OOA/OOD/OOP Example

See **`http://www.cs.toronto.edu/~matz/instruct/csc407/eg`**

Introduction

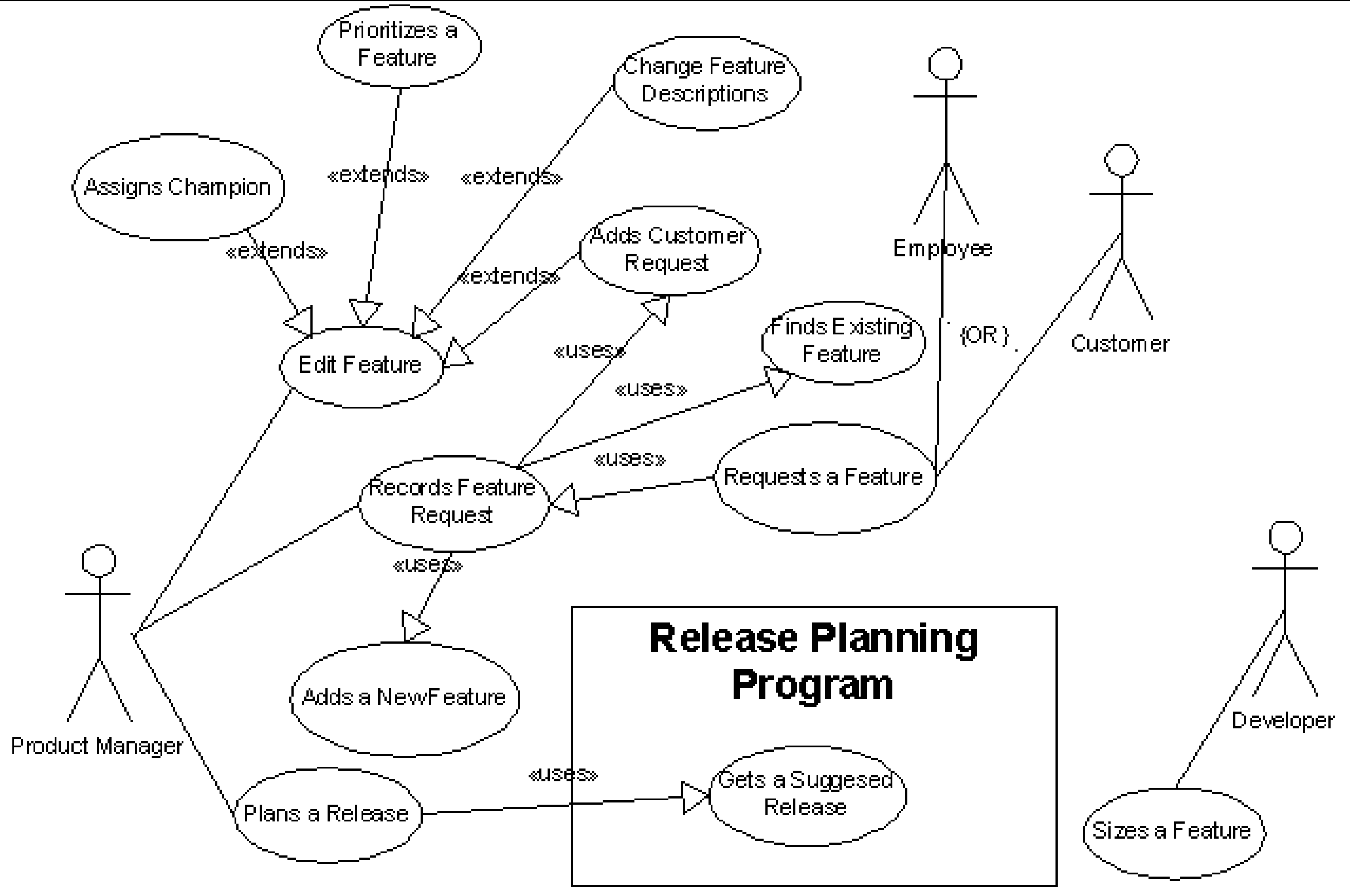
- This was David Penny’s research topic.
- Want a (Java) program to help a software company plan new releases of their software (340 refers to person-days):

```
$ java Plan features.xml Planetaria 340
```

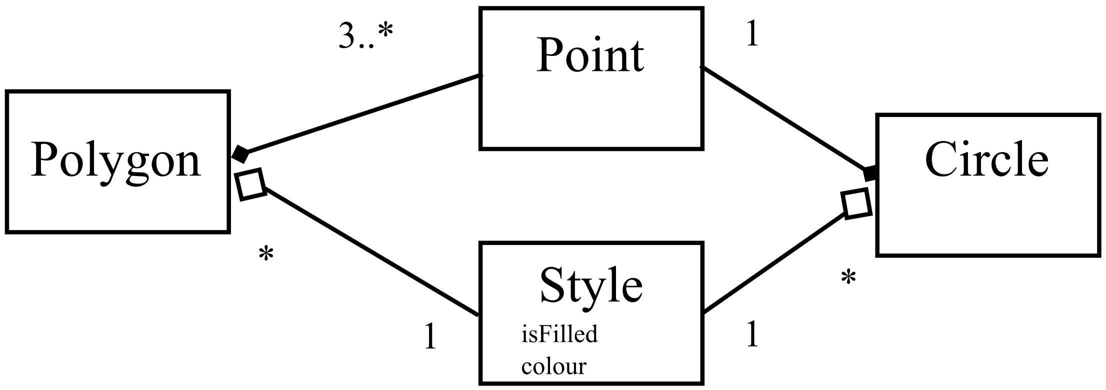
- xml file contains sized (in coder days), prioritized (hi,med,low), feature requests for various products
  - includes list of requesting customers with how much they want it (1-10).
- Suggest an "optimal" release plan given the available capacity (in coder days).
- Sample output

OOA

- See `~matz/csc407/eg/ooa/index.html`
- Introduction
  - why are we doing this
  - what is the current document for
  - where did the information come from
  - general points (change & XML file in this case)
- Use Cases
  - what is the bigger problem
  - how does this particular program fit into it
- Class Diagrams
  - restate information from the requirements statement in UML
  - (mostly you have no "requirements statement")
  -


Prioritizes a Feature
Change Feature Descriptions
Assigns Champion
«extends»
«extends»
«extends»
«extends»
Adds Customer Request
Edit Feature
Finds Existing Feature
«uses»
«uses»
«uses»
Employee
Customer
{OR}
Records Feature Request
Requests a Feature
«uses»
Product Manager
Adds a NewFeature
Release Planning Program
Plans a Release
«uses»
Gets a Suggesed Release
Developer
Sizes a Feature

