Institute of Automation

C++ Basics and Applications in technical Systems

Lecture 6 - Constructors and inheritance

Institute of Automation
University of Bremen

30th 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

Classes and objects

Institute of Automation

Keep the difference between class and object in mind!

Example

`std::string sMyString;`

`std::string` is a class

Example

`std::string sMyString;`

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

Universität Bremen

Constructors

Institute of Automation

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)

Universität Bremen

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

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

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

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

Destructor

Institute of Automation

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

Rules for manual implementation:

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

Universität Bremen

Access restrictions on members

Institute of Automation

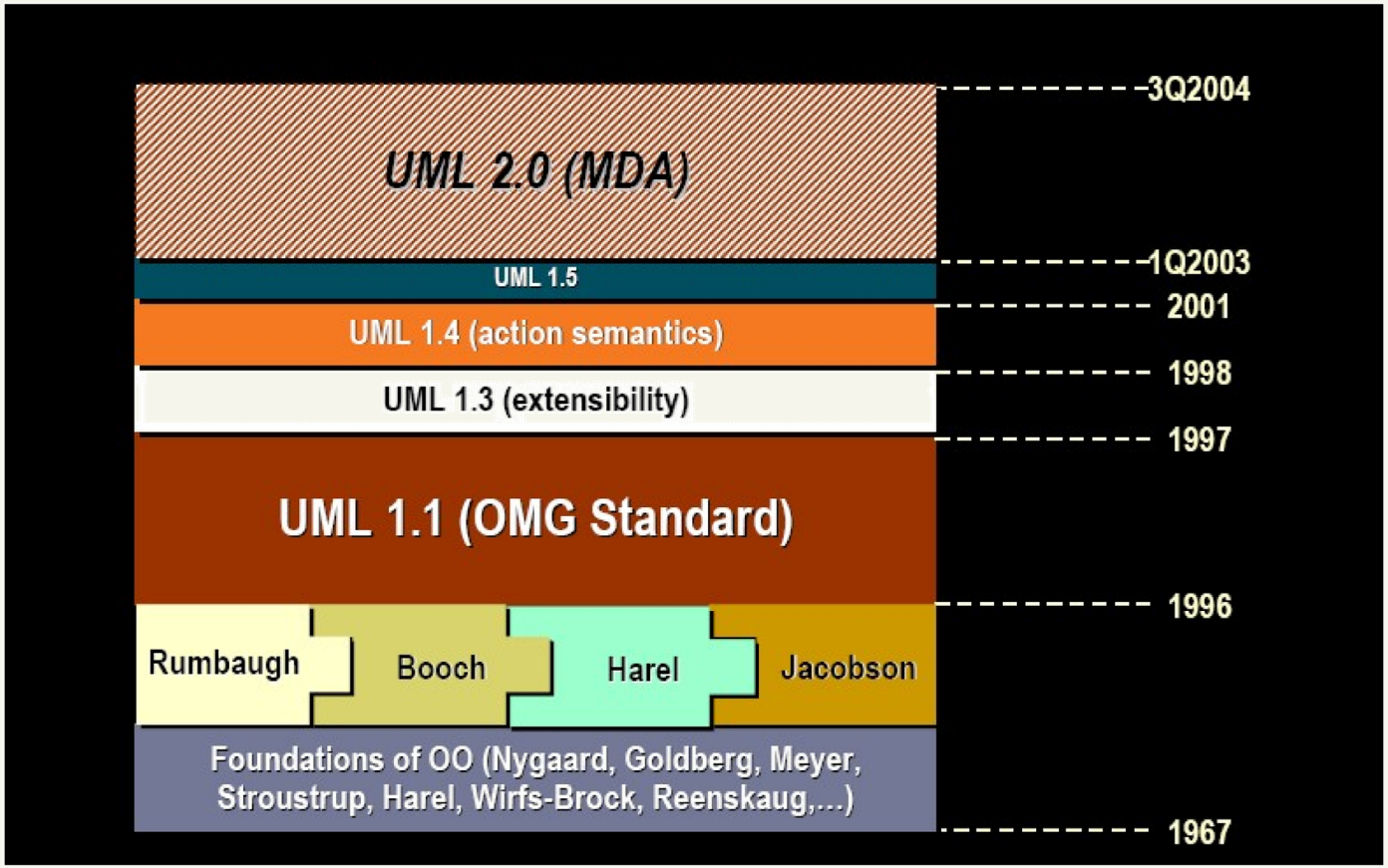
Attributes and methods fall under one of three different access permissions:

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

Universität Bremen

Institute of Automation

UML (Unified Modeling Language) – the history



Universität Bremen

Institute of Automation

UML – The mission

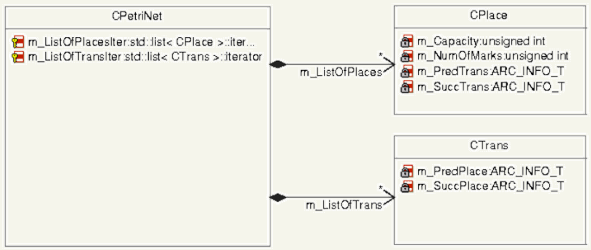
- UML is a **standardized modeling language** consisting of an integrated set of diagrams
- Developed to help system and software developers accomplish the following tasks
  - Specification
  - Visualization
  - Architecture design
  - Construction
  - Simulation and Testing
- Allows the developer to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

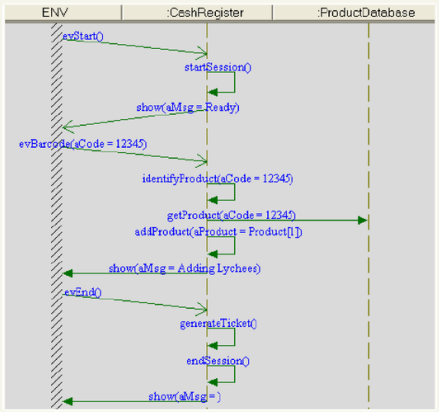
Universität Bremen

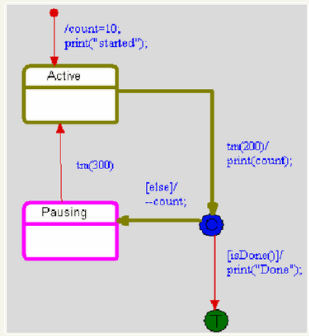
Institute of Automation

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**





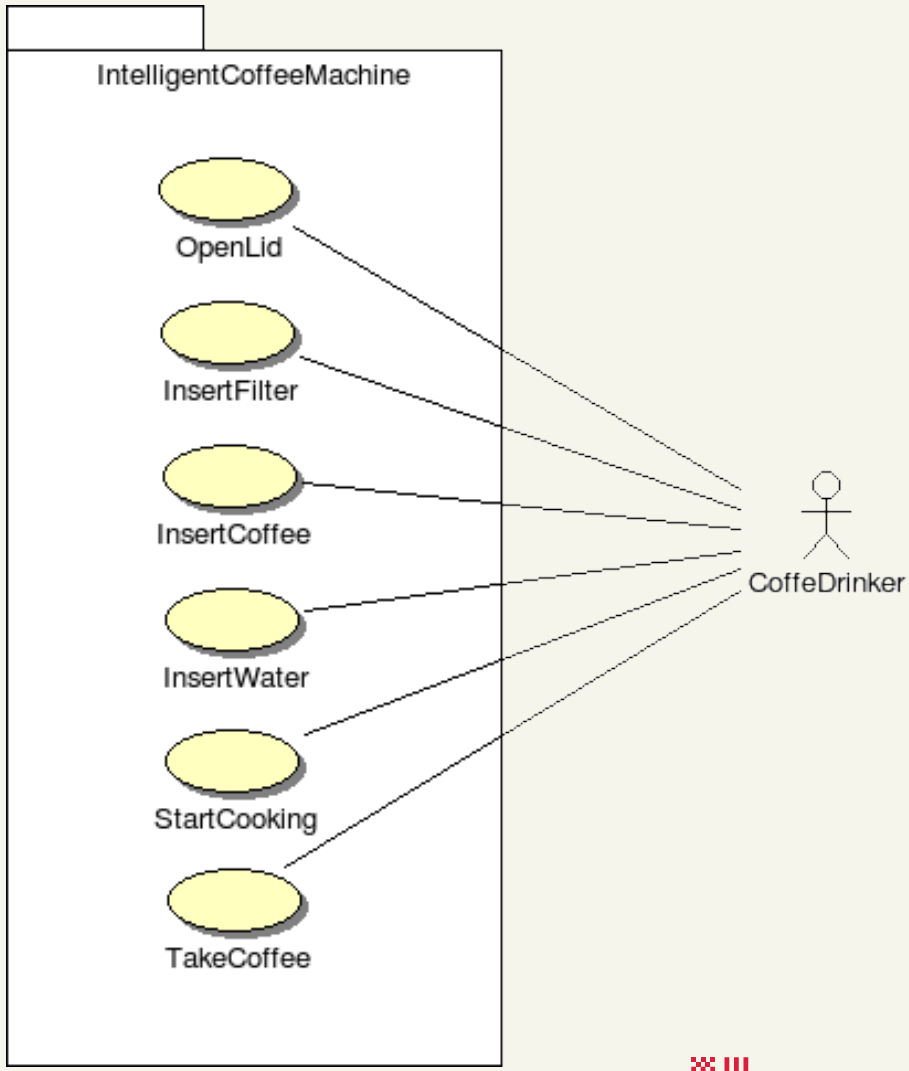


Good online summary on UML:
http://bdn.borland.com/article/31863

Universität Bremen

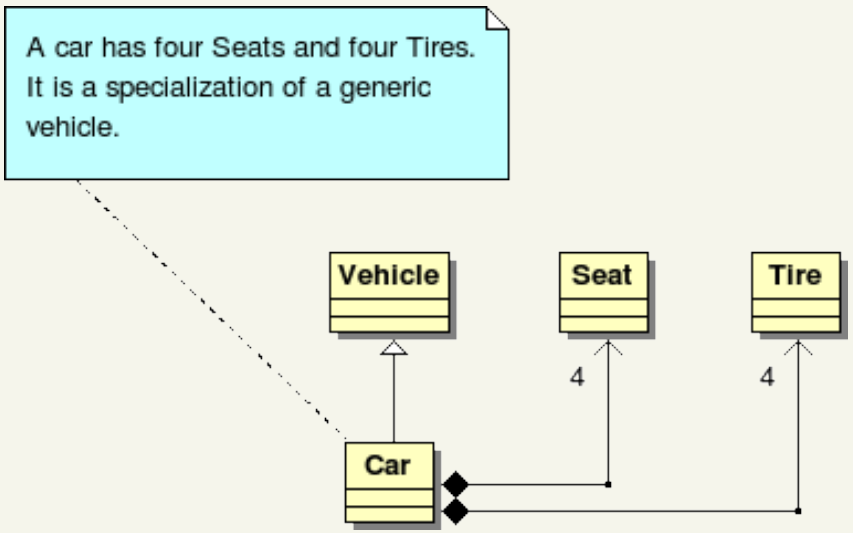
Use-Case diagram

- presents a graphical overview of the functionality provided by a system
- shows actors
- shows goals of actors (use-cases)
- shows dependencies between actors and use-cases.



