Object-Oriented Programming and Java

Part I: Introduction to Concepts and Principles of Object-Oriented Programming

Course by

Mr. Erkki Mattila, M.Sc.

Rovaniemi University of Applied Sciences

Object-Oriented Programming and Java

Instructor: M.Sc. Erkki Mattila, lecturer

Office: C136

Office hours: according to the weekly schedule published on the net

Mobile tel.: 040 740 5862

E-mail: erkki.mattila@ramk.fi

Object-Oriented Programming and Java

COURSE CODE

- 504D24A

DURATION

- 3 CU (+5 CU = 8 CU)

TEACHING

- 40 hours of classes, 40 hours of self-supervised work (first part)

Object-Oriented Programming and Java

OBJECTIVES

- The main objective of the first part of the course is to introduce the basic concepts and principles of object-oriented programming
- The second part introduces the student to Java programming (Java SE)

Object-Oriented Programming and Java

COURSE MATERIAL

- Lecture notes, available at O:\Opettajat\Erkki Mattila\Object-oriented Programming and Java 504D24A - Part 1
- Budd T. 2002. An Introduction to Object-Oriented Programming, 3rd Edition. Addison-Wesley Longman
- Sebesta R. W. 2008. Concepts of Programming Languages, 8th Edition. Pearson Education. Addison-Wesley

Object-Oriented Programming and Java

ASSESMENT

- Mid-term exam, which will be graded on a scale from 1 to 5 and F. 30 percent of the maximum points are required for grade 1.
- The final exam will be held after the second 5 CU part of the course. The course grade will be calculated as a weighted average of the mid-term and final exam grades

Student level assesment

1. Which programming courses have you participated earlier?
2. How familiar are you with the concepts and principles of object-oriented programming? You may also answer on a scale from 0 to 5, zero being not at all and five being master level?
3. Do you have prior experience of object-oriented programming languages? If so, which programming languages/tools have you used?
4. Are you familiar with UML (0-5)?
5. What do you expect/hope to get from this course?

Course Contents

- Part I: Concepts and Principles of Object-Oriented Programming
- Part II: Object-oriented Programming
  - Closer look at OOP
- Part III: Introduction to Java Programming

Part I Contents

1. Abstraction
   - The concept of abstraction, data and process abstraction, abstract data types
2. Concepts of Object-Oriented Programming
   - Class, object, attribute, method, etc.
3. Principles of Object-oriented Programming
   - Data abstraction (encapsulation and information hiding), inheritance, and polymorphism

1. Abstraction

Object-oriented Programming Paradigm

- OOP is a common programming paradigm. A programming paradigm is a way to conceptualize how to structure a program to solve a problem
  - Other programming paradigms include functional programming, logic programming, imperative programming and declarative programming
- Common OO **programming** languages nowadays are Java, C++ ja C#
- Common OO **modelling** languages are UML, OMT ja OMT++. UML is nowadays the de facto standard

C Data Types Revisited

- Which built-in, primitive data types the C programming language has?
- What are structure types (struct) in C language?
- What is the relationship between functions and data in C language?

Definition: Abstract Data Type

- An abstract data type is a data type that satisfies the following conditions:
  - The declarations of the type and (the protocols of) the operations of the type are contained in a single syntactic unit (encapsulation)
  - Other program units are allowed to create variables of the type
  - The representation of objects of the type is hidden from other program units. The only direct operations possible on those objects are those provided in the type's definition (information hiding)

Sebesta R.W. 2008. Concepts of Programming Languages , 8th Edtion. Addison Wesley

Abstract Data Types (ADT)

- ADTs are defined by the user (user-defined type)
- ADTs should provide the same characteristics provided by language-defined types, such as an integer or a floating point type:
  - A type's definition that allows program units to declare variables of the type, but hides the representation of the objects of the type
  - A set of operations for manipulating objects of the type
- In OOP abstract data types are implemented as **classes**
- An instance of an abstract data type is called an **object**

Scope

- A class consists of data fields (attributes) and operations, which manipulate the data
- The data fields/**attributes can be accessed anywhere inside the class itself**; i.e. all operations of the class have direct access to class's data

Group Work

- Define a new abstract data type
  - Name of the type
  - Attributes (Data fields)
  - Operations

