Факультет ИУ "Информатика и системы управления"

Кафедра ИУ-3 "Информационные системы и телекоммуникации"

Отчет к лабораторной работе №5

по курсу "**Разработка программного обеспечения**"

направления 2304000062

# " [Расширение Eclipse своими плагинами](http://www.agentlab.ru/confluence/pages/viewpage.action?pageId=55771143)"

Продолжительность 4 часа.

Выполнил:

студент группы ИУ3-61

Шалаев И.В.

Проверил:

Иванов А.М.

# Цели лабораторной работы

* Углубление навыков работы с системой контроля версий
* Ознакомление на практике с основами графовых БД и технологий Semantic Web

# Задание

1. Чтобы познакомиться с технологией выполните указания обучающего материала

* [Extending the Eclipse IDE - Plug-in development](http://www.vogella.com/tutorials/EclipsePlugIn/article.html)

2. Откройте Eclipse и обновите локальную копию репозитария

* В перспективе Git, надо сделать Pull

3. Переключитесь на стандартную целевую платформу Eclipse

* Настройки Eclipse > Plug-in Development > Target Platform, установить **Running Platform**

4. Импортируйте в Workspace проекты:

* **ru.agentlab.jfxed**
* **ru.agentlab.jfxed.figures**
* **ru.agentlab.jfxed.figures.class**

4. Запустите редактор

* Конфигурация запуска **Eclipse Jfxed**

Исходный код программы:

**package** JenaPruebas;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** com.hp.hpl.jena.ontology.Individual;

**import** com.hp.hpl.jena.ontology.ObjectProperty;

**import** com.hp.hpl.jena.ontology.OntClass;

**import** com.hp.hpl.jena.ontology.OntModel;

**import** com.hp.hpl.jena.rdf.model.ModelFactory;

**import** com.hp.hpl.jena.rdf.model.SimpleSelector;

**import** com.hp.hpl.jena.rdf.model.StmtIterator;

**public** **class** Lab3 {

**static** **final** String ***SCHEMA*** = "http://www.agentlab.ru/jfxed/onto/strategicrationale";

**static** **final** String ***NS*** = ***SCHEMA*** + "#";

**public** **static** **void** main (String[] args) {

OntModel m = ModelFactory.*createOntologyModel*();

//create superclasses

OntClass concept = m.createClass(***NS*** + "Concept");

OntClass relation = m.createClass(***NS*** + "Relation");

//create objects

//chelovecheck

OntClass businessActorClass = m.createClass(***NS*** + "businessActor");

businessActorClass.addSuperClass(concept);

//cilindr

OntClass businessRoleClass = m.createClass(***NS*** + "businessRole");

businessRoleClass.addSuperClass(concept);

//yellow oval

OntClass businessServiceClass = m.createClass(***NS*** + "businessService");

businessServiceClass.addSuperClass(concept);

//yellow rect

OntClass businessProcessClass = m.createClass(***NS*** + "businessProcess");

businessProcessClass.addSuperClass(concept);

//blue oval

OntClass applicationServiceClass = m.createClass(***NS*** + "applicationService");

applicationServiceClass.addSuperClass(concept);

//green oval

OntClass infrastructureServiceClass = m.createClass(***NS*** + "infrastructureService");

infrastructureServiceClass.addSuperClass(concept);

//blue rect

OntClass applicationComponentClass = m.createClass(***NS*** + "applicationComponent");

applicationComponentClass.addSuperClass(concept);

//bright green group

OntClass nodeClass = m.createClass(***NS*** + "node");

nodeClass.addSuperClass(concept);

//green pc

OntClass deviceClass = m.createClass(***NS*** + "device");

deviceClass.addSuperClass(concept);

//green rect with circle

OntClass systemSoftwareClass = m.createClass(***NS*** + "systemSoftware");

systemSoftwareClass.addSuperClass(concept);

//create relations

//oneWayArrow

OntClass usedByClass = m.createClass(***NS*** + "usedBy");

usedByClass.addSuperClass(relation);

//twoWayDots

OntClass assignmentClass = m.createClass(***NS*** + "assignment");

assignmentClass.addSuperClass(relation);

//oneWayWhiteArrow

OntClass realizationClass = m.createClass(***NS*** + "realization");

realizationClass.addSuperClass(relation);

