



Object-oriented Design, Refactoring & Testing Lecture 1

Welcome to Object-Oriented World

Bartosz Walter

<Bartek.Walter@man.poznan.pl>

Important data

Lectures

Thursdays, 11.45 AM, Polish-German Academic Centre

Duty hours: Wednesdays, 1:30-3.00 PM, room 2@PGAC

Laboratory classes

Thursdays, 8.00 PM, 9.45 PM

Teacher: Andrzej Swędrzyński
kokosz@man.poznan.pl

Method of verification

Examination 😊

Schedule outline

- | | |
|--------------------------------------|-----------|
| 1. Object-oriented world | 1 week |
| 2. Introduction to Java | 1 week |
| 3. Good programming practices | 1 week |
| 4. Unit-testing | 3-4 weeks |
| 5. Refactoring | 3-4 weeks |
| 6. Design patterns | 3-4 weeks |
| 7. Automated testing | 1 week |

What object-orientation means?

Abstraction

Polymorphism

Encapsulation

Flexibility

Inheritance

Modularization

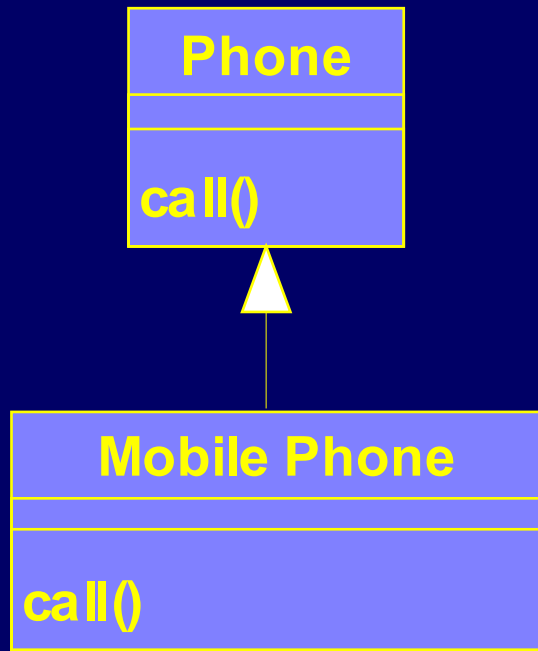
Reuse

Abstraction

Alan Kay, author of Smalltalk:

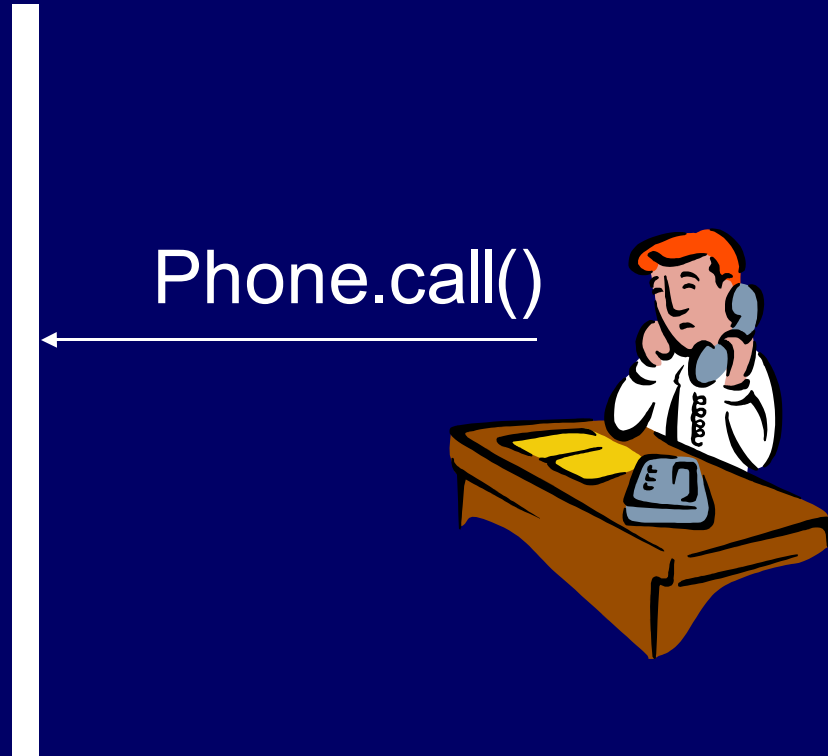
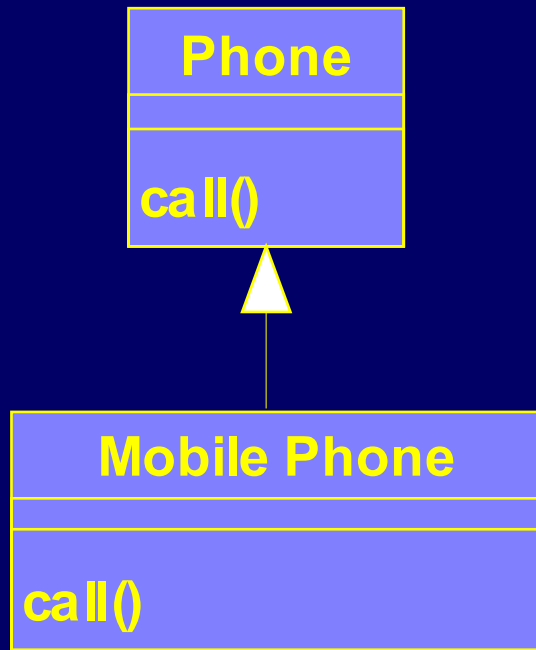
1. Everything is an object
2. Program is a set of objects, which interact by sending messages to each other
3. Every object is composed of other objects
4. Object has its type
5. All objects of given type can accept the same messages