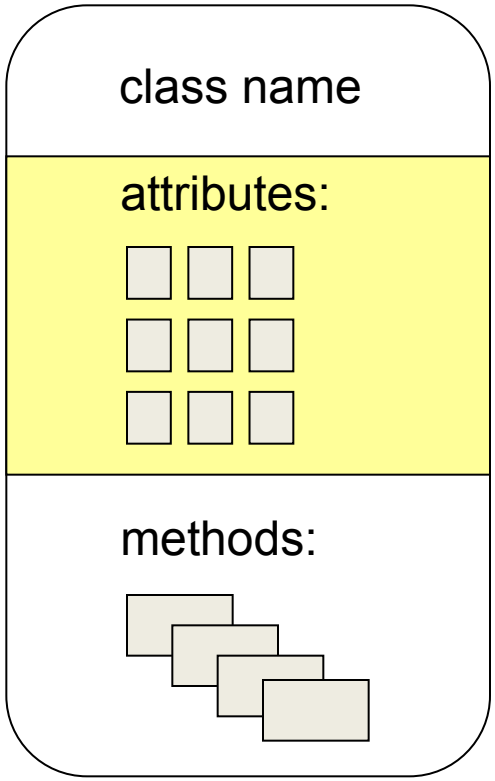
Relation of Abstract Data Types to Object-oriented Programming

- Object-oriented programming (OOP) is an outgrowth of the use of ADTs
- Data abstraction is one the most important components of OOP
- In OOP languages abstract data types are implemented as **classes**!
- An instance of a class is called an **object!**

2. Concepts of Object-oriented Programming

Class

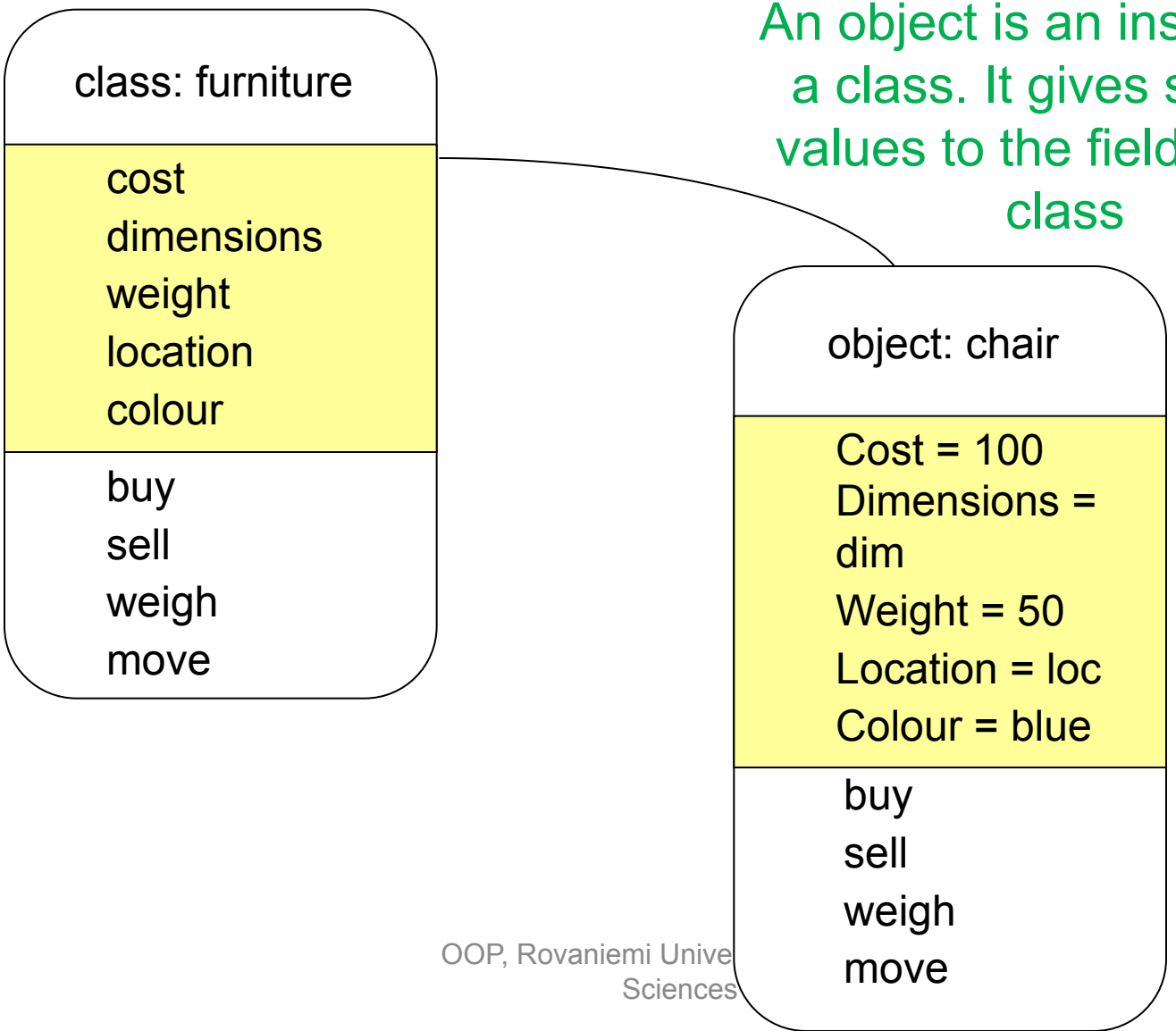
- Classes are reusable software components that model items in the real world
- A class encapsulates the data and procedural abstractions (operations) that are required to describe the content and behaviour of some real world entity
- A program written in pure OOP language consists of classes – there can be no code outside of classes
- A class is a compile-time concept



