Institute of Automation

C++ Basics and Applications in technical Systems

Lecture 5 - Object oriented Programming

Institute of Automation
University of Bremen

23th November 2012 / Bremen

WiSe 2012/2013

VAK 01-036

Universität Bremen

Institute of Automation

Overview

Universität Bremen

Lecture schedule

Institute of Automation

Time schedule

HK	**26. Oct.** - Introduction / Simple Program / Datatypes ...
HK	**02. Nov.** - Flow control / User-Defined Data types ...
CF	**09. Nov.** - Simple IO / Functions/ Modular Design ...
CF	**16. Nov.** - C++ Pointer
CF	**23. Nov.** - Object oriented Programming / Constructors
AL	**30. Nov.** - UML / Inheritance / Design principles
AL	**07. Dec.** - Namespace / Operators
AL	**14. Dec.** - Polymorphism / Template Classes / Exceptions
HK	**11. Jan.** - Design pattern examples

Universität Bremen

Important dates

Submission of exercises

1-3 **16. Nov.** - Deadline for submission of Exercise I, 13:00

4-6 **07. Dec.** - Deadline for submission of Exercise II, 13:00

For admission to final exam you need at least 50% of every exercise sheet.

Final project

1-9 **15. Feb.** - Deadline for submission of final project, 13:00

Final exam

1-9 **06. Feb.** - Final exam, 10:00-12:00, H3

Pointer in C++

Institute of Automation

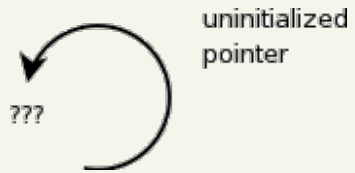
Definition

The value of a pointer variable is an address of a specific memory cell.

Declaration

```
int * pPointerToInteger;
char * pPointerToChar;
float * pPointerToFloat;
```

Attention: The pointer value is not initialized after declaration!



Universität Bremen

Constant pointers on constant values

Institute of Automation

Pointer on constant character

```
const char * pChar;          // character fixed
```

Constant pointer on character

```
char const * pChar;          // pointer fixed
```

Constant pointer on constant character

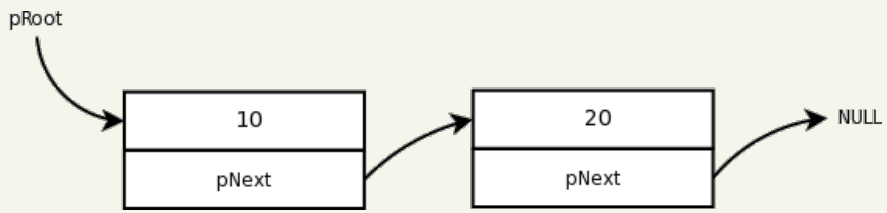
```
const char const * pChar;    // both fixed
```

Universität Bremen

Structures for chained lists / trees

Institute of Automation

```
struct Node_T
{
  int m_iValue;
  Node_T * m_pNext;
};
```




```
// 1. Create root "object"
Node_T * pRoot = new Node_T;

// 2. Initialize elements
pRoot->m_iValue = 10;
pRoot->m_pNext = NULL;
```

```
// 3. Create second "object"
Node_T * pTemp = new Node_T;
pRoot->m_pNext = pTemp;

// 4. Initialize elements
pTemp->m_iValue = 20;
pTemp->m_pNext = NULL;
```

Universität Bremen

Deallocation of memory

Institute of Automation

Memory for allocated objects has to be deallocated after usage:

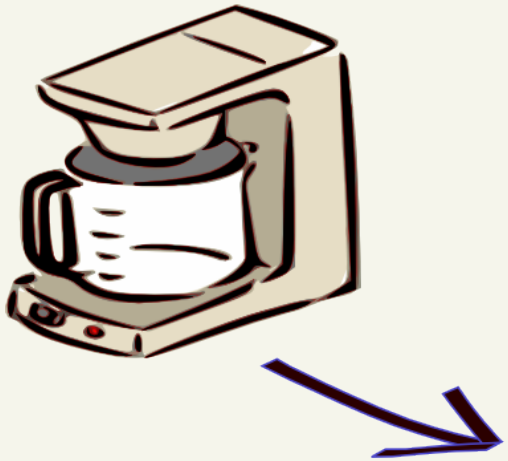
```
1  int * pValue1 = new int;                          // Pointer to int
2  ...                                                    // usage ...
3  delete pValue1;                                  // Deallocate memory
4  pValue1 = NULL;                          // Reinitialize dangling pointer
```

```
1  int iNumber;
2  std::cin >> iNumber;
3  int * pValue2 = new int[iNumber];      // Create array on run-time
4  ...                                                    // usage ...
5  delete [] pValue2;       // Deletes the array (deallocates memory)
6  pValue2 = NULL;                          // Reinitialize dangling pointer
```

Universität Bremen

Institute of Automation

Mission of Object-Oriented-Programming (OOP)



Mapping of

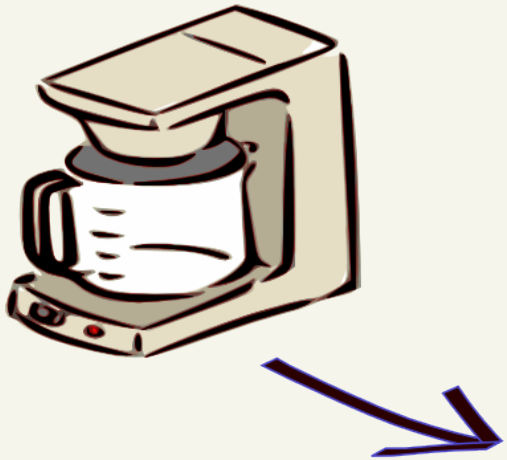
- real world objects
- artificial objects
- elements of a technical application (e.g a GUI)
- concepts (e.g. Petri-Nets)

to software.

Universität Bremen

Institute of Automation

Mission of Object-Oriented-Programming (OOP)



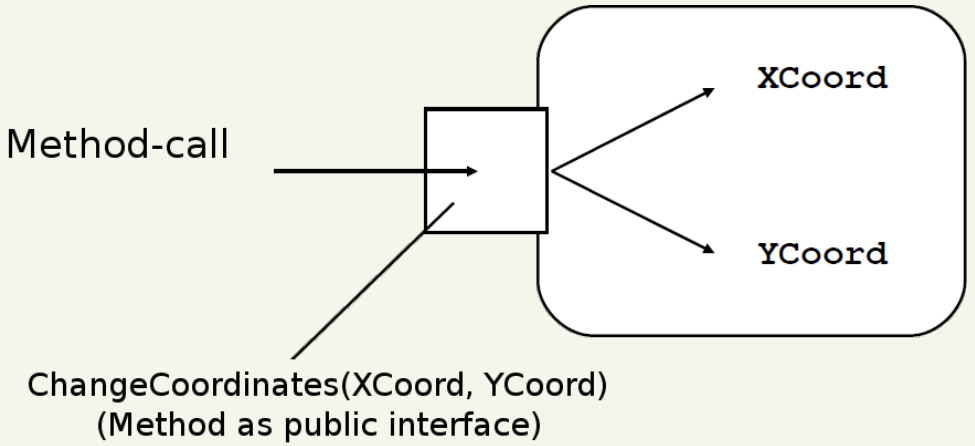
Like real world objects, objects in SW have

- attributes (e.g. states like IsFilled)
- behavior (e.g. moves on the screen)
- relationships (e.g. aggregations like: a Petri-Net has arcs and transitions)

Universität Bremen

Encapsulation principle

Access to the data (attributes) exclusive via member functions of a class (methods) prevents un-authorized access and enables the programmer to influence the way of changing the attributes.



OOP - You already used it ...

Institute of Automation

You already got in contact with convenient to use elements based on classes: Classes from the STL (Standard Template Library):

STL - Standard template library

- `std::string`
- `std::complex<typename>` (template-class)
- `std::vector<typename>` (container-template-class)

⇒ Now you will learn how to construct your own classes, e.g. construct your own abstract datatypes.

Universität Bremen

Classes and objects

Institute of Automation

Keep the difference between class and object in mind!

class

- Is an abstract data-type
- Is the description for objects
  - A class defines the structure of all by its sample produced objects (members + methods)
  - Serves the compiler a description of later defined objects

Example

```
std::string sMyString;
```

`std::string` is a class

Universität Bremen

Classes and objects

Institute of Automation

object

Concrete instantiation of the data-type defined by the class:

- Occupies in contrast to the class memory space
- Has an internal state (represented by the value of the member variables)
- This state can be changed by object activities i.e. operations that are performed with the member variables
- Has a unique identity i.e. address in memory space, that makes it distinguishable from other objects (even with same data content)

Example

```
std::string sMyString;
```

`sMyString` is an object (also called instance of class `std::string`)

Bremen

Classes and objects

Institute of Automation

Declaration and instanciation

1. You define a class (abstract data type, declaration)
2. You instantiate objects of that class (similar to the process of declaring a variable)
3. Each object (or instance):
   - allocates memory from the OS
   - has a unique identity
   - has an internal state (contents of attributes) that can be changed with the class' methods

Components of a class: Attributes

Institute of Automation

Definition

Data members (attributes) have the syntax of variable definitions.

Example

Declaration:

```
int m_iAttribute;
```

Access (if granted):

```
myObject.m_iAttribute = iNewValue;
```

Universität Bremen

Components of a class: Methods

Institute of Automation

Definition

Methods have the syntax of function prototypes and specify the class operations.

Example

Declaration:

```
void MyMethod(void);
```

Implementation:

```
void MyClass::MyMethod(void) { ... }
```

Access (if granted):

```
myObject.MyMethod();
```

Universität Bremen

Components of a class: `const` Methods

Institute of Automation

Definition

`const` methods are not allowed to change the attributes of a class. They are the only permitted operations on const objects.

Example

Declaration:

```
void MyMethod(void) const;
```

Implementation:

```
void MyClass::MyMethod(void) const { ... }
```

Access (if granted):

```
myObject.MyMethod();
```

Universität Bremen

Declaration of classes

Institute of Automation

```
1  // To declare a class, use the class keyword followed by an
2  // opening brace and then list the members of that class.
3  // End the declaration with a closing brace and a semicolon.
4  class Classname
5  {
```

```
1  // Keyword public: specifies those members that are
2  // accessible from outside the class
3  public:
4      ReturnType Method1(ArgType Arg1);
5      ReturnType Method2(void);
```

```
1  // Keyword private: specifies those members that are
2  // not accessible from outside the class directly.
3  private:
4      ReturnType Method3(void);
5      Type m_Attribute1;
6      Type m_Attribute2;
```

```
1  };
```

Universität Bremen

Institute of Automation

Example class header file (*.h)

```
1  #ifndef POINT_H
2  #define POINT_H
3  class Point
4  {
```

```
1  public:
2      int GetXCoord(void) const; // Keyword const ensures that the
3      int GetYCoord(void) const; // methods can't change the object.
4      // The data can be changed only by this method
5      void ChangeCoordinates(int iXCoord, int iYCoord);
```

```
1  private:
2      int m_iXCoord;
3      int m_iYCoord;
```

```
1  };
2  #endif // POINT_H
```

Universität Bremen

Example class implementation file (*.cpp)

Institute of Automation

```
1  #include "Point.h"
```

```
1  int Point::GetXCoord(void) const
2  {
3      return m_iXCoord;
4  }
```

```
1  int Point::GetYCoord(void) const
2  {
3      return m_iYCoord;
4  }
```

```
1  void Point::ChangeCoordinates(int iXCoord, int iYCoord)
2  {
3      m_iXCoord = iXCoord;
4      m_iYCoord = iYCoord;
5  }
```

Universität Bremen

Institute of Automation

Example main programm

```
1  #include "Point.h"
2  #include <iostream>
3  using std::cout;
4  using std::endl;
```

```
1  int main()
2  {
```

```
1      // create object
2      Point newPoint;
3      newPoint.ChangeCoordinates(2, 5); // change coordinates
4      cout << "The Point has the coordinates:" << endl
5           << "x = " << newPoint.GetXCoord() << endl
6           << "y = " << newPoint.GetYCoord() << endl;
```

```
1      return 0;
2  }
```

Universität Bremen

Access restrictions on members

Institute of Automation

Attributes and methods fall under one of three different access permissions:

- **Public** members are accessible by all class users.
- **Private** members are only accessible by the class members.
- **Protected** members are only accessible by the class members and the members of a derived class.

