# Object Oriented Programming

Object Decomposition and Class as Object

2023/24

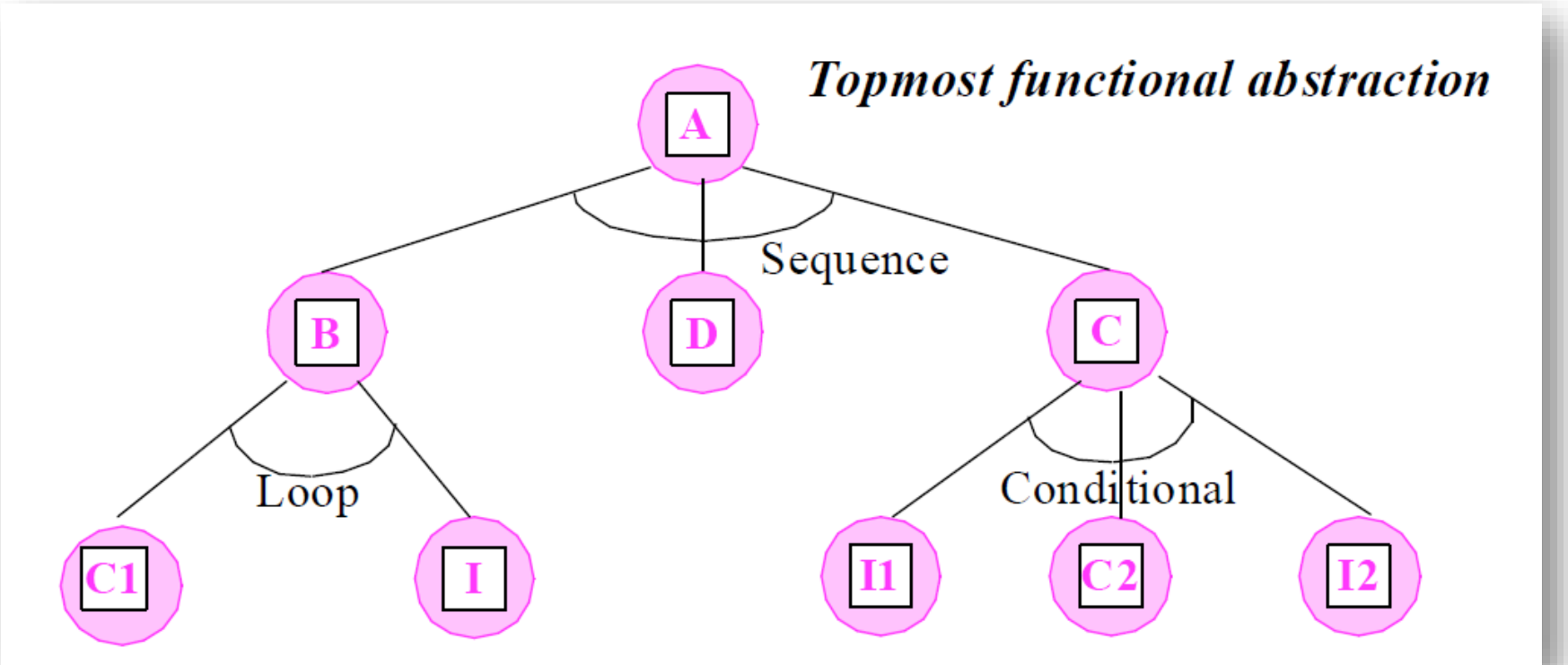
# Lecture Outline

- What is better? Functions or objects?
- Can a class be an object at the same time?
- Example

# Functions or Objects?

# Functions x Objects

- Is it better to have the structure of a program based on functions or data?
- We can see the design of the system from two perspectives:
  - As a set of functions (it answers a questions **WHAT** the system will do)
  - As a set of objects that cooperate (it answers a question **WHO** is responsible for performing of functions)


Topmost functional abstraction
A
Sequence
B
D
C
Loop
Conditional
C1
I
I1
C2
I2

# Issues

- Extendibility
- Reusing
- Composibility

## Object-oriented software construction (definition 1)

Object-oriented software construction is the software development method which bases the architecture of any software system on modules deduced from the types of objects it manipulates (rather than the function or functions that the system is intended to ensure).

## OBJECT MOTTO

***Ask not first what the system does:***

***Ask what it does it to!***

# Example of Assignment

- Consider a small **bank** with a limited number of **clients** and **accounts**. At the bank, the number of clients and accounts increase.
- Each account has one owner and may have one partner, both are clients of the bank and have name and code. We can deposit, withdraw and check the status of an account (balance). If there is not enough money in the account, we are not able to withdraw.
- Individual accounts have an interest rate, either basic or specific. Once upon a time, amount based on the corresponding interest rate is added to all bank accounts.
- Accounts and clients can be searched by number or code.

