Tampereen yliopisto
Tampere University

Abstract Base Classes Inheritance

20.9.2018

Tampereen yliopisto
Tampere University

Abstract base classes

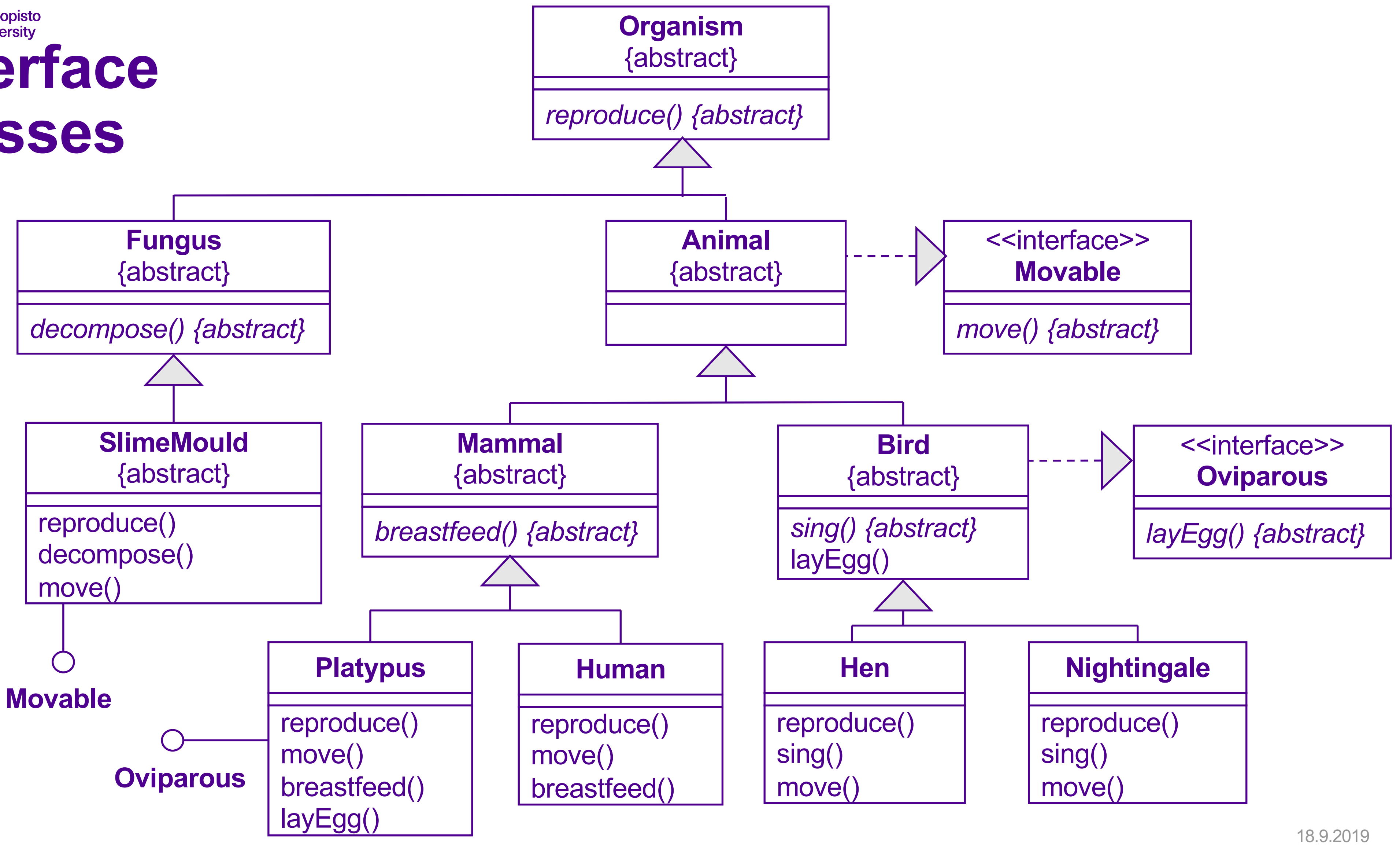
- Include no member variables nor implementations for member functions
  - To be used only as a base class
  - Cannot be instantiated
  - Typically includes interface functions with no (sufficient) implementation

Abstract base classes

- To reveal:
  - pure hierarchic interface
- To hide:
  - the implementation of interface functions is deferred into concrete classes

