## 1. Vorbemerkungen

In dieser Aufgabe soll eine einfache objektorientierte Middleware konzipiert und in Java realisiert werden, mit deren Hilfe Methodenaufrufe eines entfernten Objektes möglich sind. Die Middleware soll dabei auch nebenläufige Aufrufe von Methoden - auch desselben Objektes - unterstützen.
Die Semantik der Applikationen wird hier bewusst ausgeklammert, da sie für die Funktion der Middleware nicht relevant ist.

## 2. Middleware und Schnittstellen

Die Middleware soll auf der obersten Ebenen wie folgt aufgeteilt werden:

- Namensdienst (1 Package/Verzeichnis, alternativ 1 JAR mit mehreren Packages/Verzeichnissen), der Namen auf Objektreferenzen abbildet. Der Namensdienst muss auf einem gesonderten Rechner laufen. Der Port muss zur Laufzeit einstellbar sein. (Parameter!)
- Package **mware_lib** mit den allgemein für den Betrieb der Middleware benötigten Klassen (Bibliothek), unabhängig von der Applikation und ihren Schnittstellen.
- Packages der Form **accessor_*** mit den jeweiligen applikationsspezifischen Schnittstellen der entfernt aufrufbaren Methoden der Applikationen. Diese Packages werden von Applikationen nur nach Bedarf eingebunden. (Sowohl client- als auch serverseitig.)

### Zulässige Abhängigkeiten und maximaler Codeumfang:

Grundsätzlich darf nur eine Abhängigkeit von JRE bestehen. Innerhalb der Middleware dürfen zusätzlich nur die *accessor_** -Packages von *mware_lib* abhängen. Den Code des Namensdienst-Pakets bindet eine Applikation generell nicht ein.

In den oben genannten Packages muss der gesamte Middleware-Code enthalten sein. Die Packages *mware_lib* und *accessor_** dürfen zusammen nicht grösser als 4 MByte sein.

### Schnittstellendefinitionen:

Die nachfolgenden Klassen müssen im Package auf der obersten Ebenen definiert sein:

Package **mware_lib***:* (Schnittstellen zu den Diensten der Middleware)

```
public class ObjectBroker {  //- Front-End der Middleware -
       public static ObjectBroker init(String serviceHost,
                                  int listenPort, boolean debug) { ... }
          // Das hier zurückgelieferte Objekt soll der zentrale Einstiegspunkt
          // der Middleware aus Applikationssicht sein.
          // Parameter: Host und Port, bei dem die Dienste (hier: Namensdienst)
          //            kontaktiert werden sollen. Mit debug sollen Test-
          //            ausgaben der Middleware ein- oder ausgeschaltet werden
          //            können.

       public NameService getNameService() {...}
          // Liefert den Namensdienst (Stellvetreterobjekt).

       public void shutDown() {...}
          // Beendet die Benutzung der Middleware in dieser Anwendung.
}

public abstract class NameService {     //- Schnittstelle zum Namensdienst -
       public abstract void rebind(Object servant, String name);
```

```
        // Meldet ein Objekt (servant) beim Namensdienst an.
        // Eine eventuell schon vorhandene Objektreferenz gleichen Namens
        // soll überschrieben werden.

    public abstract Object resolve(String name);
        // Liefert eine generische Objektreferenz zu einem Namen. (vgl. unten)
}
```

Verwendungsbeispiel in einer Serverapplikation:

```
    ...
ObjectBroker objBroker = ObjectBroker.init(host, port, false);
NameService nameSvc = objBroker.getNameService();
nameSvc.rebind((Object) myObject, "zumsel");
    ...
objBroker.shutDown();
```

Verwendungsbeispiel in einer Clientapplikation:

```
    ...
ObjectBroker objBroker = ObjectBroker.init(host, port, false);
NameService nameSvc = objBroker.getNameService();
Object rawObjRef = nameSvc.resolve("zumsel"); // = generische Referenz
ClassOneImplBase remoteObj = ClassOneImplBase.narrowCast(rawObjRef);
                        // liefert spezialisiertes Stellvertreterobjekt
    ...
try {  // Entfernter Methodenaufruf
       String s = remoteObj.methodOne("hi there!", 567);
catch ( ... )
       ...
 ...
objBroker.shutDown();
```

Package **accessor_one** (applikationsspezifische Schnittstelle):

```
public abstract class ClassOneImplBase {
       public abstract String methodOne(String param1, int param2)
                           throws SomeException112;
       public static ClassOneImplBase narrowCast(Object rawObjectRef) {...}
}

public abstract class ClassTwoImplBase {
       public abstract int methodOne(double param1) throws SomeException110;
       public abstract double methodTwo() throws SomeException112;
       public static ClassTwoImplBase narrowCast(Object rawObjectRef) {...}
}

public class SomeException110 extends Exception {
       public SomeException110(String message) { super(message);}
}
public class SomeException112 extends Exception {
       public SomeException112(String message) { super(message);}
}
```

Package **accessor_two** (applikationsspezifische Schnittstelle):

```
public abstract class ClassOneImplBase {
       public abstract double methodOne(String param1, double param2)
                           throws SomeException112;
       public abstract double methodTwo (String param1, double param2)
                           throws SomeException112, SomeException304;
       public static ClassOneImplBase narrowCast(Object rawObjectRef) {...}
}
```

```
public class SomeException112 extends Exception {
      public SomeException112(String message) { super(message);}
}
public class SomeException304 extends Exception {
      public SomeException304(String message) { super(message);}
}
```

Anmerkungen: Die Middlware kann in den **ImplBase*-Klassen weitere Methoden und Interfaces implementieren. Die Anwendungen kennen und benutzen aber nur die oben definierten Klassen und Methoden. Die Methode `narrowCast` soll ein (spezialisiertes) Stellvertreterobjekt zu einer generischen Objektreferenz liefern. Sie ist von der Middleware zu implementieren.

