1.4 OWL API

- Basic API for accessing, querying and manipulating OWL-Ontologies
- Used by some SemWeb application (especially Protege)
- Based on the idea that an ontology is a set of axioms, therefore everything manipulated with this API has to be also part of an axiom
- Provides interface for reasoners, used by
  - FaCT++
  - HermiT (This course)
  - Pellet (Semantic Web Lecture)
  - JFact
  - Chainsaw
  - TrOWL
  - ELK
  - Snorocket

Download Owl API

- as Maven Dependency (you will need a lot of dependencies so strongly consider this option)

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>net.sourceforge.owlapi</groupId>
        <artifactId>owlapi-distribution</artifactId>
        <version>5.1.9</version>
    </dependency>
</dependencies>
```

[Filename: OwlAPI/MavenDep]

- as Github `https://github.com/owlcs/owlapi`
- as Zip `http://owlcs.github.io/owlapi/`

EXCURSION: SOME JAVA BITS AND PIECES

Lambda Expressions

- Lambda expression are a convenient way to declare anonymous classes which only have a single method with void as a return value

```
JButton butt = new JButton("Click");
butt.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button clicked");
    }
});
```

- Can be reduced to the expression below where e is a variable of the parameter type of the method and the -> defines a lambda expression

```
JButton butt = new JButton("Click");
butt.addActionListener(e -> {System.out.println("Button clicked");});
```

EXCURSION: SOME JAVA BITS AND PIECES

The Stream class

- No to be confused with InputStream/IOStream/StreamReader etc.
- Mostly used instead of an iterators
- The foreach method executes a Lambda expression for each element of the streamable object
- Foreach ignores the order of the collection!

```
List<Object> listy = new LinkedList<Object>();
listy.forEach(x -> System.out.println(x.toString()));
```

- For arrays the static method from the Stream class can be used

```
Object[] array = new Object[2];
Stream.of(array).forEach(x -> System.out.println(x.toString()));
```

EXCURSION: SOME JAVA BITS AND PIECES

The Stream class

- parallel() executes the following expression for each element in parallel without any user management efforts

```
List<Object> listy = new LinkedList<Object>();
listy.parallelStream().forEach(x -> System.out.println(x.toString()));
```

- Further filtering, mapping into other classes are possible
- With distinct(), reduce(x,y),. . . the set can be changed
- Mapped Stream like the integer Stream provide additional methods like sum

```
//How many different words in this array have more than 3 letters?
Stream.of(args).distinct().filter(x -> x.length() > 3).mapToInt(x -> 1).sum()
```

- And much more . . .

Excursion: Some Java bits and pieces

Arbitrary number of Arguments in Methods

- You might encounter pretty often methods with `method(ClassType... x)` which means those methods can handle arbitrary number of arguments from this type
- They where handle intern as an array of this type
- But it does not work for lists (You can use the toArray() method but remember that it is an array with type Object)

```
private static void methodWithArbitraryNumberofArguments(String... args) {
    System.out.println("You gave me " + args.length + " Arguments");
    Stream.of(args).forEach( x -> System.out.println("\t" + x));
}
```

EXCURSION: SOME JAVA BITS AND PIECES

Arbitrary number of Arguments in Methods

- And are called like . . .

```
methodWithArbitraryNumberofArguments("Hi","I","can","take","any","number","of"
    ,"arguments");
String[] array = {"even","arrays"};
methodWithArbitraryNumberofArguments(array);
methodWithArbitraryNumberofArguments();
```

BACK TO THE OWLAPI - THE MAJOR COMPONENTS

- **OWLEntity**: anything that can be identified with an IRI, i.e., classes, data and object properties (and annotation properties) and named individuals
- **OWLAxiom**: the basic unit
  - TBox axioms describe relations between classes and classexpressions (equivalence, subsumption, disjointness)
  - ABox axioms (assertions) describe relations between individualsand between individuals and classes/class expressions
  - RBox axioms describe relations between properties
- **OWLAnonymousIndividual**, **OWLClassExpression**, **OWLPropertyExpression**: unnamed individuals, classexpressions such as restrictions, property expressions such asthe inverse of a property
- **OWLAnnotation**: an annotation for any entity, ontology,expression or axiom; characterized by an **OWLAnnotationProperty** and an **OWLAnnotationValue**

OWLAPI - OWL ONTOLOGY MANAGER

- Base class for any operation over ontologies
- Manages one or more ontologies
  - Create new, empty ontologies
  - load from local file or web source
  - saves ontologies in many different formats
  - merges ontologies
  - renaming of IRIs
  - applies changes to ontologies

OWLAPI - INITIALIZE THE OWL ONTOLOGY MANAGER IN JAVA

