Class Diagrams Advanced

Ferd van Odenhoven

Fontys Hogeschool voor Techniek en Logistiek

May 28, 2015

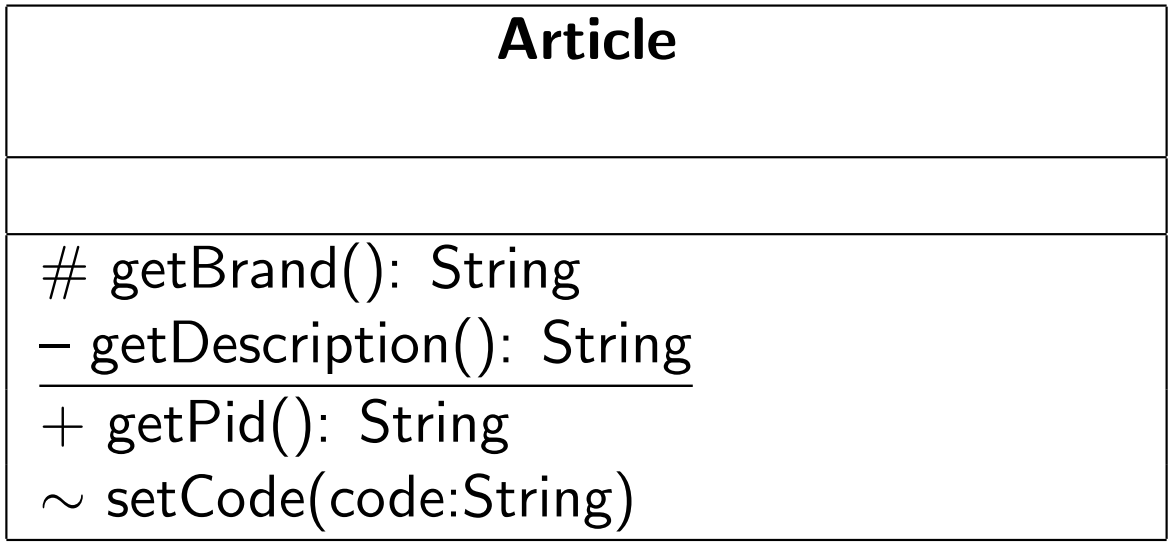
Attributes

Article
– brand: String ~ pid: String + code: String # description: String

computerMouse:Article
brand = "Logitech" pid = "LZ5280C0AN1" code = "RX300" description = "wired wheelmouse"

- *Access in this class only:* **this.brand**
- *Access in this package only:* **computerMouse.pid**
- *Access for everybody:* **computerMouse.code**
- *Static access for everybody:* **Article.code**
- *Access in this package and in derived class:* **computerMouse.description**

Methods



- `protected String getBrand() {return brand;}`
- `private static String getDescription() {return description;}`
- `public String getPid() {return pid;}`
- `void setCode(String code) {this.code = code;}`

Attributes and Methods

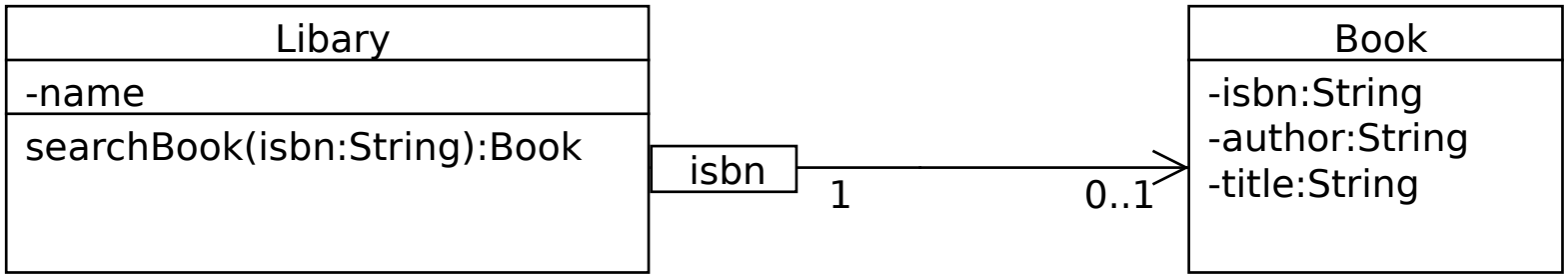
- **Attributes:**

  ```
  visibility name:  type multiplicity
  = default-value {property-string}
  ```