multiplicity review



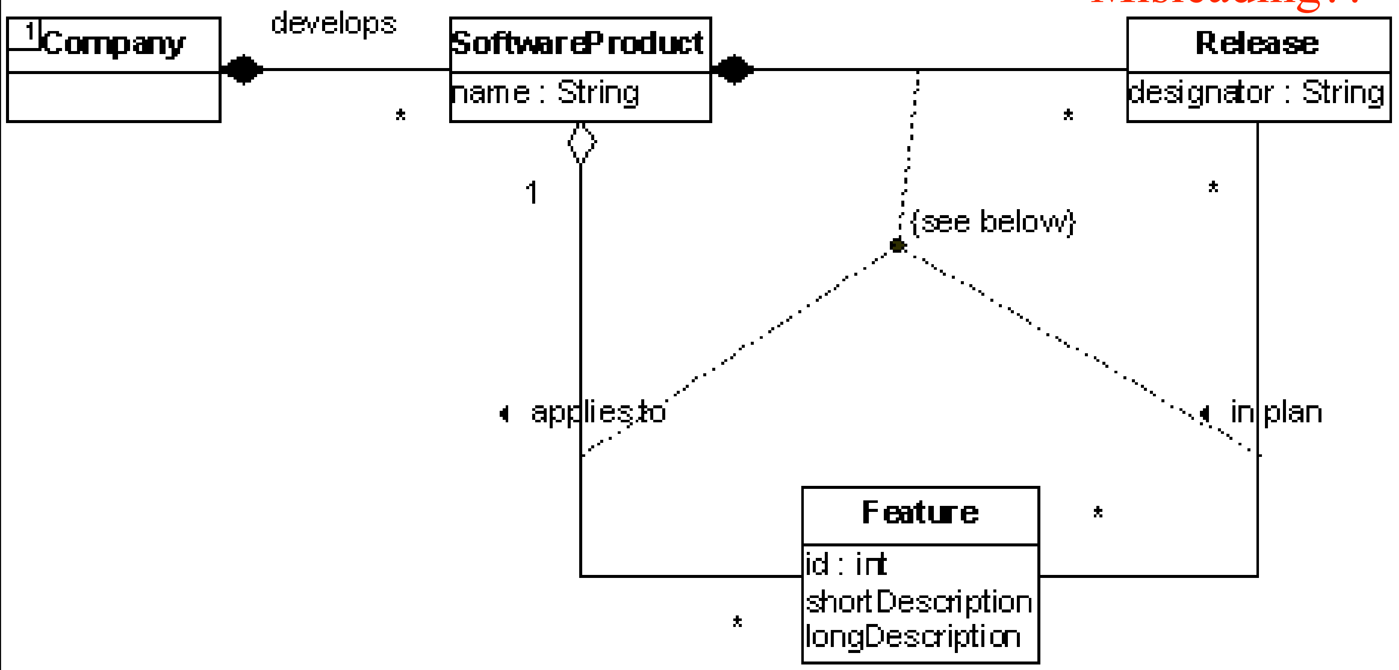
From Fowler, pp 86

Sometimes detailed multiplicity has important things to tell the reader about the domain model.

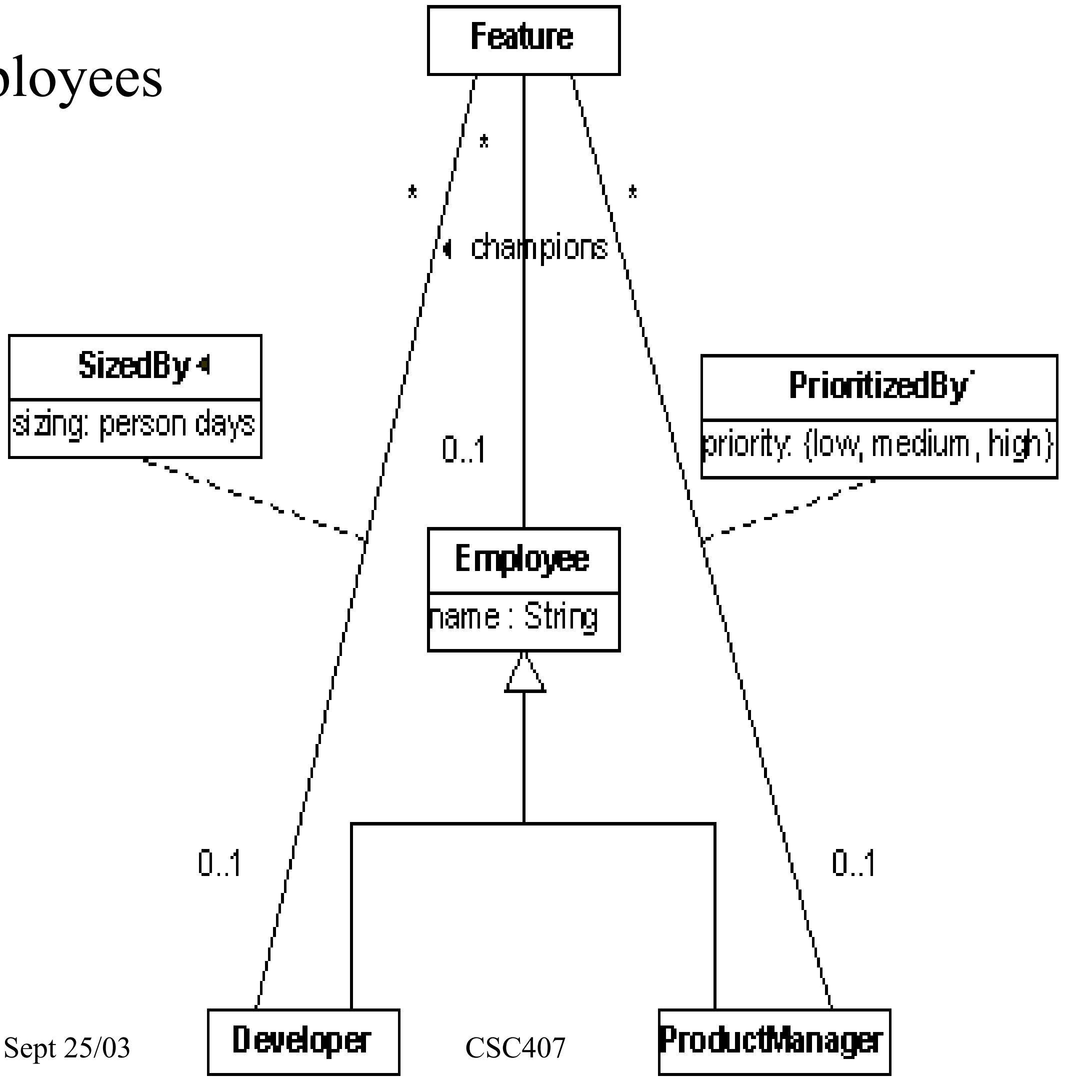
Often not..

Features

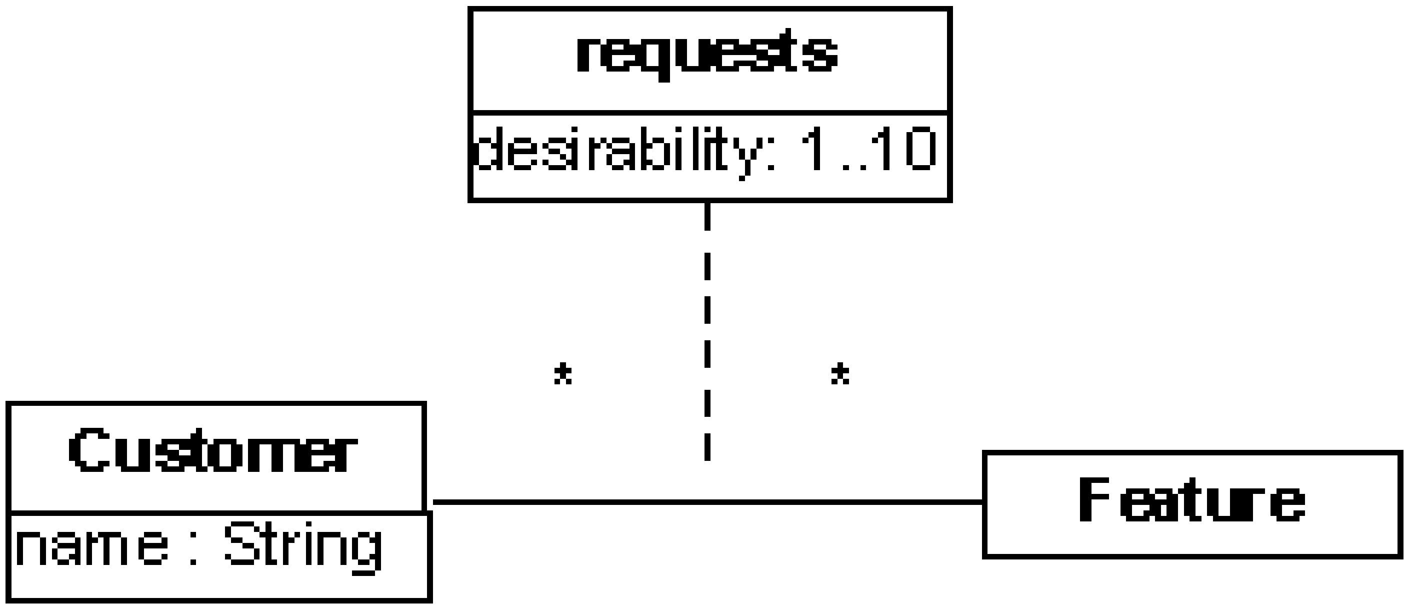
Misleading??



