FROM RESEARCH TO INDUSTRY

cea tech

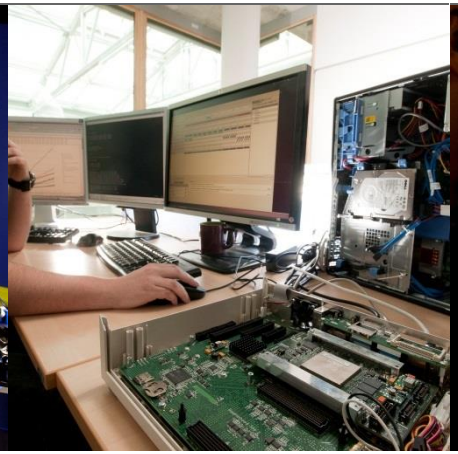
list

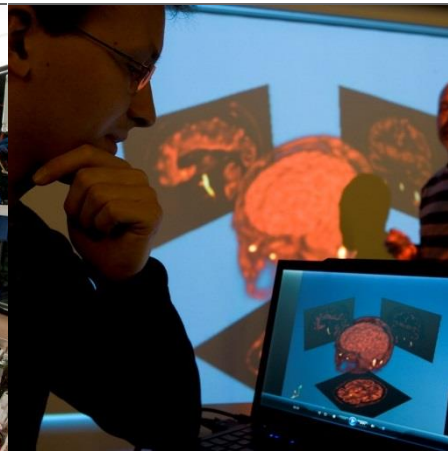
STRUCTURE MODELING

Introduction on UML for Industrial Systems

Shuai Li, Jérémie Tatibouët, François Terrier, Sébastien Gérard

{first_name}.{last_name}@cea.fr





INSTITUT CARNOT CEA LIST

digiteo

CONTENT

CONTENT

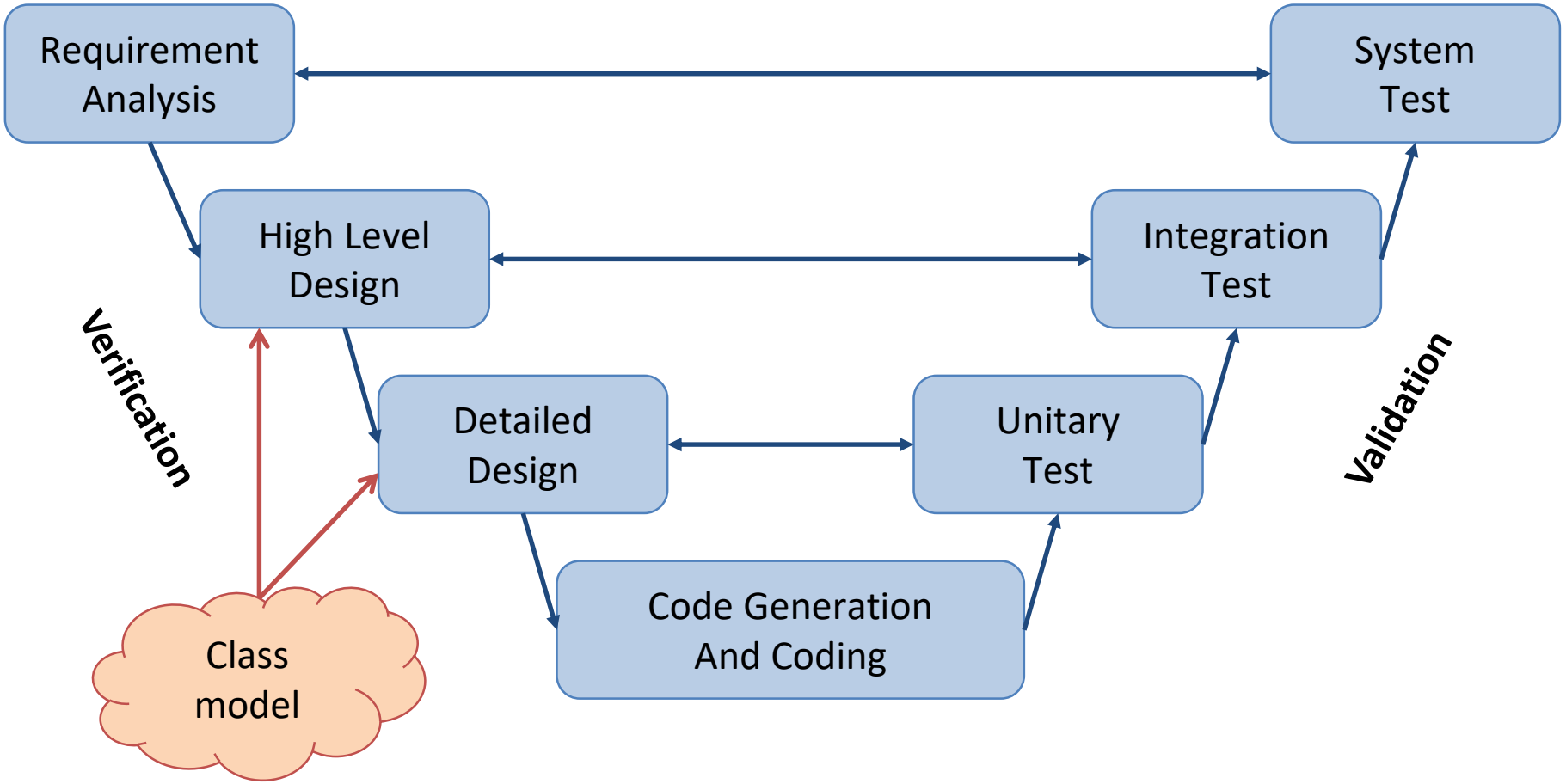
Why model the structure of a system?

- Use-case are used to model WHAT are the functionalities of the system
- The system is actually composed of entities that interact with each other, and actors, to realize a use-case
- Modeling the structure of the system describes WHO will fulfill the functionalities of the system, and the relationships between these entities

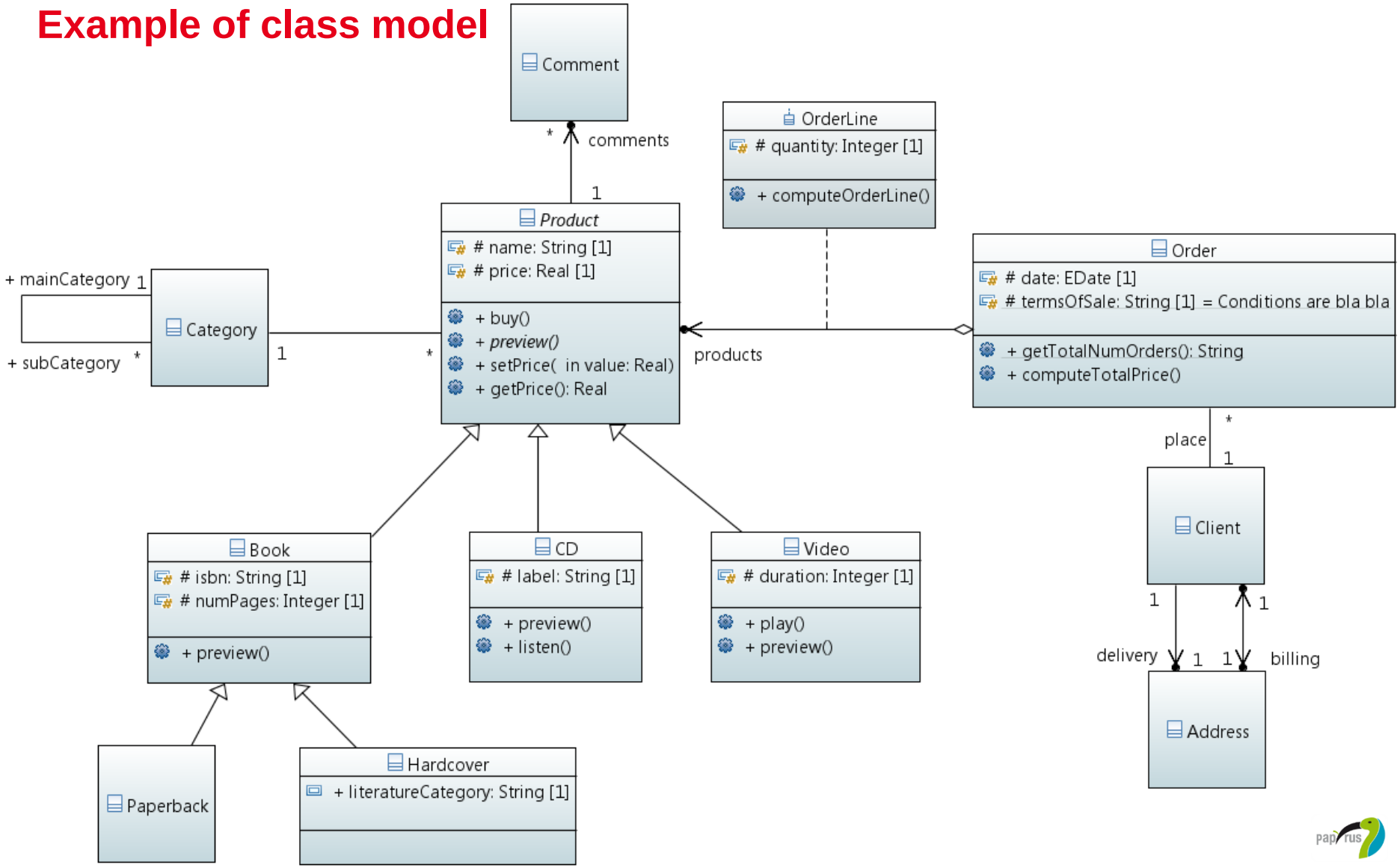
Structure modeling in UML

- Structures can be modeled in UML through a class model, with class diagrams

When to model classes?



