Vorbemerkung

Die Aufgabenstellung stammt von Hartmut Schulz und wurde von mir in bzgl. der Abgabemodalitäten für die Gruppen in dieser Woche modifiziert.

Ziel

Es soll eine einfache OO-Middleware entworfen und in Java realisiert werden mit deren Hilfe Methodenaufrufe auf entfernten Objekten möglich sind. Es sollen auch nebenläufige Aufrufe – auch desselben Objektes – unterstützt werden.

Applikationen, die die Middleware nutzen existieren bereits. Deren Schnittstellen werden hier nur formal definiert.

Middleware

Aufteilung der Middleware in Packages:

- Package **name_service** mit dem globalen Namensdienst, der auf einem separaten Rechner läuft und Namen auf entfernte Objektreferenzen auflöst. Der Port muss einstellbar sein.
- Package **mware_lib** mit den allgemein für den Betrien der Middleware benötigten Klassen unabhängig von der Applikation.
- Package **bank_access**
- Package **cash_access**

Die letzen beiden Packages enthalten die Schnittstellen für Applikationsobjekte.

Abhängigkeiten: Extern nur von JRE, innerhalb dürfen **bank_access** und **cash_access** von **mware_lib** abhängen. Eine Applikation bindet immer das Package mware_lib ein, ggf. noch die Packages **bank_access** oder **cash_access**.

In den genannten vier Packages muss die gesamte Middleware enthalten sein.

Schnittstellen:

Package **mware_lib**: (Dienste der Middleware)

```
/**
 * core of the middleware:
 * Maintains a Reference to the NameService
 * Singleton
 */
public class ObjectBroker {
	/**
	 * @return an Implementation for a local NameService
	 */
	public NameService getNameService() {...}

	/**
	 * shuts down the process, the OjectBroker is running in
	 * terminates process
	 */
	public void shutdown() { ... }

	/**
	 * Initializes the ObjectBroker / creates the local NameService
	 * @param serviceName
	 *            hostname or IP of Nameservice
	 * @param port
	 *            port NameService is listening at
	 * @return an ObjectBroker Interface to Nameservice
	 */
	public static ObjectBroker init(String serviceName, int port) { ... }
}

public abstract class NameService {
	/**
	 * Registers a remote object / service for name
	 * @param servant object, processing remote methods
	 * @param name a global unique name of the object / service
	 */
	public abstract void rebind(Object servant, String name);
	/**
	 * Resolves name to a generic object reference
	 * @param name
	 * @return a generic object reference
	 */
	public abstract Object resolve(String name);
}
```

Verwendung in ClientApplikationen:

```
//..
ObjectBroker ob = ObjectBroker.init(host, port);
NameService ns = ob.getNameService();
Object gor = ns.resolve(kontoID);
// yields a specific proxy object
AccountImplBase account = AccountImplBase.narrow_cast(gor);
//...
account.transfer(300.78);
//...
```

```
ob.shutdown();
```

Verwendung in ServerApplikationen:

```
//..
ObjectBroker ob = ObjectBroker.init(host, port);
NameService ns = ob.getNameService();
ns.rebind((Object) konto, kontoID);
//...
ob.shutdown();
```

Package: **bank_access**

```
public abstract class ManagerImplBase {
	public abstract String createAccount(String owner,String branch);
	public static ManagerImplBase narrowCast(Object gor) {...}

}

public abstract class AccountImplBase {
	public abstract void transfer(double amount) throws OverdraftException;
	public abstract double getBalance();
	public static AccountImplBase narrowCast(Object o) {...}
}

public class OverdraftException extends Exception {
	public OverdraftException(String msg) {super(msg);}
}
```

Package: **cash_access**

```
public abstract class TransactionImplBase {
	public abstract void deposit(String accountId,double amount)
		throws InvalidParamException;
	public abstract void withdraw(String accountId,double amount)
		throws InvalidParamException,OverdraftException;
	public abstract double getBalance(String accountId)
		throws InvalidParamException;
	public static TransactionImplBase narrowCast(Object o) {...}
}


public class InvalidParamException extends Exception  {
	public InvalidParamException(String message) {super(message);}
}
public class OverdraftException extends Exception {
	public OverdraftException(String msg) {super(msg);}
}
```

Hinweise:

- Die Middleware kann in den Schnittstellenklassen weitere Methoden implementieren.
- Schnittstellenklassen dürfen weitere Interfaces implementieren.
- Applikationen arbeiten nur mit den Schnittstellenklassen.

Fehlerbehandlung:

- Die Fehler in den Schnittstellenklassen müssen an die aufrufenden Applikationen unverändert weitergegeben werden.
- Alle anderen Fehler sollen gewrapped und als **RuntimeException** weitergereicht werden. Das ist zwar nicht die empfohlene Richtlinie im Umgang mit Exceptions in Java, entledigt die Lösung aber von komplexer Fehlerbehandlung und sei daher ausnahmsweise erlaubt.

Anforderungen und letzte Hinweise:

- Entwerfen Sie ein geeignetes Request / Response Protokoll.
- Versuchen Sie soweit möglich die Objekt Serialisierung in Java zu nutzen.
- Sockets sollen nur in möglichst wenig Klassen benutzt werden.
- Alle beteiligten Komponenten müssen auf separaten Rechnern im Labor laufen.

Abgabemodalitäten:

Für die Gruppen, deren Praktikumstermin in Woche 48 und 49 liegt, gelten die in der Vorlesung und Script bekanntgegebenen Richtlinien und Termine.

Gruppen der Woche 47 müssen bis zum Praktikumstermin einen vorläufigen Entwurf vorlegen / erläutern. Abgabe des Entwurfs ist in der Woche 48 (Montags). Abgabe des Codes in der Woche 49 (Samstag).

Final abzugeben sind:

Aktuelles Entwurfsdokument
README die Angaben zum Starten des globalen NameService enthalten
Quellcode und Binärcode der vier Packages
Logs der Testläufe