Employees


Feature
*
*
*
champions
SizedBy
sizing: person days
PrioritizedBy
priority: {low, medium, high}
0..1
Employee
name : String
0..1
0..1
Developer
ProductManager

Customers



OOD

- See ood document
  - David's presentation is excellent.
- Package design
  - what rationale for the package breakdown
- Main driver
  - sequence diagram explaining how (one) use case is executed
- For each package
  - a collection of class diagrams
    - shows important methods
    - shows important attributes
    - shows association navigability
    - indicates how associations are implemented
    - indicates inheritance and interface implementation

important = helps in understanding the design

About Source and Javadoc

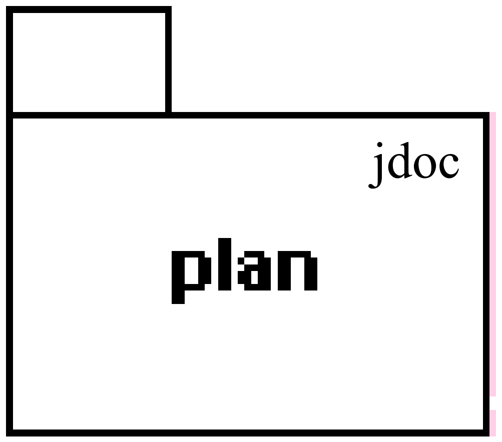
- Javadoc is a tool that extracts comments formatted in a certain manner and produces Web pages documenting the details of a class design.
  - See example
-
- To display source code, I used a tool called java2html for pretty-printing Java source to HTML.
  - See example

Experiments show..

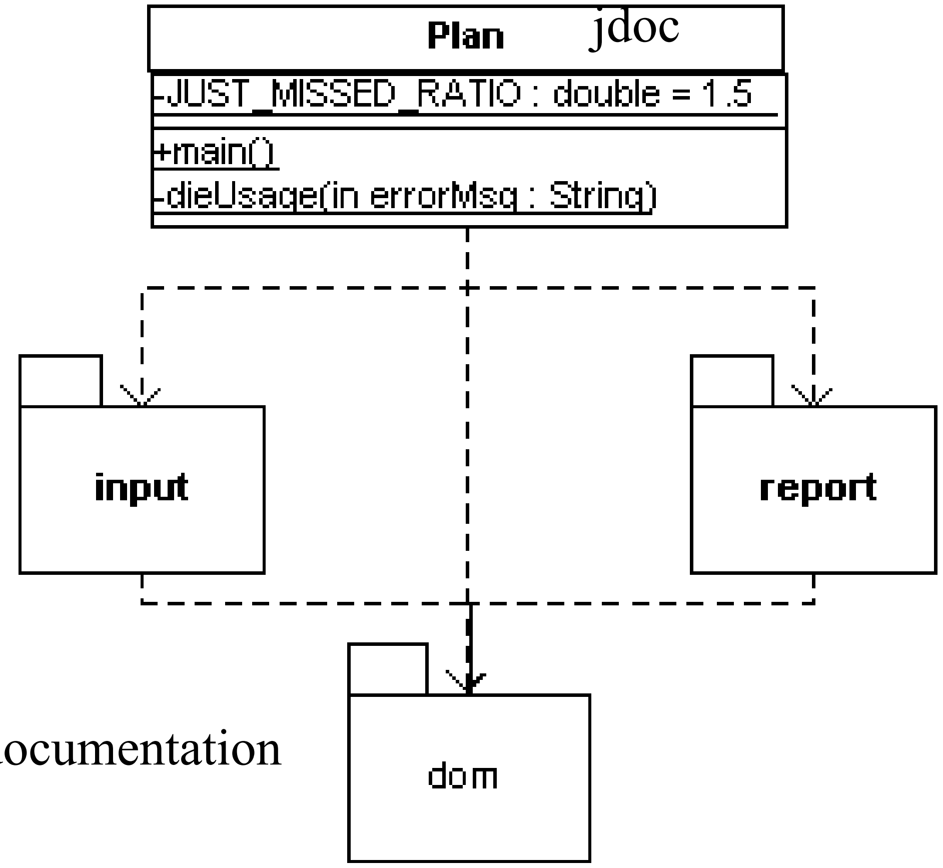
```
/** Suggests a release of this software product.
 *  @param capacity number of person-days of  effort available to work release
 *  @return a Release containing a suggested list of features
 */

public Release planRelease(double capacity) {
       double inplan = 0.0;
       //Sort in order of desirability somehow
       sortFeatures(ReverseFeaturePlanningOrder.get());
       Release r = new Release();
       for (Iterator i = featureIterator(); i.hasNext(); ) {
            Feature f = (Feature)i.next();
            if (inplan + f.getSizing() <= capacity) {
                 r.addFeature(f);
                 inplan += f.getSizing();
            }
       }
       return r;
  }
```