Polymorphism

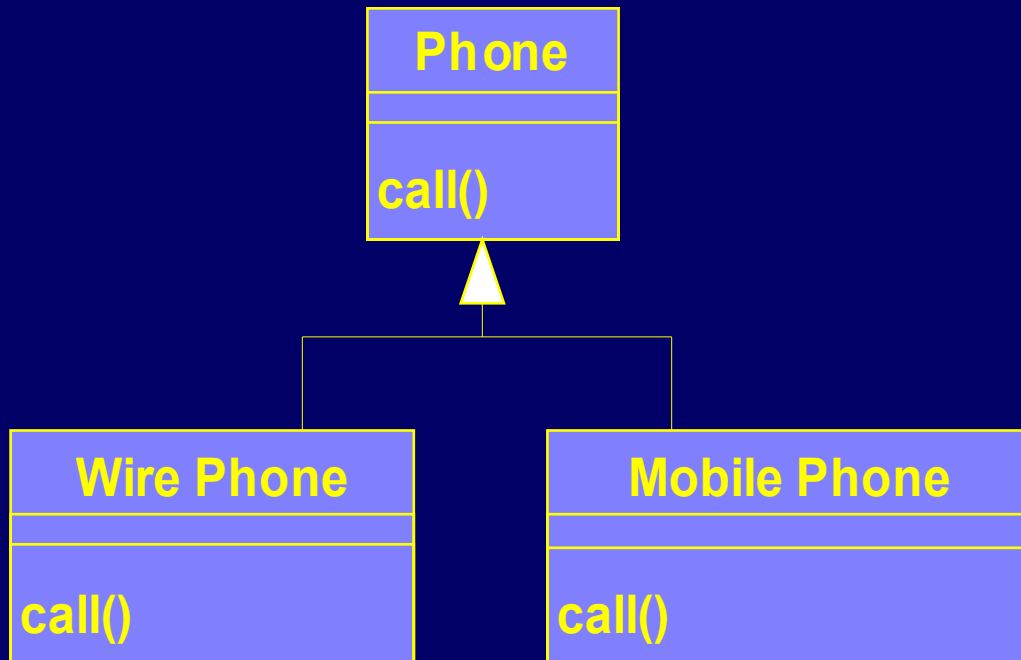


Polymorphism: ability to accept the message by an appropriate class known at runtime