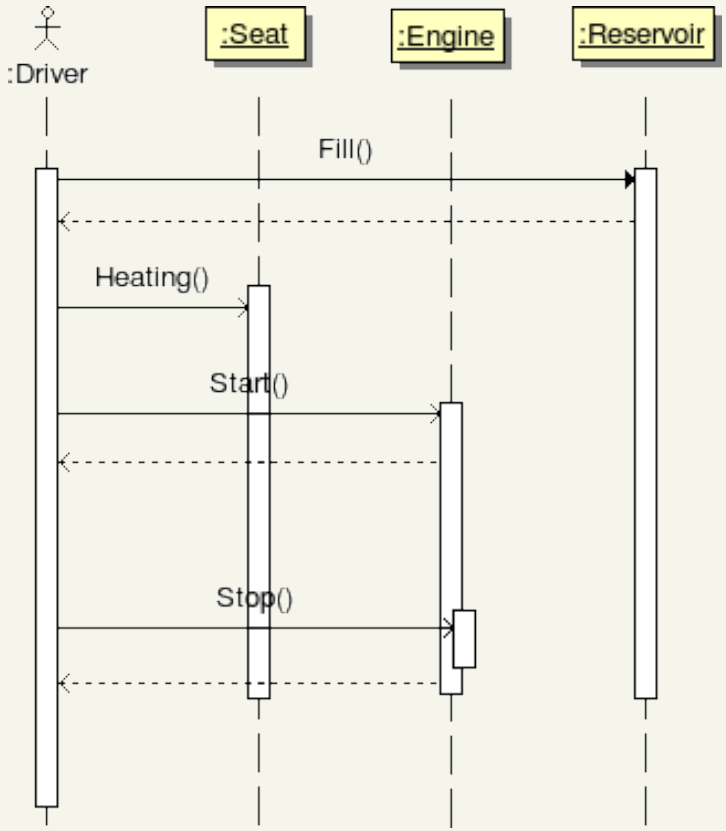
Class diagramm

- diagram of static structure
- depicts the system's classes
- depicts class attributes
- depicts the relationships between classes



Sequence diagram

- interaction diagram
- shows how processes operate with one another
- shows in which order proccesses interoperate
- shows a snapshot of one possible interaction sequence

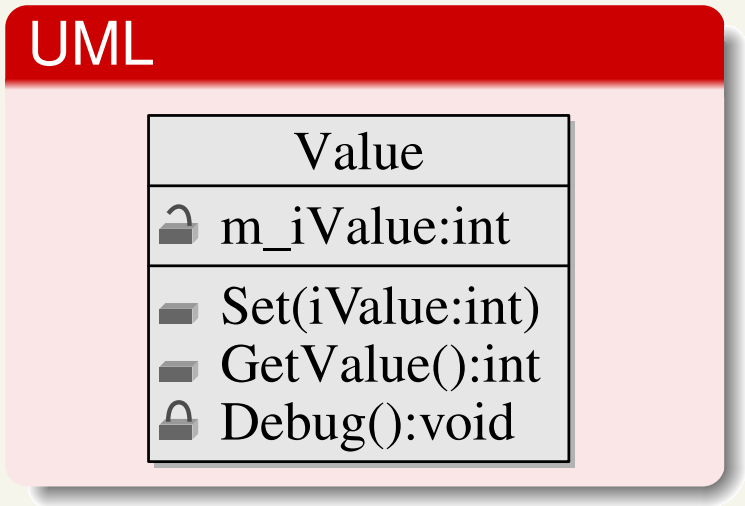


Visibility of elements

Institute of Automation

The visibility of elements of a class declared as `public`, `protected` or `private` are depicted via symbols. Those symbols may vary between different UML tools.

```
class Value
{
protected:
  int m_iValue;

public:
  void Set(int iValue);
  int GetValue();

private:
  void Debug(void);
};
```

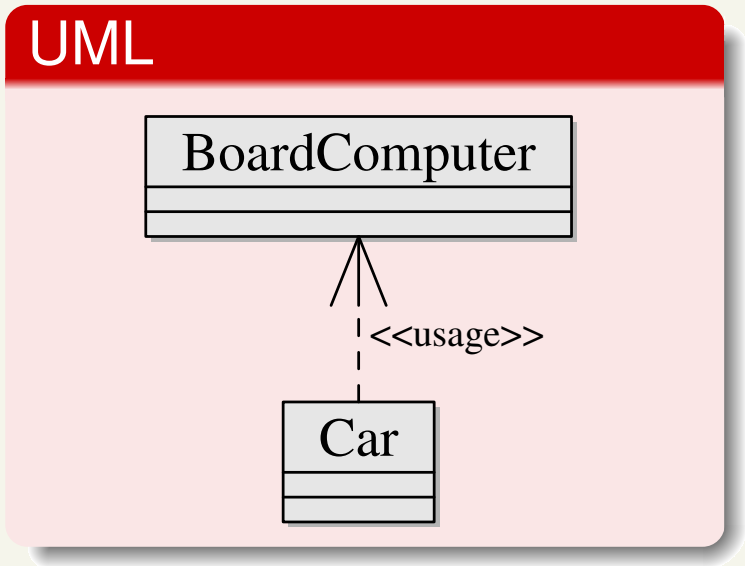


Universität Bremen

Dependency

Dependencies are C++ `#include`-statements. In this example class `Car` uses class `BoardComputer` which is declared in file `BoardComputer.h`. It depends on it, meaning that a change in the class `BoardComputer` may cause a change in the class `Car`[1].

```
#include "BoardComputer.h"

class Car
{
public:
  void Start()
  {
    BoardComputer myComputer;
    myComputer.Enable();
  }
};
```

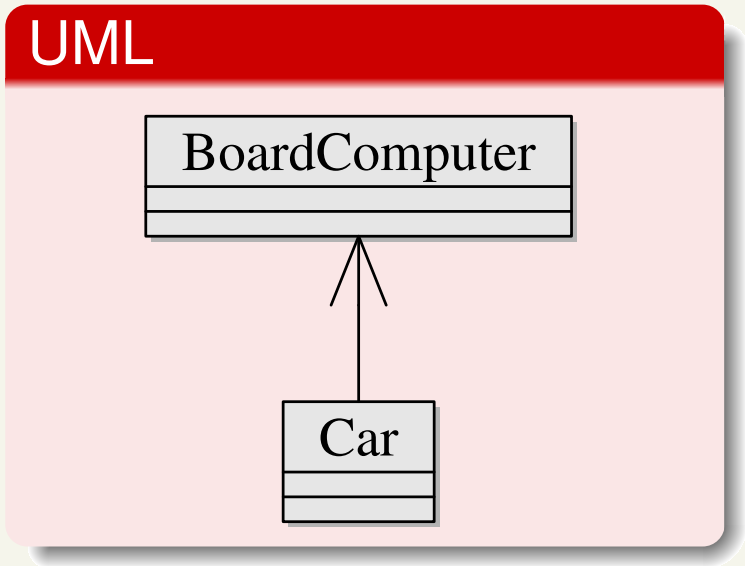


[1] This is only true as long as each class is stored in it's own module!

Association

If a class `BoardComputer` is associated[2] to a class `Car`, this means that the class `Car` has access to an object of type `BoardComputer` via a pointer. It does not own this linked object, so it must not e.g. delete it.

```
class Car
{
public:
  void TestBoardComputer(
   BoardComputer * pComputer);
};
```

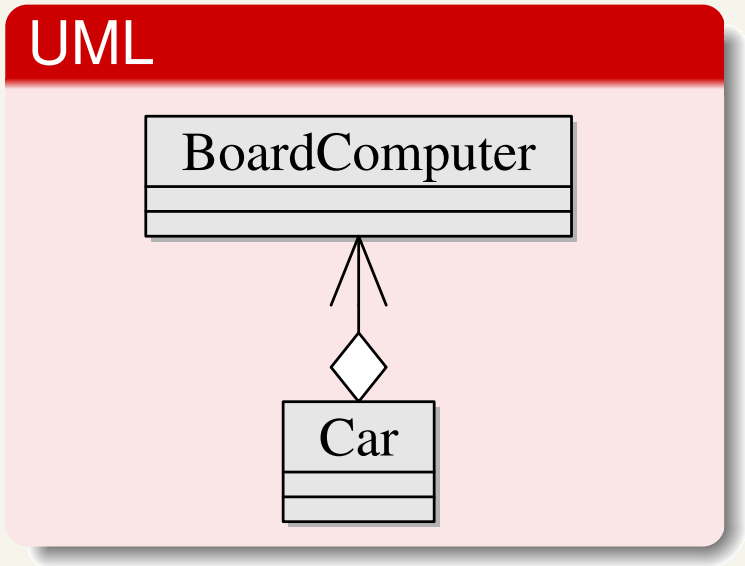


[2] Association is a special form of dependency!

Aggregation

If a class `BoardComputer` is aggregated[3] to a class `Car`, this means that class `BoardComputer` is a member of class `Car`. The ownership is via a pointer, so the linked object must be deleted by the class `Car` after usage.

```
class Car
{
public:
  void SetBoardComputer(
   BoardComputer * pComputer);

private:
  BoardComputer * m_pMyComputer;
};
```

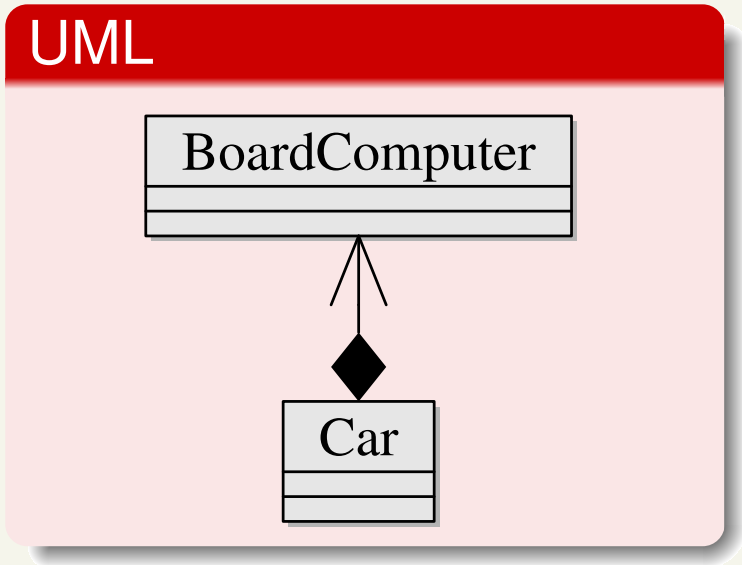


[3] Aggregation is a special form of association!