Universität Bremen

Default access restrictions

Institute of Automation

If no keyword is given, the following access permissions are used:

- **struct:** public
- **class:** private

Universität Bremen

Accessor and Mutator

Due to the data encapsulation principle, attributes should be declared as `private` (or `protected`) and classes have to provide the following methods:

- an accessor method to get the data of a class.
- a mutator to change the data of class.

Besides the prevention of unauthorized data access, there is a significant design advantage:

Advantage

The type of attribute can be changed without having to alert all programmers using the class. Just a modification of accessor and mutator is necessary (keeping the signature).

Constructors

Purpose:

- Initialization during definition
- Supply of memory space

Syntax:

- Method name is equal to class name
- No return-value (nor void)

Constructor types:

- Standard constructor
- General constructor
- Copy constructor
- Type conversion constructor (not discussed)

Standard constructor

The standard constructor has no arguments! Implementation is done by:

- the compiler (automatically, not view/changeable, all values of attributes are uninitialized!)
- the programmer (see example)

Example

```
MyClass::MyClass()
{
   ...
}
```

Standard constructor example

Institute of Automation

Header

```
// "Position.h"
class Position
{
public:
    Position();
    int GetXCoord() const;
    int GetYCoord() const;
    void ChangeCoords(
                int iXCoord,
                int iYCoord);

private:
    int m_iXCoord;
    int m_iYCoord;
};
```

This implementation assures initialized attributes.

Implementation

```
// "Position.cpp"

Position::Position()
{
    m_iXCoord = 0;
    m_iYCoord = 0;
}
```

Universität Bremen

Destructor

Clearing up work inside an object before destruction (Most important purpose is the de-allocation of memory space before the object leaves the range of validity). Implementation is done by:

- the compiler (automatically, not view/changeable)
- the programmer (see example on next slide)

Rules for manual implementation:

- no parameters
- no return value
- tilde (~) used as prefix for declaration

Destructor example

Institute of Automation

The destructor has to be declared as public method of the class.

Example

```
// Position.h
class Position
{
public:
    ~Position();
};
```

```
// Position.cpp
Position::~Position()
{
    // Clearing up work
    ...
}
```

Universität Bremen

Exercise

Create a class `Cuboid` that offers the following functionalities (first implement empty methods):

- Set and get the required parameters height, width and depth, e.g. `SetDimensions()`, `GetDimensions()`, ...
- Calculation of surface area and volume, e.g. `CalculateVolume()`, ...
- Returning of the calculated values, e.g. `GetVolume()`, ...
- Implement all these methods.
- Add a standard constructor and a destructor.
- Verify the class with the help of a test program.

General constructor

Institute of Automation

Characteristics:

- Contrary to a standard constructor, a general constructor has parameters
- They can be overloaded (like functions), i.e., there may be several general constructors with different parameters

Attention:

- The introduction of a general constructor prevents the automatic creation of the standard constructor

Universität Bremen

General constructor example

Institute of Automation

Header

```
// "Vehicle.h"
class Vehicle
{
public:
    Vehicle();
    Vehicle(
            double dHeight,
            double dWidth);
    double GetHeight() const;
    double GetWidth() const;

private:
    double m_dHeight;
    double m_dWidth;
};
```

This implementation assures attributes with individual initialization.

Implementation

```
// "Vehicle.cpp"

Vehicle::Vehicle(
            double dHeight,
            double dWidth)
{
    m_dHeight = dHeight;
    m_dWidth = dWidth;
}
```

Universität Bremen

General constructor execution

Institute of Automation

Implementation

```
// "Vehicle.cpp"

Vehicle::Vehicle(
            double dHeight,
            double dWidth)
{
    m_dHeight = dHeight;
    m_dWidth = dWidth;
}
```

- Before entering the block `{...}`, memory for the data elements `m_dHeight` and `m_dWidth` is allocated (header-file).
- The program code inside the block is executed and the parameters `dHeight` and `dWidth` are assigned to the member variables

Universität Bremen

Initialization with lists

Institute of Automation

Header

```
class Vehicle
{
public:
    Vehicle();
    Vehicle(double dHeight, double dWidth);
private:
    double m_dHeight;
    double m_dWidth;
};
```

Implementation

```
Vehicle::Vehicle(double dHeight, double dWidth)
: m_dHeight(dHeight), m_dWidth(dWidth)
{ ... }
```

Bremen

Initialization with lists

Institute of Automation

Characteristics:

- Memory allocation and parameter assignment are summarized
- Run time advantages in case of large or many objects

Processing:

- The initialization order depends on the order of declaration within the class and **NOT** on the order within the list
- Attention when an initialization depends on the result of a preceding initialization!
- The block $\{...\}$ is executed

Universität Bremen

General constructor usage

Institute of Automation

Parameter initialization with object declaration.

Header

```
// "Vehicle.h"
class Vehicle
{
public:
    Vehicle();
    Vehicle(
            double dHeight,
            double dWidth);
    double GetHeight() const;
    double GetWidth() const;

private:
    double m_dHeight;
    double m_dWidth;
};
```

Application

```
// "main.cpp"
#include "Vehicle.h"

int main()
{
    double dHeight = 5.0;
    double dWidth = 3.2;

    Vehicle myVehicle(dHeight,
                      dWidth);

    return 0;
}
```

Copy constructor

Institute of Automation

Definition

Initialization of an object with an already existing object

Characteristics:

- The first parameter is a const reference to an *object of the same class*

Attention:

- If no copy constructor is available, one is created by the compiler automatically (bit by bit!!)
- Problem when elements with manually allocated memory are used

Universität Bremen

Copy constructor example

Institute of Automation

Header

```
class Vehicle
{
public:
    Vehicle(const Vehicle& vehicleObj);
private:
    double m_dHeight;
    double m_dWidth;
};
```

Implementation

```
Vehicle::Vehicle(const Vehicle& vehicleObj)
 : m_dHeight(vehicleObj.m_dHeight),
   m_dWidth(vehicleObj.m_dWidth)
{ ... }
```

Bremen

Copy constructor usage

Creation of an object `SecondVehicle`, which is initialized with the values of the already existing object `FirstVehicle`

Example

```
Vehicle firstVehicle(5.0, 3.5);          // General constructor
Vehicle secondVehicle = firstVehicle;    // Copy constructor
```

Although the equal-sign is used, an **initialization** and **not** an **assignment** takes place.

Example

```
Vehicle thirdVehicle;
thirdVehicle = firstVehicle;             // Assignment
```

Important

Remember

A copy constructor is **only** necessary during the **creation** of a new object, not for the assignment with an object, i.e., for changes of already existing objects!

Small Exercise

Institute of Automation

Extend the class `Cuboid` that offers the following functionalities:

- A general constructor to set these height, width and depth with the creation of an object.
- A copy constructor to initialize an object with another object of the same class.
- Instantiate an object of that class with the help of a test program. Use the general constructor.
- Instantiate another object of that class by using the copy constructor.
- Get the parameters of the second object and print them on the screen.

Universität Bremen

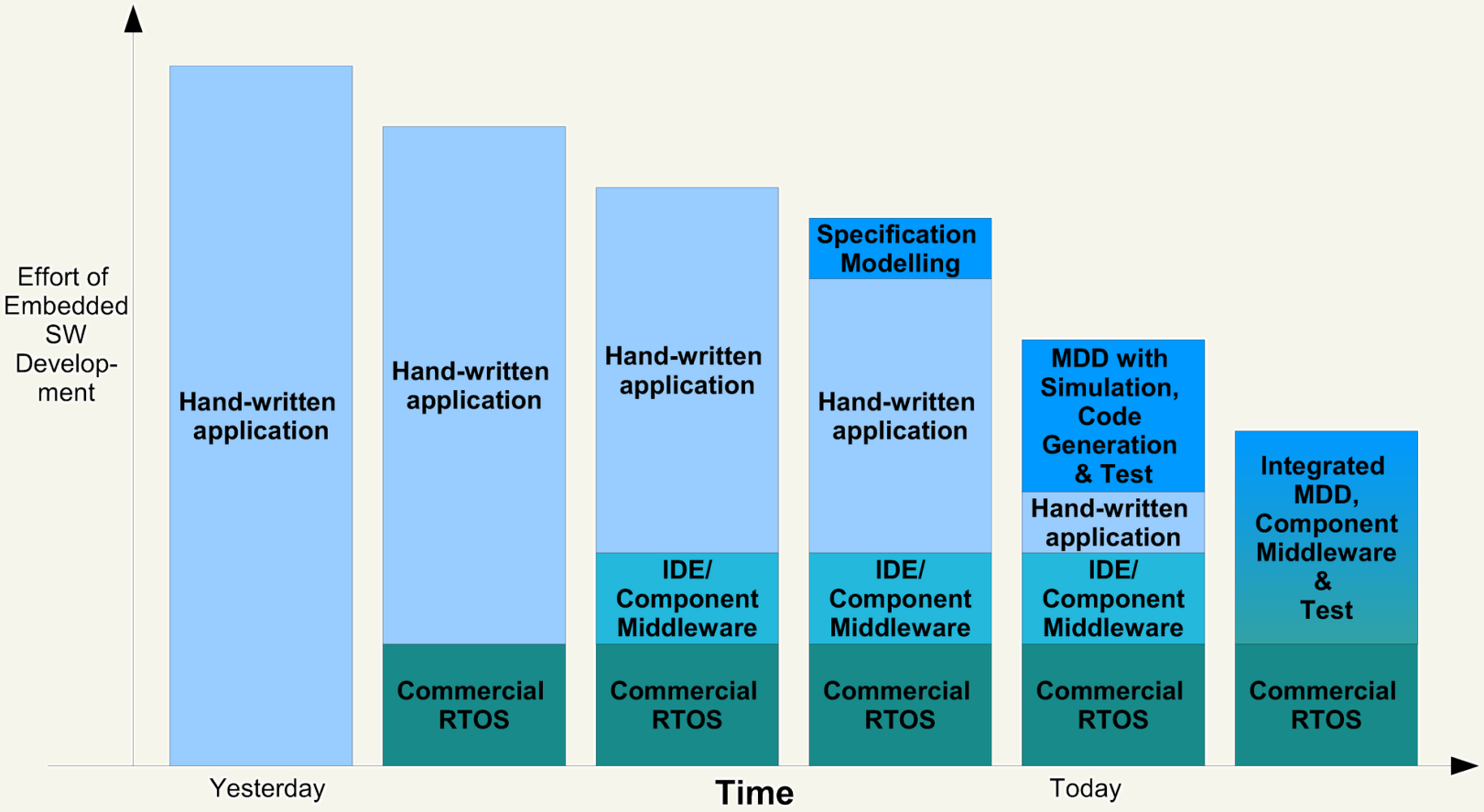
Institute of Automation

Introduction to MDD (Model Driven Development)