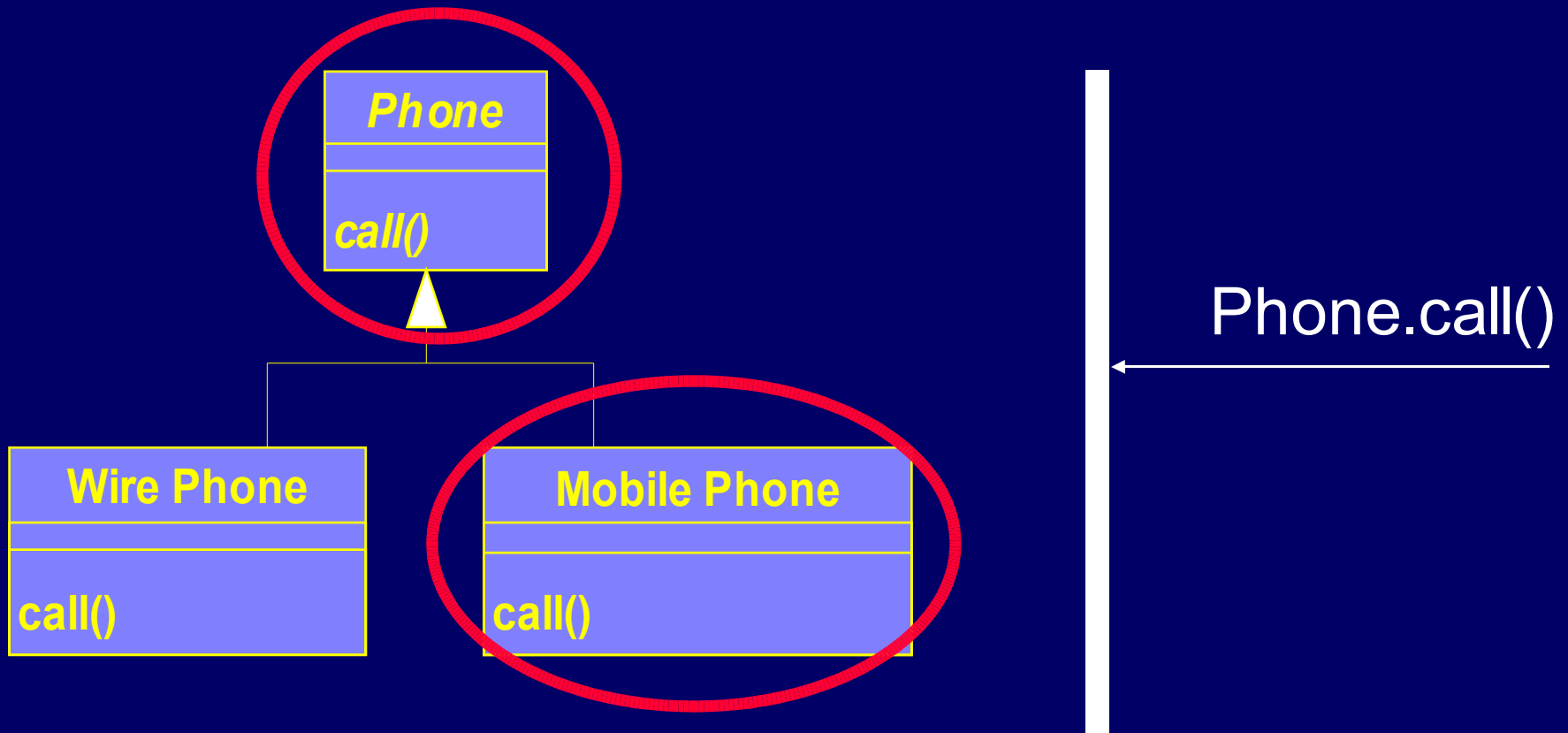
Interfaces



Interfaces



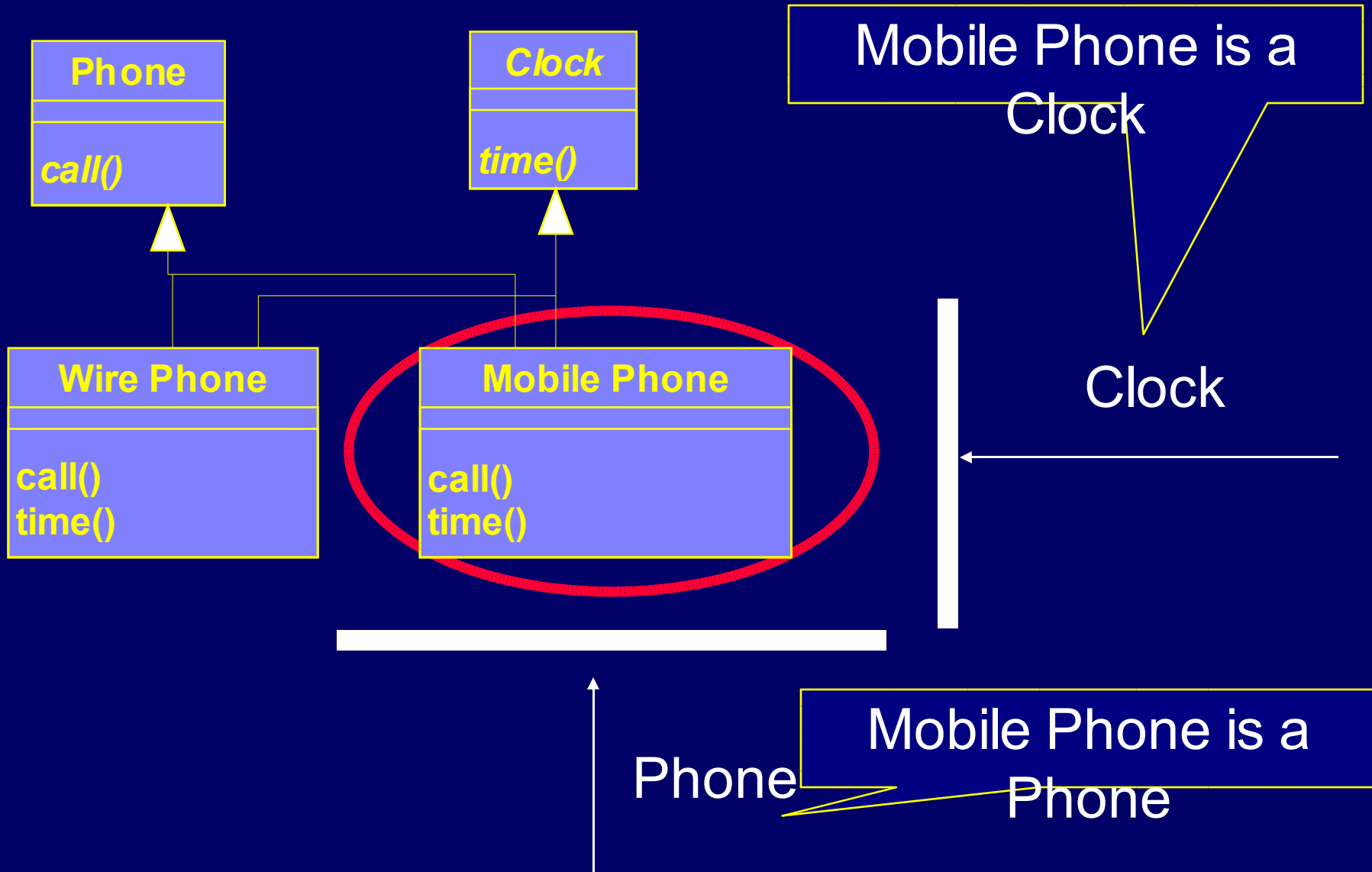
Interfaces



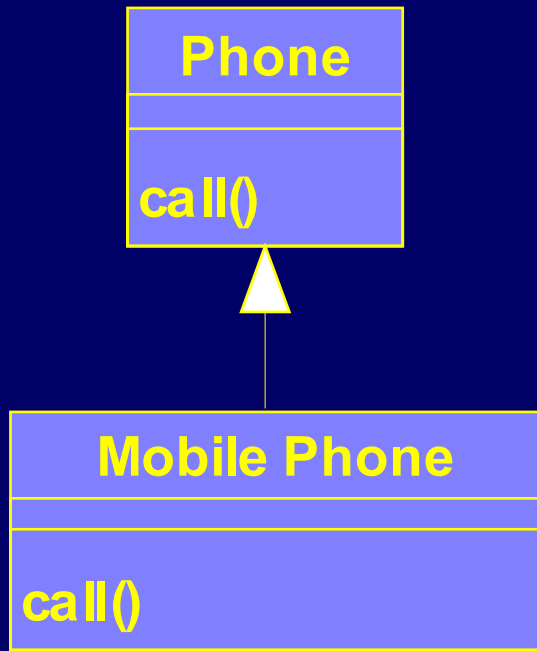
Interface: visible part of the class

Program to interfaces, not implementation.