//oneWayBlackArrow

OntClass triggeringClass = m.createClass(***NS*** + "triggering");

triggeringClass.addSuperClass(relation);

//like background

OntClass compositionClass = m.createClass(***NS*** + "composition");

compositionClass.addSuperClass(relation);

//create properties

//for usedBy

ObjectProperty propUsedBy = m.createObjectProperty(***NS*** + "usedBy");

propUsedBy.addDomain(businessServiceClass);

propUsedBy.addRange(businessRoleClass);

ObjectProperty propUsedBy1 = m.createObjectProperty(***NS*** + "usedBy1");

propUsedBy1.addDomain(infrastructureServiceClass);

propUsedBy1.addRange(applicationComponentClass);

ObjectProperty propUsedBy2 = m.createObjectProperty(***NS*** + "usedBy2");

propUsedBy2.addDomain(applicationServiceClass);

propUsedBy2.addRange(businessProcessClass);

ObjectProperty propUsedBy3 = m.createObjectProperty(***NS*** + "usedBy3");

propUsedBy3.addDomain(applicationServiceClass);

propUsedBy3.addRange(applicationComponentClass);

//for assignment

ObjectProperty propAssignment = m.createObjectProperty(***NS*** + "assignment");

propAssignment.addDomain(businessRoleClass);

propAssignment.addRange(businessActorClass);

ObjectProperty propAssignment1 = m.createObjectProperty(***NS*** + "assignment1");

propAssignment1.addDomain(businessProcessClass);

propAssignment1.addRange(businessRoleClass);

//for realization

ObjectProperty propRealization = m.createObjectProperty(***NS*** + "realization");

propRealization.addDomain(applicationComponentClass);

propRealization.addRange(applicationServiceClass);

ObjectProperty propRealization1 = m.createObjectProperty(***NS*** + "realization1");

propRealization1.addDomain(businessProcessClass);

propRealization1.addRange(businessServiceClass);

ObjectProperty propRealization2 = m.createObjectProperty(***NS*** + "realization2");

propRealization2.addDomain(nodeClass);

propRealization2.addRange(infrastructureServiceClass);

ObjectProperty propRealization3 = m.createObjectProperty(***NS*** + "realization3");

propRealization3.addDomain(deviceClass);

propRealization3.addRange(infrastructureServiceClass);

//for triggering

ObjectProperty propTriggering = m.createObjectProperty(***NS*** + "triggering");

propTriggering.addDomain(businessProcessClass);

propTriggering.addRange(businessProcessClass);

//for composition

ObjectProperty propComposition = m.createObjectProperty(***NS*** + "composition");

propComposition.addDomain(businessProcessClass);

propComposition.addRange(businessProcessClass);

ObjectProperty propComposition1 = m.createObjectProperty(***NS*** + "composition1");

propComposition1.addDomain(nodeClass);

propComposition1.addRange(systemSoftwareClass);

ObjectProperty propComposition2 = m.createObjectProperty(***NS*** + "composition2");

propComposition2.addDomain(nodeClass);

propComposition2.addRange(deviceClass);

//create individuals

Individual insurant = m.createIndividual(***NS*** + "insurant", businessRoleClass);

Individual client = m.createIndividual(***NS*** + "client", businessActorClass);

Individual cliamReg = m.createIndividual(***NS*** + "cliamregistration", businessServiceClass);

Individual custInf = m.createIndividual(***NS*** + "custominformation", businessServiceClass);

Individual cliamPay = m.createIndividual(***NS*** + "cliampayment", businessServiceClass);

Individual reg = m.createIndividual(***NS*** + "register", businessProcessClass);

Individual accept = m.createIndividual(***NS*** + "accept", businessProcessClass);

Individual valuate = m.createIndividual(***NS*** + "valuate", businessProcessClass);

Individual pay = m.createIndividual(***NS*** + "pay", businessProcessClass);

Individual handleClaim = m.createIndividual(***NS*** + "HandleClaim", businessProcessClass);

Individual insurer = m.createIndividual(***NS*** + "insurer", businessRoleClass);

Individual archiSur = m.createIndividual(***NS*** + "archiSurance", businessActorClass);

