

15-413

***Lecture Notes on
CASE-Tools: Together/J***

Guenter Teubner
Technische Universitaet Muenchen
Institut fuer Informatik

Michael A. Smith
Carnegie Mellon University

17 September 1998

Outline of the lecture

- ❖ **What is CASE?**
 - ◆ **The acronym**
 - ◆ **Typical components of CASE tools**
- ❖ **Major goals and concepts**
 - ◆ **Lifecycle support**
 - ◆ **Roundtrip engineering**
- ❖ **Working with Together/J**
 - ◆ **The windows of Together/J**
 - ◆ **Creating and modifying class diagrams**
 - ◆ **Handling complexity with packages**
 - ◆ **Code and documentation generation**

What means CASE?

- ❖ The acronym CASE stands for
 - ◆ Computer
 - ◆ Aided
 - ◆ Software
 - ◆ Engineering

- ❖ The term *CASE tool* covers tools supporting the software engineering *process*. In reality, often even tools which support only one particular part of this process (such as compilers, editors, UI generators) are called CASE tools.

- ❖ Our definition is: *CASE tools are browsers and editors for models in graphical and textual form.*

Typical components of CASE tools

- ❖ Typical functionality
 - ◆ **browsing and editing with a graphical user interface**
 - ◆ **automatic code generation**
 - ◆ **documentation generation**
- ❖ Project repository
 - ◆ **persistent storage of all development documents**
 - ◆ **integrated version control system**
 - ◆ **concurrent, distributed modeling**
- ❖ Interface to other tools
 - ◆ **software development tools**
 - ◆ **process and workflow modeling tools**
 - ◆ **offering a scripting language**

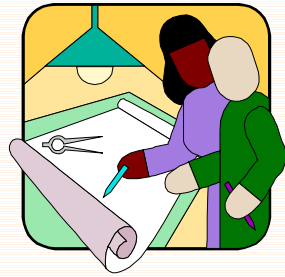
The goal: Full lifecycle support

- ❖ The goal behind CASE is to support all the activities of software development with a single tool.

Analysis



Design



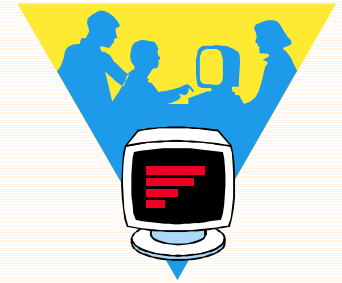
Implementation



Testing



Maintenance



Current situation: Quality of support differs

- ❖ Not all aspects of the software engineering process are supported by today's CASE-tools !

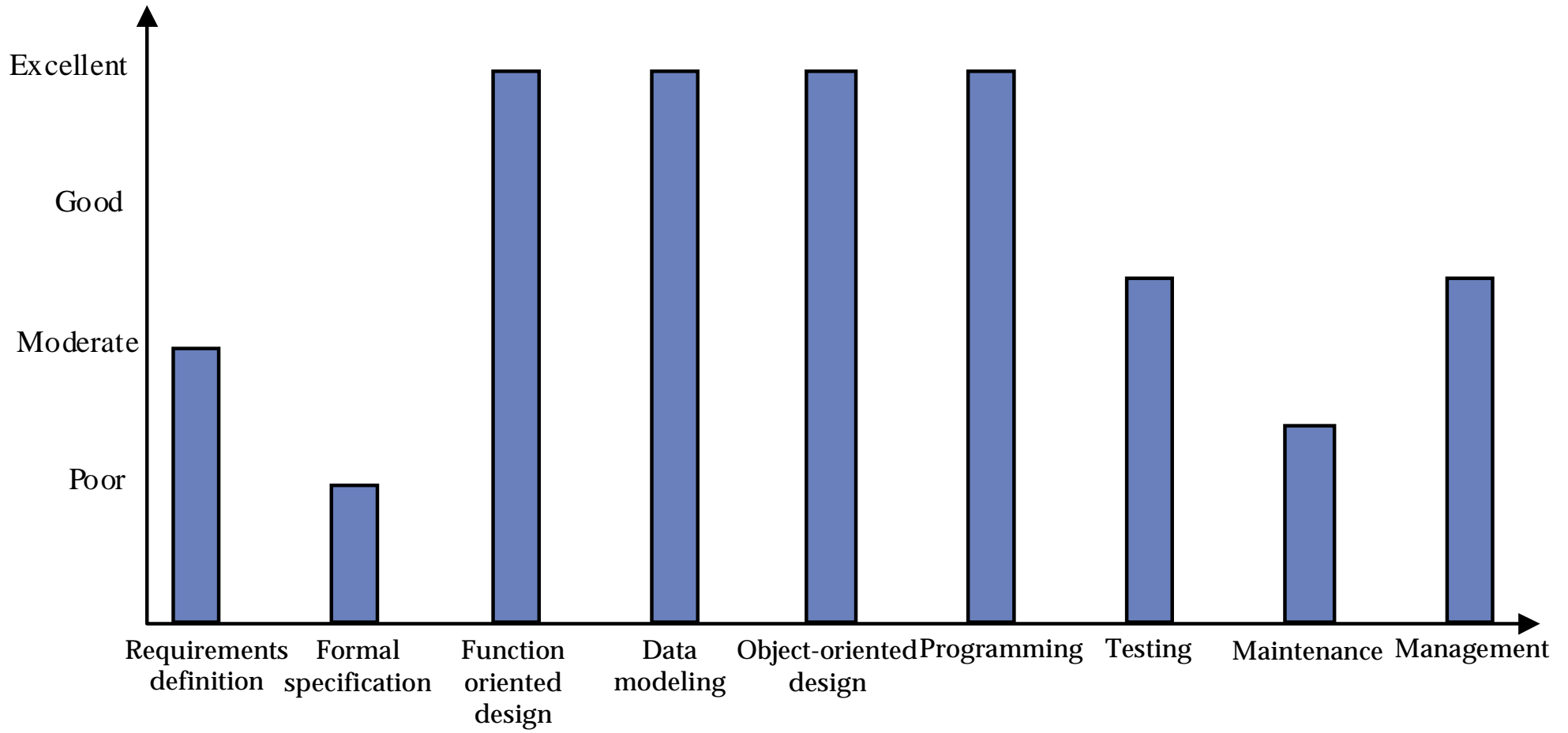
- ❖ Good support for
 - ◆ requirements analysis (class diagrams, use cases, etc.)
 - ◆ implementation

- ❖ Moderate support for
 - ◆ system design
 - ◆ testing
 - ◆ maintenance

- ❖ Poor support for
 - ◆ requirements elicitation

Quality of CASE support today

Quality of support (according to Ian Sommerville)



Level of integration

❖ not integrated

- ◆ **separate CASE tools exist for different parts of the software engineering activities**
- ◆ **each tool has its own set of project documents and a unique user interface**
- ◆ **the user works with multiple tools**

❖ integrated

- ◆ **all tools are working on the same project documents**
- ◆ **a tool can trigger activities of other tools (e.g. start an formal integrity check after a model has been changed)**
- ◆ **the tools share one common user interface**
- ◆ **the user has the feeling of working with one tool**

Advantages and promises of CASE tools

- ❖ Integrated development environment
 - ◆ **unique user interface**
 - ◆ **automation of tedious tasks (e.g. code generation)**
- ❖ Guidance in developing
 - ◆ **common language for all developers**
 - ◆ **correct use of description techniques**
 - ◆ **methodical developing steps**
- ❖ Consistency between model and documentation
 - ◆ **documentation is generated out of the model instead being written separately.**
- ❖ Reuse of existing models for new systems