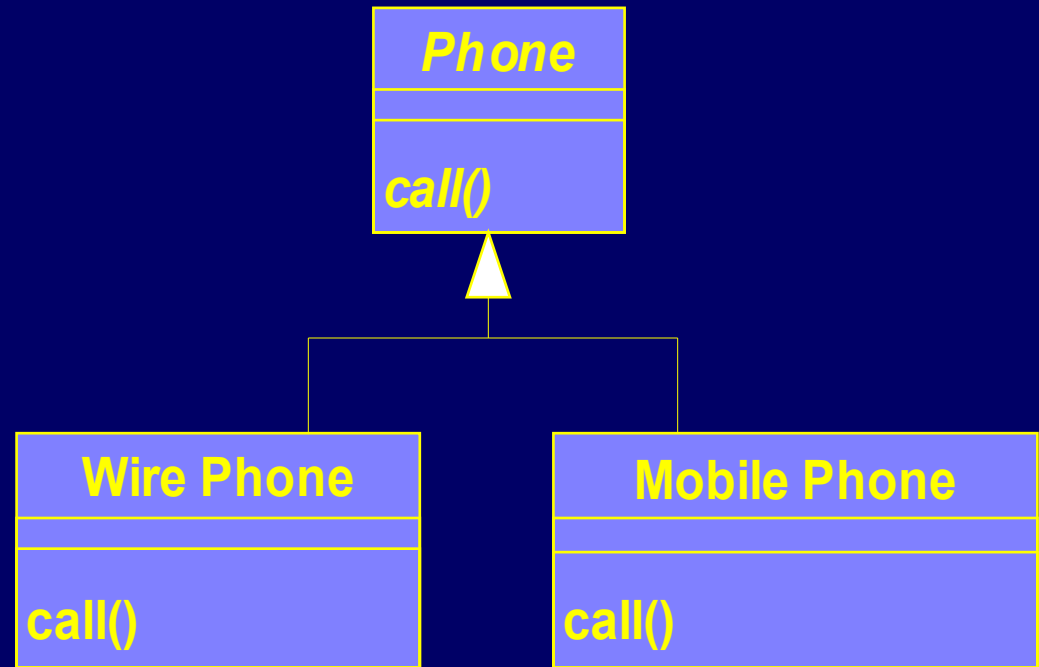
Multiple interfaces



Interfaces vs. inheritance



Mobile Phone is a kind of **Phone**.
It can call like a

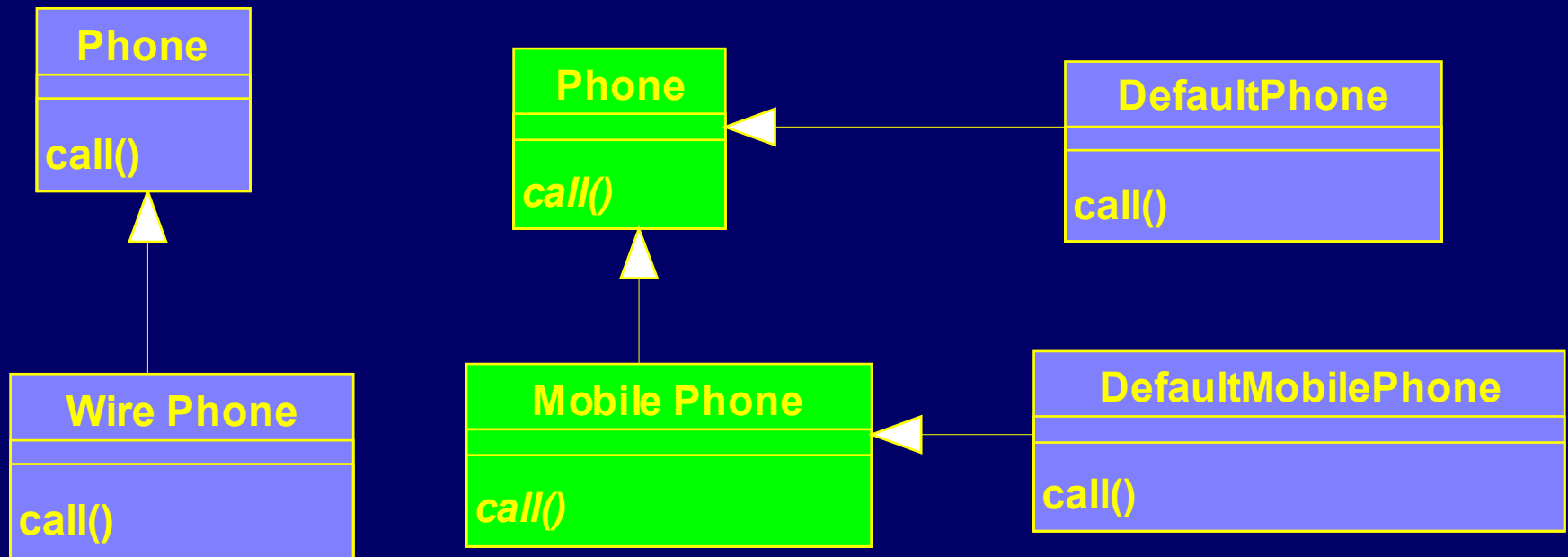


Wire Phone and **Mobile Phone** are of type **Phone**.