Object

- An object is an instance of a class
- Objects are created during program execution – object is a run-time concept
- Multiple objects can be instantiated of the same class
- Each object has its own values for the instance variables (attributes) of its class
  - These values define the state of the object

Classes and Objects



Attributes

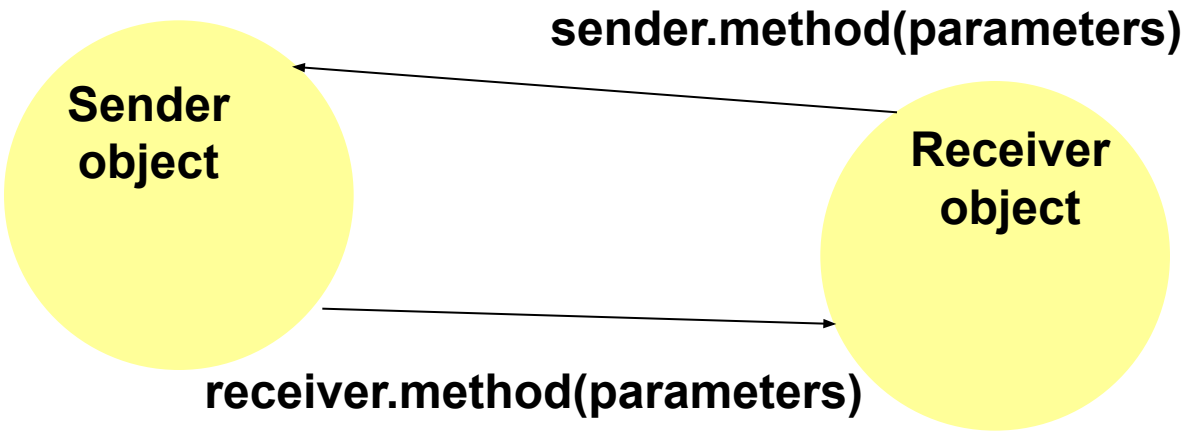
- People have features including date of birth, name, height and eye colour
- Physical objects have features such as shape, weight, colour, etc
- Similarly classes have attributes (=fields, member variables)
- The values assigned to an object's attributes make that object unique; they define the state of an object

Methods

- A class encapsulates data and the algorithms that process that data. These algorithms are called methods, operations, functions, routines or services
- Each of the methods provide a representation of one of the behaviours of the object
  - Behaviour = method implementation
- Whenever an object receives a message, it initiates some behaviour by executing a method
  - Message = method call

Messages (Method Calls)

- Messages are the means by which objects interact. A message stimulates some behaviour to occur in the receiving object. The behaviour is accomplished when a method is called and executed



Pressman R., p.533

Constructor

- A specialized method used to instantiate an object
- The constructor function has the same name as the class
- It is never called directly, but the run-time system calls it when a new object is created
- Constructors are usually overloaded; a class contains multiple constructors, which have a different set of parameters

A Class Definition in Java Language

```
public class Shape
{
   private static int a_numberOfShapes=0;
   private Color a_color;

   public Shape()
   {
      Shape.a_numberOfShapes++;
      a_color = Color.BLACK;
   }
   public Shape(Color c)
   {
      Shape.a_numberOfShapes++;
      a_color = c;
   }
```

A Class Definition in Java Language

```
public class Circle extends Shape      // Class header
{
   private int radius ;              // Member variable

   public Circle()                        // Constructor
   {
      radius = 0;
   }

   public Circle(int r, Color c) {
      super( c );    // Calling superclass contructor
      radius = r;
   }

   public long calculateArea()         // Member function
   {
      return Math.round(Math.PI*radius*radius);
   }
}
```

Group Work

- Your task is to design an ATM (Automatic Bank Teller Machine) system
  - Which classes (and objects) should the system contain?
  - Which attributes and operations the classes should have?