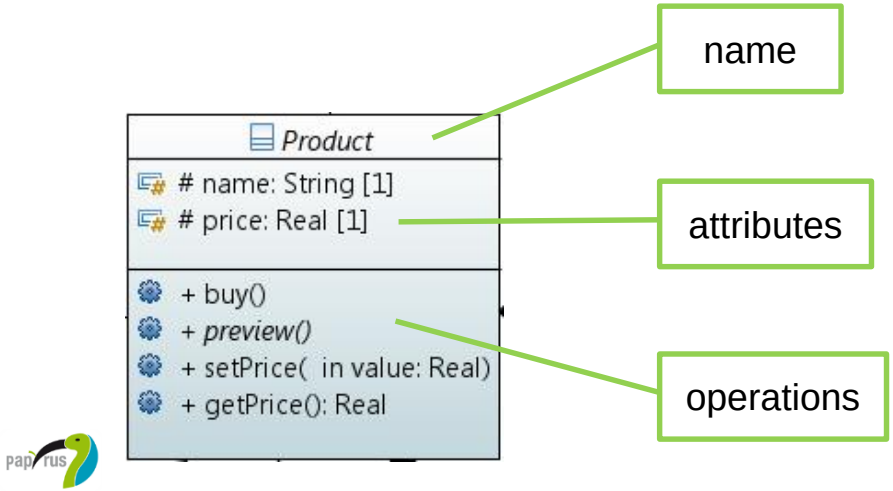
Example of class model



CONTENT

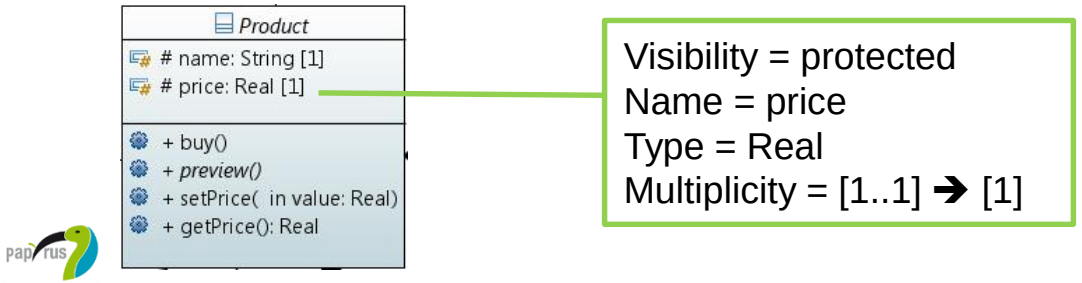
Overview

- Description of a set of objects with common semantics, features, and constraints
- Among features, a class has attributes and methods
- In UML, features are gathered within compartments: a class has a name, a compartment of attributes, and a compartment of operations
- Example: a “Product” class



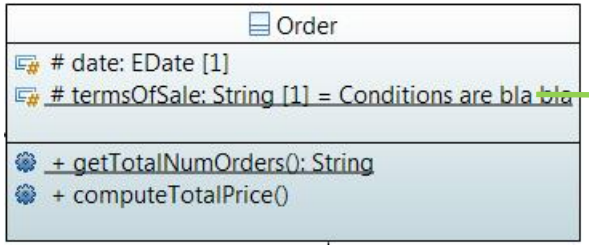
Attribute

- **An attribute describes data of the class, that take value when the class is instantiated as an object**
- **Specification of an attribute:**
  - Visibility: constraint on the access of the attribute (more on this later)
  - Name
  - Type: class that the attribute instantiates at runtime
  - Multiplicity: cardinality, i.e. number of elements
    - Syntax: [MultMin..MultMax]
    - * means several
    - n..n is also noted n
    - 0..* is also noted *
  - Default value
- **In UML, an attribute is modeled with a Property element (Properties are not only used for attributes)**
- **Example:**



Static attribute

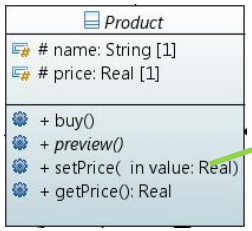
- An attribute takes value when the class is instantiated as an object…
- …unless the attribute is static
- A static attribute can be accessed without instantiating the class
- Example: “termsOfSale” is a static attribute since we want to access it without having to instantiate any “Order”



Static attribute with default string value (underlined)

Operation

- A class has methods with signature and body
- In UML, an operation is the specification of a method, i.e. the method signature, independently of its implementation
- In UML, the word “method” is used to designate an implementation of an operation, i.e. the body of a method in a class
- Specification of an operation:
  - Visibility
  - Name
  - Parameters:
    - Name
    - Direction: in, out, inout, return (only one return parameter)
    - Type
    - Multiplicity
    - Default value
- Example:

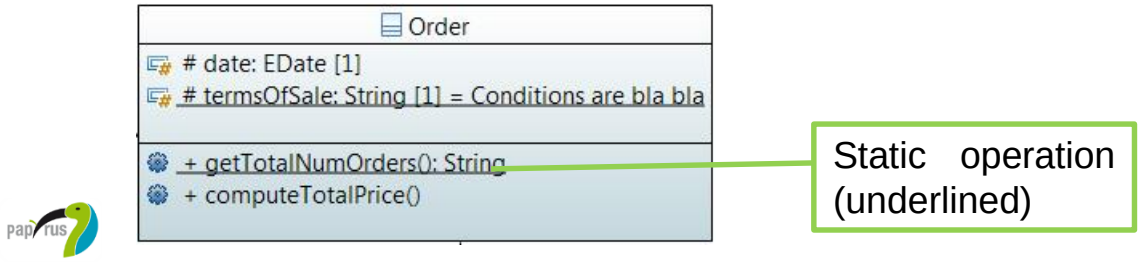


Visibility = public (“+”)
Name = “setPrice”
Parameter = “in value : Real”

Careful, the same “method” word in UML and OOP do not designate the same things. Yes this can be confusing…

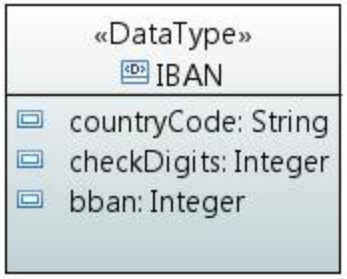
Static operation

- Usually an operation is accessed from the object instantiating the class…
- …unless the operation is static
- A static operation can be accessed without instantiating the class
- Example: “getTotalNumOrders” is a static operation since we want to access the total number of orders without instantiating an “Order”



DataType

- A class can type attributes, i.e. the attribute is an instance of the class at runtime
- DataType is similar to a class; it is typically used to represent value types of a certain domain, or primitives, or structured types
- Instances of a data type are identified by the values of the attributes
- Example: “IBAN” is a structured type, defined by a country code, check digits, and a BBAN

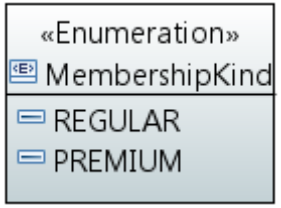


PrimitiveType

- An atomic data type, i.e. without structure
- UML primitive types: Boolean, Integer, UnlimitedNatural, String, Real
- Example:

Enumeration

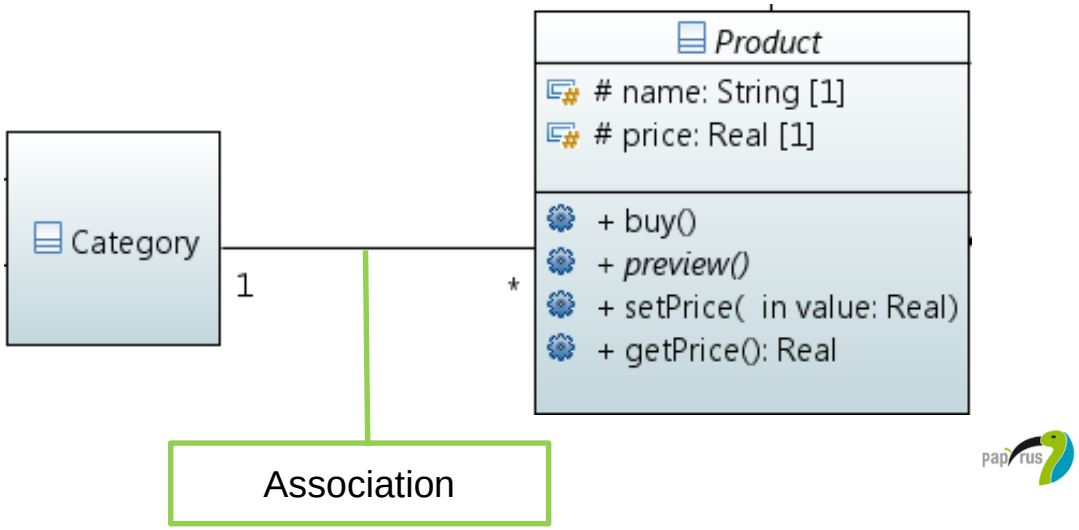
- Data type whose values are enumerated as user-defined enumeration literals
- Example:



CONTENT

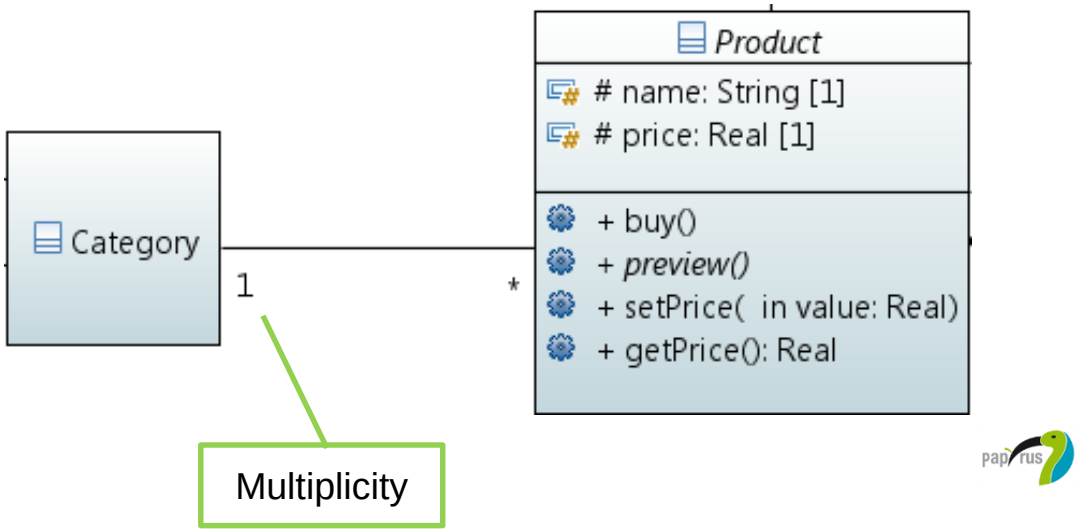
Overview

- An association is a semantic relationship between classes
- An association between classes represent links between objects instantiating the classes
- Example: A “Product” is associated to a “Category”