- **Methods:**

  ```
  visibility name(parameter-list):
  return-type {property-string}
  ```

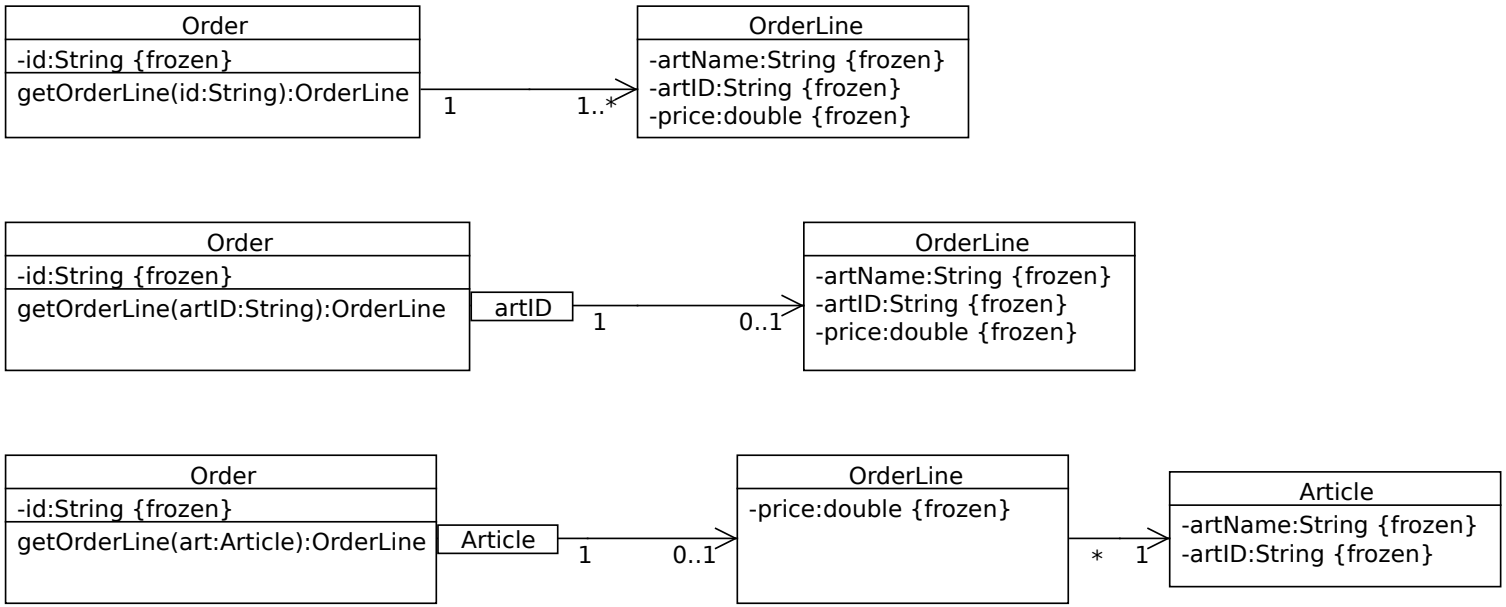
Implementation detail: a qualified association



A qualified association indicates: an ISBN-number is referring to a book. This number can thus be used to find a specific book.
The following code could be an implementation:

```
public class Library {
    private HashMap<String,Book> books;

    Book searchBook(String isbn) {
        return books.get(isbn);
    }
}
```

