekte
Operation	Methoden
In-Parameter einer Operation	Methodenparameter (call by value)
Out-Parameter einer Operation	gibt es nicht in Java (könnte nur über ein Object-Array als Rückgabewert simuliert werden) in CORBA IDL: Set doSomething(in int a, inout List b, out List c, out string d)
Inout-Parameter einer Operation	Methodenparameter (call by reference)
Typ des Rückgabewerts einer Operation	Methodensignatur
Komponente (Konstrukt, bei der die Implementierung versteckt wird und die Funktionalität von einem Interface beschrieben wird)	gibt es in der Programmiersprache Java so als Konstrukt direkt nicht

2. Wird das Konzept der Kapselung in Java komplett umgesetzt? Geben Sie ein Gegenbeispiel.

   Kaspelung bedeutet, dass auf interne Attribute von außen nicht direkt zugegriffen werden kann, sondern dies nur über Getter- und Setter-Methoden möglich ist. Bei Java können interne Attribute jedoch auch als public deklariert werden. Dies kann z.B. für die Performanz in eingebetteten Systemen sinnvoll sein.

3. Was geschieht in Java, wenn die Durchführung einer Ausführungsanforderung einer Operation scheitert?

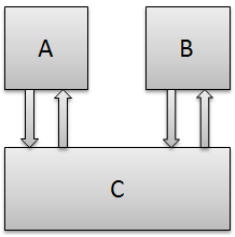
   Scheitert die Durchführung einer Ausführungsanforderung einer Operation, so wird eine Exception geworfen.

4. Welche Konzepte komponentenbasierter Systeme sind nicht in der Java-Syntax abgebildet?
   - Destructor
   - Beschreibung der Verwendung
   - Mehrfachvererbung

   Anmerkung: der entsprechende Folienteil fehlt im aktuellen Skript

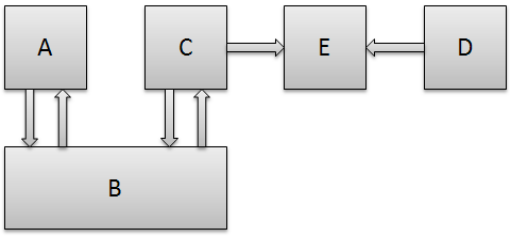
5. Modellieren Sie die folgenden Beziehungen zwischen Komponenten. Nutzen Sie dafür die Modellierungssprache, die in der Vorlesung verwendet wurde.

Beispiel: A kommuniziert mit B über C

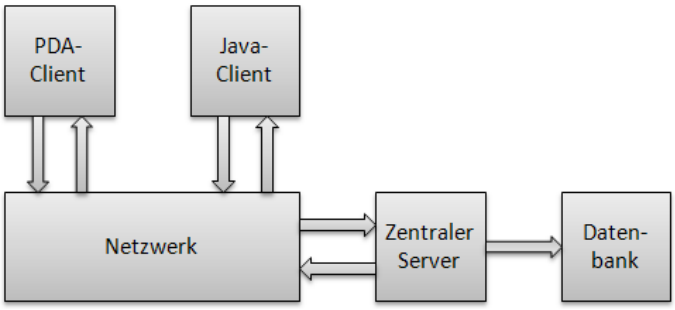


1. A benutzt B, B benutzt C und A

2. A kommuniziert über B mit C, C und D benutzen E



3. “PDA-Client” und “Java-Client” kommunizieren über “Netzwerk” mit “Zentraler Server”. “Zentraler Server” nutzt “Datenbank”.



Übungsblatt 2: Grundlagen Sichten und Modelle

1. Nennen Sie Ihnen bekannte Modelltypen für die folgenden Elemente.

Element	Modelltyp
Klassenstruktur	UML Klassendiagramm
Komponenten und Schnittstellen	UML Komponentendiagramm, UML Kompositionsstrukturdiagramme für Schnittstellen: Corba IDL, Java IDL (beziehen sich auf die Struktur von Interface), OCL (Verhalten von Interfaces, Pre-/Post-Conditions)
Verteilung	UML Verteilungsdiagramm
Zustandswechsel	UML Zustandsdiagramm, UML Aktivitätsdiagramm, Statechart, Petrinetz, UPPAAL, Markov-Kette, Touring-Maschine
Prozesse	EPK, BPMN, UML Sequenzdiagramm
Daten	ER-Model, UML Klassendiagramm
Interaktion	UML Kollaborationsdiagramm, UML Sequenzdiagramm, UML Interaktionsdiagramm, DFDs, Message Sequence Chart
Organisatorische Hierarchie	Organigramm
Zeitinformationen zu Projekten	Gantt-Chart, Pert-Diagramm, Netzplan
Anforderungen	Feature-Tree, UML Anwendungsfalldiagramm