Tampereen yliopisto
Tampere University

Interface classes



Tampereen yliopisto
Tampere University

Interface class definition

```
class Movable
{
public:
     // Compiler generates the empty default
     //constructor automatically
     virtual ~Movable() { }  // Empty virtual destructor
                             // inside definition (inline)
    virtual void move(Location destination) = 0;
};
```

Inheritance

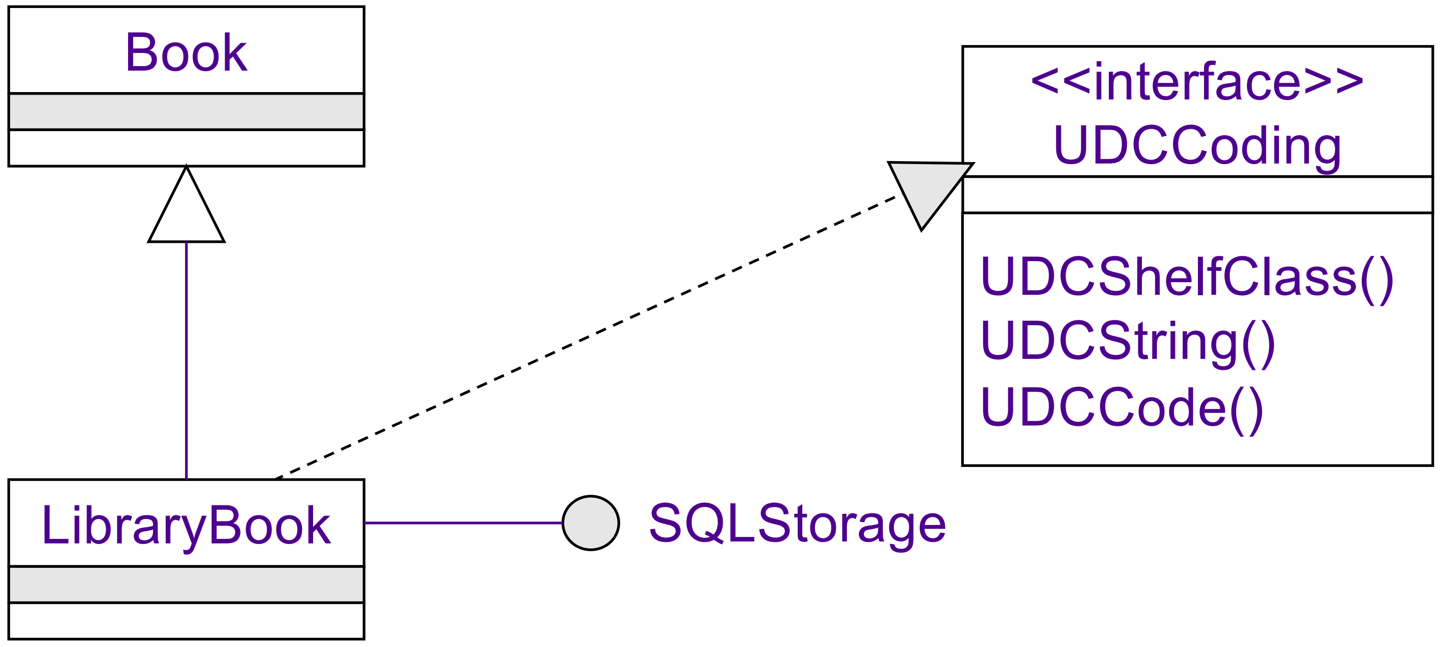
A property in object-oriented programming, where a new class is constructed based on an existing one

- Subclass (inherited class) includes features of the base class (attributes and interface)
- Subclass can add new features and change the features of the base class

“is-a” relationship

- An important property in object-oriented programming and design
- Interface of an inherited class is by default the same as that of the base class ⇒ an object of the inherited class is an instance of the base class (an extended version of it)