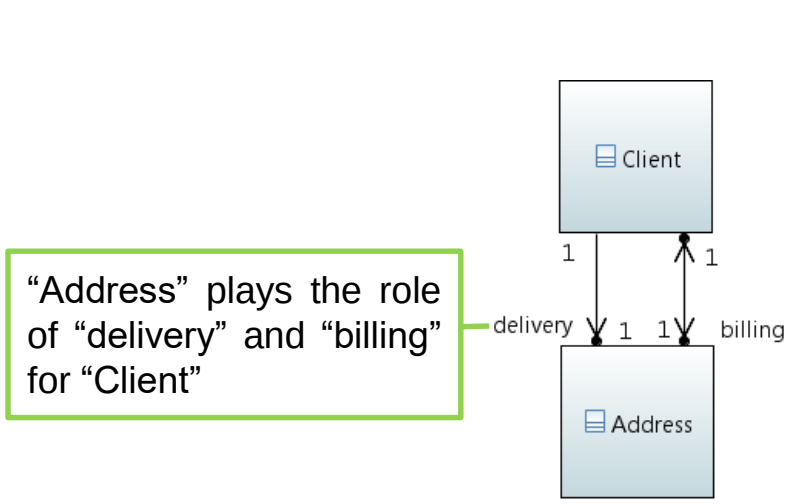
Multiplicity

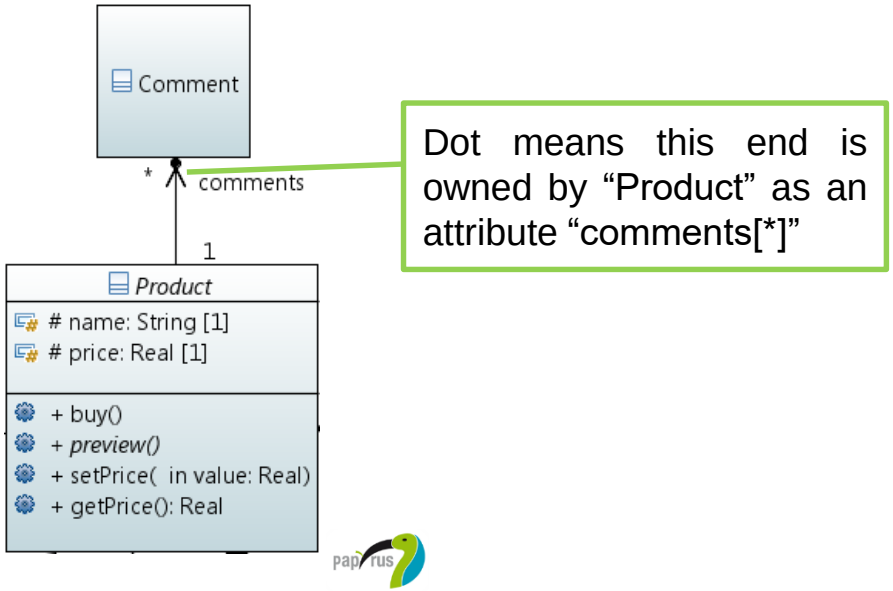
- Multiplicity allows to control the number of objects intervening in an instance of an association
- Example: A “Product” is in 1 “Category”; a “Category” contains 0 to many (*) “Products”



Association ends

- An association connects classes at ends
- The end connecting a class is commonly referred to as the role of the class involved in the association
- In UML, an end is a Property that can be owned by:
  - The association
  - The class connected by the opposite end: it is then an attribute of the class
- Example:

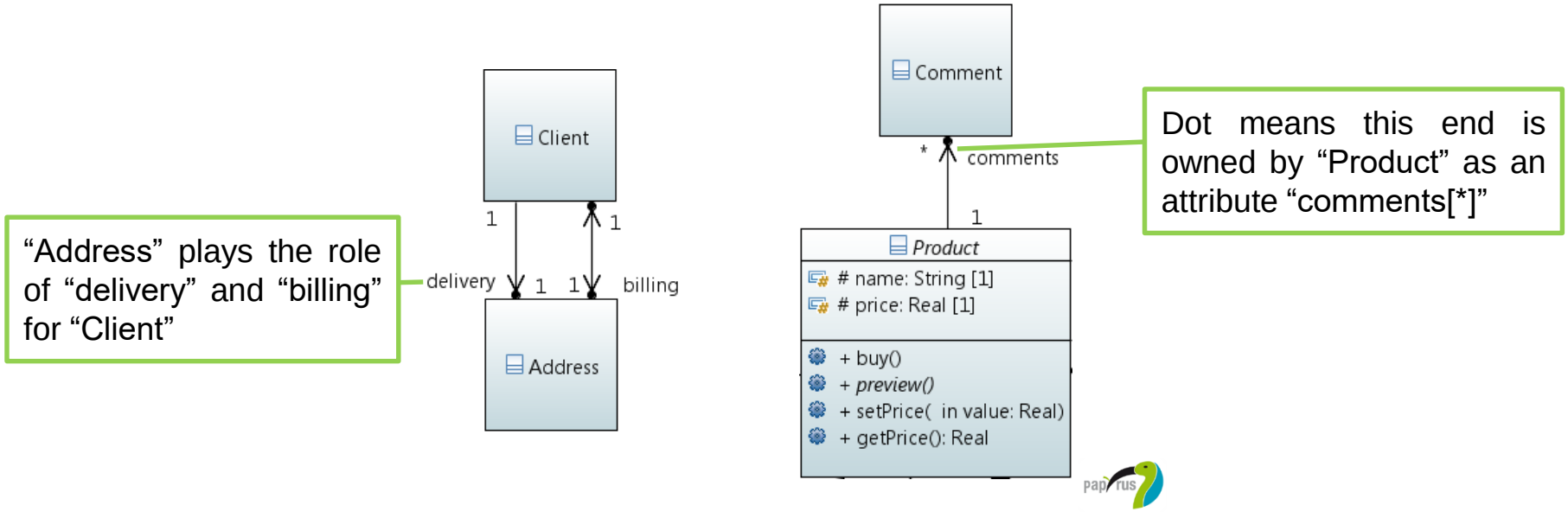




Association ends

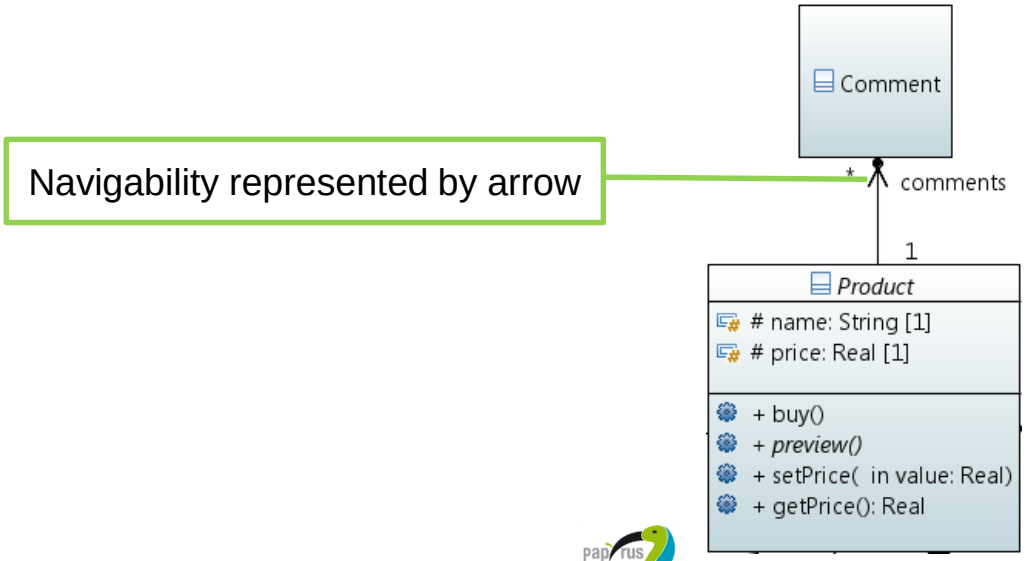
- An association connects classes at ends
- The end connecting a class is commonly referred to as the role of the class involved in the association
- In UML, an end is a Property that can be owned by:
  - T…
  - T… s… the class
- Example:

Here you see that a property in UML represents not only a class attribute, but also an association end.

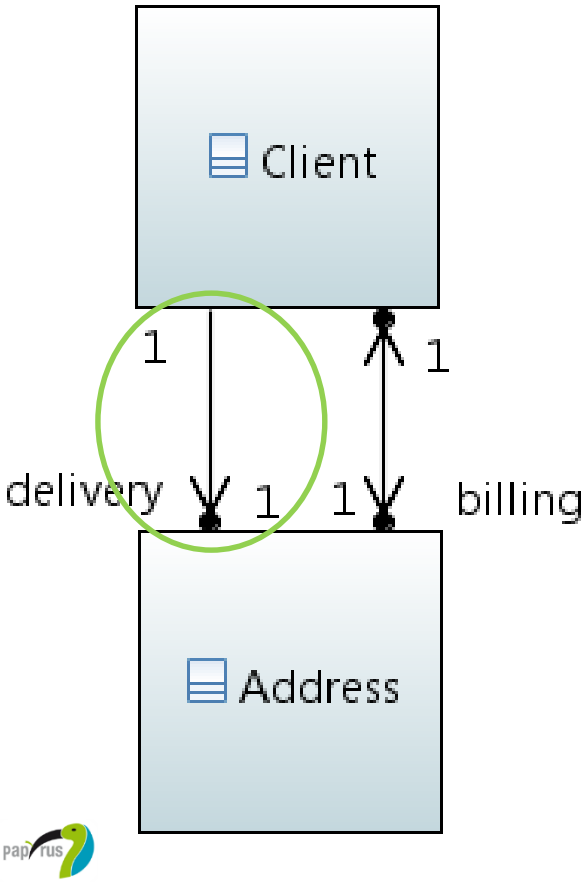


Navigability

- An end A may be navigable: objects instantiating the class at the opposite end B may access “efficiently” objects instantiating the class at A
- An end may also be non-navigable, or have unspecified navigability
- Both ends of an association may be navigable
- Example: From a “Product”, we can access its “Comments”; this does not necessarily mean that from a “Comment” we cannot access the “Product”!

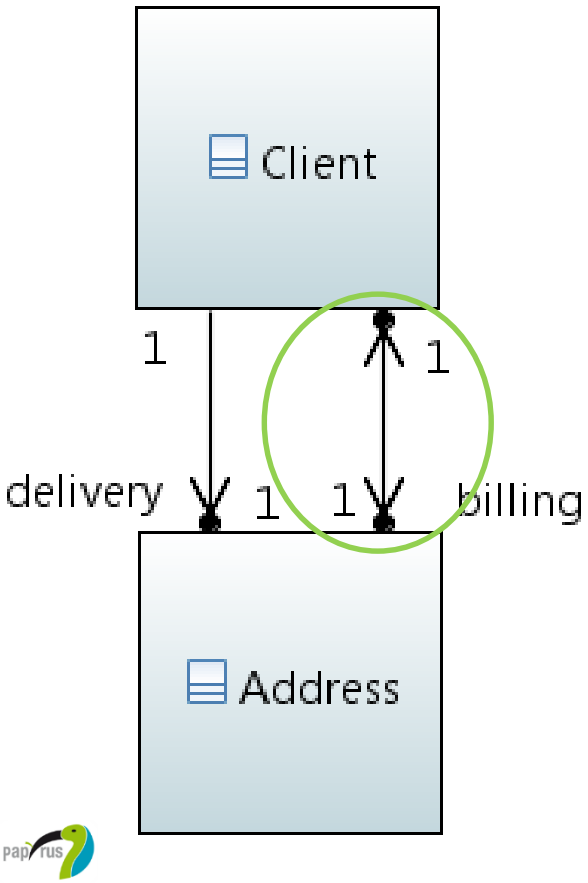


Example: implementation of unidirectional [1..1] association



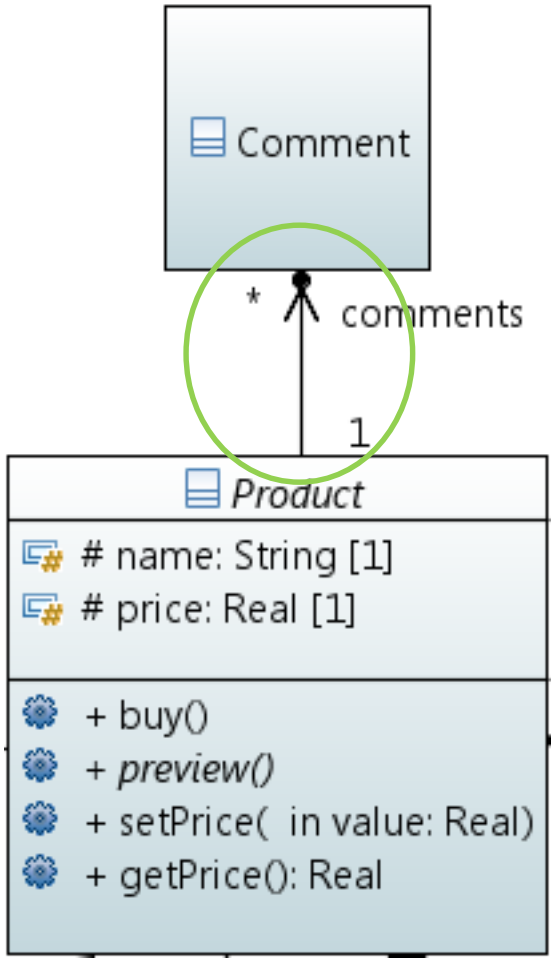

```
public class Address{…}

public class Client {
  private Address delivery;

  public void setDelivery(Address address) {
    this.delivery = address;
  }

  public Address getDelivery() {
    return delivery;
  }
}
```

Example: implementation of bidirectional [1..1] association



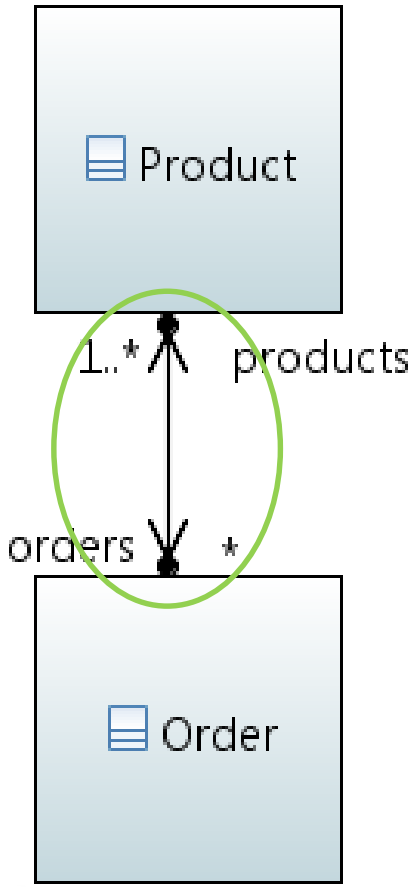

```
Public class Client {
  private Address billing;

  public void setBilling(Address address) {
    this.billing = address;
    address.client = this;
  }
}

Public class Address{
  private Client client;

  public void setClient(Address address) {
    this.client = client;
    client.billing = this;
  }
}
```

Example: implementation of unidirectional [1..*] association



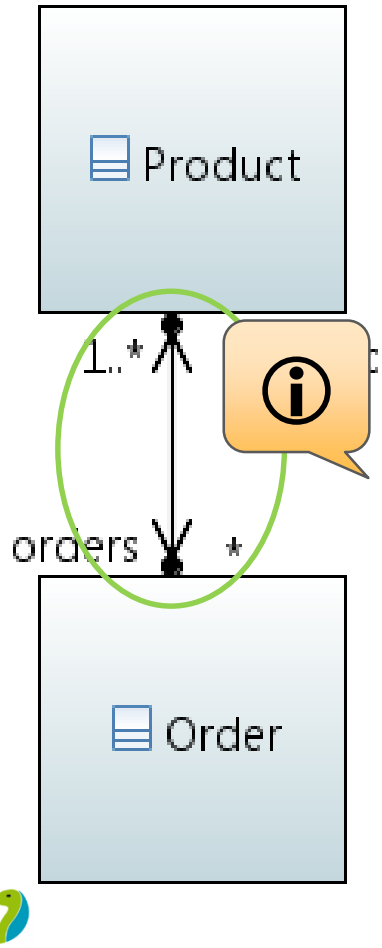

```
public class Comment{…}

Public class Product{
  private List comments = new ArrayList();

  public void addComment(Comment comment) {
    comments.add(comment);
  }

  public void removeComment(Comment comment) {
    comments.remove(comment);
  }
}
```

Example: implementation of bidirectional [*..*] association




```
public class Product{…}

Public class Order {
  private List products = new ArrayList();

  public void addProduct(Product product) {
    products.add(product)
    products.get(product).addOrder(this)
  }

  public void removeProduct(Product product) {
    products.get(product).removeOrder(this);
    products.remove(product)
 }
}
```

Example: implementation of bidirectional [*..*] association

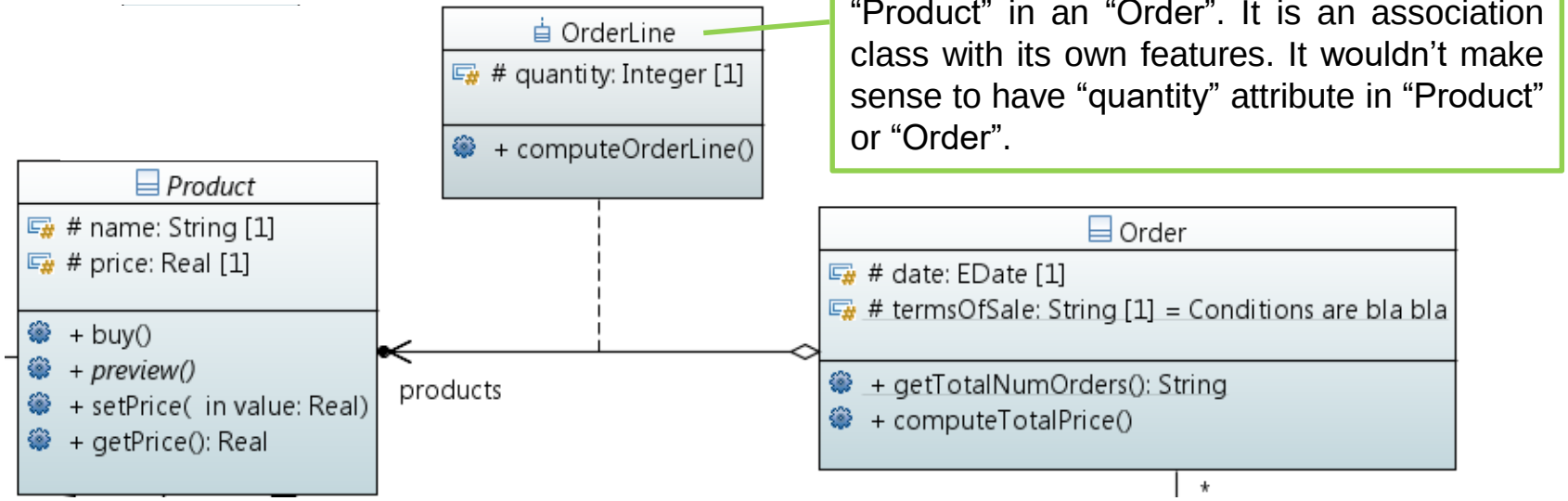


In reality it is more complicated since you need to handle coherence of collections for example.

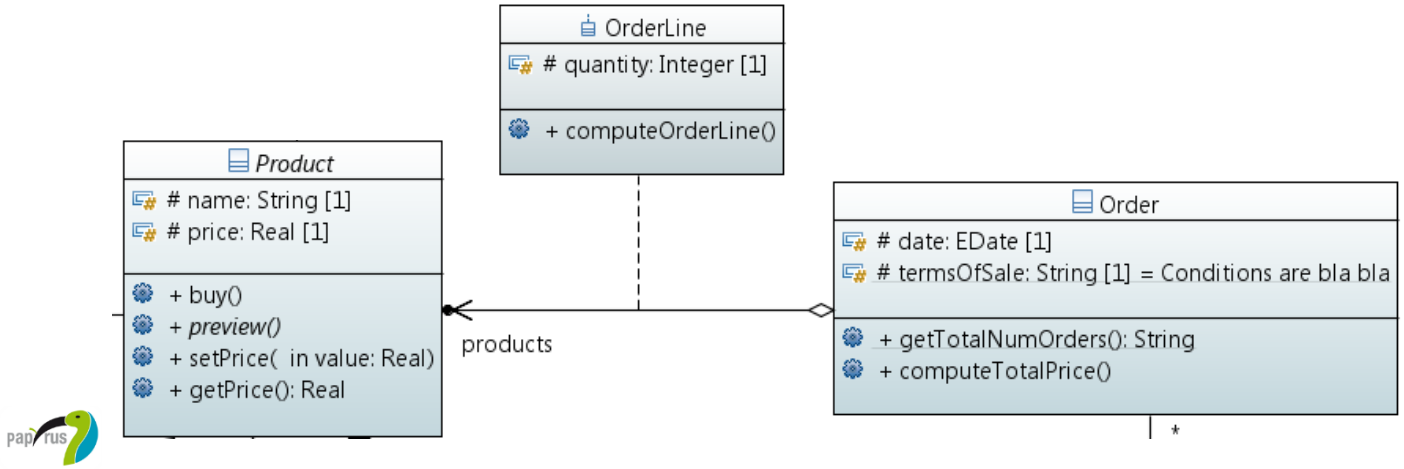
```
public class Product{…}

Public class Order {
  private List products = new ArrayList();

  public void addProduct(Product product) {
                                    e(); i++) {
                                    ;
    }
  }

  public void removeProduct(Product product) {
    products.get(product).removeOrder(this);
    products.remove(product)
  }
}
```

Association class

- An association may need to have its own features (attribute and operation), that are not in the classes its ends connect
- We then use an association class
- An association class is both an association (i.e. relationship connecting classes) and a class (i.e. it can have features and might be included in other associations)
- Graphical notation: a class linked to an association with dashed line
- Example:



Association class



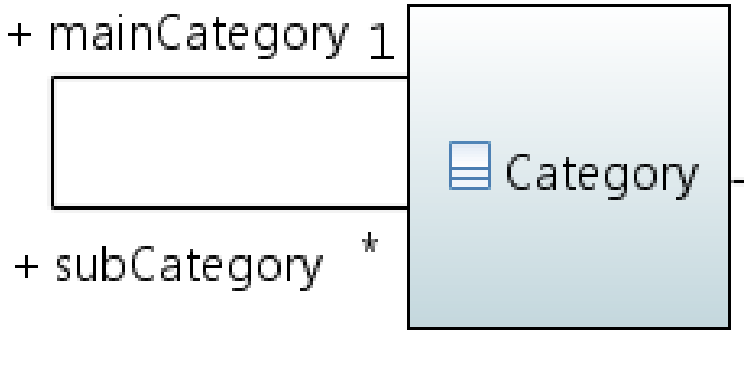
How would you implement this?

There is no unique way to implement an association class.

```
public class Product{…}
public class Order{…}

public class OrderLine {
  protected Product product;
  protected Order order;
  protected int quantity;

  public computeOrderLine() {…}
}
```

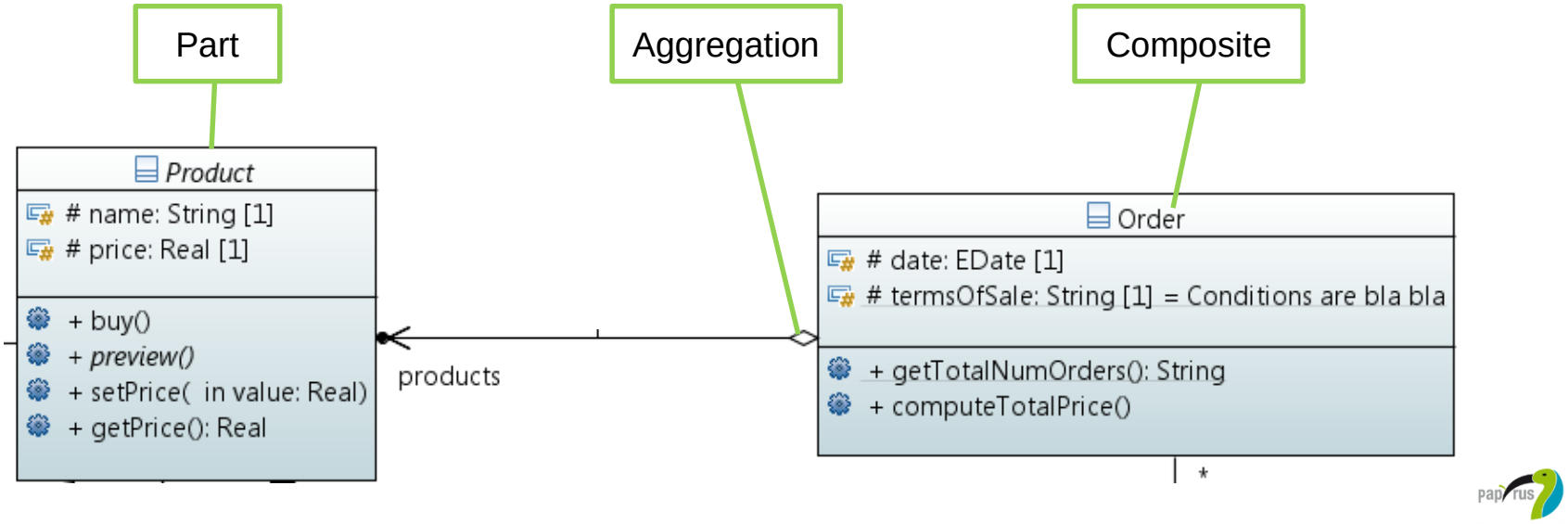
Reflexive association

- Usually an association is between two classes, i.e. a binary association
- But an association can point to the same class ➔ reflexive association
- Example: A main category has several (*) sub categories



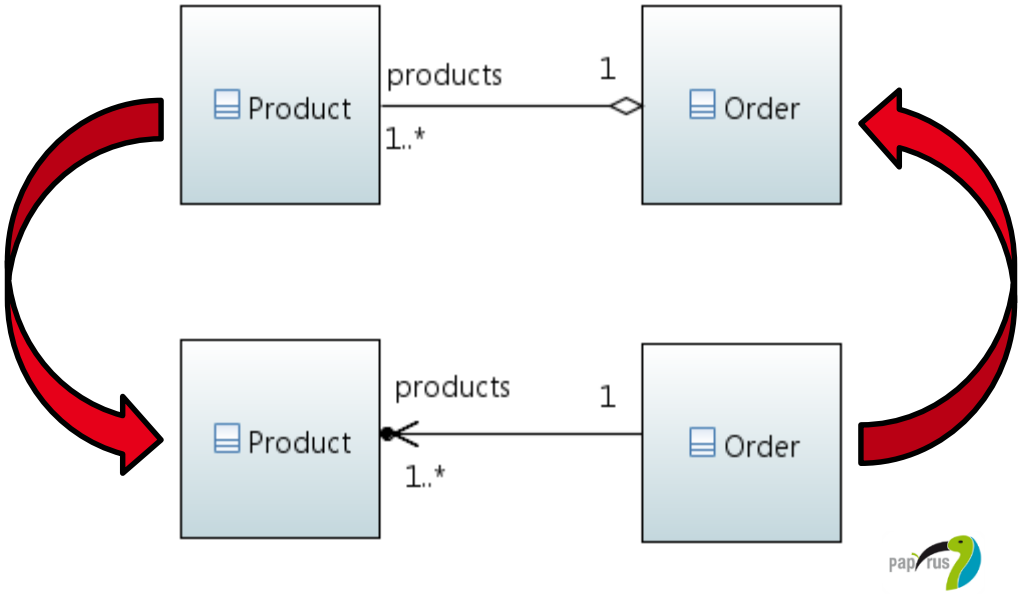
Aggregation

- **An aggregation is a binary association representing a whole/part kind of relationship**
- **In an aggregation, the part is independent of the composite:**
  - A same part can be in several composites
  - When a composite is destroyed, the part may still exist
- **Example: “Products” are part of an “Order”**



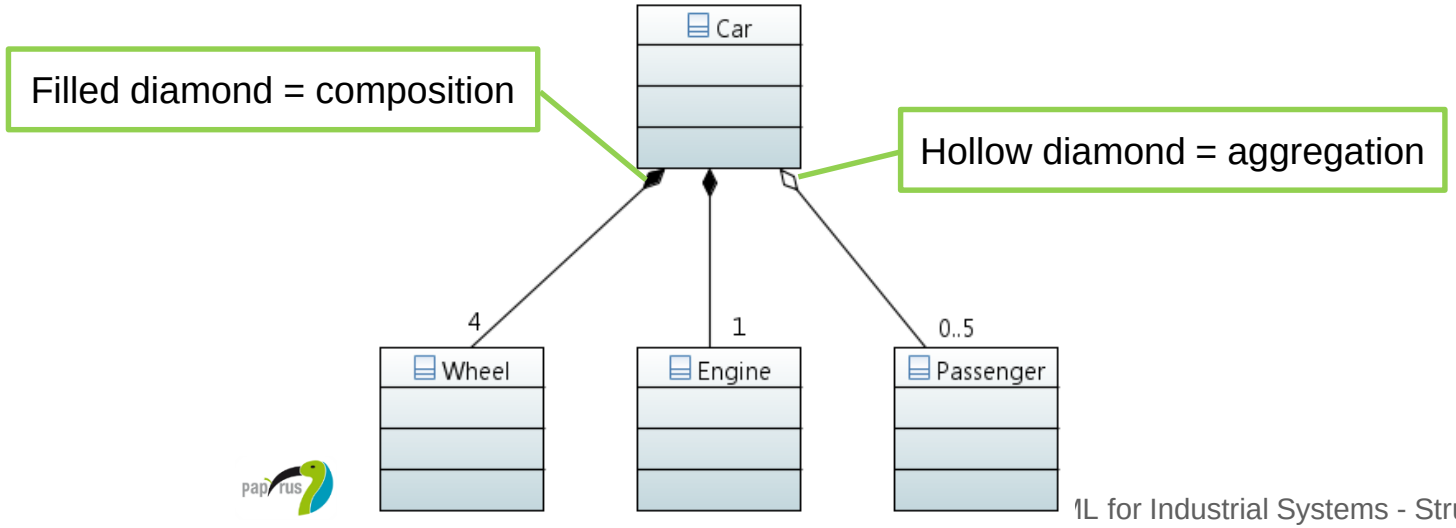
Aggregation

- Unless anything specific is defined in the development method, an aggregation may be redundant since it can be represented with an association
- Example:



Composition

- **A composition is a “strong” form of aggregation**
- **In a composition, the part is dependent of the composite:**
  - A same part can only be in one composite
  - When a composite is destroyed, the part is destroyed
- **Graphical notation: filled diamond at the end connecting the composite**
- **Example: “Car” has 4 “Wheels”, 1 “Engine”, and 0 to 5 “Passengers”; if the car is destroyed, the wheels and engine are destroyed, not the passengers (hopefully ☺)**



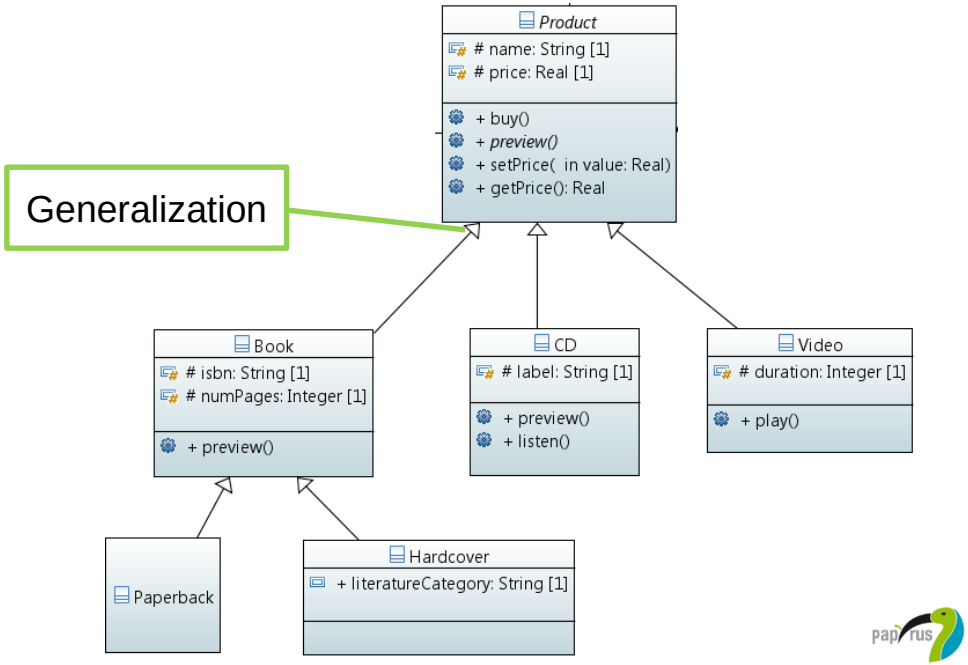
Aggregation vs. composition: which one to use?

