Tampereen yliopisto
Tampere University

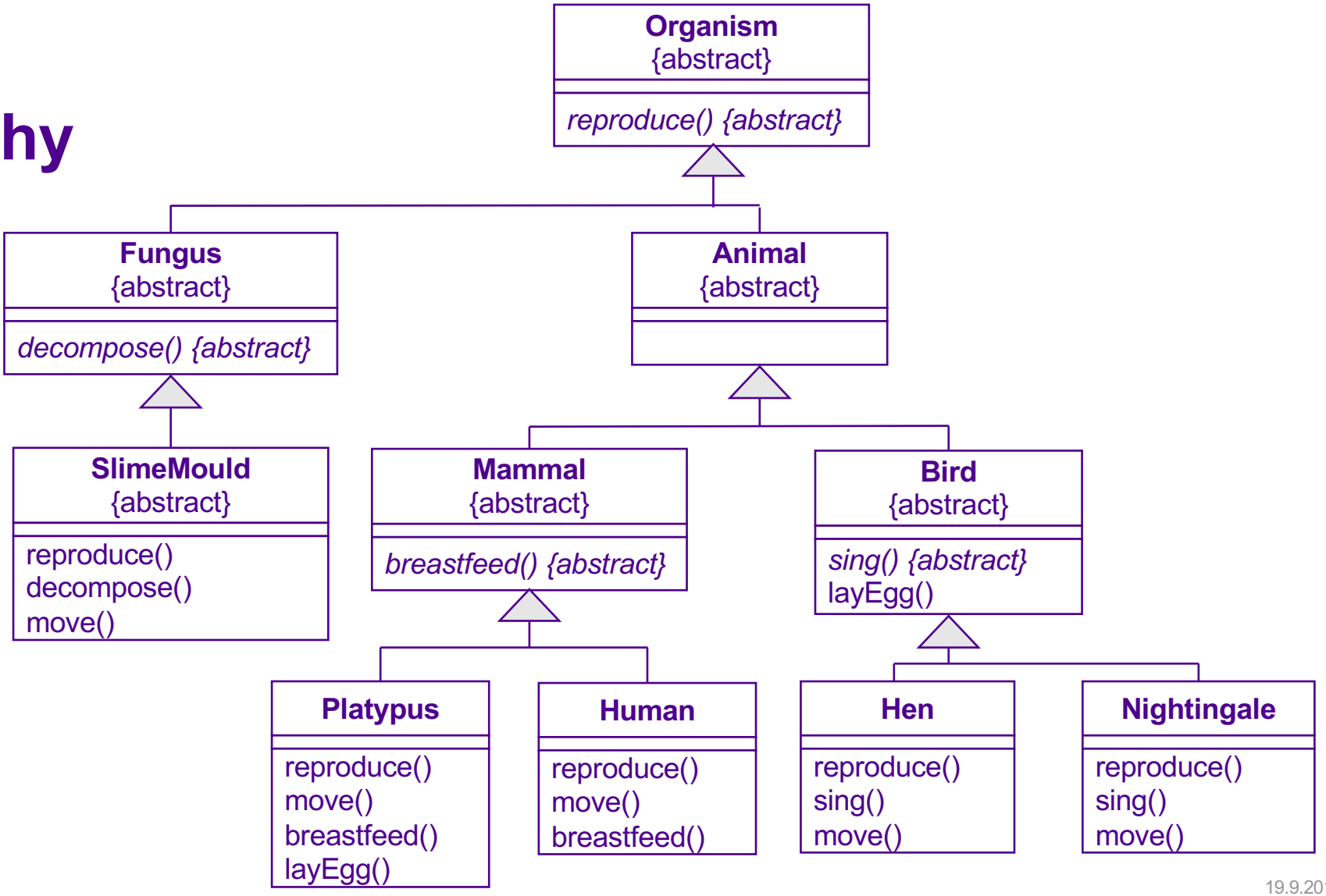
Class Hierarchies

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

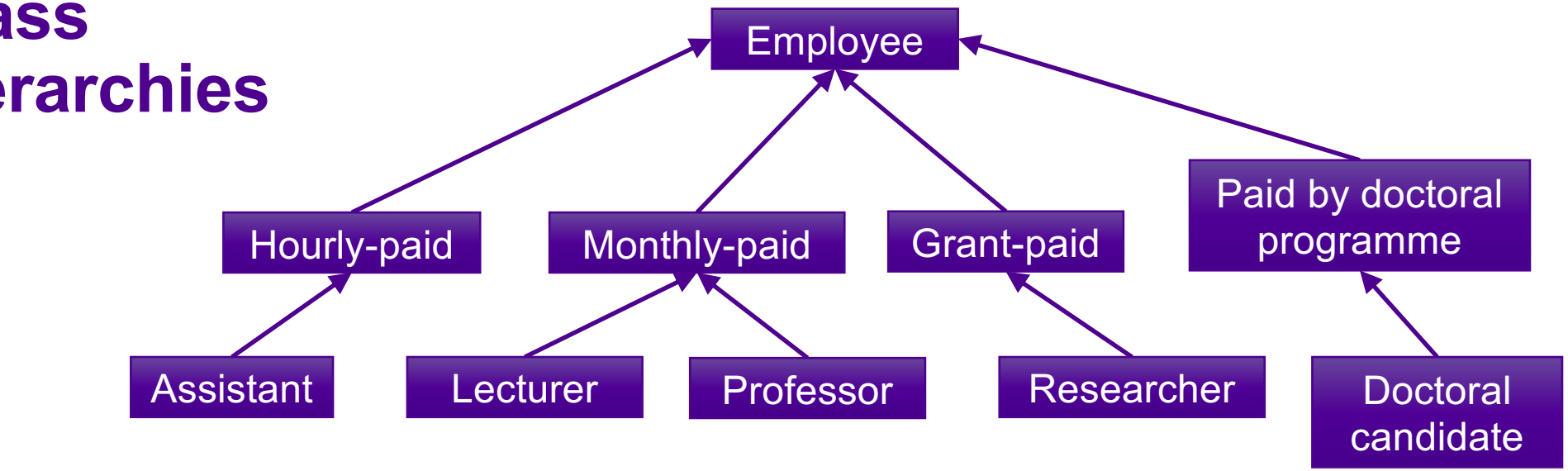
- Upper level classes of a hierarchy
  - “hypernyms”
  - Prescribe the interface of lower level classes
  - Polymorphism: allows referencing to objects with a “common name”