2. Worin unterscheiden sich statische und dynamische Sichten?

   Statische Sichten sind invariant, das heißt, sie sind zu Beginn, während und nach der Laufzeit gültig. Dynamische Sichten (wie Objektdiagramm) sind nur in einem spezifischen Moment gültig.

   Sequenzdiagramme können beide Sichten darstellen. Werden ganz allgemeine Abläufe dargestellt, die immer gültig sind, so stellt dies eine statische Sicht dar. Bei ganz speziellen Ausschnitten, die nur unter bestimmten Umständen ausgeführt werden, hat man eine dynamische Sicht.

3. Ein Modell im Software Engineering ist eine Abstraktion des Programms. Wie kann diese Abstraktionsbeziehung überbrückt werden?

   Soll aus dem Modell als Ergebnis Quellcode entstehen, lassen sich zwei Alternative unterscheiden. Bei der modellbasierten Entwicklung wird das Modell als Grundlage für die Entwickler genommen, die sich an den Konzepten des Modells orientieren. Einen Schritt weiter geht die modellgetriebene Entwicklung, bei der aus dem Modell der Code generiert wird (Nachteil: werden Änderungen am Code vorgenommen, und anschließend das Modell geändert, ist die erste Änderung u.U. weg). Es gibt aber auch die Möglichkeit, dass nur das Modell spezifiziert wird (z.B. in XML), und später durch ein Werkzeug zur Laufzeit interpretiert wird (ausführbares Modell).

4. Warum werden Sichten auf bzw. Modelle von Software überhaupt benötigt? Welche Probleme ergeben sich daraus, dass mehrere Sichten auf Software bestehen?

   Durch Sichten und Modelle kann die Komplexität von Software beherrschbar gemacht werden, da jeweils nur relevante Bereiche betrachtet werden und andere ausgeblendet werden können. Sichten und Modelle dienen der Strukturierung der verschiedenen Aspekte. Die Summe aller Sichten müssen ein konsistentes, aktuelles und vollständiges Gesamtsystem ergeben. Da nicht alles integriert ist, besteht die Gefahr, dass etwas übersehen wird. Außerdem muss häufig zwischen den Sichten hin- und hergewechselt werden. Es wird somit neue Komplexität erzeugt.

Übungsblatt 3.2: Nebenläufigkeit und Fehlerbehandlung

2. Verbindungen ohne Nebenläufigkeit

   a) Warum funktioniert das Programm nicht richtig?

   Geworfene Exceptions werden nicht aufgefangen, wenn die IP-Adresse nicht verfügbar ist.

   b) Was ist der Grund für eventuelle Fehlermeldungen?

   IP-Adressen sind nicht verfügbar.

   c) Was könnte der Grund dafür sein, dass es so lange dauert, bis Ausgaben erscheinen? Gibt es ein grundsätzliches Programm mit der Art, wie das Programm geschrieben ist?

   Das Programm wartet auf den Timeout. Besser wäre es, nebenläufige Prozesse zu haben.

3. Verbindungen mit Nebenläufigkeit

   a) Welche Methoden erscheinen?

   Für jede IP-Adresse erscheint die Anzahl der angebotenen Services. Ist eine IP-Adresse nicht erreichbar, wird eine Exception angezeigt.

   b) Wieso geht (hoffentlich) alles schneller?

   Aufgrund der Tatsache, dass die Abfragen nun in nebenläufigen Threads ablaufen.

   c) Ziehen Sie zusätzlich die Javadoc-Dokumentation zu den RMI-Methoden Registry.list und Registry.lookup zu Rate: Welche Fehlermeldungen können sonst noch auftreten? Welche Ausgaben könnte man Benutzern eines Programms im Ernstfall bei Fehlern anzeigen?