Universität Bremen

Institute of Automation

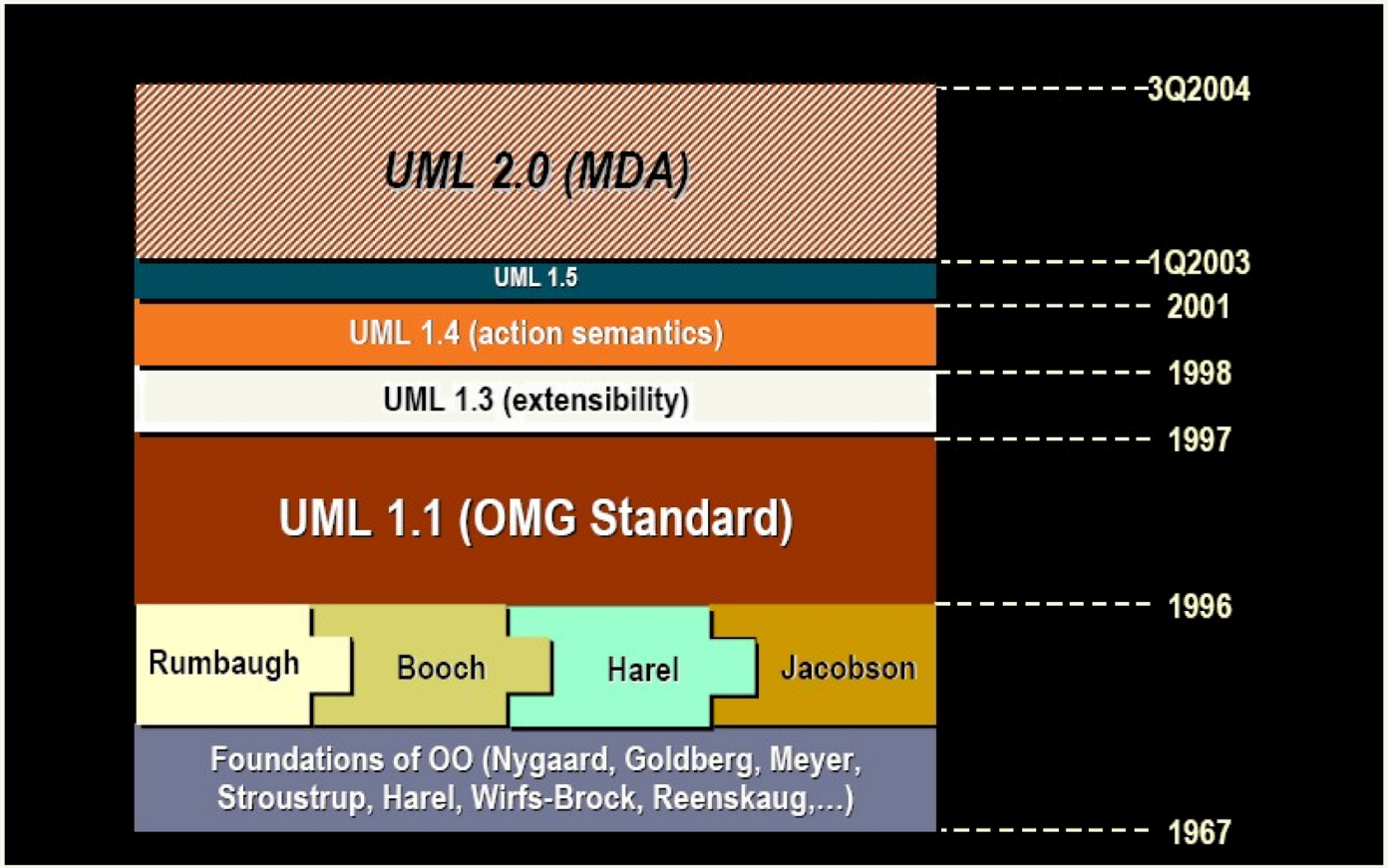
MDD: Emerging future technology



Universität Bremen

Institute of Automation

UML (Unified Modeling Language) – the history



Universität Bremen

Institute of Automation

UML – The mission

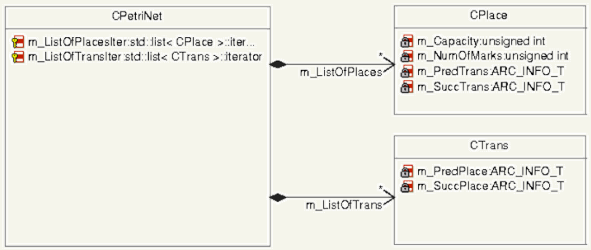
- UML is a **standardized modeling language** consisting of an integrated set of diagrams
- Developed to help system and software developers accomplish the following tasks
  - Specification
  - Visualization
  - Architecture design
  - Construction
  - Simulation and Testing
- Allows the developer to concentrate "**on the big picture**"
- Properly constructed diagrams and models are **efficient communication techniques**

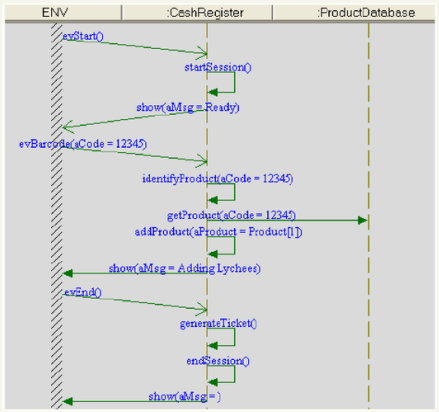
Universität Bremen

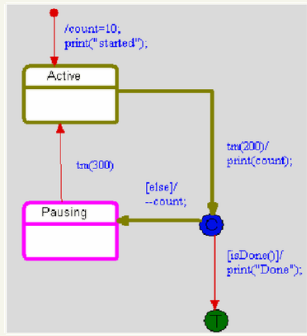
Institute of Automation

UML – 13 diagram types

- Activity Diagram
- **Class Diagram**
- Communication Diagram
- Component Diagram
- Composite Structure Diagram
- Deployment Diagram
- Interaction Overview Diagram
- **Object Diagram**
- **Package Diagram**
- **Sequence Diagram**
- **State Machine Diagram**
- Timing Diagram
- **Use Case Diagram**







Good online summary on UML:
http://bdn.borland.com/article/31863

Universität Bremen

Institute of Automation

Rhapsody – Technical issues

- Easiest and preferred access:
  - Usage on notebook
  - Need connection to IAT license server via OpenVPN
  - Apply for certificate by writing email to
    kampe@iat.uni-bremen.de
  - Follow instructions provided at
    www2.iat.uni-bremen.de/~friend2/sw/rhapsody
    - user: guest
    - password: iatguest2006

Universität Bremen

Institute of Automation

Rhapsody – Sources of information

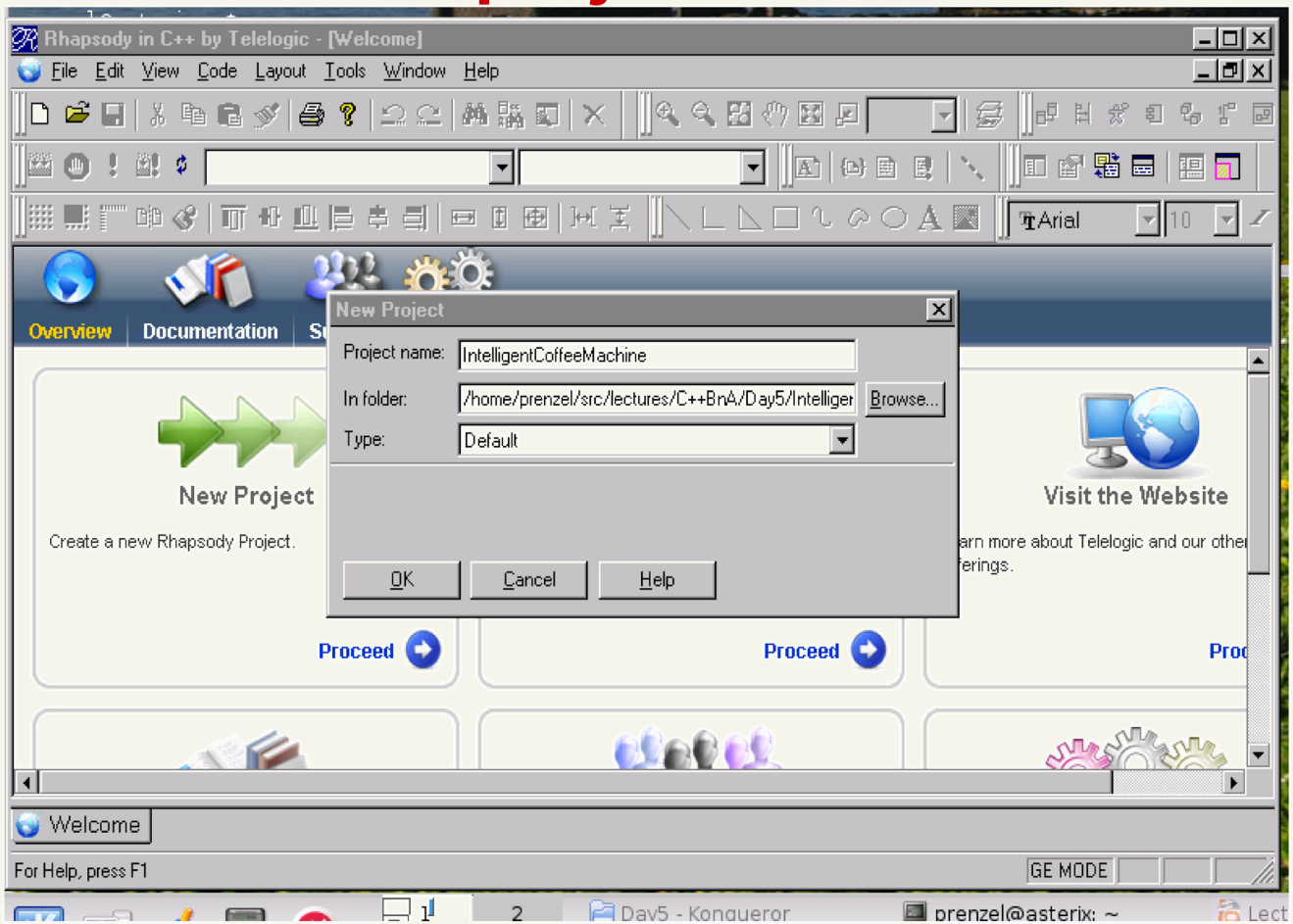
- RhapsodyInstallDir/Doc
  - user guide, properties manual, tutorials
- iatWiki extracts are provided in the elearning system

