CS302 - Data Structures

using C++

Topic: Object-Oriented Concepts

Kostas Alexis

AUTONOMOUS ROBOTS LAB

Object-Oriented Solutions

- **Functions & methods implement algorithms**
- **Algorithm**
  - A step-by-step recipe for performing a task within a finite period of time
  - Algorithms operate on a collection of data
- **Create a good set of modules**
  - Modules must store, move, and alter data
  - Modules communicate with one another
  - Organize your data collection to facilitate operations on the data

Object-Oriented Analysis

- **Process to develop**
  - An understanding of the problem
  - The requirements of a solution
    - What a solution must be and do
    - Not how to design or implement it
- **Generates an accurate understanding of what end users will expect the solution to be and do**
- **Think about the problem**
  - Not how to solve it
- **Objects can represent**
  - Real-world objects
  - Software systems
  - Ideas

Object-Oriented Analysis

- **Process to develop**
  - An understanding of the problem
  - The requirements of a solution
    - What a solution must be and do
    - Not how to design or implement it
- **Generates an accurate understanding of what end users will expect the solution to be and do**
- **Think about the problem**
  - Not how to solve it
- **Objects can represent**
  - Real-world objects
  - Software systems
  - Ideas

Expresses an understanding of the problem and the requirements of a solution in terms of objects within the problem domain.

Object-Oriented Design

- **Expresses an understanding of a solution that fulfills the requirements discovered during OOA**
- **Describes a solution in terms of**
  - Software objects
  - Collaborations of these objects with one another

Object-Oriented Design

- **Design Collaborations Among Objects**
  - Objects collaborate when they send messages
    - Call each other's methods
  - Collaborations should be meaningful and minimal
- **Design creates models of a solution**
  - Some emphasize interactions among objects
    - methods
  - Others emphasize relationships among objects
    - **"is a"** and **"has a"** relationships

Object-Oriented Design

- **Cohesion**
  - Degree to which methods in a class operate on data within the class
  - High cohesion (good)
    - Methods only operate on class data
    - Easy to reuse in other software projects
    - Easy to revise or correct
    - Robust:
      - Less likely to be affected by change
      - Performs well under unsual conditions
  - A person with low cohesion has “too many irons in the fire”

Object-Oriented Design

- **Coupling**
  - Degree to which a class depends on other classes
  - Modules with low coupling are independent of one another
  - Low coupling (good)
    - Easier to change
      - Change to one module won't affect another
    - Easier to understand
    - Easier to reuse
    - Has increased cohesion
  - Coupling cannot be (and should not) be eliminated entirely
    - Objects must collaborate
    - Class diagrams show dependencies among classes, hence coupling

Object-Oriented Design

- **Minimal and complete interfaces**
  - A class interface declares publicly accessible methods (and data)
  - Describes only way for programmers to interact with the class
  - Classes should be easy to understand, and so have few methods
  - Desire to provide power is at odds with this goal

- **Complete interface**
  - Provides methods for any reasonable task consistent with the responsibilities of the class
  - Important that an interface is complete

- **Minimal interface**
  - Provides only essential methods
  - Classes with minimal interfaces are easier to understand, use, and maintain
  - Less important than completeness

Object-Oriented Design

- **Signature**
  - Programmer's interface for a method or function
  - **Name**
    - How to refer to the abstracted code
  - **Arguments (number, order, type)**
    - What is sent to the method
    - Should not modify parameters (ideal world)
      - Qualifiers (**`const`**)
  - **Return Type**
    - How code communicates results back to caller
    - Should return single value (ideal world)

Object-Oriented Programming

- **Object-oriented languages**
  - Enable us to build classes objects
- **A class combines**
  - **Attributes** (characteristics) of objects
    - Typically data
    - Called data members
  - **Behaviors** (operations)
    - Typically operate on the data
    - Called methods or member functions

Object-Oriented Programming

- **Encapsulation**
  - Objects combine data and operations
  - Hides inner details
- **Inheritance**
  - Classes can inherit properties from other classes
  - Existing classes can be reused
- **Polymorphism**
  - Objects can determine appropriate operations at execution time
    - In C++ this requires pointers