Qualified Association:another example

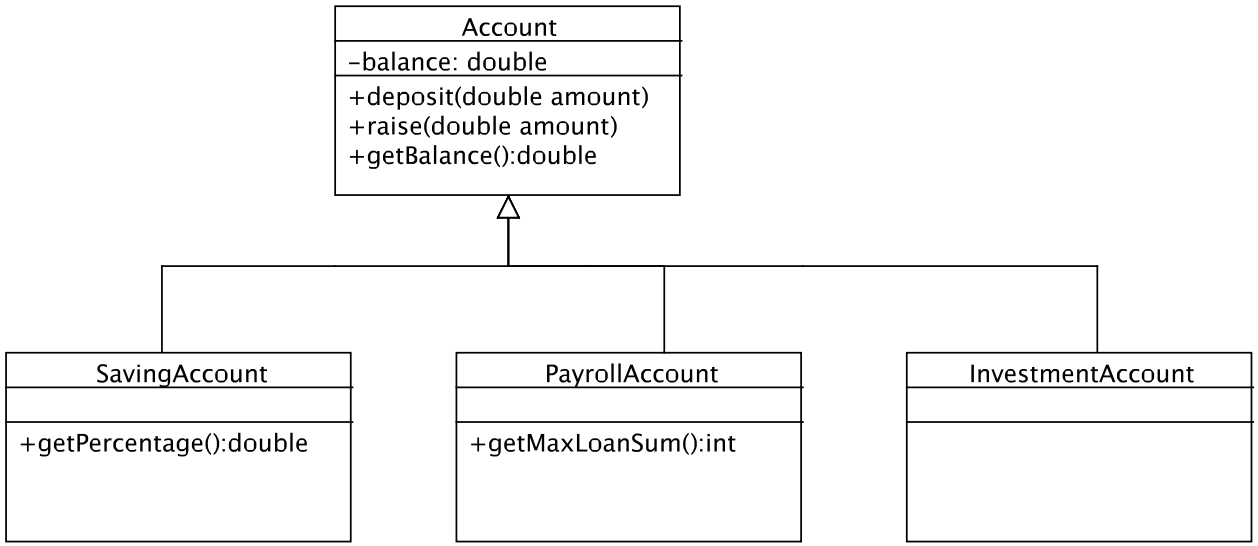


We could find an order line by the id of the article or by an Article-object

Generalisation

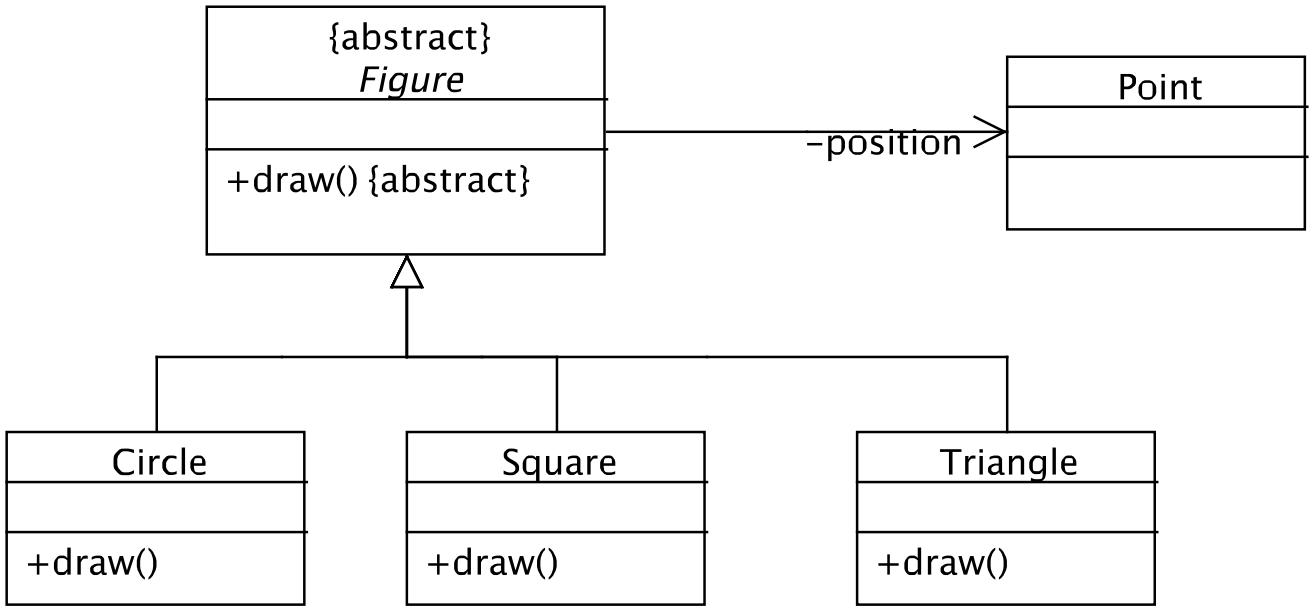
Classification of classes with identical properties