Class Relationships

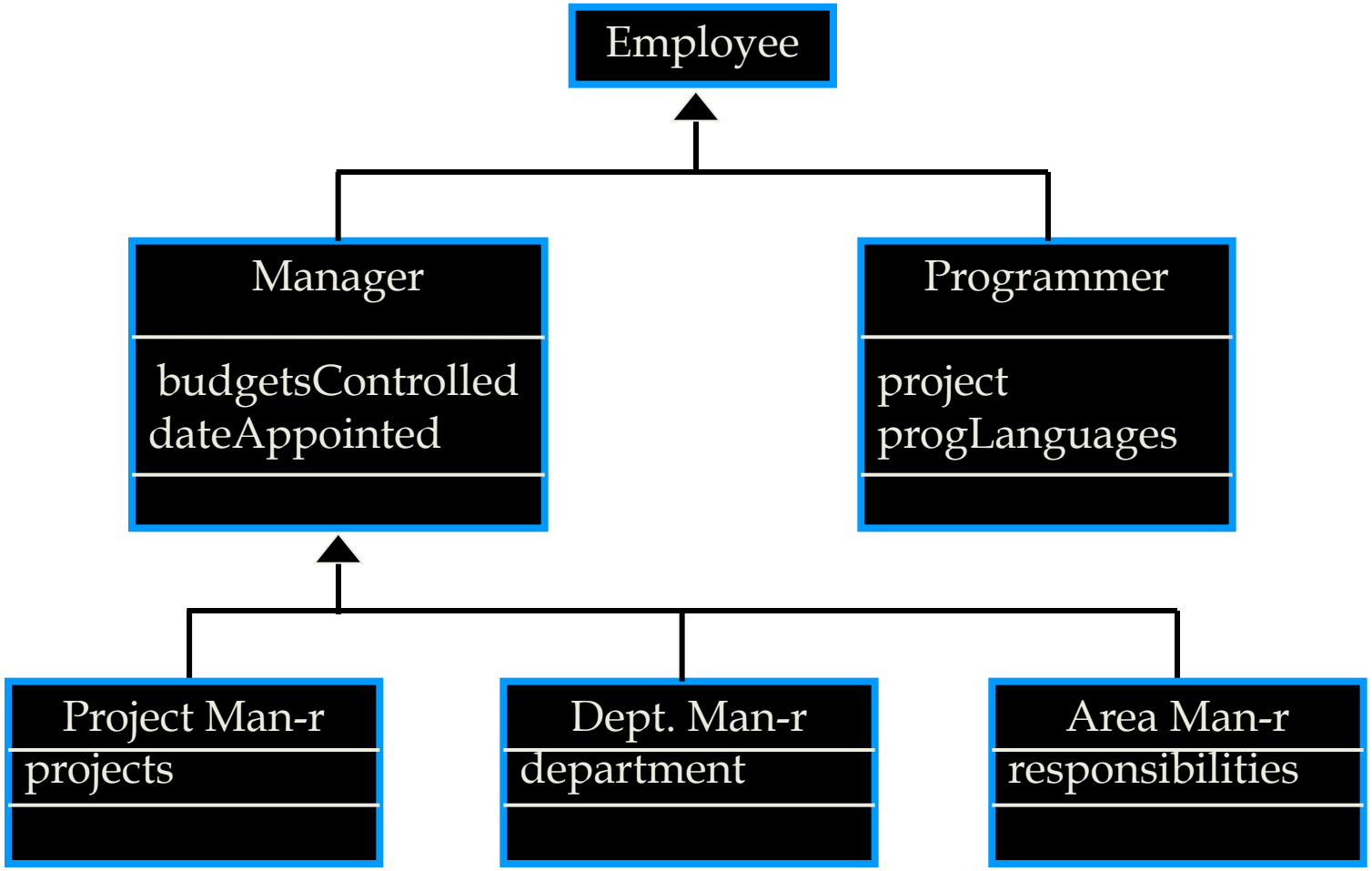
1) Generalization (Inheritance)
2) Aggregation
   - Special case: composition
3) Association

- Dependency and Realization relationships will be coverered in the Design Methods course

Inheritance (“is a”) Relationship

- Classes may be arranged in a class hierarchy where one class (a superclass) is a generalisation of one or more other classes (subclasses)
- Generalization: creating a common supertype for a group of classes
  - Dog, Cat, Horse => supertype Animal
- Specialization: creating extended versions of existing classes or objects. This is also called subtyping. Subclass is a special case of the superclass
  - Animal => subtypes Dog, Cat, Horse
- A subclass inherits the attributes and operations from its superclass and may add new methods or attributes of its own
  - For instance Animal class contains the features common to all animals

Generalization (Inheritance)