Abstraction

- **Separates the purpose of a module from its implementation**
  - Specifications for each module are written before implementation
    - Specifications are in the C++ header file
- **Functional abstraction**
  - Separates the purpose of a function from its implementation
  - C++ function prototypes (headers)
- **Data abstraction**
  - Focuses on the operations of data, not on the implementation of the operations
  - C++ structures

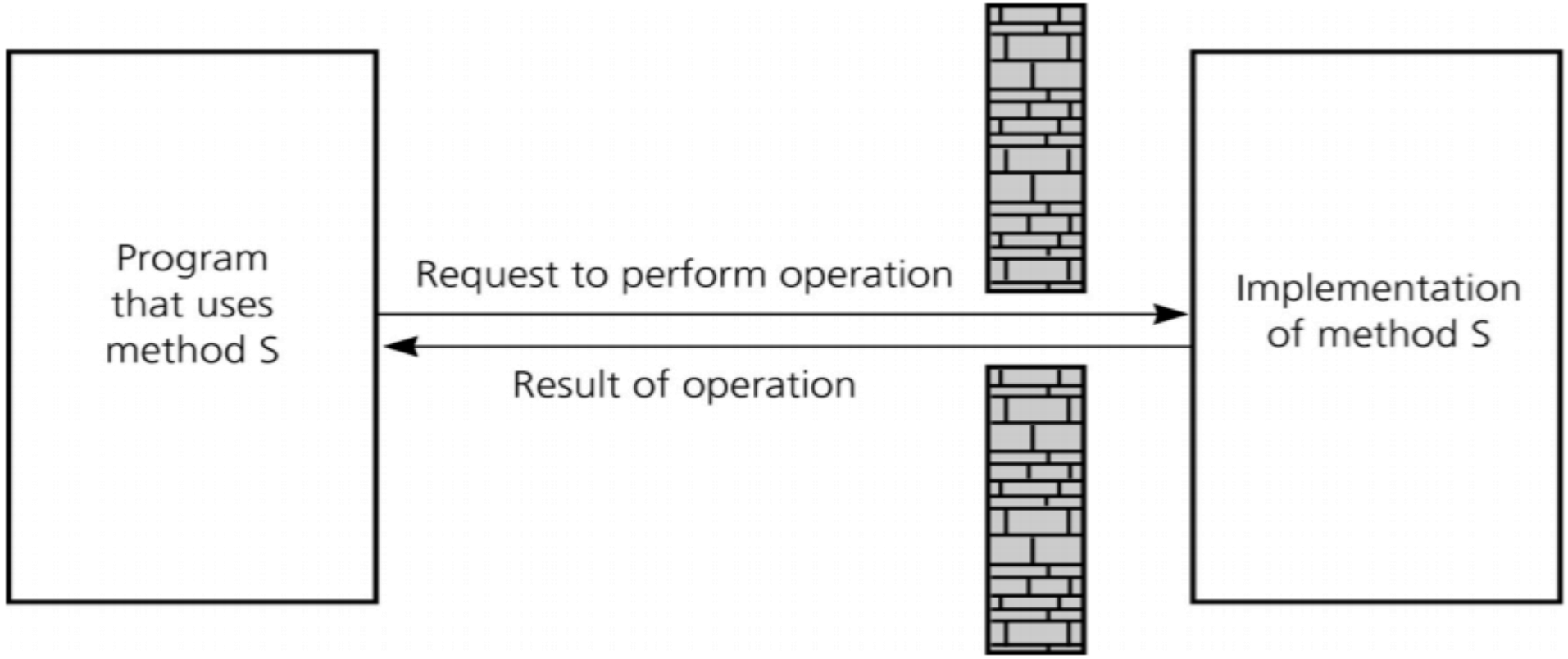
Abstraction

- **Abstract data type (ADT)**
  - A collection of data and operations on that data
  - ADT operations can be used without knowing their implementations or how data is stored
  - Classes
- **Data Structure**
  - A construct that within a programming language used to store a collection of data
  - An implementation of an ADT

