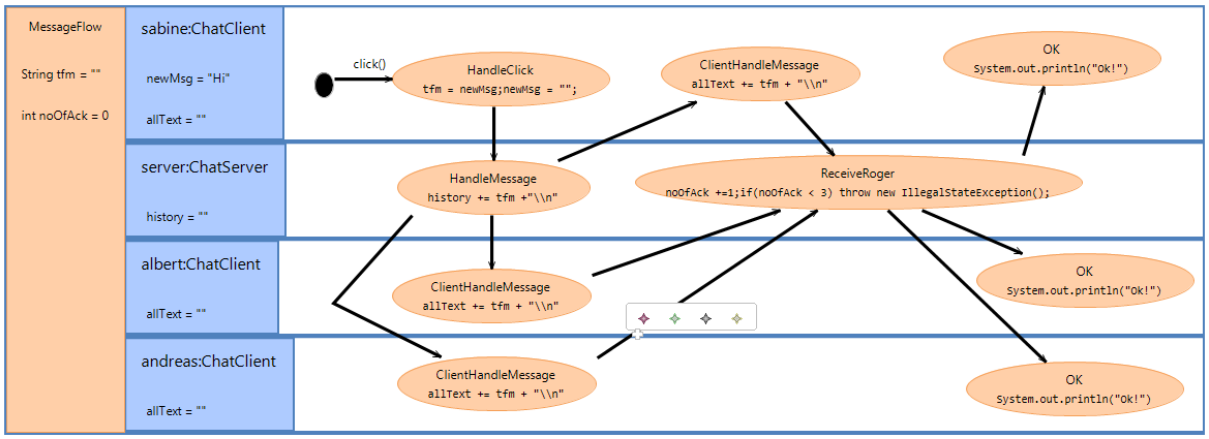
ALLGEMEIN

In dieser Übung geht es darum, einen Interpreter für Task Flow Diagramme zu implementieren. Soll ein Diagramm interpretiert werden, wird der Startknoten mit einem *ActiveToken* versehen. Dann führt der Interpreter grundsätzlich folgende Schritte aus:

- Für jeden Knoten, dem gerade ein ActiveToken zugewiesen ist:
  a. Globale Variablen einlesen
  b. (Lane-)lokale Variablen einlesen
  c. Code des Tasks ausführen
  d. Globale Variablen zurückschreiben
  e. (Lane-)lokale Variablen zurückschreiben
  f. ActiveToken entfernen
- Falls keine Fehler während der Ausführung aufgetreten sind, lege für jeden folgenden Knoten einen neuen ActiveToken an.
- So lange noch Token übrig sind, gehe zu 1.

Als Beispiel soll der in der Veranstaltung gezeigte MessageFlow verwendet werden:



Um den Code des Diagramms ausführen zu können, benötigen Sie die BeanShell[1]. Laden Sie die das All-In-One BeanShell Paket herunter und binden Sie es wie in der Vorlesung gezeigt in ihr Projekt ein.

Abgabe: Bis spätestens zum ***Sonntag den 03.02.2012*** über das Hausaufgabenabgabesystem zur Veranstaltung. Die Abgabe muss eine .zip Datei mit folgendem Inhalt sein:

- Alle 4 Eclipse Plugin Projekte (Modell, Edit, Editor, Diagramm)
- Beispiel-Modell inkl. Diagramm (siehe *Aufgabe 2 –Beispiel erzeugen* **(5P)**)
- Im Modell Plugin:
  - JUnit Test für Interpreter (siehe TODO: AUFGABE)

[1] http://beanshell.org

AUFGABE 1 – MODELL ERWEITERN (4P)

Erweitern Sie ihr Modell:

- Fügen Sie die Klasse *ActiveToken* hinzu. Sie soll beliebigen Knoten hinzugefügt werden können um den „Aktiv" Zustand eines Knotens kennzeichnen zu können.
- Überlegen Sie sich eine Möglichkeit, Kopien von Variablen im Modell ablegen zu können. Dies gilt für LaneContainer sowie für Lanes!
- Erzeugen Sie die nötigen Eugenia Annotationen, um die ActiveToken im Diagramm anzeigen zu können. Orientieren Sie sich bei der Darstellung des ActiveToken an folgender Darstellung:

Hier besitzt der Task „ReceiveRoger" gerade 3 ActiveToken.

Hinweis 1: Bedenken Sie, dass ein Knoten durchaus mehrere ActiveToken besitzen kann!

Hinweis 2: Zur Darstellung des ActiveToken eignet sich die Eugenia Annotation „@gmf.affixed", welche an eine Rolle einer ContainmentReference geschrieben werden kann.

AUFGABE 2 –BEISPIEL ERZEUGEN (5P)

Erzeugen Sie mit ihrem Task Flow Diagramm Editor das unter Allgemein gezeigte Beispiel. Achten Sie im Besonderen auf die Vorbelegung der Variablen und auf den Code der Tasks.