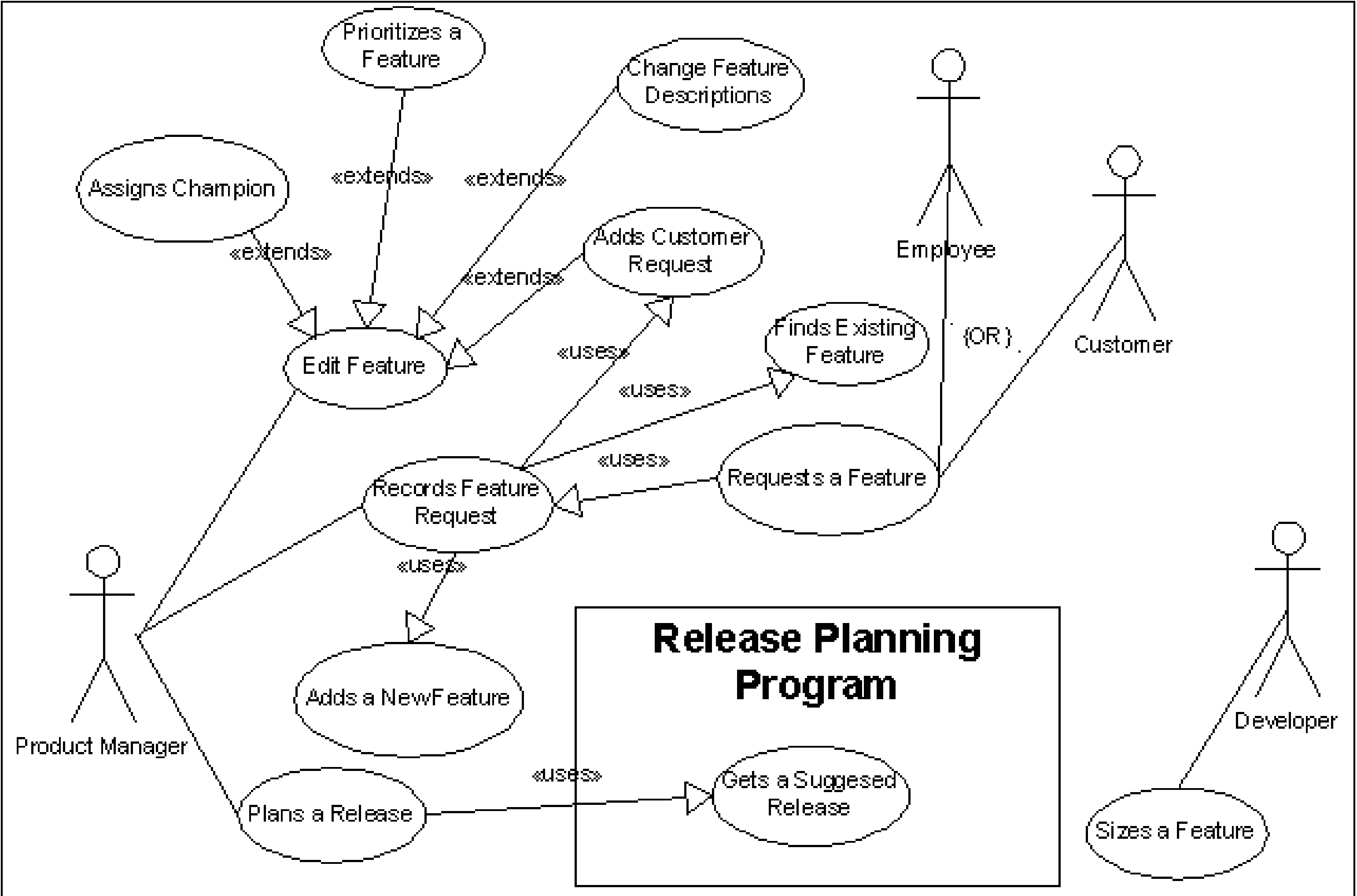
Top Package

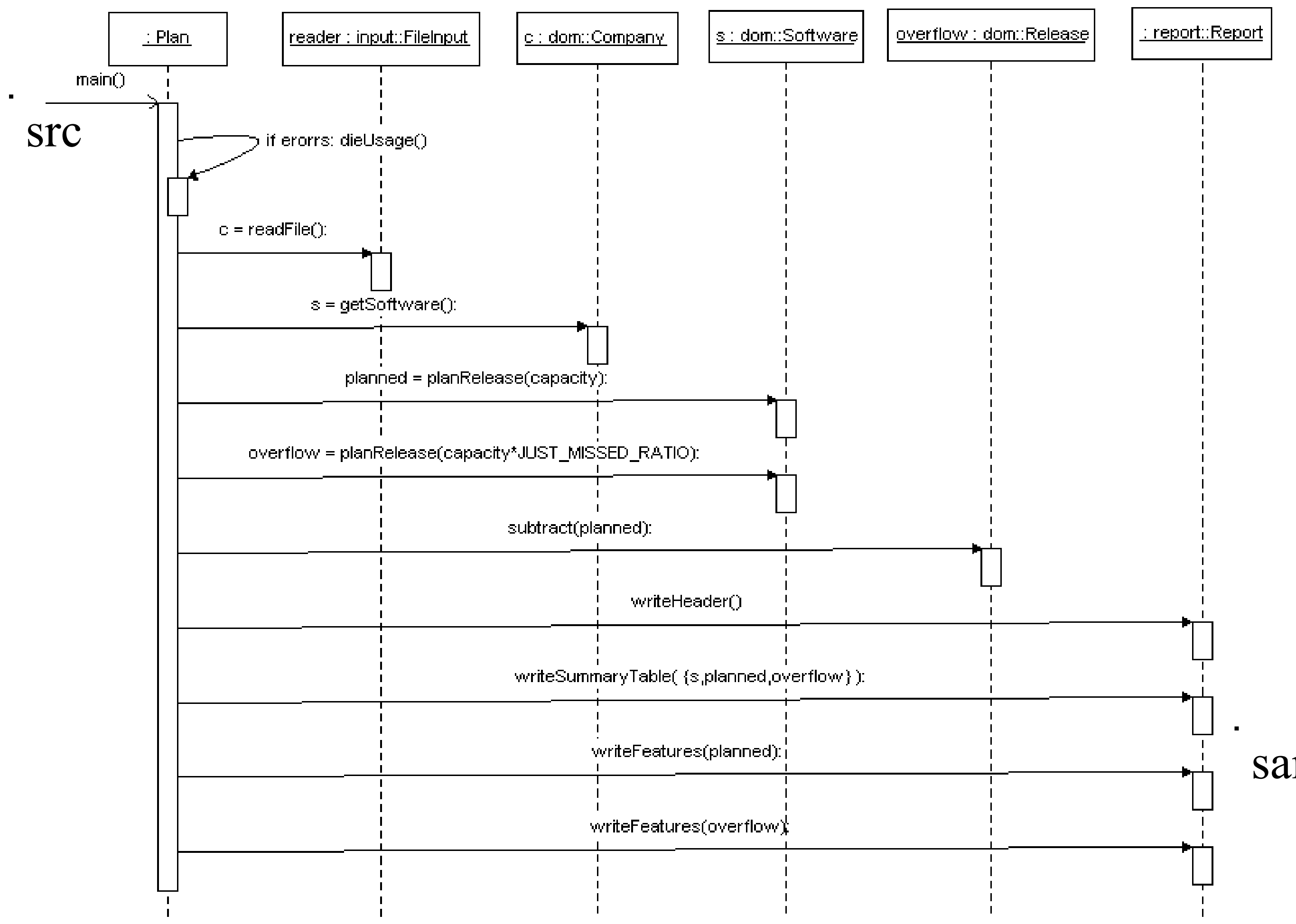


Package Plan


Plan
jdoc
-JUST_MISSED_RATIO : double = 1.5
+main()
-dieUsage(in errorMsg : String)
input
report
dom