Class inheritance vs. interface inheritance

Class inheritance: inherits (default) implementation

Interface inheritance: inherits method signatures



Encapsulation



hair color

height

name

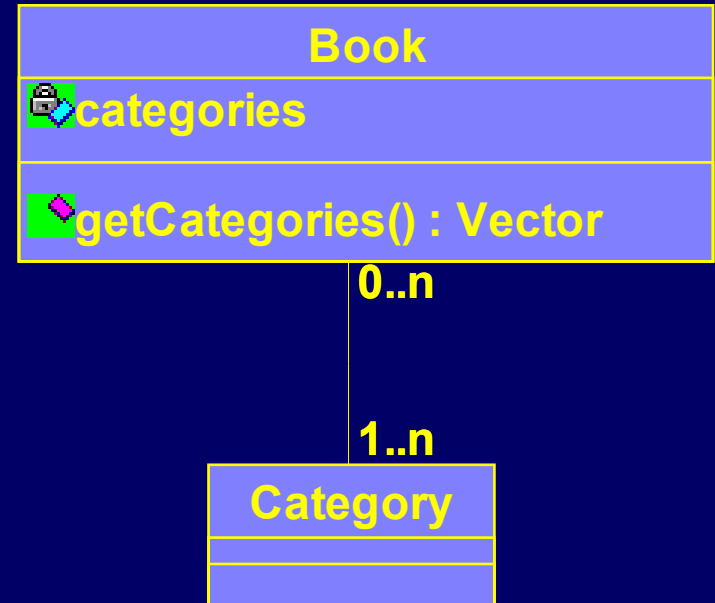
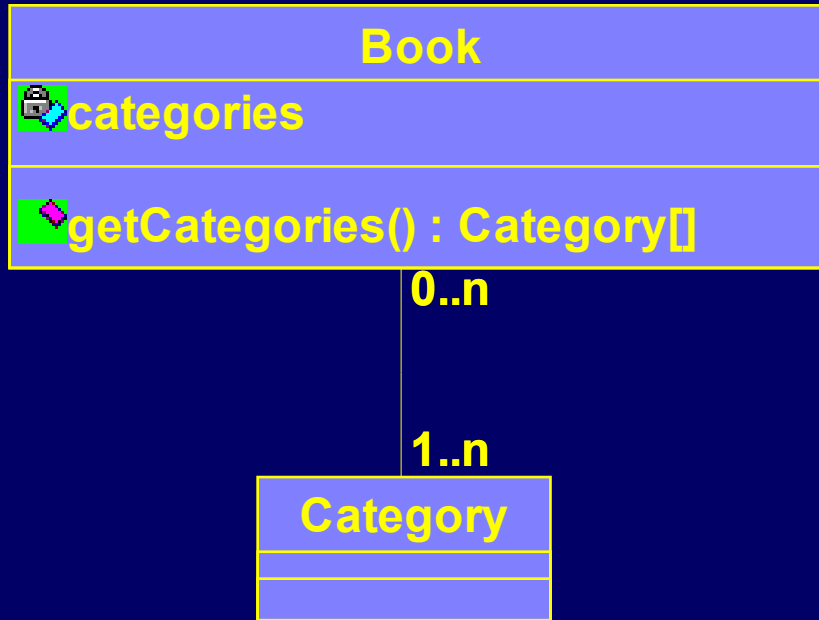
```
...  
h = new Human();  
h.setName();  
h.getHeight();  
...
```