Composition

If a class `BoardComputer` is composited[4] by a class `Car`, this means that class `BoardComputer` is a member of class `Car`. The ownership is via an actual object (not a pointer).

```
class Car
{
private:
  BoardComputer m_MyComputer;
};
```

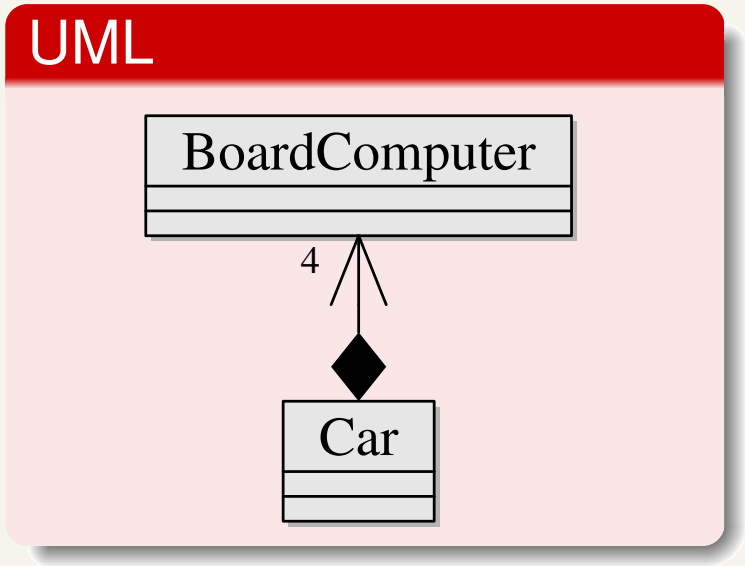


[4] Composition is a special form of association!

Multiplicity (fixed)

The multiplicity has to be defined for associations. If no multiplicity is given a value of 1 is assumed. A multiplicity other than one is used for one-to-many associations.

```
class Car
{
private:
  BoardComputer m_MyComputers[4];
};
```

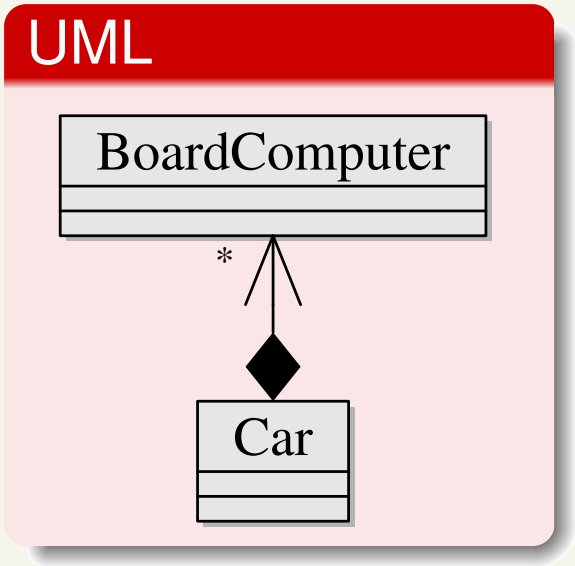


Multiplicity (unbounded)

Institute of Automation

The multiplicity has to be defined for associations. If no multiplicity is given a value of 1 is assumed. A multiplicity other than one is used for one-to-many associations.

```
class Car
{
private:
  std::vector<BoardComputer> m_MyComputers;
};
```

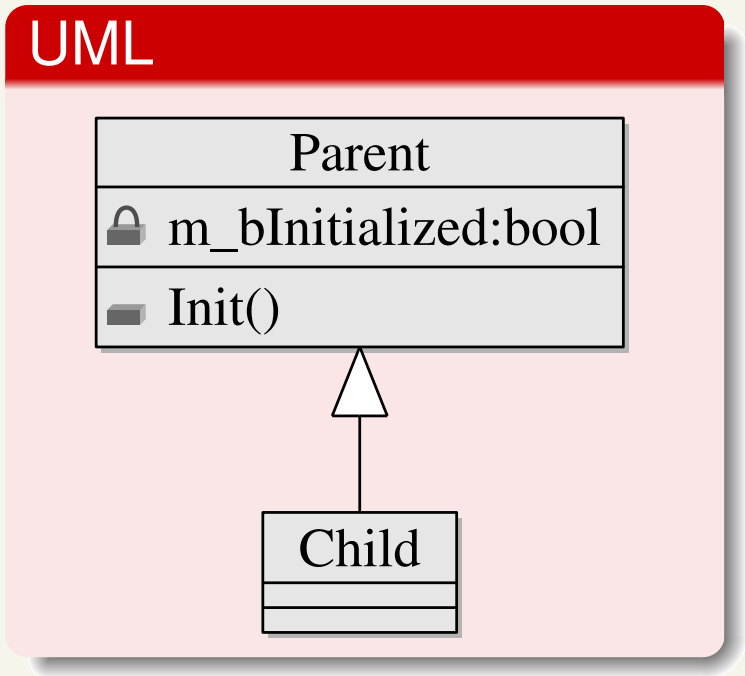


Universität Bremen

Inheritance

The class `Child` inherits all elements of class `Parent`.
Inheritance is explained later in this lecture.

```
class Parent
{
public:
  void Init(void);

private:
  bool m_bInitialized;
};

class Child
  : public Parent
{
};
```

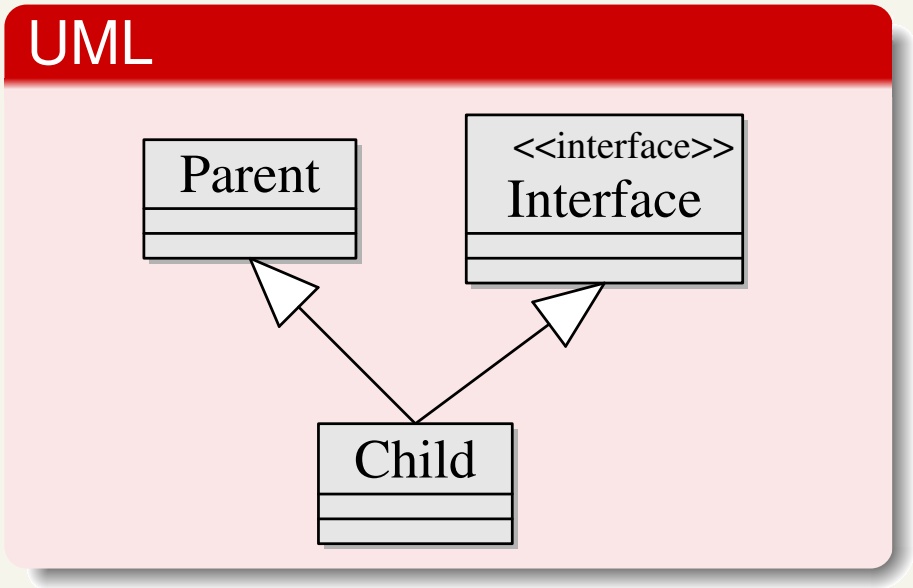


Multiple inheritance

The class `Child` inherits all elements of class `Parent` and class `Interface`.

The concept of an interface is explaned in a later lecture.

```
#include "Interface.h"

class Parent
{
};

class Child
  : public Parent,
    public Interface
{
};
```



Available CASE-tools

Institute of Automation

There are several CASE-tools (tools for Computer-Aided Software Engineering) available. We will present here two of them. Within CASE-tools the software development is based on UML-diagrams and the C++ code is generated.

IBM Rational Rhapsody

`http://www.ibm.com/`

- **Commercial** (license available by IAT, contact Henning Kampe)
- **Platform**: Windows (with limits also Linux)

Bouml

`http://bouml.free.fr/`

- **Free** OpenSource (GNU General Public License)
- **Platform**: Windows, Linux, Mac OS