Problems and disadvantages of CASE Tools

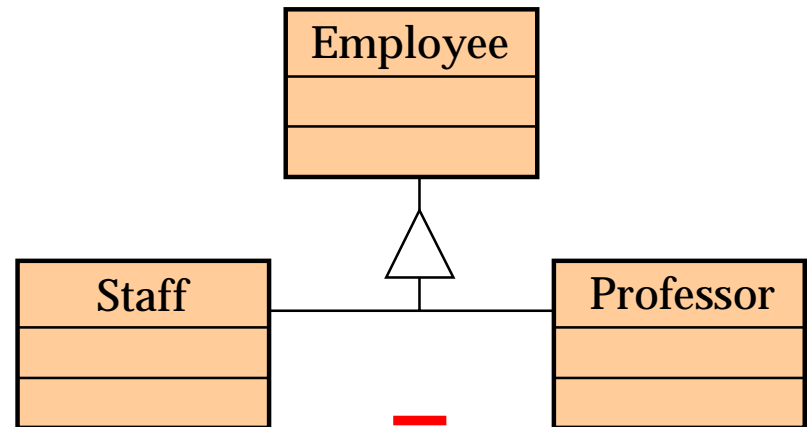
- ❖ Long learning curves
 - ◆ **complex functionality**
 - ◆ **confusing user interfaces**
- ❖ Limited to
 - ◆ **one notation**
 - ◆ **one language**
- ❖ Multi-User support is weak
 - ◆ **“merging” of models is poorly automated**
- ❖ Costs
 - ◆ **CASE tools belong to the most expensive tools in SE**
 - ◆ **CASE tools require high administration effort**

Impact of CASE technology

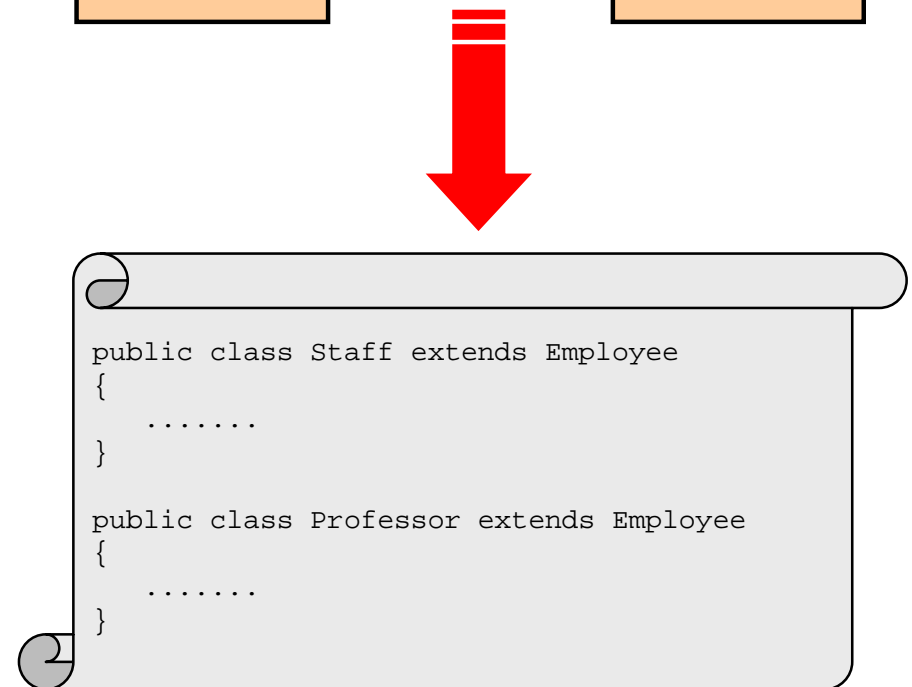
- ❖ CASE technology has resulted in significant improvements in quality and productivity.
- ❖ However, the scale of these improvements is less than was initially predicted by early technology developers
 - ◆ **Many project management problems are not amenable to automation.**
 - ◆ **CASE systems are still not integrated.**
 - ◆ **Adopters of CASE technology underestimated the training and process adaptation costs.**

Forward Engineering

- ❖ Forward engineering is the generation of skeleton code out of the analysis or design models. The developer still has to write the bodies of the methods.



- ❖ Typical flow of events
 - Create or modify an object model for a system**
 - ◆ **Generate the code for this model**
 - ◆ **Allow external modification of this code**