Encapsulation



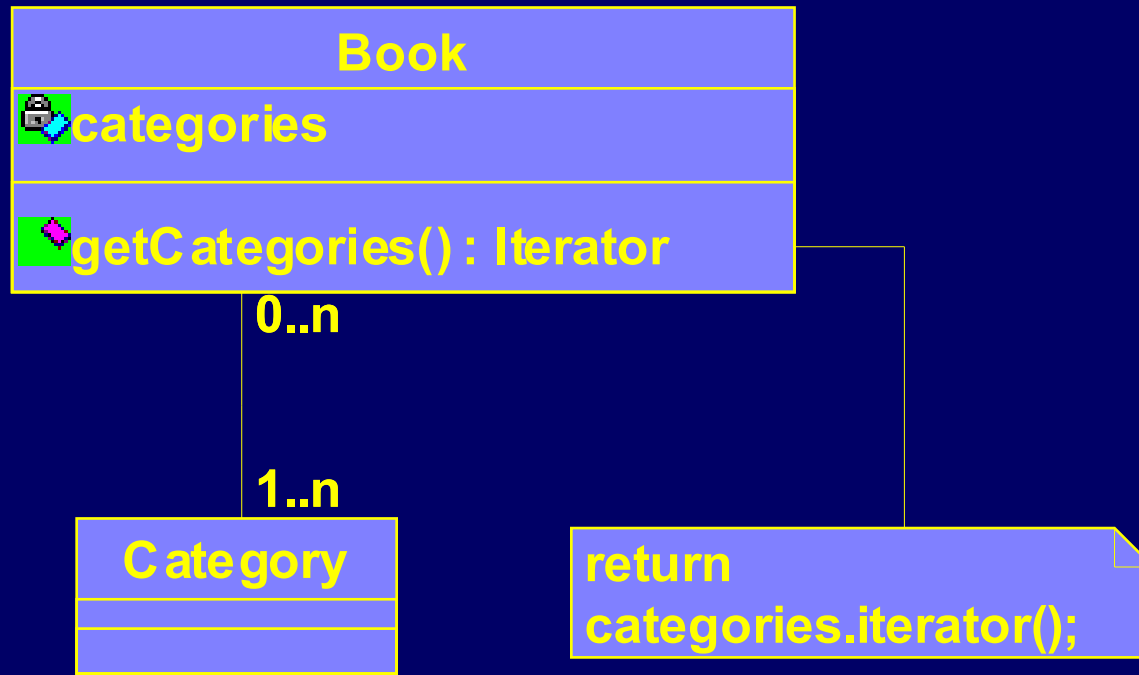
Do not create invalid objects

Encapsulation



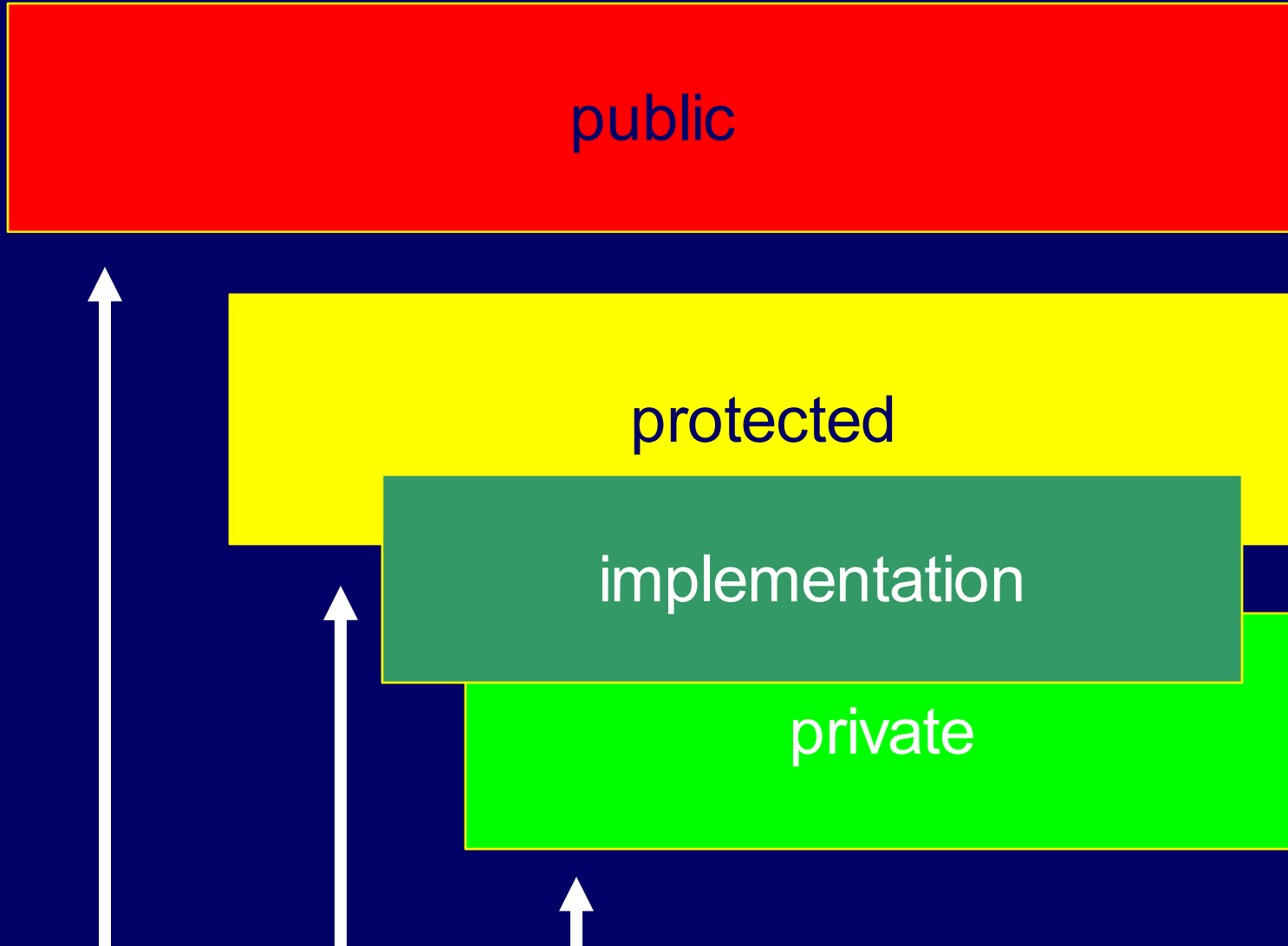
How about safety?

Encapsulation

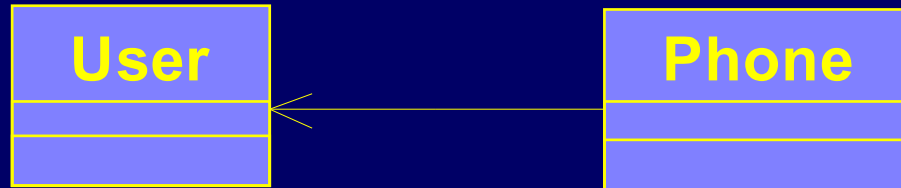


Provide safe access to objects.

Access levels

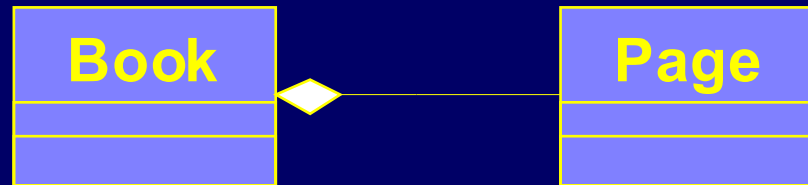


Different types of relations: association



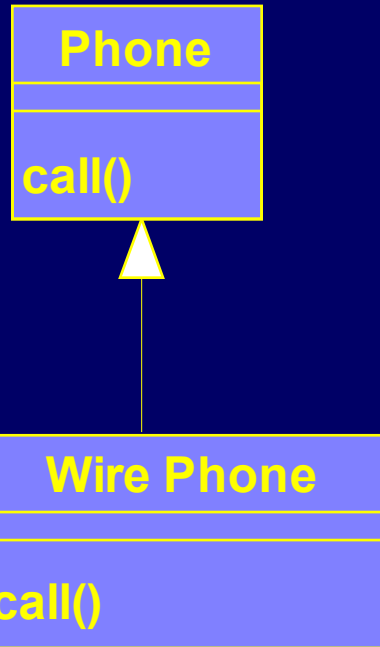
- ✓ Phone **belongs** to the User.
- ✓ Phone **can change** the User
- ✓ User **does not need to know** the Phone.
- ✓ User **can possess many** Phones.
- ✓ Phone and User **can exist independently**.

