Programming technology

UML,

Inheritance

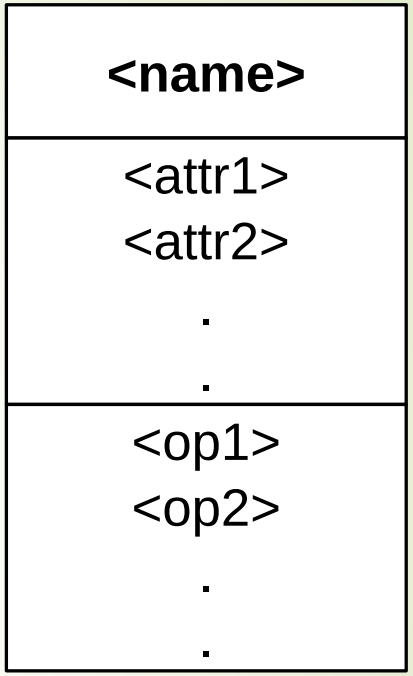
Dr. Rudolf Szendrei

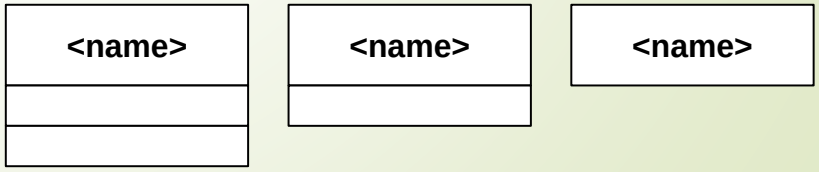
ELTE IK

2020.

UML – Class diagram

- Class diagram shows the name, the attributes and the method declarations of the class
- The name of the class is written in bold
- The name of an abstract class is bold and italic
- Simplified class diagrams:

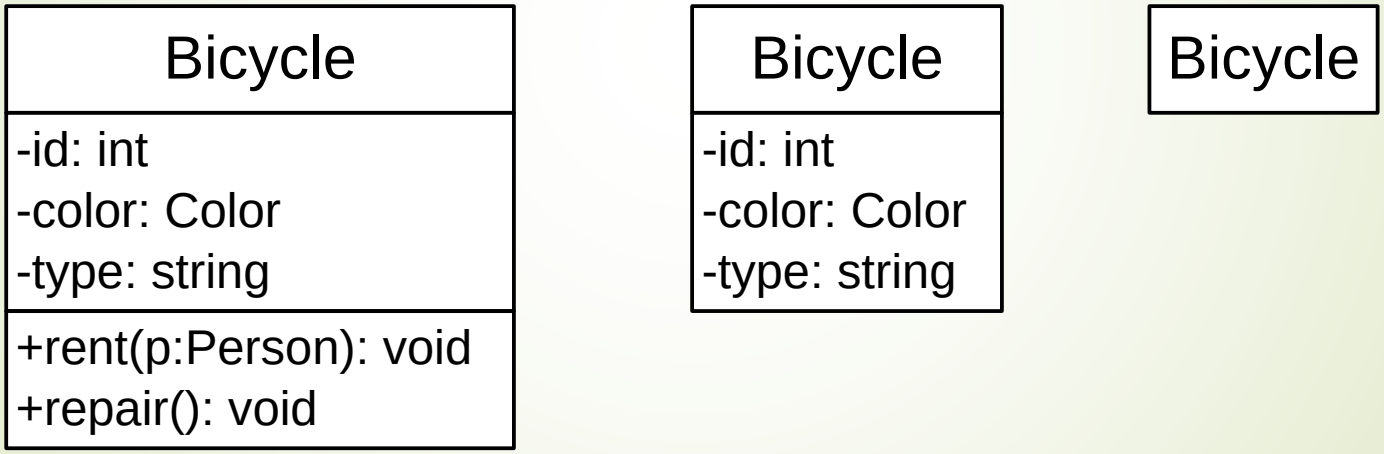




UML – Class diagram example

Example: Bicycle

- We know the color, type and identifier of a bicycle
- Possible methods: rent, repair



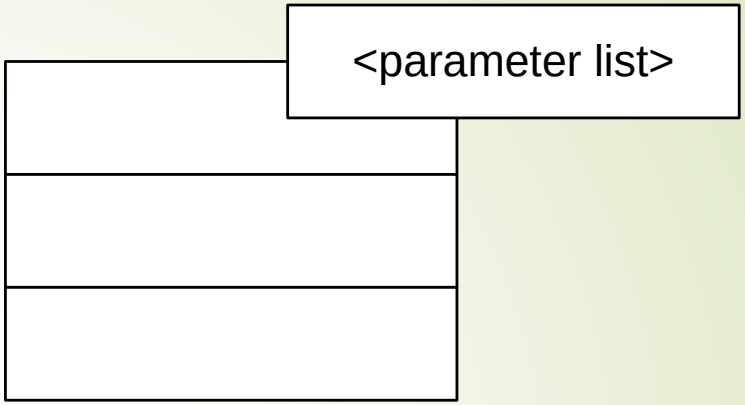
UML – Class diagram example

Bicycle class in Java

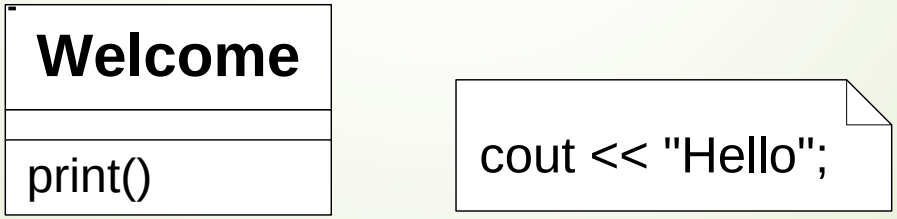
```
public class Bicycle {
  private int    id;
  private Color  color;
  private String type;


  public Bicycle(int id, Color color, String type) {...}


  public void rent(Person p){...}
  public void repair(){...}
}
```

UML – Class diagram

- Generic (template) class

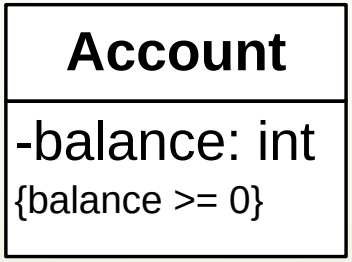


- Annotation: a semantic information about the implementation strategy



UML – Constraints

- We can add constraints to the values of the attributes
- Constraints are logical statements written between curly braces
- Constraints can be made in a UML diagram not just about attributes, but they always have to be put between curly braces
- Example: the balance of an account should be non negative



UML – Class diagram definition

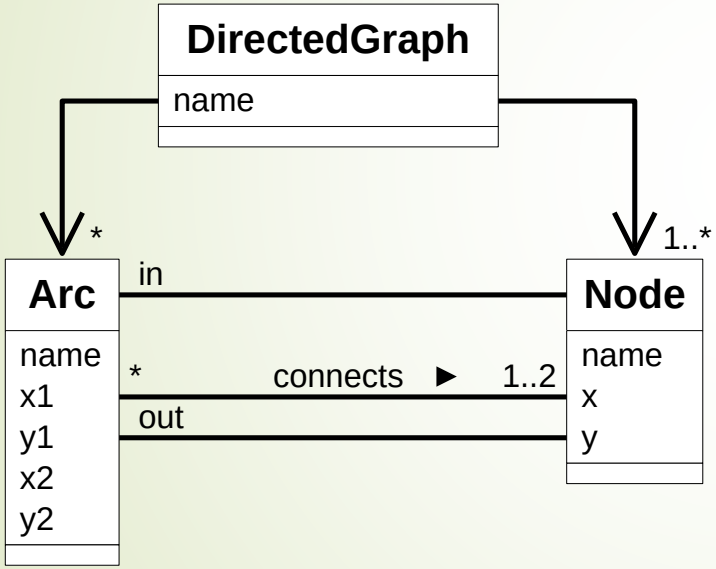
- The class diagram is a connected graph, which describes the structure of the solution in the problem space, where
  - a node represents a class, and
  - an edge represents a relation between two classes
- Possible relations between classes:
  - association
  - dependency
  - aggregation
  - composition
  - inheritance
- Note: inheritance acts between classes, while other relations act between the instances of classes

Association

- A *connection* of two or more classes with a kind of relation
- It can be reflexive, so we connect the objects of the same class
- An association may have a name or *identifier*
- It may have a direction, which goes from the active to the passive object
- Connected objects may have multiplicity and role, and also the connection can have a qualifier
- We can define navigability to express which objects know about each other
- By leaving navigability we assume that the connection is mutual (both objects know about each other)

Assocation

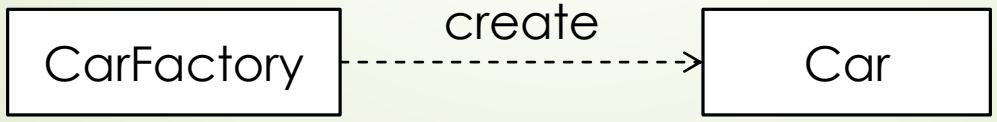
- One-way navigaibility: an arrow at the end of the connection
- Role of association: in, out
- Multiplicity: *, 1..*, 1..2




```
class DirectedGraph{
  private String name;
  private List<Arc> arcs;
  private List<Node> nodes;
}

class Node{
  private String name;
  private List<Arc> inArcs;
  private List<Arc> outArcs;
}

class Arc{
  private String name;
  private Node inNode;
  private Node outNode;
}
```

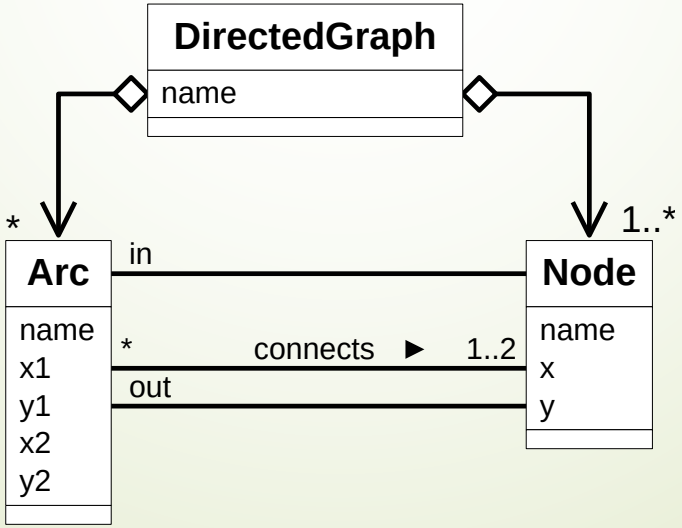
Dependency

- A class depends on an other class, if we can see the name of the other class as a parameter type, or local variable type in a method of the independent class.
- Difference from association: in association the instance of the dependent class is an attribute of the independent class
- In case of dependency the relation can be sometimes so weak that no local variables are introduced with the type of the dependent class



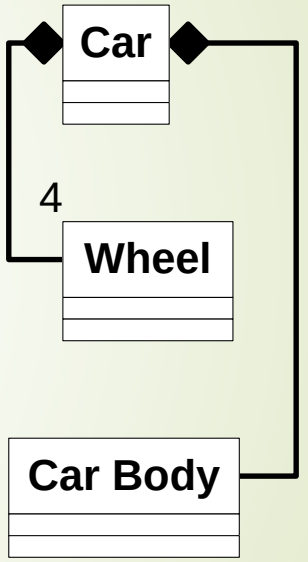
Aggregation

- A special association
- This relation is stronger than the general association, e.g.:
  - A whole and its parts,
  - A structure and its componens
- It describes that the objects of a class are parts of an object of an other class
- Aggregation is transitive, antisymmetric, and non reflexive



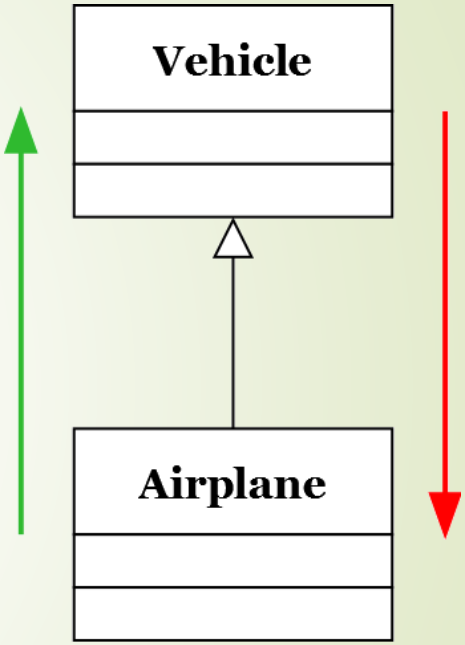
Composition

- Special aggregation
- The objects of a class physically contain the objects of an other class
- The realization of aggregation and composition in Java is the same, only UML makes a difference between them
- A component object can belong to at most one host object, while
- A host object can have any number of components
- The host object and its components have the same life cycle, so they are created (destroyed) at the same time



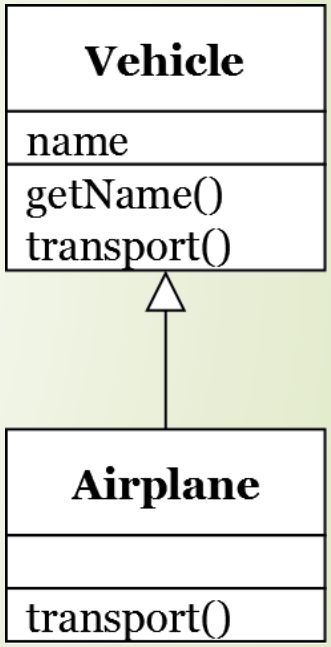
Generalization and specialization

- Acts between two classes
  - General class (super class)
  - Has general (common) properties
  - May have abstract methods
  - Special class
    (derived class, subclass)
  - Has special properties
    (besides the common ones)
  - Inherits the properties of the general class,
    and may extend or override them
- Example: the plain " is a kind of " vehicle



Generalization and specialization

- The general class has its attributes and methods as shown earlier
- The specialized class inherits all attributes and methods of its super class, so we can omit these on the diagram
- We have to show only those attributes and methods on the diagram for the specialized class, which are declared or overriden in it.



Generalization and specialization

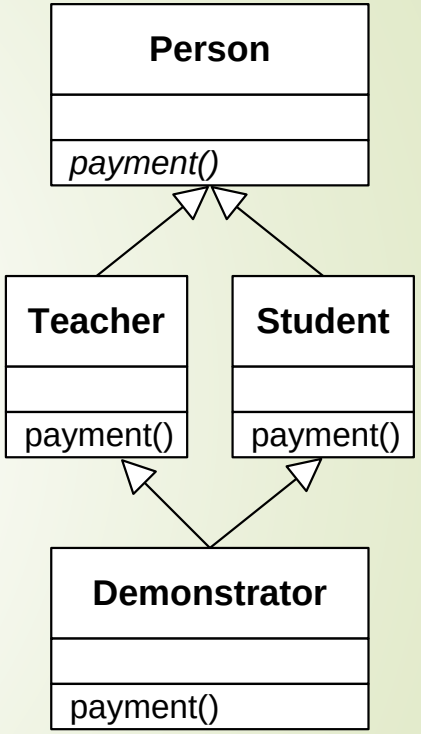
- Specialization is realized using derivation

```
class Vehicle {...}
class Airplane extends Vehicle {...}
```

- Derivation cannot be symmetric or reflexive
- Derivation is a method to create new classes, with we can create both abstract and concrete classes
- Specialization can be multiple, so many classes can be derived from a general class.
- Generalization also can be multiple, so a class may be derived from several classes
  - In Java, a class can be derived from only one class, while it can implement many interfaces.

Generalization and specialization

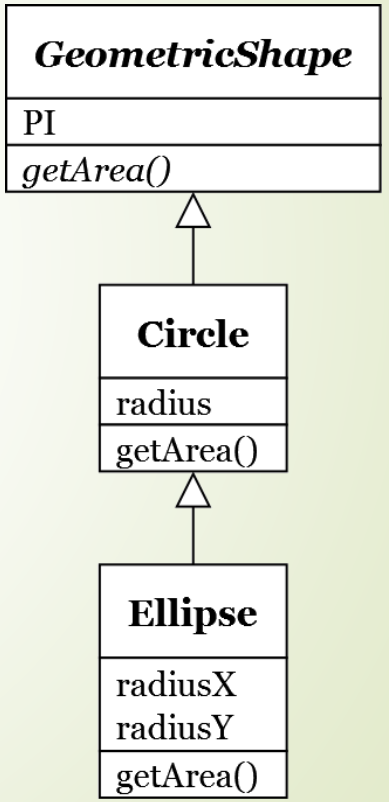
- A general class can be derived multiple times
- and a special class may have the properties of several general classes
- In case of multiple generalization, constraints should be applied to make the programming language able to distinguish clearly between the inherited attributes/methods
- Applying further specializations, we can define a class hierarchy



Inheritance

All inherited methods can be overriden (except...), but overriding works only, if the signature of the method in the general class and in the special class is exactly the same

```
public abstract class GeometricShape{
    protected final double PI = Math.PI;
    public abstract double getArea();
}

public class Circle extends GeometricShape{
    protected double radius;
    @Override
    public double getArea(){ return radius * radius * PI; }
}

public class Ellipse extends Circle{
    private double radiusY, radiusX = radius;
    @Override
    public double getArea(){
        return radiusX * radiusY * PI; }
}
```



Inheritance

- Abstract method
  - All methods can be declared without a realization (body), (except, when a method or its class is `final`).
    In this case the method will be abstract.
- Abstract class
  - If a class has an abstract method, the class becomes also abstract.
  - If a derived class has an abstract super class, then it has to define/realize all the derived abstract methods, or it has to be abstract
  - An abstract class cannot be instantiated

Inheritance

- In Java, the super class can be referenced with the `super` keyword.
- The methods can be marked as `final`, if we do not want them to be overriden later
  (an abstract method cannot be final).
- Even the whole class can be marked as `final`
  (if it is not abstract), so we can forbid the derivation of it.

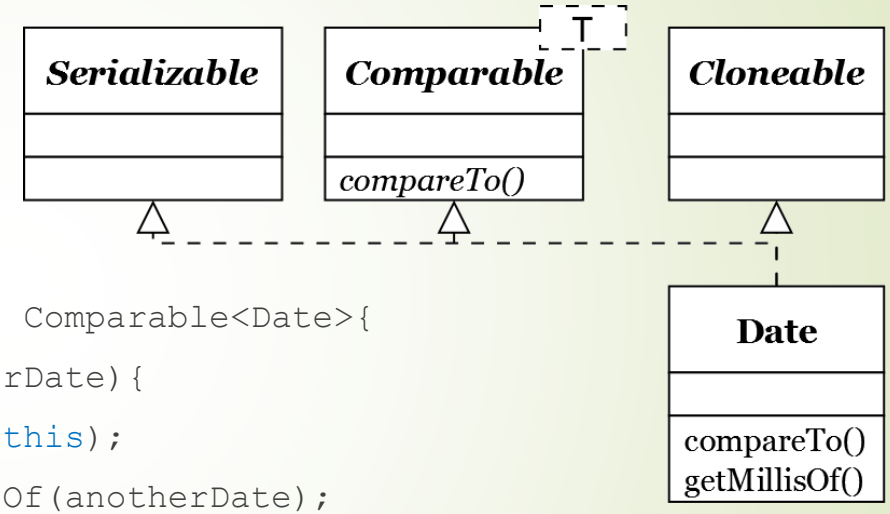
Inheritance – Interface

- An interface is a completely abstract class, and it can be absolutely empty as well (e.g.: in case of a type annotation).
- An interface does not have a realization, only at most some constant attributes
  - None of its methods has a body
- Deriving a class from an interface is called implementation or realization
- Java interfaces are similar to the C++ header files
- Interfaces can also be derived from interfaces
- Java let us use only interfaces as more than one super class (we can derive only from one class)
- An interface is realized with a concrete class

Inheritance – Interface (example)

- In this case the Comparable interface is also a generic type, but this is only a coincidence.

```
public interface Comparable<T> {
  public int compareTo(T o);
}

public class Date implements
    java.io.Serializable, Cloneable, Comparable<Date>{
    public int compareTo(Date anotherDate){
        long thisTime = getMillisOf(this);
        long anotherTime = getMillisOf(anotherDate);
        return (thisTime<anotherTime ? -1 :
                (thisTime==anotherTime ? 0 : 1));
    }
}
```



Inheritance

- The abstract class can be realized during the instantiation on-the-fly
  (this time an anonymous class is created)

```
GeometicShape shape = new GeometicShape(){
    @Override
    public double getArea(){
        return 0.0;
    }
}
```

Inheritance – Polymorphism

- Because inheritance is an " is a kind of..." relation, we can use the specialized classes in place of the general class
- Polymorphism: a `Square` object can have the properties of a `Square` class, but it also have the properties of the general `GeometricShape` class (because it inherits them).

```
List<GeometricShape> shapes = new ArrayList<>();
shapes.add(new Circle());
shapes.add(new Ellipse());
shapes.add(new Square());

for (GeometricShape geomShape : shapes) {
    System.out.println(geomShape.getArea());
}
```