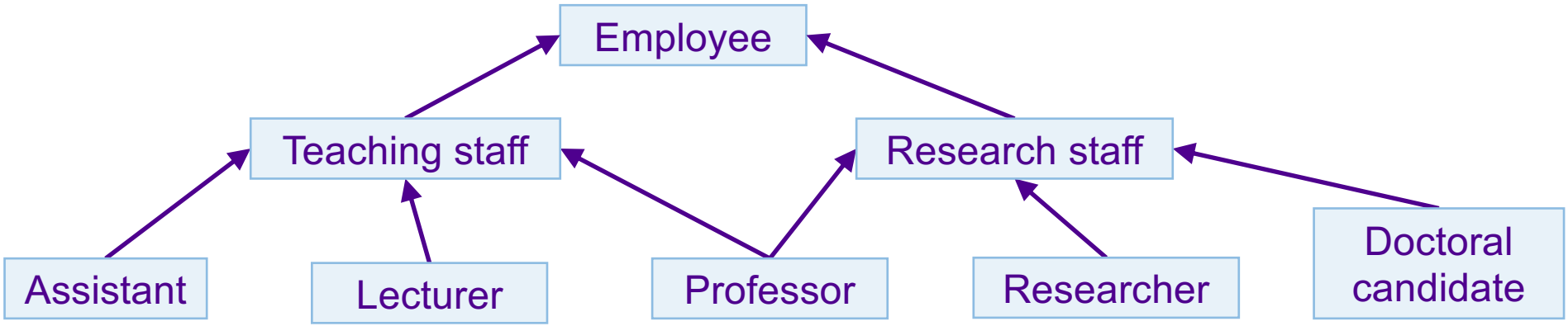
Class hierarchies

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

Tampereen yliopisto
Tampere University

Class hierarchies





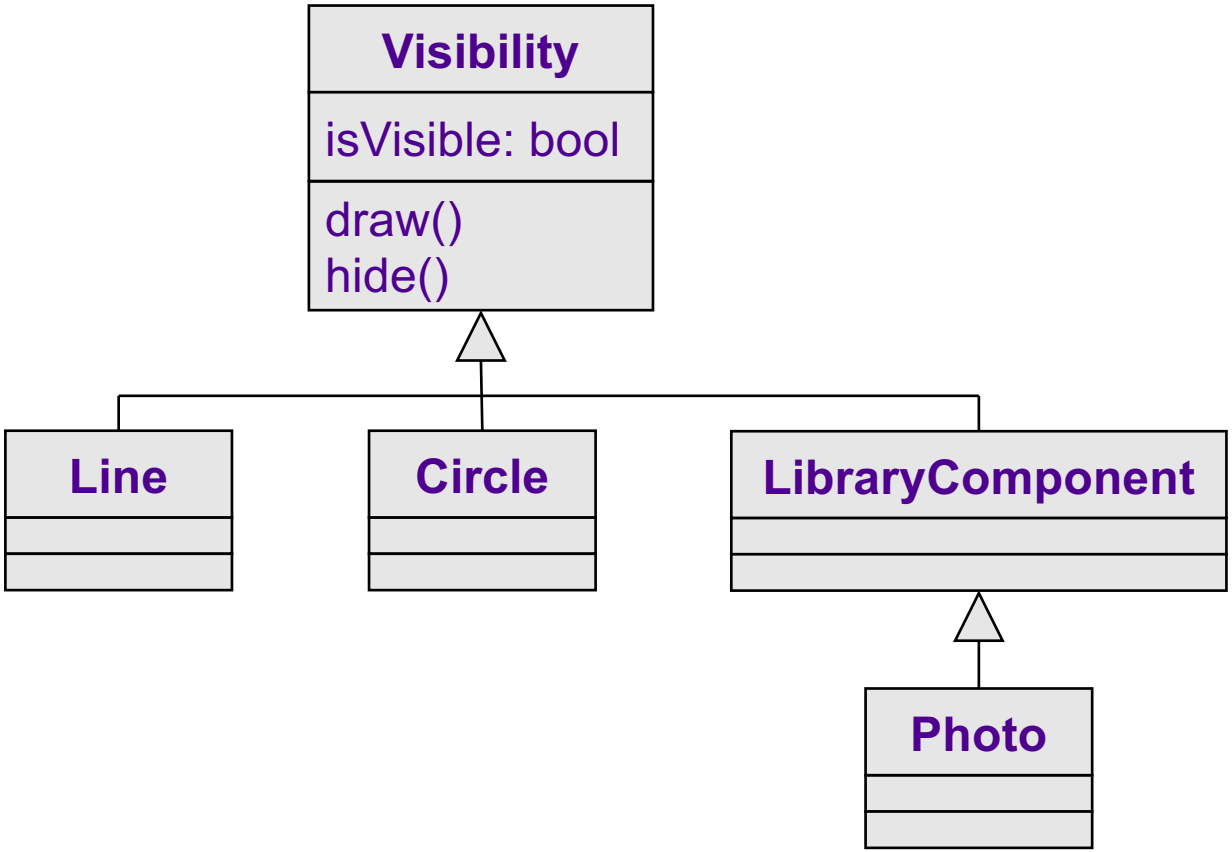
Inheritance and reuse

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

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

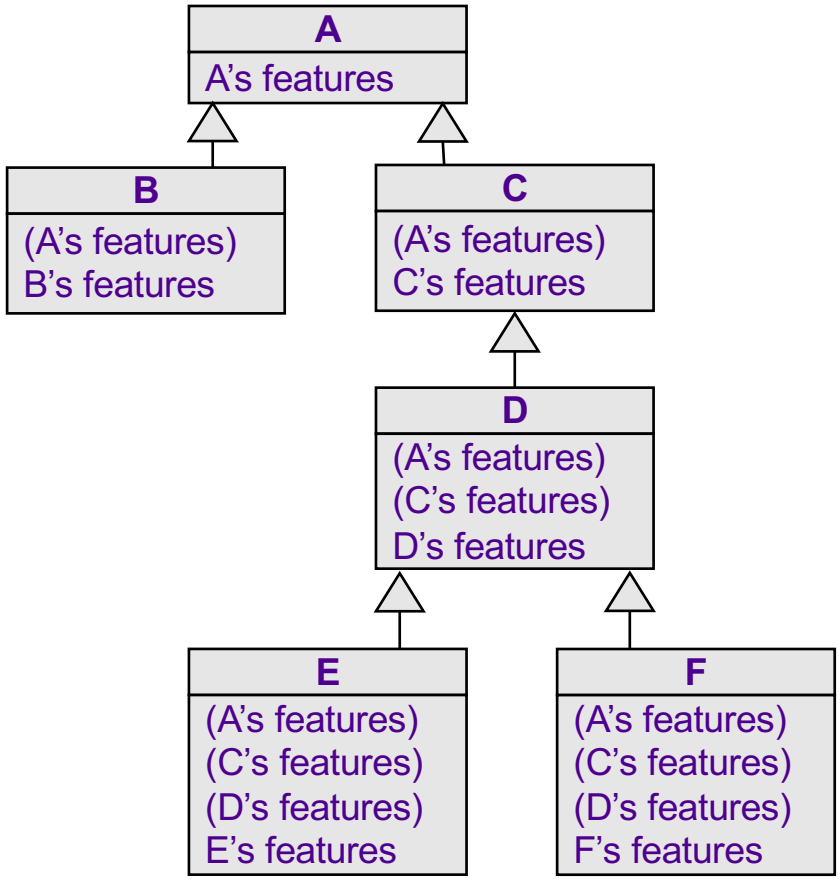
Inheritance and reuse

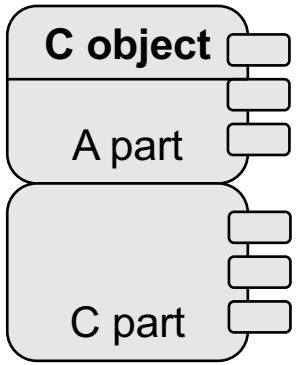