Different types of relations: composition



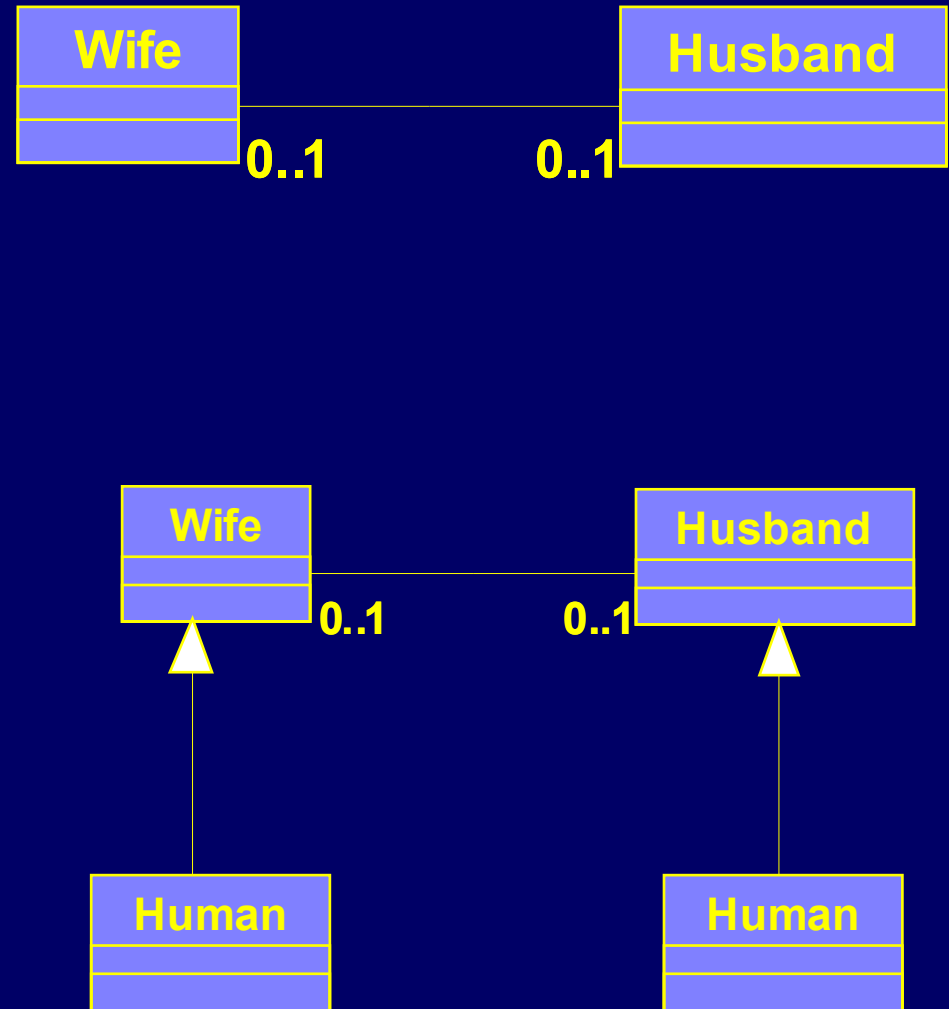
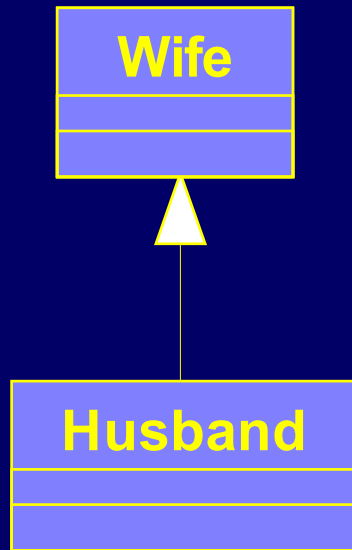
- ✓ Book **is composed** of pages.
- ✓ Page **is a part** of book.
- ✓ Page **cannot exist** apart from book.
- ✓ Book **manages** pages (adds, removes etc.)

Different types of relations: inheritance



- ✓ Wire Phone **is a kind of** Phone.
- ✓ Wire Phone **can access** Phone.
- ✓ Phone **cannot be used** instead of Wire
- ✓ Wire Phone **can be used** instead of Ph
- ✓ Wire Phone **must inherit** all features of
- ✓ Phone **is a indistractable part** of Wire P

Inheritance vs. composition



Inheritance vs. composition: pros & cons

Inheritance

Fixed at compile-time

More error-resistant

Heavyweight,

Exhibits internals

Indistractable

Composition

Evaluated at runtime

More error-prone

Lightweight

Flexible

Prefer composition over inheritance

Value objects

Identified by value

There may exist multiple equal objects

Objects are immutable

Money

Time objects

Physical values

```
If (objectA.equals(objectB)) {  
    // objectA & objectB have common content  
}
```

Reference objects

Identified by reference

There may exist only one instance (or caching is used)