- Superclass: set of classes
- Subclass: class in a set
- Inheritance: subclass inherits the properties of a superclass



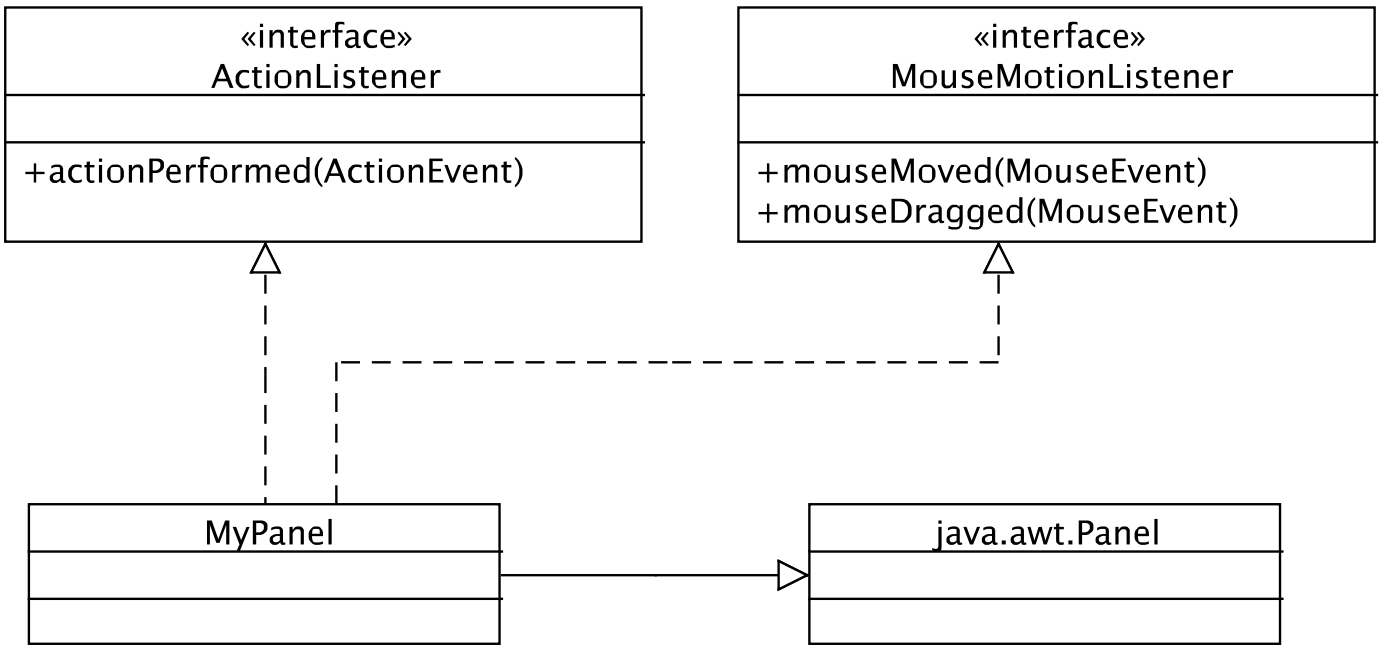
Abstract classes and operations

These are classes of which no objects can be made. Abstract classes therefore are always superclasses. Operations of abstract classes can be abstract as well. In that case the implementation has to implemented in a sub class.