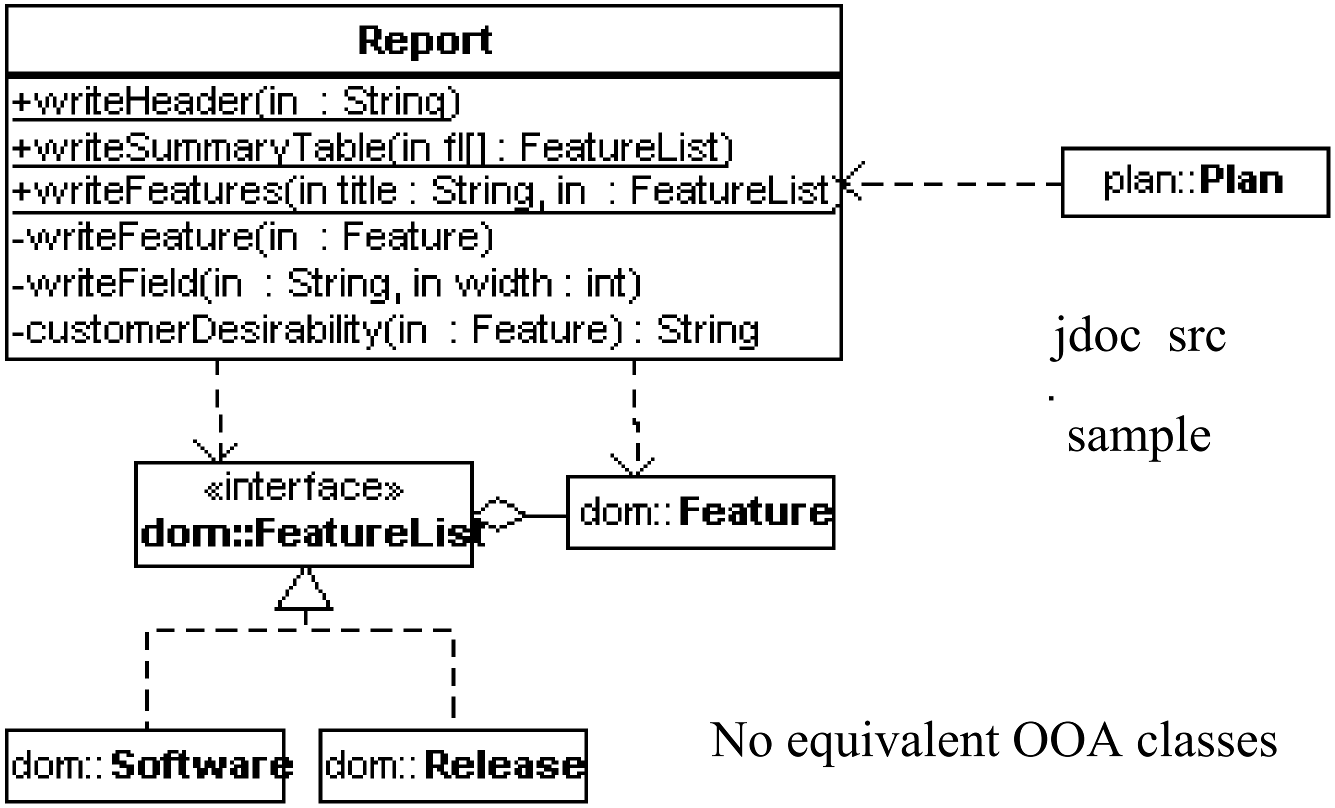
package documentation


Prioritizes a Feature
Change Feature Descriptions
Assigns Champion
«extends»
«extends»
«extends»
«extends»
Adds Customer Request
Employee
Customer
Edit Feature
Finds Existing Feature
«uses»
«uses»
{OR}
«uses»
Records Feature Request
Requests a Feature
«uses»
Release Planning Program
Adds a New Feature
Developer
Product Manager
«uses»
Plans a Release
Gets a Suggesed Release
Sizes a Feature



: Plan
reader : input::FileInput
c : dom::Company
s : dom::Software
overflow : dom::Release
: report::Report
main()
if erorrs: dieUsage()
c = readFile():
s = getSoftware():
planned = planRelease(capacity):
overflow = planRelease(capacity*JUST_MISSED_RATIO):
subtract(planned):
writeHeader()
writeSummaryTable({s,planned,overflow}):
writeFeatures(planned):
writeFeatures(overflow):
src
sample

Package Report


Report
+writeHeader(in : String)
+writeSummaryTable(in fl[] : FeatureList)
+writeFeatures(in title : String, in : FeatureList)
-writeFeature(in : Feature)
-writeField(in : String, in width : int)
-customerDesirability(in : Feature) : String
plan::Plan
«interface»
dom::FeatureList
dom::Feature
dom::Software
dom::Release

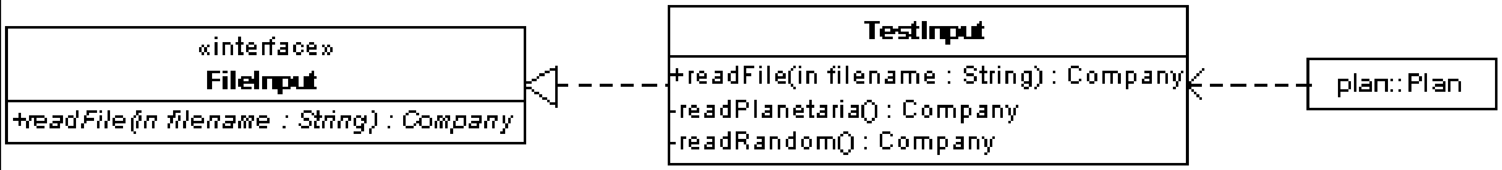
jdoc src

sample

No equivalent OOA classes

Package Input

jdoc src


«interface»
FileInput
+readFile(in filename : String) : Company
TestInput
+readFile(in filename : String) : Company
-readPlanetaria() : Company
-readRandom() : Company
plan::Plan

- No equivalent OOA classes
- Sequence diagram for readFile is fairly clear just from the class description (see also Report class)

Package dom

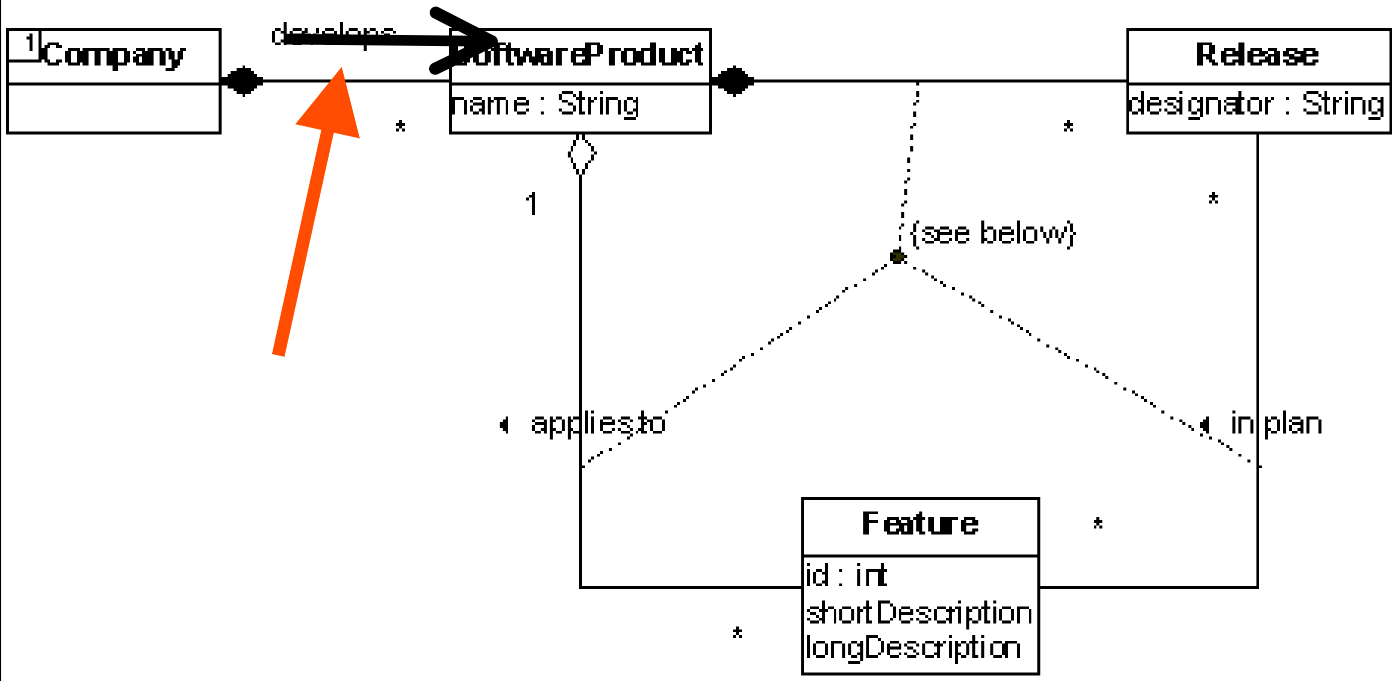
- For "Domain Object Model"
-
- Coad's "Problem Domain Component"
-
- Implements an in-memory, object-oriented data model reflecting the OOA
-
- Must be modified/extended to work in a program