Individual insuranceApl = m.createIndividual(***NS*** + "insuranceApplication", applicationServiceClass);

Individual custDataMod = m.createIndividual(***NS*** + "customerDataModfication", applicationServiceClass);

Individual premiumPay = m.createIndividual(***NS*** + "premiumPayment", applicationServiceClass);

Individual cis = m.createIndividual(***NS*** + "CIS", applicationServiceClass);

Individual claimsInfServ = m.createIndividual(***NS*** + "ClaimsInfoServ", applicationServiceClass);

Individual crmSystem = m.createIndividual(***NS*** + "crmSystem", applicationComponentClass);

Individual policyDataManagment = m.createIndividual(***NS*** + "policyDataManagment", applicationComponentClass);

Individual finApl = m.createIndividual(***NS*** + "FinantialApplication", applicationComponentClass);

Individual claimFilSer = m.createIndividual(***NS*** + "CliamFilesService", infrastructureServiceClass);

Individual custFilServ = m.createIndividual(***NS*** + "CustomerFileService", infrastructureServiceClass);

Individual mainFrame = m.createIndividual(***NS*** + "Mainframe", nodeClass);

Individual nasFile = m.createIndividual(***NS*** + "NASfileserver", deviceClass);

Individual unixServ = m.createIndividual(***NS*** + "UNIXServer", deviceClass);

Individual unixServ1 = m.createIndividual(***NS*** + "UNIXServer", deviceClass);

Individual cics = m.createIndividual(***NS*** + "CICS", systemSoftwareClass);

Individual dbms = m.createIndividual(***NS*** + "DBMS", systemSoftwareClass);

Individual mesQue = m.createIndividual(***NS*** + "MessageQueuing", systemSoftwareClass);

//\*\*\*\*

//create ind links

//\*\*\*\*

//create all usedBy links

//create 1 links

Individual link11 = m.createIndividual(***NS*** + "cliamregistration", usedByClass);

link11.addProperty(propUsedBy, insurant);

Individual link12 = m.createIndividual(***NS*** + "custominformation", usedByClass);

link12.addProperty(propUsedBy, insurant);

Individual link13 = m.createIndividual(***NS*** + "cliampayment", usedByClass);

link13.addProperty(propUsedBy, insurant);

Individual link14 = m.createIndividual(***NS*** + "insuranceApplication", usedByClass);

link14.addProperty(propUsedBy2, reg);

Individual link15 = m.createIndividual(***NS*** + "customerDataModfication", usedByClass);

link15.addProperty(propUsedBy2, reg);

Individual link16 = m.createIndividual(***NS*** + "premiumPayment", usedByClass);

link16.addProperty(propUsedBy2, pay);

Individual link17 = m.createIndividual(***NS*** + "CIS", usedByClass);

link17.addProperty(propUsedBy3, policyDataManagment);

Individual link18 = m.createIndividual(***NS*** + "ClaimsInfoServ", usedByClass);

link18.addProperty(propUsedBy3, finApl);

Individual link19 = m.createIndividual(***NS*** + "CliamFilesService", usedByClass);

link19.addProperty(propUsedBy1, crmSystem);

link19.addProperty(propUsedBy1, policyDataManagment);

Individual link110 = m.createIndividual(***NS*** + "CustomerFileService", usedByClass);

link110.addProperty(propUsedBy1, policyDataManagment);

link110.addProperty(propUsedBy1, finApl);

//create all assignment links

//create 2 links

Individual link21 = m.createIndividual(***NS*** + "insurant", assignmentClass);

link21.addProperty(propAssignment, client);

Individual link22 = m.createIndividual(***NS*** + "insurer", assignmentClass);

link22.addProperty(propAssignment, archiSur);

Individual link23 = m.createIndividual(***NS*** + "HandleClaim", assignmentClass);

link23.addProperty(propAssignment1, insurer);

//create all triggering links

//create 3 links

Individual link31 = m.createIndividual(***NS*** + "register", triggeringClass);

link31.addProperty(propTriggering, accept);

Individual link32 = m.createIndividual(***NS*** + "accept", triggeringClass);

link32.addProperty(propTriggering, valuate);

Individual link33 = m.createIndividual(***NS*** + "valuate", triggeringClass);

link33.addProperty(propTriggering, pay);