Objects are changeable

People

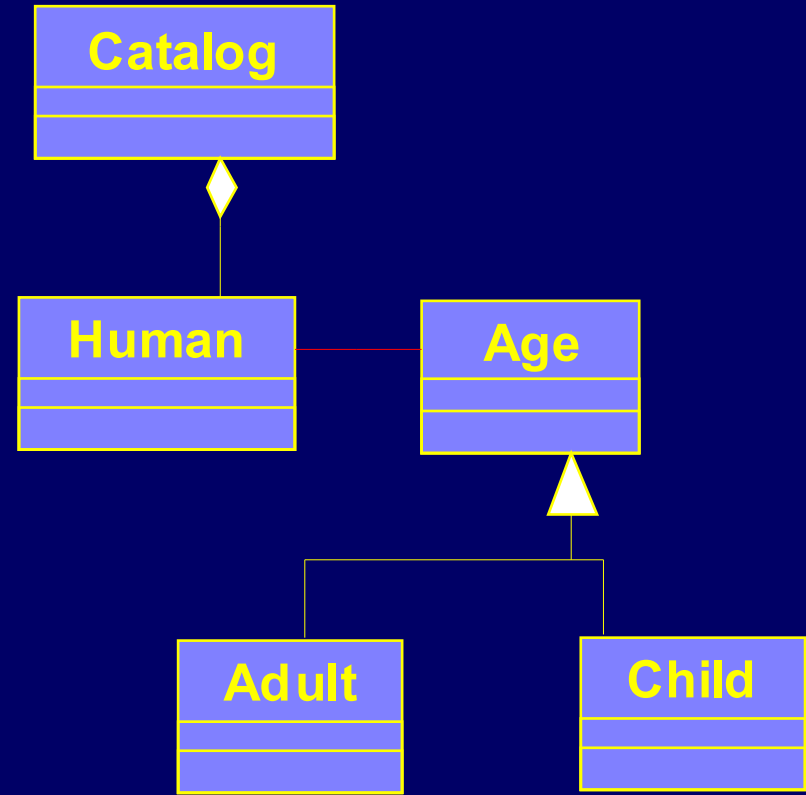
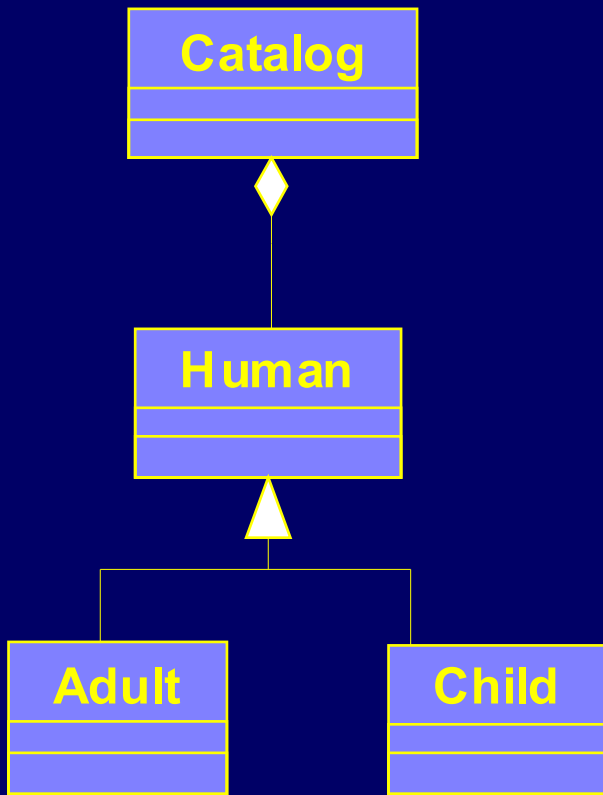
Mobile phones

Books

```
If (objectA == objectB) {  
    // objectA & objectB refer to the same object  
}
```

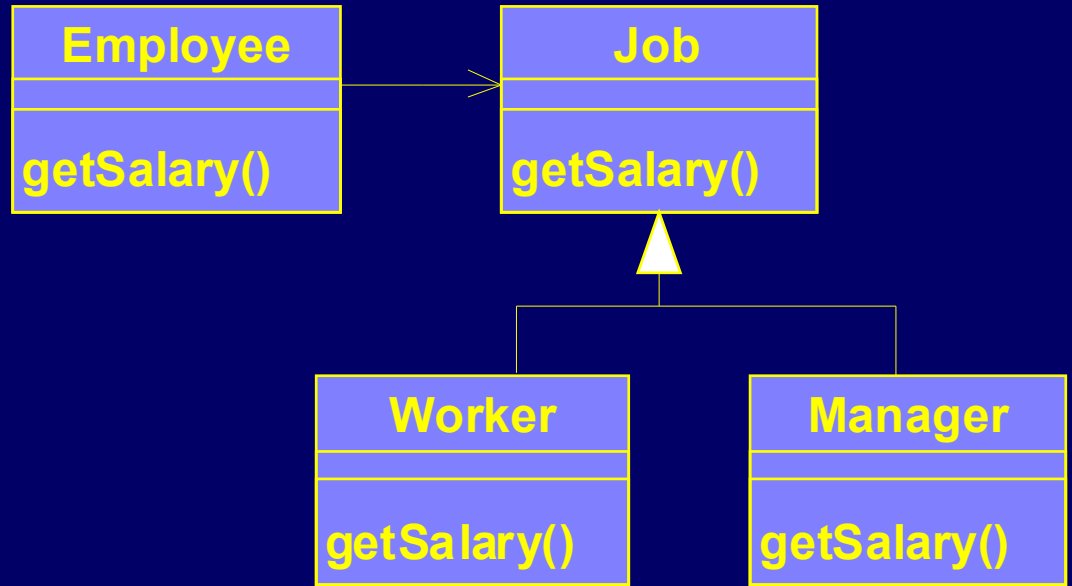

Example

The library catalog contains adults and children.



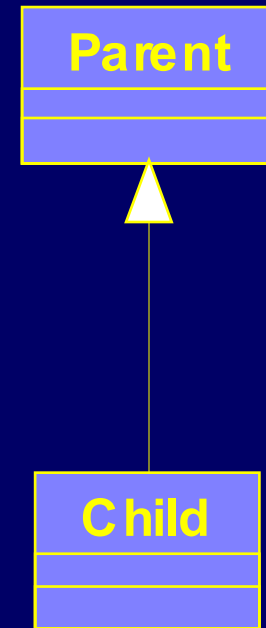
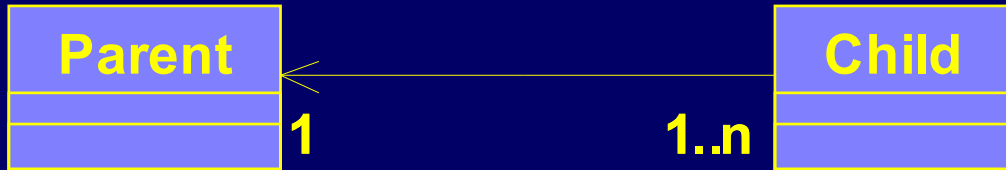
Another example

Some of employees are managers.



Yet another...

Every child knows their parent(s)



Bibliography

1. J. W. Cooper: *Java. Wzorce projektowe.* Helion 2001
2. B. Eckel: *Thinking in Java.* Helion 2001
3. J. Shalloway, J. Trott: *Projektowanie zorientowane obiektowo. Wzorce projektowe.* Helion 2001
4. E. Gamma et al.: *Design patterns. Elements...* Addison-Wesley 1995
5. M. Fowler: *Refactoring. Improving design...* Addison-Wesley 1999
6. J. Langer: *Java style. Patterns for implementation.* Prentice Hall 2000