jdoc

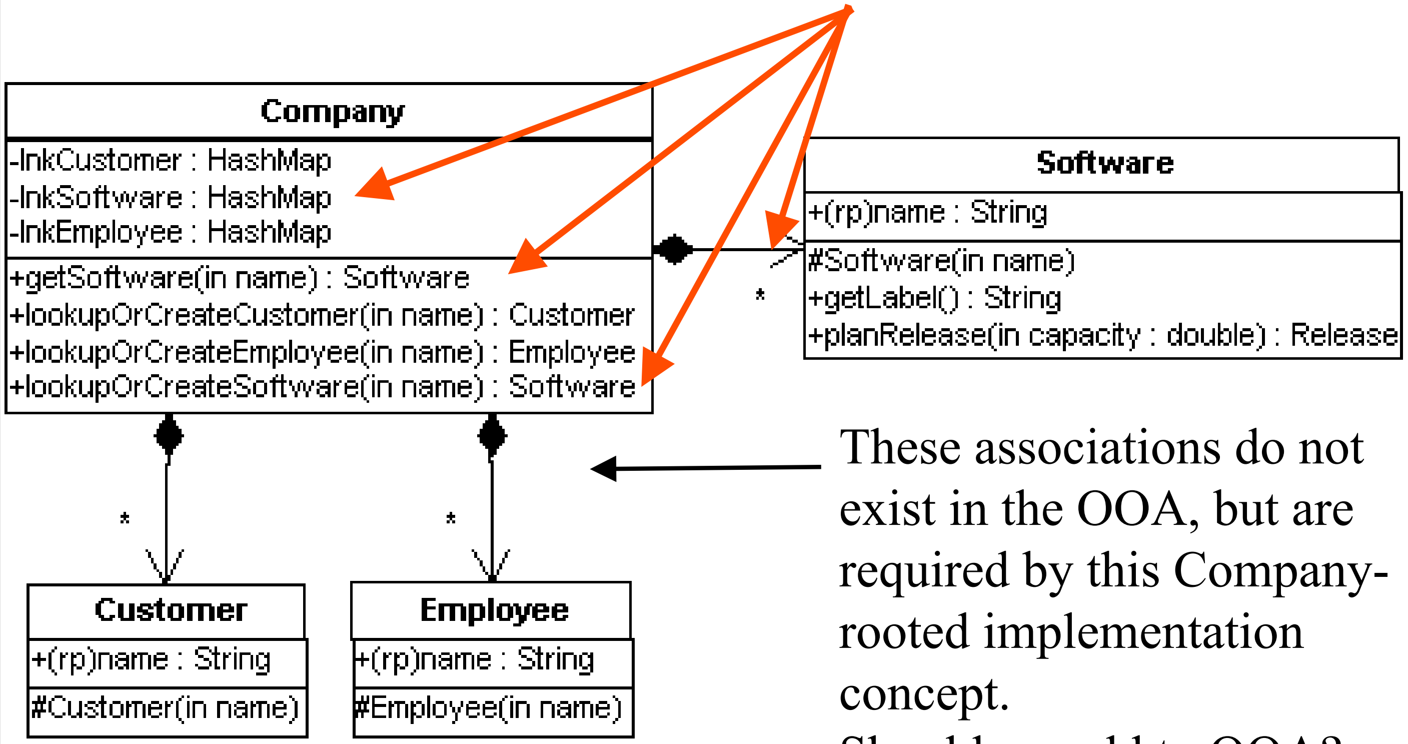
Implementing Associations

- Decide on navigability
  - The direction in which the association can be efficiently navigated
    - If you have one object of the Left class, can you in O(n) time access all objects of the Right class linked to that Left object.
- Decide on interface for
  - Navigating the links
    - usually get method for 1 side, iterator for * side.
  - Adding new links
  - Deleting links (if necessary)
- Decide on implementation
  - Simple pointer to implement the [0..1] side
    - (if required by navigatability)
  - Array, Vector, Map, Linked List to do the [*] side
    - (if required by navigatability)

Features (from OOA)



DOM Company

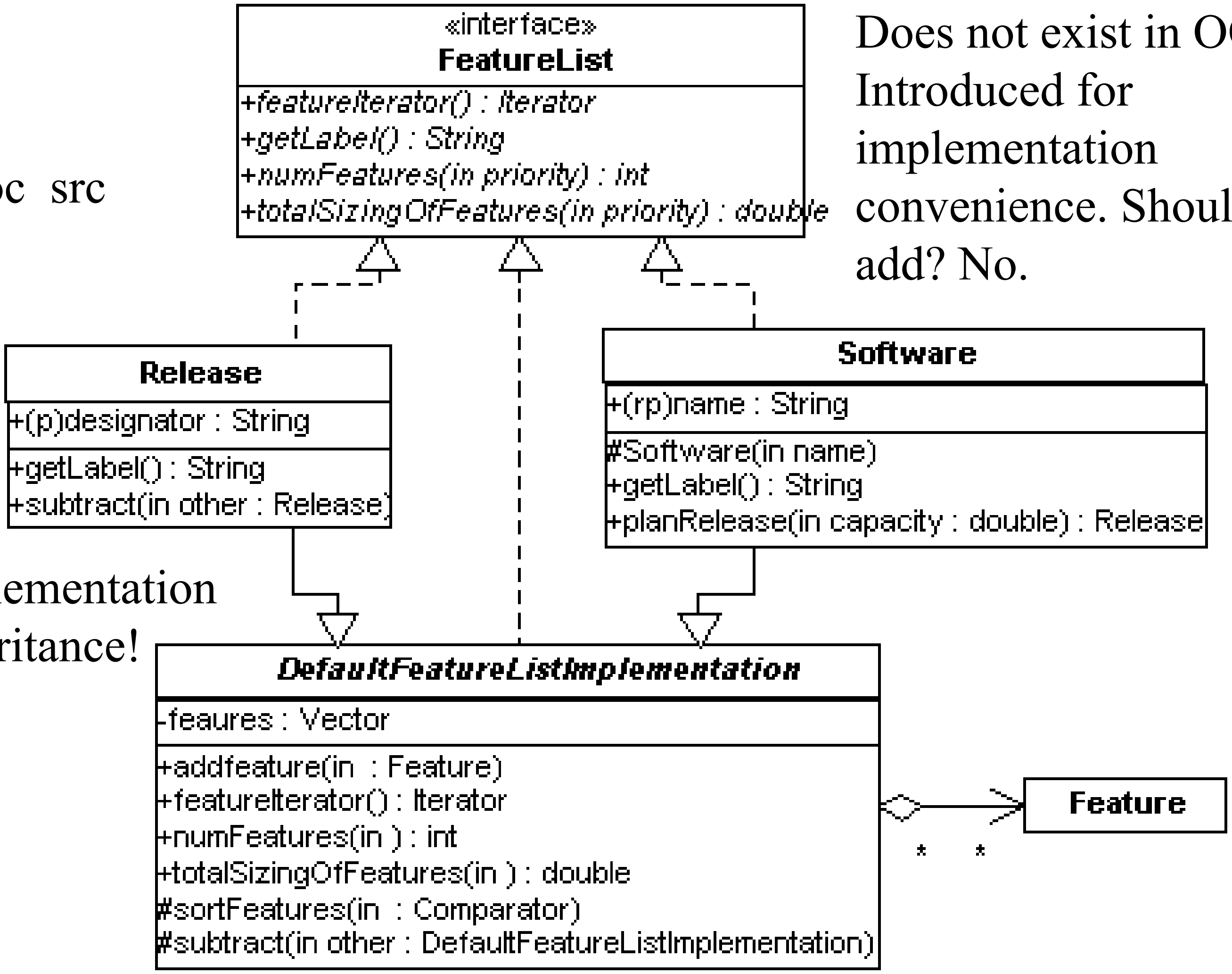


These associations do not exist in the OOA, but are required by this Company-rooted implementation concept.
Should we add to OOA?
Maybe.