   Auszug aus der Javadoc zu Registry.list

   ```
   "RemoteException - if remote communication with the registry failed; if exception is a ServerException containing an AccessException, then the registry denies the caller access to perform this operation

   AccessException - if this registry is local and it denies the caller access to perform this operation"
   ```

   Auszug aus der Javadoc zu Registry.looup

   ```
   "NotBoundException - if name is not currently bound

   RemoteException - if remote communication with the registry failed; if exception is a ServerException containing an AccessException, then the registry denies the caller access to perform this operation

   AccessException - if this registry is local and it denies the caller access to perform this operation

   NullPointerException - if name is null"
   ```

Übungsblatt 4: DoxNews

Programmcode Client.java

```
package doxnews;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.InetAddress;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.Collection;

public class Client implements ClientAuthenticator
{
	private MessageServer server;
	private String userName;
	private String identityId;
	private Registry localRegistry;

	public static void main(String[] args) throws Exception
	{
		new Client();
	}

	public Client() throws Exception
	{
		connect();

		BufferedReader reader = new BufferedReader(new
		 InputStreamReader(System.in));
		while(true) {
			System.out.print("\nEingabe> ");
			String input = reader.readLine();

			if (input == null || input.equals("exit")) {
				exit();
				return;
			}
			else if (input.length() > 5 && input.indexOf("send ") == 0)
				send(input.substring(5));
			else if (input.length() > 9 && input.indexOf("register ") == 0)
				register(input.substring(9));
			else if (input.equals("read"))
				read();
			else
				System.out.println("Ungueltige Eingabe! Kommandos: send,
				  read, register, exit");
		}
	}

	public boolean confirmUserName(String userName) throws RemoteException
	{
		if (this.userName.equalsIgnoreCase(userName))
			return true;
		else
			return false;
	}

	public boolean confirmIdentity(String identityId) throws RemoteException
	{
		if (this.identityId.equalsIgnoreCase(identityId))
			return true;
		else
			return false;
	}
```

```
	public void connect() throws Exception
	{
		ClientAuthenticator stub = (ClientAuthenticator)UnicastRemoteObject.
		  exportObject(this, 0);
		localRegistry = LocateRegistry.createRegistry(1099);
		localRegistry.bind("DoxNews/ClientAuthenticator", stub);
		System.out.println("Client Registry gefunden");
		String[] services2 = localRegistry.list();
		for (int i=0; i<services2.length; i++)
			System.out.println("Service gefunden: " + services2[i]);

		System.out.println("\nServer Registry gefunden");
		Registry registry = LocateRegistry.getRegistry("192.168.69.23");
		server = (MessageServer)registry.lookup("DoxNews/MessageServer");
		String[] services = registry.list();
		for (int i=0; i<services.length; i++)
			System.out.println("Service gefunden: " + services[i]);
	}

	public void register(String userName) throws Exception
	{
		this.userName = userName;
		identityId = server.register(userName, InetAddress.getLocalHost().
		  getHostAddress());
		System.out.println(identityId);
	}

	public void send(String message) throws Exception
	{
		server.sendMessage(identityId, message);
	}

	public void read() throws Exception
	{
		Collection<Message> nachrichten = server.readMessages(identityId);
		for(Message text: nachrichten)
			System.out.println(text.toString());
	}

	public void exit() throws Exception
	{
		server.unregister(identityId);
	}
}
```

Übungsblatt 5: Grundlagen verteilter Systeme und Middleware

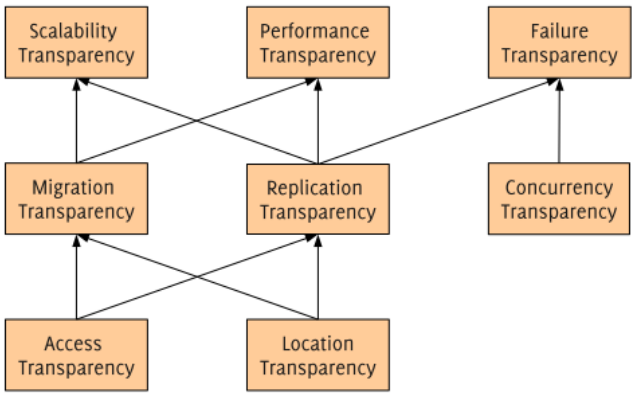
1. Nennen und erklären Sie die Heterogenitäten, die bei verteilten Softwaresystemen betrachtet werden sollten.