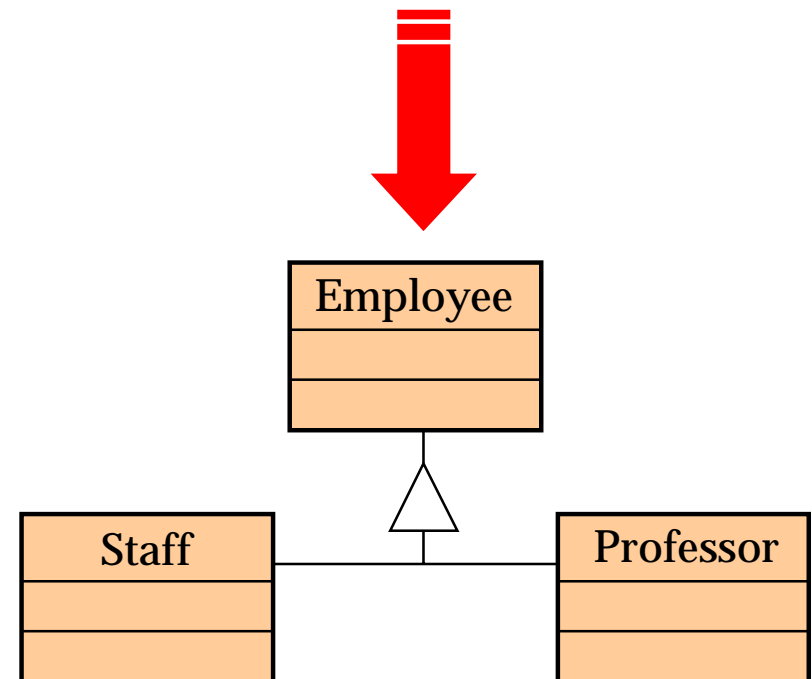


Reverse Engineering

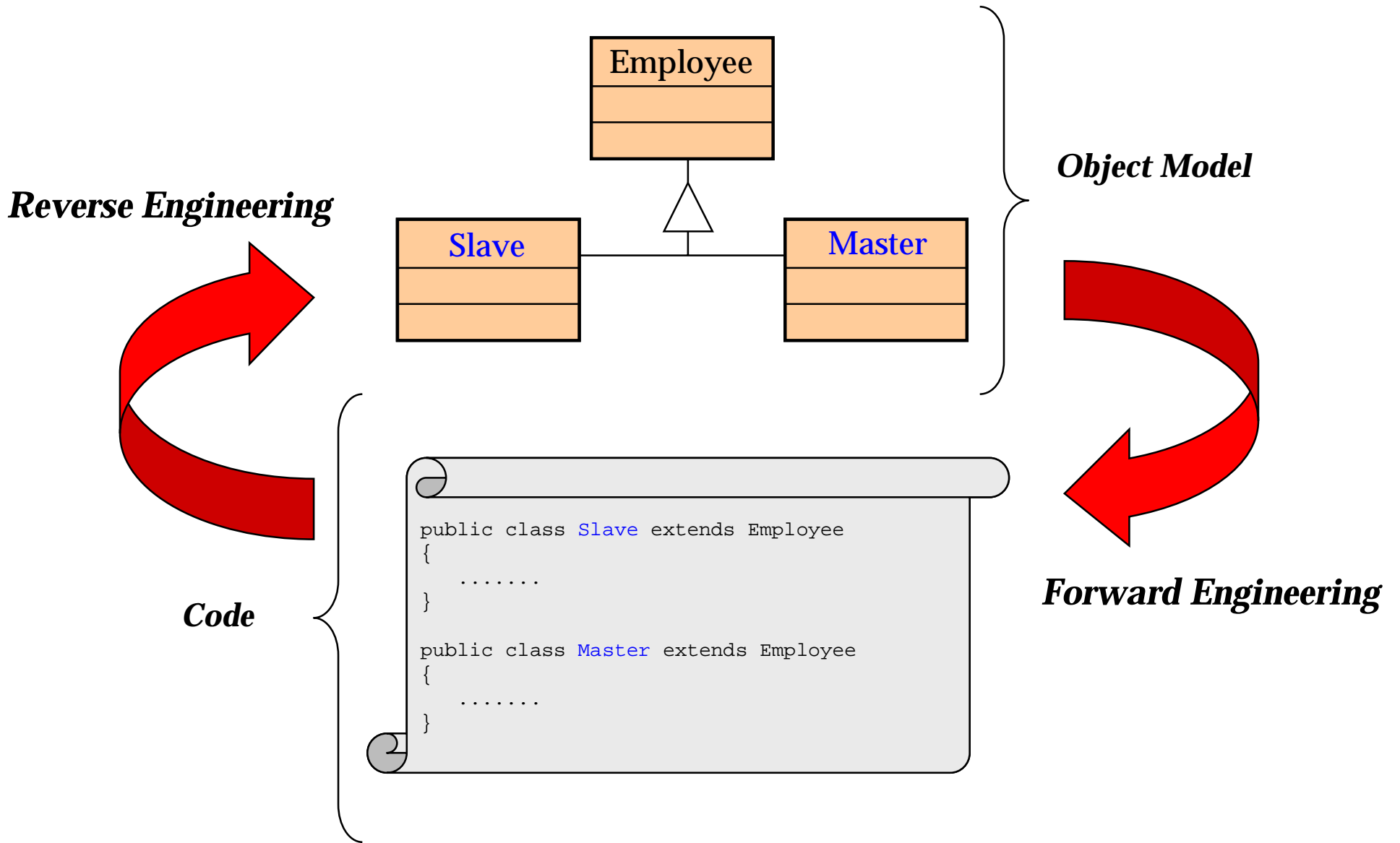
- ❖ Reverse engineering is the recreation of an analysis or design model from existing code.
- ❖ Typical flow of events
 - ◆ Scan a set of already existing source code files
 - ◆ Generate the object model for these files
 - ◆ Allow now modifications on this object model

```
public class Staff extends Employee
{
    .....
}

public class Professor extends Employee
{
    .....
}
```



Roundtrip Engineering



Why roundtrip engineering?

- ❖ Automatic code generation out of the models developed during the design phase is easier, faster and error free than doing it manually.
- ❖ Developers can use specialized tools for editing and debugging that allow faster and easier editing and shorter turnaround cycles during debugging.
- ❖ With reverse engineering, existing code can be discussed and modified on a better manageable basis. Reverse engineering also allows developers to create models for old, never modeled systems.

Reverse engineering vs. Reengineering

❖ Reverse Engineering

- ◆ means analyzing existing software with the purpose of understanding its design and specification.
- ◆ may be part of a reengineering project but may also be used to respecify a system for reimplementation.

❖ Reengineering

- ◆ means restructuring or rewriting parts or all of a legacy system without changing its functionality.
- ◆ involves adding effort to make it easier to maintain. The system may be restructured and redocumented.



Together/J

- ❖ supports UML 1.1
 - ❖ supports Java, C++ and Object Cobol
 - ❖ supports forward and reverse engineering
 - ❖ supports generation of documentation from the model
 - ❖ is written in 100% Java
-
- ❖ A free version (whiteboard edition) can be found under
www.togetherj.com

Working with Together/J (continued)

- ❖ Together/J supports
 - ◆ class diagrams
 - ◆ sequence diagrams
 - ◆ collaboration diagrams
 - ◆ use case diagrams
 - ◆ state transition diagrams

- ❖ Diagrams can be modified in two ways:
 - ◆ **Graphically:** by drawing lines (associations, ...), rectangles (classes, packages, ...) in the diagram pane.
 - ◆ **Menu-based:** by selecting an entity in the diagram pane and using the options in the inspector pane to change its properties.

Model management in 15-413

- ❖ Many models will be created during 15-413
- ❖ Together/J doesn't have a configuration management system

- ⇒ A model management strategy and has to be defined for PAID to avoid conflicts. This is to be done by the architecture team and includes the following topics:
 - ◆ **which models should be created for PAID**
 - ◆ **how are the models organized (e.g. by subsystems)**
 - ◆ **where are the models stored**
 - ◆ **who is allowed to access different models**
 - ◆ **selection of a configuration management system**

Skills expected from you

- ❖ Handling the windows of Together/J
- ❖ Creating and modifying classes
- ❖ Creating and modifying attributes
- ❖ Creating and modifying associations
- ❖ Creating and using packages
- ❖ Creating and using logical packages
- ❖ Handling the other diagram types
- ❖ Creating documentation
- ❖ Code-Generation

Together/J's windows

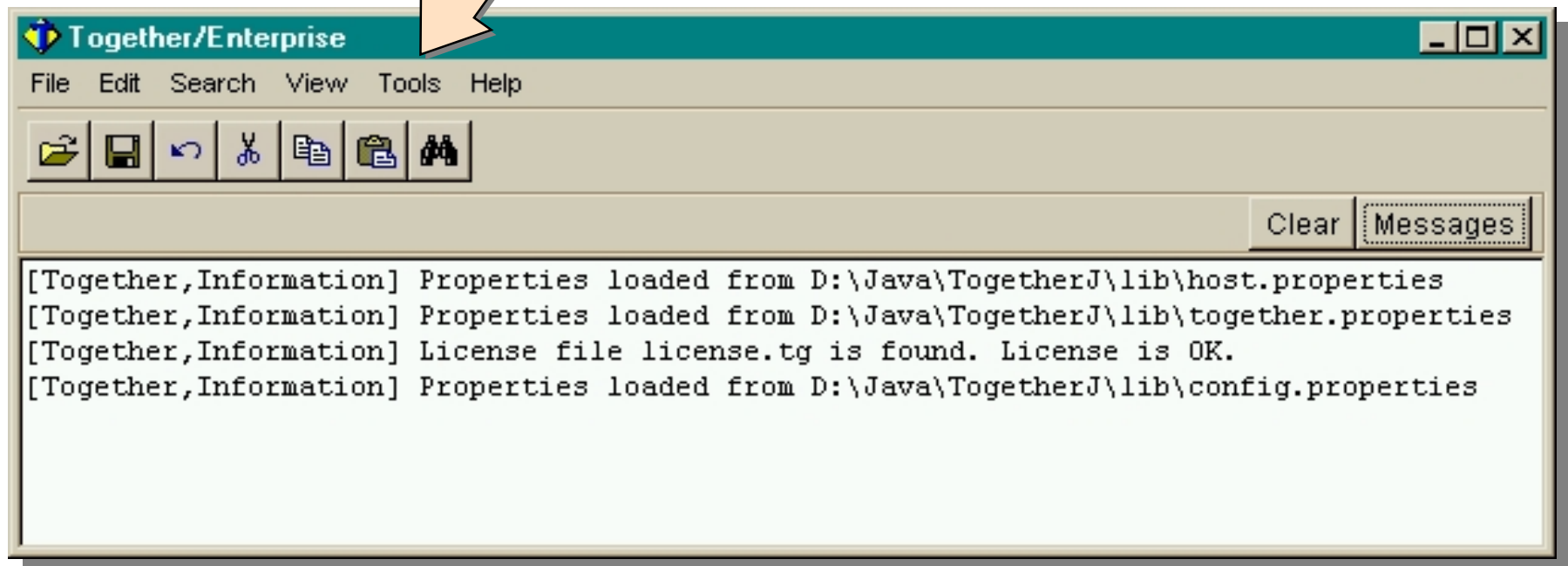
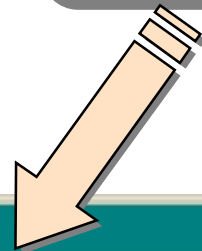
- ❖ After starting Together/J, the *main window* appears
 - ◆ it contains all project-wide commands such as “Open ...”, “Save...” and “Exit” as well as the menus for creating documentation or calling scripts.
 - ⇒ There is exactly one main window for the project.