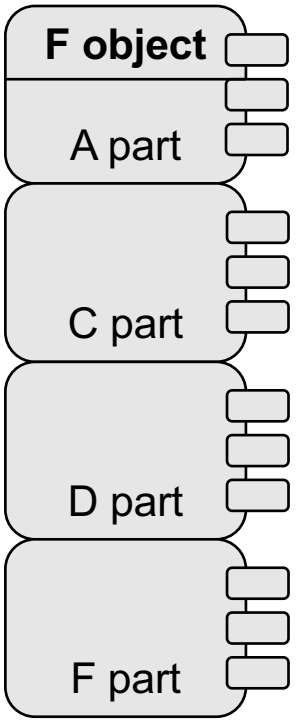
Inheritance in C++

- 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







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)
};
```

C++: Syntax of inheritance

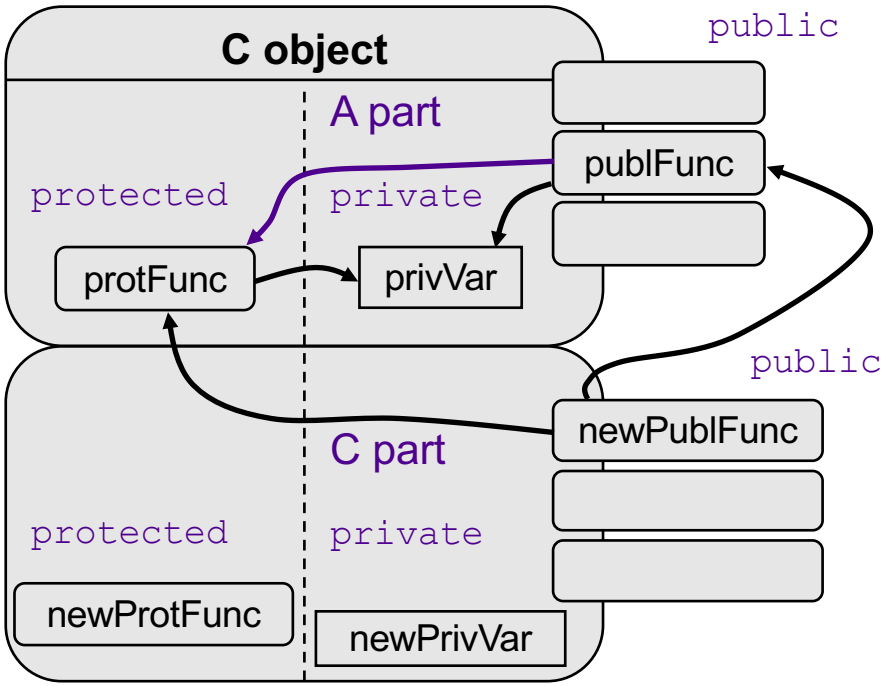
```
// ...
class E : public D {
    // Features that E has added (besides those of D)
};
class F : public D {
    // Features that F has added (besides those of D)
};
```

Tampereen yliopisto
Tampere University

Inheritance

23.9.2019

Inheritance and scopes



Inheritance and constructors

- Base class and the “additional part” of a subclass
- Subclass has no access to the private part of the base class
- Work distribution:
  - Base class constructor – initialization of the base class
  - Subclass constructor – call of base class constructor, initialization of the additional part
- Order of construction goes top-down in the hierarchy

Base class constructor