```
import org.semanticweb.owlapi.apibinding.OWLManager;
import org.semanticweb.owlapi.model.IRI;
import org.semanticweb.owlapi.model.OWLOntology;
import org.semanticweb.owlapi.model.OWLOntologyCreationException;
import org.semanticweb.owlapi.model.OWLOntologyManager;

public class Manager {
   public static void main(String[] args) {
      OWLOntologyManager man = OWLManager.createOWLOntologyManager();
      IRI iri = IRI.create("http://test-Ontology");
      try {
    // Generate a new Ontology
         OWLOntology newOnt = man.createOntology(iri);
      } catch (OWLOntologyCreationException e) {
          e.printStackTrace();
      }
   }
}
```

[Filename: OwlAPI/Manager.java]

OWLAPI - LOAD ONTOLOGIES

- The Ontology Manager provides methods to load ontologies
- The format is recognized by the file extension
- Ontologies can be loaded with or without mapping the IRI to the specified IRI.
  - **loadOntologyFromOntologyDocument(SpecifiedIRI)** all IRIs of Ontology remain unchanged. This should be the default choice to keep consistency
  - **loadOntology(SpecifiedIRI)** all IRIS of the ontology are mapped to the SpecifiedIRI
- For Linked Open Data sources both method should have the same result

```
OWLOntology mappedOntology = man.loadOntology(IRI.create( //Mapped to the IRI
   "https://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3"));
OWLOntology unmappedOntology = m.loadOntologyFromOntologyDocument(IRI.create(
   "C:/MondialFiles/mondial.n3");
```

[Filename: Load/Manager.java]

OWLAPI - SAVING ONTOLOGIES

- Saving the ontology is also done with the Ontology Manager
- An Ontology Document Format class decides the type of the output

Format	Class Name
N3/Turtle	N3DocumentFormat
DL	DLSyntaxDocumentFormat
DL for Latex	LatexDocumentFormat
Manchester Format	ManchesterSyntaxDocumentFormat
Functional Format	FunctionalSyntaxDocumentFormat

- more found in the package org.semanticweb.owlapi.formats

OWLAPI - SAVING ONTOLOGIES IN JAVA

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;

import org.semanticweb.owlapi.apibinding.OWLManager;
import org.semanticweb.owlapi.formats.N3DocumentFormat;
import org.semanticweb.owlapi.model.IRI;
import org.semanticweb.owlapi.model.OWLOntology;
import org.semanticweb.owlapi.model.OWLOntologyCreationException;
import org.semanticweb.owlapi.model.OWLOntologyManager;
import org.semanticweb.owlapi.model.OWLOntologyStorageException;

public class SaveOntology {
   public static void main(String[] args) throws OWLOntologyStorageException,
   FileNotFoundException, OWLOntologyCreationException{
      OWLOntologyManager man = OWLManager.createOWLOntologyManager();
      File fileout = new File("pizza.func.owl");
      IRI pizzaontology = IRI.create("http://protege.stanford.edu/ontologies/pizza/pizza.owl");
      OWLOntology o = man.loadOntology(pizzaontology);
      man.saveOntology(o, new N3DocumentFormat(),new FileOutputStream(fileout));
   }
}
```

[Filename: OwlAPI/SaveOntology.java]

OWLAPI - MERGING ONTOLOGIES

- Merging is straight forwardly done with a merger
- The ontologies must be manage by the same OntologyManager

```
OWLOntology o1 = m.loadOntology(iri1);
OWLOntology o2 = m.loadOntology(iri2);
mergerOWLOntologyMerger merger = new OWLOntologyMerger(m);
IRI mergedOntologyIRI =IRI.create("http://www.semanticweb.com/mymergedont");
OWLOntology merged = merger.createMergedOntology(m,mergedOntologyIRI);
```

[Filename: OwlAPI/Merging.java]

OWLAPI - OWLDATAFACTOR

- Any new content for the ontology is generated by an OWLDataFactory
- This content must be part of an axiom to be applied to an ontology
- The OWLDataFactory can be derived from the Ontology Manager
- As long as not applied to a specific ontology, the content is ontology independent
- OWLDataFactory can produce . . .
  - Axioms
  - Individuals
  - Properties
  - Classes
  - Restrictions
  - Annotations
  - Relations
  - Full list `https://owlcs.github.io/owlapi/apidocs_4/org/semanticweb/owlapi/model/OWLDataFactory.html`

OWLAPI - GENERATING INDIVIDUALS

Individuals . . .

- are generated by the OWLDataFactory
- can only be added inside an axioms to an ontology

```
OWLDataFactory df = OWLManager.getOWLDataFactory();

OWLClass personClass = df.getOWLClass(IRI.create(iri + "#Person"));

OWLIndividual luke = df.getOWLNamedIndividual(IRI.create(iri + "#luke"));
OWLIndividual anakin = df.getOWLNamedIndividual(IRI.create(iri + "#anakin"));

o.add(df.getOWLClassAssertionAxiom(personClass, luke));
o.add(df.getOWLClassAssertionAxiom(personClass, anakin));

OWLObjectProperty hasFather = df.getOWLObjectProperty(IRI.create(iri + "#hasFather"));


OWLAxiom assertion = df.getOWLObjectPropertyAssertionAxiom(hasFather, luke, anakin);

o.addAxiom(assertion); //Short Version for ...
//AddAxiom addAxiomChange = new AddAxiom(o, assertion);
//man.applyChange(addAxiomChange);
```

[Filename: OwlAPI/Individuals.java]

OWLAPI - REMOVE INDIVIDUALS

- To remove individuals from an ontology a remover object is necessary
- An OWLEntityRemover removes all axioms in which the individual was mentioned.
- If the ontology does not declare certain facts but implicit mentioned facts via this individual they are also lost. E.g the only usage of an undeclared class.
- The remover only lists the necessary changes, to actually change the ontology the OntologyManager must apply these changes

```
OWLEntityRemover remover = new OWLEntityRemover(Collections.singleton(o));
o.IndividualsInSignature.forEach(ind -> ind.accept(remover));
man.applyChanges(remover.getChanges());
```

[Filename: OwlAPI/Remove.java]

OWLAPI - ADDING AND REMOVE AXIOMS

- To add or remove an AddAxiom respectivly RemoveAxiom is necessary
- This is not an axiom from the ontology but a class which adds/remove axioms to a specific ontoglogy
- Additionally the AddAxiom/RemoveAxiom must be applied to the OntologyManager to actually make them happen
- The methods add and remove from the ontology doing the same, but more convenient

```
OWLClass classA = df.getOWLClass(IRI.create(iri + "#A"));
OWLClass classB = df.getOWLClass(IRI.create(iri + "#B"));

OWLAxiom axiom = df.getOWLSubClassOfAxiom(classA, classB);
AddAxiom addAxiom = new AddAxiom(o, axiom);// add the axiom to the ontology.
man.applyChange(addAxiom);
System.out.println(o);

RemoveAxiom removeAxiom = new RemoveAxiom(o, axiom);// remove the axiom from the ontology
man.applyChange(removeAxiom);
System.out.println(o);
```

[Filename: OwlAPI/Axiom.java]

OWLAPI - CLASS EXPRESSIONS

- OWLClassExpression are used to model restrictions
- Can be either for data type or object types
- OWLClassExpression can be . . .
  - OWLClassExpression, OWLClass, OWLAnonymousClassExpression . . .
  - OWLDataAllValuesFrom, OWLDataHasValue . . .
  - OWLObjectAllValuesFrom, OWLObjectHasValue . . .
  - OWLObjectUnionOf, OWLObjectIntersectionOf, OWLObjectComplementOf . . .
  - OWLObjectCardinalityRestriction, OWLDataMaxCardinality, OWLDataMinCardinality, OWLDataExactCardinality . . .
  - etc.

OWLAPI - OBJECT TYPE & DATA TYPE RESTRICTIONS

For object types . . .

```
OWLObjectProperty hasEntrance = df.getOWLObjectProperty(IRI.create(iri + "#hasEntrance"));
OWLClass door = df.getOWLClass(IRI.create(iri + "#Door"));
OWLClassExpression hasEntranceDoor = df.getOWLObjectSomeValuesFrom(hasEntrance, door);
OWLClass building = df.getOWLClass(IRI.create(iri + "#Building"));
OWLSubClassOfAxiom ax = df.getOWLSubClassOfAxiom(building, hasEntranceDoor);
man.applyChange(new AddAxiom(o, ax));
```

[Filename: OwlAPI/Existential.java]