# Function or Objects?

- Computation involves three kinds of ingredient: processors (or threads of control), actions (or functions), and data (or objects).
- A system's architecture may be obtained from the functions or from the object types.
- A description based on object types tends to provide better stability over time and better reusability than one based on an analysis of the system's functions.
- It is usually artificial to view a system as consisting of just one function. A realistic system usually has more than one "top" and is better described as providing a set of services.

# Top-down approach or vice versa?

- It is preferable not to pay too much attention to ordering constraints during the early stages of system analysis and design. Many temporal constraints can be described more abstractly as logical constraints.

- Top-down functional design is not appropriate for the long-term view of software systems, which involves change and reuse.

# Why Objects?

- Object-oriented software construction bases the structure of systems on the types of objects they manipulate.
- In object-oriented design, the primary design issue is not what the system does, but what types of objects it does it to. The design process defers to the last steps the decision as to what is the topmost function, if any, of the system.
- To satisfy the requirements of extendibility and reusability, object-oriented software construction needs to deduce the architecture from sufficiently abstract descriptions of objects.
- Two kinds of relation may exist between object types: client and inheritance.

# Classes as Objects? Why?

# Class as Object

- The object-oriented approach is generally based on the assumption that "everything is an object" (thus, types are also objects).
- Can a class also be an object? And under what conditions?
- *Objects have states and behavior…*

# State and Behavior of Classes

- State is represented by data.
- Behavior is represented by methods.
- Encapsulation and information hiding must be accomplished.
- It must be possible to send a message to the class (call its method or ask for its value).

# Example

# Declaration and Definition

```
class StaticValue
{
private:
    static int value;

public:
    static void IncValue();
    static int GetValue();
};
```

```
int StaticValue::value = 0;

void StaticValue::IncValue()
{
    StaticValue::value += 1;
}

int StaticValue::GetValue()
{
    return StaticValue::value;
}
```

# Using

```
int main()
{
    cout << StaticValue::GetValue() << endl;
    StaticValue::IncValue();
    cout << StaticValue::GetValue() << endl;

    StaticValue *sv = new StaticValue();
    cout << sv->GetValue() << endl;

    getchar();
    return 0;
}
```

# Explanation

- The class owns data and methods declared as static
- Class objects (instances) also have access to them
- It is necessary to distinguish between **class** and **instance** variables and methods

# Appropriate Conventions

- The recipient of the message does not have to be specified in the context of the class when accessing data or methods, however…
- …to avoid misunderstandings, it is good:
  - To access instance data/methods us this form:
    **OBJECT_NAME->METHOD_NAME**
  - To access class data/methods us this form:
    **CLASS_NAME::METHOD_NAME**

# Message Recipient

- The recipient of the message is, therefore, either
  - an object (instance) of this class in the case of an instance variable or method, or
  - a class itself in the case of a class (static) variable or method.

# Where is the difference?

```
class StaticValue
{
private:
    static int value;
    StaticValue();

public:
    static void IncValue();
    static int GetValue();
};
```

# Class Without Objects

```
int main()
{
    cout << StaticValue::GetValue() << endl;
    StaticValue::IncValue();
    cout << StaticValue::GetValue() << endl;

    StaticValue *sv = new StaticValue();
    cout << sv->GetValue() << endl;

    getchar();
    return 0;
}
```