- For OOP-experienced users:
  - Rhapsody Hands-On-Workshop, provided on www2.iat.uni-bremen.de/~friend2/sw/rhapsody

Universität Bremen

Institute of Automation

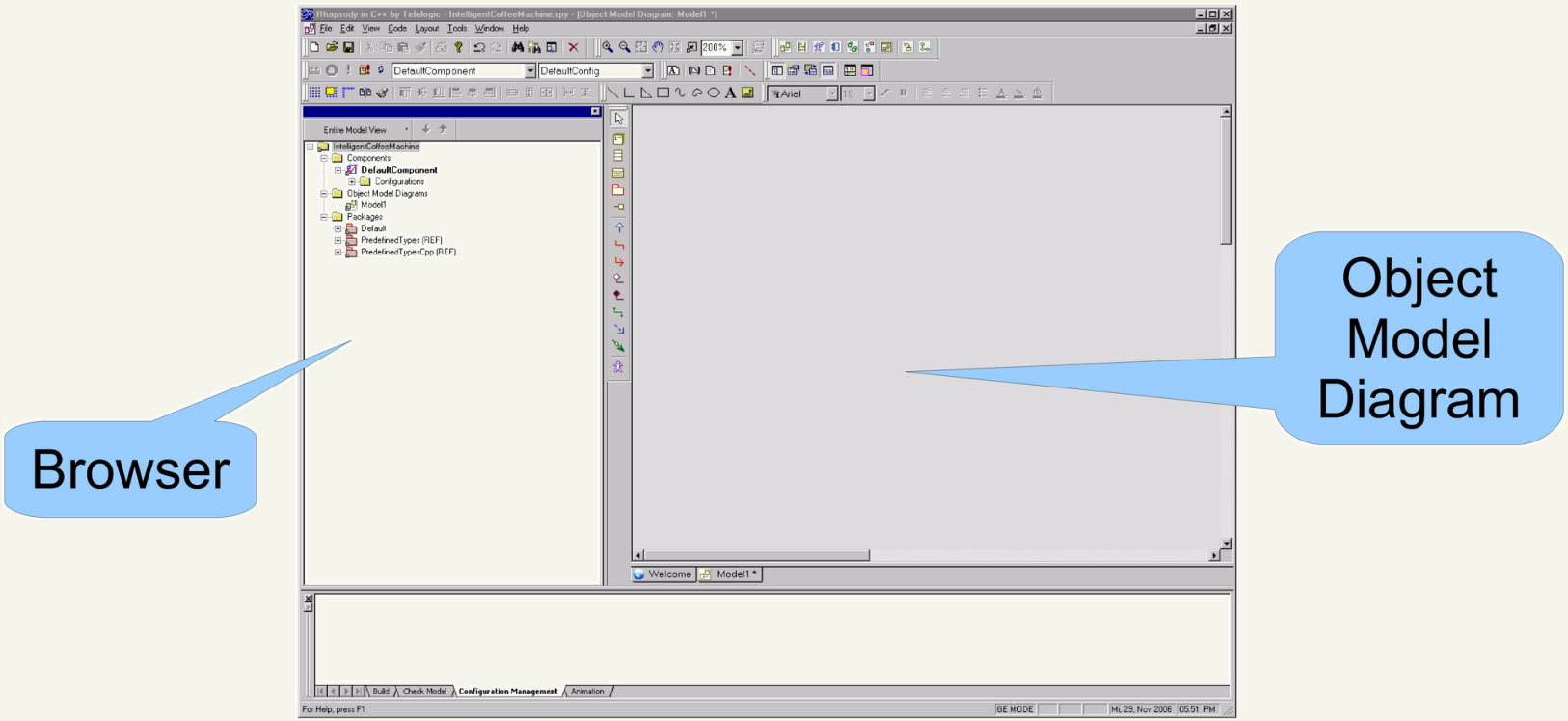
Rhapsody – first steps
#1: Create a new project



Universität Bremen

Institute of Automation

Rhapsody – first steps
The browser



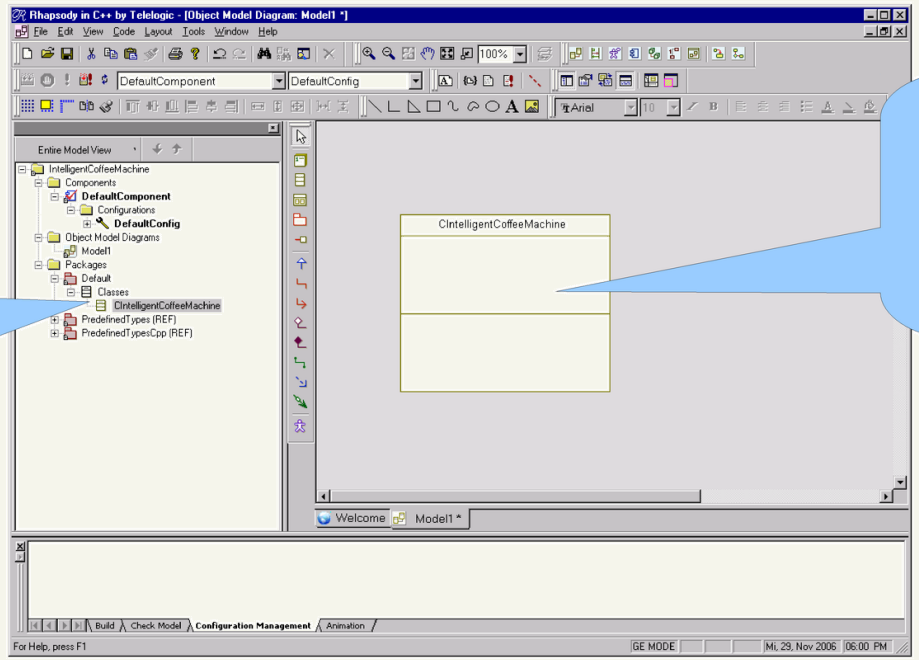
The browser shows everything that is in the model

Universität Bremen

Institute of Automation

Rhapsody – first steps
#2: Create a new class

- In the Object Model Diagram (OMD), use the class icon  to draw a new class
- Name it *CIntelligentCoffeeMachine*



Expand the browser to see the new class ...

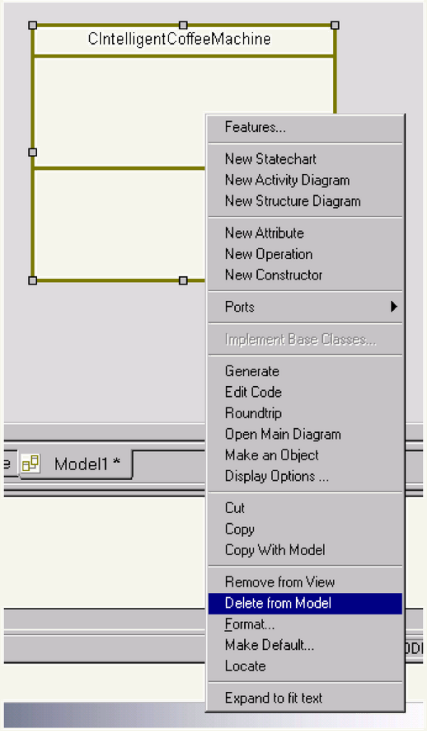
... or right-click on the class and select *Locate* *(short: Ctrl + L)*

Universität Bremen

Institute of Automation

Rhapsody – first steps
Remove from View/Model

- There are two ways of deleting a class:
  - either remove the class from the view (same effect as with “delete” key)
  - delete the class from the model

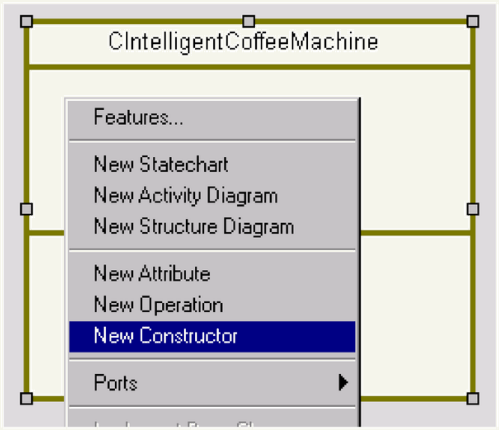


Universität Bremen

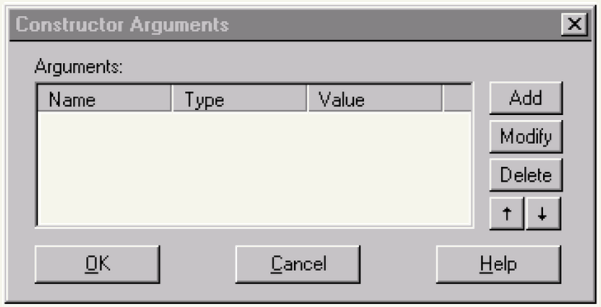
Institute of Automation

Rhapsody – first steps #3: Create a constructor

- A constructor initializes a class (more details later)



- Right-click on the class and choose *New Constructor*
- We need no constructor arguments, so click “OK” in the subsequent dialog

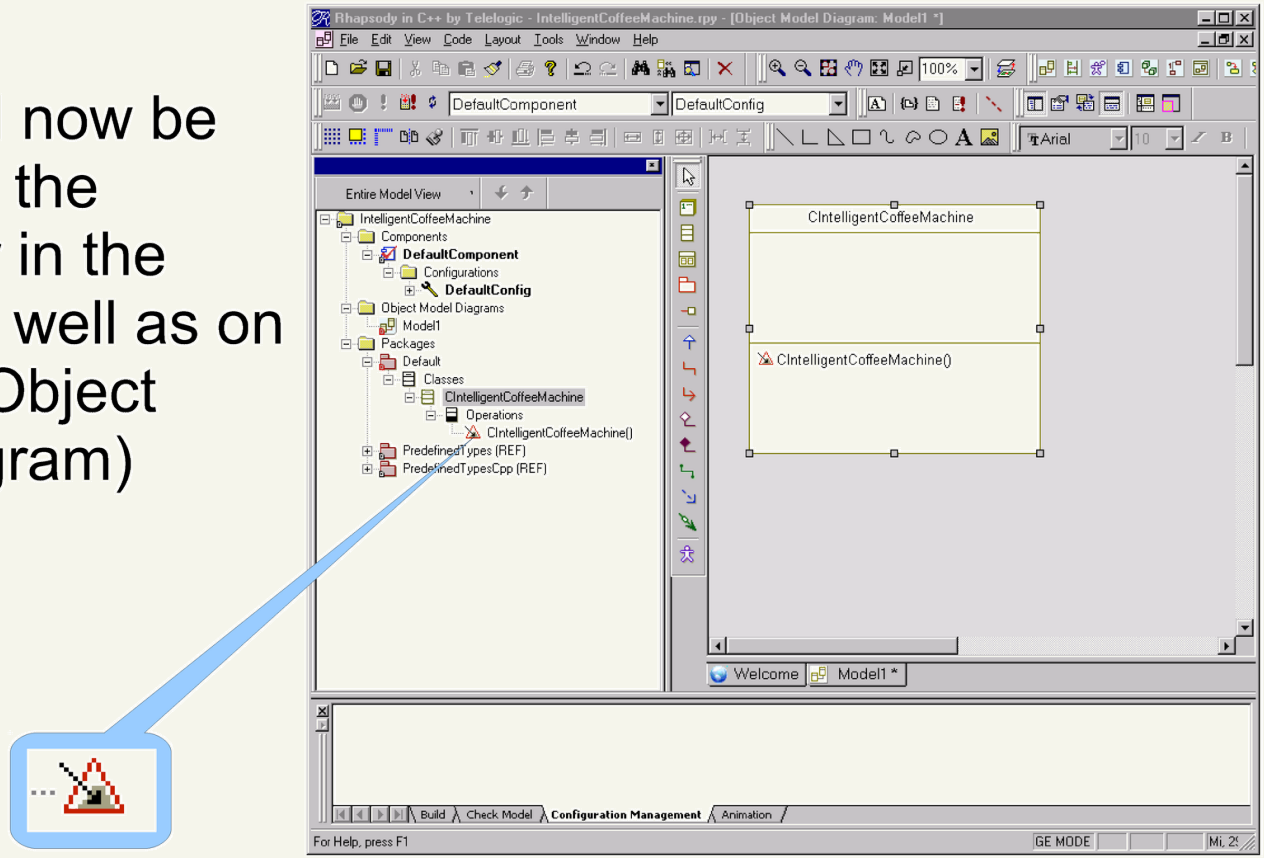


Universität Bremen

Institute of Automation

Rhapsody – first steps
Display constructor

- You should now be able to see the constructor in the browser as well as on the OMD (Object Model Diagram)