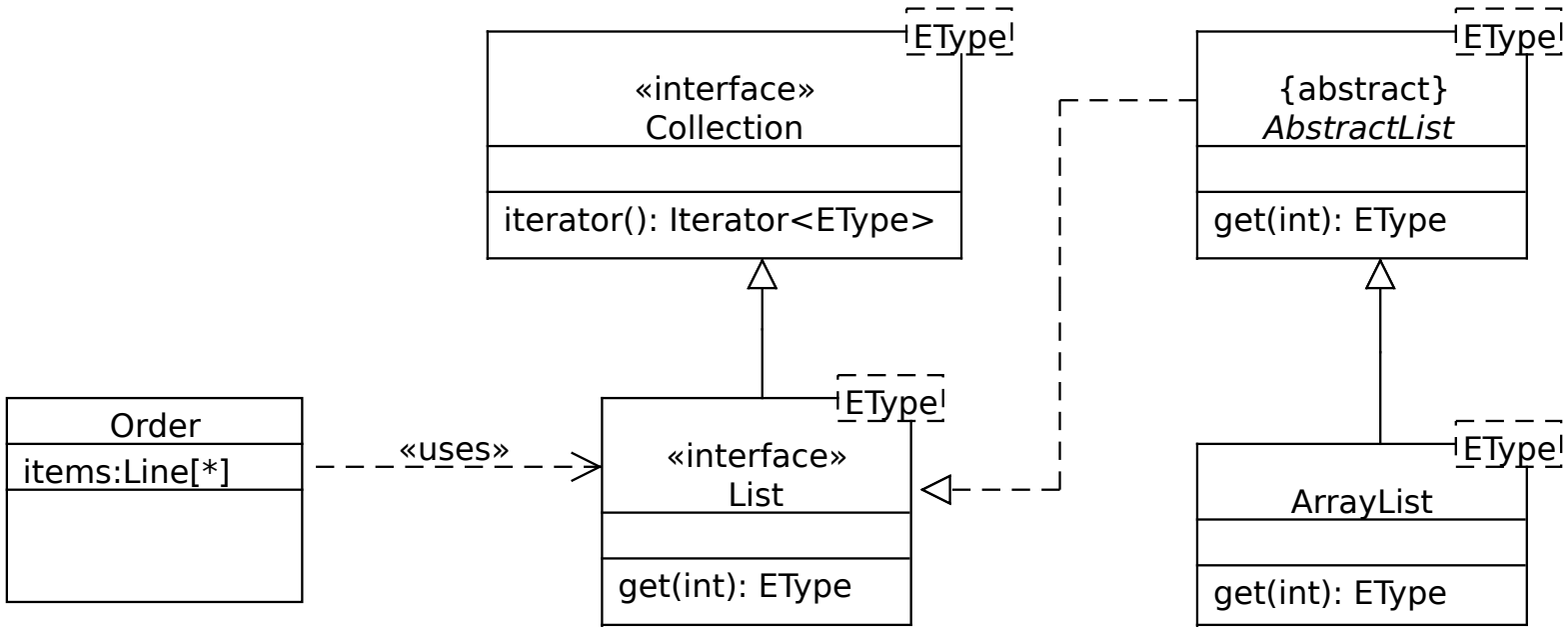
Interface

Model element describing how instances of a class that implements this interface, can be approached by other classes.



Interface example with dependancy

(Figures: chapter 5, UML Distilled)



Somewhere inside the Order class possibly:

```
List<Line> list = new ArrayList<Line>();
```

Interface: more compact notation

(Figures: chapter 5, UML Distilled)