   - Programmiersprache
   - Betriebssysteme
   - Komponente
   - Netzwerk
   - Hardware

2. Nennen und erläutern Sie kurz die 6 generellen Anforderungen an verteilte Softwaresysteme.

   - Teilung der Betriebsmittel
     - Möglichkeit jede HW/SW/Daten in dem verteilten System zu benutzen
     - Resource-Manager kontrollieren Zugriffsrechte, bieten Namensschemata, kontrollieren nebenläufigen Zugriff
     - gemeinsames Beziehungsmodell regelt, welche Ressourcen angeboten werden, wie sie benutzt werden können und wie Anbieter und Nutzer der Ressource interagieren müssen
   - Offenheit
     - bezieht sich auf Erweiterbarkeit und Verbesserung eines verteilten Systems
     - Schnittstellen müssen publiziert werden
     - Integration alter und neuer Komponenten
     - Unterschiede in Darstellung und Präsentation müssen ausgeglichen werden
   - Nebenläufigkeit
     - Komponenten in verteilten Systemen werden in nebenläufigen Prozessen ausgeführt
     - Komponenten greifen u.U. verändernd auf gemeinsame Ressourcen zu (Variablen, DBs, Geräte)
     - Verlust der Integrität (Lost Update, Inkonsistenz)
   - Skalierbarkeit
     - Anpassung eines verteilten Systems an Vergrößerung der Benutzerschaft und Verbesserung der Antwortzeit
     - Erweiterung der Prozessorkapazität (mehr und schnellere Prozessoren)
     - Komponenten sollten nicht geändert werden müssen, um Skalierung zu erzielen
     - muss im Design der Komponenten berücksichtigt werden
   - Fehlertoleranz
     - alle Komponenten versagen potenziell ihren Dienst (HW/SW/Netzwerk)
     - verteilte Systeme haben die Möglichkeit die Zuverlässigkeit auf allen Ebenen (HW/SW/Netzwerk) zu erhöhen
     - Verbesserung der Zuverlässigkeit (Recovery, Redundanz)
   - Transparenz
     - der Verteilungsaspekt sollte gegenüber Benutzern und Entwicklern verborgen bleiben
     - Transparenz hat verschiedene Dimensionen
     - orientieren sich an den verschiedenen Eigenschaften verteilter Systeme

3. Nennen und erläutern Sie kurz die Transparenzen in verteilten Softwaresystemen und zeigen Sie ihre Beziehungen untereinander auf.



- Access Transparency: auf ein entferntes Objekt kann genauso zugegriffen werden wie auf ein lokales (ist in RMI gegeben)
- Location Transparency: der Entwickler weiß, dass es sich um ein entferntes Objekt handelt, aber es ist egal, wo es sich befindet (ist in RMI – bis auf die Registry – gegeben)
- Replication Transparency: dem Entwickler und dem Benutzer ist es transparent, dass die Middleware ein Objekt lokal kopiert und mit dem ursprünglichen Objekt synchronisiert
  Zugriffstransparenz nötig: es muss dann egal sein, ob man auf das entfernte oder das lokal kopierte Objekt zugreift
  Ortstransparenz nötig: der Ort des replizierten Objekts ist für den Benutzer transparent
- Migration Transparency: ähnlich wie Replication Transparency, nur dass ein Objekt verschoben wird
- Concurrency Transparency: die Middleware kümmert sich um das Ausführen von nebenläufigen Prozessen, im lokalen Java-Programm muss der Entwickler selbst Threads starten
- Scalability Transparency: die Middleware kümmert sich selbständig um die Verwaltung von Rechnerknoten
- Performance Transparency: automatische Lastverteilung
- Failure Transparency: Fehler können vor dem Benutzer durch erstellte Replikationen versteckt werden (ähnlich RAID 1 bei Festplatten), durch Nebenläufigkeit können Komponenten selbst auf ihre Fehler reagieren

4. Was is Marshalling / Unmarshalling?

Marshalling bezeichnet das Umwandeln von strukturellen Daten in Binärdaten, die sich für die Netzwerkübertragung serialisieren lassen. Mit Hilfe von Unmarshalling werden diese Binärdaten wieder in strukturierte Daten zurückgewandelt.