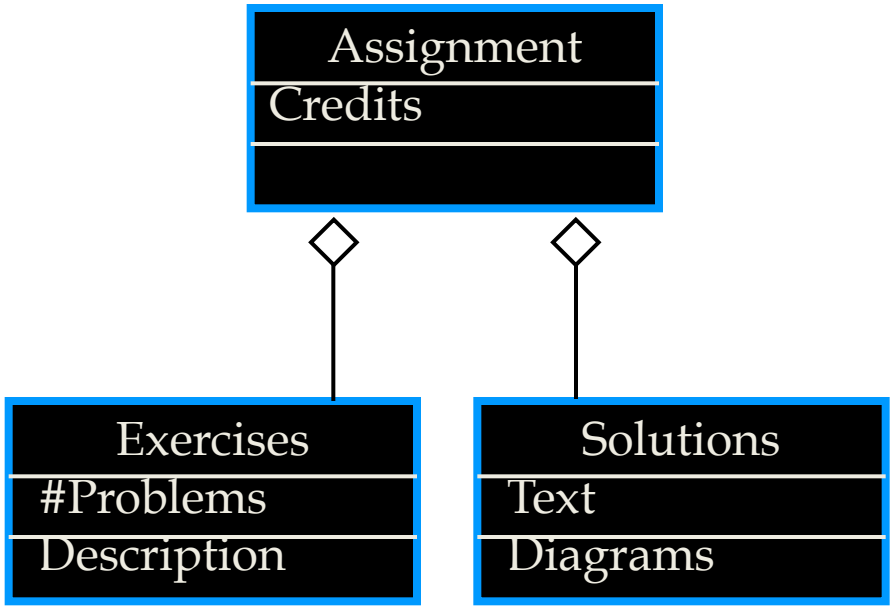


Group work

- Define a rectangle and a square as classes.
- Can they be modelled with a single class?
- If they are modelled as separate classes, is one a superclass of the other? Why?
- Start by defining the Rectangle class. Declare attributes and constructor methods. Initilize attributes in contructors.
  - Remember that constructors can be overloaded
  - Remember that superclass constructor method can be called in the subclass using the keyword **super**
- Add a method for calculating the area of the rectangle

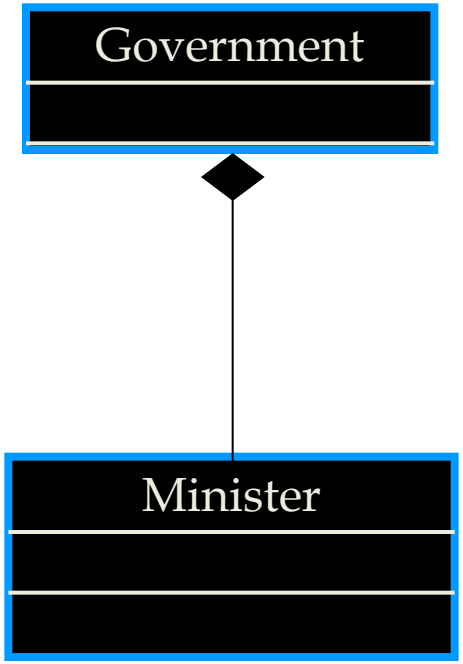
Aggregation (“part of“) Relationship

- Shows how classes that are collections are composed of other classes.
- Models the notion that one object uses another object without "owning" it and thus is not responsible for its creation or destruction.
- Similar to the part-of relationship in semantic data models.



Composition (“has a “) Relationship

- **Composition is a special form of aggregation** describing the situation where an object contains a number of other objects and when the containing object is deleted, all the instances of the objects that are contained disappear
- The contained (member) object cannot exist without its owning object and can belong to one owner only at any given time

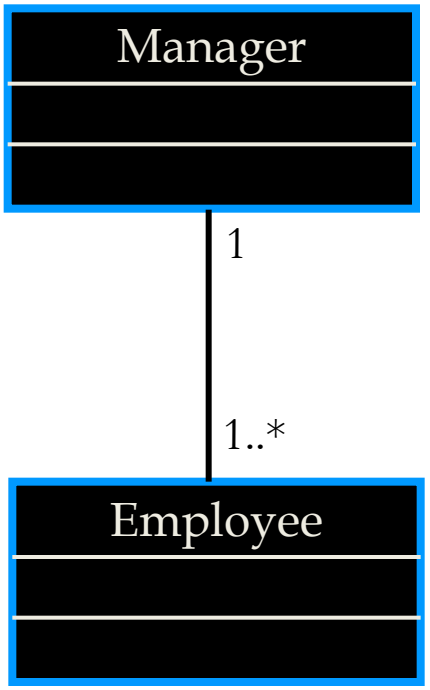


Composition vs. Aggregation

- Composition is a stricter relationship than aggregation:
  1. Member objects cannot exist without the containing object.
  2. A member object can belong to only one containing object at a time.
- Example of composition: a minister cannot exist without a government, and a minister can be a part of only one government at a time.

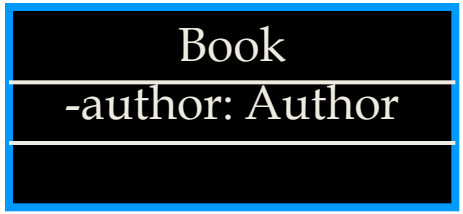
Association (“uses”) Relationship

- One entity uses another entity as part of its behavior
- Used when the relationship is permanent
- In the code level one class has a member variable of another class type



Attributes and Associations

- Attributes and associations are exchangeable!
  - When the relationship exists between classes in your own class model, use an association in the UML class diagram
  - When the relationship exists between a class in your own class model and a class from a class library, use an attiribute



is the same as:

Group Work

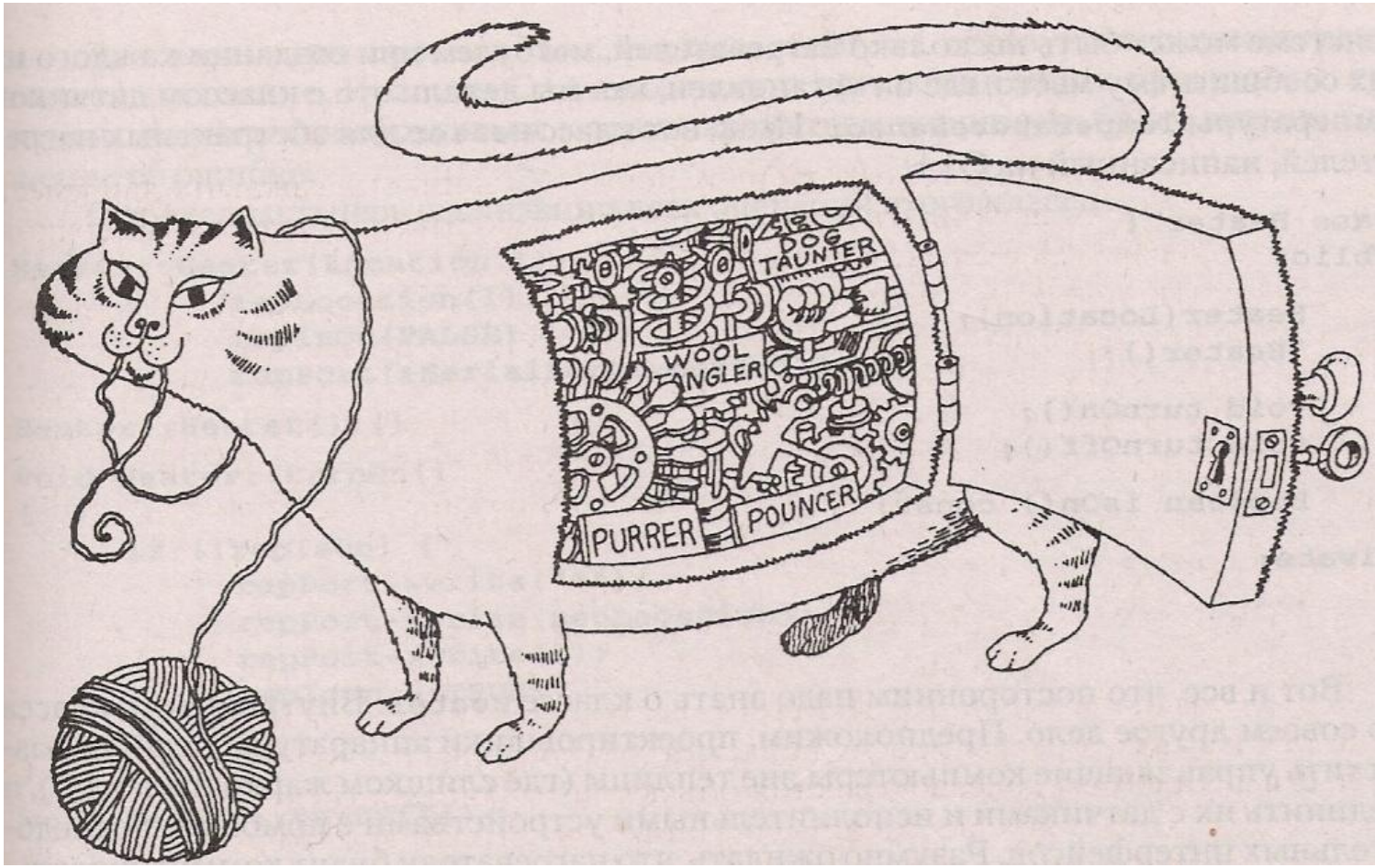
- Draw an UML class diagram containing the following classes: Car, Bicycle, Motorcycle, Steamship, Sailboat, Sail, Train, Wheel (tyre), Engine, Motorship, Watercraft, Vehicle (means of transportation), Landvehicle, Road, Railway, Waterway, Seat (bench).
- Put classes to a class diagram and add some attributes to them. Do the classes have some common attributes?
- Add appropriate relationships between the classes
  - Start by defining the class hierarchy
  - Add common features as high as possible in the inheritance hierarchy

3. Principles of Object-oriented Programming

Encapsulation

- A class encapsulates together data (attributes), methods, constants, and other related information
- Encapsulation means that all of this information is packaged under one name and **can be reused as one specification or program component**

Encapsulation: Example



G. Booch, “Object-oriented analysis and design with applications”

Benefits of Encapsulation

- Data structures and the methods that manipulate them are merged in a single named entity – the class
  - Facilitates component reuse
- Interfaces among encapsulated objects are simplified. An object that sends a message need not be concerned with the details of internal data structures
  - Greatly reduces programmer's memory load

Data Abstraction and Information Hiding

- DA isolates how a compound data object is used from the details of how it is constructed from more primitive data objects
  - Consider a class Stack and its operations push and pop
- The client cares about what services a class offers, not about class's internal data structures or how the services are implemented

Data Abstraction and Information Hiding

- Classes normally hide the details of their implementation from their clients. This is called **information hiding**
- Although programmers might know the details of a class's implementation, they should not write code that depends on these details as the details may later change

Data Abstraction and Information Hiding

- From the definition of an abstract data type:
  - The declarations of the type and the protocols of the operations on objects of the type, which provide **type's interface**, are contained in a **single syntactic unit**
    - Data Abstraction (type's interface) and Encapsulation (single syntactic unit)
  - The representation of objects of the type is hidden from other program units. The only direct operations possible on those objects are those provided in the type's definition
    - Information hiding

Information Hiding

- A type's internal form is hidden behind a set of access functions
- Values of the type are created, inspected and modified only by calls to the access functions
- This allows the implementation of the type to be changed without requiring any changes outside the module in which it is defined