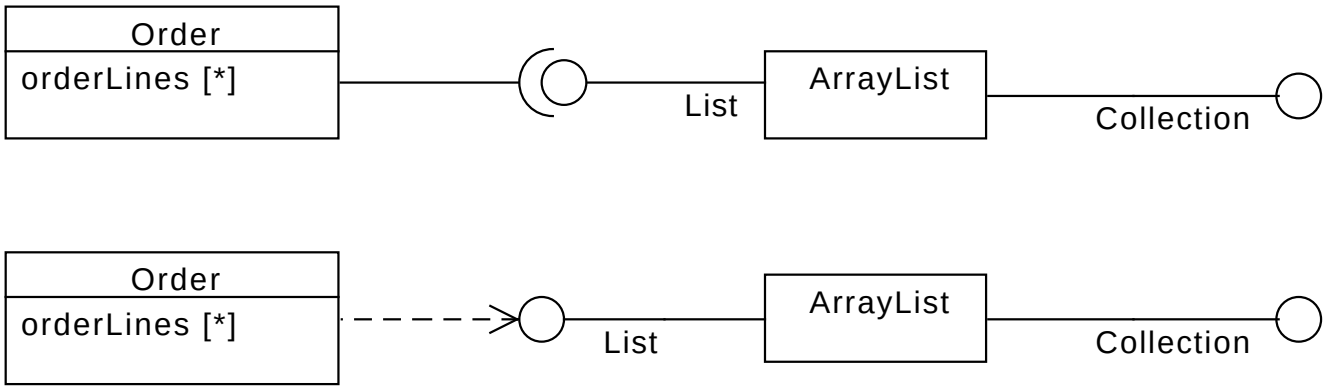
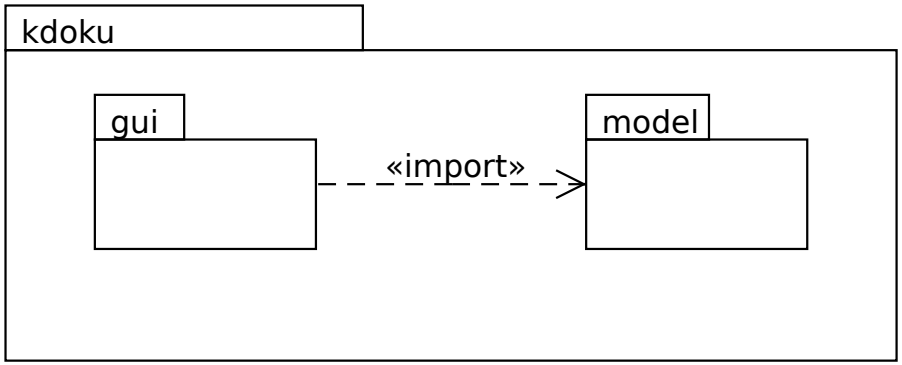
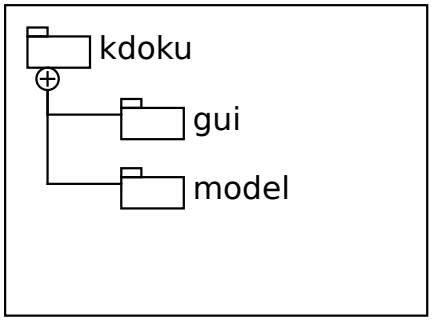


Figure : The upper diagram with the 'Ball-and-Socket'-notation is conform UML2; the lower with the 'lollipop'-notation is UML1.

Packages

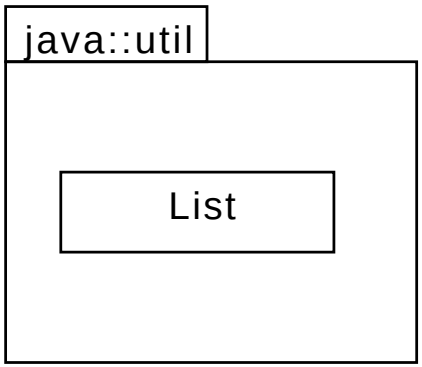


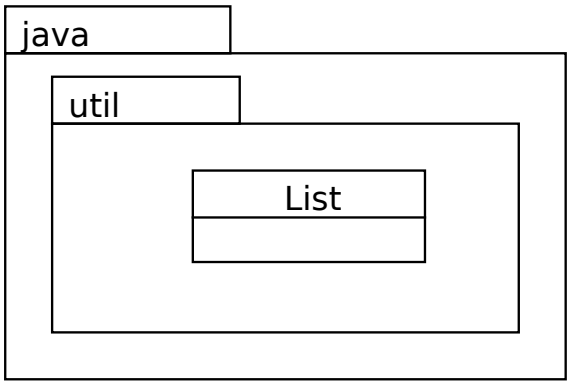


Packages are often found in other packages and may have dependency relations.

