DATA STRUCTURE AND ALGORITHMS

LECTURE 3

Abstract Data Type and List ADT

DATA STRUCTURE AND ALGORITHMS

LECTURE 3a

Abstract Data Type

Reference links:

https://cs.nyu.edu/courses/fall07/V22.0102-002/index.html

By Prof Evan Korth- NYU

https://www.comp.nus.edu.sg/~stevenha/cs2040.html

By Dr. Steven Halim - NUS

Lecture outline

- Abstraction in Programs
- Abstraction Data Type (ADT)
  - Definition
  - Benefits
- Abstraction Data Type Examples

Abstraction

- Abstraction:
  - The process of isolating implementation details and extracting only essential property from an entity.
- Program = data + algorithms
- Abstraction involving a program:
  - Data abstraction
    - What operations are needed by the data
  - Functional abstraction
    - What is the purpose of a function (algorithm)

Abstraction Data Type (ADT)

- Abstract Data Type (ADT):
  - End result of data abstraction
  - A collection of data together with a set of operations on that data
  - ADT = Data + Operations
- ADT is a language independent concept
  - Different language supports ADT in different ways
  - In C++, the class construct is the best match
  - In Java, an ADT can be expressed by an interface

Abstraction Data Type (ADT)

- Important Properties of ADT:
  - Specification:
    - The supported operations of the ADT
  - Implementation:
    - Data structures and actual coding to meet the specification

ADT: Specification and Implementation

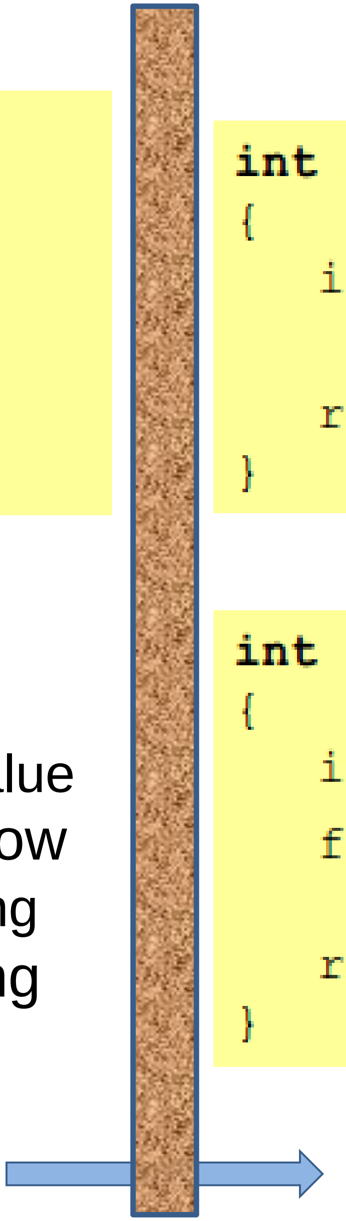
- Specification and implementation are disjointed:
  - One specification
  - One or more implementations
    - Using different data structure
    - Using different algorithm
- Users of ADT:
  - Aware of the specification only
    - Usage only base on the specified operations
  - Do not care / Need not know about the actual implementation
    - i.e. Different implementation do not affect the user

Abstraction as Wall: Illustration

```
int main()
{
    int fac5;

    fac5 = factorial(5);

    ... ...
}
```

Call function

- ❑ **main()** needs to know
  - ▪ **factorial()**'s purpose
  - ▪ Its parameters and return value
- ❑ **main()** does not need to know
  - ▪ **factorial()** internal coding
- ❑ Different **factorial()** coding
  - ▪ Does not affect its users!
- ❑ We can build a wall to shield **factorial()** from **main()** !



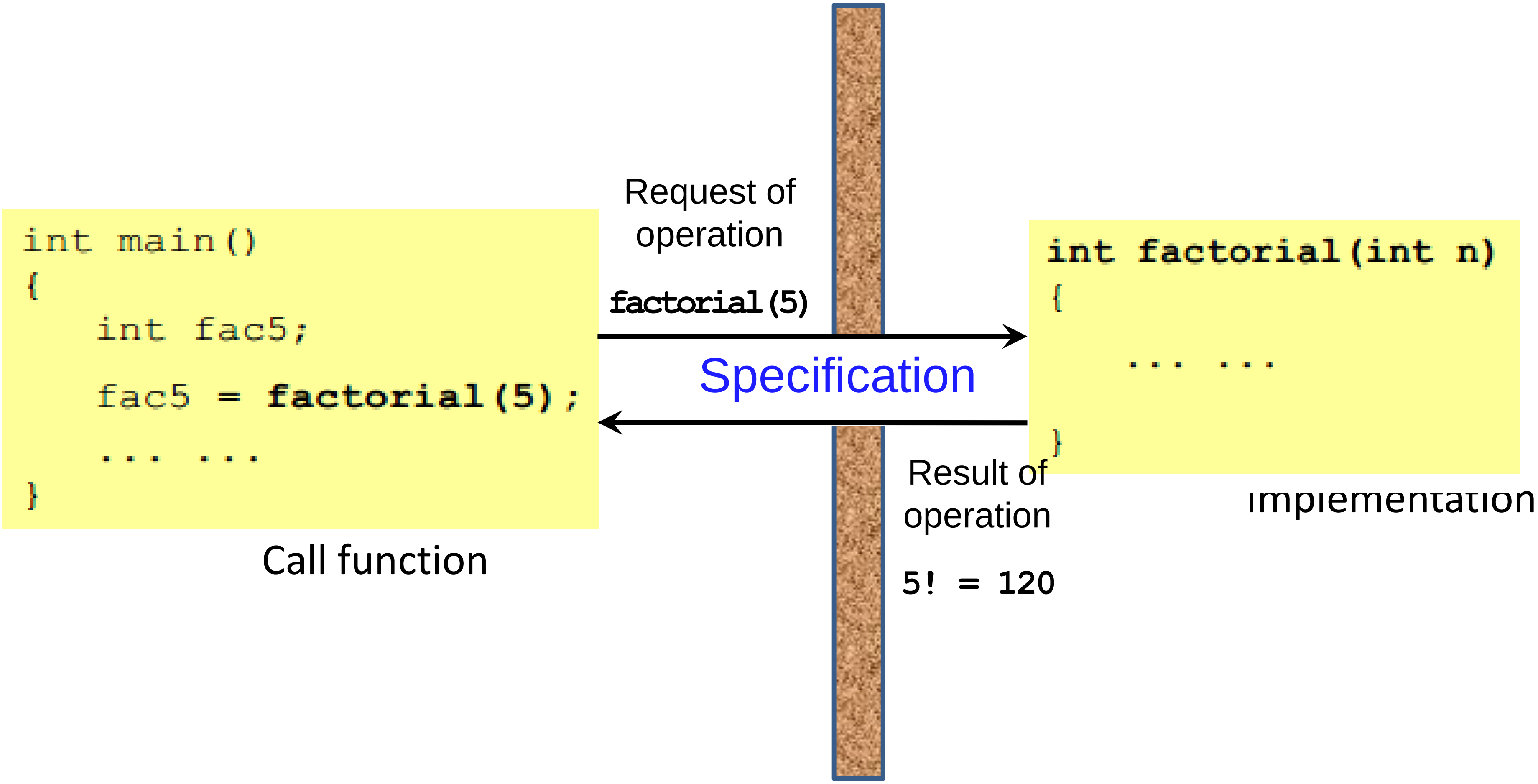
```
int factorial(int n)
{
    if (n == 0)
        return 1;

    return n * factorial(n-1);
}
```

Implementation 1

```
int factorial(int n)
{
    int i, result = 1;

    for (i = 2; i <= n; i++)
        result *= i;

    return result;
}
```

Implementation 2

Specification as Slit in the Wall



```
int main()
{
    int fac5;

    fac5 = factorial(5);

    ... ...
}
```

Call function

Request of operation

factorial(5)

Specification

Result of operation

5! = 120

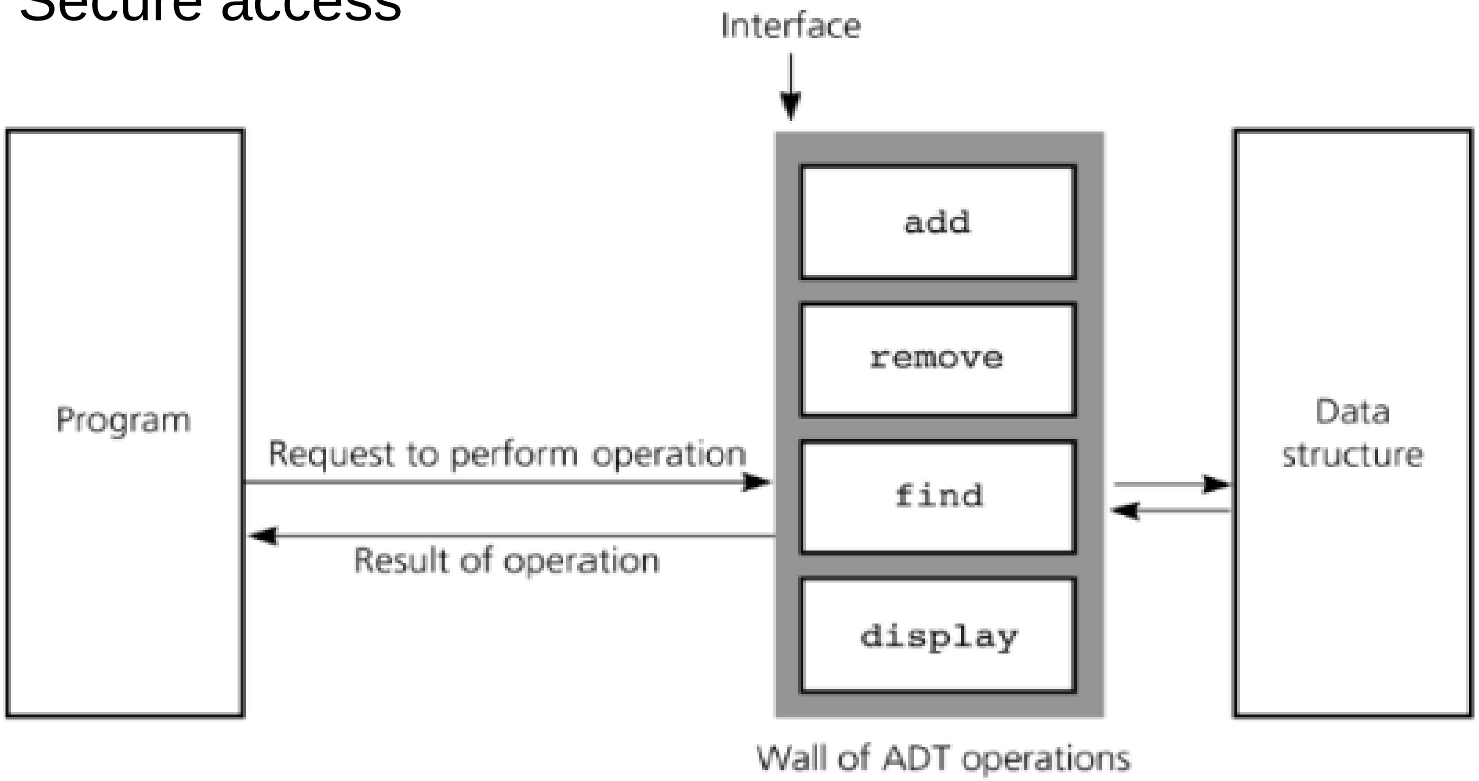
```
int factorial(int n)
{

    ... ...

}
```

Implementation

- User only depends on specification

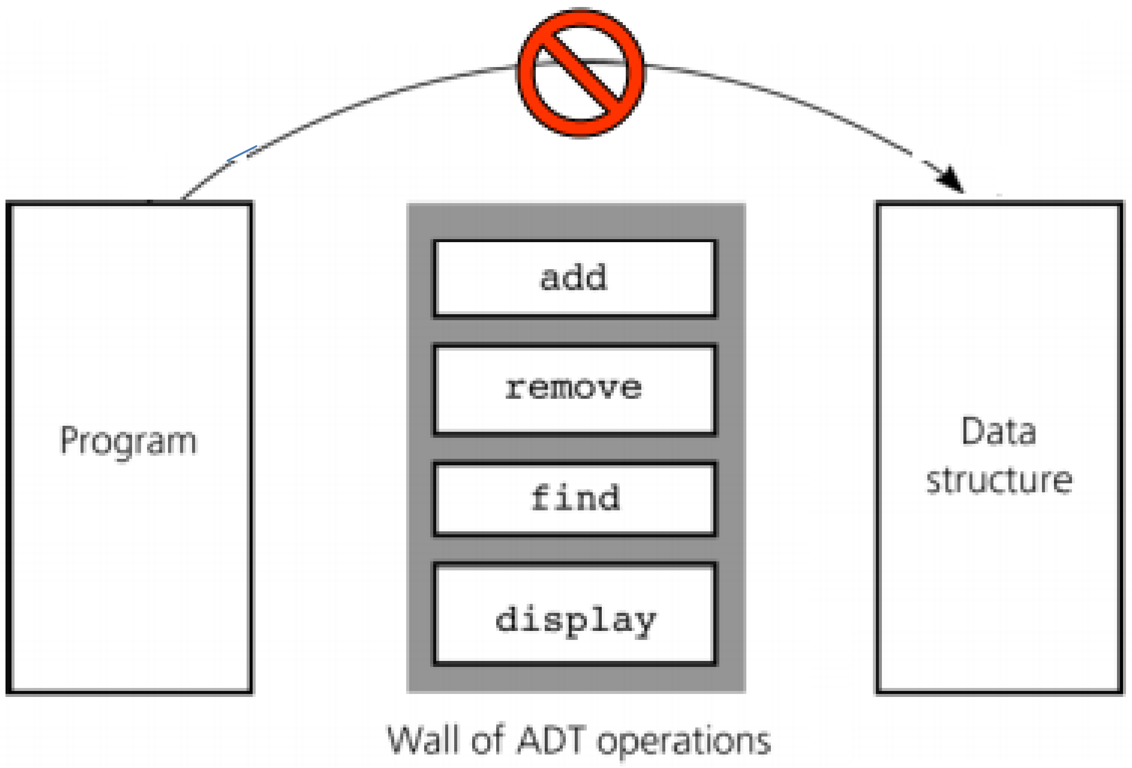
A wall of ADT operation

- ADT operation provides:
  - Interface to data structure
  - Secure access



Violating the Abstraction

- User programs should not
  - Use the underlying data structure directly
  - Depend on implementation details



When do we need ADT?

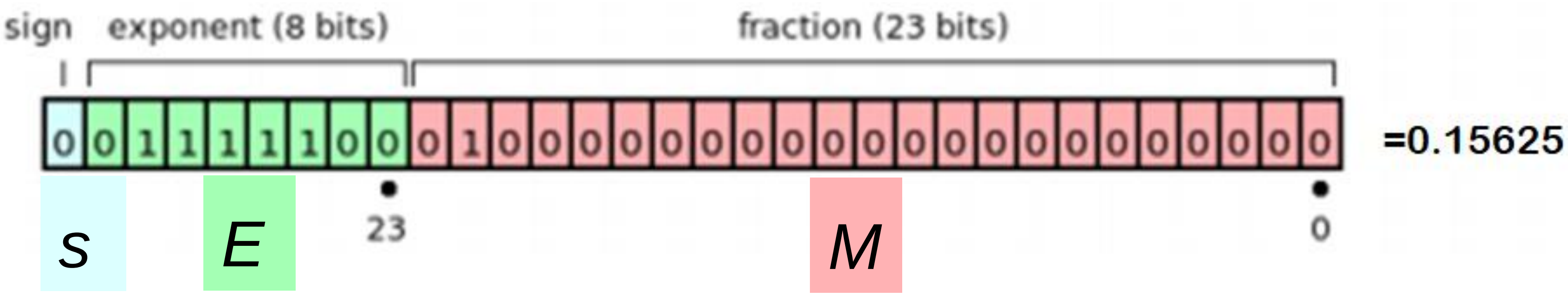
- When you need to operate on data that are not directly supported by the language
  - E.g. Complex Number, Module Information, Bank Account etc
- Simple Steps:
  1. Design an abstract data type
  2. Carefully specify all operations needed
     - Ignore/delay any implementation related issues
  3. Implement them

ADT Examples

- Primitive Type as ADT
- Complex Number ADT
- Sphere ADT

ADT 1: Primitive Data Type

- Predefined data types are examples of ADT
  - E.g. int, float, double, char, boolean
- Representation details are hidden to aid portability
  - E.g. float implementation (số thực dấy phảy động 4 bytes)

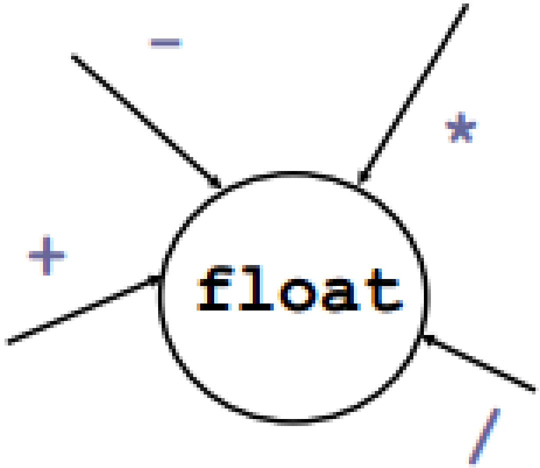


$$X = (s)\ M * R^{E} = (+)\ 1.01 * 2^{-3} = 0.15625$$

https://ttmn.mobi/floating-point-number-la-gi/

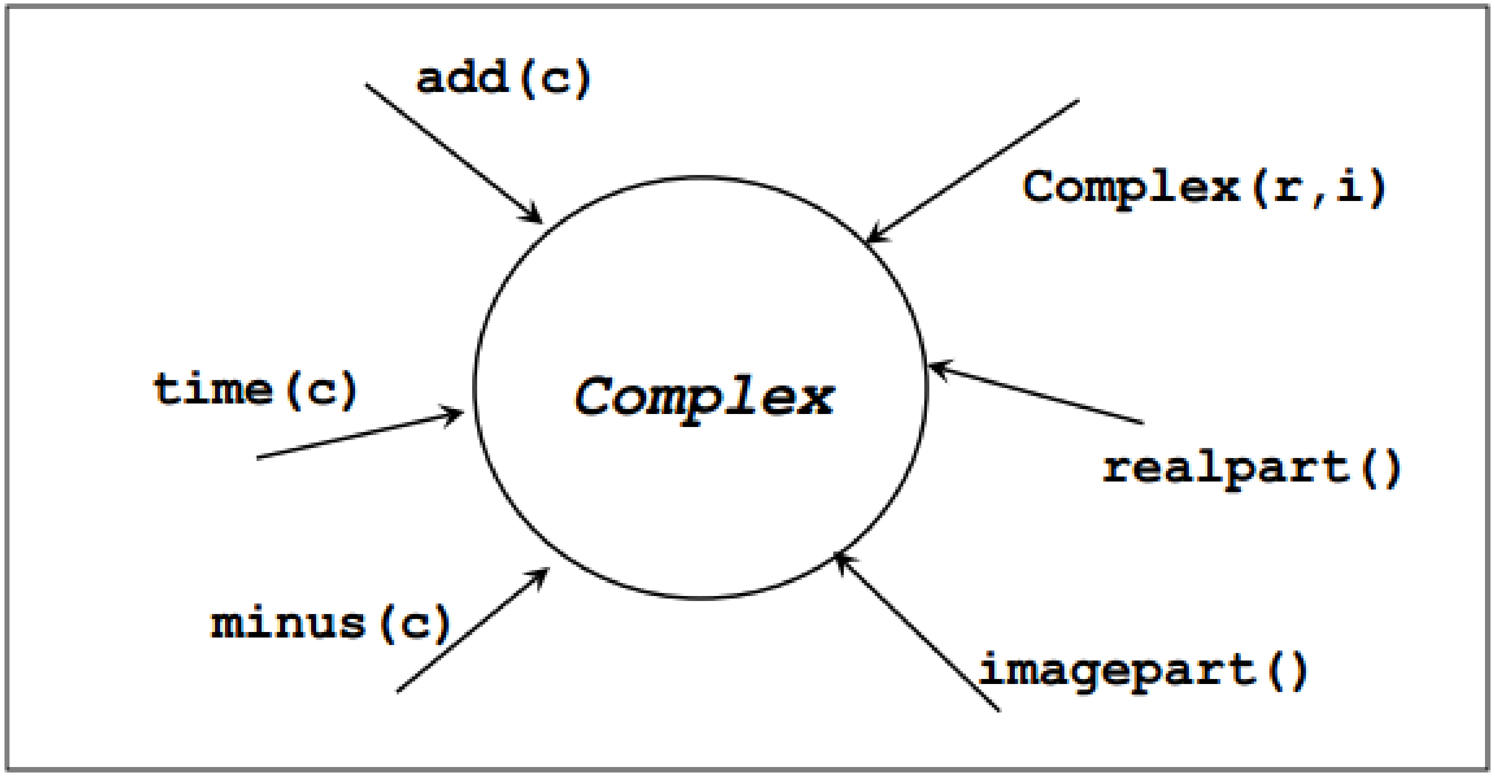
ADT 1: Primitive Data Type

- However, as a user, you don’t need to know the implementation to use float variable in statements



The `float` ADT

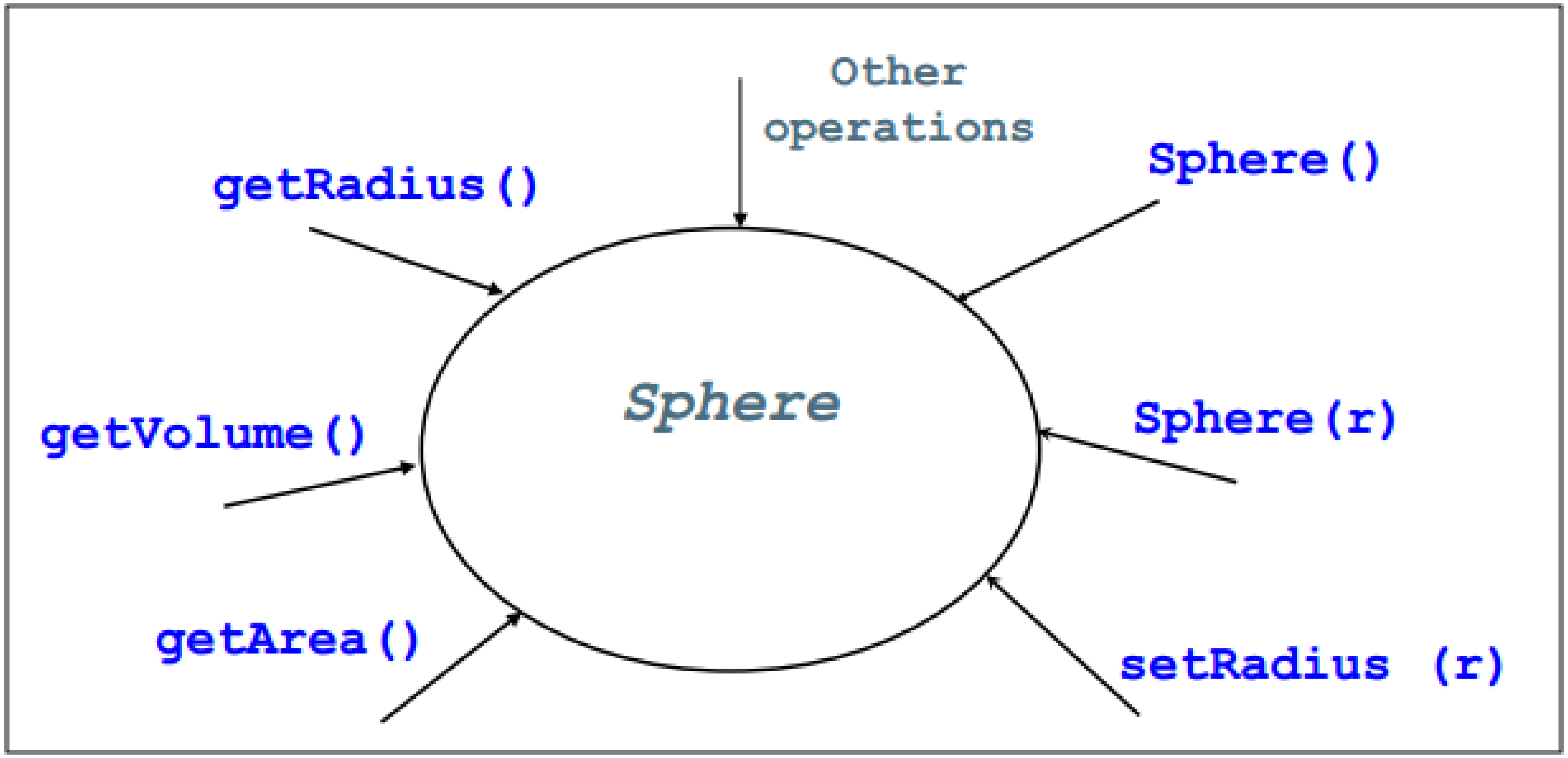
ADT 2: Complex Number



The `complex` ADT

`c`: Complex Number. `r,i`: float value.

ADT 3: Sphere



The sphere ADT

r: Radius, float value.

Conclusion: Benefits of ADT

- Hide the unnecessary details by building walls around the data and operations
  - So that changes inside will not affect other program components that use them
- Functionalities are less likely to change
- Localize rather than globalize changes
- Help manage software complexity
- Easier software maintenance