//create all realization links

Individual link41 = m.createIndividual(***NS*** + "register", realizationClass);

link41.addProperty(propRealization1, cliamReg);

Individual link42 = m.createIndividual(***NS*** + "accept", realizationClass);

link42.addProperty(propRealization1, custInf);

Individual link43 = m.createIndividual(***NS*** + "pay", realizationClass);

link43.addProperty(propRealization1, cliamPay);

Individual link44 = m.createIndividual(***NS*** + "FinantialApplication", realizationClass);

link44.addProperty(propRealization1, premiumPay);

Individual link45 = m.createIndividual(***NS*** + "crmSystem", realizationClass);

link45.addProperty(propRealization, insuranceApl);

link45.addProperty(propRealization, cis);

Individual link46 = m.createIndividual(***NS*** + "policyDataManagment", realizationClass);

link46.addProperty(propRealization, custDataMod);

link46.addProperty(propRealization, claimsInfServ);

Individual link47 = m.createIndividual(***NS*** + "mainframe", realizationClass);

link47.addProperty(propRealization2, claimFilSer);

Individual link48 = m.createIndividual(***NS*** + "NASfileserver", realizationClass);

link48.addProperty(propRealization3, custFilServ);

//create all composition links

Individual link51 = m.createIndividual(***NS*** + "HandleClaim", compositionClass);

link51.addProperty(propComposition, reg);

link51.addProperty(propComposition, accept);

link51.addProperty(propComposition, valuate);

link51.addProperty(propComposition, pay);

Individual link52 = m.createIndividual(***NS*** + "Mainframe", compositionClass);

link52.addProperty(propComposition1, cics);

link52.addProperty(propComposition1, dbms);

link52.addProperty(propComposition1, mesQue);

Individual link53 = m.createIndividual(***NS*** + "UnixServerFarm", compositionClass);

link53.addProperty(propComposition2, unixServ);

link53.addProperty(propComposition2, unixServ1);

System.***out***.println("Request1\n");

**for**(StmtIterator it = link11.listProperties(); it.hasNext();)

{

System.***out***.println(it.nextStatement());

}

System.***out***.println("Request2\n");

**for**(StmtIterator it = reg.listProperties(); it.hasNext();)

{

System.***out***.println(it.nextStatement());

}

System.***out***.println("Request3\n");

//вывод всего, что связано с insurer для всей модели

OntClass c = insurer.getOntClass();

SimpleSelector s = **new** SimpleSelector(**null**, **null**, c);

**for**(StmtIterator it = m.listStatements(s); it.hasNext();)

{

System.***out***.println(it.nextStatement());

}

**try** {

m.write(**new** FileWriter("Ilya.owl"), "RDF/XML");

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**package** ru.agentlab.jfxed.diagramms.shalaev

**import** com.hp.hpl.jena.ontology.OntModel

**import** com.hp.hpl.jena.query.QueryExecutionFactory

**import** com.hp.hpl.jena.query.QueryFactory

**import** com.hp.hpl.jena.query.QuerySolution

**import** com.hp.hpl.jena.rdf.model.Resource

**import** de.fxdiagram.core.XDiagram

**import** ru.agentlab.jfxed.IDiagram

**import** ru.agentlab.jfxed.figures.clazz.Coub

**public** **class** ClazzDiagram **implements** IDiagram {

**static** String *SOURCE* = "http://www.agentlab.ru/jfxed/onto/shalaev"

**static** String *NS* = *SOURCE* + "#"

**override** createJfx(OntModel jenaModel, XDiagram jfxDiagram) {

**val** queryString ='''

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

select ?uri

where {

?uri rdf:type <«*SOURCE*»#Concept>

}

'''

**val** query = QueryFactory.*create*(queryString)

// Execute the query and obtain results

**val** qe = QueryExecutionFactory.*create*(query, jenaModel)

**val** results = qe.execSelect()

**for** ( ; results.hasNext() ; )

{

**val** QuerySolution soln = results.nextSolution()

**val** Resource x = soln.getResource("uri") // Get a result variable by name.

**val** target = **new** Coub() => [

layoutX = 280

layoutY = 280

name = x.localName

]

jfxDiagram => [

nodes += target

]

}

qe.close()

}

}

Результаты запуска программы:

