Xtext

Juan de Lara, Elena Gómez, Esther Guerra
{Juan.deLara, MariaElena.Gomez, Esther.Guerra}@uam.es
Computer Science Department
Universidad Autónoma de Madrid

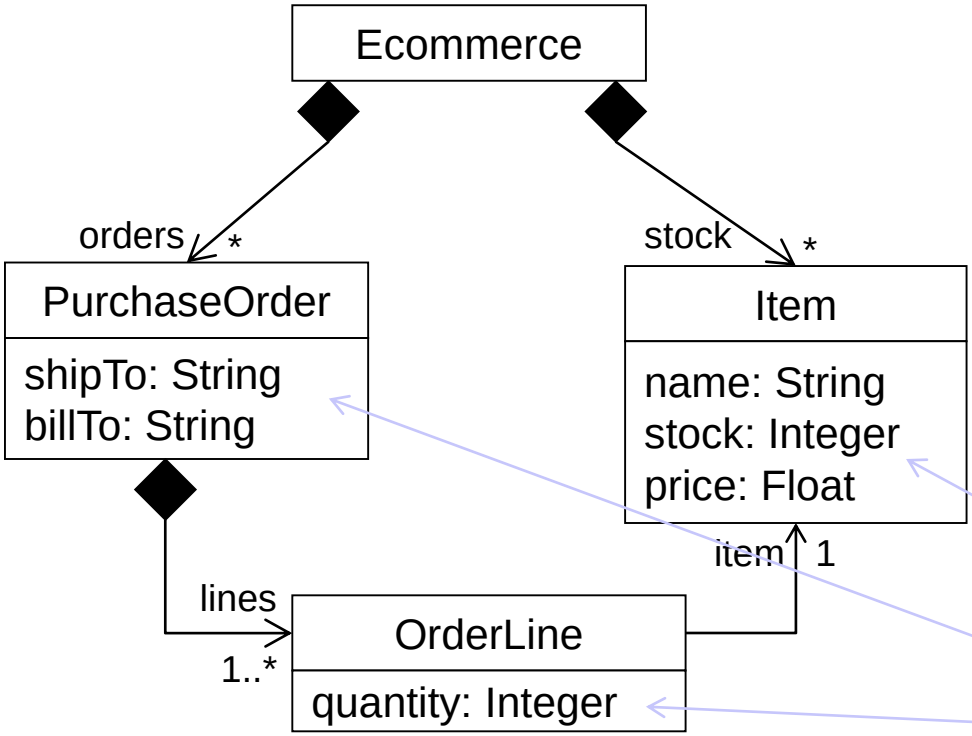
Masters: I2TIC and Formal methods

Xtext

- Framework for the development of textual editors for programming languages and domain-specific languages
- Editors are created as Eclipse plugins
- Editors can be generated from Ecore meta-models
- Integration with EMF
- The following language features can be customised:
  - language concrete syntax
  - syntax colouring
  - code completion and quick fixes
  - static analysis and validation of programs / models
  - code generation from programs / models

- Web page: http://www.eclipse.org/Xtext/
- Documentation and tutorials: http://www.eclipse.org/Xtext/documentation/

Example



DSL to define the stock of products of a company, and the purchase orders from its clients.

Goal → build editor with the following concrete syntax for models:

```
Stock:
  100 book (20.0 each)
  150 cd   (10.0 each)
Orders:
  { 3 book, 2 cd } to Mary paying John
  { 2 cd } to Mary paying Mary
```

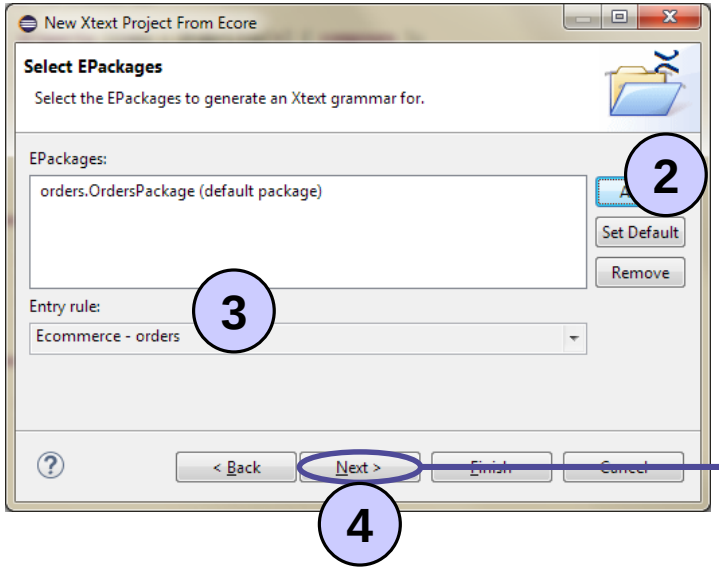
(we start from an ecore meta-model, from which we have generated the domain Java classes)

Building an editor with Xtext

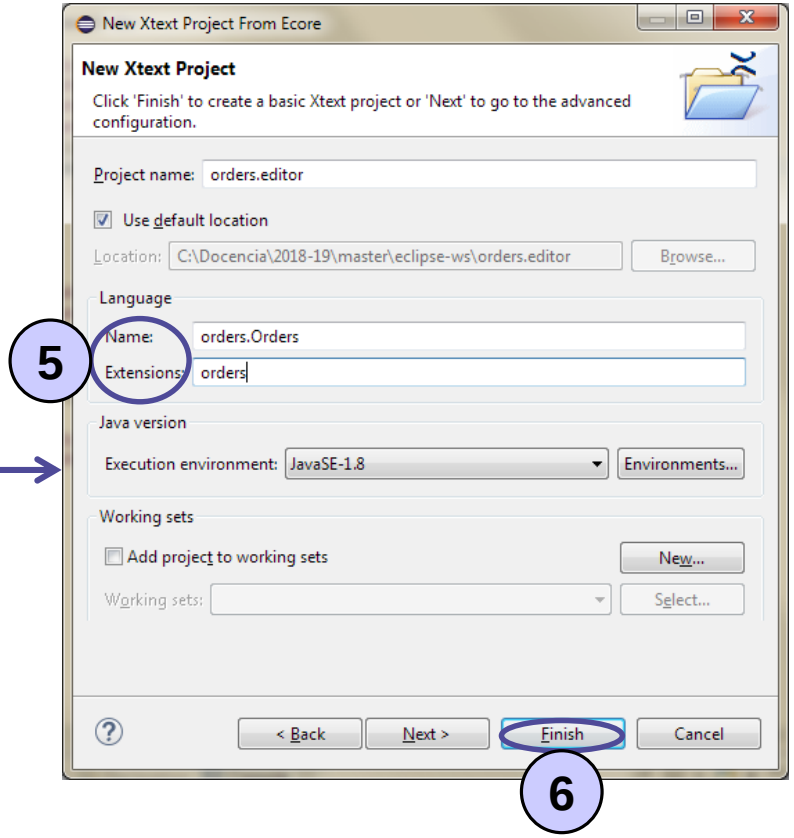
Create project from ecore meta-model

1 *File / New / Project… / Xtext / Xtext Project From Existing Ecore Models*

Select genmodel file.
Select root class as *Entry Rule.*



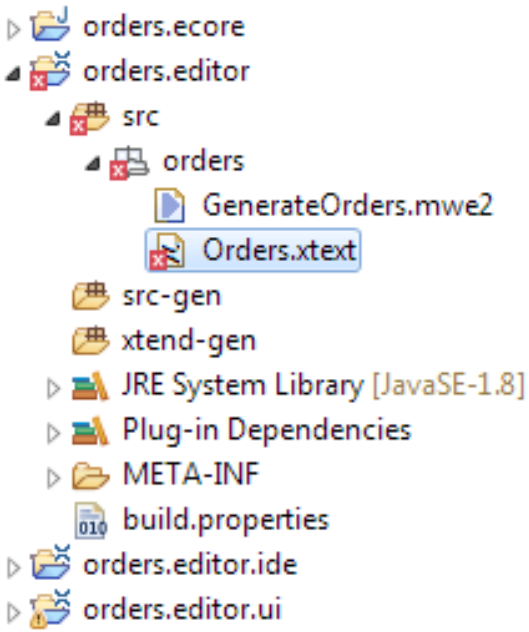
Select language name and
extension for language files.