- ❖ When you open a project, a *browser window* appears
 - ◆ it shows one diagram from the project. The user can modify this diagram in the browser window.
 - ◆ the first browser window always displays the top-level object model of the project.
 - ⇒ The user can then open and work with multiple browser windows for different diagrams simultaneously.



The main window

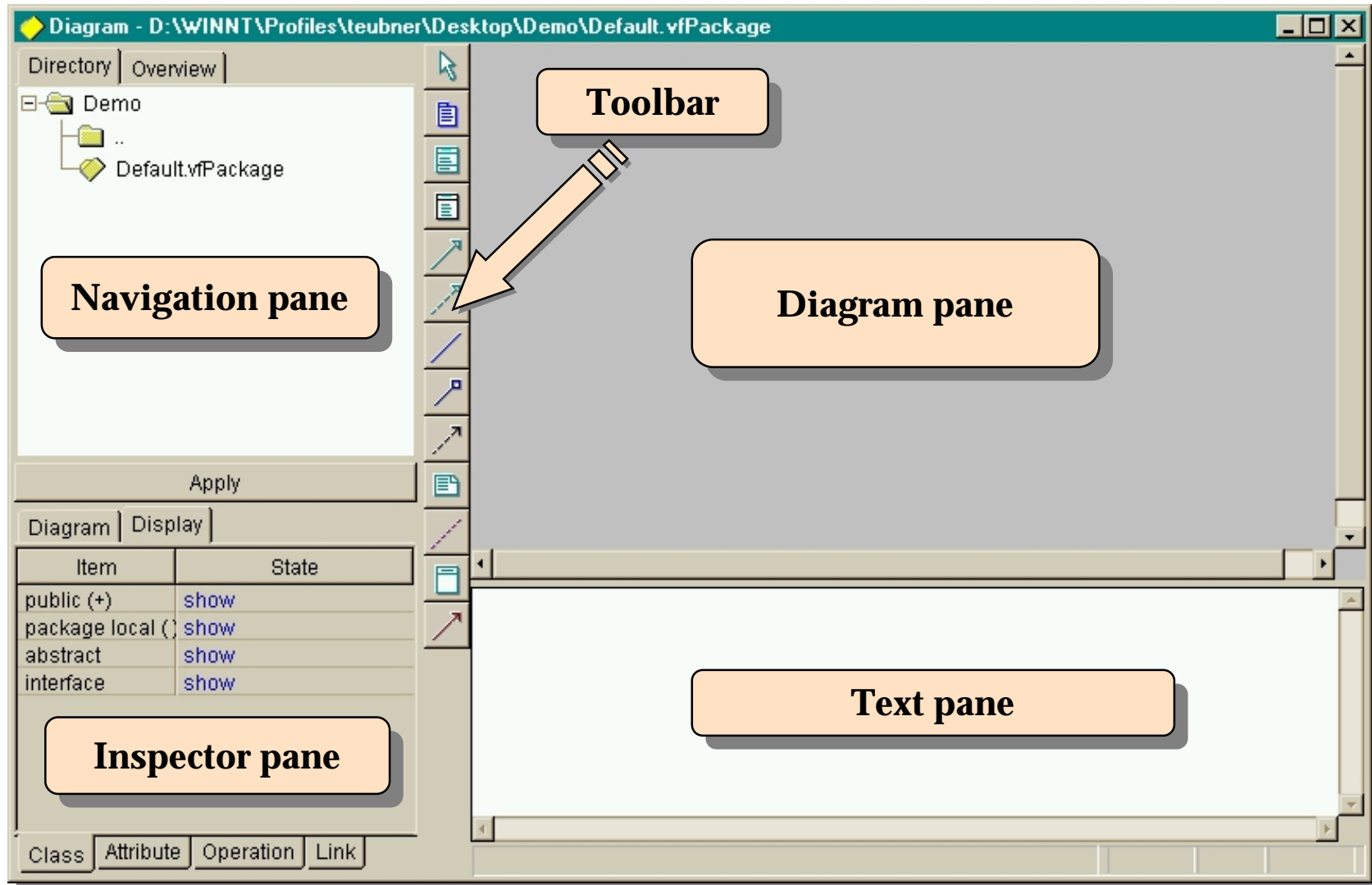
The main window contains all standard menus plus the some features like documentation generation in the “Tools” menu.



A browser window of Together/J

- ❖ shows exactly one diagram of the project
- ❖ is split into 5 parts
 - ◆ ***The navigation pane:*** a hierarchy tree representing the package hierarchy of the project. It is used to switch to other diagrams.
 - ◆ ***The diagram pane:*** a drawing area containing the diagram itself. Allows graphical modifications of the diagram.
 - ◆ ***The toolbar.*** It contains different buttons for each diagram type.
 - ◆ ***The text pane:*** shows the source code for a class that is selected in the diagram pane. Is not visible for other diagrams than class diagrams.
 - ◆ ***The inspector pane:*** an area where specific attributes of the currently selected item can be edited. The contents of this area changes when you click on different elements of the diagram.

Elements of a browser window



Starting or opening a project

- ❖ Select “New Project ...” or “Open Project ...” from the “File” menu of the main window.
- ❖ Give your project a name and enter a path for all the files which are produced while you are modeling your system.
- ❖ When you click on the “Advanced” option, two text areas appear where you can specify additional paths for your class files (*sourcepath*) and directories with Java classes (*classpath*) which you want to use in your project.

Starting a new project

Every project must have a name and a directory where all project files and the Java sources are stored.

Language: Java C++ Objective-C FORTRAN COBOL

Project Name: (a legal filename for your Operating System)

Location:

Java Sourcepath:

Java Classpath:

You can specify alternative source paths and additional classpaths for your project.

Creating classes

- ❖ To create a new class
 - ◆ select the “New Class” button in the toolbar
 - ◆ draw a rectangle in the diagram pane
 - ◆ change the default name for the class to the proper one
 - ◆ use the inspector pane to modify other properties of the class (author, version, etc.)

- ❖ You can always change the properties of the class later.

Creating classes (example)

Diagram - D:\WINNT\Profiles\teubner\Desktop\Demo\Default.vfPackage

Directory Overview

- Demo
 - ..
 - Default.vfPackage
 - Class1.java
 - Person.java
 - Professor.java
 - Staff.java

Apply

Property	Value
name	Class1
package	
file	Class1.java
extends	
implements	
visibility	public
override	none
stereotype	
alias	
note	

Properties Hyperlink To Doc

Col 1 Modified

Select the “New class” button in the toolbar and draw a rectangle in the diagram pane. You can then insert the name of your new class.

Adding and editing attributes

- ❖ **To add an attribute**
 - ◆ **right click on the class in the diagram pane**
 - ◆ **select “New attribute ...” from the context menu**
 - ◆ **enter the name of the attribute**

- ❖ **To modify an attribute**
 - ◆ **click on the attribute in the diagram pane**
 - ◆ **use the inspector pane to change to change to properties of the attribute**

Adding and editing attributes

The screenshot shows a UML diagram editor window titled "Diagram - D:\WINNT\Profiles\teubner\Desktop\Demo\Default.vfPackage". The main diagram area displays four class boxes: Person, Professor, Staff, and Student. The Person class box shows a private attribute `-firstName: int`. Below the diagram is a code editor showing the following Java code:

```
public class Person {  
    private int firstName;  
}
```

The properties inspector pane on the left shows the following table:

Property	Value
name	firstName
type	String
pattern	choose ...
associates	
initial value	
static	false
visibility	private
override	none
transient	false
volatile	false

An orange arrow points from the `-firstName: int` attribute in the diagram to the `type` field in the properties inspector, which is currently set to `String`. Another orange arrow points from a text box to the `type` field.