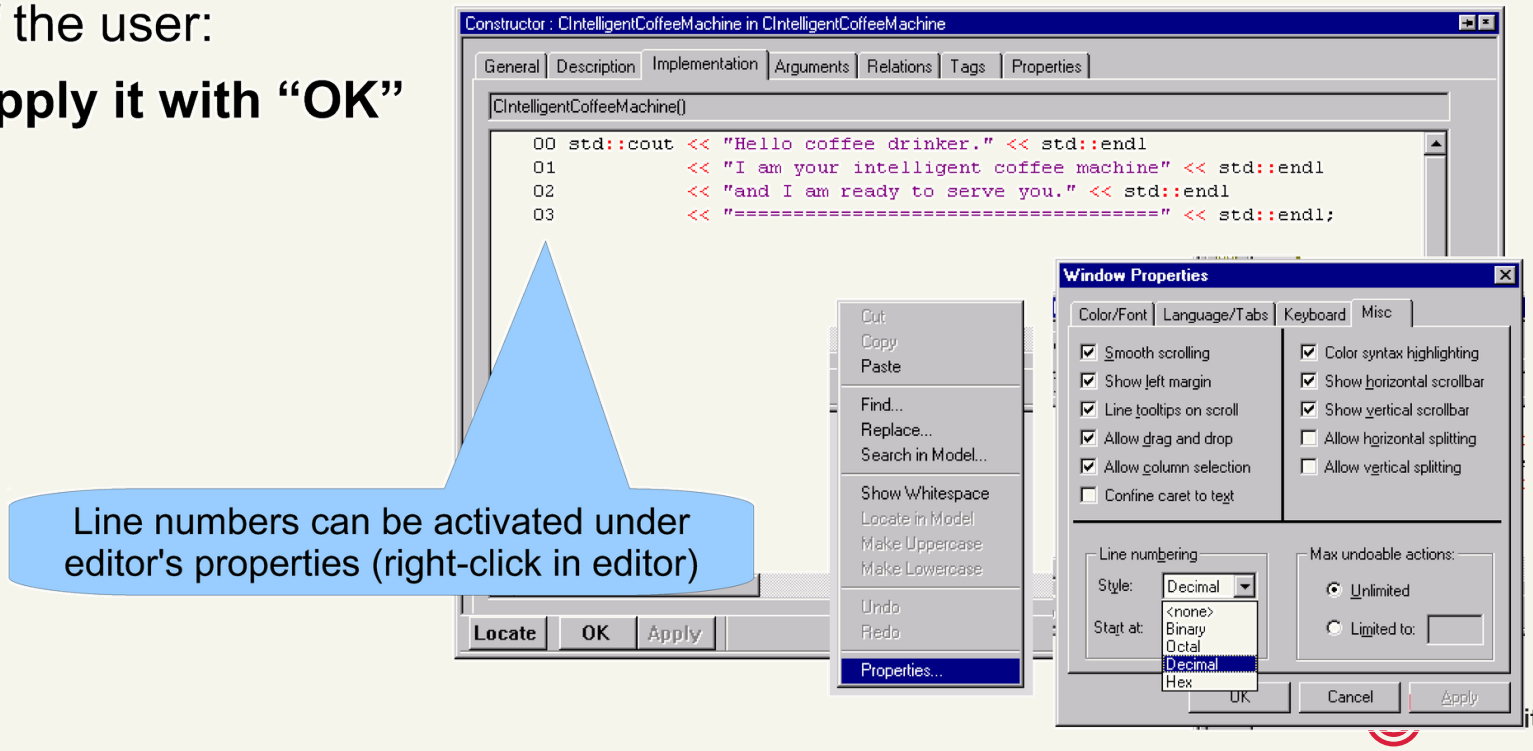


Universität Bremen

Institute of Automation

Rhapsody – first steps
#4: Adding implementation

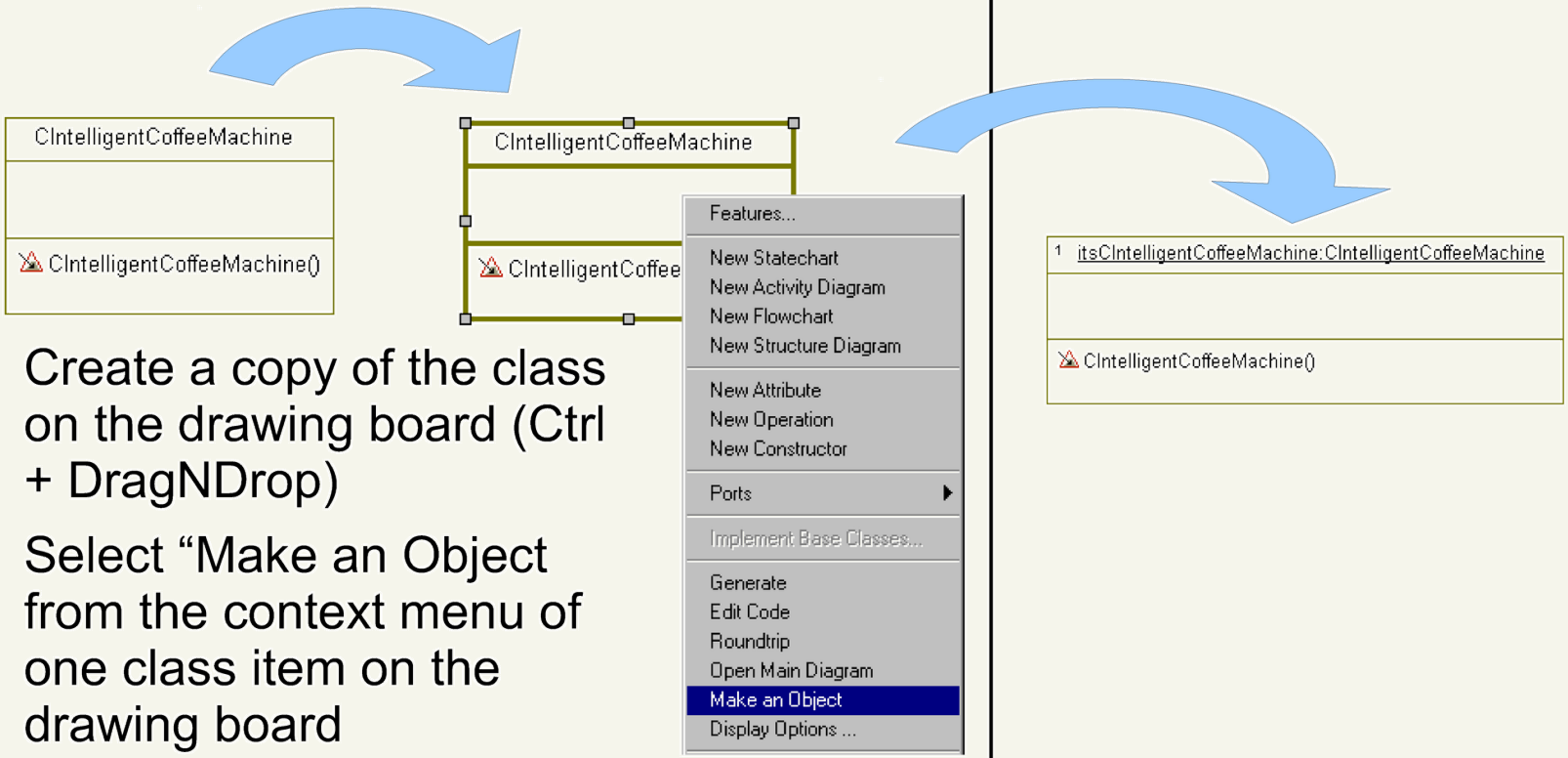
- Select the *CIntelligentCoffeeMachine* constructor in the browser and double-click to open the *Features* dialog.
- Select the implementation tab and enter an implementation for the initialization of the intelligent coffee machine, e.g. a greeting of the user:
- **Apply it with "OK"**



ität Bremen

Institute of Automation

Class ➡ Object: #5: Make an Object



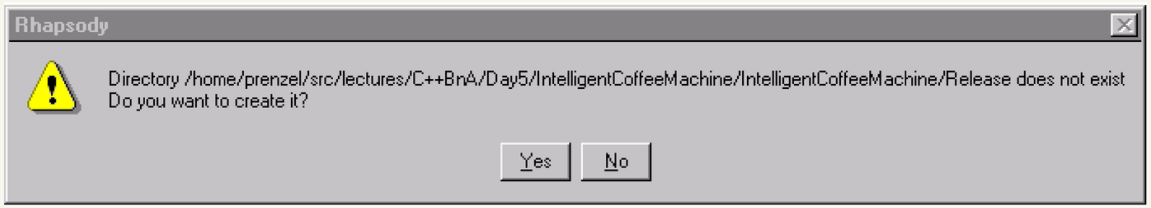
- Create a copy of the class on the drawing board (Ctrl + DragNDrop)
- Select “Make an Object from the context menu of one class item on the drawing board

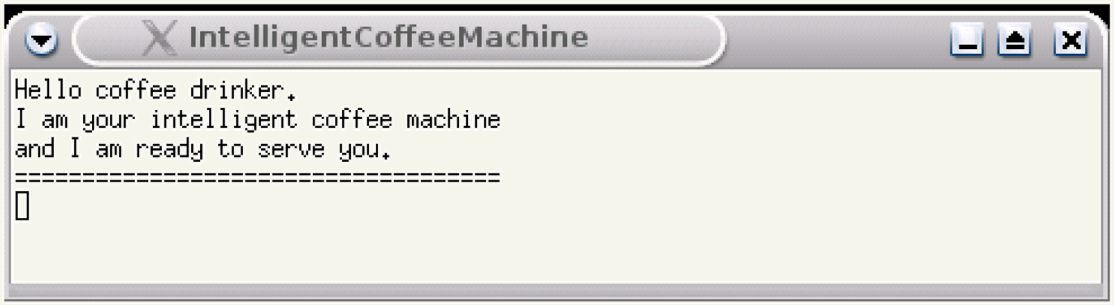
Universität Bremen

Institute of Automation

Rhapsody – first steps
#6: Generate code, build, run ...