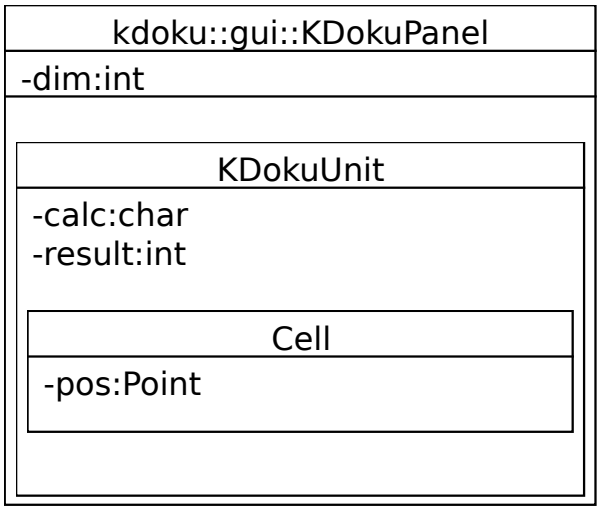
Package diagrams: nested packages

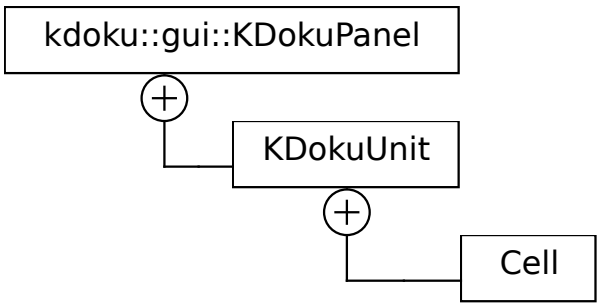
- **Fully qualified name**:
- Java examples
  - `library.Library`
  - `java.util.List`
- UML:
  - `library::Library`
  - `java::util::List`





Inner classes

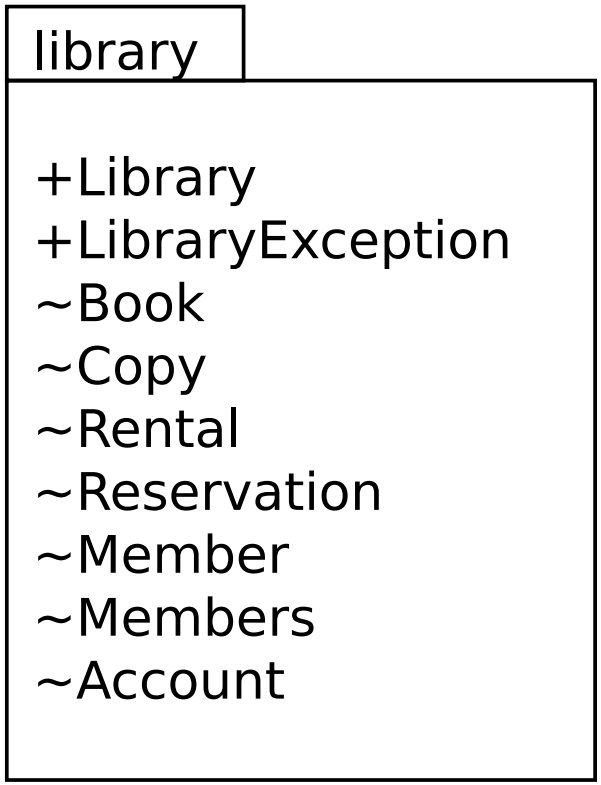





```
package kdoku.gui;

public class KDokuPanel {
    private int dim;

    private class KDokuUnit {
        private int result;
        private char calc;


        private class Cell {
            private Point pos;
        }
    }
}
```