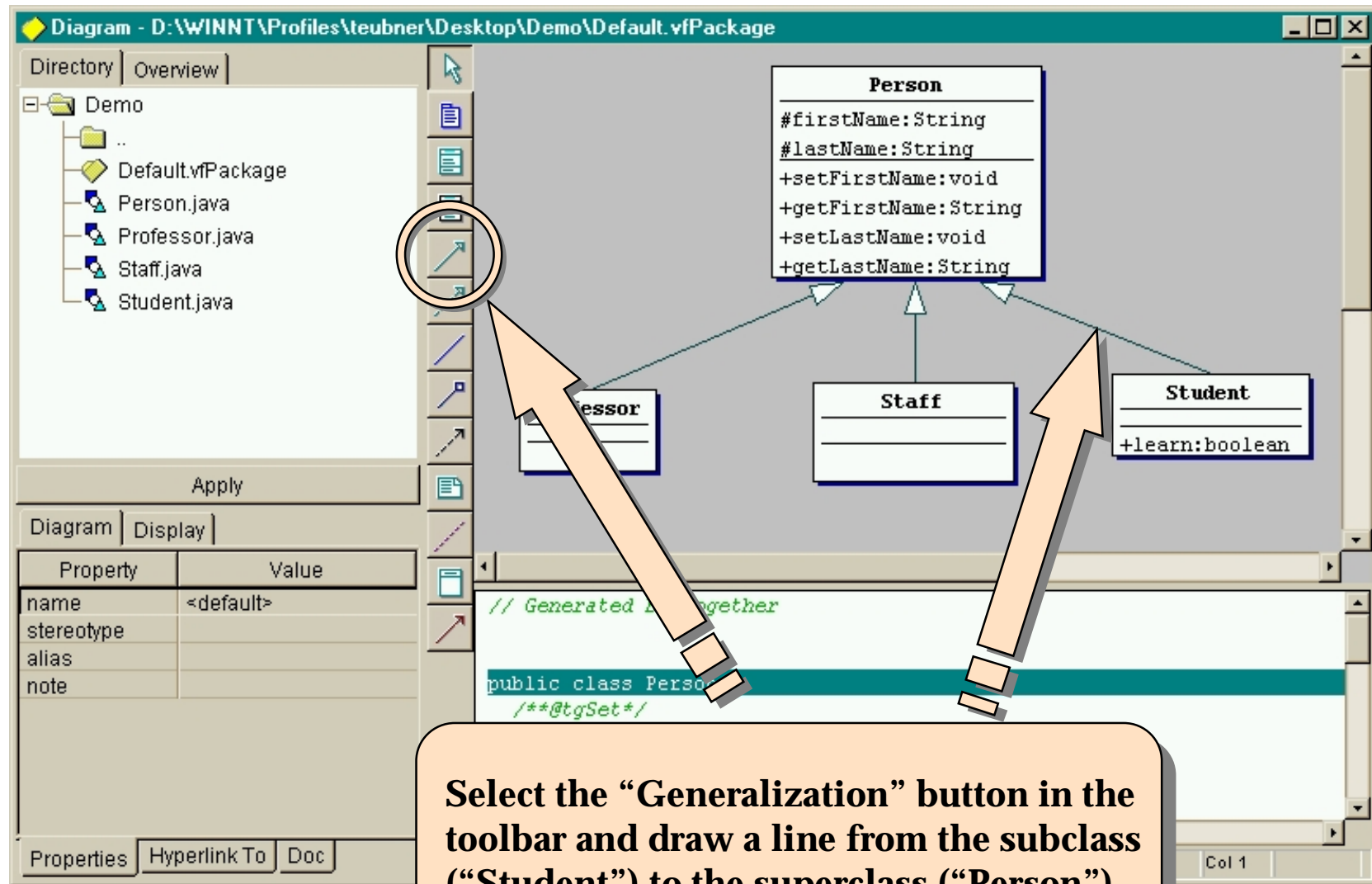
Click on the attribute in the diagram pane and use the inspector pane to modify the attribute.

Generalization (Inheritance)

- ❖ To define a generalization
 - ◆ **click on the “Generalization” button in the toolbar**
 - ◆ **draw a line from the subclass to the superclass. You don’t have to hit certain points of the rectangles; it’s enough when you start the line within the subclass and release the mouse button in the superclass.**

- ❖ To change a generalization
 - ◆ **click on one end of the arrow and drag it to the new subclass respective superclass.**

Generalization (Inheritance)

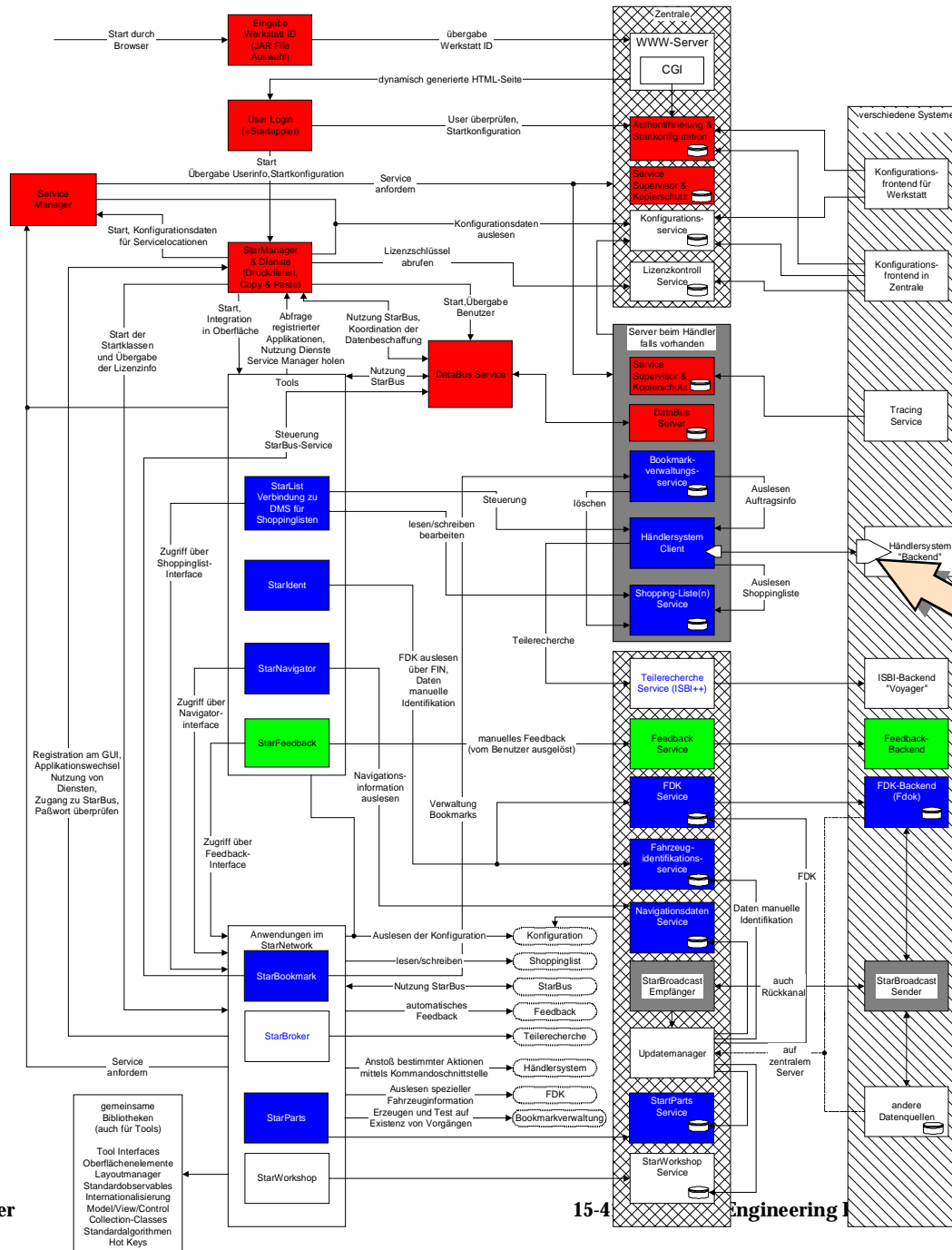
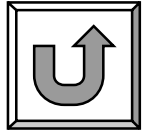