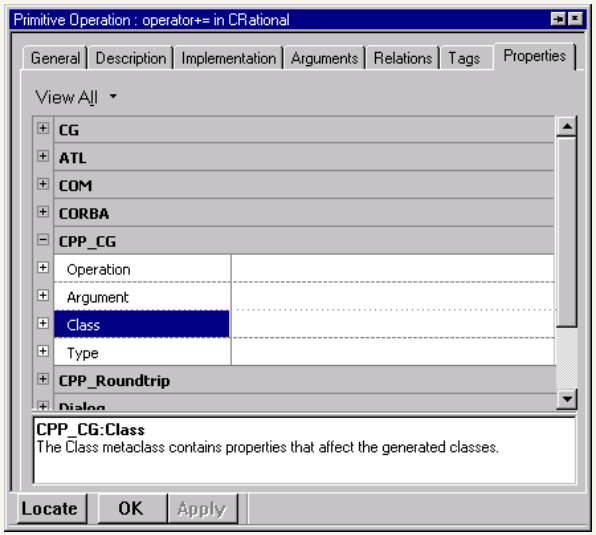
Universität Bremen

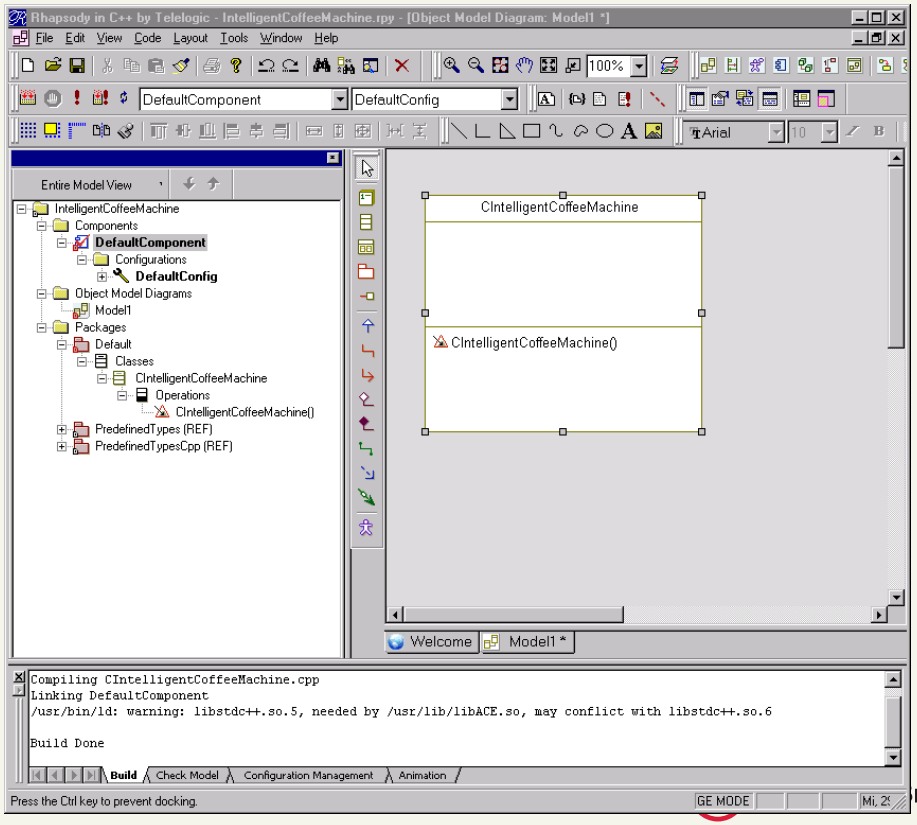
IBM Rational Rhapsody

Institute of Automation

More informations at `http://www-01.ibm.com/software/rational/products/rhapsody/developer/`.
A free version at `http://www.ibm.com/developerworks/downloads/r/modeler/`.

- C++ code generation
- UML diagrams
- ...



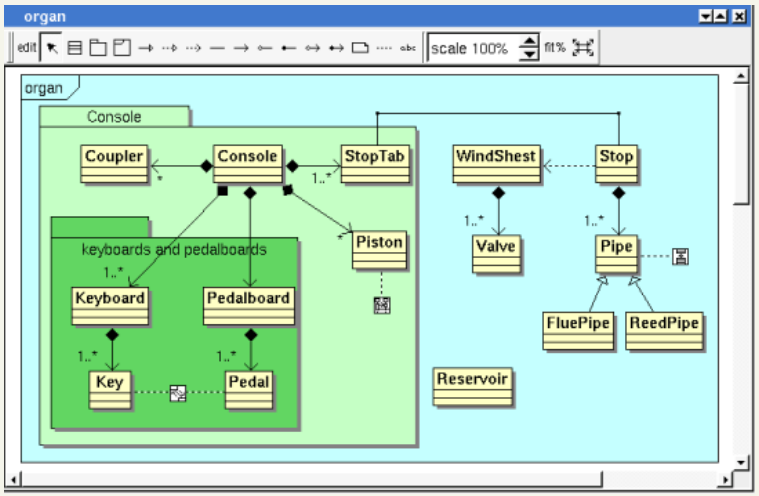


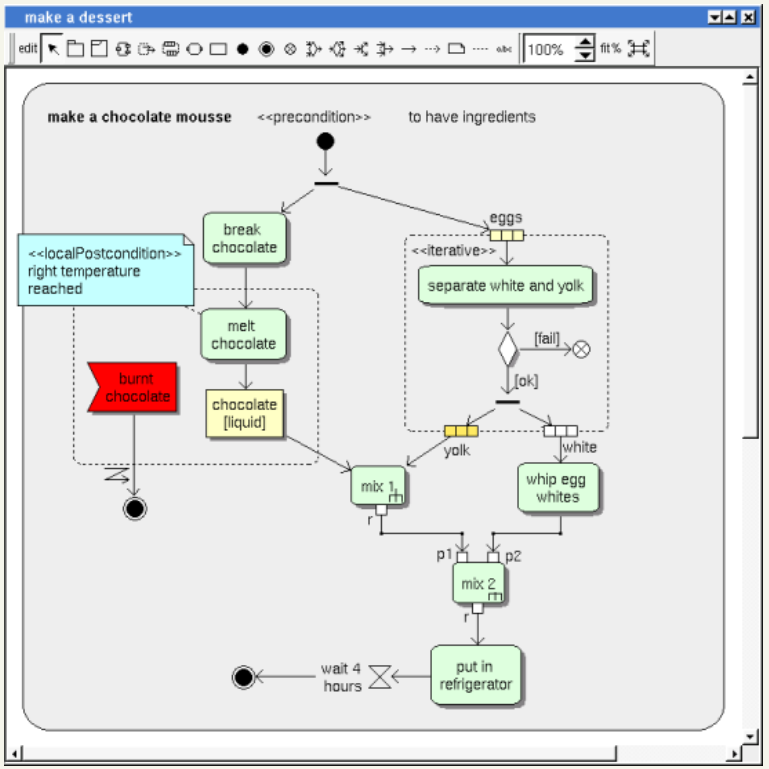
Bouml

Institute of Automation

More screenshots at
`http://bouml.free.fr/screenshots.html`.

- Class diagrams
- State-charts
- ...





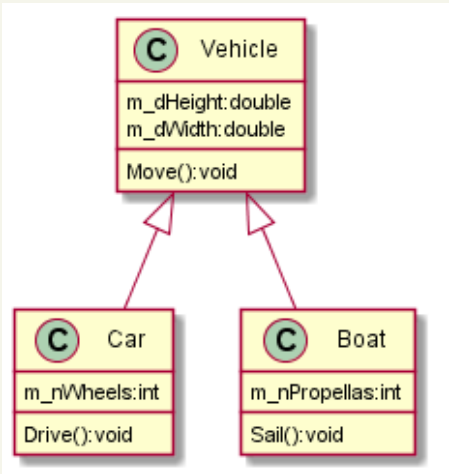
Universität Bremen

Institute of Automation

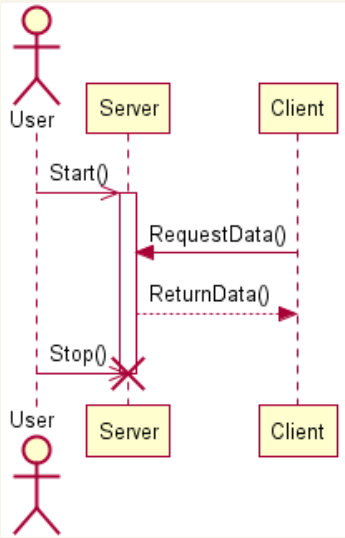
PlantUML

More informations at
`http://plantuml.sourceforge.net/`.

- Class diagrams
- Sequence diagrams
- ...




```
@startuml

Vehicle <|-- Car
Vehicle <|-- Boat

Vehicle : m_dHeight:double
Vehicle : m_dWidth:double
Vehicle : Move():void

Car : m_nWheels:int
Car : Drive():void

Boat : m_nPropellas:int
Boat : Sail():void

@enduml
```




```
@startuml

actor User

User ->> Server: Start()
activate Server

Client -> Server: RequestData()
Server --> Client: ReturnData()

User ->> Server: Stop()
destroy Server

@enduml
```

Universität Bremen

Purpose

Keep in mind

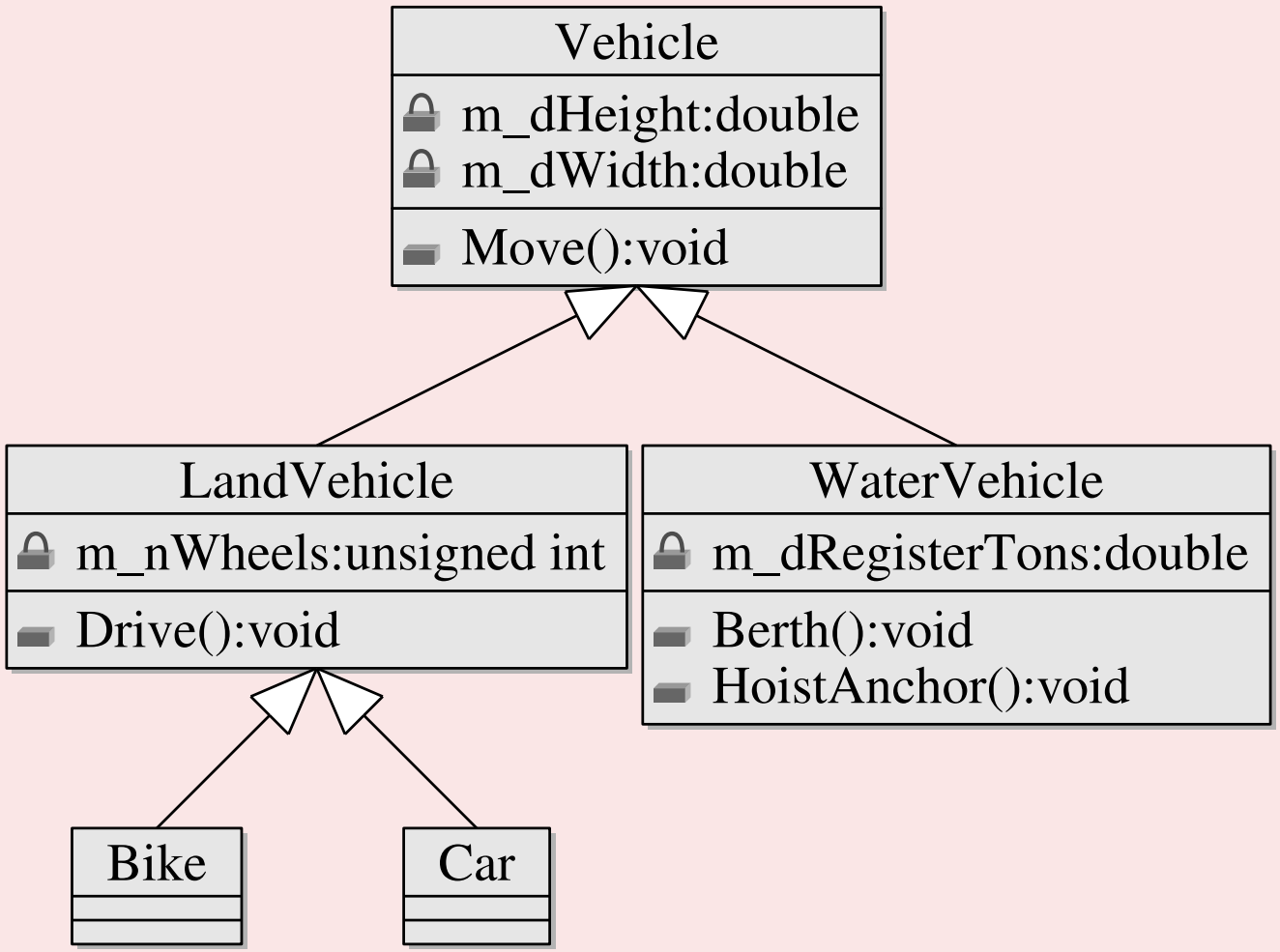
- Inheritance is one of the concepts for software re-usage.
- Inheritance allows us to *use* what already exists and incorporate changes.
- Inheritance also allows us to *extract* commonality from classes into a single inherited class.

