


# Programming II

Object Decomposition  
Class as Object





# Lecture Outline

- What is better? Functions or objects?
  - Can a class be an object as well?
  - Example
- 



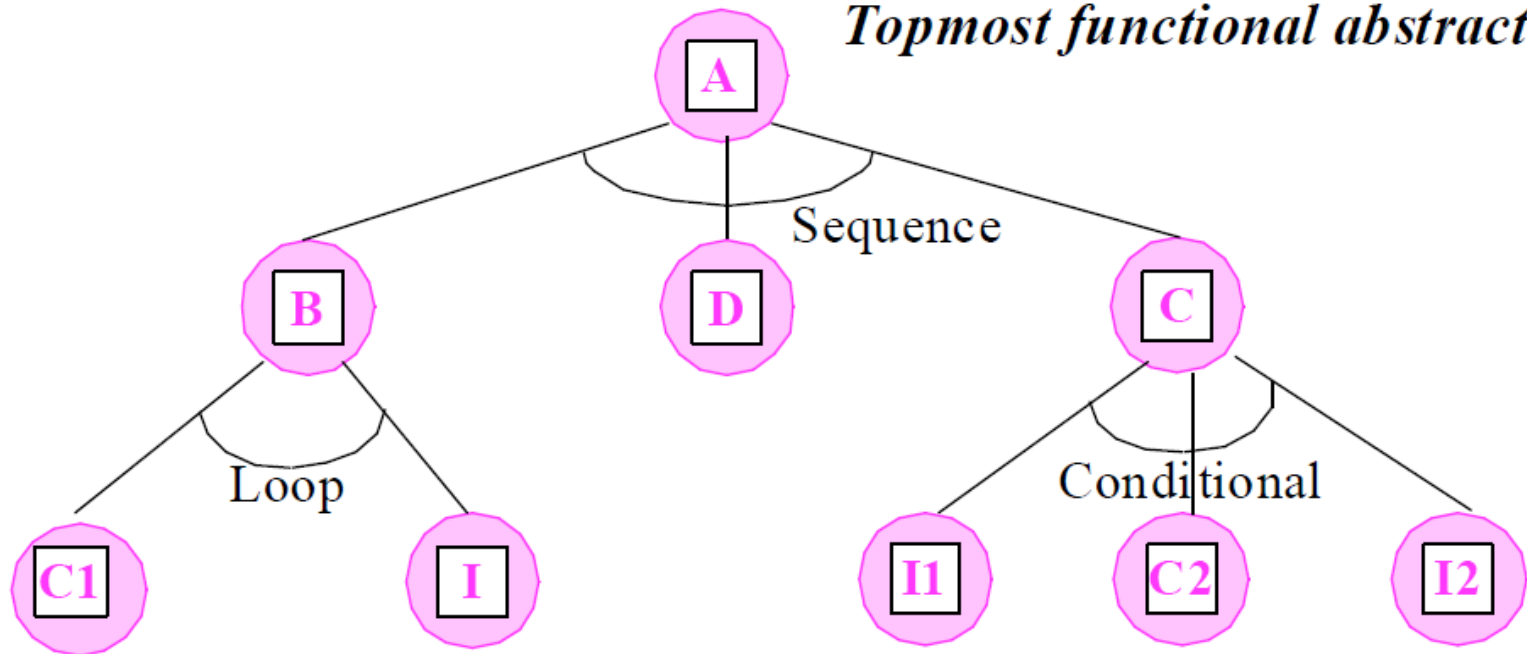
# **Functions or objects?**



# Functions x Objects


- What is better? A program structure based on functions or data?
- We can see a program design in two different ways:
  - As a set of functions (it answer a question “WHAT and HOW the system will do”).
  - As a set of cooperating objects (it answer a question “WHO is responsible for the functionality”).