Mastering complexity with Packages

- ❖ Large systems can easily lead to ravioli models which are nearly unreadable.
 - ◆ Example: The system architecture of StarNetwork (next slide)

- ❖ UML packages are organizing constructs on project level
 - ◆ they are hierarchical (a package can contain other packages)
 - ◆ a UML package can correspond to
 - a Java package (such as java.util, java.math)
 - a PAID subsystem
 - ◆ during code generation every package becomes a directory. Together/J parses all packages recursively.

StarNetwork Systemüberblick



7 ± 2 = 41 ?

Is this UML ?

What does this arrow mean ?

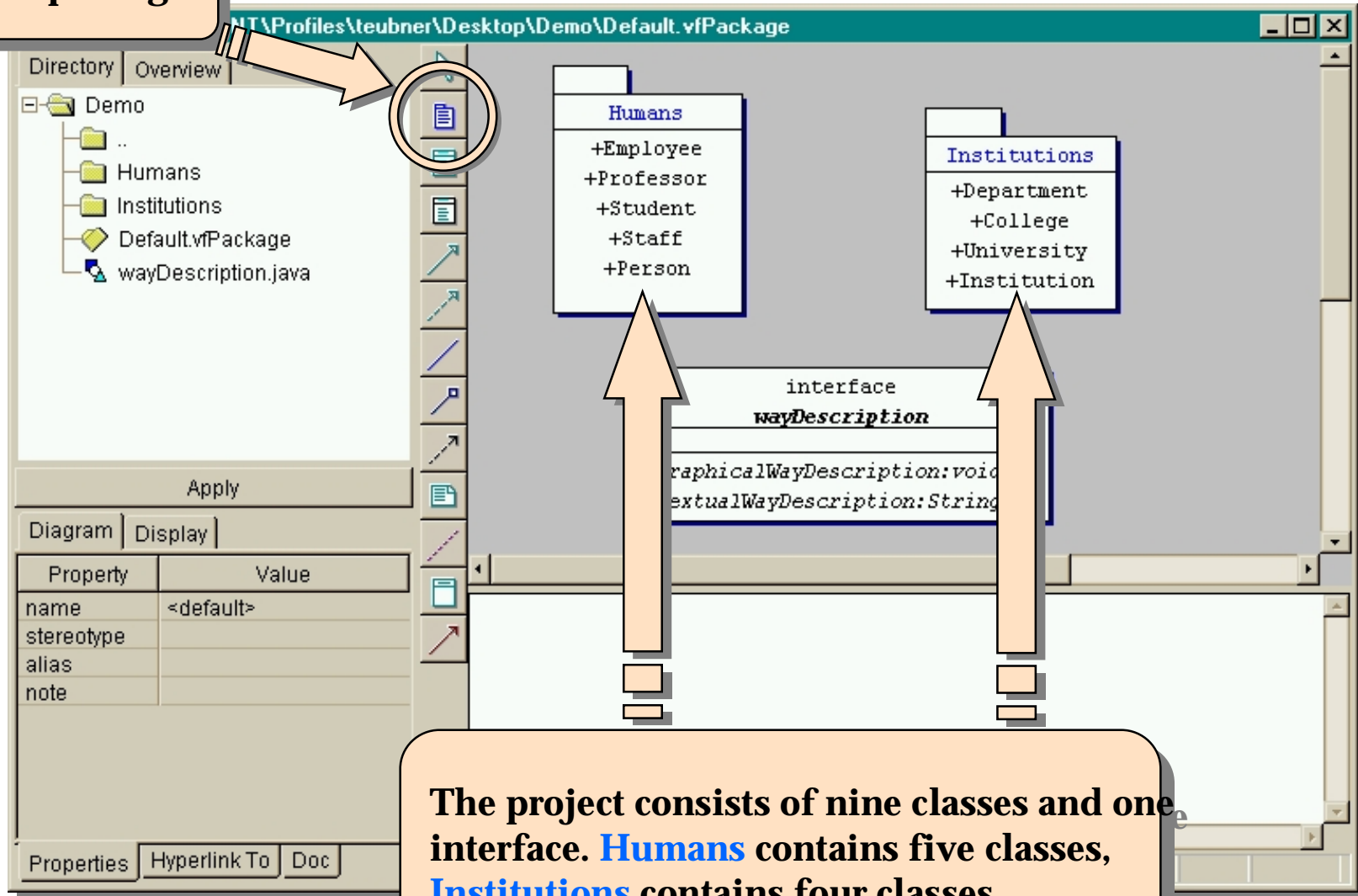
Creating packages

- ❖ Packages are created in the same way as classes are:
 - ◆ select the “New Package” button in the toolbar
 - ◆ draw a rectangle in the diagram pane
 - ◆ change the default name for the package to the proper one
 - ◆ use the inspector pane to modify other properties of the package (author, version, etc.)

- ❖ To organize classes and interfaces into packages simply drag the class/interface into the package.

Mastering complexity with Packages

Create a package



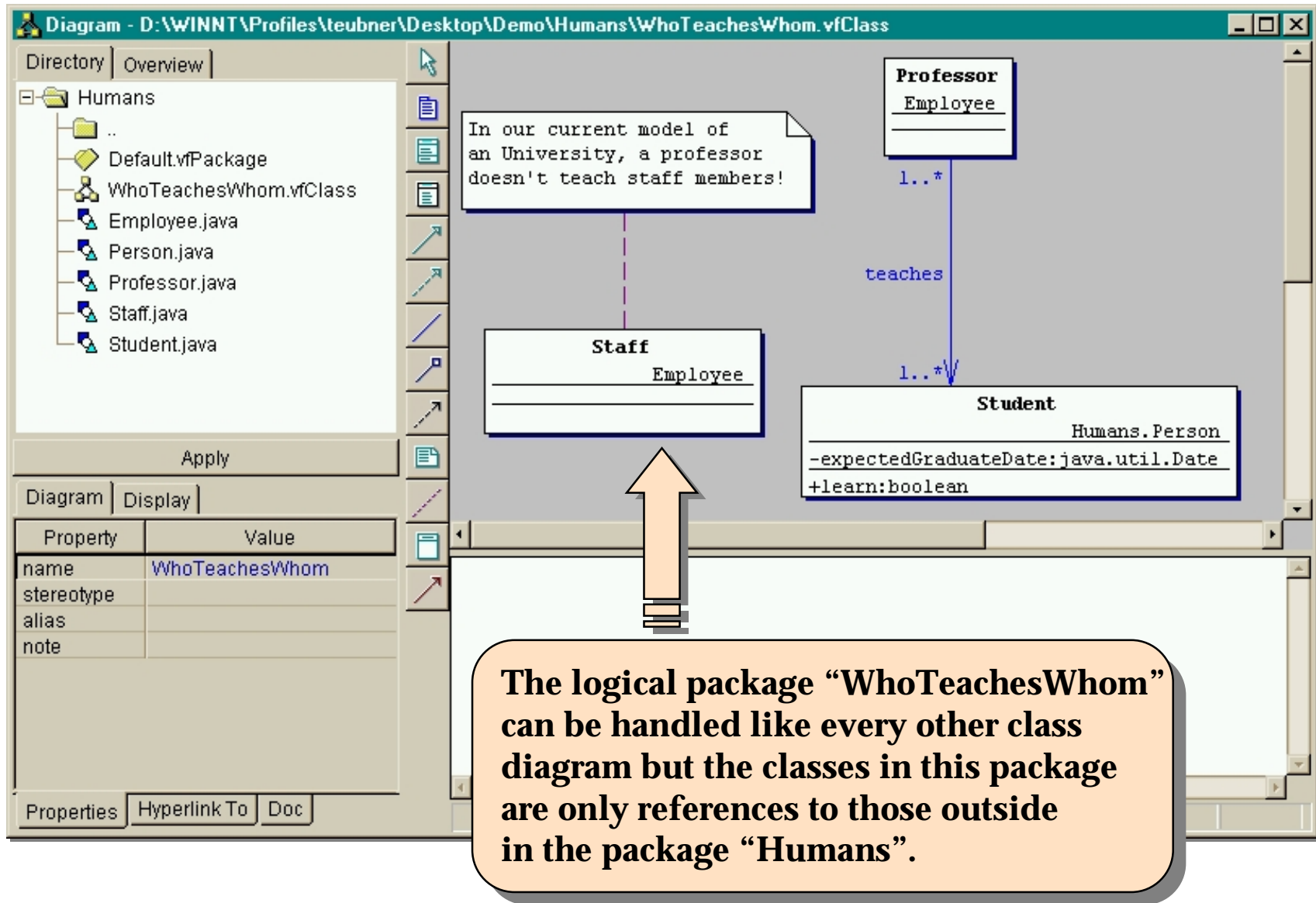
Providing different views: Logical packages