OBJECT TYPE & DATA TYPE RESTRICTIONS

For data types . . .

```
OWLDataProperty hasAge = df.getOWLDataProperty(IRI.create(iri + "#hasAge"));
OWLDataRange greaterThan18 = df.getOWLDatatypeRestriction(df.getIntegerOWLDatatype(),
    OWLFacet.MIN_INCLUSIVE, df.getOWLLiteral(18));

OWLClassExpression adultDefinition = df.getOWLDataSomeValuesFrom(hasAge, greaterThan18);

OWLClass adult = df.getOWLClass(IRI.create(iri + "#Adult"));
OWLSubClassOfAxiom ax = df.getOWLSubClassOfAxiom(adult, adultDefinition);

man.applyChange(new AddAxiom(o, ax));
```

[Filename: OwlAPI/Datatype.java]

ANNOTATIONS

Annotation . . .

- can be assigned with the OWLAnnotation
- are created by the DataFactory
- are also add with an instance of the AddAxiom class
- can be either RDFSLabel, RDFSComment, RDFSIsDefinedBy or RDFSSeeAlso

```
OWLClass buildingClass = df.getOWLClass(IRI.create(iri + "#Building"));

OWLAnnotation commentAnnoEN = df.getOWLAnnotation(df.getRDFSComment(),
   df.getOWLLiteral("A class which represents Buildings", "en"));
OWLAxiom axEN = df.getOWLAnnotationAssertionAxiom(buildingClass.getIRI(), commentAnnoEN);

OWLAnnotation commentAnnoDE = df.getOWLAnnotation(df.getRDFSComment(),
   df.getOWLLiteral("Eine Klasse, die Gebaeude representiert", "de"));
OWLAxiom axDE = df.getOWLAnnotationAssertionAxiom(buildingClass.getIRI(), commentAnnoDE);

man.applyChange(new AddAxiom(o, axEN));
man.applyChange(new AddAxiom(o, axDE));
```

[Filename: OwlAPI/Annotations.java]

OWLAPI - CLASS DECLARATION AND EXPLORATION

- Classes can be implicit defined, by just creating an individual with this class
- But explicit class declarations avoid mistakes and are requested by the validation.
- Declared classes can exist without any individual of this class (other then implicit classes)

```
OWLClass person = df.getOWLClass(iri + "#Person");
OWLDeclarationAxiom da = df.getOWLDeclarationAxiom(person);
o.add(da);
```

[Filename: OwlAPI/Declaration.java]

- The existing classes can be explored via a JavaStream

```
o.classesInSignature().forEach(i -> System.out.println(i));
o.subClassAxiomsForSubClass(classA).forEach(i -> System.out.println(i));
```

[Filename: OwlAPI/Validation.java]

OWLAPI - THIS WALKER IS MADE FOR WALKING ...

- Visits all axioms of an ontology
- Executes the `visit(OWLEntity e)` method on each
- The `visit` method can be overriden with any subclass of OwlEntity to specify the behavior

```
OWLOntology o = m.loadOntologyFromOntologyDocument(pizza_iri);
OWLOntologyWalker walker =new OWLOntologyWalker(Collections.singleton(o));
OWLOntologyWalkerVisitor<Object> visitor =new OWLOntologyWalkerVisitor<Object>(walker) {
    @Override
        public void visit(OWLObjectSomeValuesFrom desc) {
            System.out.print("Some Values From Axiom: " + desc+ " ");
            System.out.println(getCurrentAxiom());
        }

        @Override
        public void visit(OWLSubClassOfAxiom desc) {
            System.out.print("SubclassOf Axiom: "  + desc + " ");
            System.out.println(getCurrentAxiom());
        }
};
walker.walkStructure(visitor)
```

[Filename: OwlAPI/OntologyWalker.java]

OWLAPI - RENDERER

Renderers are used to format the ontology in a certain format similar to the Ontology Document Format classes but change the toString methods