5. Was sind Stubs, und wofür werden Sie benötigt?

Stubs übernehmen die Kommunikation (und somit auch das Marshalling und Unmarshalling) zwischen Komponenten über das Netzwerk. Sie bedienen die Interfaces der jeweils entfernten Komponente und stellen Typsicherheit sicher.

6. Wie generiert man in RMI serverseitig einen Stub?

UnicastRemoteObject.exportObject(Object a, int b);
Obejct a muss dabei extends Remote beinhalten, b ist der Port

7. Wie kommt man in RMI clientseitig an einen Stub?

   registry.lookup(String name);

8. Wofür wird eine Middleware benötigt und was sind ihre Hauptaufgaben? Wo befindet sich die Middleware?

   Ermöglicht und organisiert die Kommunikation zwischen verteilten Komponenten. Eine Hauptaufgabe besteht darin, die Heterogenitäten aufzulösen. Außerdem sollte sie die Transparenzen in unterschiedlichen Graden erfüllen. Sie befindet sich auf jedem Netzwerkknoten und liegt zwischen Betriebssystem und Anwendung.

Übungsblatt 6: DoxShare

1. Entwerfen Sie die Interfaces, die vom Server und vom Client des DoxShare-Systems als RMI-Service im Netzwerk zur Verfügung gestellt werden.

```
interface ServerInterface extends Remote
{
   String register(String userName, String ipAddress) throws RemoteException;
   void share(String userID, String filename) throws RemoteException;
   Collection<Result> search(String userID, String searchTerm) throws
    RemoteException;
}

interface ClientInterface extends Remote
{
   byte[] getFile(String filename) throws RemoteException;
}
```

2. Entwerfen Sie die Programmierschnittstelle der Surchergebnis-Klasse „Result“.

```
interface Result extends Serializable {
   String getFilename();
   String getUserName();
   String getIpAddress();
}
```

(da alle Attribute Strings sind, funktioniert die Serialisierung; Sockets z.B. gehen nicht)

3. Beschreiben Sie kurz die Inhalte der einzelnen Methoden, insb. (a) die RMI-relevante Funktionalität sowie (b) die Daten, die der Server über die Benutzer und die freigegebenen Dateien speichern muss.

1) eigene Registry erstellen: LocateRegistry.createRegistry(port)
2) wir holen uns die Registry vom Server: r = LocateRegistry.getRegistry(ip)
3) Stub generieren mit Typecast: server = (ServerInterface)r.lookup(„servicename“)
4) Benutzer registrieren: id = server.register(..,..,..)
5) Datei veröffentlichen: server.share(…,id)
6) Datei suchen: s.search(…,id)
7) zur Client-Registry verbinden: cr = LocateRegistry.getRegistry(client-ip)
8) Stub generieren: client = (ClientInterface)cr.lookup(“…”)
9) Datei downloaden: client.getFile(…)

Übungsblatt 7: Web-Anwendungen mit Servlets

- 3 Dateien erstellt in der Übung
  - TestServlet Java-Datei: enthält die Logik, beinhaltet request- und response-Objekte sowie doGet- und doPost-Methoden
  - web.xml: beschreibt welche Klasse bei welcher URL aufgerufen werden soll
  - build.xml: wird mit Ant aufgerufen, Build-Skript
- beim Deployment sucht der Server die web.xml und wertet sie aus; sobald ein Browser den Application Server aufruft, wird versucht, die URL auf ein Servlet zu mappen
- mit Hilfe der Methode getParameterMap des request-Objektes können die übergebenen Parameter ausgelesen werden
- Java Server Pages/Faces werden auch über Servlets realisiert