- **Ask yourself the following questions:**
  - If the composite object is destroyed, does this necessarily imply the destruction of the part objects?
  - When I copy the composite object, do I have to also copy the part objects, or can I simply re-use them, e.g. by referencing them?
- **If the answer is yes to these two questions, then a composition is necessary**

CONTENT

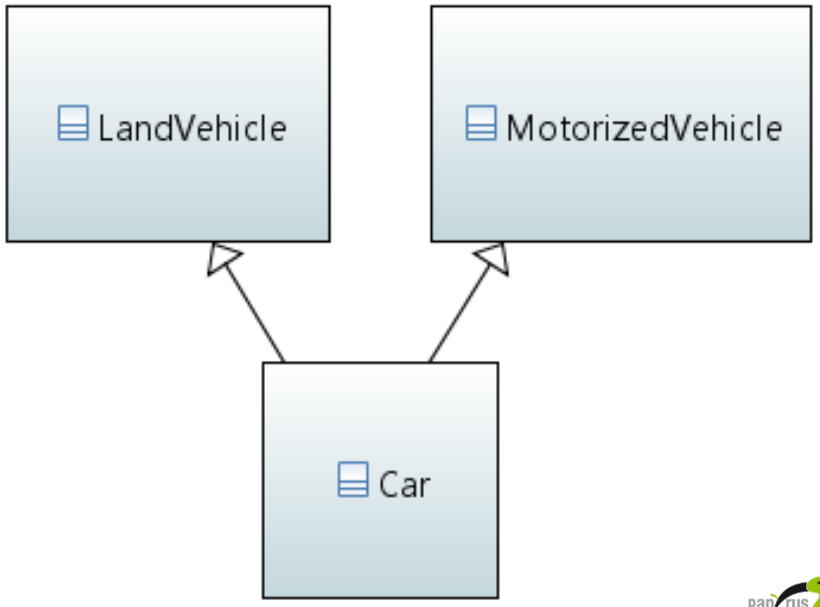
Overview

- **Generalization** used in UML to represent inheritance
- Relationship between a general class and a more specific class that inherits its features, but may override them.
- Example: “Book”, “CD”, and “Video” are products so they inherit “Product”; e.g. they inherit the “price” attribute and can override the “preview” method



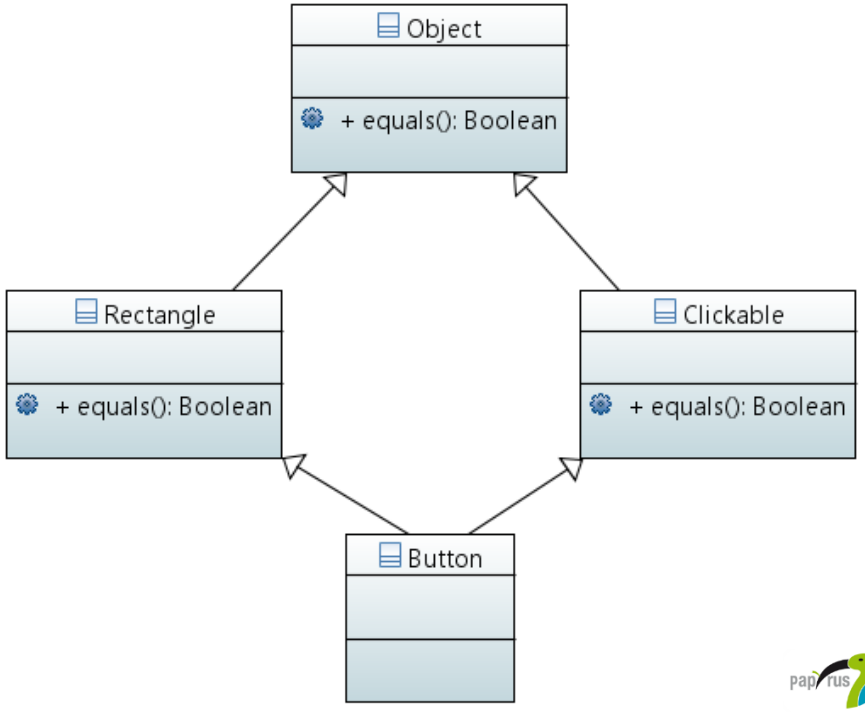
Multiple inheritance in UML

- UML allows multiple inheritance, i.e. a class inherits several other classes
- Example: “Car” is both a “LandVehicle” and a “MotorizedVehicle”



Multiple inheritance in UML

- No clear definition, and no explicit resolution for well-known ambiguities, in UML specification document
- Example: “Button” inherits “Rectangle” and “Clickable”, which both have an “equals” method. Which one is called at runtime?



- Multiple inheritance not allowed in some languages, i.e. Java

CONTENT

Encapsulation

- An element hides its internal data and state to the outside world
- Packaging of elements into a single element
- External access to the elements is controlled (e.g. through methods)

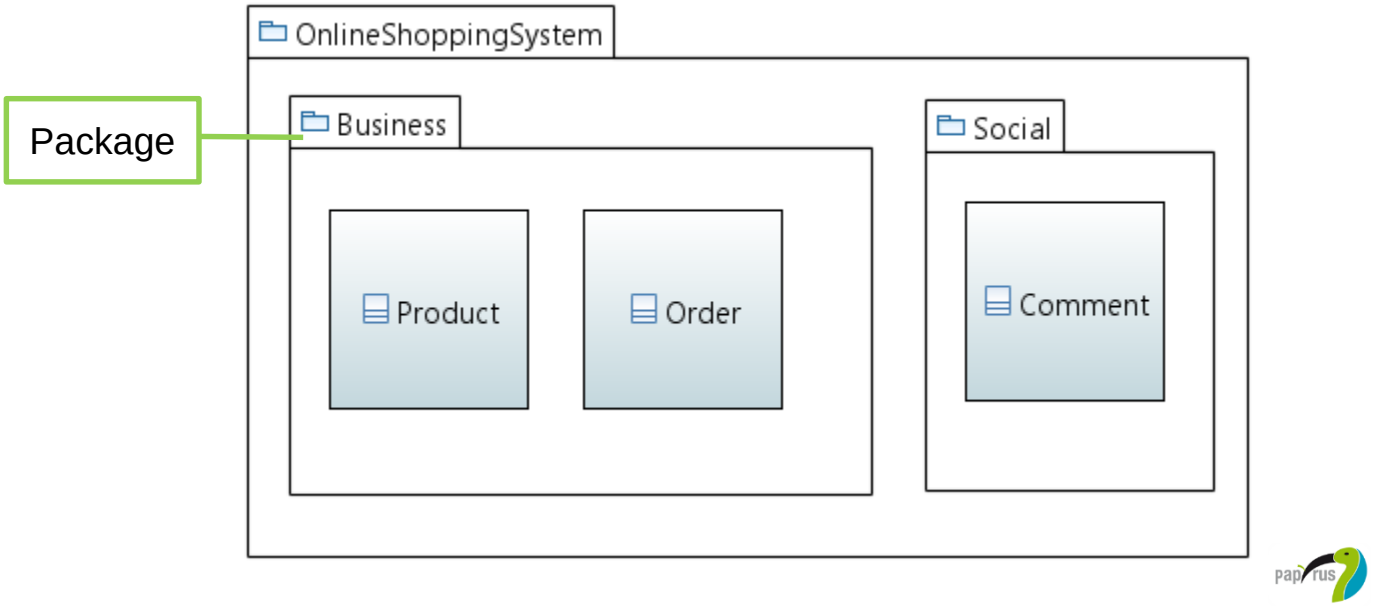
Namespace

- Set of symbols to organize objects so they may be referred by name
- Within UML, a namespace is any named element (i.e. an element that may have a name) that contains a set of other named elements
- Example of use: How to distinguish two classes both named “Comment”? Answer:
  - com.cea.onlineshoppingsystem.social.Comment
  - org.eclipse.papyrus.Comment

How to model encapsulation and namespace in UML?

Package

- Package is a namespace used to group together elements (e.g. class, package)
- Organize elements into groups to provide better structure for system model
- Example:



Visibility

- An encapsulated element has a visibility that specifies to whom it is visible
- Visibility of an element can be defined by access modifiers

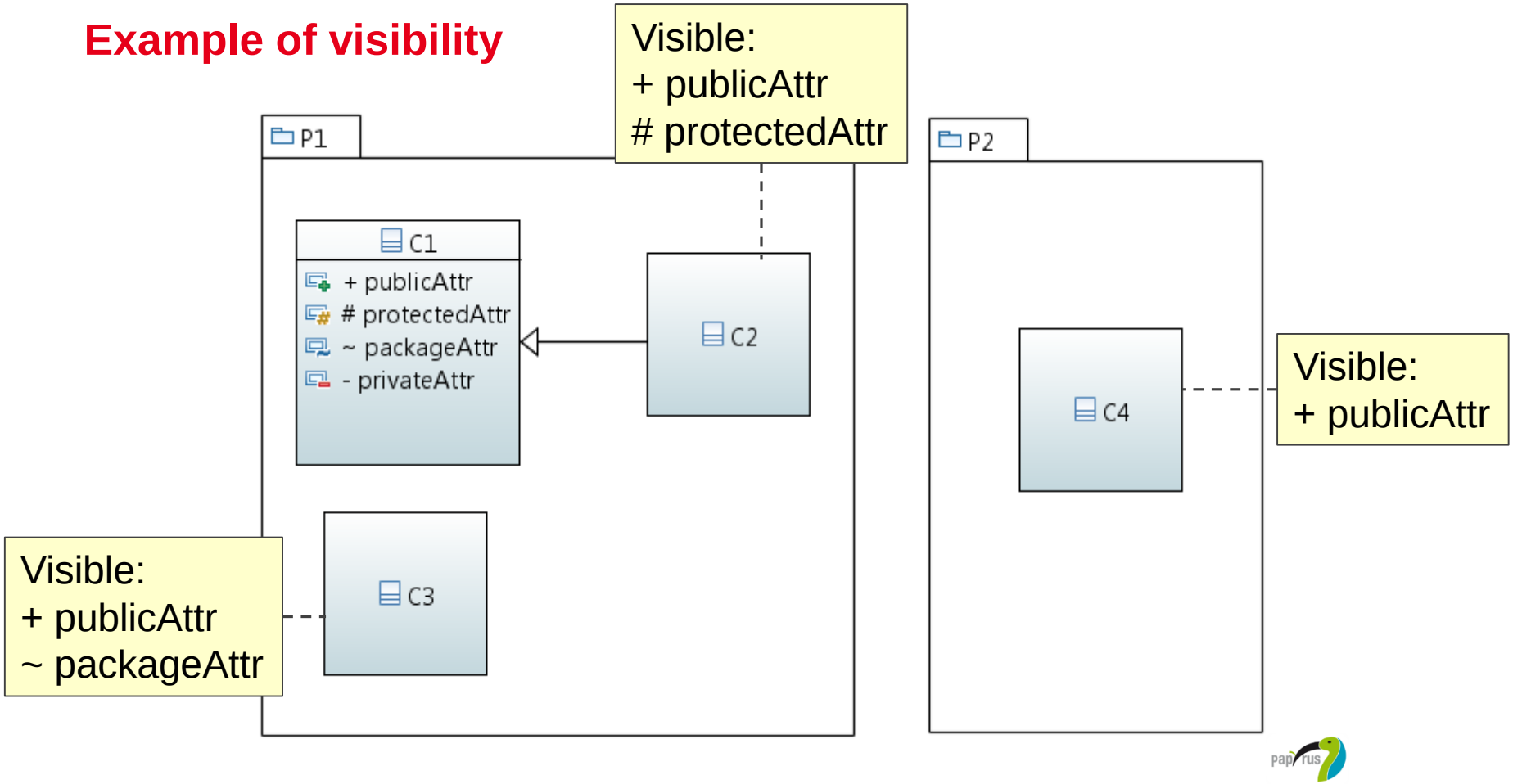
Visibility of features in a class

Modifier	Class	Package	Child class	World
public “+”	Yes	Yes	Yes	Yes
protected “#”	Yes	No	Yes	No
package “~”	Yes	Yes	No	No
private “-”	Yes	No	No	No