  based on:
  - characteristics that are common for elements of a set
  - small differences between the elements
  - hierarchical organization

Inheritance example

Institute of Automation

UML



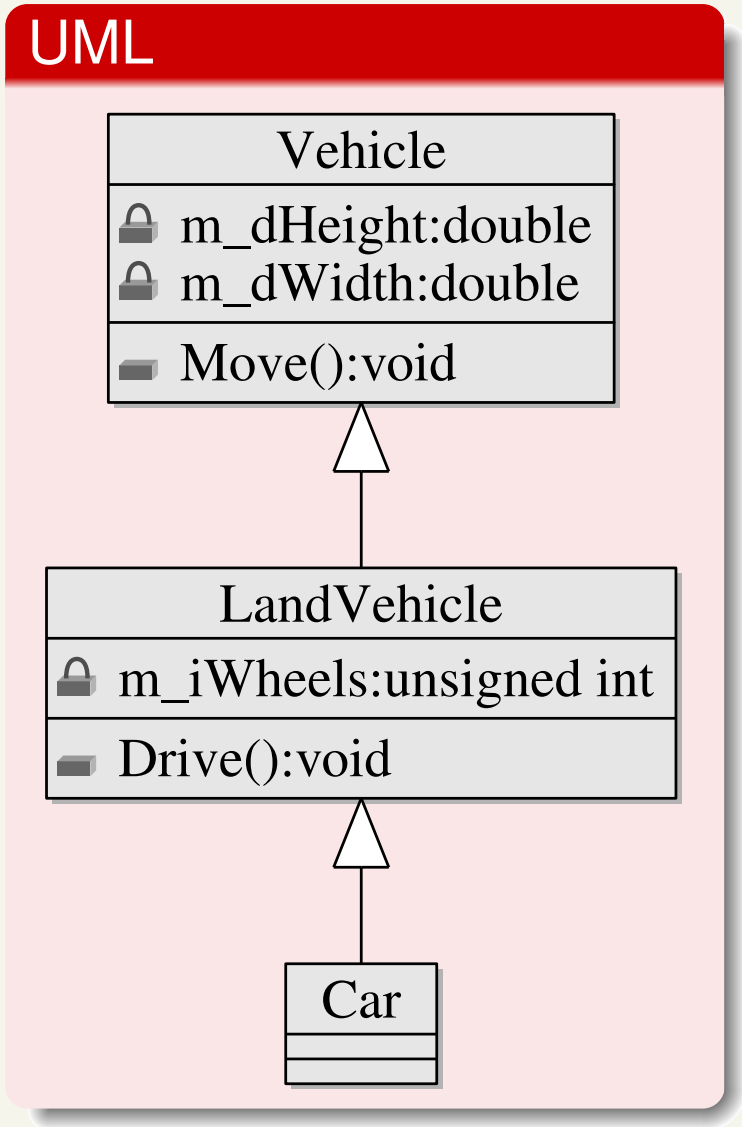
Bremen

Inheritance terminology

- Top level class is called ***base class***.
- The class from which is inherited is called ***parent class***.
- The class that inherits is called ***derived*** or ***child class***.

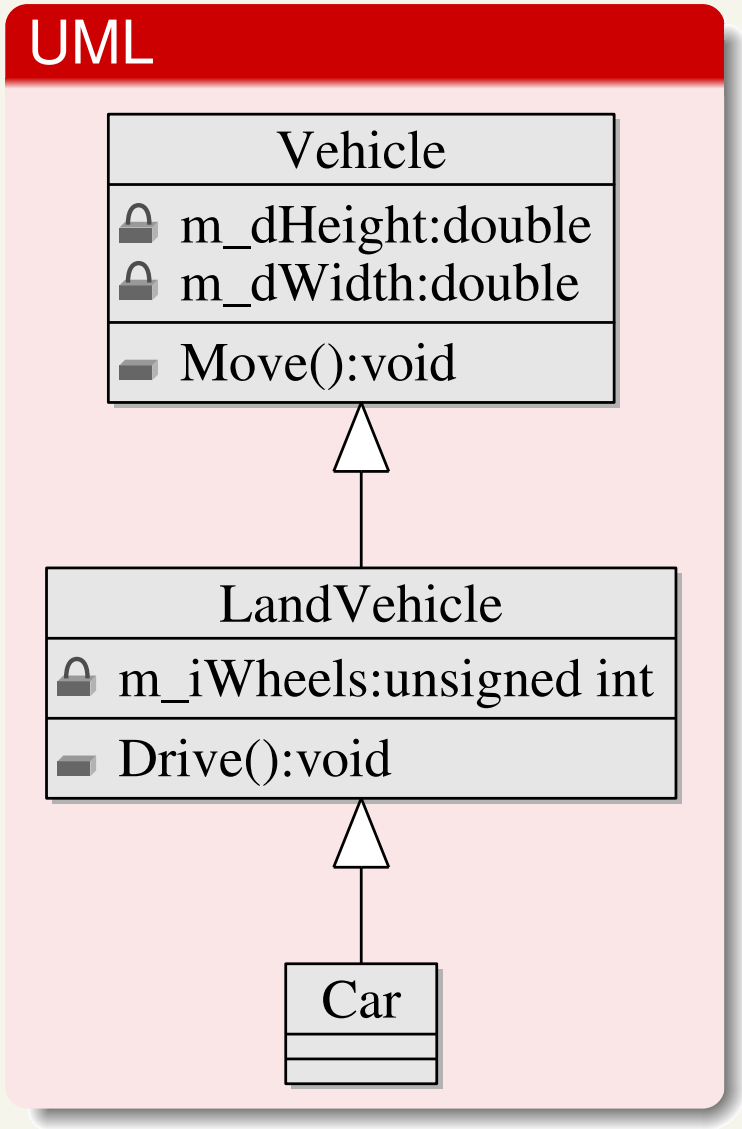
The child class inherits from its parent class

- the characteristics (element variables)
- the behavior (member functions)



Syntax of inheritance

```
class Vehicle
{
public:
  Vehicle(double dHeight,
          double dWidth);
  ...
};

class LandVehicle : public Vehicle
{
public:
  void Drive();
protected:
  unsigned int m_iWheels;
};

class Car : public LandVehicle
{
  ...
}
```



What happens during inheritance?

Institute of Automation

- Each object of a derived class contains an anonymous object of its parent class (sub-object).
- The sub-object will be created automatically via implicit call of the parent constructor.
- All public member functions can be used within the derived class.
- A child class is able to enhance the functionality of the parent class.
- Member functions can be overloaded within the child class (equivalent signature).

Universität Bremen

Accessing the anonymous sub-object

Institute of Automation

Classes Definition

```
class Vehicle
{
public:
  void Show();
};

class LandVehicle :
  public Vehicle
{
public:
  void Show();
};

class Car : public LandVehicle
{
public:
  void Show();
}
```

Call example

```
Car myCar;

// Show() of class Car
myCar.Show()

// Show() from LandVehicle
myCar.LandVehicle::Show()

// Show() from Vehicle
myCar.Vehicle::Show()

// Also possible to call
//  within class Car
void Car::Show() {
  Vehicle::Show();
}
```

Bremen

Type-relations

Institute of Automation

Objects of derived classes can be assigned to objects of the corresponding parent class.

```
Vehicle vehicleObj(dHeight, dWidth);
LandVehicle landVehicleObj(dHeight, dWidth, dNrWheels);
vehicleObj = landVehicleObj;  // OK, but loss of additional
                              // data from derived class
```

A reversal assignment (parent => derived) is not allowed and also not reasonable.

```
landVehicleObj = vehicleObj;  // NOT possible, NrOfWheels
                              // not given by VehicleObj
```

The same statements apply for (class-)pointers and references!

Universität Bremen

Initialization with lists

Institute of Automation

If objects of a derived class are initialized using a constructor with an initialization list, this list

- must **not** contain elements of the base class
- may contain constructor calls of the base class

Example

```
LandVehicle(const Vehicle& vehicleObj, int iWheelNumber)
  : Vehicle(vehicleObj),
    m_iWheelNumber(iWheelNumber)
{
  ...
}
```

Universität Bremen

Inheritance of access rights

Institute of Automation

The weakest access right for the members of the inherited class is set to the access type of the inheritance.

Member \ Inheritance	`public`	`protected`	`private`
`public`	`public`	`protected`	`private`
`protected`	`protected`	`protected`	`private`
`private`	`private`	`private`	`private`

Universität Bremen

Small Exercise

Institute of Automation

- Implement a class `LandVehicle` that is derived from the class `Vehicle`.
- Implement a class `Car` that is derived from `LandVehicle`.
- In the standard constructor of each class, print a message naming the class that is called.
- Test the behavior of inheritance in a small test program.
- Draw a UML class diagram (on paper), which illustrates the relationship between the classes `Vehicle`, `LandVehicle` and `Car`.

Universität Bremen

Inline methods

Institute of Automation

Idea:

- Programming with classes uses typically many small functions
- With an inline definition efficiency profit is possible. (Reminder: inline is a recommendation to the compiler to directly put into the function's body instead of the input of a function call)

Realization:

- Declaration and definition within the class (decreases readability of class interface)
- Declaration and definition within the header-file

Universität Bremen

Institute of Automation

Inline example

```
1  class Vehicle
2  {
3  public:
4      Vehicle() {} // inline definition within the class
5      Vehicle(double dHeight, double dWidth);
6      double GetHeight() const;
7      double GetWidth() const;
8  private:
9      double m_dHeight;
10     double m_dWidth;
11 };
```

```
1  // Inline implementation within the header-file
2  inline double Vehicle::GetHeight() const { return m_dHeight; }
3  inline double Vehicle::GetWidth() const { return m_dWidth; }
4  inline Vehicle::Vehicle(double dHeight, double dWidth)
5  {
6          m_dHeight = dHeight;
7          m_dWidth = dWidth;
8  }
```

Universität Bremen

Class related constants

Institute of Automation

Definition

Class related constants are constants that are valid for all objects of a certain class.