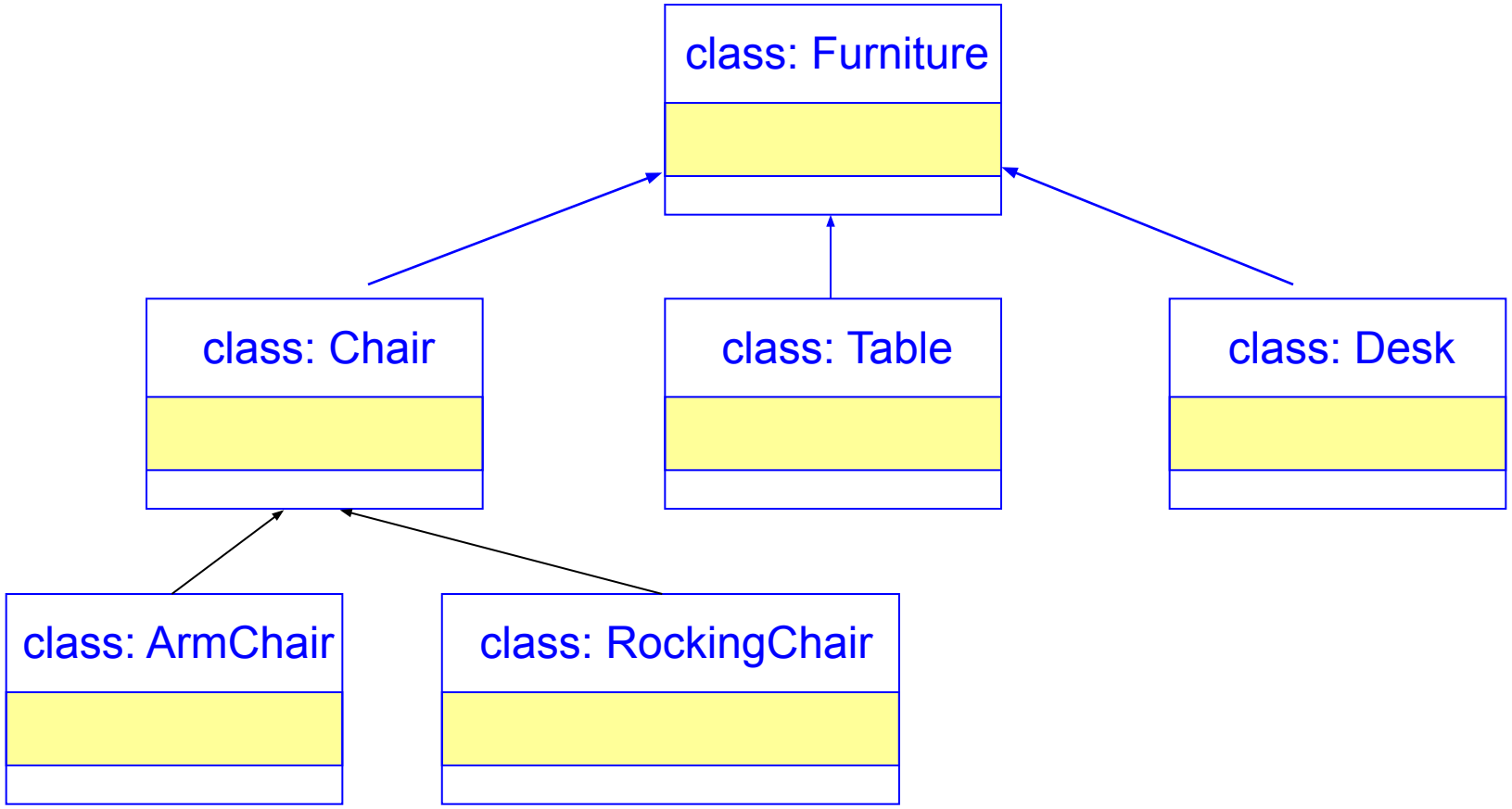
Benefits of Information Hiding

- The internal implementation details of data and procedures are hidden from the outside world. This reduces the propagation of side effects when changes occur
- Preserving the integrity of objects
  - Illegal attribute values not allowed
- More on information hiding and using access specifiers to implement it in the second slide set

Inheritance

- Inheritance is a mechanism that enables the responsibilities of one object to be propagated to other objects
- In a class hierarchy the attributes and methods of the superclass are inherited by subclasses that may each add additional “private” attributes and methods
- Inheritance can also be called as
  - Specialization: creating extended versions of existing classes or objects. This is also called subtyping. Subclass is a special case of the superclass
  - Generalization: creating a common supertype for a group of classes

Class Hierarchy



Class Hierarchy

- Direct superclass
  - Inherited explicitly (one level up hierarchy)
- Indirect superclass
  - Inherited two or more levels up hierarchy
- Single inheritance
  - Inherits from one direct superclass
- Multiple inheritance
  - Inherits from multiple direct superclasses
    - C++ supports multiple inheritance, Java does not

Benefits of Inheritance

- Inheritance occurs throughout all levels of a class hierarchy. Changes at the higher level are immediately propagated through a system
- Reuse of components is accomplished directly

Group Work

- Consider the following classes:
  - Building (properties: address, in use, build year, value,…)
  - Apartment (properties: number, size,…)
  - Office (business premises, properies: number, size,…)
  - Resident (add properties)
  - Company (properties: founding year, turnover,…)
  - Public Company (publicly listed company, which has its own premises. Add properties)
  - Private Company (unlisted company, which has its own premises. Add properties)
  - Home Business / Company (operates in someone's home. Add properties)
  - Employee (add properties)
- Define the properties of the classes. Define the public interface of the classes. Define the relationships among classes.

Additional Material: Forms of Inheritance

- Specialization
  - The child class is a special case of the parent class
- Specification
  - The parent class defines behaviour that is implemented in the child class, but not in the parent class
  - Used to guarantee that classes maintain a certain common interface – that is, they implement the same methods

Additional Material: Forms of Inheritance

- Construction
  - The child class makes use of the behaviour provided by the parent class, but is not a subtype of the parent class
- Generalization
  - The child class modifies or overrides some of the methods of the parent class **to create a more general kind of object**
  - Bad style. Use only if you cannot modify the parent class

Additional Material: Forms of Inheritance

- Extension
  - The child class adds new functionality, but does not change any inherited behaviour
- Variance
  - The child class and parent class are variants of each other, and the class-subclass relationship is arbitrary
  - Bad style. Declare a common abstract superclass instead

Additional Material: Forms of Inheritance

- Combination
  - The child class inherits features from more than one direct parent class
  - Known as multiple inheritance

Benefits of OOP

- Characteristics of OOP software:
  1. Natural
     - Instead of programming in terms of an array or region of memory, you can program using the terminology of your particular problem
  2. Reliable
     - The modular nature of objects allows you to make changes to one part of your program without affecting other parts

Benefits of OOP

3. Reusable
   - OOP introduces inheritance to allow you to extend existing classes and polymorphism allows you to write generic code
4. Maintainable
   - To fix a bug, you only need to correct it in one place
5. Extendable
   - OOP presents techniques to extend the code: inheritance, polymorphism and overriding

Benefits of OOP

6. Timely
   - Natural software simplifies design of complex systems
   - Multiple developers can work on classes independently (parallel development)