- Now, code can be generated
- Save the model 
- Select *Generate/Make/Run* 
- Answer *Yes*

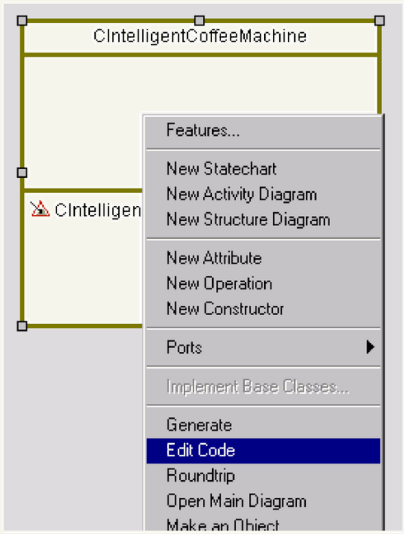




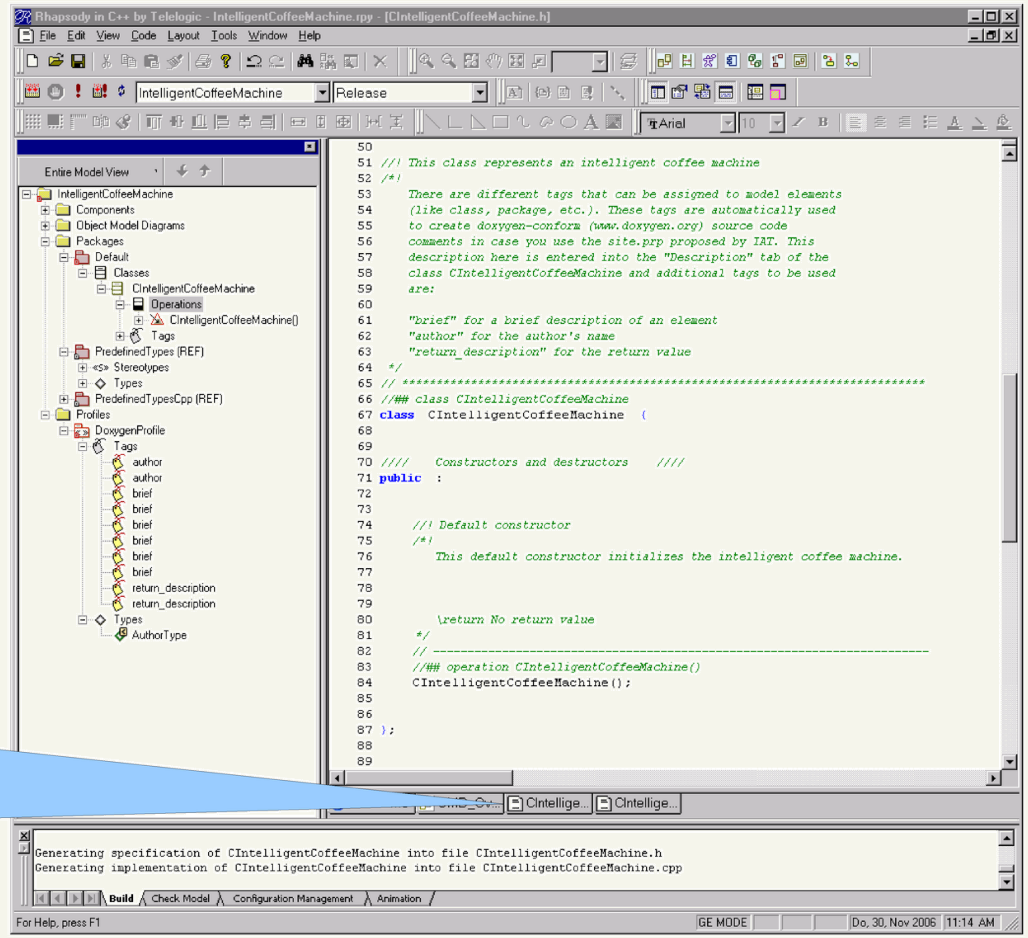
Universität Bremen

Institute of Automation

Rhapsody – first steps
Having a look at the code



Select Edit Code from the class context menu and the header as well as implementation of your class will be displayed (two new tabs at bottom).



Universität Bremen

Institute of Automation

Rhapsody – first steps
Having a look at the code

CIntelligentCoffeeMachine.h

```
//! This class represents an intelligent coffee machine
// ****************************************************
//## class CIntelligentCoffeeMachine
class CIntelligentCoffeeMachine  {


////    Constructors and destructors    ////
public :
    //! Default constructor
    /*!
        This default constructor initializes the intelligent
        coffee machine.


        \return No return value
     */
    // ----------------------------------------------------
    //## operation CIntelligentCoffeeMachine()
    CIntelligentCoffeeMachine();
};
```

Public keyword for defining public interfaces (methods) to the class. Constructors normally have to be public.

A constructor is similar to a function but
- **has no return value**
- has the **class name as function name**

A default constructor has no arguments. More about constructors in the next lecture ... .

Universität Bremen

Institute of Automation

Rhapsody – first steps
Having a look at the code

CIntelligentCoffeeMachine.cpp

In the implementation file the **class name + "::" + method name** (in this case again the class name, due to defining the constructors body) have to be specified.

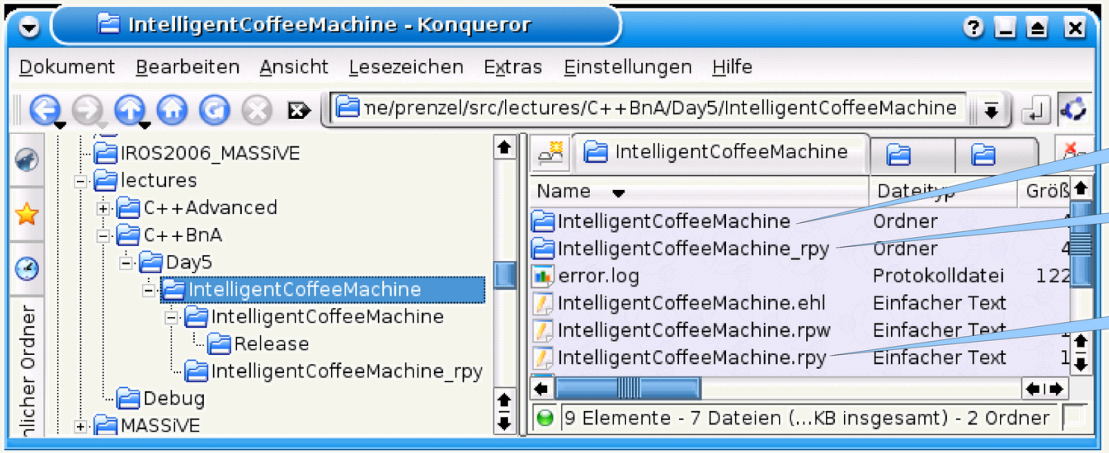
Implementation

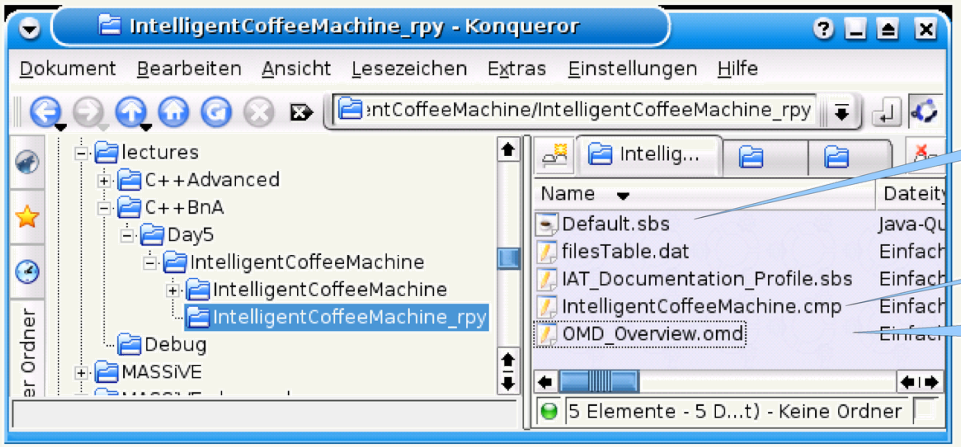
```
//---------------------------------------------------------
// CIntelligentCoffeeMachine.cpp
//---------------------------------------------------------
//## package Default
//## class CIntelligentCoffeeMachine
CIntelligentCoffeeMachine::CIntelligentCoffeeMachine() {
    //#[ operation CIntelligentCoffeeMachine()
    std::cout << "Hello coffee drinker." << std::endl
              << "I am your intelligent coffee machine" << std::endl
              << "and I am ready to serve you." << std::endl
              << "====================================" << std::endl;
    //#]
}
// ---------------------------------------------------------------
// End of CIntelligentCoffeeMachine.cpp
// ---------------------------------------------------------------
```

Universität Bremen

Institute of Automation

Rhapsody – first steps
The model files and generated files





Component directory (generated files)

Model directory

Project file

Package file

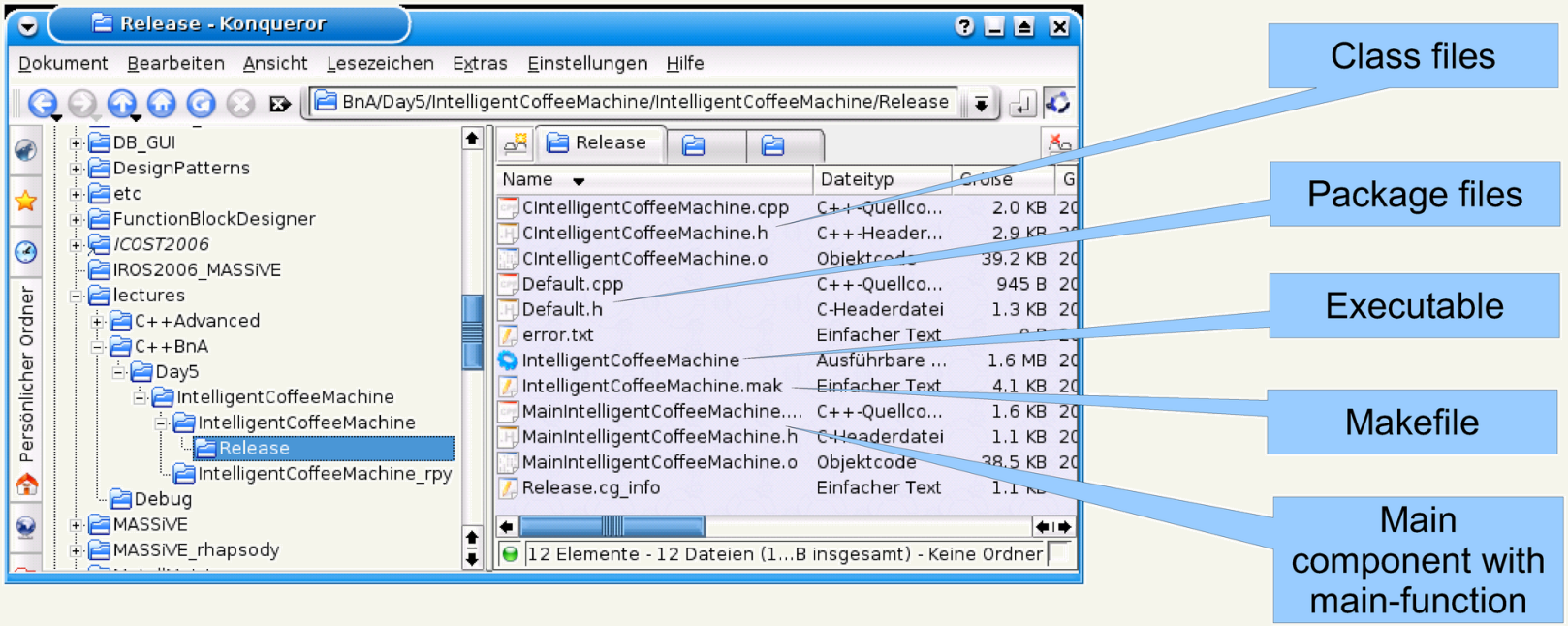
Component file

OMD file (Object modeldiagram)