Building an ed

Language gramma

A grammar is automatically created (xtext file in main project)




```
import "platform:/resource/orders/model/orders.ecore" // import "Orders"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

Ecommerce returns Ecommerce : {Ecommerce}
 'Ecommerce'
 '{'
   ('orders' '{' orders+=PurchaseOrder ( ","
                 orders+=PurchaseOrder)* '}' )?
   ('stock' '{' stock+=Item ( "," stock+=Item)* '}' )?
 '}';

PurchaseOrder returns PurchaseOrder:
 'PurchaseOrder'
 '{'
   'shipTo' shipTo=EString
   'billTo' billTo=EString
   'lines' '{' lines+=OrderLine ( "," lines+=OrderLine)* '}'
 '}';

Item returns Item:
 'Item' name=EString
 '{'
   'stock' stock=EInt
   'price' price=EFloat
 '}';

OrderLine returns OrderLine:
 'OrderLine'
 '{'
   'quantity' quantity=EInt
   'item' item=[Item|EString]
 '}';

EString returns ecore::EString: STRING | ID;
EInt    returns ecore::EInt: '-'? INT;
EFloat  returns ecore::EFloat:
   '-'? INT? '.' INT (('E'|'e') '-'? INT)?;
```

Language grammar

our ecore meta-model
(and any other used meta-model)

```
import "platform:/resource/orders/model/orders.ecore" // import "Orders"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

Ecommerce returns Ecommerce : {Ecommerce}
 'Ecommerce'
 '{'
   ('orders' '{' orders+=PurchaseOrder ( ","
                 orders+=PurchaseOrder)* '}' )?
   ('stock' '{' stock+=Item ( "," stock+=Item)* '}' )?
 '}';

PurchaseOrder returns PurchaseOrder:
 'PurchaseOrder'
 '{'
   'shipTo' shipTo=EString
   'billTo' billTo=EString
   'lines' '{' lines+=OrderLine ( "," lines+=OrderLine)* '}'
 '}';

Item returns Item:
 'Item' name=EString
 '{'
   'stock' stock=EInt
   'price' price=EFloat
 '}';

OrderLine returns OrderLine:
 'OrderLine'
 '{'
   'quantity' quantity=EInt
   'item' item=[Item|EString]
 '}';

EString returns ecore::EString: STRING | ID;
EInt    returns ecore::EInt: '-'? INT;
EFloat  returns ecore::EFloat:
   '-'? INT? '.' INT (('E'|'e') '-'? INT)?;
```

rule “invocation” (derived from containment ref.)

a grammar rule per class:
`<rule-name> returns <class-name>: … ;`

attributes specified by their name and value

rule “reference” (derived from non-containment ref.)

the vertical line means ‘OR’
STRING | ID

& means ‘AND’ with arbitrary order
STRING & ID

STRING requires quotation marks, ID does not

Building an editor with Xtext

Language grammar

Keywords

Words in single quotes, like `'Ecommerce'` or `'{'`.

Cardinality of expressions

	it must appear once
?	it must appear 0 or once, e.g. `('price' price=EFloat)?`
*	it must appear 0 or more times, e.g. `(',' lines+=OrderLine)*`
+	it must appear 1 or more times

Assignment of values to features

=	mono-valued features, e.g. `name=EString`
+=	multi-valued features, e.g. `lines+=OrderLine`
?=	boolean features; if the expression to the right is found, the feature is assigned the value *true*, e.g. (`isFree ?= 'free'`)?