AUFGABE 3 – TFDINTERPRETER STUB ERSTELLEN (2P)

Erzeugen Sie eine neue Klasse TFDInterpreter mit den folgenden Methoden:

- public static void interprete(Diagram diagram)
- public static void step(Diagram diagram)
- public static void reset(Diagram diagram)

Der TFDInterpreter dient als Wrapper für den eigentlichen BeanShell Interpreter.

AUFGABE 4 – JUNIT TEST ERSTELLEN (12P)

Erzeugen Sie eine JUnit Testklasse, die die drei zu implementierenden Actions

- interprete
- step
- reset

testet. Schreiben Sie hierzu die folgenden beiden Tests:

1. testInterpretationOfMessageFlow

   hier soll das Beispiel aus Aufgabe 2 eingelesen und einmal komplett interpretiert werden.

   Hinweis: Achten Sie darauf, dass vor UND nach einem Interpretationsschritt wichtige Stellen im Modell überprüft werden. Hierzu gehört insbesondere der Zustand aller Variablen sowie die Anzahl und Position der ActiveToken.

2. testResetOfMessageFlow

   hier soll das Beispiel aus Aufgabe 2 eingelesen und bis zu dem Zustand interpretiert werden, dass die globale Variable noOfAck den Wert 3 besitzt. Danach soll der Interpreter das Diagramm zurücksetzen und anschließend überprüft werden, ob der Zustand tatsächlich dem Ursprungszustand entspricht.

Tipp: Um in einem JUnit Test ein EMF Modell einzulesen, ist zunächst eine Initialisierung der EMF Registrierungs notwendig. In ihrer Testklasse können Sie sinngemäß folgenden Code verwenden:

```
@BeforeClass
public static void init()
{
   // Initialize EMF
   @SuppressWarnings("unused")
   ModelPackage modelPackage = ModelPackage.eINSTANCE;
   @SuppressWarnings("unused")
   ModelFactory modelFactory = ModelFactory.eINSTANCE;

   Resource.Factory.Registry.INSTANCE.getExtensionToFactoryMap().put("model", new
   XMIResourceFactoryImpl());
}
```

Das eigentliche einlesen kann vor der Ausführung des jeweiligen Tests folgendermaßen erledigt werden:

```
private Diagram diagram;

@Before
public void setup()
{
   ResourceSet rSet = new ResourceSetImpl();
   Resource resource = rSet.getResource(URI.createFileURI(
                              new File("res/TFDChatTest.model").getAbsolutePath()), true);
   diagram = (Diagram)resource.getContents().get(0);
}
```

AUFGABE 5 – TFDINTERPRETER IMPLEMENTIEREN (10P)

Implementieren Sie die Methoden der Klasse TFDInterpreter so, dass die in Aufgabe 4 erstellten Tests erfolgreich durchlaufen. Orientieren Sie sich bei der Verwendung der BeanShell an den Beispielen aus der Vorlesung.

Tipp: Hier eine genauere Erklärung, was die Methoden im Einzelnen tun sollten.

- public static void interprete(Diagram diagram)

  Die Methode soll zunächst alle Variablen des Diagramms sichern, um sie bei einem möglichen *reset* wiederherstellen zu können. Danach soll der Startknoten des Diagramms mit einem *ActiveToken* versehen werden.

- public static void step(Diagram diagram)

  Führt für jeden Knoten mit ActiveToken folgende Schritte durch:
  a. Globale Variablen einlesen
  b. (Lane-)lokale Variablen einlesen
  c. Code des Tasks ausführen
  d. Globale Variablen zurückschreiben
  e. (Lane-)lokale Variablen zurückschreiben
  f. ActiveToken entfernen
  g. Falls kein Fehler in c. aufgetreten ist: Für jeden folgenden Task ein neues ActiveToken erstellen und zuweisen.

- public static void reset(Diagram diagram)

  Setzt das Diagramm in seinen Ursprungszustand zurück. Konkret heißt das, alle in der Methode *interprete(Diagram diagram)* gesicherten Variablen zurückzukopieren.

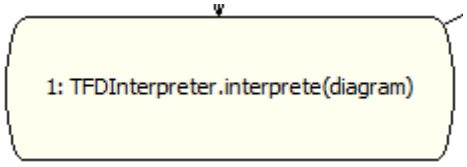
AUFGABE 6 – ACTIONS ERSTELLEN (6P)

Erzeugen Sie in ihrem Fujaba Klassendiagramm 3 Actions:

- InterpreteDiagramAction
- InterpreteDiagramStepAction
- InterpreteDiagramResetAction

Gehen Sie hierzu analog zu Hausaufgabe 3 vor. Die Actions sollen auf dem Diagramm aufrufbar sein.

Tipp: Um in einem Storydiagramm eine statische Methode aufzurufen, erzeugen Sie Collaboration Statement der Form



Tipp 2: Damit während der Codegenerierung, der Import der Klasse TFDInterpreter automatisch hinzugefügt wird, fügen Sie die Klasse als *<<reference>>* zu ihrem bisherigen Klassendiagramm hinzu. Achten Sie darauf, dass sowohl Package als auch Klassenname korrekt sind. Danach können Sie für jede der 3 erzeugten Actions in ihrem Actions Klassendiagramm den Import hinzufügen: