Software Agent Computing

Using ontology

Maria Ganzha, Maciej Gawinecki,

Ambroise Ncho, Jean Vaucher

Why agents need ontology?

- Communication problem:
  - Language
  - Vocabulary
  - Protocol

- New domain – 3 ways to represent content :
  - Using strings:
    John is 35 years old. Bill is his father and is 67 years old.
  - Exploiting Java technology to transmit **serialized** Java objects directly (not readable by humans)
  - Defining the objects to be transferred as extension of predefined classes, so that Jade can encode and decode messages in a standard FIPA format

Content type	Getting content	Setting content
Strings	`getContent()`	`setContent()`
Java Objects	`getContentObject()`	`setContentObject()`
Ontology	`extractContent()`	`fillContent()`

What will we do…

- Bank example – 2 implementations:
  1) communication between agents using serialized Java objects
  2) communication between agents using BankOntology
     - convert our objects into an ontology with the support provided by JADE
     - FIPA SL in Coder/Decoder classes

How it works – example (1)

Bank apllication:

- **`BankServerAgent`** – acts as a *server*
- **`BankClientAgent`** – acts as a *client*
- **`BankVocabulary`** – defines the constants which represent the terms that constitute the specific language of the agents

http://www.iro.umontreal.ca/~vaucher/Agents/Jade/Ontologies.htm

How it works – example (2)

- Conversation between two agents follows very simple protocol:
  - To create an account or to make an operation,
    the *client agent* sends a REQUEST message to the *server agent*
  - The *server agent* responds with an INFORM message after processing the request
    or with a NOT_UNDERSTOOD message if it cannot decode the content of the message
  - To query information about a specific account, the client agent sends a QUERY_REF to the server agent which responds with an INFORM after processing the query
    or with a NOT_UNDERSTOOD if it cannot decode the content of the message

http://www.iro.umontreal.ca/~vaucher/Agents/Jade/Ontologies.htm

Messages with serialized Java objects

- First step – let's identify the pertinent concepts and actions and to define these as classes:
  - **`Account`** – concept of a bank savings account
  - **`Operation`** – concept of a bank operation
  - **`MakeOperation`** – action of making an operation such as deposit or withdrawal
  - **`OperationList`** – concept of the list of last operations
  - **`CreateAccount`** – action of creating an account
  - **`Information`** – concept of querying information about an account such as the balance and the list of last operations
  - **`Problem`** – result of an action that fails

Implementation (1) – definition MakeOperation class

```
class MakeOperation implements java.io.Serializable {
    private String accountId;
    private int type;
    private float amount;

    public String getAccountId() {
        return accountId;
    }
    public int getType() {
        return type;
    }
    public float getAmount() {
        return amount;
    }
    public void setAccountId(String accountId) {
        this.accountId = accountId;
    }
    public void setType(int type) {
        this.type = type;
    }
    public void setAmount(float amount) {
        this.amount = amount;
    }
}
```

Implementation (2)

`BankClientAgent` sends a REQUEST to **`BankServerAgent`** to carry out a given operation:

```
MakeOperation mo = new MakeOperation();
mo.setAccountId(acc.getId());
mo.setType(command);
mo.setAmount(amount);

ACLMessage msg = new
ACLMessage(CLMessage.REQUEST);
msg.addReceiver(server);
try {
    msg.setContentObject(mo);
} catch (Exception ex) {
    ex.printStackTrace();
}
send(msg);
```

Implementation (3)

The BankServerAgent receives and decode the content of the message as implemented in the inner classes **ReceiveMessages**

```
class ReceiveMessages extends CyclicBehaviour {
    public ReceiveMessages(Agent a) {
            super(a);
        }
    public void action() {
    ACLMessage msg = receive();
    if (msg == null) { block(); return; }
    try {
        Object content = msg.getContentObject();
        switch (msg.getPerformative()) {
       case (ACLMessage.REQUEST):
       if (action instanceof CreateAccount)
            addBehaviour(new HandleCreateAccount(myAgent, msg));
       else
            if (content instanceof MakeOperation)
               addBehaviour(new HandleOperation(myAgent, msg));
       ...
    }
 }
```

Implementation (4)

… and **HandleOperation:**

```
class HandleOperation extends OneShotBehaviour {
    ACLMessage request;

    public HandleOperation(Agent a, ACLMessage request) {
        super(a);
        this.request = request;
    }

    public void action() {
        try {
            Operation op = (Operation) request.getContentObject();
            ACLMessage reply = request.createReply();
            // Process the operation Object
            result = processOperation(op);
            ...
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

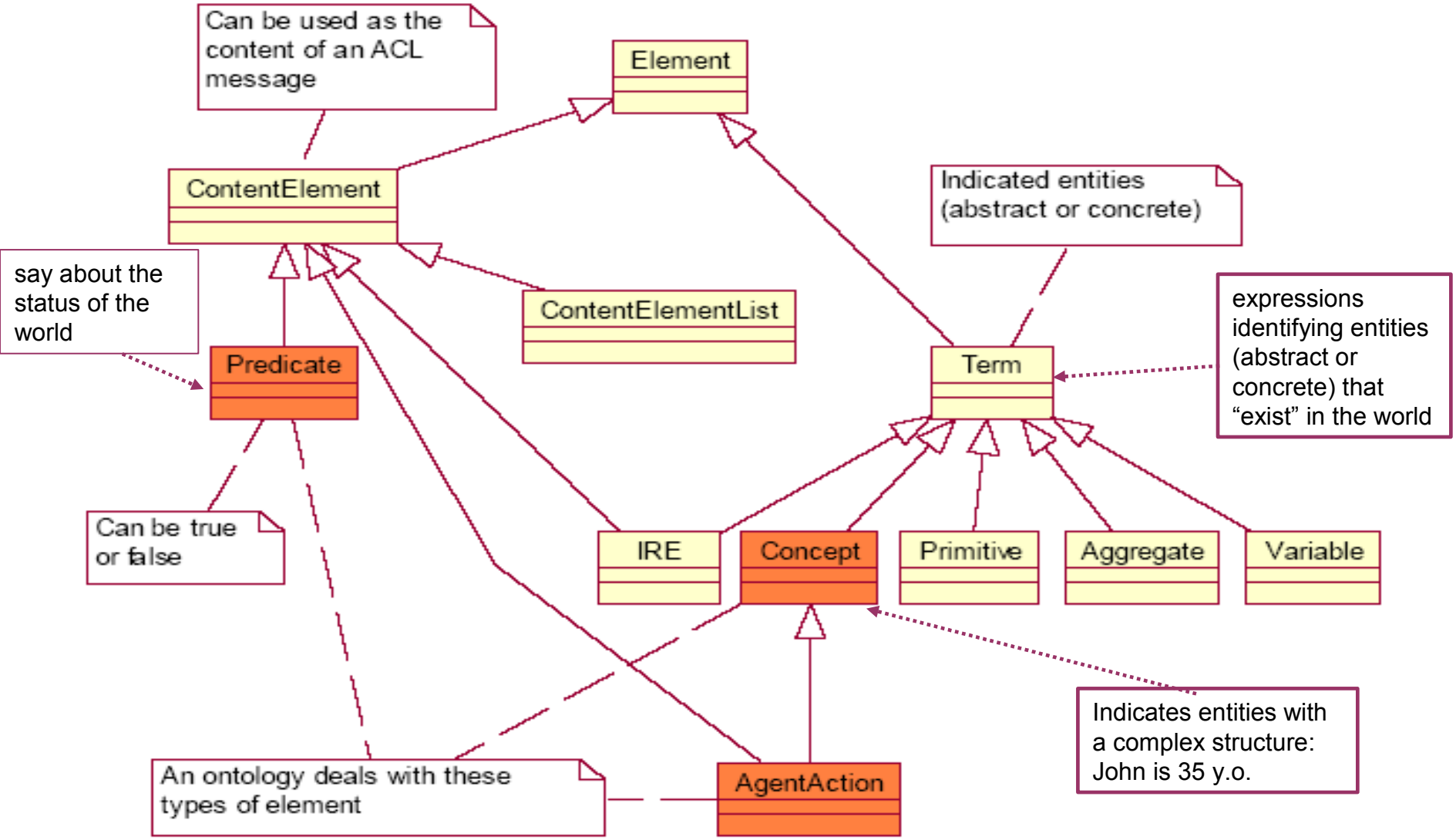
Defining an application-specific ontology

- An application-specific ontology describes the **elements** that can be used as **content** of agent messages
  - a vocabulary that describes the terminology of concepts used by agents in their space of communication
  - the nomenclature of the relationships between these concepts
  - their semantic and structure

- The implementation of an ontology for an application:
  - to extend the class `Ontology` predefined in JADE (`jade.content.onto.Ontology`)
  - to add a set of element schemas describing the structure of
    - concepts
    - actions
    - predicates

These elements are allowed to compose the content of messages

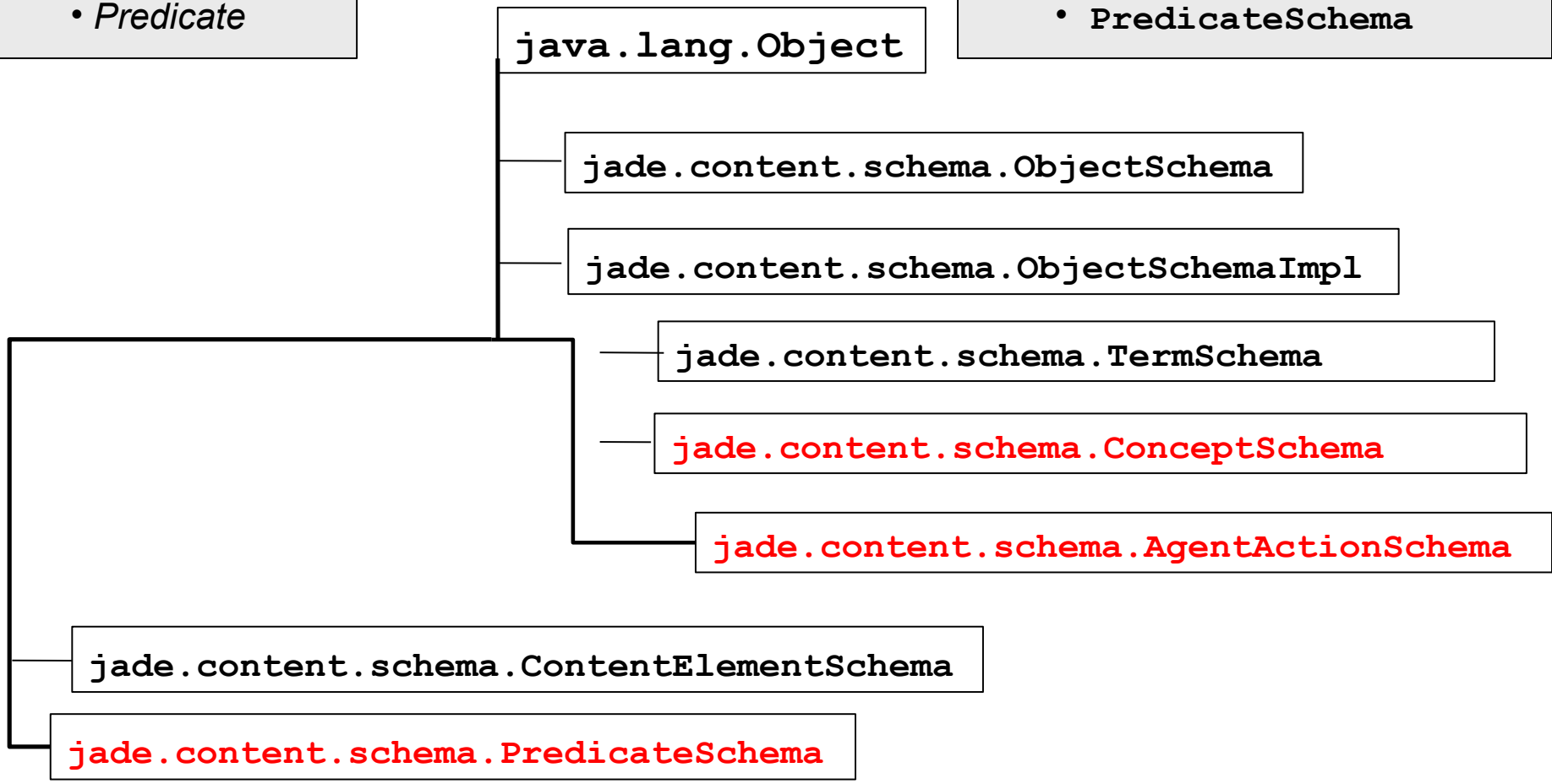
Content reference model



Hierarchy and inheritance…

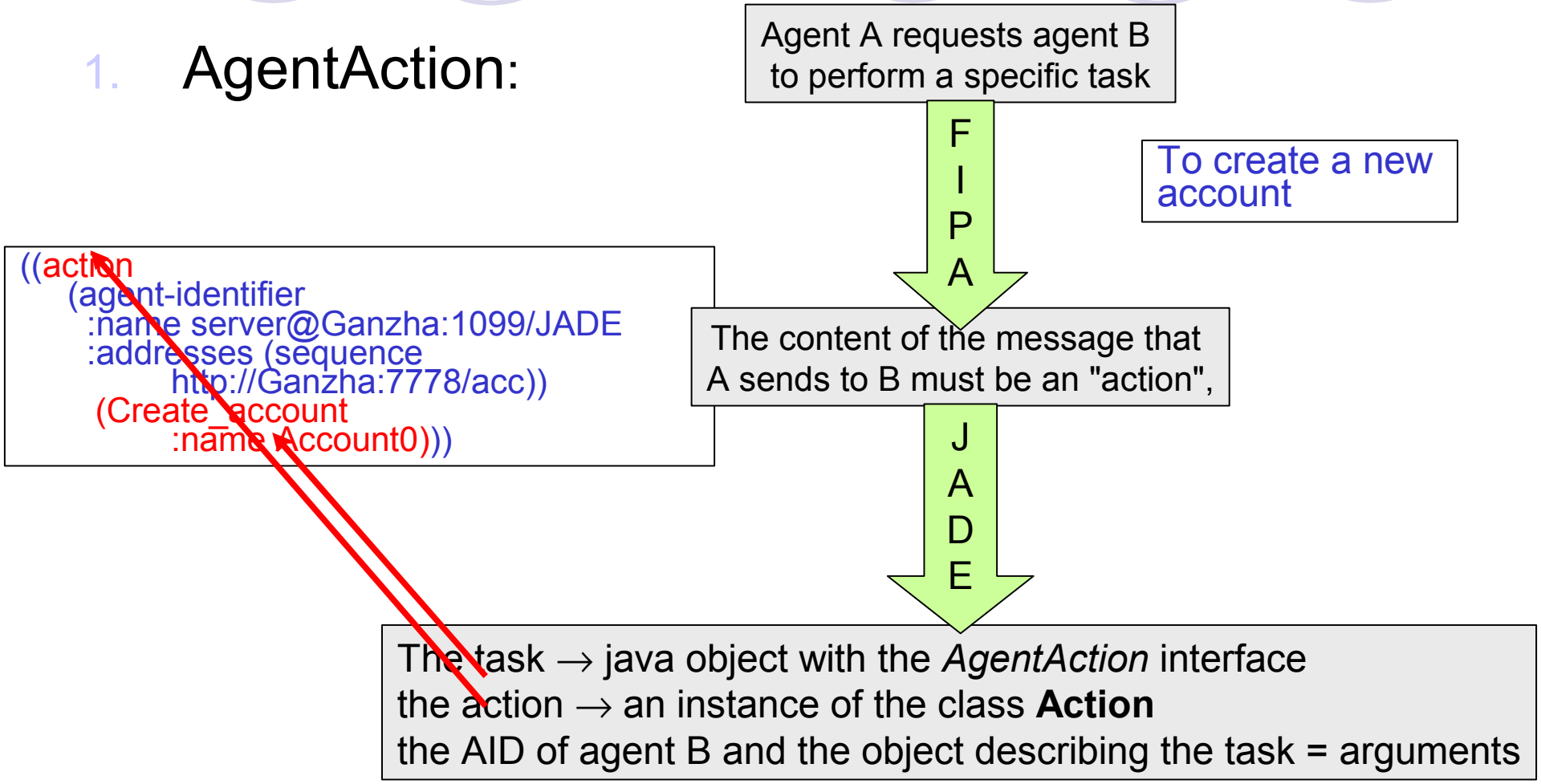
Start point:
- *Concept,*
- *AgentAction,*
- *Predicate*

Ontology classes:
- **`ConceptSchema`,**
- **`AgentActionSchema`,**
- **`PredicateSchema`**



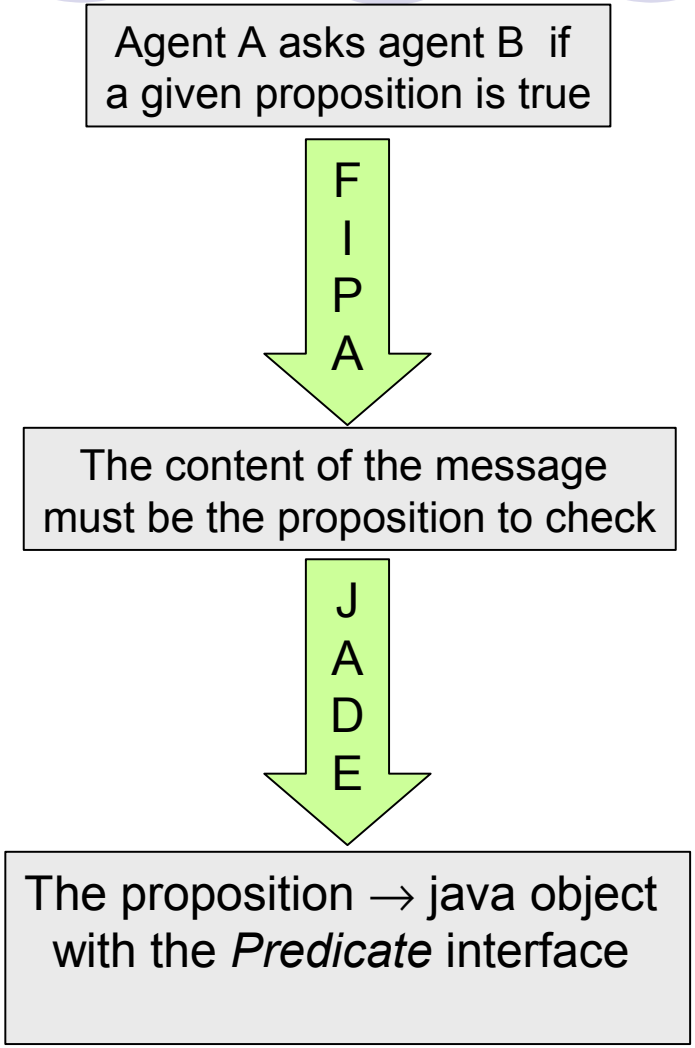
How create an ontology “from” java classes

1. AgentAction:



How create an ontology “from” java classes

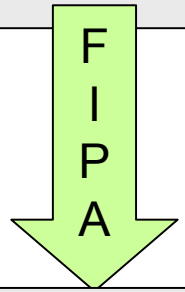
1. Predicate :



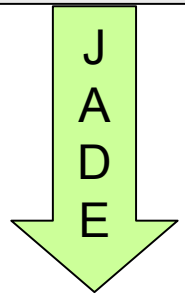
How create an ontology “from” java classes

1. Concept:

BanClientAgent requests the BankServerAgent to perform an action = make a deposit on a given account



Request → Action (MakeOperation)
→ Result = Account/Problem



The result → java object with the *Concept* interface

```
((result
   (action
     (agent-identifier
       :name server@Ganzha:1099/JADE
       :addresses (sequence http://Ganzha:7778/acc))
     (MakeOperation
       :type 2
       :amount 340.0
       :accountId "81409330"))
   (Account
     :id "81409330" :name Account0 :balance 336.25)))
```

How create an ontology “from” Java classes

- the **`Account`** class
  - now implements the ***`Concept`*** interface
- the **`Operation`** class
  - implements the ***`Concept`*** interface
- the **`MakeOperation`** class
  - implements the ***`AgentAction`*** interface
- the **`CreateAccount`** class
  - implements the ***`AgentAction`*** interface
- the **`Information`** class
  - implements the ***`AgentAction`*** interface
- the **`Problem`** class
  - implements the ***`Concept`*** interface

The **`OperationList`** class disappeared
The **`Result`** class (provided by JADE) holds the result of actions that are performed by the server agent already contains a **`List`** object as attribute

How it works – step 1

To define the vocabulary of agents communication space:

```
public interface BankVocabulary {
   ...
   public static final String MAKE_OPERATION = "MakeOperation";
   public static final String MAKE_OPERATION_TYPE = "type";
   public static final String MAKE_OPERATION_AMOUNT = "amount";
   public static final String MAKE_OPERATION_ACCOUNTID = "accountId";
   ...
}
```

How it works – step 2

To define the java class that specifies the structure and semantic of the object ***MakeOperation***

```
class MakeOperation implements AgentAction {
    private String accountId;
    private int type;
    private float amount;

    public String getAccountId()

    public int getType() { return

    public float getAmount() { return amount; }

    public void setAccountId(String accountId) {
        this.accountId = accountId; }

    public void setType(int type) {
        this.type = type; }

    public void setAmount(float amount) { this.amount = amount; }
 }
```

!!!You can not choose any name you like at this step - they must imperatively match (case insensitive) the names that you gave to these attributes when defining the vocabulary: e.g. **`MAKE_OPERATION_TYPE`** is "type"→ the name of the attribute must be *type* and the corresponding *get* and *set* methods must be **`getType()`** and **`setType()`**.

How it works – step 3 (a)

Define the schema of the object

```
public class BankOntology extends Ontology implements BankVocabulary {
    // ----------> The name identifying this ontology
    public static final String ONTOLOGY_NAME = "Bank-Ontology";
    // ----------> The singleton instance of this ontology
    private static Ontology instance = new BankOntology();
    // ----------> Method to access the singleton ontology object
    public static Ontology getInstance() {
        return instance;
    }
    // Private constructor
    private BankOntology() {
        super(ONTOLOGY_NAME, BasicOntology.getInstance());
        try {
            // ------- Add Concepts
            ...
            // ------- Add AgentActions
            ...
```

How it works – step 3 (b)

In the **BankOntology** class the lines of code that specifiy the schema of the concept **MakeOperation**:

```
    // MakeOperation
    add(as = new AgentActionSchema(MAKE_OPERATION),
        MakeOperation.class);
    as.add(MAKE_OPERATION_TYPE,
        (PrimitiveSchema)getSchema(BasicOntology.INTEGER),
        ObjectSchema.MANDATORY);
    as.add(MAKE_OPERATION_AMOUNT,
        (PrimitiveSchema) getSchema(BasicOntology.FLOAT),
        ObjectSchema.MANDATORY);
    as.add(MAKE_OPERATION_ACCOUNTID,
        (PrimitiveSchema) getSchema(BasicOntology.STRING),
        ObjectSchema.MANDATORY);
    ...
} catch (OntologyException oe) {
    oe.printStackTrace();
}
```

MANDATORY – cannot be null
OPTIONAL – can have a null value

How it works – step 4

- To register with the agent's content manager:
  - the **ontology** and the **language** that will be used for:
    - assembling (**encoding**)
    - parsing (**decoding**) the content of messages.

```
public class BankClientAgent extends Agent
    implements BankVocabulary {
    ...
    private Codec codec = new SLCodec();
    private Ontology ontology = BankOntology.getInstance();
    protected void setup() {
        // Register language and ontology
        getContentManager().registerLanguage(codec);
        getContentManager().registerOntology(ontology);
        ...
    }
    ...
}
// class BankClientAgent
```

Using a new ontology

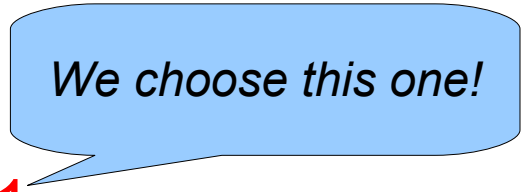
```
public class BankClientAgent extends Agent implements
    BankVocabulary {
    // ...
    void requestOperation() {
        // ...
        MakeOperation mo = new MakeOperation();
        mo.setType(command);
        mo.setAmount(amount);
        mo.setAccountId(acc.getId());
        sendMessage(ACLMessage.REQUEST, mo);
    }
    // ...
    void sendMessage(int performative, AgentAction action) {
        // ...
        ACLMessage msg = new ACLMessage(performative);
        msg.setLanguage(codec.getName());
        msg.setOntology(ontology.getName());
        try {
            getContentManager().fillContent(msg,
                new Action(server, action));
            msg.addReceiver(server);
            send(msg);
            // ...
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

1) set the attributes of your java object

2) set the language and ontology

Installing tools for ontologies

- Ontology editor:
  - Protégé 3.2.1
    http://protege.stanford.edu/download/release/full/
- Bean generator:



  - Jadex Beanynizer 0.941
    http://vsis-www.informatik.uni-hamburg.de/projects/jadex/download.php

    or
  - Acklin's Ontology Bean Generator
    http://protege.cim3.net/cgi-bin/wiki.pl?OntologyBeanGenerator

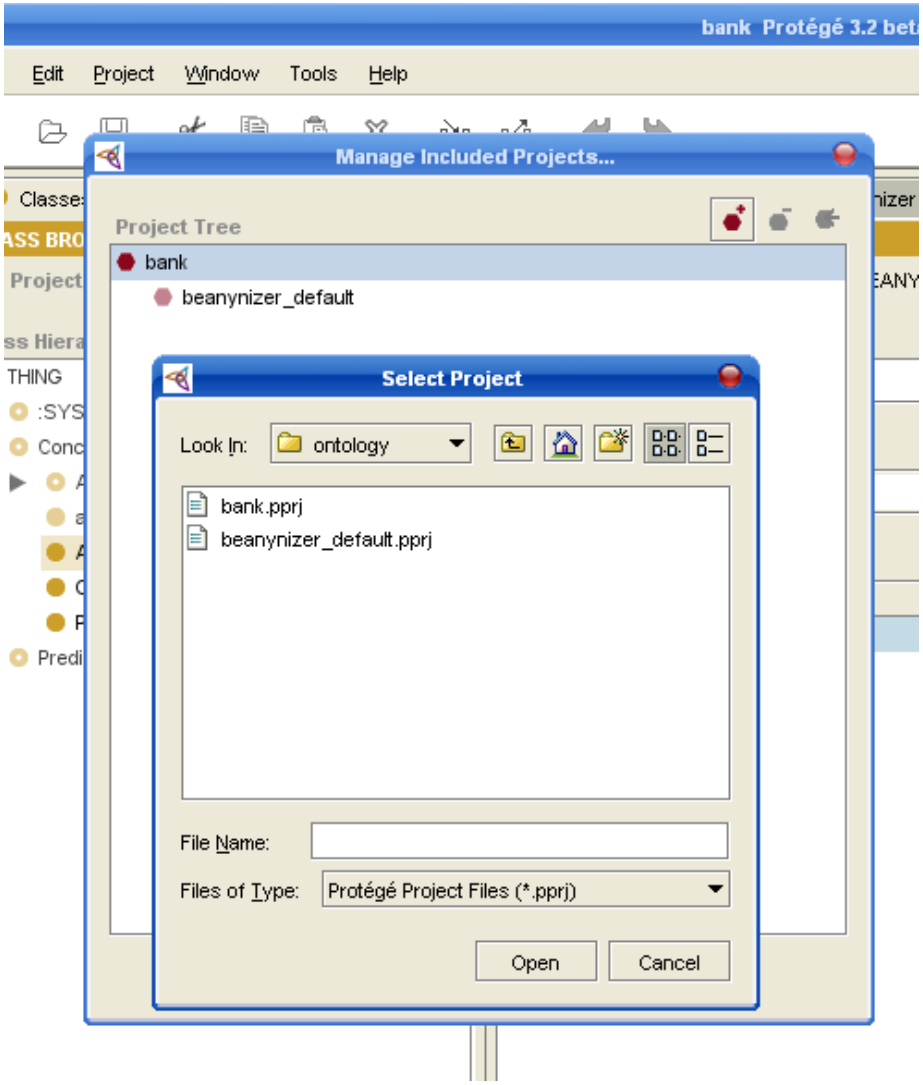
Creating an Ontology (1)

- Create a new ontology e.g. with the Project --> New... menu item.
- Note that Beanynizer currently does not support OWL, so you have to choose a standard or RDF ontology format.
- Save the new ontology to a directory of your choice.

Creating an Ontology (2a)

- Include one of the Beanynizer default ontologies (beanynizer_default.pprj)
- Use the Project -->Manage Included Projects...
- Click icon with + select the beanynizer_default.pprj file in the appearing file chooser
- You should copy the beanynizer default files (.pprj, .pins, .pont) to the directory of your ontology, and include the ontology from there. In this case Protégé will use a relative path name.

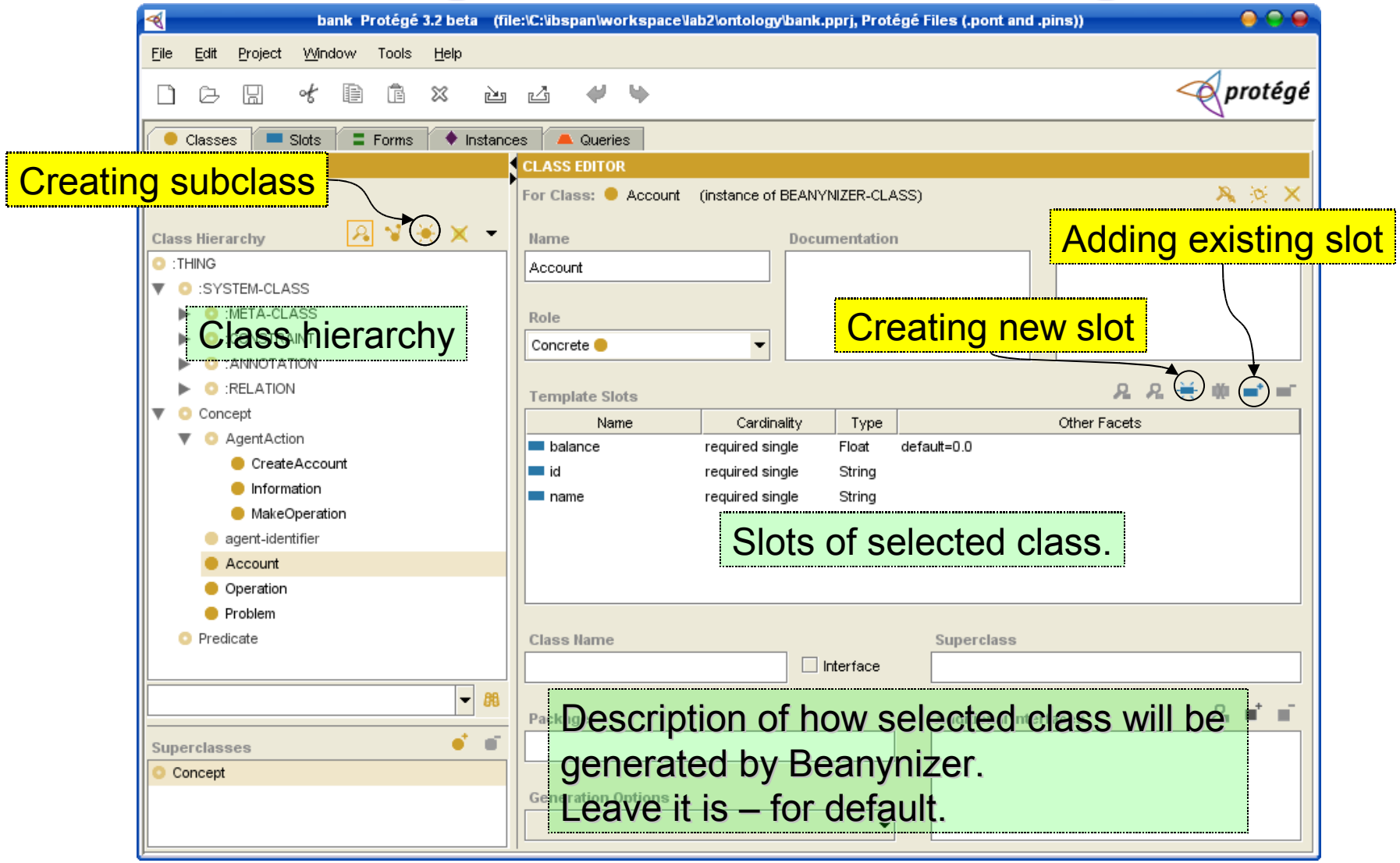
Creating an Ontology (2b)



Creating an Ontology (3a)

- Add classes and slots to your ontology.
- The JADE default ontology provides four base classes (Concept, AgentAction, agent-identifier, Predicate) that you should use as superclasses for your own concepts.

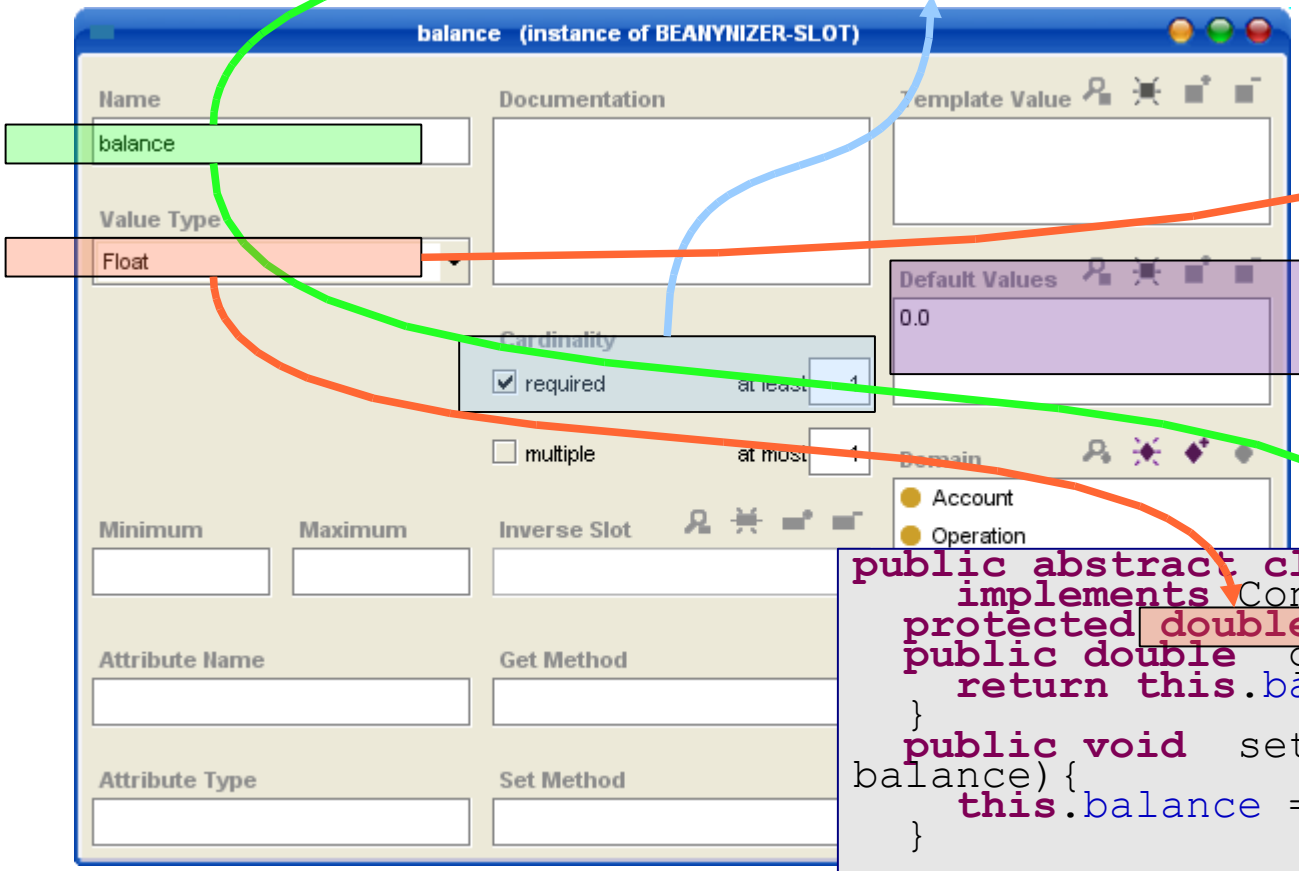
Creating an Ontology: defining class



Creating an Ontology: defining slot

ibspan.lab2.ex5.ontology.BankOntology

```
oAccountSchema.add(ACCOUNT_BALANCE,
    (PrimitiveSchema)getSchema(BasicOntology.FLOAT),
        ObjectSchema.MANDATORY);
```




```
public abstract class AccountData
    implements Concept {
  protected double balance = 0.0;
  public double getBalance() {
    return this.balance;
  }
  public void setBalance(double
balance){
    this.balance = balance;
  }
  ...
```

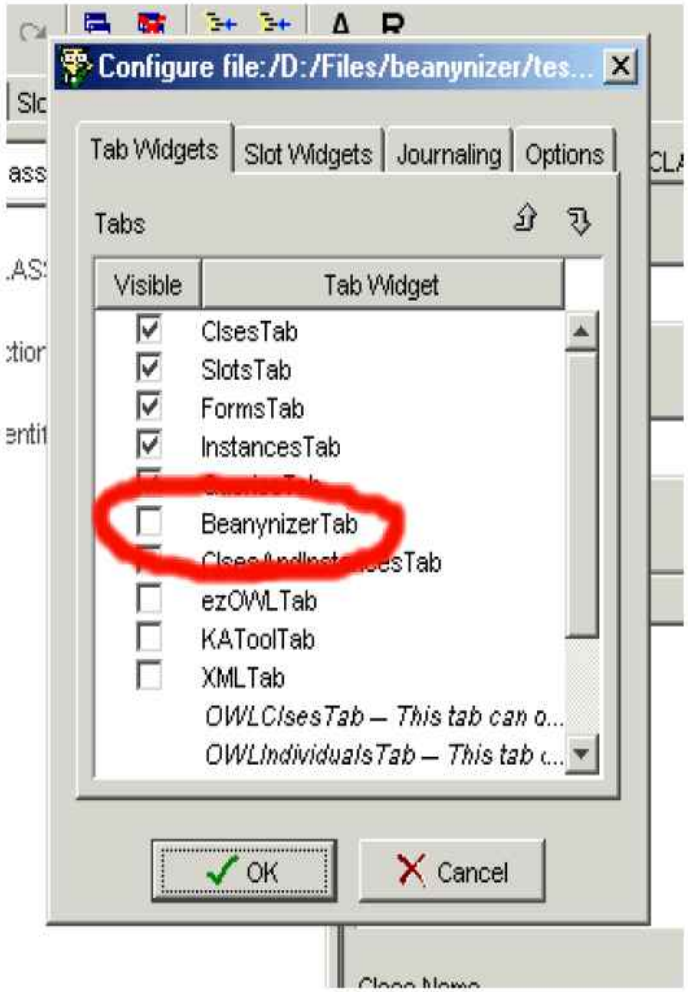
Slot to Java type mappings

Slot Type	Java Type
Float	`double`
Instance	a java class
Integer	`int`
String	`java.lang.String`
Symbol	`java.lang.String`

Creating an Ontology (4a)

- Select the Project --> Configure... menu and open the Tab Widgets tab
- Activate the Beanynizer tab and close the dialog by hitting Ok.

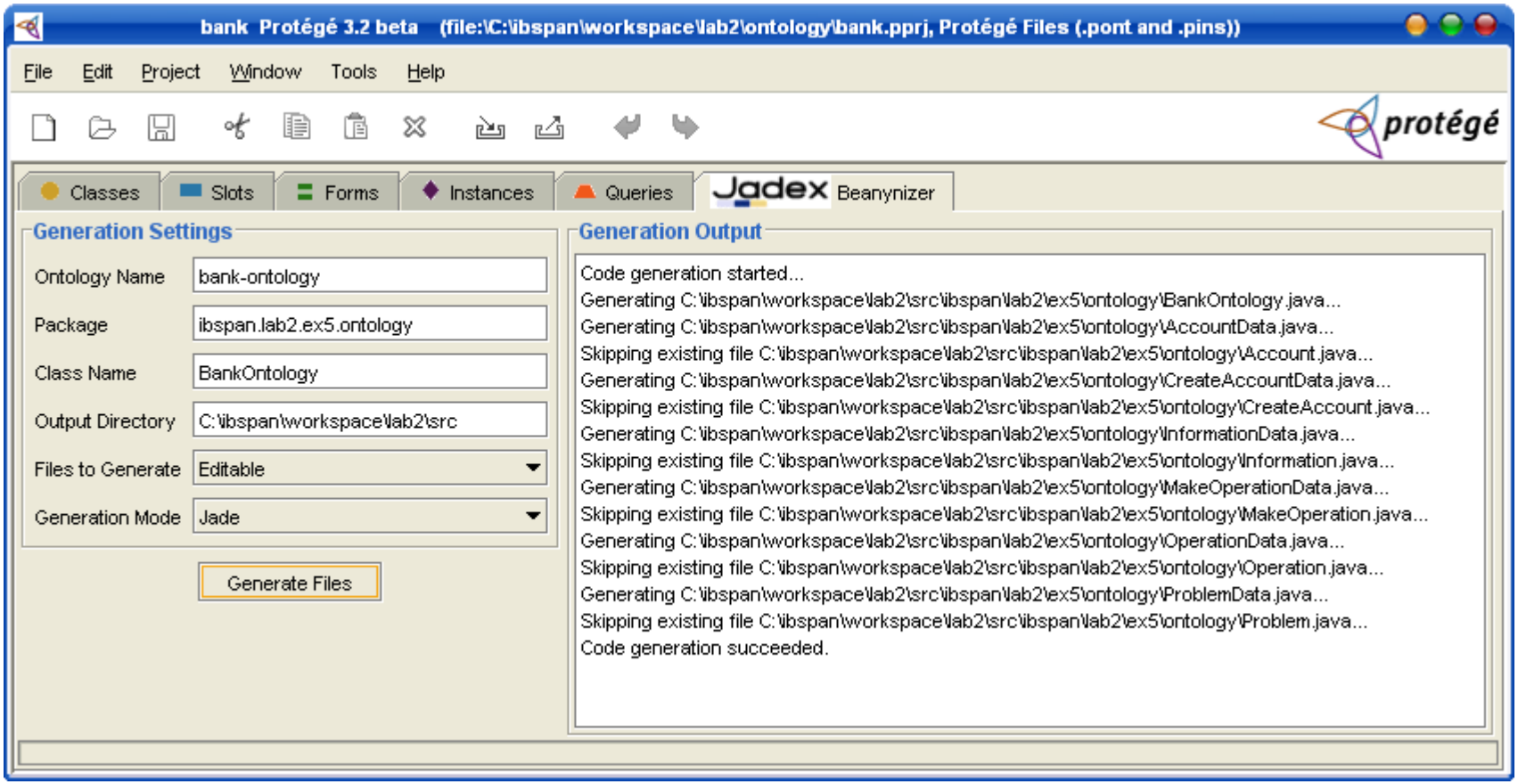
Creating an Ontology (4b)



