Copyright Warning

ViSLAB
High-Performance Visualization, Communication, Computing

Object Oriented Design (OOP)

Week 4-1

Three new elements in OOP

- OOP has three programming mechanisms
- Class (Encapsulation)
- Polymorphism
- Inheritance

These three are for...

- avoid using Global Variables
- make other resources re-usable (in addition to subroutine).

Class

- puts variables/subroutines in order (aggregate)
- hides variables/subroutines only necessary within an instance of a class (hide)
- you can instantiate many objects. (create)

Class...Aggregation

- simple variable and functions to read a character from a file

```
// the number of the currently accessed file.
int fileNO;

// open the specified file
void openFile(const string filename){...}

// close the currently opened file
void closeFile() {...}

// read one char from the currently opened file
char readChar() {...}
```

Aggregation

- simple variable and functions to read a character from a file

```
Class CharReader {
public:
    // the number of the currently accessed file.
    int fileNO;

    // open the specified file
    void open(const string filename);

    // close the currently opened file
    void close();

    // read one char from the currently opened file
    char read();
}
```

Aggregation

- Global Variables.....Member Variables
- Subroutines.....Member functions

Aggregation

- 100,000 - 1,000,000 LOC
- 1,000 - 10,000 S.R. (50-100 LOC/S.R)
- 100 - 1000 Classes (avg. 10 S.R./class)
- more?....namespace (C++), package (Java)

Aggregation

- you can have more simple and meaningful names.

```
Class CharReader {
public:
    // the number of the currently accessed file.
    int fileNO;

    // open the specified file
    void open(const string filename); // openFile

    // close the currently opened file
    void close();                          // closeFile

    // read one char from the currently opened file
    char read();                           // readChar
}
```

Aggregation

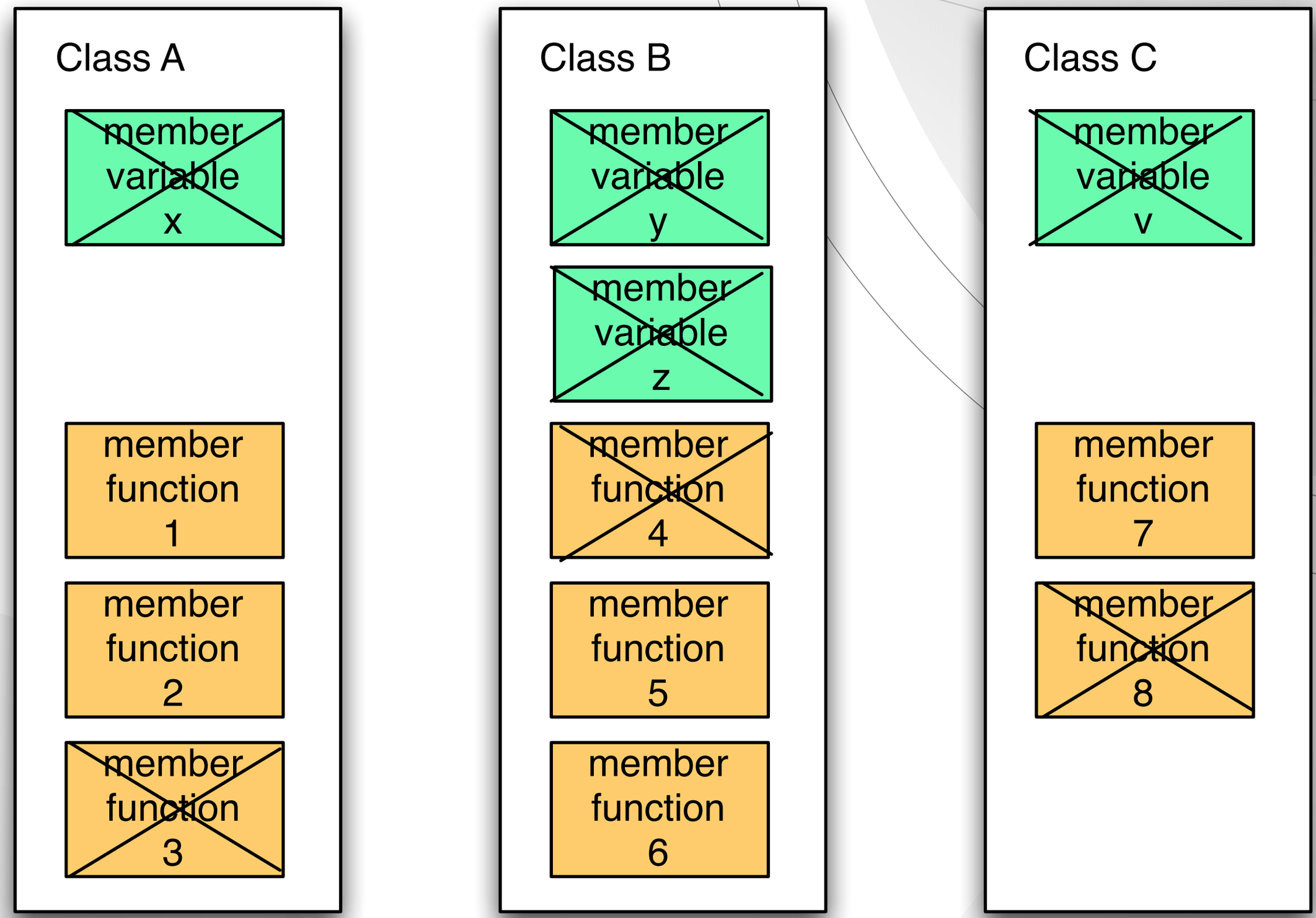
- group related subroutines and global variables into a class
- reduce the number of software components
- ease the naming of member functions
- ease of finding member functions