Inheritance and interfaces



Tampereen yliopisto
Tampere University

Inheritance terms

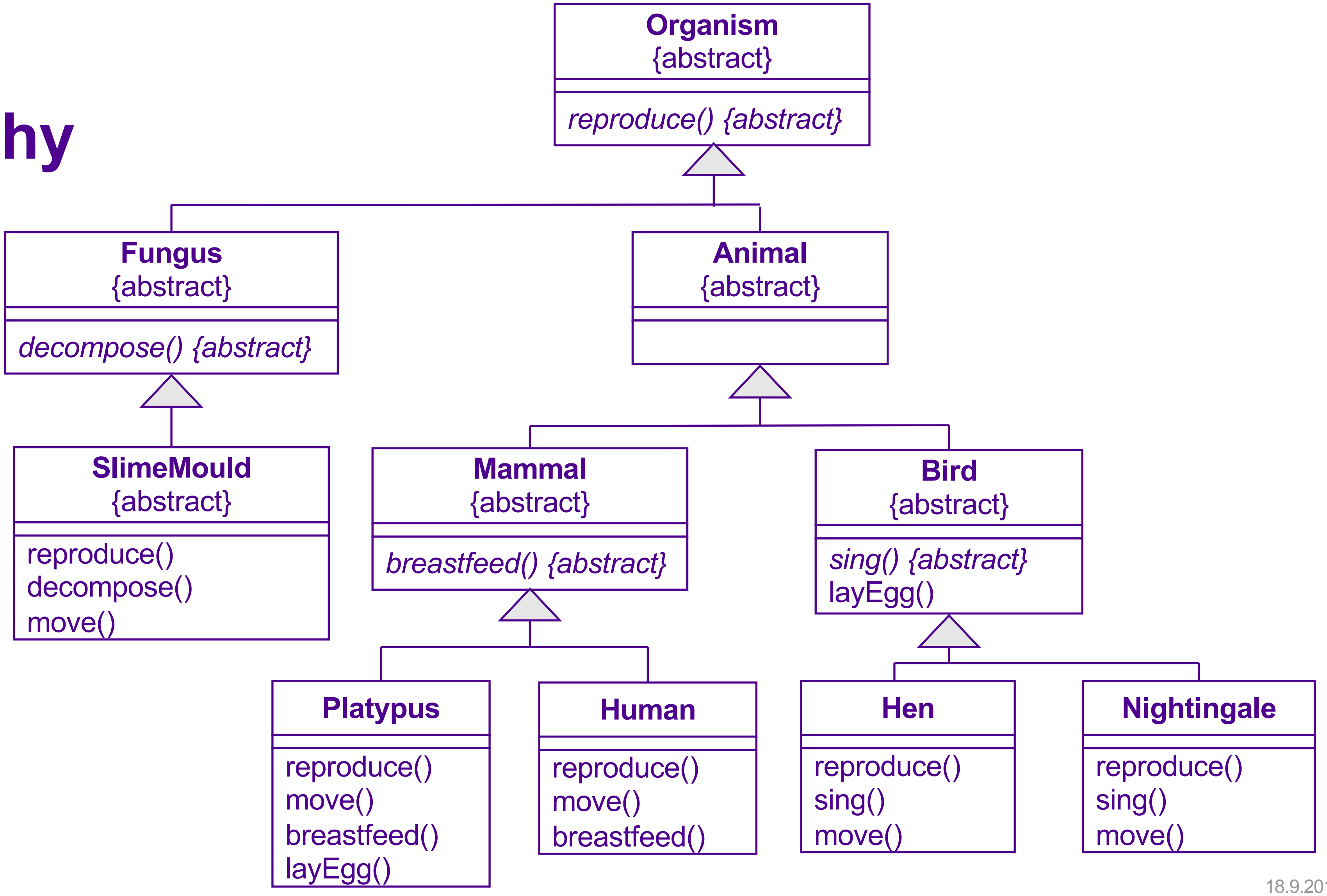
- Inheritance, derivation
- Inheritance hierarchy
- Base class, superclass, parent class
- Subclass, derived class
- Ancestor
- Descendant

Inheritance

- Classification and categorization are natural for humans
- Applied widely in science, languages, etc.
- Object-oriented programming:
  - Classification based on common interfaces
  - Classification based on common implementation
- In many languages, the above two are the same mechanism: *inheritance*

Tampereen yliopisto
Tampere University

Class hierarchy



Class hierarchy

- Different roles of classes:
  - Interface classes
    - only pure virtual methods
  - Abstract base classes
    - at least one pure virtual method
  - Concrete classes
    - no pure virtual methods