ood

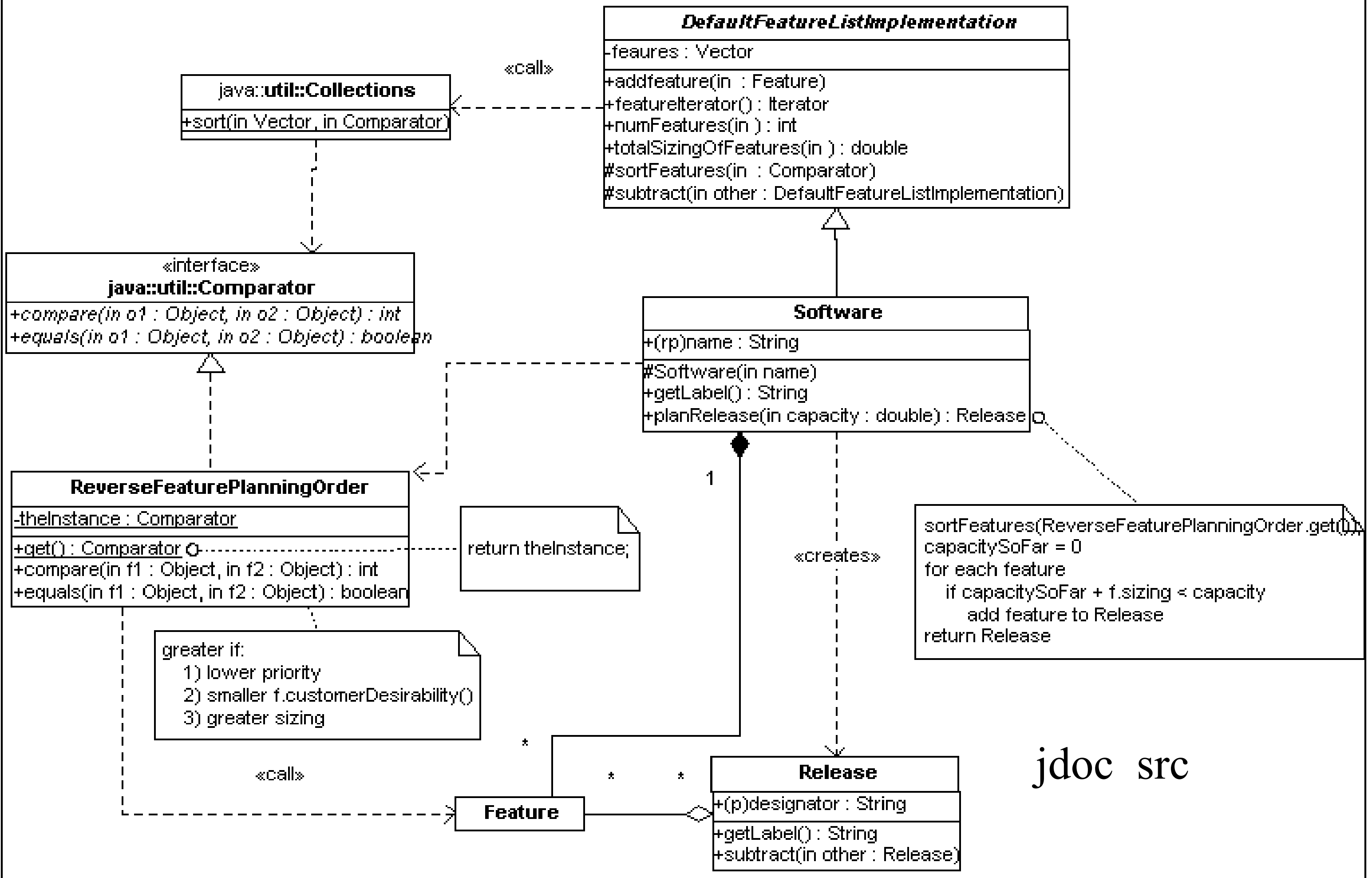
jdoc src

Does not exist in OOA. Introduced for implementation convenience. Should we add? No.


«interface»
FeatureList
+featureIterator() : Iterator
+getLabel() : String
+numFeatures(in priority) : int
+totalSizingOfFeatures(in priority) : double
Release
+(p)designator : String
+getLabel() : String
+subtract(in other : Release)
Software
+(rp)name : String
#Software(in name)
+getLabel() : String
+planRelease(in capacity : double) : Release
DefaultFeatureListImplementation
-feaures : Vector
+addfeature(in : Feature)
+featureIterator() : Iterator
+numFeatures(in) : int
+totalSizingOfFeatures(in) : double
#sortFeatures(in : Comparator)
#subtract(in other : DefaultFeatureListImplementation)
Feature
*
*

Implementation inheritance!

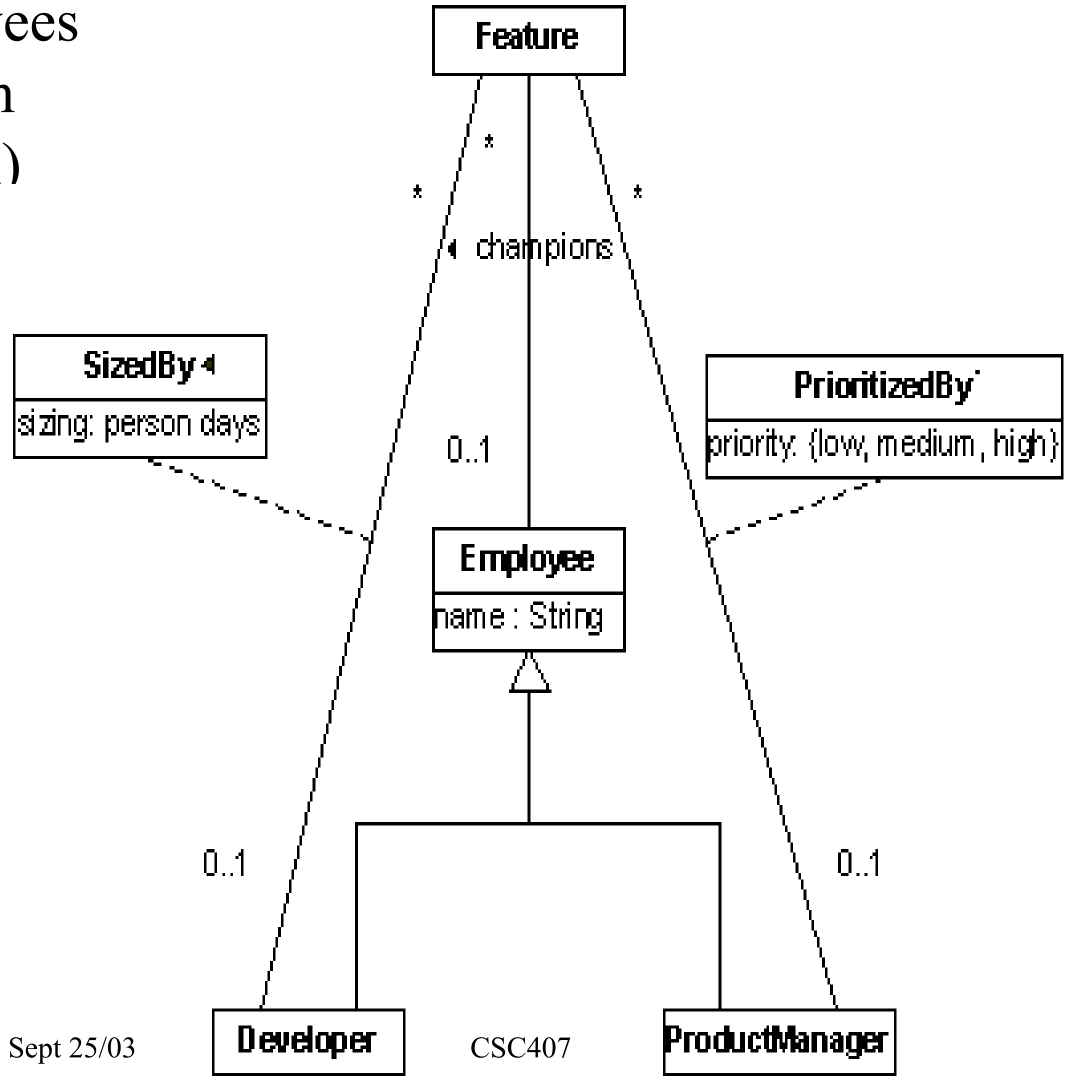
ood


«call»
java::util::Collections
+sort(in Vector, in Comparator)
DefaultFeatureListImplementation
-feaures : Vector
+addfeature(in : Feature)
+featureIterator() : Iterator
+numFeatures(in) : int
+totalSizingOfFeatures(in) : double
#sortFeatures(in : Comparator)
#subtract(in other : DefaultFeatureListImplementation)
«interface»
java::util::Comparator
+compare(in o1 : Object, in o2 : Object) : int
+equals(in o1 : Object, in o2 : Object) : boolean
Software
+(rp)name : String
#Software(in name)
+getLabel() : String
+planRelease(in capacity : double) : Release
ReverseFeaturePlanningOrder
-theInstance : Comparator
+get() : Comparator
+compare(in f1 : Object, in f2 : Object) : int
+equals(in f1 : Object, in f2 : Object) : boolean
return theInstance;
1
«creates»
greater if:
1) lower priority
2) smaller f.customerDesirability()
3) greater sizing
*
«call»
*
*
Feature
Release
+(p)designator : String
+getLabel() : String
+subtract(in other : Release)