- SimpleRenderer - Default used Renderer
- LatexOWLObjectRenderer - Latex-Document-Format
- ManchesterOWLSyntaxOWLObjectRendererImpl - Manchester Syntax
- DLSyntaxObjectRenderer - DL Syntax
  - the encoding of the output must be UTF-8 otherwise some symbols are not displayed correctly
  - To switch in Eclipse: Window→preferences→General→Workspace

```
OWLOntologyWalker walker = new OWLOntologyWalker(Collections.singleton(o));
ManchesterOWLSyntaxOWLObjectRendererImpl ren = new ManchesterOWLSyntaxOWLObjectRendererImpl();
OWLOntologyWalkerVisitor visitor = new OWLOntologyWalkerVisitor(walker) {
    @Override
    public void visit(OWLObjectSomeValuesFrom desc) {
        System.out.println(ren.render(getCurrentAxiom()));
    }

    @Override
    public void visit(OWLSubClassOfAxiom desc) {
        System.out.println(ren.render(getCurrentAxiom()));
    }
};
walker.walkStructure(visitor);
```

[Filename: OwlAPI/SimpleRenderer.java]

PROFILE VALIDATION

- An OWL 2 profile (commonly called a fragment or a sublanguage in computational logic) is a trimmed down version of OWL 2 that trades some expressive power for the efficiency of reasoning
- The profile can check an ontology if its valid for this profile see also `https://www.w3.org/TR/owl2-profiles/`
- Profiles are . . .
  - DL - expressiveness of Describtion Logic
  - EL - very large numbers of classes and/or properties
  - QL - complete query answering is in LOGSPACE
  - RL - scalable reasoning without sacrificing too much expressive power
  - etc. . .

```
OWL2DLProfile profile = new OWL2DLProfile();
OWLProfileReport report = profile.checkOntology(o);
for(OWLProfileViolation v:report.getViolations()) {
System.out.println(v);
}
```

[Filename: OwlAPI/Validation.java]

OWLAPI - CLASS HIERARCHY

- For hierarchy generation a reasoner is necessary
- The only build in reasoner is the "StructuralReasoner"
- The StructuralReasoner is generated by a "StructuralReasonerFactory"

```
private void getHierarchy() {
    OWLClass clazz = df.getOWLThing();
    System.out.println("Class   :  " + clazz);
    StructuralReasonerFactory srf = new StructuralReasonerFactory();
    reasoner = srf.createReasoner(o);
    try {
        printHierarchy(clazz, 0);
    } catch (OWLException e) {
        e.printStackTrace();
    }
}
```

[Filename: OwlAPI/getHierarchy.java]

OWLAPI - CLASS HIERARCHY HELPER

```
public void printHierarchy(OWLClass clazz, int level, HashSet<OWLClass> visited) throws OWLException {
    ManchesterOWLSyntaxOWLObjectRendererImpl r = new ManchesterOWLSyntaxOWLObjectRendererImpl();
    if (reasoner.isSatisfiable(clazz)) {
        visited.add(clazz);
        for (int i = 0; i < level * 4; i++) {
        System.out.print(" ");
    }
    System.out.println(r.render(clazz));
    NodeSet<OWLClass> classes = reasoner.getSubClasses(clazz, true);
    classes.nodes().forEach(subClass -> subClass.forEach(c -> {
    try {
        printHierarchy(c, level + 1, visited);
    } catch (OWLException e) {
        e.printStackTrace();
    }
    }));
}
}
```

[Filename: OwlAPI/printHierarchy.java]

OWLAPI - NECESSARY PROPERTIES

- Idea:
  - Check if the class fits into the domain of the property
  - Get all classes which might have the property range over something
  - Compute the intersection of class and the inverse of possible property owners
  - If the ontology is still satisfiable the class not necessarly need this property

```
private void printProperties(OWLObjectProperty prop, OWLClass cls) {
    ManchesterOWLSyntaxOWLObjectRendererImpl r = new ManchesterOWLSyntaxOWLObjectRendererImpl();
    o.objectPropertyDomainAxioms(prop).forEach(ax -> {
        OWLClassExpression c = ax.getDomain();
        Node<OWLClass> allEquivalentClasses = reasoner.getEquivalentClasses(c);
        allEquivalentClasses.forEach(ec -> {
            if (ec.equals(cls)) {
            OWLClassExpression restriction = df.getOWLObjectSomeValuesFrom(prop, df.getOWLThing());
            OWLClassExpression intersection =
                df.getOWLObjectIntersectionOf(cls, df.getOWLObjectComplementOf(restriction));
            if (!reasoner.isSatisfiable(intersection))
                System.out.println("Instances of " + r.render(cls) + " must have " + r.render(prop));
            }
        });
    });
}
```

[Filename: OwlAPI/printNeededProperties.java]

1.5 HermiT OWL Reasoner - "The New Kid on the OWL Block"

- HermiT is a reasoner for ontologies written in OWL based on hypertableau algorithm.
- Given an OWL file, HermiT can . . .
  - Determine whether or not the ontology is consistent and satisfiable
  - Identify subsumption relationships between classes
  - Give decent reports why something is not working

WHAT HERMIT CANNOT DO

HermiT is not a replacement for pellet or the internal Jena reasoner because . . .