Building an editor with Xtext

Language grammar

Cross-references

```
Item returns Item:
 'Item' name=EString
 '{'
   'stock' stock=EInt
   'price' price=EFloat
 '}';

OrderLine returns OrderLine:
 'OrderLine'
 '{'
   'quantity' quantity=EInt
   'item' item=[Item|EString]
  '}';
```

For non-containment references (e.g. item), their type must define an attribute called **name**. This attribute will be used to resolve the reference.

Example:

```
Item book { stock 100 price 20.0 }
OrderLine { quantity 2 item book }
```

Building an editor with Xtext

Language grammar

Rules for enumerate types

```
enum Visibility :
  PUBLIC='public' | PRIVATE='private' | PROTECTED='protected' ;

Attribute returns Attribute :
  visibility=Visibility typeName=('int' | 'string') name=ID ;
```

Example of use: `public int attributeName`

Building an editor with Xtext

Generate textual editor

- Generate language infrastructure (parser, serializer…)
  - Right-click on xtext file, *Run As -> Generate Xtext Artifacts*

Building an editor with Xtext

Execute textual editor

- start new Eclipse runtime instance
- create new project (of any kind)
- create new file. Use as file extension the one indicated when creating the Xtext project (.orders in this example)

Building an editor with Xtext

Generated editor

myorder.orders

```
Ecommerce {
    orders {
        PurchaseOrder {
            shipTo Mary
            billTo John
            lines {
                OrderLine {
                    quantity 3
                    item book
                },
                OrderLine {
                    quantity 2
                    item cd
                }
            }
        }
        // ...
    }
    stock {
        Item book {
            stock 100
            price 20.0
        },
        Item cd {
            stock 150
            price 10.0
        }
    }
}
```

But we would like to have:

```
Stock:
  100 book (20.0 each)
  150 cd   (10.0 each)
Orders:
  { 3 book, 2 cd } to Mary paying John
  { 2 cd } to Mary paying Mary
```

We have to change the language grammar!

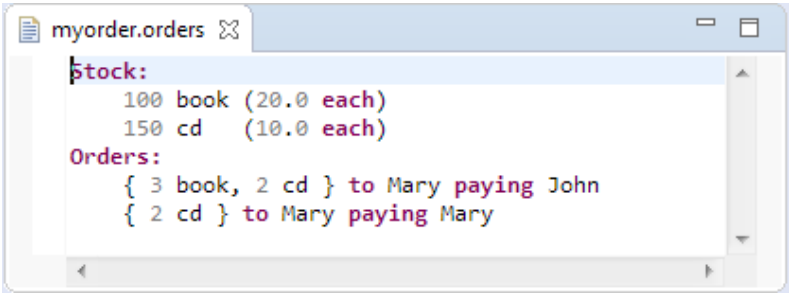
Building an editor with Xtext

Modifying the grammar…

New grammar:

```
Ecommerce returns Ecommerce : {Ecommerce}
  'Stock:'  (stock+=Item)+
  'Orders:' (orders+=PurchaseOrder)+;

PurchaseOrder returns PurchaseOrder:
  '{' lines+=OrderLine ( "," lines+=OrderLine)* '}'
  'to'     shipTo=EString
  'paying' billTo=EString;

Item returns Item:
  stock=EInt
  name=EString
  '(' price=EFloat 'each' ')';

OrderLine returns OrderLine:
  quantity=EInt item=[Item|EString];

EString returns ecore::EString: STRING | ID;
EInt    returns ecore::EInt: '-'? INT;
EFloat  returns ecore::EFloat:
  '-'? INT? '.' INT (('E'|'e') '-'? INT)?;
```

- Generate Xtext artifacts
- Start Eclipse runtime
- Create “orders” file
- New editor:



The Xtext project for the new grammar is available in *moodle*.