Class ... Hiding

```
   Class CharReader {
public:
    // the number of the currently accessed file.
    int fileNO;

    // open the specified file
    void open(const string filename); // openFile

    // close the currently opened file
    void close();                     // closeFile

    // read one char from the currently opened file
    char read();                      // readChar
};
```

Class ... Hiding

```
   Class CharReader {
private:
     // the number of the currently accessed file.
     int fileNO;

public:

     // open the specified file
     void open(const string filename); // openFile

     // close the currently opened file
     void close();                          // closeFile

     // read one char from the currently opened file
     char read();                           // readChar
}
```



Hiding

- hide subroutines (member functions) and global variables (member variables) from others
- Use Access Specifiers
  - reduce the use of Global Variables

Class ... Creating

- Gathering, and Hiding can be done in other non-OOP lang.
  - How?
- Creating “lots of” objects (from a Class)
  - bit difficult in non-OOP lang.

Class...Creating

- let's modify the following C code to handle multiple files....(ex. for computing file diffs)

```
// the number of the currently accessed file.
int fileNO;

// open the specified file
void openFile(const string filename){...}

// close the currently opened file
void closeFile() {...}

// read one char from the currently opened file
char readChar() {...}
```

Class...Creating

```
  // the number of the currently accessed file.
int fileNO;
```

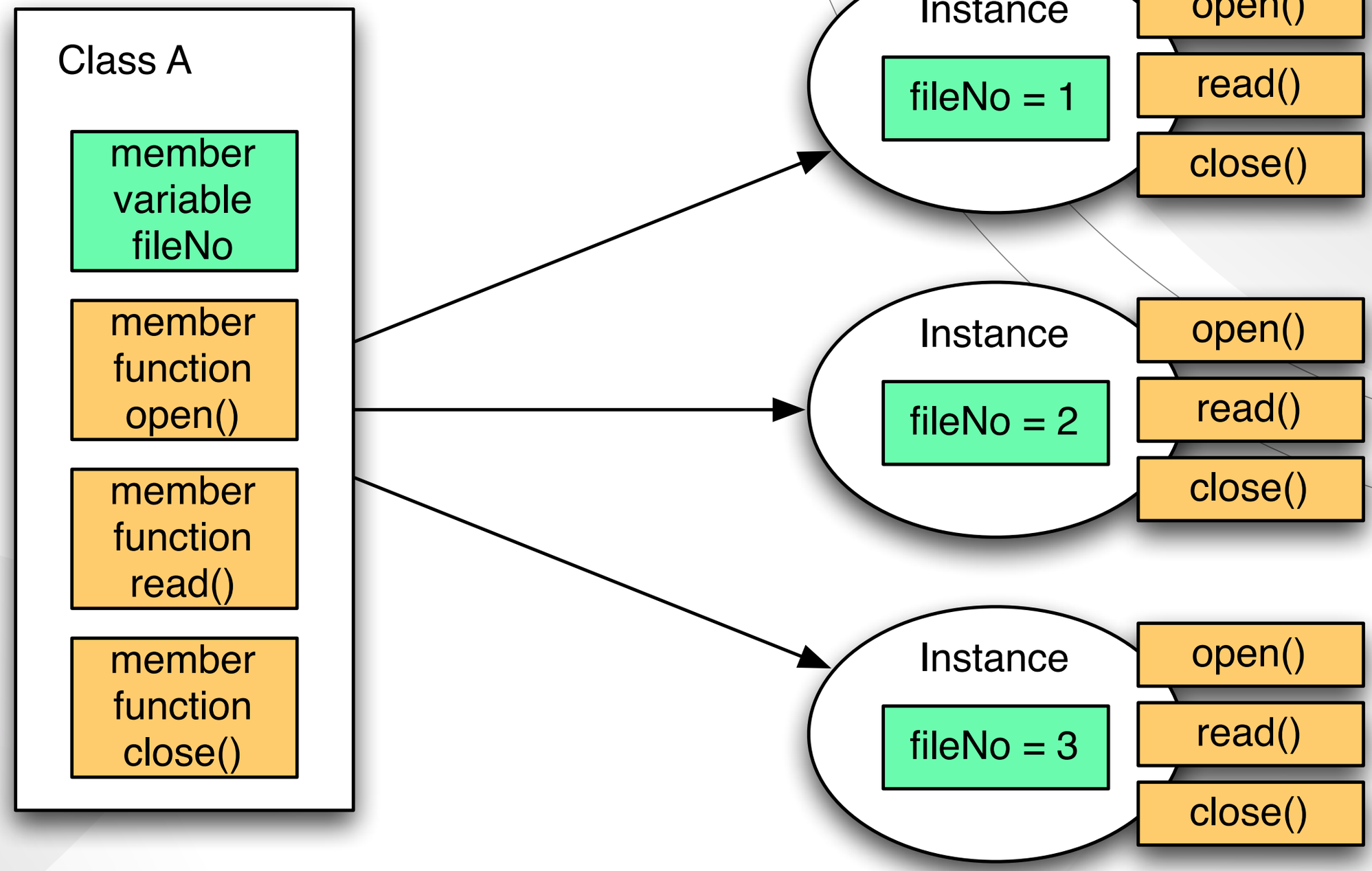
- have an array to store file numbers....

```
// open the specified file
void openFile(const string filename){...}

// close the currently opened file
void closeFile() {...}

// read one char from the currently opened file
char readChar() {...}
```

- add a file descriptor to the arg list of functions.