What are the access modifiers for a class in package?

Public and private. “Package" makes no sense since "private" already means the class is only visible to other elements by its owner, i.e. its package. In UML, “protected" can only be applied to elements that are not owned by a package.

Example of visibility



?

Which attributes of C1 are visible to C2, C3, C4?

CONTENT

Abstraction

- Hide details and only show the essential features of some object A
- Another object B has an outside view of A, i.e. the interface of A
- Through abstraction the implementation of A is hidden to B and changes to A does not necessarily mean changes to B if interfaces of A do not change

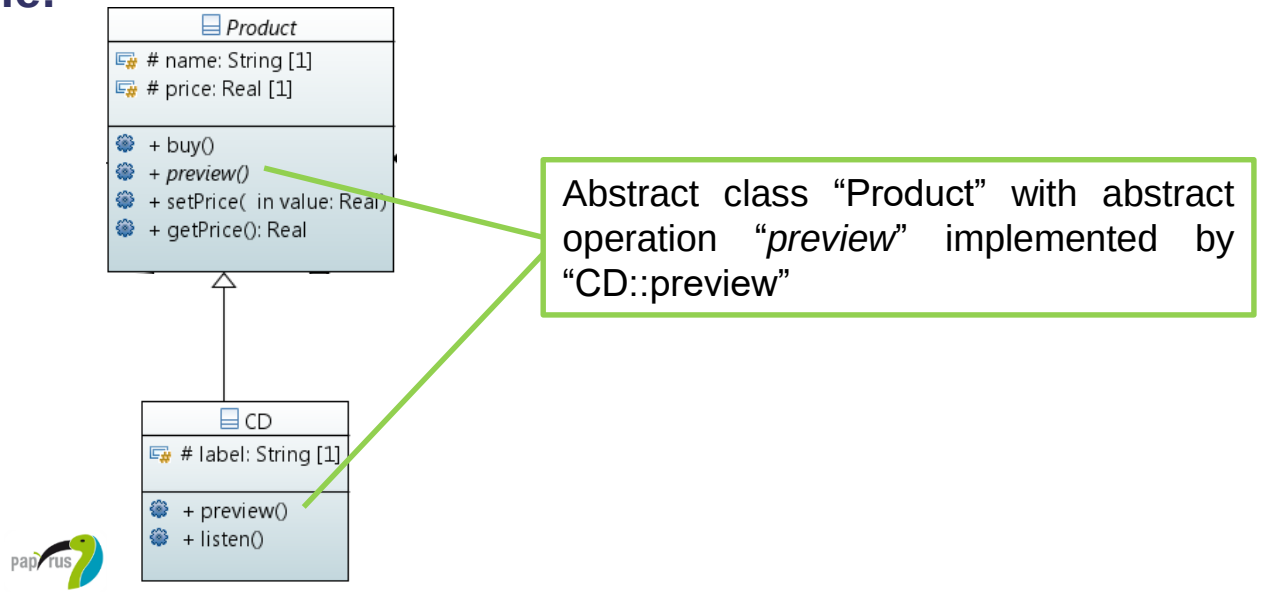
Difference between encapsulation and abstraction?

Abstraction is more generic. Encapsulation is a strategy used as part of abstraction.

How to model abstraction in UML?

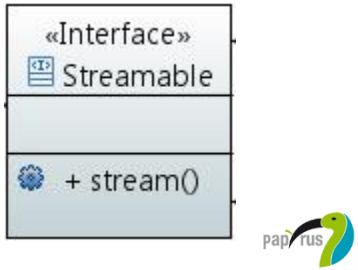
Abstract class

- An operation is said abstract if we know its signature, but its implementation is not specified in its class
- If class A has an abstract operation, then one of its children B should implement the operation
- A class is said abstract if it has an abstract operation, or if one of its parents has an abstract operation that hasn't been defined yet
- Graphical notation: italic name of class or operation
- Example:



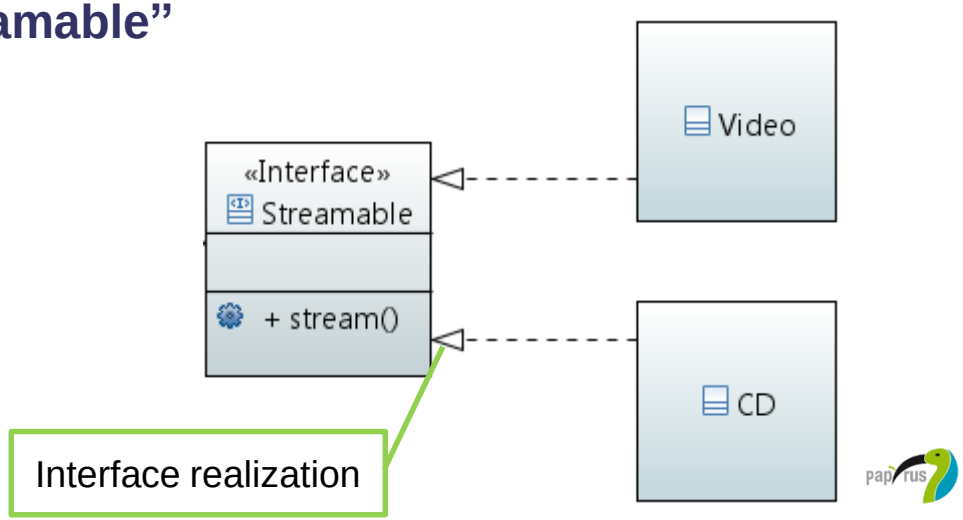
Interface

- An interface declares a set of coherent public features
- An interface defines a contract that must be fulfilled by any class implementing the interface
- Example:



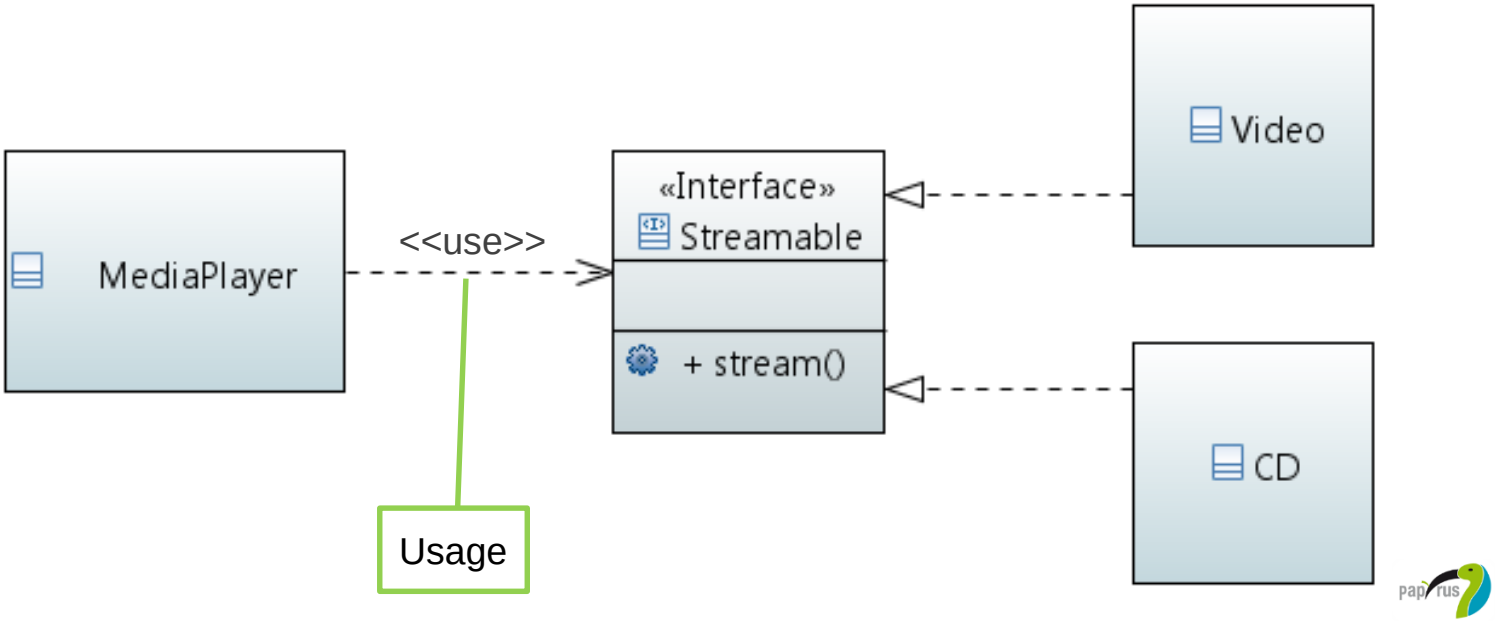
Class implementing an interface

- A class implements an interface, i.e. it implements all of the operations of the interface
- An association between an interface I and any other class C implies that a conforming association must exist between any implementation of I and C
- Modeled with an interface realization relationship: an interface realization between a class and an interface means the class conforms to the contract of the interface
- Example: “Video” and “CD” must implement the “stream” operation of “Streamable”



Class requiring an interface

- A class C may, require for its implementation, an interface I
- Any other class implementing I may be used by C
- Modeled with an usage relationship: an usage between A and B means A requires B for its full implementation or specification
- Example: “MediaPlayer” uses “Streamables” to be able to play media



Polymorphism

- **Ability for some object A to be substituted by some object B, if A and B share some common interface even if they have different underlying data**
- **E.g. Polymorphism can be achieved through inheritance and abstraction, i.e. an instance of class A can substituted by an instance of class B if:**
  - Class B is a child of class A
  - Class B and A implement the same interfaces

CONTENT

What are the specifications of an attribute?