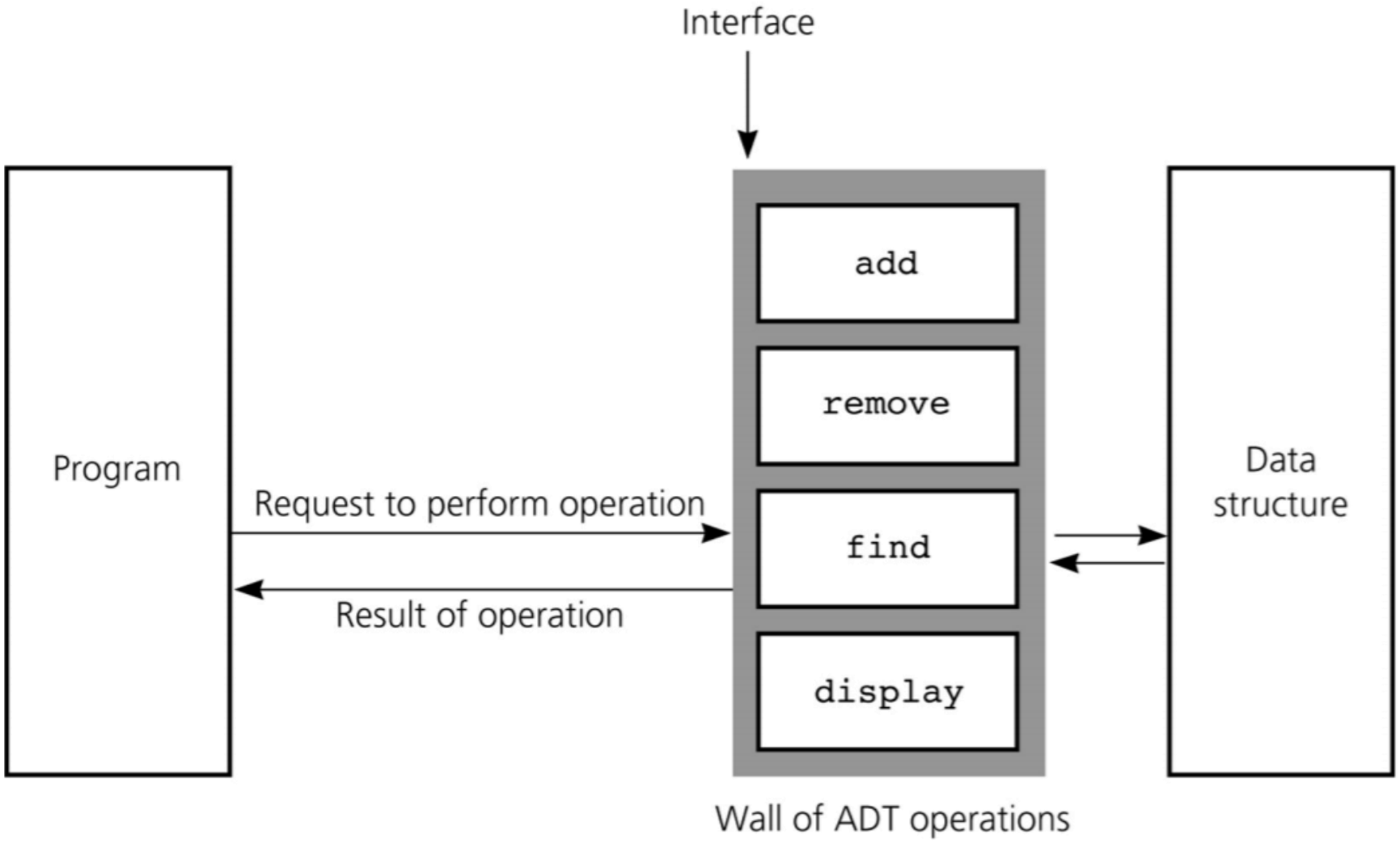
Information Hiding

- **Hide details within a module**
  - Ensure that no other module can tamper with these hidden details
- **Public view of a module**
  - Described by its specifications
- **Private view of a module**
  - Implementation details that the specifications should not describe

Communicate through a slit in the wall



Isolate data structure from program using it



Thank you