- ❖ TogetherJ allows you to create additional views on your project. These views are called *logical packages* but have nothing in common with the UML or Java packages.
- ❖ You can use this feature to layout the same class diagram in different ways
- ❖ To create a logical package simply create a new class diagram within a package. This diagram gets the suffix “.vfClass” and can be manipulated just like every other class diagram.
- ❖ You can now simply drag some classes into this new logical package or create new classes, associations, etc. there.

Logical packages: An example

- ❖ In this example we create a new class diagram called “WhoTeachesWhom” in the package “Humans”.
- ❖ We then drag “Professor”, “Staff” and “Student” into this new view and create an association between the classes “Professor” and “Student”. We also add a little note that a professor does not teach staff members.
- ❖ Note that the new association is handled as if it has been defined in the default view on the package “Humans” but it is visible only in our newly created logical package “WhoTeachesWhom”.

Inside a logical package

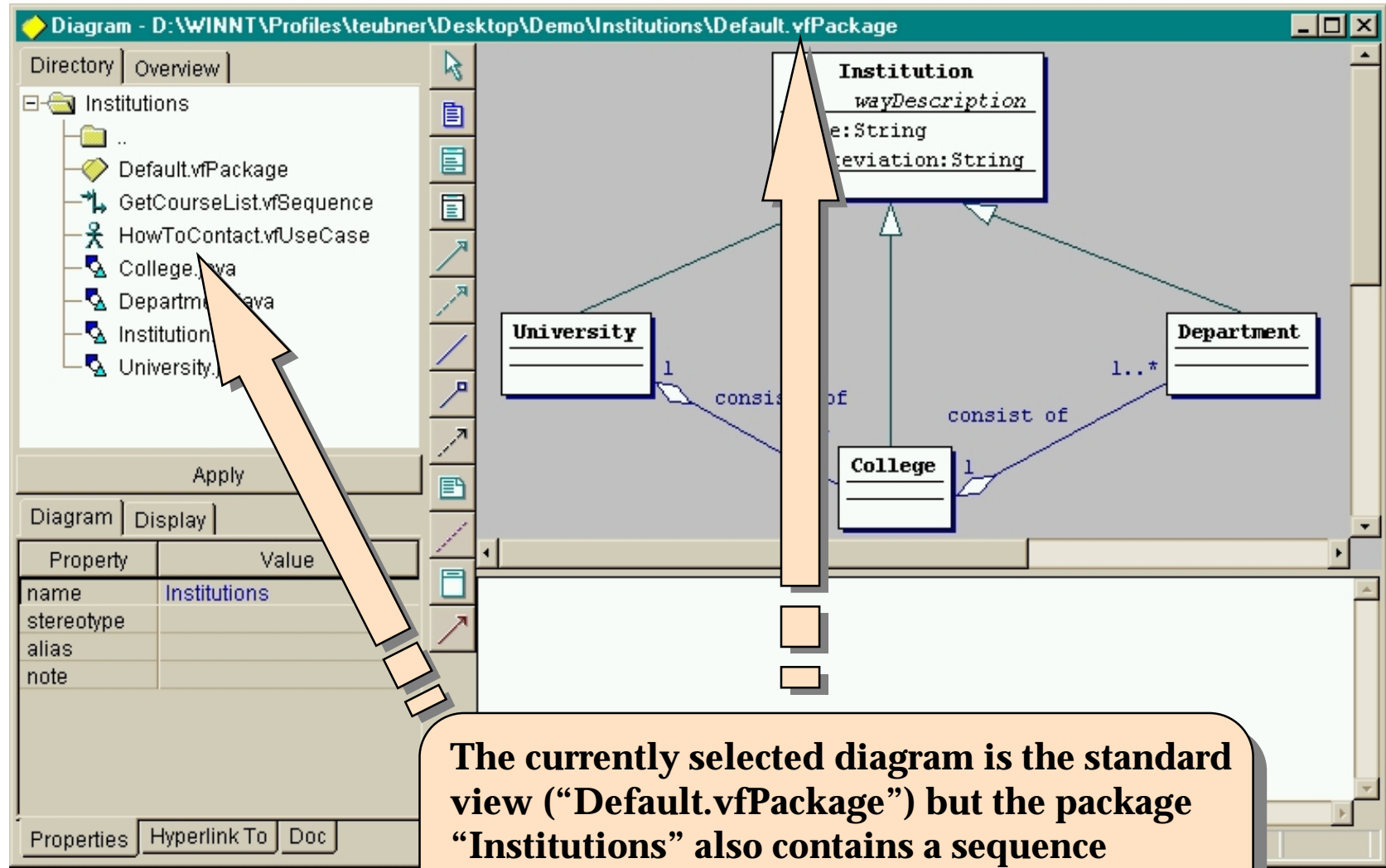


Creating and navigating through diagrams

- ❖ To create a new diagram
 - ◆ right-click on the package name for which you want to create a new diagram
 - ◆ select “New Diagram ...” from the context menu
 - ◆ select the diagram type and give the diagram a name

- ❖ To switch to another diagram
 - ◆ right-click on the name of the diagram in the navigation pane
 - ◆ select “Browse ...” to open the new diagram in the current window
 - ◆ select “Browse in new window ...” to open a new browser window