Visibility, name, type, multiplicity, default value

What are the specifications of an operation?

Visibility, name, parameters (themselves with a specification)

What is a static attribute/operation?

A feature that can be accessed from a class, without instantiating the class as an object

What is a “method” in UML?

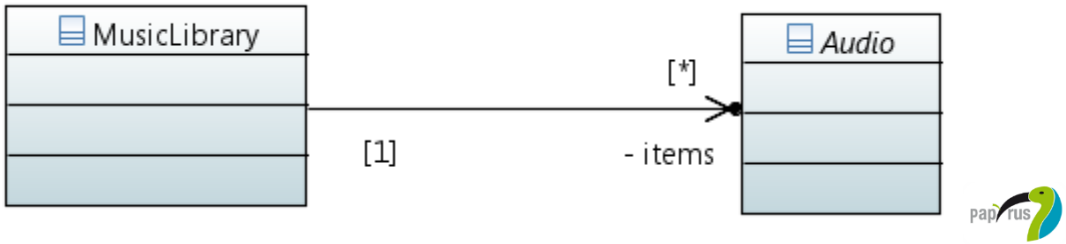
The implementation of an operation (which is the signature)

What are the attributes of a primitive type

None, a primitive type is an atomic data type without structure

How can the following model be represented otherwise?

With an association, where the end connecting Audio is owned by MusicLibrary, named items (role of Audio relative to MusicLibrary), with a multiplcity of [0..*].



QUIZ

What is the difference between an aggregation and a composition?

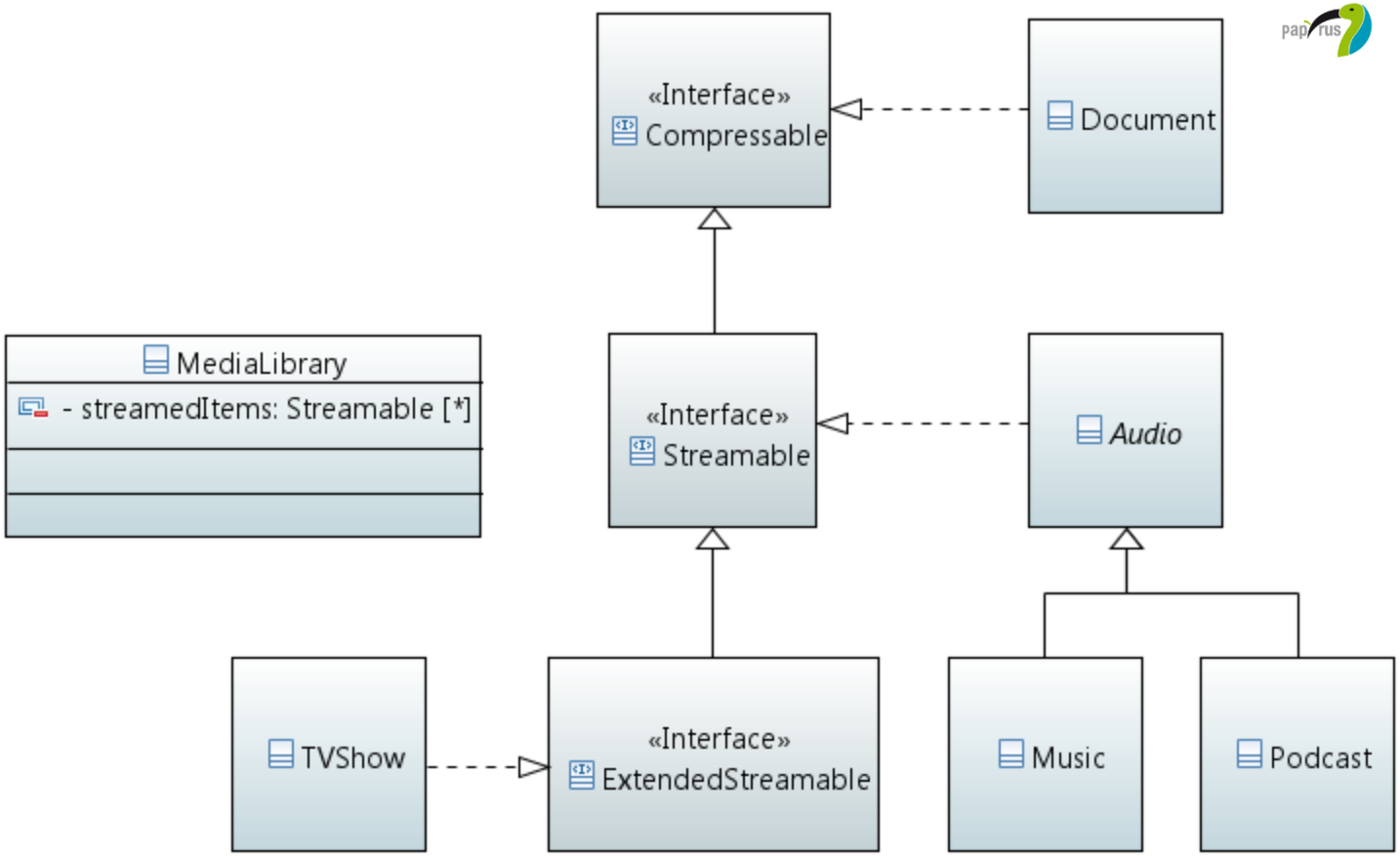
Both are whole/part relationships but the composition is a stronger aggregation. In a composition, if the composite is destroyed, all of its parts are destroyed.

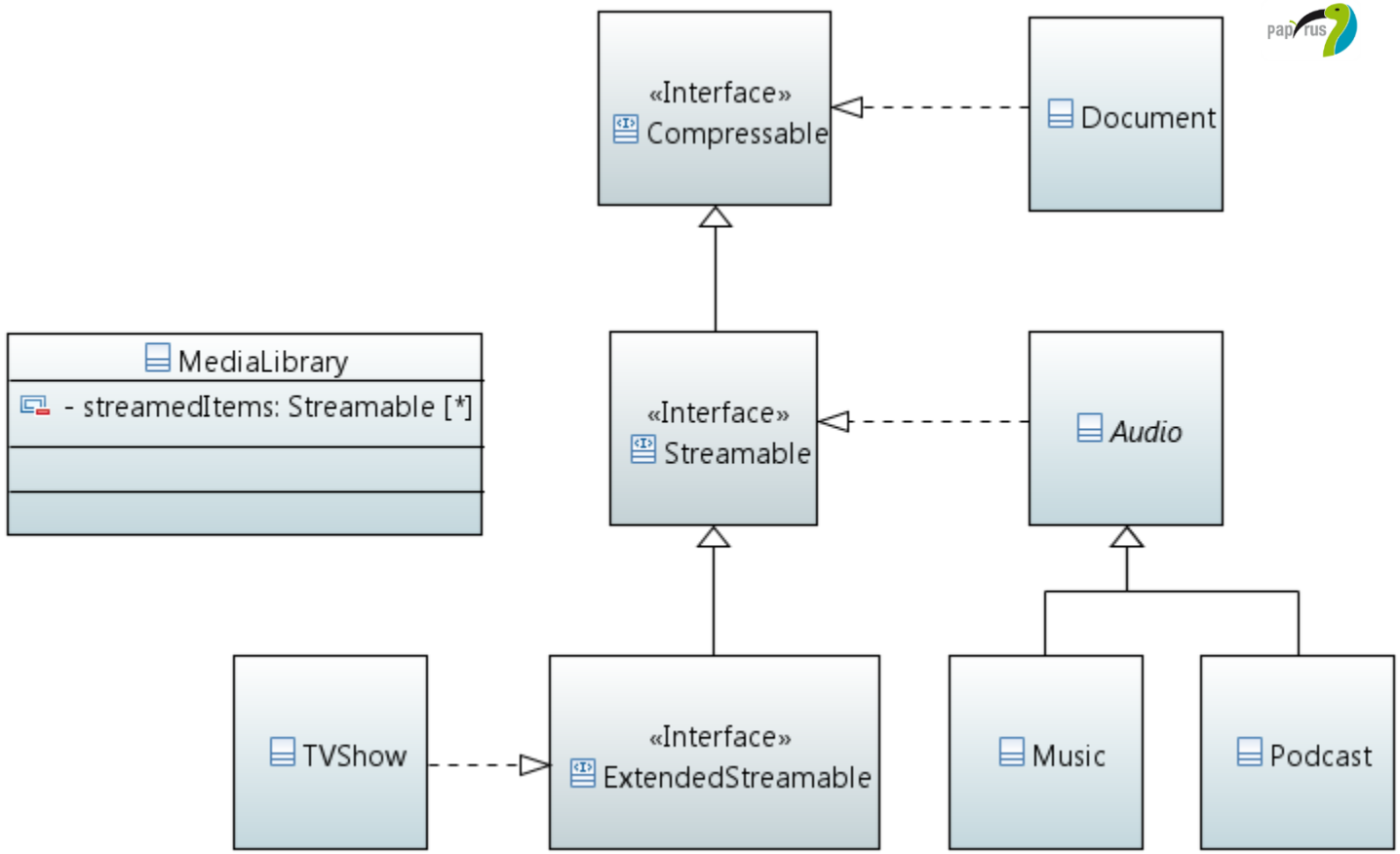
QUIZ

What are the access modifiers for a feature? And for a class?

Public, protected, package, private for a feature. Public, private for a class.

Which classes can be instantiated within the list of *streamedItems* owned by **MediaLibrary**?





!

Only classes realizing **Streamable**, or its child, can be instantiated in *streamedItems*. **Music** and **Podcast** realize **Streamable** since they inherit **Audio**. (Note that **Audio** cannot be instantiated since it is abstract.) **TVShow** realizes **ExtendedStreamable** a child of **Streamable**.

CONTENT

SUMMARY

- Modeling the structure of the system describes who will fulfill the functionalities, the relationships between entities of the system
- A class describes a set of objects with common semantics, features, and constraints, including attributes and operations
- An association is a semantic relationship between classes where multiplicity (how many objects involved), role (of objects), and navigability (who can access who) can be specified
- An aggregation/composition are associations that represent whole/part kind of relationships; a composition is stronger: when the composite is destroyed, all of its parts are destroyed
- Inheritance is modeled in UML with a generalization relationship; specialized classes inherit features of general classes
- Encapsulation is modeled in UML by defining visibility of elements and features: public, protected, package or private; the package element is used to group together other elements and specify a namespace
- Abstraction is modeled in UML by abstract class, interface, and dependency/realization relationships