- It cannot understand SPARQL
- The New Kid is actually pretty old and does not seem to be updated in a while . . .
- It cannot handle ontologies with XMLSchema#date like Mondial. . .
  but XMLSchema#dateTime works just fine

SO WHY USE HERMIT ANYWAYS?

- It is very useful during ontology creation
- Cooperates better with Protégé (less craches and better information)

HERMIT - DOWNLOAD

- Hermit can be downloaded at `http://www.hermit-reasoner.com/download.html`
- The ZIP-File contains 3 items (and a readme)
  - `HermiT.jar` - the command line tool
  - `org.semanticweb.HermiT.jar` - the Java API
  - A folder called "`project`" which is actually an eclipse project which can directly be imported into eclipse. It contains some demo files to play arround with.

HERMIT - MARVEN

- For maven (at least I) had to add not only the artifact but also explicitly the compiler

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>

<dependency>
    <groupId>net.sourceforge.owlapi</groupId>
    <artifactId>org.semanticweb.hermit</artifactId>
    <version>1.4.3.517</version>
</dependency>
```

HERMIT - USED AS A COMMAND LINE TOOL

- HermiT is called as usual with `java -jar HermiT.jar`
- For help you can add `--help`
- Commands can be chained and usually have a short form and a written out form
  - `java -jar HermiT.jar --classify --classifyOPs --verbose=2 --output=output.owl https://protege.stanford.edu/ontologies/pizza/pizza.owl`
  - `java -jar HermiT.jar -cO -v2 -P -ooutput.owl https://protege.stanford.edu/ontologies/pizza/pizza.owl`

HERMIT - BLACK BOX DEBUGGING

- Together the OWLAPI & HermiT are able check if an ontology is not satisfiable or inconsistent
- And if so to actually produce useful reports which explain why that is the case
- To generate a report the following steps a necessary:
  1. The reasoner has to be configured in a way that it does not throw an exception if unsatisfiability, inconsistency . . . occurs
  2. Then isSatisfiable, isConsistence, . . . has to be called
  3. Generate the explanation classes from the OWLAPI
  4. Get and iterate the explanations

Example: Add Ice creme to the pizza ontology

```
IRI icecreamIRI=IRI.create("http://www.co-ode.org/ontologies/pizza/pizza.owl#IceCream");
OWLClass icecream=dataFactory.getOWLClass(icecreamIRI);

ReasonerFactory factory = new ReasonerFactory();
Configuration configuration=new Configuration();
configuration.throwInconsistentOntologyException=false;

OWLReasoner reasoner=factory.createReasoner(ontology, configuration);
System.out.println("Is icecream satisfiable? "+ reasoner.isSatisfiable(icecream));

System.out.println("Computing explanations...");
BlackBoxExplanation exp=new BlackBoxExplanation(ontology, factory, reasoner);
HSTExplanationGenerator multExplanator=new HSTExplanationGenerator(exp);
Set<Set<OWLAxiom>> explanations=multExplanator.getExplanations(icecream);

for (Set<OWLAxiom> explanation : explanations) {
    System.out.println("------------------");
    System.out.println("Axioms causing the unsatisfiability: ");
    for (OWLAxiom causingAxiom : explanation) {
        System.out.println(causingAxiom);
    }
    System.out.println("------------------");
}
```

[Filename: OwlAPI/IceIsSatisfiable.java]

Explanation Output for different Renderer

```
 Axioms causing the unsatisfiability:
Simple Render
---------
SubClassOf(:IceCream ObjectSomeValuesFrom(:hasTopping :FruitTopping))
DisjointClasses(:IceCream :Pizza)
ObjectPropertyDomain(:hasTopping :Pizza)
---------
Manchester Syntax
---------
IceCream SubClassOf hasTopping some FruitTopping
IceCream DisjointWith Pizza
hasTopping Domain Pizza
---------
DL Syntax
---------
IceCream ⊑ ∃ hasTopping.FruitTopping
⊤ ⊒ ∀ isToppingOf.Pizza
isToppingOf ≡ hasTopping⁻
IceCream ⊒ ¬ Pizza
```

Example: Add an inconsistent individual