Class hierarchies

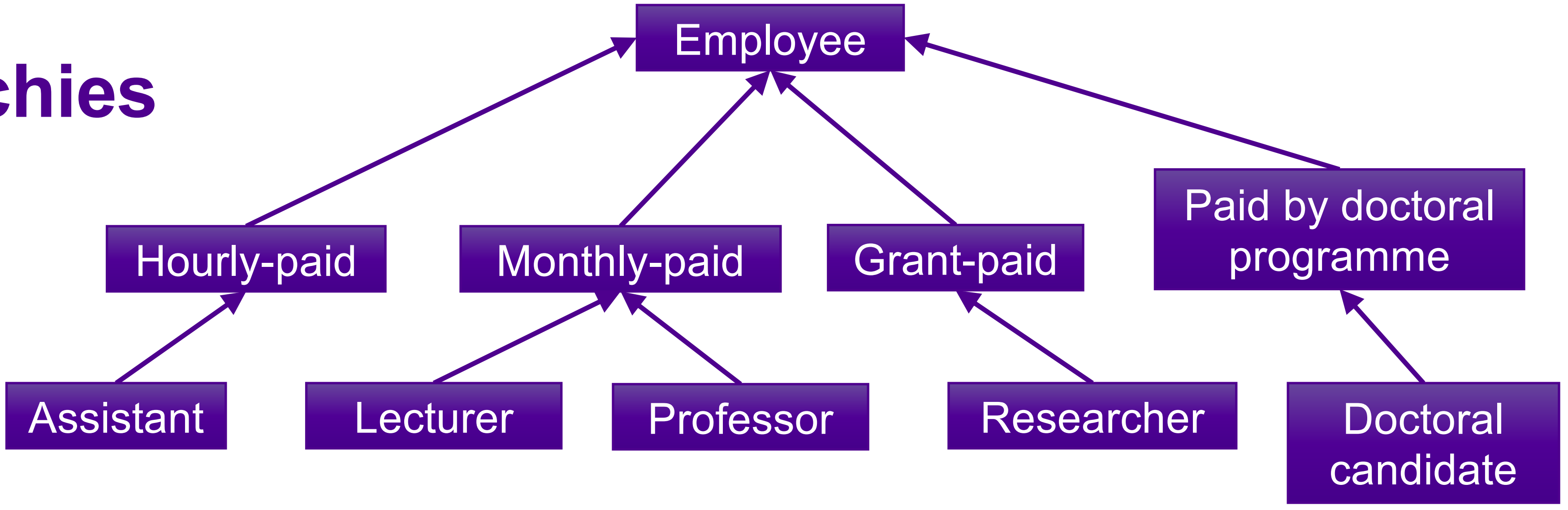
- Upper level classes of a hierarchy
  - “hypernyms”
  - Prescribe the interface of lower level classes
  - Polymorphism: allows referencing to objects with a “common name”

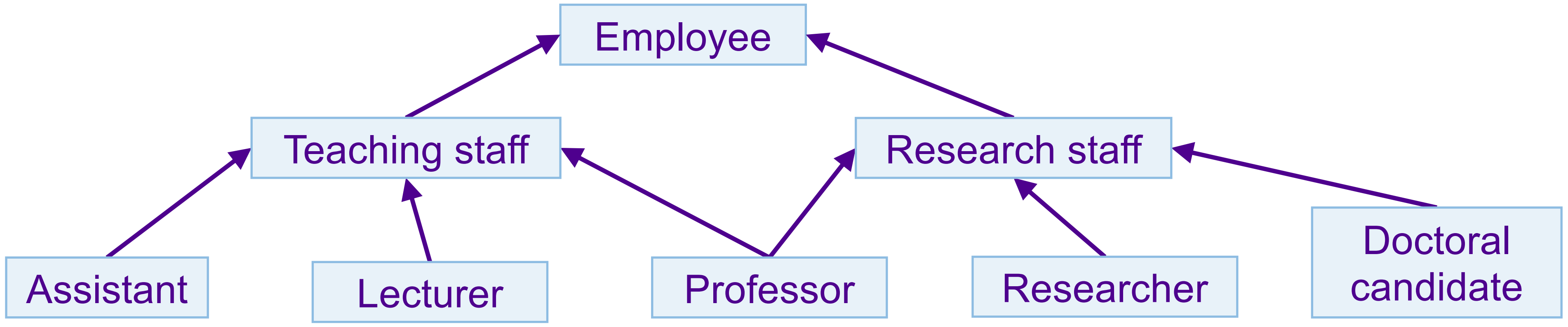
Class hierarchies

- Lower level classes in a hierarchy
  - Service implementation
  - Specialization
  - Dynamic binding