Building an editor with Xtext
Model validation

- Xtext editors validate models as they are being written
  - grammar and OCL meta-model constraints

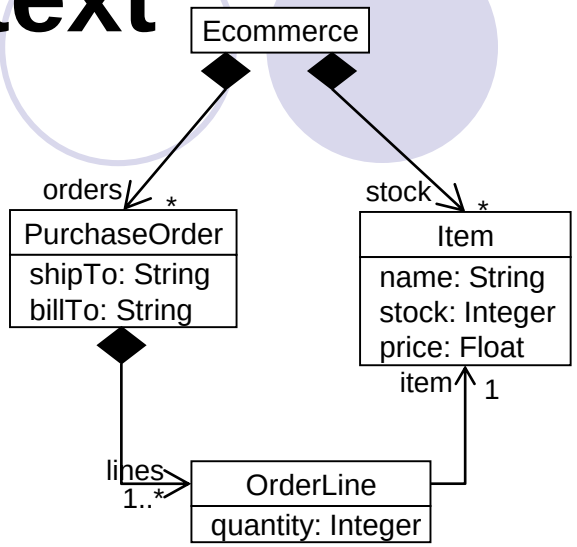
- Xtext generates file XXXValidator.java

```
package orders.validation;
public class OrdersValidator extends AbstractOrdersValidator {}
```

- In order to add new error and warning messages, add new methods annotated with @Check to this class, which should receive an object of the type to be validated

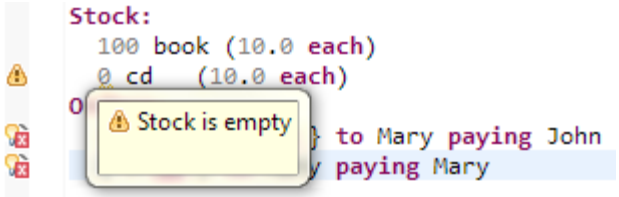
Building an editor with Xtext

Model validation - example



- show *warning* if stock is 0
- show *error* if stock is negative

```
package orders.validation;

import org.eclipse.xtext.validation.Check;
import orders.OrdersPackage;

public class OrdersValidator extends AbstractOrdersValidator {
  @Check
  public void stockBiggerThan0 (Item item) {
    if (item.getStock() == 0)
      warning("Stock is empty",
              OrdersPackage.Literals.ITEM__STOCK,
              "emptyStock");
  }
  @Check
  public void stockPositive (Item item) {
    if (item.getStock() < 0)
      error("Stock must be positive",
            OrdersPackage.Literals.ITEM__STOCK,
            "invalidStock");
  }
}
```



Building an editor with Xtext
Quick fixes

- For any defined validation, it is posible to define one or mores quick fixes

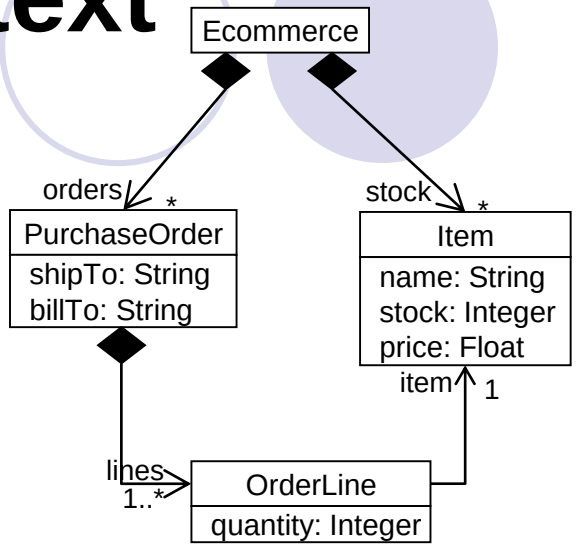
- Xtext generates file XXXQuickfixProvider.java (in folder src of project XXX.ui)

```
package orders.ui.quickfix;
public class OrdersQuickfixProvider extends DefaultQuickfixProvider {}
```

- In order to define a quick fix, add method annotated with @Fix to this class. The annotation should be configured with the identifier of the validation associated to the quick fix

Building an editor with Xtext

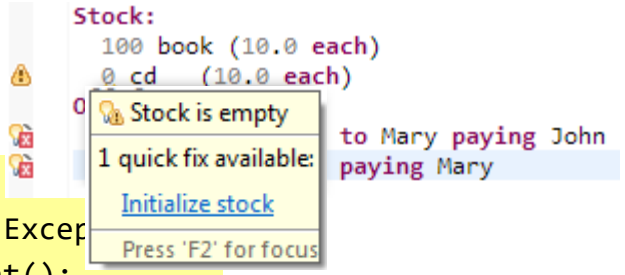
Quick fixes - example



- set stock to 1, when it is empty

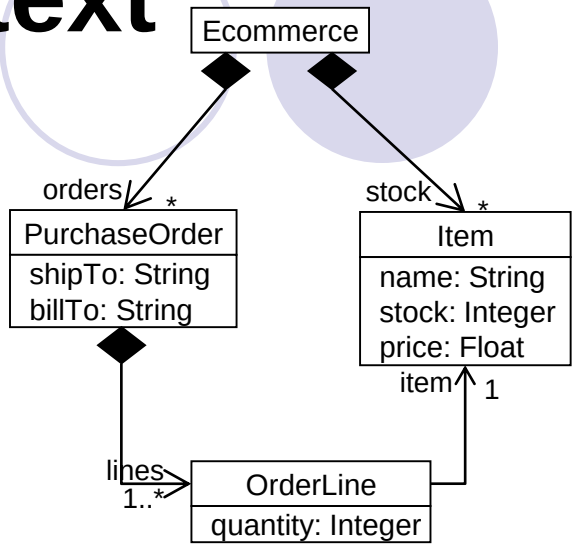
```
package orders.ui.quickfix;
import org.eclipse.xtext.ui.editor.model.IXtextDocument;
import org.eclipse.xtext.ui.editor.model.edit.*;
import org.eclipse.xtext.ui.editor.quickfix.*;
import org.eclipse.xtext.validation.Issue;

public class OrdersQuickfixProvider extends DefaultQuickfixProvider {
  @Fix("emptyStock")
  public void populateStock (final Issue issue, IssueResolutionAcceptor acceptor) {
    acceptor.accept(issue,
      "Initialize stock",       // label
      "Stock is set to 1.",     // description
      "icon.png",               // icon
      new IModification() {
        public void apply (IModificationContext context) throws Excep
          IXtextDocument xtextDocument = context.getXtextDocument();
          xtextDocument.replace(
            issue.getOffset(),  // text offset of problem
            issue.getLength(),  // number of characters to change
            "1");               // new text
      }});}}
```

text manipulation



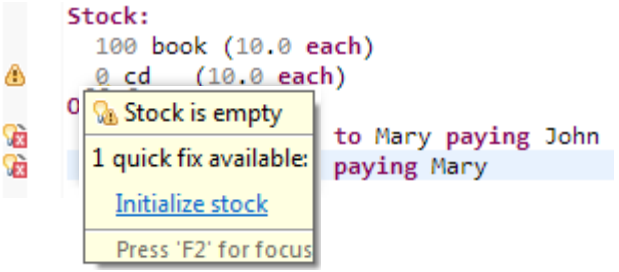
Building an editor with Xtext

Quick fixes - example



- set stock to 1, when it is empty

```
package orders.ui.quickfix
import org.eclipse.ecore.EObject;
import org.eclipse.xtext.ui.editor.model.edit.*;
import org.eclipse.xtext.ui.editor.quickfix.*;
import org.eclipse.xtext.validation.Issue;

public class OrdersQuickfixProvider extends DefaultQuickfixProvider {
  @Fix("emptyStock")
  public void populateStock (final Issue issue, IssueResolutionAcceptor acceptor) {
    acceptor.accept(issue,
      "Initialize stock",       // label
      "Stock is set to 1.",     // description
      "icon.png",               // icon
      (EObject element, IModificationContext context) -> {
        ((Item)element).setStock(1);
      }
    );
  }
}
```



model manipulation

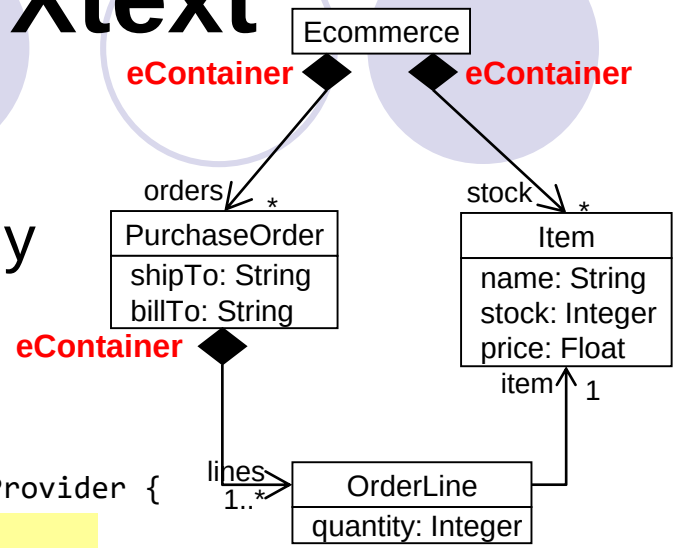
Building an editor with Xtext
Scope provider

- Xtext editors have a content assistant (Ctrl + Space)
- For cross-references, it suggests all objects that can be assigned to the reference (all objects of the reference type)
  - validators are ignored
  - meta-model constraints are ignored

- Xtext generates the file XXXScopeProvider.java
- In order to change the default behaviour of the scope provider, override the method `getScope(EObject, EReference)`