# Constructor? Destructor?

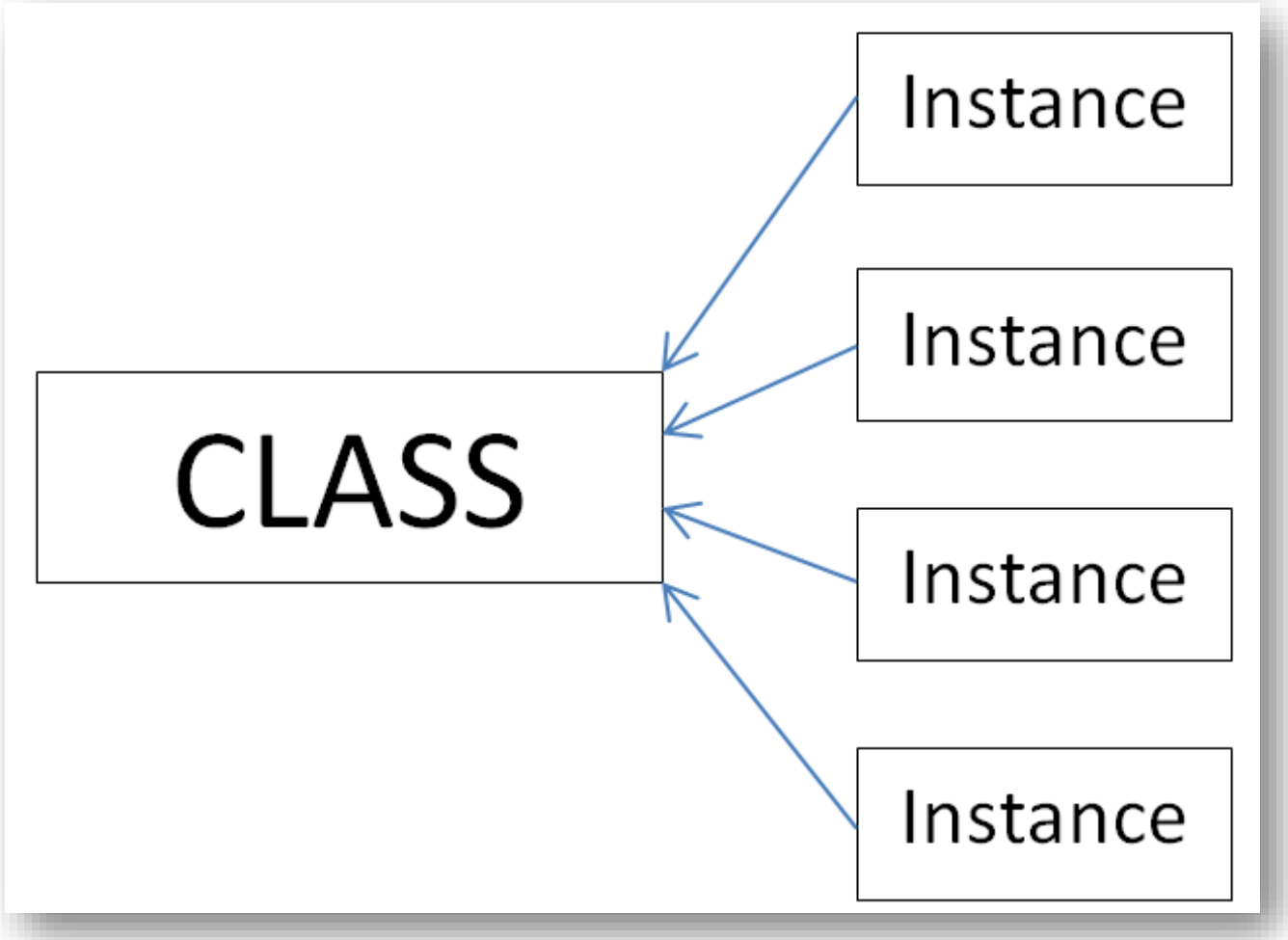
- Each class exists throughout the program execution
- If the class has class (static) variables, then they must be initialized separately

```
int StaticValue::value = 0;
```

- Neither the constructor nor the destructor exists for the class as an object.

# Who knows about whom?

- We create objects (instances) using the class constructor, but the class knows nothing about these instances.
- We cannot access members of an object from any class method.
- Objects (instances) of the class have access to (static) members of the class (when used, it may not be recognizable which kind of member we are working with).


CLASS
Instance
Instance
Instance
Instance

# Which Form is Correct and Why?

```
int main()
{
    cout << StaticValue::GetValue() << endl;
    StaticValue::IncValue();
    cout << StaticValue::GetValue() << endl;

    StaticValue *sv = new StaticValue();
    cout << sv->GetValue() << endl;

    getchar();
    return 0;
}
```

# When to Use Class as Object?

- Creating a library of functions (e.g., mathematics)
- We need objects (instances) to share common (static) data or methods
- E.g., counting of the number of objects (instances) of a class

# Complete the Class for Counting Objects

```
class Client
{
private:
    int code;
    string name;

public:
    Client(int c, string n);

    int GetCode();
    string GetName();
};
```

# Declaration and Definition

```
class Client
{
private:
    static int objectsCount;
    int code;
    string name;

public:
    static int GetObjectsCount();

    Client(int c, string n);
    ~Client();

    int GetCode();
    string GetName();
};
```

```
int Client::objectsCount = 0;

Client::Client(int c, string n)
{
    this->code = c;
    this->name = n;
    Client::objectsCount += 1;
}

Client::~Client()
{
    Client::objectsCount -= 1;
}

int Client::GetObjectsCount()
{
    return Client::objectsCount;
}
```

# Seminar Assignments

- Implement the examples from the lecture and add the counting of existing objects to the *Client* and *Account* class

- Design and implement additional examples of class variables and methods (e.g., the shared interest rate for all accounts)

# Seminar Questions

- What is the difference between functional, and object decomposition of a program?
- Why do we prefer object decomposition and what are the main issues of functional decomposition?
- Under what conditions can we consider a class as an object and how can it be implemented in C ++?
- Explain the difference between class and instance members and describe their availability.
- How can we consistently differentiate work with class and instance members in C ++?
- Does the class in the role of the object need a constructor or destructor, respectively, and why?

# Sources


- Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall 1997. [101-120]