Tampereen yliopisto
Tampere University

Class hierarchies





Inheritance and reuse

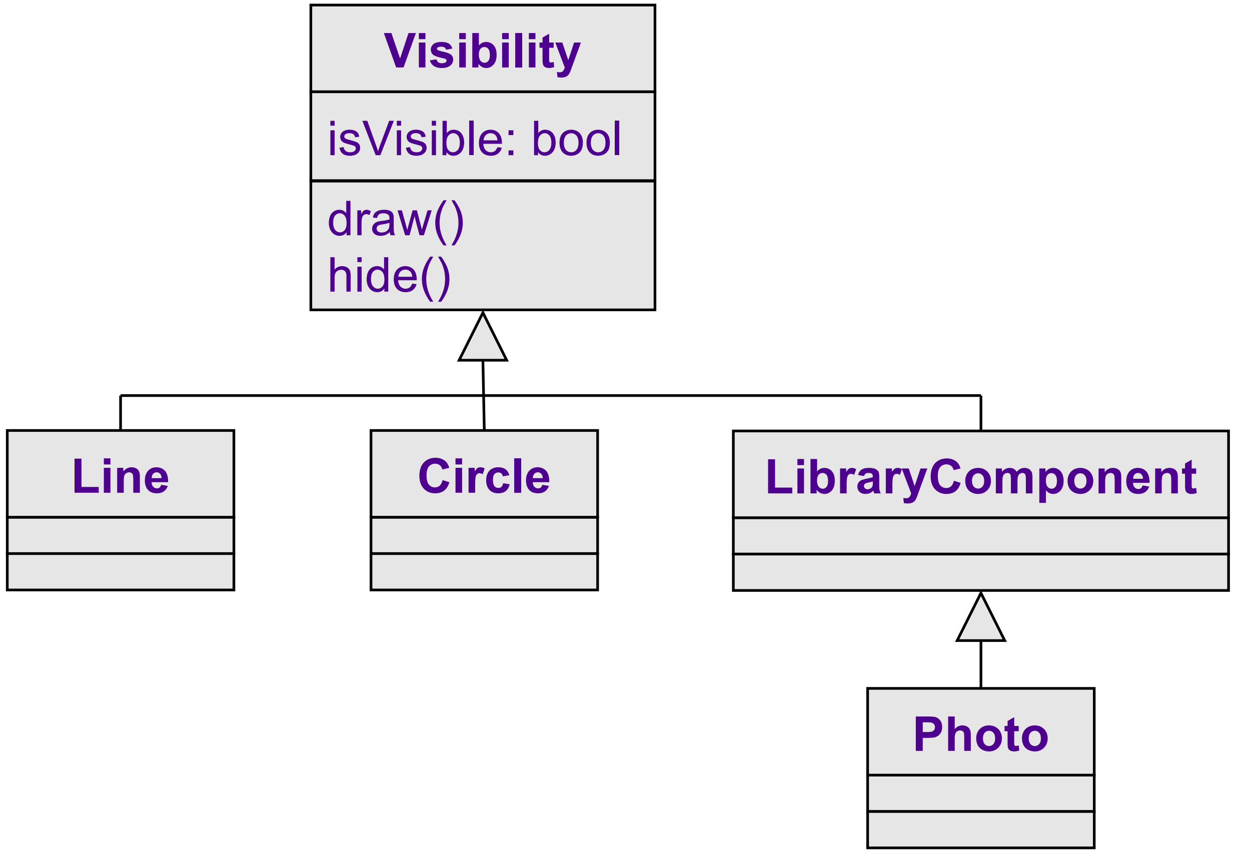
- Classes typically have common features
- Generalization
  - Common features can be written in one class (base class)
  - Common features can be taken in other classes (subclasses) by inheriting them

Tampereen yliopisto
Tampere University

Inheritance and reuse

- Subclass need not rewrite the services implemented in the base class
  - Reuse of the program code ⇒ no need to repeat the code
- Be careful about fragmentation of the code