Realization:

- For easier access they can be made public.
- The value of the constant can be defined either in a parameter list of a constructor or in an enumeration.

Universität Bremen

Constants realization

Constructor parameter list

```
class ClassWithConstants
{
public:
    ClassWithConstants() : iMIN(0), iMAX(1000) {};

    const int iMIN;
    const int iMAX;
};
```

Enum

```
class ClassWithConstants
{
public:
    enum { MIN = 0, MAX = 1000 };
};
```

Static class members

Static member variables are

- used to create class-global states/variables.
- valid for all objects of that class and only exist once in the memory.
- used e.g. for counting object instances or other shared information like common graphical origin on the screen.

Static methods

- for class-global operations not bound to a certain object (e.g., to operate on class related (static) data)

Example class header file (*.h)

```
1  #ifndef NUMBERED_H
2  #define NUMBERED_H
3  class Numbered
4  {
5  public:
6      Numbered();
7      ~Numbered();

1      static void ShowNumberOfObjects();

1  private:
2      static int m_iNumber;

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

Example class implementation file (*.cpp)

Institute of Automation

```
1  #include "Numbered.h"
```

```
1  int Numbered::m_iNumber = 0;
```

```
1  Numbered::Numbered()
2  {
3      m_iNumber++;
4  }
5  Numbered::~Numbered()
6  {
7      m_iNumber--;
8  }
```

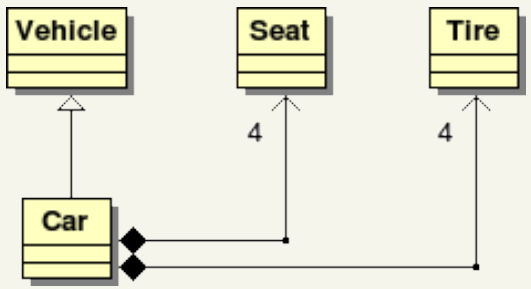
```
1  void Numbered::ShowNumberOfObjects()
2  {
3      std::cout << "Number of objects = " << m_iNumber << std::endl;
4  }
```

Universität Bremen

Small Exercise

Institute of Automation

- Enhance your class `Vehicle` so that the number of generated objects can be shown.
- Implement in your test program a small 'parking lot' that counts the number of vehicles. In a menu, make sure that vehicles can enter and leave the parking lot (use array of pointers) and the actual number of vehicles is shown.
- Implement the classes which correspond to the following UML class diagram:



Universität Bremen

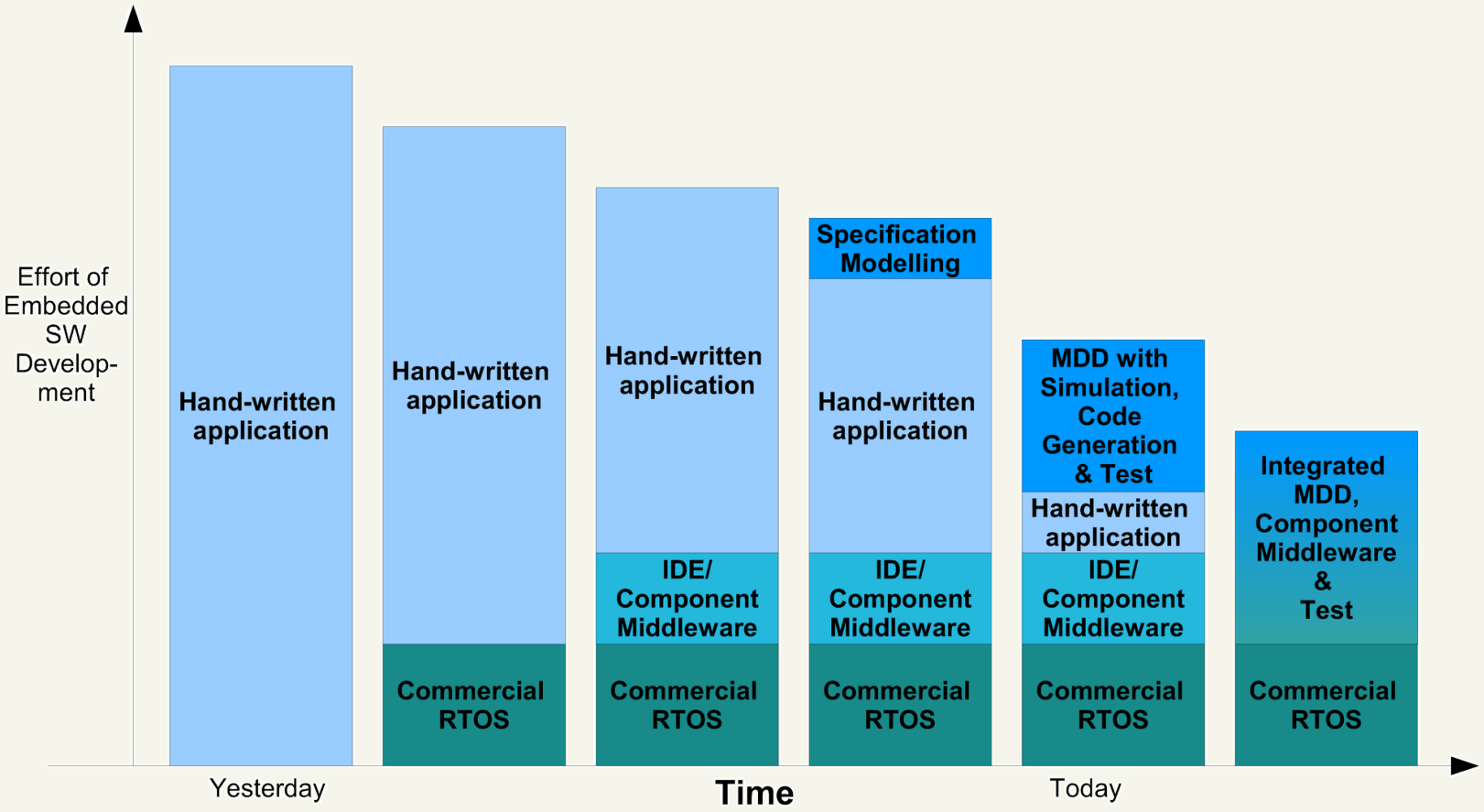
Institute of Automation

Introduction to MDD (Model Driven Development)

Universität Bremen

Institute of Automation

MDD: Emerging future technology



Universität Bremen

Institute of Automation

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

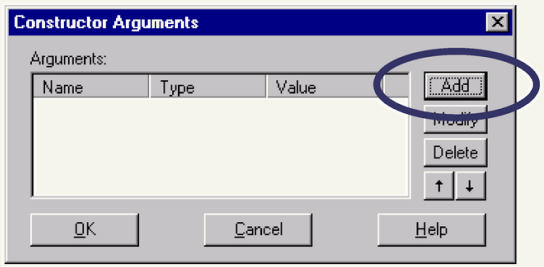
Universität Bremen

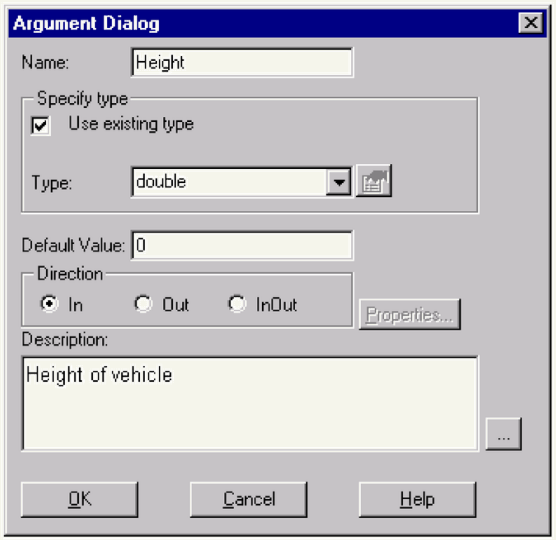
Institute of Automation

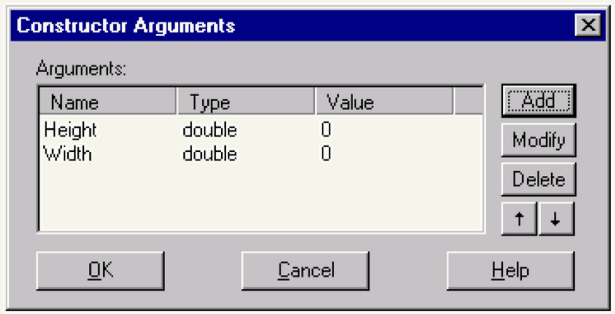
General Constructor: Exercise #1

- Introduce a general constructor for the CVehicle class
- To define standard + general constructor in one:
  - Assign default values for the constructor's parameters

- (Here we will not use initialization of attributes from the passed parameters in the initialization list. See today's last slide how to do that.)







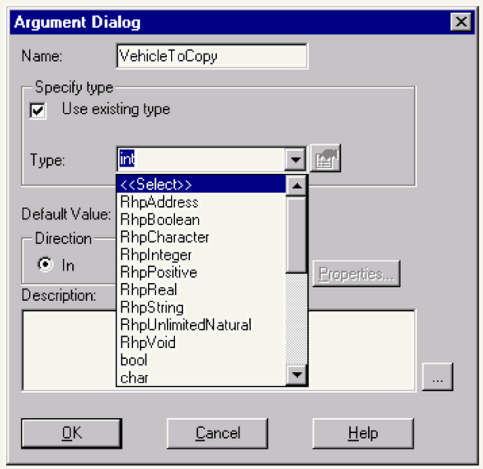
Universität Bremen

Institute of Automation

Exercise #2

- Introduce a copy constructor to the CVehicle class
  (In the add parameter dialog you have to select the CVehicle class as parameter datatype)
- Introduce a destructor to the CVehicle class
  (Right-click on the class in the model-browser (not in the diagram!), select “Add new” / “Destructor”)
- Put in an implementation like