Universität Bremen

Institute of Automation

Rhapsody – first steps
The model files and generated files



Universität Bremen

Institute of Automation

Rhapsody – first steps
Bi-directional model-code associativity

- The code in the generated files can be modified:
  - Change the constructor's implementation in CIntelligentCoffeeMachine.cpp
  - Change the window focus to some other window and the changes will be roundtripped into the model

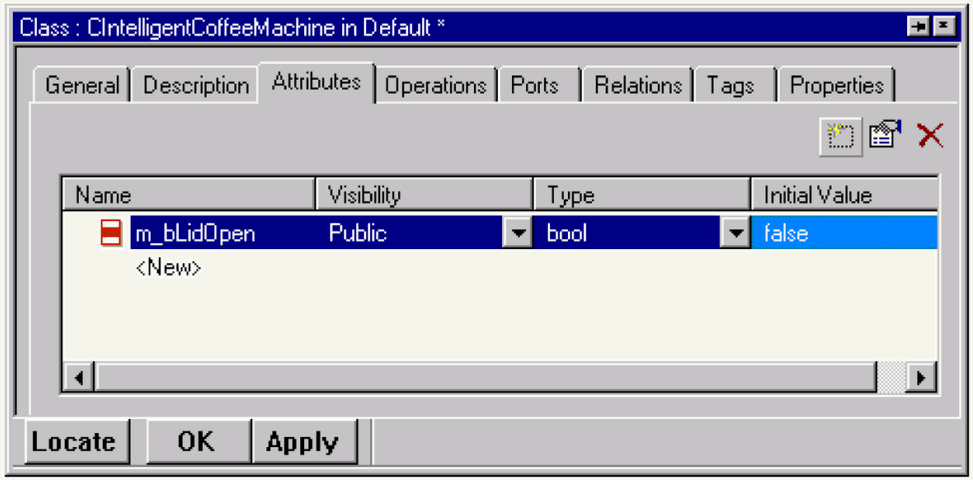
- Even though the roundtripping works very well in general be aware that not everything can be roundtripped!

Universität Bremen

Institute of Automation

Rhapsody – next steps
Adding attributes

- We now want to add an attribute to the intelligent coffee machine that indicates the opening state of the water reservoir's lid:
  - bool m_bLidOpen, with the default value “false”
- Double click on the class in the model to bring up the features dialog and select the *Attributes* tab



Universität Bremen

Institute of Automation

Rhapsody – next steps
Generated code for attribute

Select Edit Code or switch to the still open code view

Header file

```
////    Additional operations    ////
public :

    //## auto_generated
    bool getM_bLidOpen() const;

    //## auto_generated
    void setM_bLidOpen(bool p_m_bLidOpen);


////    Attributes    ////
protected :

    //! Describes the opening state of the water reservoir's lid
    bool m_bLidOpen;                    //## attribute m_bLidOp
```

Per default attributes are ***protected*** (cannot be accessed directly from outside) = **encapsulation principle**

Setting initial value in initialization list

Accessor

Mutator

```
CIntelligentCoffeeMachine::CIntelligentCoffeeMachine() : m_bLidOpen(false) {
    //#[ operation CIntelligentCoffeeMachine()
    std::cout << "Hello coffee drinker." << std::endl
                      << "I am your intelligent coffee machine" << std::endl
                      << "and I am ready to serve you." << std::endl
                      << "==================================" << std::endl;
    //#]
}

bool CIntelligentCoffeeMachine::getM_bLidOpen() const {
    return m_bLidOpen;
}

void CIntelligentCoffeeMachine::setM_bLidOpen(bool p_m_bLidOpen) {
    m_bLidOpen = p_m_bLidOpen;
}
```

Implementation file

ität Bremen

Institute of Automation

Rhapsody – next steps
Attribute visibility

- Changing the *Visibility* in the *Attribute Features* dialog changes the accessor and mutator visibility, not the data member visibility!

Note that an accessor's signature includes the ***const***-keyword. This means **this method does not change the class members**. Every method with this characteristic shall be set to ***const*** to prevent later implementation mistakes!!

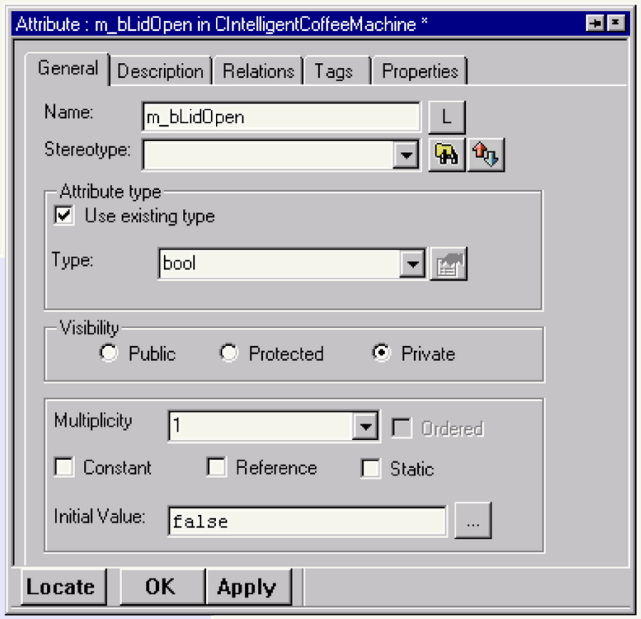
```
////    Additional operations
private :

    //## auto_generated
    bool getM_bLidOpen() const;

    //## auto_generated
    void setM_bLidOpen(bool p_m_bLidOpen);


////    Attributes    ////
protected :

    //! Describes the opening state of the water reservoir's lid
    bool m_bLidOpen;                    //## attribute m_bLidOp
```



Universität Bremen

Institute of Automation

Rhapsody – next steps
Adding a method

- Now try on your own to add the method
  bool InsertCoffee(unsigned int NumberOfSpoons)
  (use "New Operation")

  where
  - *NumberOfSpoons* is a "In"-Parameter
    (remember the concept call-by-value/call-by-reference/pointer and see next slide ...)
  - An internal state variable for the content of the coffee reservoir is set according to the number of spoons
  - The return value is set according to the success of the operation, i.e.
    - has the lid been opened before?
    - is the maximum number of spoons (introduce this constant) respected?

Universität Bremen

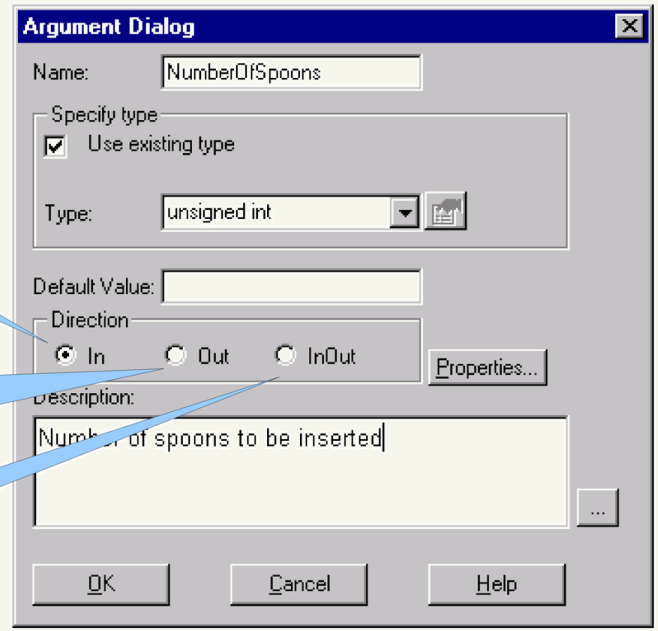
Institute of Automation

Rhapsody – next steps
Argument direction

Call by value: Parameter remains **unchanged** in the outside scope

Call by reference / pointer (depends on settings): Parameter may be **changed** within the method. Parameter is **not used to hand over a value** to the method.

Same as before, but parameter also **serves as in value** to the method.



Universität Bremen

Institute of Automation

Creating initial instances

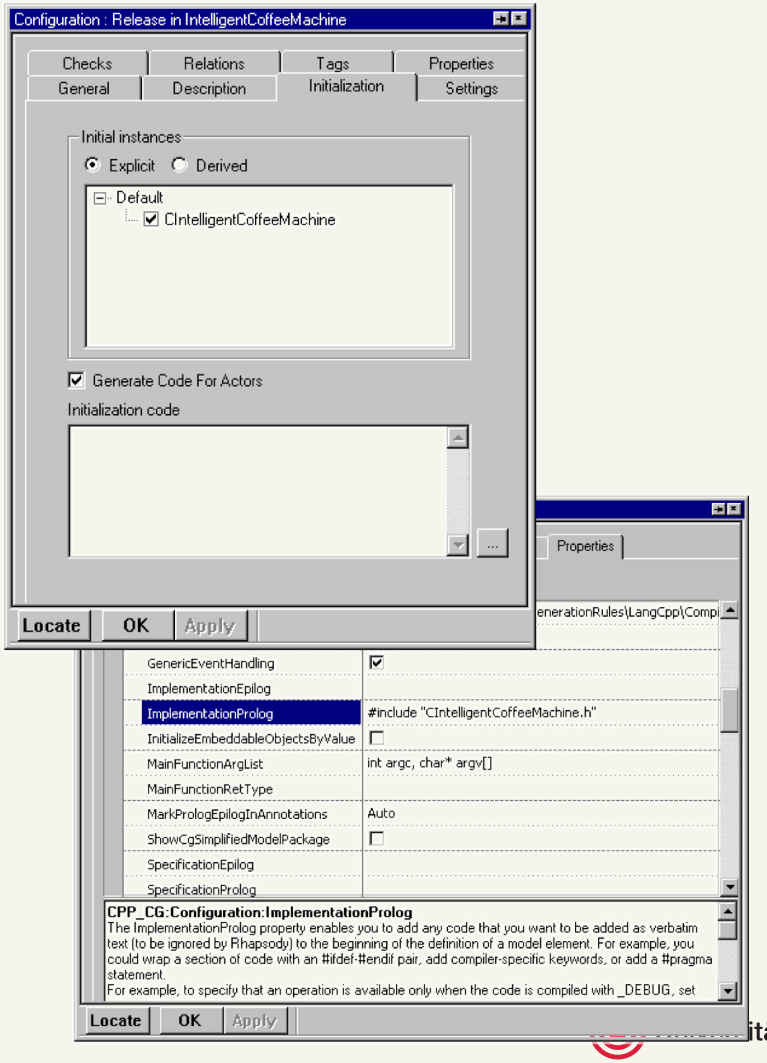
- In the lecture module global objects have been created (see “Make Object” slide)
- Another way to influence object instantiation is the Initialization section of the Component's Configuration, see following slide:

Universität Bremen

Institute of Automation

Initial instance

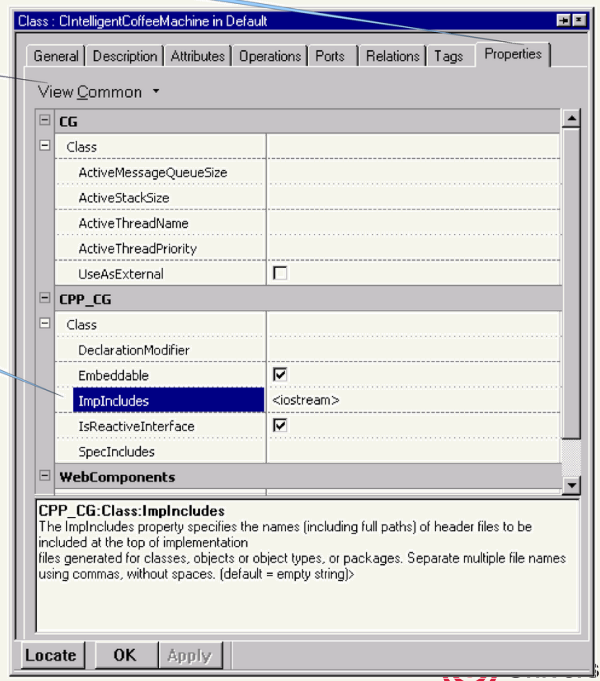
- In the Initialization tab of the *Configuration*, expand the package containing your desired class and select e.g. *CIntelligentCoffeeMachine* class.
- This means that the main will now create an initial instance (an object) of the class.
- If you want to pass arguments to the constructor of your class, deactivate “Generate Code for Actors”, insert initialization code manually and add **#include “Classname.h”** into the “Implementation Prolog” in the Properties under CPP_CG::Configuration (see right bottom)



Institute of Automation

Including of headers

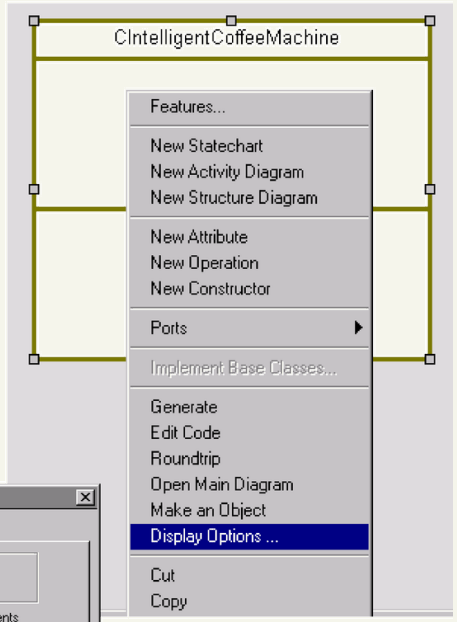
- In the browser, double click on the *CIntelligentCoffeeMachine* class to bring up the features dialog.
- Select the *Properties* tab, ensure that the *Common* filter is selected and enter <*libname*> into the
  - *ImpIncludes* (=Implementation Includes) property to create inclusion in the cpp or
  - *SpecIncludes* (=Specification Includes) property to create inclusion in the header



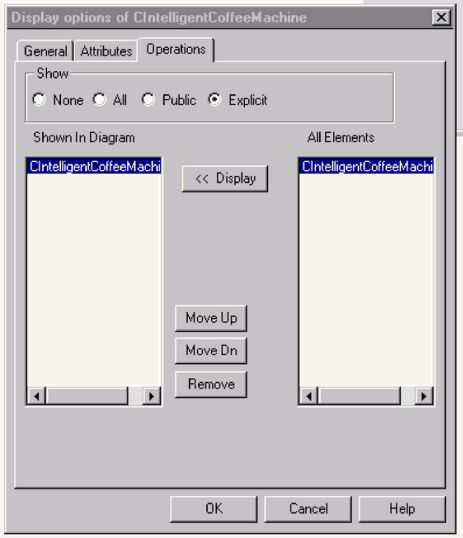
Institute of Automation

Adjust display options

- To adjust the display options of the class on the OMD, select *Display Options* from the context menu and set the option for operations to *All* or *Explicit*

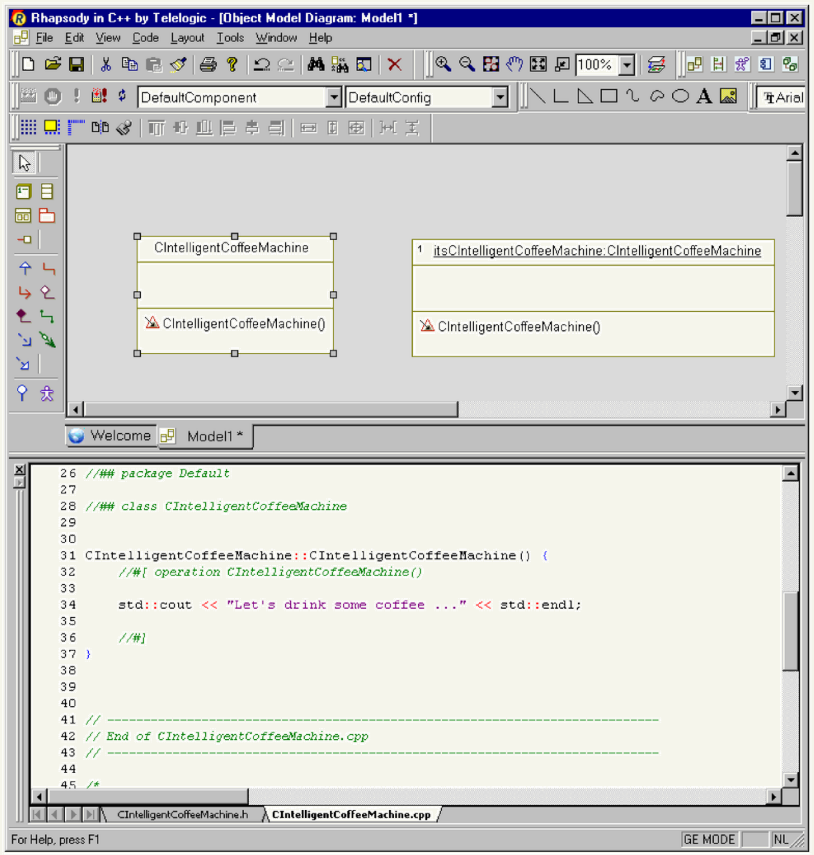


- It is also possible to set default display options



Universität Bremen

Active Code View



- To activate: “Alt + 2” or View/ActiveCodeView
- Source code focus is kept on activated model element

Institute of Automation

Hotkeys

- Ctrl + L:Locate in Browser
- Alt + 2: Active Code View
- Ctrl + Space (in Code): Auto-completion
- Ctrl + DragNDrop: Copy elements (in model and diagram)
- Alt (+ Ctrl) + resize a package on diagram:
  Do not resize elements within

Universität Bremen

Institute of Automation

Displaying the Main and Makefile

- Right-click on the configuration and select *Edit Makefile / Configuration Main File*

Universität Bremen

Institute of Automation

Add model elements to project

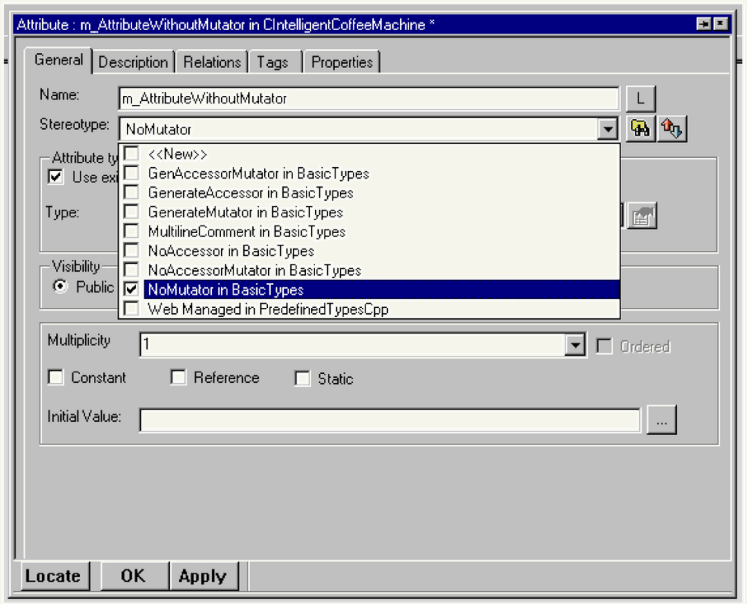
- In the menu, go to File/AddToModel
- Select the correct file type in the appearing dialog (e.g. .sbs for packages)
- Click on “Open” button to add the element to the model

Universität Bremen

Institute of Automation

Using stereotypes

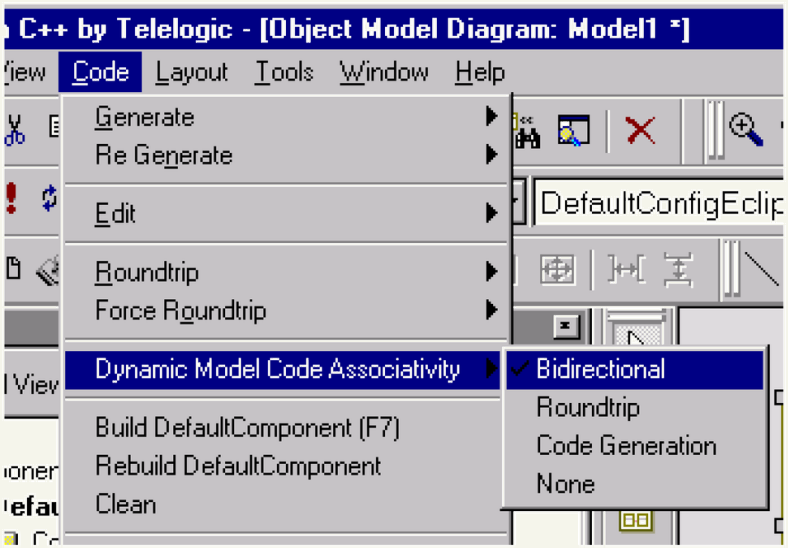
- Stereotypes simplify the application of certain settings to model elements
- The “BasicTypes.sbs” package (in elearning system) provides e.g.
  - Control of Accessors/Mutators
  - Exclusion of Rhapsody framework (to have code that is compilable without Rhapsody)
  - ...
- Add “BasicTypes” package to your project to use stereotypes



Universität Bremen

Institute of Automation

Bi-directional model-code associativity



Try it out ...

- Bidirectional
  - Changes in either the model or the code are roundtripped automatically
- Roundtrip
  - Changes in the code are updated automatically
- Code Generation
  - Changes in the model are updated automatically
- None
  - Nothing is done automatically

Universität Bremen