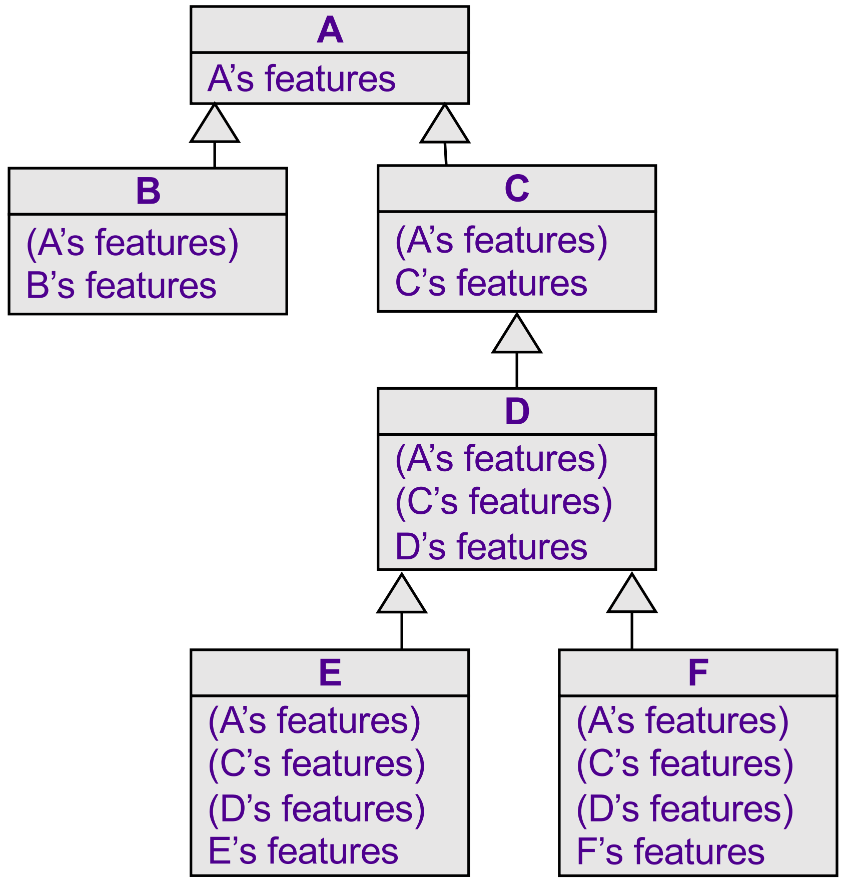
Inheritance and reuse

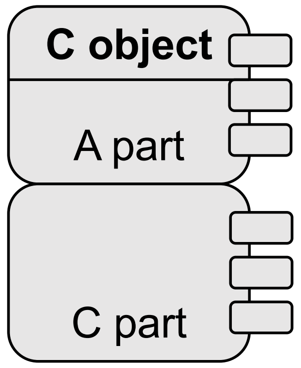


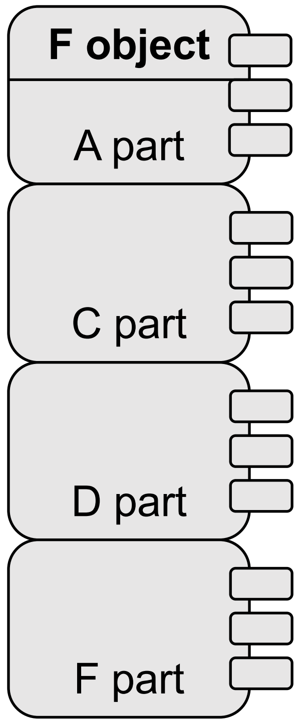
Inheritance in C++

- Subclass “inherits” all the features of the base class, and may add new ones
  - all base class features cannot be accessed
- Multiple inheritance
- Visibility specifiers (public, private, protected) in inheritance
  - visibility of the features
  - inheritance type

Inheritance and objects







Tampereen yliopisto
Tampere University

C++: Syntax of inheritance

```
class A {
    // Features of A
};
class B : public A {
    // Features that B has added (besides those of A)
};
class C : public A {
    // Features that C has added (besides those of A)
};
class D : public C {
    // Features that D has added (besides those of C)
};
```

Tampereen yliopisto
Tampere University

C++: Syntax of inheritance

```
// ...
class E : public D {
    // Features that E has added (besides those of D)
};
class F : public D {
    // Features that F has added (besides those of D)
};
```