```
class LogMessage
{
public:
    LogMessage(string const& message);
    // ...
private:
    string message_;
};

LogMessage::LogMessage(string const& message) : message_(message)
{
}
```

Subclass constructor

```
class DatedLogMessage : public LogMessage {
public:
    DatedLogMessage(Date const& day, string const& message);
    // ...
private:
    Date day_;
};

DatedLogMessage::DatedLogMessage(Date const& day, string const&
message) : LogMessage(message), day_(day)
{
}
```

Object creation

```
LogMessage message(“I went to the
movies.");

DatedLogMessage datedMessage(today(), “It
was bad.");
```

Inheritance and destructors

- “Layers” as with constructors
- Work distribution:
  - Subclass destructor – cleanup of the additional part
  - Base class destructor – cleanup of the base class part
- Order of destruction goes bottom-up in the hierarchy
- Called automatically
- Base class destructor must always be *virtual*

Subclass object in relation to the base class

- Base class interface ⊂ subclass interface

⇒ subclass object is a valid base class object

  - In pointers
  - In references

- Is-a: "*the type of a subclass object is also that of the base class*"

Subclass object in relation to the base class

```
class BaseClass { … };
class Subclass : public BaseClass { … };
void function ( BaseClass& baseObject );

BaseClass* b_p = 0;
Subclass subObject;
b_p = &subObject;
function(subObject);
```

Inheritance as extension

- Subclass adds its own services
- Services of the base class as such
  - enables reuse of code
- “Do not use inheritance in vain”
- Another way is to add services into the original class

Definition of class Book

```
class Book {
public:
    Book(std::string const& title,
         std::string const& author);
    virtual ~Book();
    std::string getTitle() const;
    std::string getAuthor() const;
private:
    std::string title_;
    std::string author_;
};
```

Implementation of Book

```
Book::Book(string const& t, string const& a) : title_(t),
author_(a) {
    cout << “Book called " << title_ << " has been created" <<
endl;
}

Book::~Book() {
    cout << “Book called " << title_ << " destructed" << endl;
}

string Book::getTitle() const {
    return title_;
}
…
```

Definition of class `LibraryBook`

```
class LibraryBook : public Book
{
public:
    LibraryBook(std::string const& title,
                std::string const& author, Date const& retDay);
    virtual ~LibraryBook();
    bool isLate(Date const& today) const;
private:
    Date retDay_;
};
```

Implementation of LibraryBook

```
LibraryBook::LibraryBook(string const& title, string const& author, Paivays
const& retDay) :
    Book(title, author), retDay_(retDay) {
    cout << “Library book called " << title << " has been created" << endl;
}

LibraryBook::~LibraryBook() {
    cout << “Library book called " << getTitle() << " has been destructed" <<
endl;
}

bool LibraryBook::isLate(Date const& today) const {
    return retDay_.howFarAhead(today) < 0;
}
…
```