Creating and navigating through diagrams



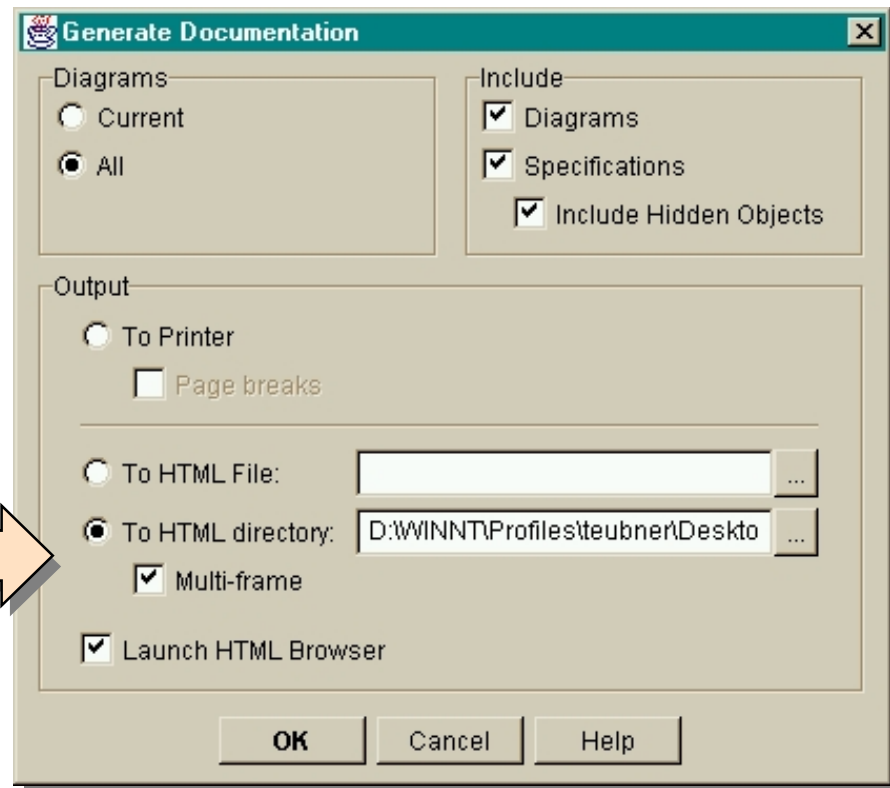
The currently selected diagram is the standard view (“Default.vfPackage”) but the package “Institutions” also contains a sequence diagram (“GetCourseList”) and a use case diagram (“HowToContact”).

Creating documentation

- ❖ You can create documentation in HTML format either for one diagram or for the complete project.
- ❖ To do this, select “Create documentation” from the menu “Tools” of the main window. Together/J creates clickable images containing the class diagrams as well as textual descriptions for all packages and classes.
 - ⇒ **Hint: Separate the documentation directory from your project directory. Otherwise the documentation folder(s) will appear as (sub)packages in your project when you open it the next time. (This is a side-effect of the recursive directory scan which is always performed when you open a project)**

Generate HTML documentation for the whole project

Select an appropriate directory and the “multi-frame” option for the documentation. You might also want to launch the browser immediately after the documentation has been generated.



An example for automatically generated documentation can be found on the next slide!

Netscape
File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Stop

Location: file:///DI/WINNT/Profiles/teubner/Desktop/Demo/Docs/_multiframe.html

Search Engines Wissenschaft Software Firmen, Vereine Reisen Das Leben ... Der ganze Rest

Component Package Diagram Demo

Component Package Diagram Docs

Component Package Diagram Humans

Component Package Diagram Institutions

Component Package diagram Institutions

Class [Institutions.College](#)

public class Institutions.College
extends Institutions.Institution

Links

- Aggregation `lnkUnnamed` to [Institutions.Department](#)

Name consist of
Client cardinality 1
Supplier cardinality 1..*

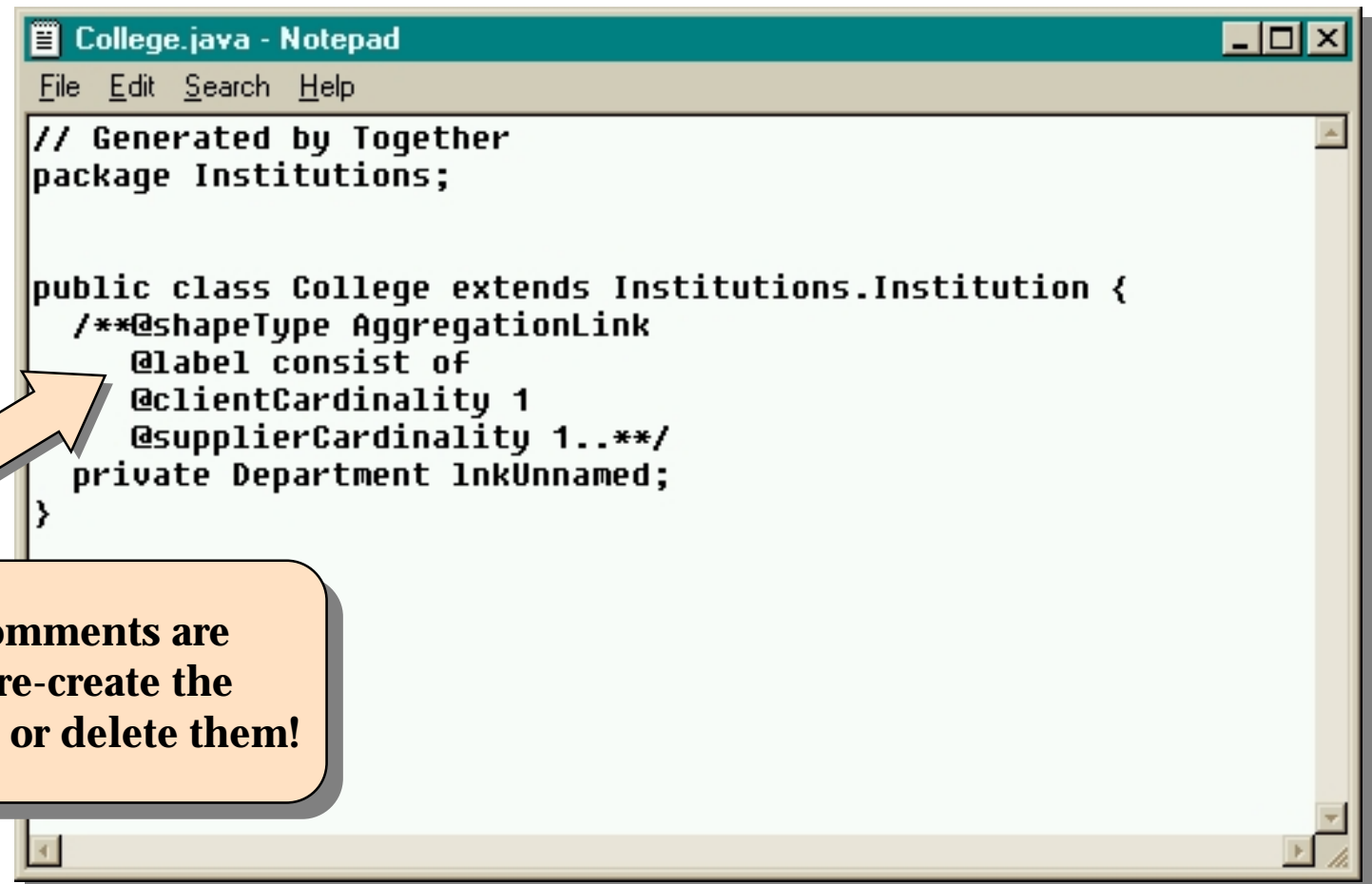
```

classDiagram
    class Institution {
        <i>wayDescription
        -name:String
        -abbreviation:String
    }
    class University
    class Department
    class College
    Institution <|-- University
    Institution <|-- Department
    Institution <|-- College
    College "1" *-- "1..*" University : consists of
    College "1" *-- "1..*" Department : consist of
  
```

Generating code

- ❖ Together/J automatically creates the files with the Java source code for all classes in a project when you save the project. By default, the source files are in the same directory hierarchy as the project files are.
- ❖ The code contains comments with Together/J-specific information. These comments are used when you open the project again. No developer should modify or delete them.

Automatically generated code (example)



```
College.java - Notepad
File Edit Search Help

// Generated by Together
package Institutions;

public class College extends Institutions.Institution {
    /**@shapeType AggregationLink
     * @label consist of
     * @clientCardinality 1
     * @supplierCardinality 1..**/
    private Department InkUnnamed;
}

}
```

These labels within comments are used by Together/J to re-create the model. Do not modify or delete them!