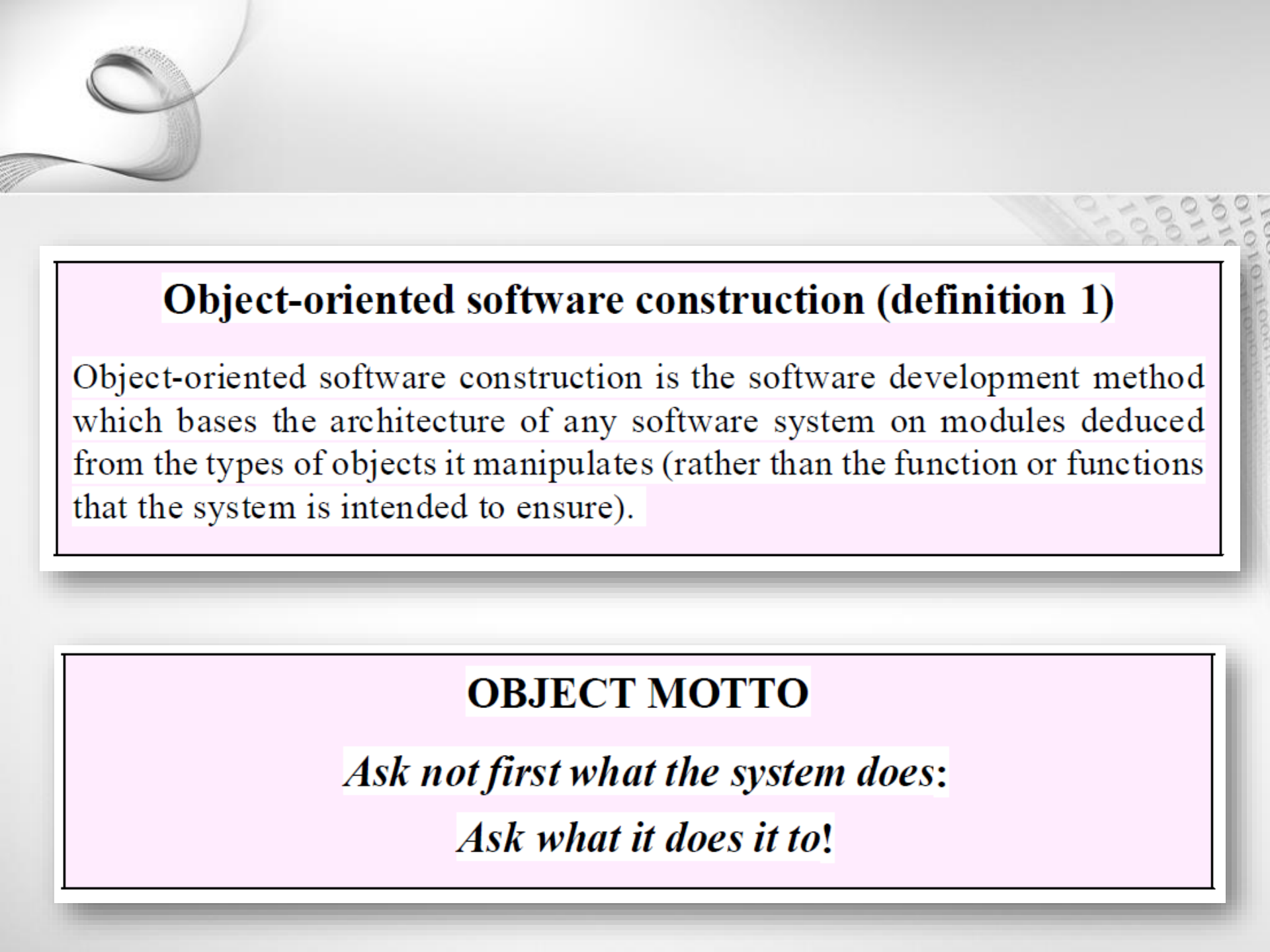
*Topmost functional abstraction*





# Problems

- Extendibility
  - Reusability
  - Compatibility
- 



## **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).

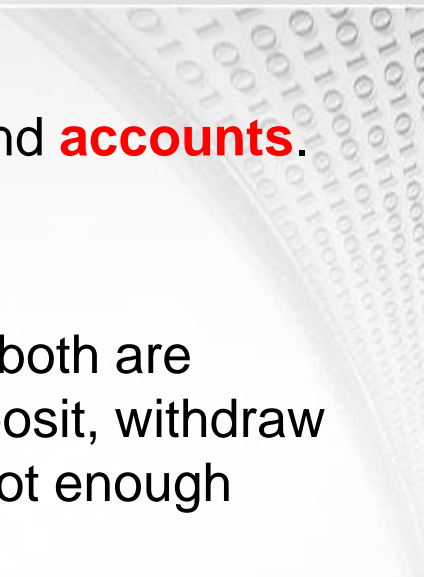
### **OBJECT MOTTO**

*Ask not first what the system does:*

*Ask what it does it to!*




# Example

- 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.
- 



- 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.
- 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.