Creating an Ontology (5a)

- In the Beanynizer Tab you can now edit the code generation options such as packagename and output directory.
- Depending on the base ontology you used, you also have to select the correct Generation Mode (Java for a pure beans ontology, Jade for a JADE ontology).
- Pressing the Generate Files button will create the desired source files.

Creating an Ontology (5b)

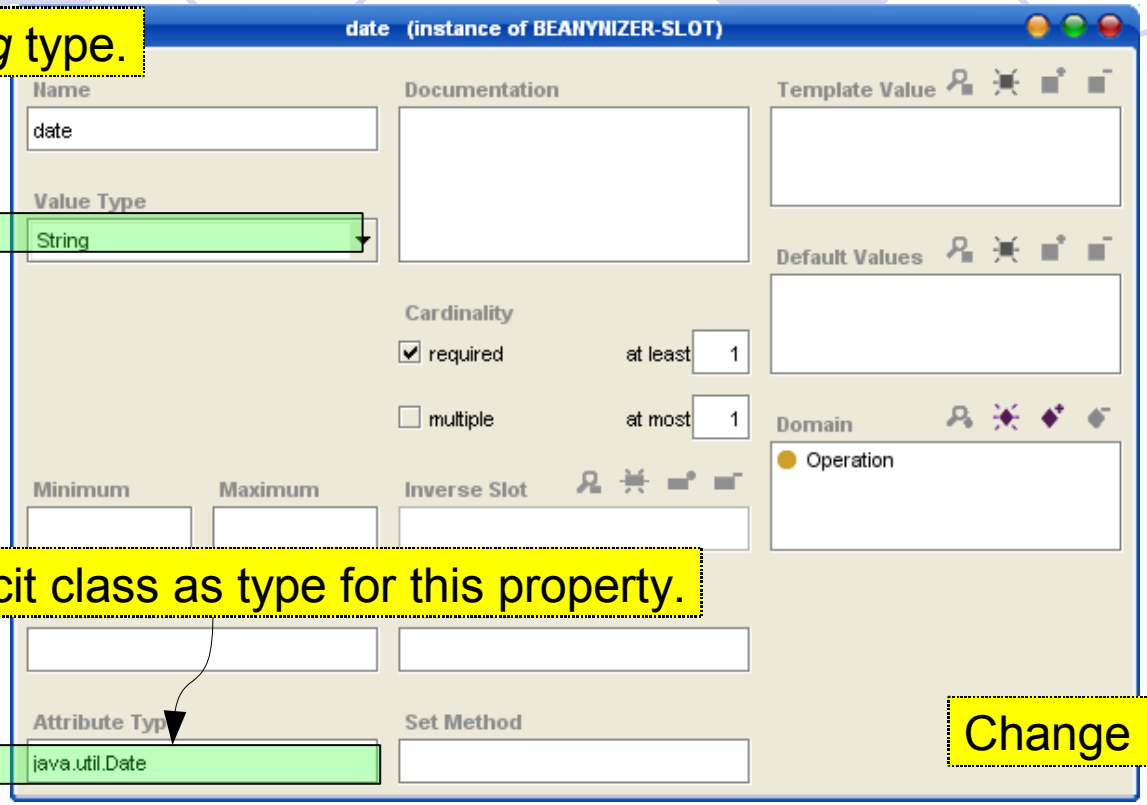


Manual corrections (1)

- Protégé has no primitive type for *date* type
- On the next slide we present the workaround for this case.

Manual corrections (2)

Pretend *String* type.



Define explicit class as type for this property.

Change in BankOntology.

```
oOperationSchema.add(OPERATION_DATE,
  (PrimitiveSchema)getSchema(BasicOntology.STRING),
  ObjectSchema.MANDATORY);
```

```
oOperationSchema.add(OPERATION_DATE,
  (PrimitiveSchema)getSchema(BasicOntology.DATE),
  ObjectSchema.MANDATORY);
```

Manual corrections (3)

- Generating files as *Editable* (which is the *default*) creates two files for any ontology class:
  - A **`classnameData.java`** file, which contains therequired fields and getter/setter methods,
  - and a **`classname.java`** file, which extends the data file, but is more or less empty.
- While the *data file* is overwritten each time you newly generate code from the ontology, the other file can be edited, because changes will be preserved.

Manual corrections (4)

- Example: **`ibspan.lab2.ex5.ontology.Operation`**
- Added:
  - New interface implements: **`BankVocabulary`**

```
public class Operation  extends OperationData
  implements Cloneable, BankVocabulary
```

  - **New method added:**

```
public String getName() {
    if (type == DEPOSIT) return "Depos.";
    if (type == WITHDRAWAL) return "Withd.";
    return "Admin.";
}
```

More documentation

- Protege
  - Tutorial: Ontology Development 101

    http://protege.stanford.edu/publications/ontology_development/ontology101.html
- Beanynizer
  - *Tools Guide for Jadex*

    http://prdownloads.sourceforge.net/jadex/toolguide-0.941.pdf?download
- Acklin's Ontology Bean Generator
  - Creating ontologies for JADE
    - JADE Tutorial: Application-defined Content Languages And Ontologies http://mia.ece.uic.edu/~papers/MediaBot/CLOntoSupport.pdf
    - *Building Multi-Agent Systems with JADE: Using ontologies*,

      http://www.iro.umontreal.ca/~vaucher/Agents/Jade/Ontologies.htm

More examples

- **`examples.ontology`** in JADE package