```
std::cout << "CVehicle destructor
called"
          << std::endl;
```

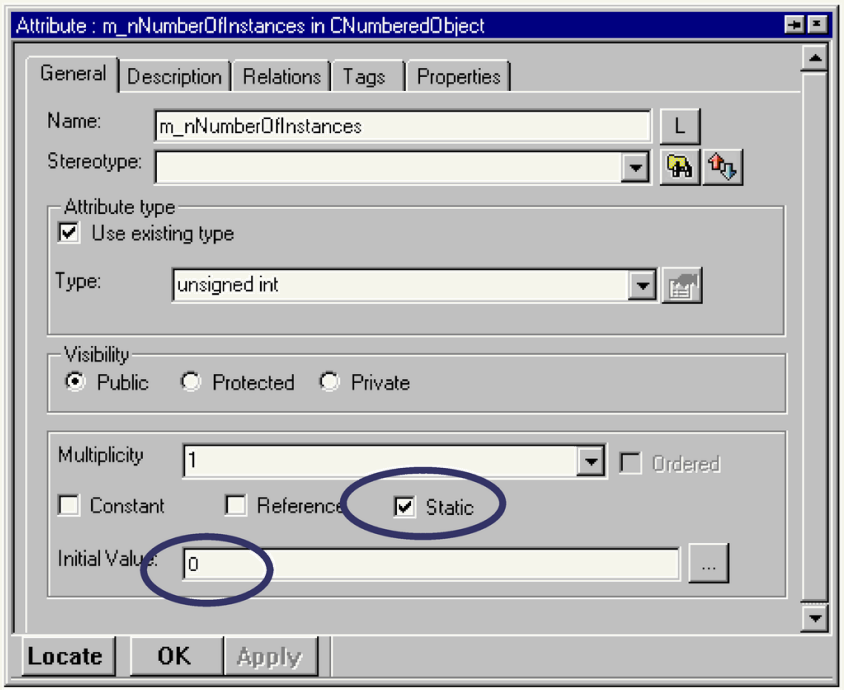


- Inspect the generated code
- Check out the behavior of destructors by creating an CVehicle object (global object with “Make Object” or initial instance in component settings).

Universität Bremen

Institute of Automation

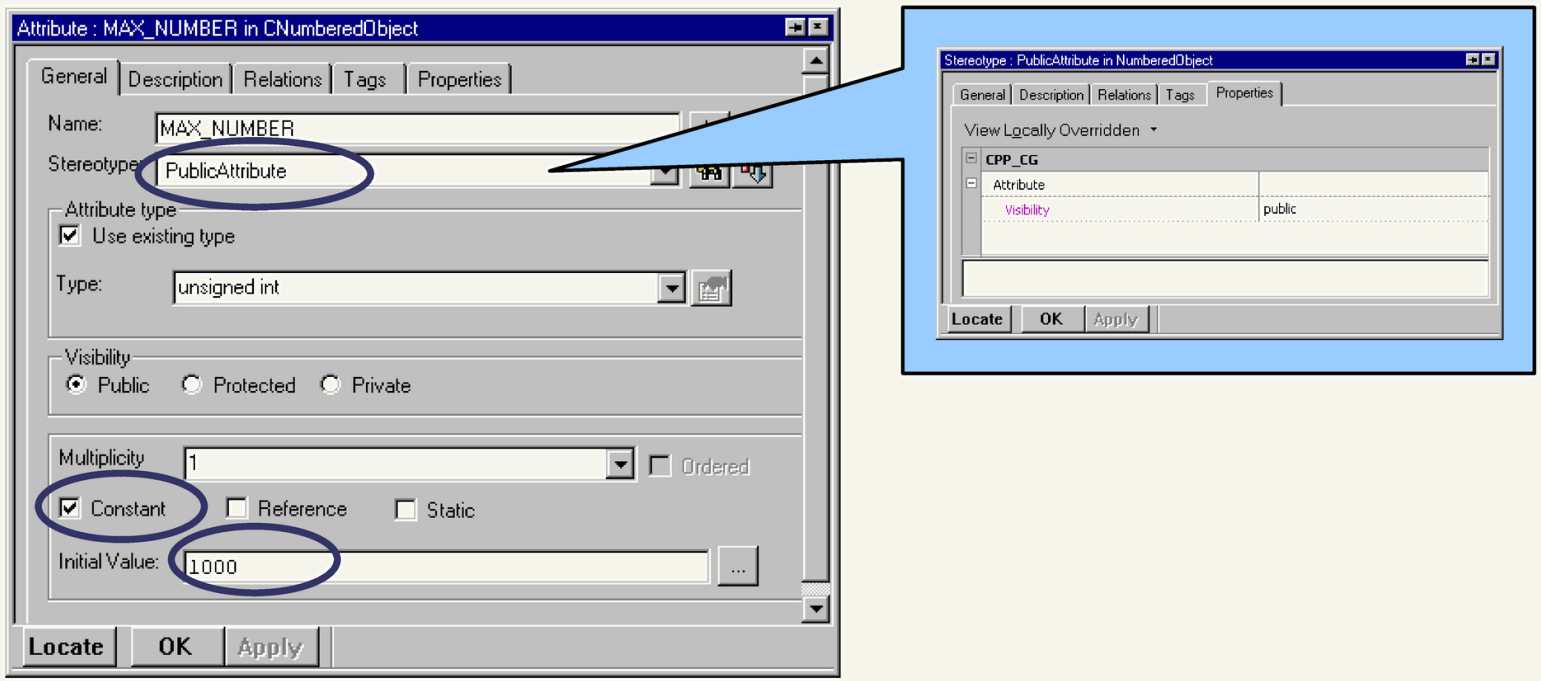
Static declaration in Rhapsody



Universität Bremen

Institute of Automation

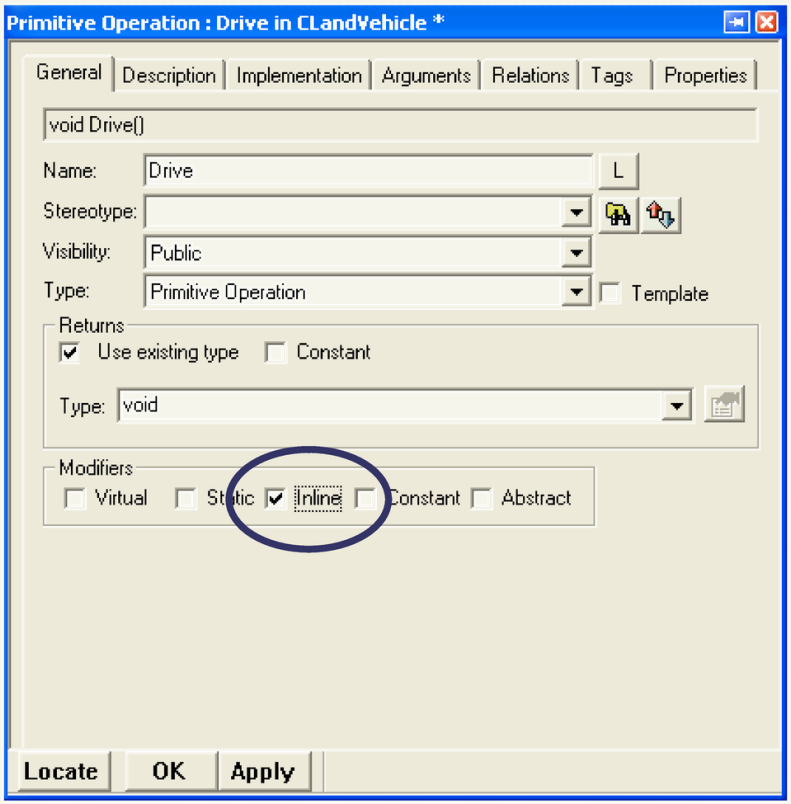
Constant declaration in Rhapsody



Universität Bremen

Institute of Automation

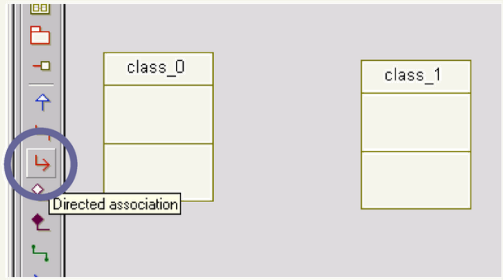
Inline declaration in Rhapsody



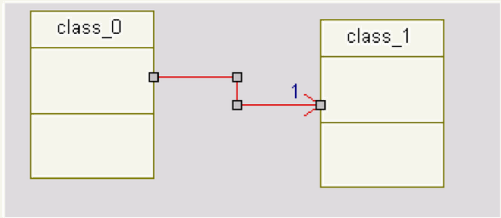
Universität Bremen

Institute of Automation

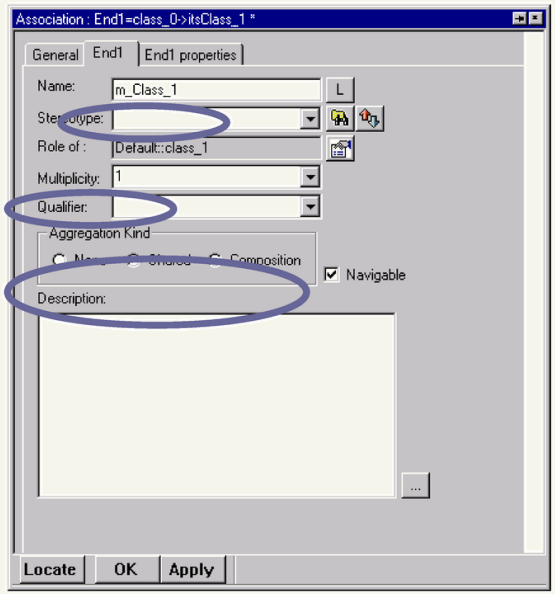
Construct relations (1)



1) Select **directed** association

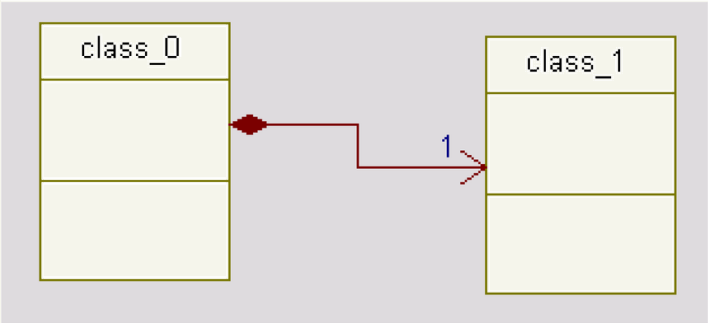


2) Draw association



3) Double-click on the arc to modify the

- association's name (e.g to m_pPublisher)
- the multiplicity
- the type of association

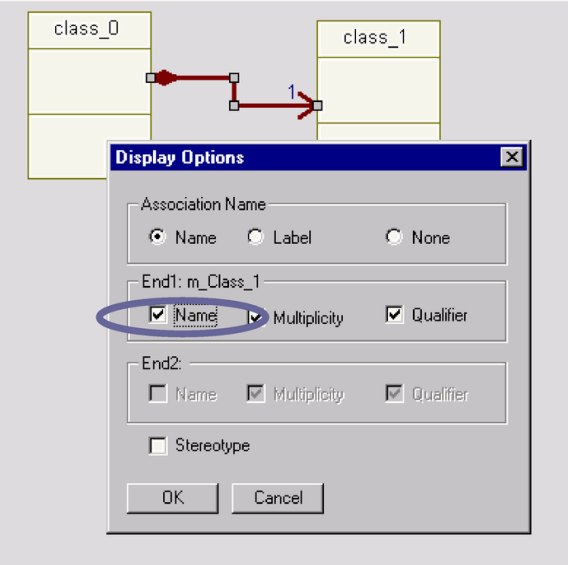


Here is now a composition with the multiplicity one

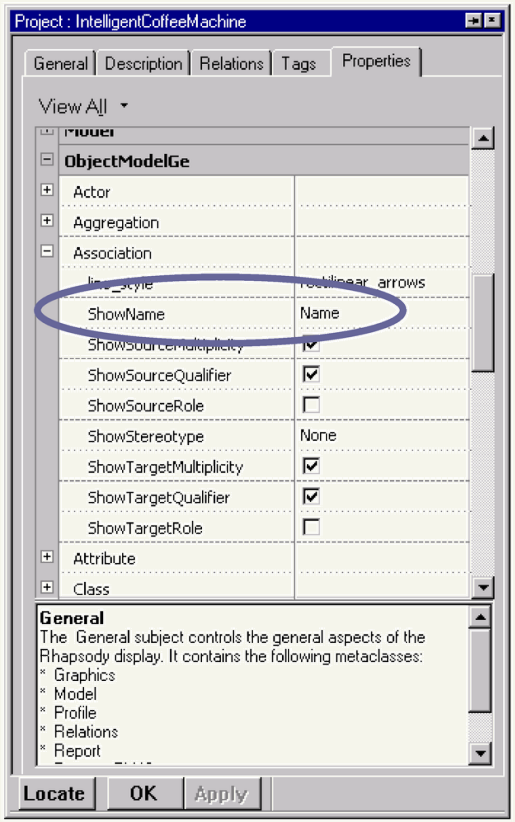
Universität Bremen

Institute of Automation

Construct relations (2)



Select *Display Options* from the arcs context menu and check the name box. Then the association's name will be shown.



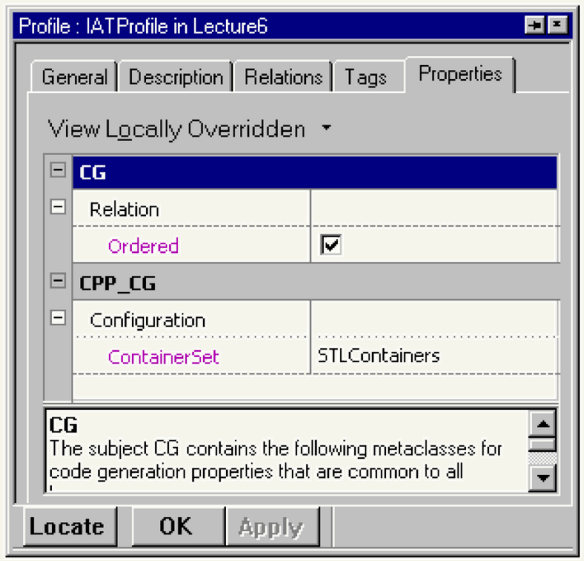
It is possible to configure the display options globally (or within a certain level of the model) via the Properties tab of an entity.