Übungsblatt 8: EJB Session Beans

- in der WAR-Klasse steckt die Servlet-Klasse, welche vom Browser angesprochen wird und auf die Java-Klassen im Hintergrund zugreifen kann
- in der Session-Variable wird das TestBean gespeichert, welches das Interface Test implementiert
- hätte man auch einfach „new TestBean“ schreiben könnne? in unserem Fall schon, aber nur weil das TestBean auch in der WAR-Datei enthalten ist
- InitialContext gleicht der RMI-Registry
- im WAR ist die TestBean nicht bekannt (nur im JAR), durch den Registry-Lookup kann auch aus der WAR-Datei auf TestBean zugegriffen werden
- Stateful = zustandsbehaftet, stateless = kennt keinen Zustand
- vor der Änderung hat sich der Server zwar den Status des Bean gemerkt, aber wir erzeugen bei jedem Get-Befehl ein neues Bean, somit bleibt die Zahl der Klicks auf 1
- in der Session wird das Bean mit dem Zustand gespeichert
- bei stateless bekommt man vom Server irgendein Objekt aus dem Heap zurück
- JAR-Dateien stellen das „Backend“ dar
- MessageBeans stellen asynchrone Kommunikation dar, in unserem Beispiel synchrone Kommunikation (Servlet wartet auf das Ende der Ausführung)

Übungsblatt 9: Service Provision

Programmcode Client.java

```
package serviceprovision;

import java.io.IOException;
import java.net.InetAddress;
import java.rmi.AlreadyBoundException;
import java.rmi.NotBoundException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.List;

import math.IPlus;
import math.plusImpl;

import de.uni_due.s3.dox.doxservice.server.DoxServerUtils;
import de.uni_due.s3.dox.doxservice.server.FileWrapper;
import de.uni_due.s3.dox.doxservice.server.IDoxServer;
import echoservice.IEchoService;
import CurrencyChange.ICurrencyChangeService;
import doxshare.*;

public class Client {

	public static void main(String[] args) throws NotBoundException, IOException,
	  AlreadyBoundException {
		String url = DoxServerUtils.provideClassesWithWebServer(45321,
		  "./bin");

		//Test von EchoService
		Registry reg1 = LocateRegistry.getRegistry("192.168.69.23",32546);
		IEchoService stub1 = (IEchoService) reg1.lookup("EchoService");
		System.out.println(stub1.echo("Teststring"));

		//Test von CurrenyCchange
		Registry reg2 = LocateRegistry.getRegistry("192.168.69.12",2099);
		ICurrencyChangeService stub2 = (ICurrencyChangeService)
		  reg2.lookup("CurrencyChangeService");
		System.out.println(stub2.EuroToDollar(1500.00));

		//eigene Registry erstellen
		int registryPort = 32547;
		String serviceName = "Add";
		Registry rlocal = LocateRegistry.createRegistry(registryPort);
		plusImpl p = new plusImpl();
		IPlus pstub = (IPlus) UnicastRemoteObject.exportObject(p,0);
		rlocal.bind(serviceName, pstub);

		//Service add anmelden
		Registry rserver = LocateRegistry.getRegistry("192.168.69.23");
		List<FileWrapper> files = DoxServerUtils.createFileWrapper("Z:\\
		  eclipse33\\workspace\\ServiceProvision\\src\\math\\IPlus.java");
		IDoxServer server = (IDoxServer) rserver.lookup("doxserver");
		server.publish(serviceName, "Adds 2 Ints.", files,
		  InetAddress.getLocalHost(), registryPort, url);
	}
}
```

Programmcode IEchoService.java

```
package echoservice;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface IEchoService extends Remote {
	public String echo(String msg) throws RemoteException;
}
```

Programmcode EchoServiceImpl.java

```
package echoservice;

import java.rmi.RemoteException;

public class EchoServiceImpl implements IEchoService {
	@Override
	public String echo(String msg) throws RemoteException {
		return "Echo: " + msg;
	}
}
```

Programmcode ICurrencyChangeService.java

```
package CurrencyChange;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ICurrencyChangeService extends Remote{
	public double EuroToDollar(double value) throws RemoteException;
}
```

Programmcode IPlus.java

```
package math;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface IPlus extends Remote{
	public int add(int a, int b) throws RemoteException;
}
```

Programmcode plusImpl.java

```
package math;

import java.rmi.RemoteException;

public class plusImpl implements IPlus {
	public int add(int a, int b) throws RemoteException {
		return a + b;
	}
}
```