```
OWLAxiom ax=dataFactory.getOWLClassAssertionAxiom(icecream,
   dataFactory.getOWLNamedIndividual(IRI.create("...#dummyIndividual")));
   manager.addAxiom(ontology, ax);
   reasoner=factory.createReasoner(ontology, configuration);
   System.out.println("Is the changed ontology consistent? "+reasoner.isConsistent());
   System.out.println("Computing explanations for the inconsistency...");
   factory=new Reasoner.ReasonerFactory() {
       protected OWLReasoner createHermiTOWLReasoner(
           org.semanticweb.HermiT.Configuration configuration,OWLOntology ontology) {
           configuration.throwInconsistentOntologyException=false;
           return new Reasoner(configuration,ontology);
       }
   };
   exp=new BlackBoxExplanation(ontology, factory, reasoner);
   multExplanator=new HSTExplanationGenerator(exp);
   explanations=multExplanator.getExplanations(dataFactory.getOWLThing());

   for (Set<OWLAxiom> explanation : explanations) {
       System.out.println("------------------");
       System.out.println("Axioms causing the inconsistency: ");
       for (OWLAxiom causingAxiom : explanation) {
           System.out.println(causingAxiom);
       }
       System.out.println("------------------");
   }
```

[Filename: OwlAPI/Inconsistent.java]

Explanation Output for different Renderer

Axioms causing the inconsistency:

Simple Render

```
SubClassOf(:IceCream ObjectSomeValuesFrom(:hasTopping :FruitTopping))
DisjointClasses(:IceCream :Pizza)
ObjectPropertyDomain(:hasTopping :Pizza)
ClassAssertion(:IceCream :dummyIndividual)
```

Manchester Syntax

```
IceCream SubClassOf hasTopping some FruitTopping
IceCream DisjointWith Pizza
hasTopping Domain Pizza
dummyIndividual Type IceCream
```

DL Syntax

$$\texttt{IceCream} \sqsubseteq \exists\ \texttt{hasTopping.FruitTopping}$$
$$\top \sqsupseteq \forall\ \texttt{isToppingOf.Pizza}$$
$$\texttt{isToppingOf} \equiv \texttt{hasTopping}^{-}$$
$$\texttt{IceCream} \sqsupseteq \neg\ \texttt{Pizza}$$
$$\texttt{IceCream(dummyIndividual)}$$

HERMIT - ENTAILMENT CHECKS

- To check if an axiom (or a set of axioms) holds for a given ontology hermit can perform so called entailment checks
- Therefore . . .
  - An ontology has to be loaded into the OWLAPI-Ontology-Manager
  - OWLAxiom has to be generated
  - Generate a HermiT-Instance for the ontology and call the method isEntailed(Axiom)

Example: Do margherita pizzas have a topping that is mozarella or goats cheese?

```
OWLOntology ontology = manager.loadOntology(IRI.create(
        "https://protege.stanford.edu/ontologies/pizza/pizza.owl"));

OWLClass margherita = dataFactory.getOWLClass(IRI.create("...#Margherita"));
OWLClass mozzarellaTopping = dataFactory.getOWLClass(IRI.create("...#MozzarellaTopping"));
OWLClass goatsCheeseTopping = dataFactory.getOWLClass(IRI.create("...#GoatsCheeseTopping"));

OWLObjectProperty hasTopping = dataFactory.getOWLObjectProperty(IRI.create("...#hasTopping"));

OWLClassExpression mozarellaOrGoatsCheese =
        dataFactory.getOWLObjectUnionOf(mozzarellaTopping, goatsCheeseTopping);
OWLClassExpression hasToppingMozarellaOrGoatsCheese =
        dataFactory.getOWLObjectSomeValuesFrom(hasTopping, mozarellaOrGoatsCheese);

OWLAxiom axiom = dataFactory.getOWLSubClassOfAxiom(margherita, hasToppingMozarellaOrGoatsCheese);

Reasoner reasoner = new Reasoner(ontology);

System.out.println("Do margherita pizzas have a topping that is mozarella or goats cheese?"
    + reasoner.isEntailed(axiom));
```

[Filename: OwlAPI/Mozarella.java]

HERMIT - QUERY THE HERMIT

- Hermit can give you information about individuals as well as classes
- But it does not speak any Sparql
- Generating open answer set which are not specific for a single entity are quiet tedious
- Therefore queries must be ask in a programmatical way
- E.g. the Fish puzzle can be answered by . . .
  - Create a new Reasoner (with the default configuration)
  - Create the iri and the individual you what to have information about
  - Use methods of the reasoner to generate the answer set
- **Surprisingly the output is correct, but incomplete and not deterministic**