Universität Bremen

Institute of Automation

Construct relations (3)

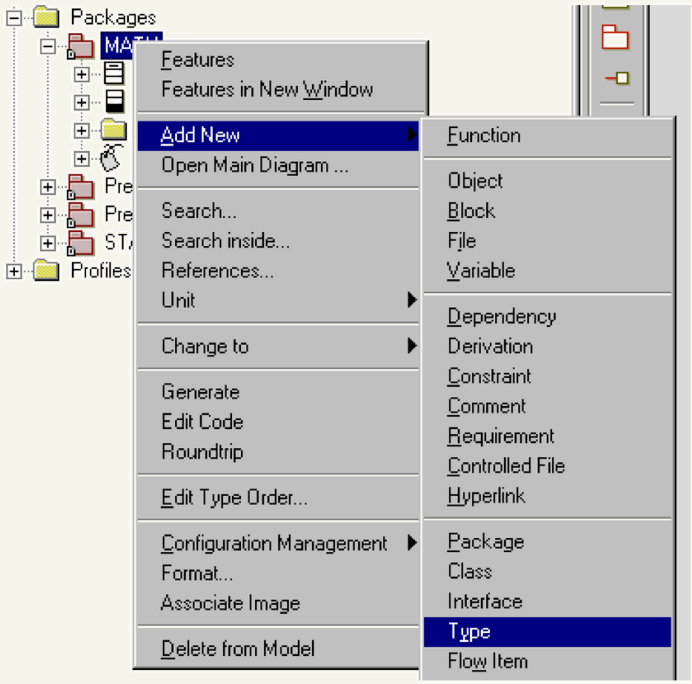
- Per default, Rhapsody uses its own container classes to realize associations with *-multiplicity (“infinite” number of elements).
- We want to use STL-containers (e.g. std::vector)
- Activate the respective setting for your project (i.e. within the properties of the root element in the model browser) as shown here or simply add the IATprofile file (given in elearning, lecture #6) to your project for automatically and globally applying the settings.
- (The “Ordered” checkbox specifies to use vectors instead of lists.)



Universität Bremen

Institute of Automation

Typedefs

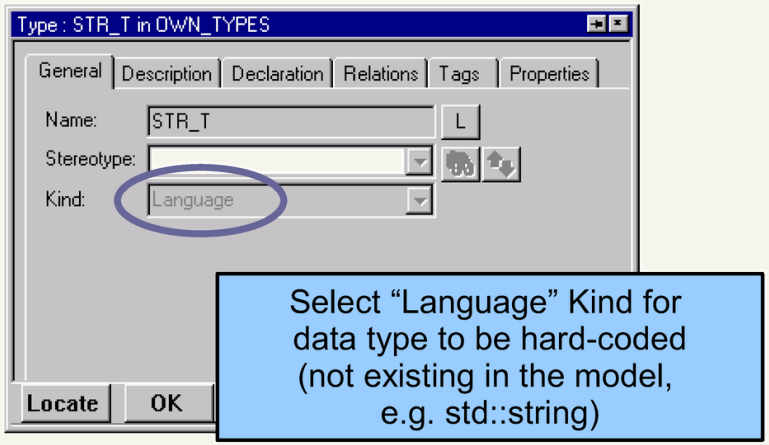


- Add new type

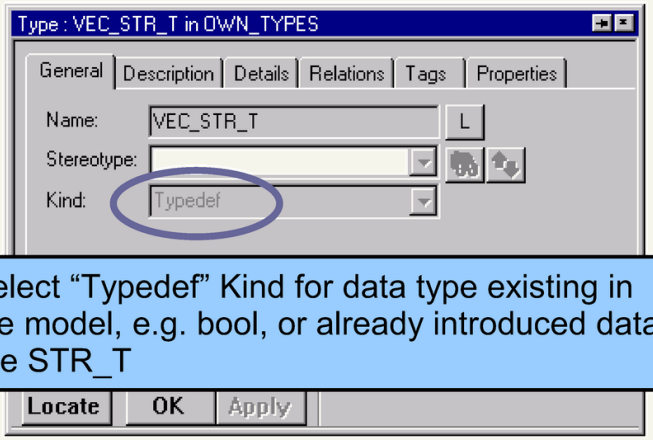
Universität Bremen

Institute of Automation

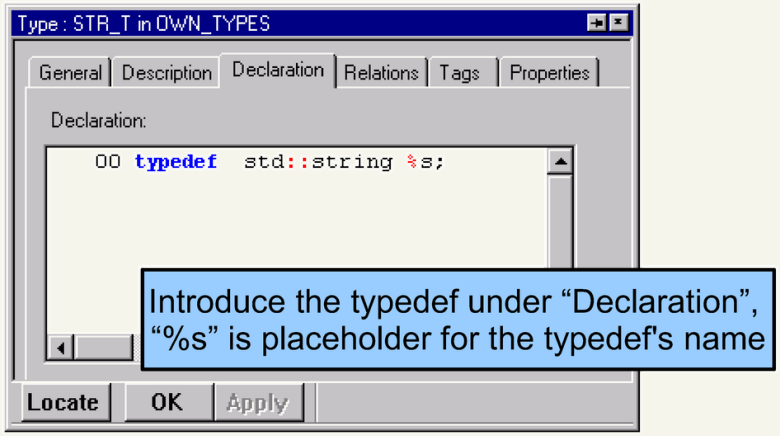
Typedefs



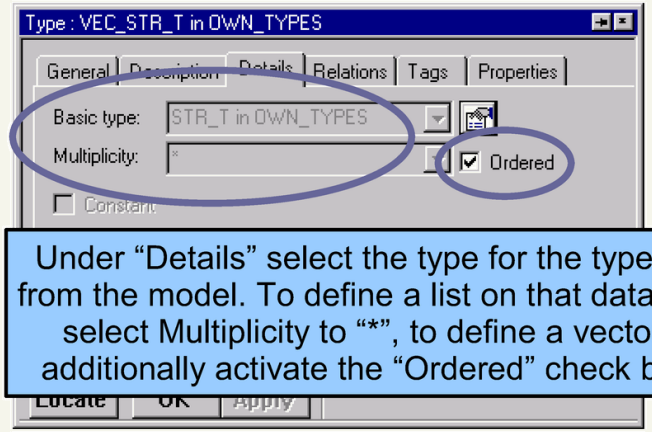
Select “Language” Kind for data type to be hard-coded (not existing in the model, e.g. std::string)



Select “Typedef” Kind for data type existing in the model, e.g. bool, or already introduced data type, like STR_T



Introduce the typedef under “Declaration”, “%s” is placeholder for the typedef's name



Under “Details” select the type for the typedef from the model. To define a list on that data type select Multiplicity to “*”, to define a vector, additionally activate the “Ordered” check box.

Universität Bremen