- 
- 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 based on assumption that “everything is an object”.
- Can a class be at the same time an object? Under what conditions?
- Objects have their state and behavior.



# State and Behavior of Class

- The state is represented by data.
- Behavior is represented by methods.
- Encapsulation and information hiding must be applied.
- It must be possible to send a message to class (call its method).



# Example

# Class Declaration

```
class StaticValue
{
private:
    static int Value;
public:
    static void IncValue();
    static int GetValue();
};
```

# Class Implementation (definition)

```
int StaticValue::Value = 0;

void StaticValue::IncValue(){
    StaticValue::Value++;
}

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



# Using the Class

```
int main() {  
    cout << StaticValue::GetValue() << endl;  
    StaticValue::IncValue();  
    cout << StaticValue::GetValue() << endl;  
  
    StaticValue * v = new StaticValue();  
    cout << v->GetValue() << endl;  
  
    getchar();  
    return 0;  
}
```



# How it works...

- Data and methods declared as “static” belong to the class.
- Instances of the class can approach these data and methods.
- We need to distinguish **class** and **instance** members (variables and methods).



# Suitable conventions

- In the context of the class when accessing the data or methods, there is no need to use the recipient of the message.
- To avoid any misunderstanding, it is good:
  - To approach instance data/methods, use a form **OBJECT\_NAME->METHOD\_NAME**
  - To approach class data/methods, use a form **CLASS\_NAME::METHOD\_NAME**



# Message Recipient

- A recipient of a message can be:
  - An object (instance) of this class in case of instance variable or method
  - The class itself in case of 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 Instances

```
int main() {  
    cout << StaticValue::GetValue() << endl;  
    StaticValue::IncValue();  
    cout << StaticValue::GetValue() << endl;  
  
    StaticValue * v = new StaticValue();  
    cout << v->GetValue();  
  
    getchar();  
    return 0;  
}
```



# Constructor? Destructor?

- Class exists throughout the runtime of a program.
- When a class has its (static) variables they must be defined individually:

```
int StaticValue::Value = 0;
```

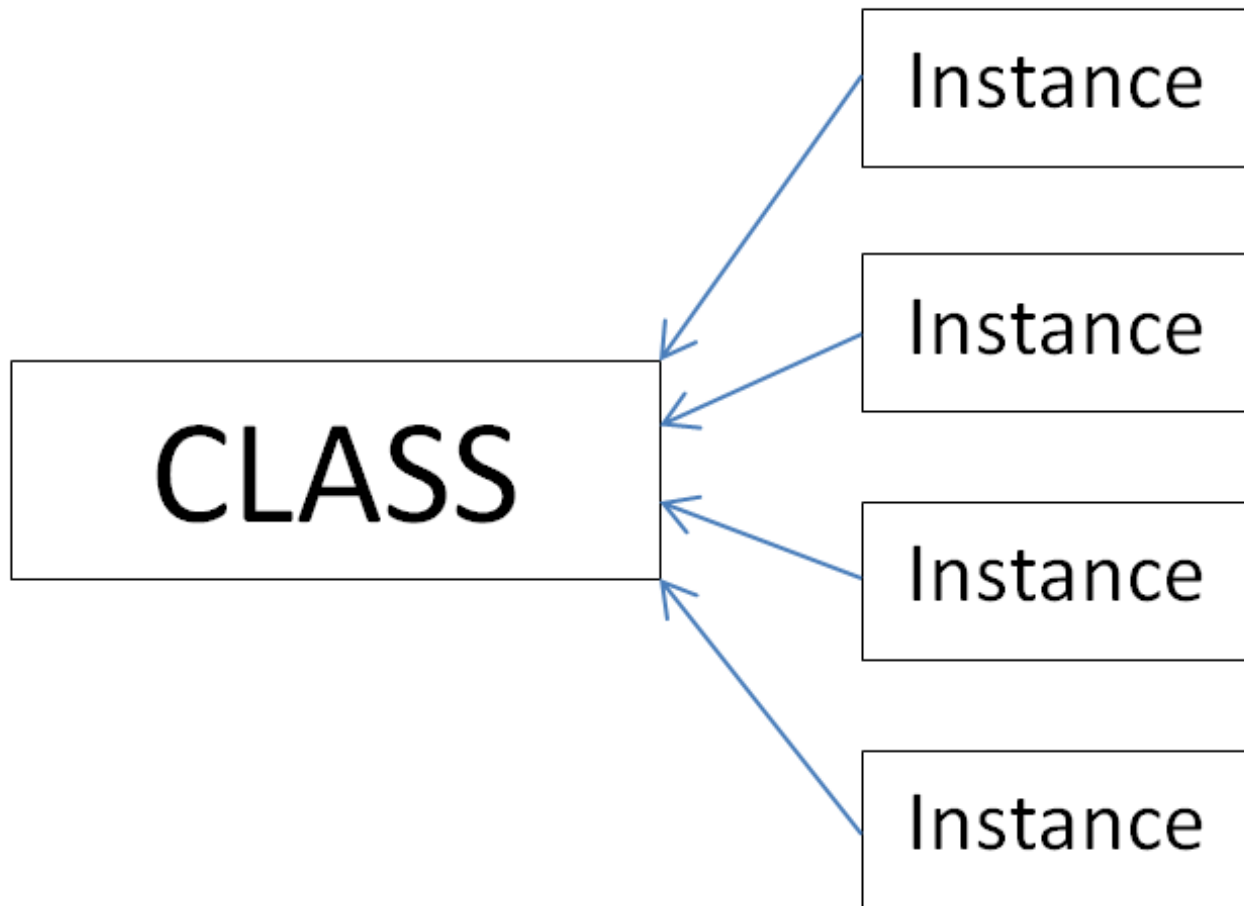
- There is no need to have constructors and destructors.



# Who knows for whom?

- Using the constructor of class, objects (instances) are created. However, the class knows nothing about them.
- We cannot approach object (instance) members from a class (static) method.
- Objects (instances) of a class have access to class (static) members.





# What is better?

```
int main() {  
    cout << StaticValue::GetValue() << endl;  
    StaticValue::IncValue();  
    cout << StaticValue::GetValue() << endl;  
  
    StaticValue * v = new StaticValue();  
    cout << v->GetValue() << endl;  
  
    getchar();  
    return 0;  
}
```



# When to use Class as Object?

- Library of functions (e.g. Math).
- Objects (instances) share common data.
- Counting of existing objects (instances) of a class.



# Extend Class for Counting Objects


```
class Client {  
private:  
    int code;  
    string name;  
  
public:  
    Client(int c, string n);  
  
    int GetCode();  
    string GetName();  
};
```

```
class Client {  
private:  
    static int objectsCount;  
    int code;  
    string name;  
  
public:  
    Client(int c, string n);  
    ~Client();  
  
    static int GetObjectsCount();  
    int GetCode();  
    string GetName();  
};
```

```
int Client::objectsCount = 0;  
  
Client::Client(int c, string n){  
    this->code = c;  
    this->name = name;  
    Client::objectsCount += 1;  
}  
  
Client::~~Client(){  
    Client::objectsCount -= 1;  
}  
  
int Client::GetObjectsCount(){  
    return Client::objectsCount;  
}
```



# Sources

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



# Questions

- What is the difference between functional and object-based decomposition?
- Why do we prefer object-based decomposition? What are the key problems of functional decomposition?
- Under what conditions can be a class considered as an object and how it implement in C++?
- Explain the difference between the members of classes and instances, and describe their accessibility.
- How can we, in C++, clearly distinguish work with members of classes and instances?
- In the role of object, do the class need constructor and destructor? Why?