**Fehlerbehandlung:**

Tritt beim Aufruf einer entfernten Methode eine der o.g. Exceptions auf, ist diese von der Middleware in Typ und Meldungstext unverändert an die aufrufende Applikation zurückzuleiten - so, als wäre sie dort lokal geworfen worden.

Bei Fehlern innerhalb der Middleware soll eine geeignete Exception mit aussagefähiger Meldung geworfen werden. Bei Methoden mit Objekten als Rückgabewert kann auch `null` zurückgeliefert werden.

**Stringparameter und -rückgabewerte** können die Zeichen 'A'...'Z', 'a'...'z', '0'...'9', '.', '!', '-' und *Blank* enthalten. Auch der Wert `null` ist zulässig.

## 3. Applikationen

Zum Testen der Middleware sollen einfache Applikationen geschrieben werden, die alle Anwendungsfälle der *accessor_**-Schnittstellen überprüfen, insbesondere auch o.g. Fehlerfälle.

Die aufgerufene Methode soll nach eigenem Ermessen Returnwerte für die übergebenen Parameter liefern oder Exceptions werfen.

Die aufrufende Applikation soll zu jedem Methodenaufruf folgendes ausgeben:

- Name der Schnittstellenklasse mit Package und Name des referenzierten Objektes (z.B.: `accessor_two.ClassOneImplBase (“zumsel“)`).
- Name der aufgerufenen Methode (z.B.: `methodOne`)
- Parameterwerte (z.B.: `param1 = “yxv“, param2 = ... )`
- Returnwert (`Return value = ...`) oder Klasse und Meldungstext der Exception (z.B. `accessor_two.SomeException112 with message “You must be out of your mind“`)

Beispiel für eine Ausgabe nach einem Methodenaufruf::

```
accessor_two.ClassOneImplBase  (“zumsel“)
methodOne
param1 = “yxv“, param2 = 765
Return value = 7
```

bzw. im Fehlerfall statt Returnwert:

```
accessor_two.SomeException112 with message “You must be out of your mind“
```

Eine GUI ist in allen Fällen nicht erforderlich. Es genügt die Ausgabe auf der Kommandozeile.

## 4. Hinweise und Tipps:

- Überlegen Sie sich, welche Informationen für einen Aufruf ausgetauscht werden müssen. Entwerfen Sie ein geeignetes Request/Reply-Protokoll.
- Sockets sollen nur in möglichst wenigen Klassen verwendet werden.
- Es ist nicht notwendig, die Java-Reflection einzusetzen. Nutzen Sie die Vererbung!
- Alle beteiligten Komponenten müssen auf separaten Rechnern im Labor unter Linux laufen können.

**Vorbereitung**: Bis zum **Freitagabend 20:00 Uhr** vor dem Praktikumstermin ist ein Konzeptpapier als **PDF**-Dokument (mit ausgefülltem Dokumentationskopf) per E-Mail über den Abgabeverteiler abzugeben. Darin ist der geplante Aufbau der Middleware sowie alle wesenlichen Interaktionen und Abläufe mit geeigneten Diagrammen zu dokumentieren. Die wichtigsten Klassen und Methoden müssen hier bereits erkennbar sein.

Bis zum Praktikumstermin ist der Entwurf soweit zu entwickeln, dass die Implementierung im Praktikum abgeschlossen werden kann.

Die **Vorführung** beginnt um ca. **15:00**. Die Middleware wird dabei mit drei Applikationen getestet, zwei davon werden fremde Applikationen sein, die im Praktikum per Zufall ausgewählt werden.

**Abgabe** als ZIP-Archiv **am Ende des Praktikums** von allen Teams an den o.g. Abgabeverteiler mit CC an den Praktikumspartner.

Die Abgabe muss folgendes enthalten:

- README-Datei, die beschreibt, wie der Namensdienst von der Kommandozeile zu starten ist,
- Binär- und Quellcode der Middleware-Pakete sowie der Testapplikationen,
- *mware_lib* und *accessor_** binär in Form von Verzeichnissen,
- Ausgaben der Applikationen aus den Vorführungsläufen,
- aktualisierter Dokumentationskopf.

Abgaben, die diese Form nicht erfüllen oder unvollständig sind, werden nicht akzeptiert.

Voraussetzung für die erfolgreiche Bearbeitung ist eine fristgerecht eingereichte, ausreichende Vorbereitung. Die Befragung und die Vorführung müssen erfolgreich absolviert werden. Im Übrigen gelten die aus den vorangegangen Aufgaben bekannten Regelungen.