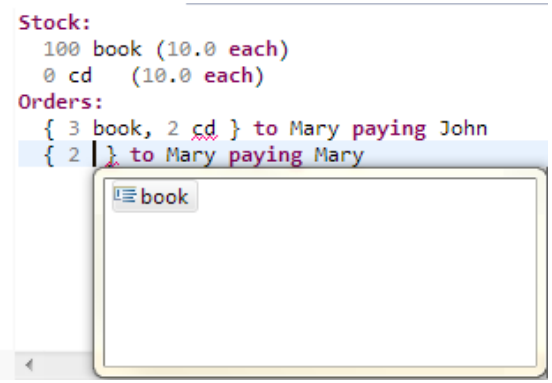
Building an editor with Xtext

Scope provider - example



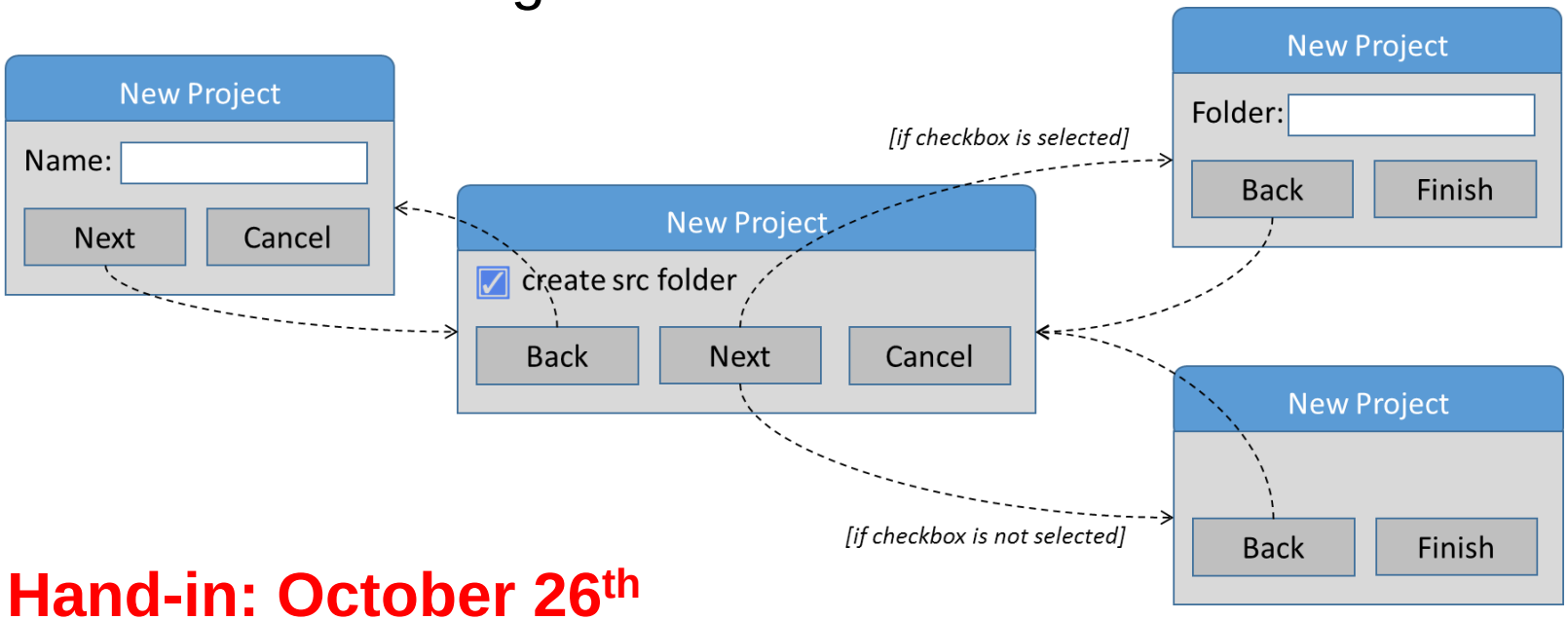
- When creating an order line, show only the items with stock bigger than 0

```
package orders.scoping;

public class OrdersScopeProvider extends AbstractOrdersScopeProvider {
  @Override
  public IScope getScope (EObject context, EReference reference) {
    // if it is the reference to constrain...
    if (context instanceof OrderLine &&
        reference == OrdersPackage.Literals.ORDER_LINE__ITEM) {
      // get root object (Ecommerce)
      Ecommerce ecommerce = (Ecommerce)EcoreUtil2.getRootContainer(context);
      // get items in stock (i.e., with stock > 0)
      List<Item> items = new ArrayList<>();
      for (Item item : ecommerce.getStock())
        if (item.getStock() > 0)
          items.add(item);
      // return items in stock
      return Scopes.scopeFor(items);
    }
    else return super.getScope(context, reference);
  }}
```



Exercise

- Build an EMF meta-model for defining wizards (the detailed list of requirements is published in moodle)
- Build the corresponding tree-like editor
- Build a model using the tree-like editor
- Build a textual editor with xtext
- Build a model using the xtext-editor



Hand-in: October 26th

Bibliography

- Página web de Xtext: http://www.eclipse.org/Xtext
- Página web de Xtend: https://www.eclipse.org/xtend/

- Desarrollo de Software Dirigido por Modelos: Conceptos , métodos y herramientas. Capítulo 8. Jesús García Molina, Félix O. García Rubio, Vicente Pelechano, Antonio Vallecillo, Juan Manuel Vara y Cristina Vicente-Chicote. Ra-Ma (2013)

- Model-Driven Software Engineering in Practice (Synthesis Lectures on Software Engineering). Marco Brambilla, Jordi Cabot & Manuel Wimmer. Morgan & Claypool Publishers, 1st Edition (2012)

- Implementing Domain-Specific Languages with Xtext and Xtend. Lorenzo Bettini. Packt Publishing (2013)