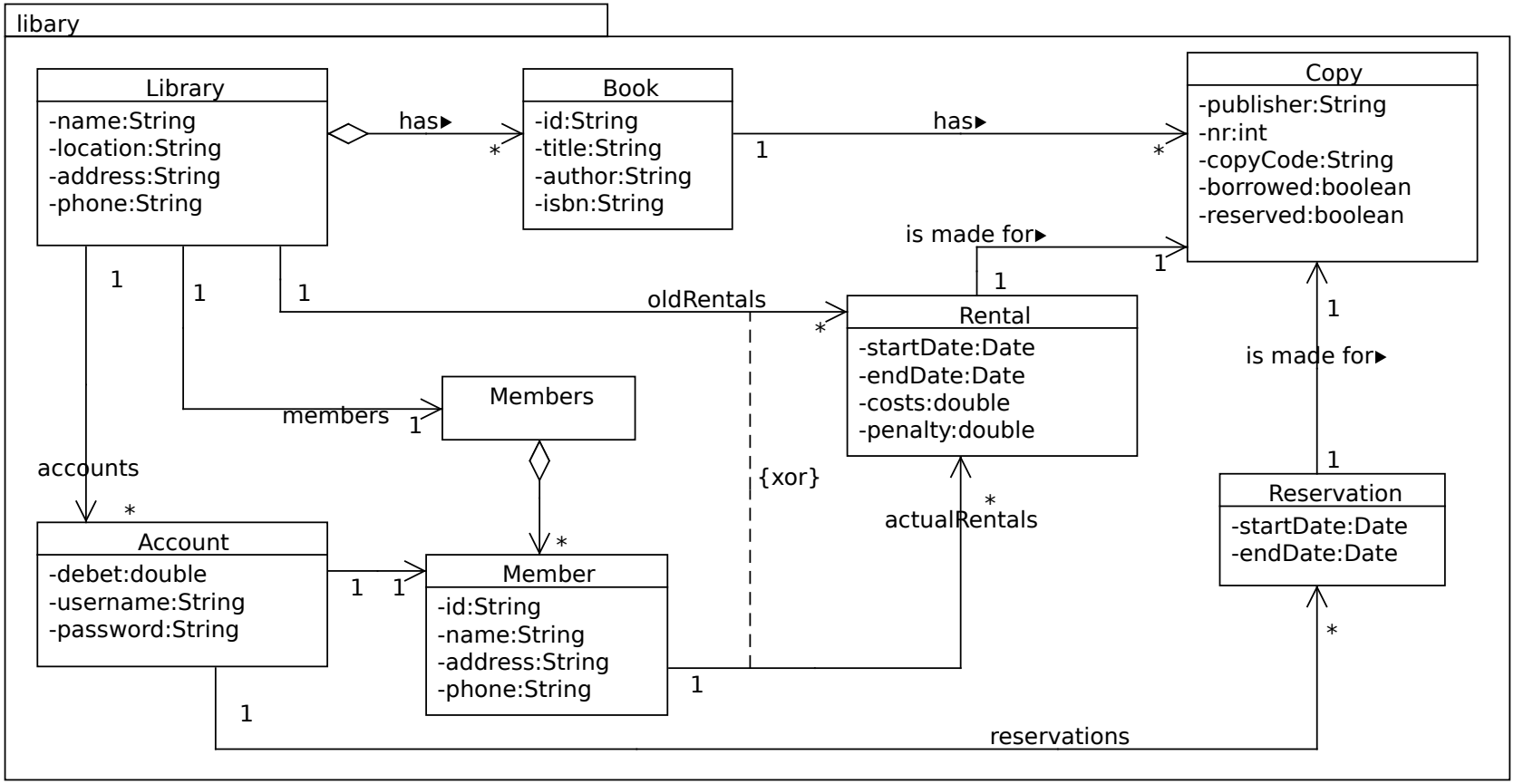
Class visibility in Package diagram

- A package diagram can contain the class names and show their visibility.



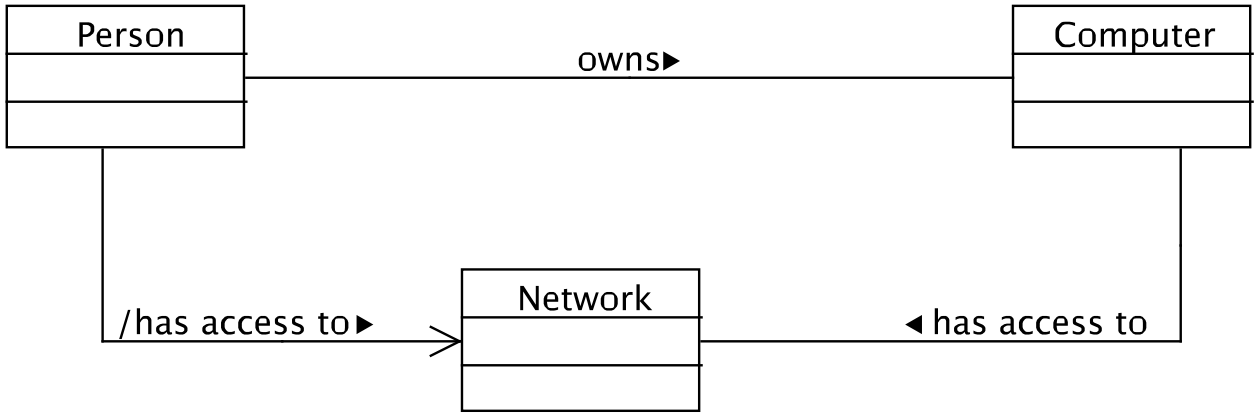
Package diagram includes class diagram

- A package diagram can also contain the classdiagram.



Derived Associations

- An associaton that can be derived from other associatons **don't use them**



Derived Attributes

- Derived properties or attributes