```
Configuration configuration = new Configuration();
IRI iri = IRI.create("foo:bla#fish");
OWLNamedIndividual fish = df.getOWLNamedIndividual(iri);
OWLReasoner reasoner = new Reasoner(configuration , o);
reasoner.sameIndividuals(fish).forEach(x -> System.out.println(x));
```

[Filename: OwlAPI/fishpuzzle.java]

1.6 Protégé

- A free, open-source ontology editor and framework for building intelligent systems
- It is build with the OwlAPI and supports nearly all of its features
- Protégé was developed by the Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine.
- The core Idea is providing a user-friendly environment to generate ontologies to enable medical researches to create own ontologies
- Actually if ontologies are used outside of the database research area, they are very likely created with Protégé

PROTÉGÉ IS HIGHLY EXTENDABLE WITH PLUGINS

- Actually everyone and their dog have written a plugin
- In my experience most of them are bad and neglected long ago and/or commercial
- The other plugins are often already part of the distributed Protégé versions
- Word of Advice: Be sure that you actually need a plugin, if not stay with the distributed version
- Otherwise: Consider writing your own, equally buggy, plugin which works for you but no one else, like everyone else did!
- Somewhat curated collection of higher quality plugins can be found at the wiki site
  `https://protegewiki.stanford.edu/wiki/Protege_Plugin_Library`
  Some of them are already in the current version
  Other were once in a distributed version but are removed for one or another reason

Protégé - Create Classes, Object-Properties and Data-Properties

- Select the main tab **Entity**
- Select the sub tab **Class hierarchy**, **Object-restriction creator hierarchy** or **Data-Properties hierarchy** (It all works the same and is very consistent through the whole program)
- Right-Click on the tree structure on the left side and **new subclass/subproperty**. There is a singular and a plural version, to create one or multiple at once.
- To generate multiple classes/properties write just one name per line
- The new Entity is now displayed in the tree according to its superclasses
- This Entity can be drag and drop in the tree to change its superclasses

PROTÉGÉ - ADDING ATTRIBUTES TO ENTITY

- Select the entity in the Hierarchy
- The main view shows on top the annotation and on the button half the equivalence classes, the superclasses, disjoint with, disjoint Union of
- You can add new element to each element by clicking the plus icon
- Depending on the context either the **class hierarchy**, **object restriction creator** and **data restriction creator** or **the class expression editor**
  - The hierarchy tab allows multi-selection (you can use shift and ctr)
  - The Object/Data expression tab let you select one class and one property and the cardinality
  - the class expression editor tab let you write an expression of arbitrary complexity in the Manchester syntax (like when ever you have to type you have auto completion if possible ctr + space)

PROTÉGÉ - INDIVIDUALS

-

PROTÉGÉ - REASONERS

-

PROTÉGÉ - SPARQL QUERIES

-

PROTÉGÉ - MANCHESTER SYNTAX IN CLASS EXPRESSIONS

Keyword	Example	Meaning
some	hasChild some Person	property hasChild ranging at least over one Person
value	hasChild value Alice	property hasChild ranging at least over the individual Alice
only	hasChild only Female	class has the property hasChild and this property ranges only over Female
min	hasChild min 3 Female	has at least 3 times the property hasChild ranging over Female (but maybe also some hasChild ranging over Male)
max	hasChild max 5 Female	has at most 5 times the property hasChild ranging over Female
exactly	hasChild exactly 4 Female	has exactly 4 times the property hasChild ranging over Female

PROTÉGÉ - MANCHESTER SYNTAX IN CLASS EXPRESSIONS

Keyword	Example	Meaning
and	Person and (hasChild some Female)	Is a Person and has a female Child
or	(hasChild some Male) or (hasChild some Female)	Person which has at least a Male or a Female child
not	not(hasChild some Female)	Does not have a Female child

PROTÉGÉ - REASONER HEAP SPACE

- For actual serious reasoning you can not have something like to much heap space (but for your OS)
- Protégé only takes about 300 mb by default
- To speed things up open the run.bat (in Windows maybe something different in the linux version)
- Change the parameters accordingly -Xmx (Maximal Heap space in megabyte) -Xms (Minimal Heap space in megabyte)