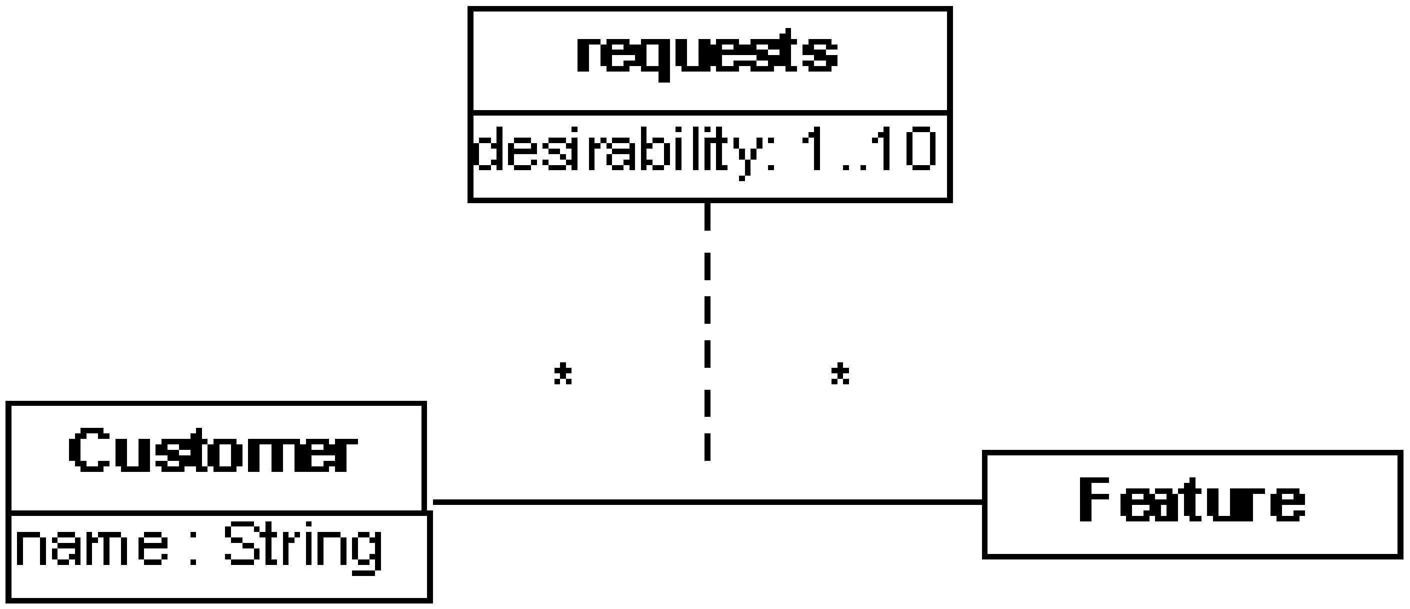
```
sortFeatures(ReverseFeaturePlanningOrder.get());
capacitySoFar = 0
for each feature
   if capacitySoFar + f.sizing < capacity
      add feature to Release
return Release
```

jdoc src

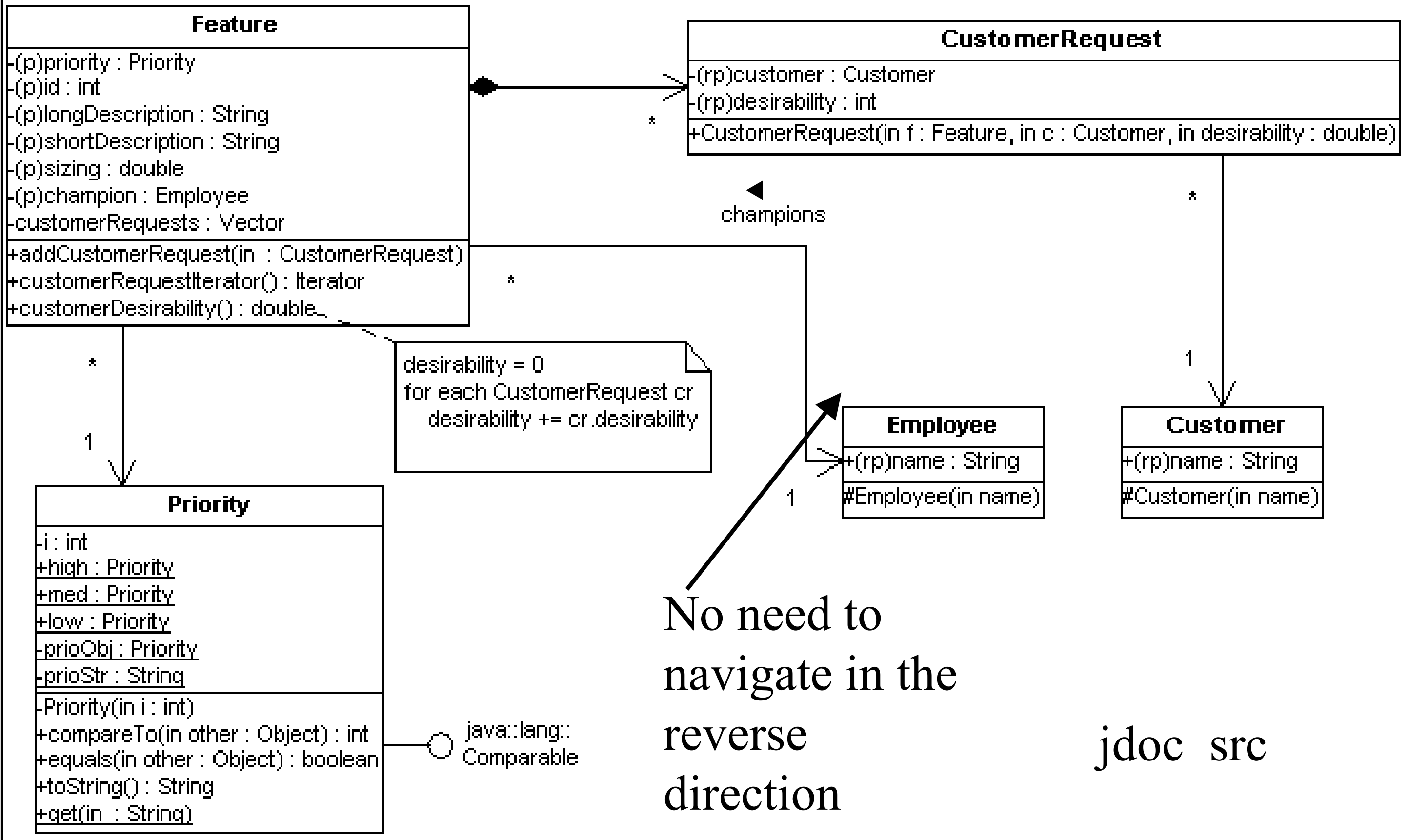
Employees (from OOA)



Customers (from OOA)



ood



No need to navigate in the reverse direction

jdoc src