- Class == Kind
- Object (Instance) == concrete manifestation of the class

- Dog (Class)
- Jack (Instance/Object)



calling a function

- for Subroutine
  - `functionName(arguments);`
- for member function
  - `variable.memberFunction(arguments);`

- call (invoke) a function of an instance (object)
- similar to sending a message to an object

Class ... Creating

```
   Class CharReader {
private:
     // the number of the currently accessed file.
     int fileNO;

public:

     // open the specified file
     void open(const string filename); // openFile

     // close the currently opened file
     void close();                          // closeFile

     // read one char from the currently opened file
     char read();                           // readChar
}
```

Class ... Creating

```
  CharReader reader1 = new CharReader();
CharReader reader2 = new CharReader();

reader1.open("file1");
reader2.open("file2");

char c1, c2;
c1 = reader1.read();
c2 = reader2.read();

...
reader1.close();
reader2.close();
```

- "Creating" instances from a class
- allows the logic in a member function to be simple

Creating

- you can create many instances of a class at the runtime.
- even when multiple instances process similar pieces of information
  - the logic can be simple.

Member Variables (Instance Variables)

- Global variables in a Class.
- Hide them from others.
- Stay in the memory space until the instance is no longer needed.

	Local Var	Global Var	Member Var
from other S.R.	X	O	O
Limit access scope	only from the S.R. defines it	from anywhere	from any member functions in the class
life	created at the S.R call and deleted on exit	while application is running	while the instance is used
replication	only one at the time	only one	any numbers

Polymorphism

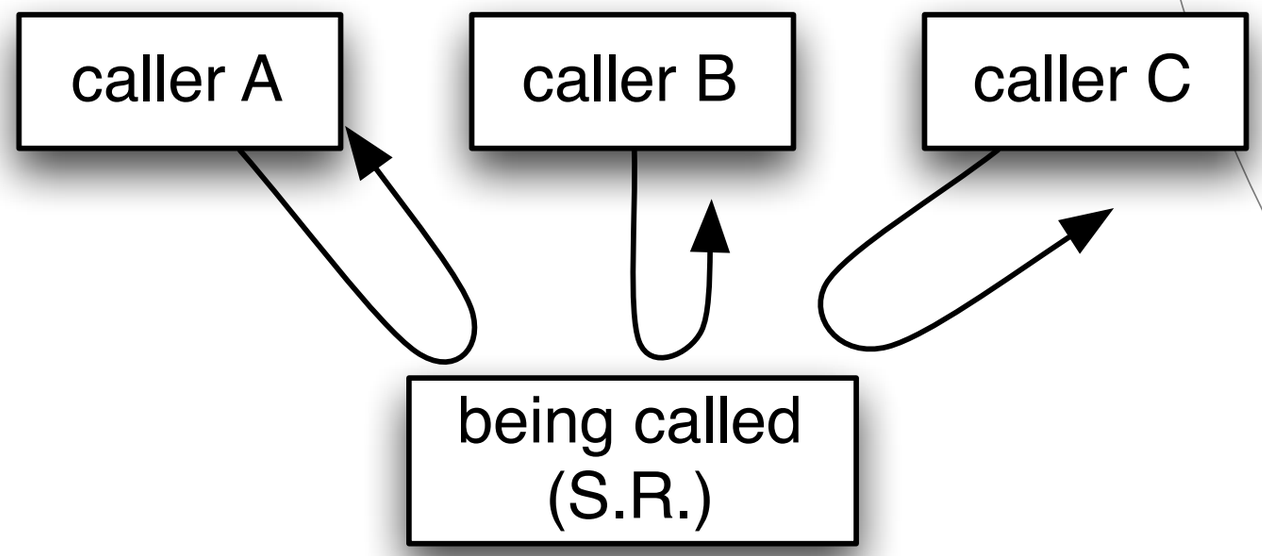
- a mechanism to not to worry about whom/ how you call a member function.

```
    ...
// define Brother
class Brother : public Animal {
public:
    Brother(string n): Animal(n) {}  // constructor (executed at the time of
instance creation)
    string cry() {    // make him cry.
        return "*sob*";
    }
};

// define Dog
class Dog : public Animal {
public:
    Dog(string n): Animal(n) {}

    string cry() {    // make him cry.
        return "bowwow";
    }
};

int main() {
    Brother john("John");
    Dog jack("Jack");

    // make them cry
    cout << john.cry() << endl;
    cout << jack.cry() << endl;

    return 0;
}
```

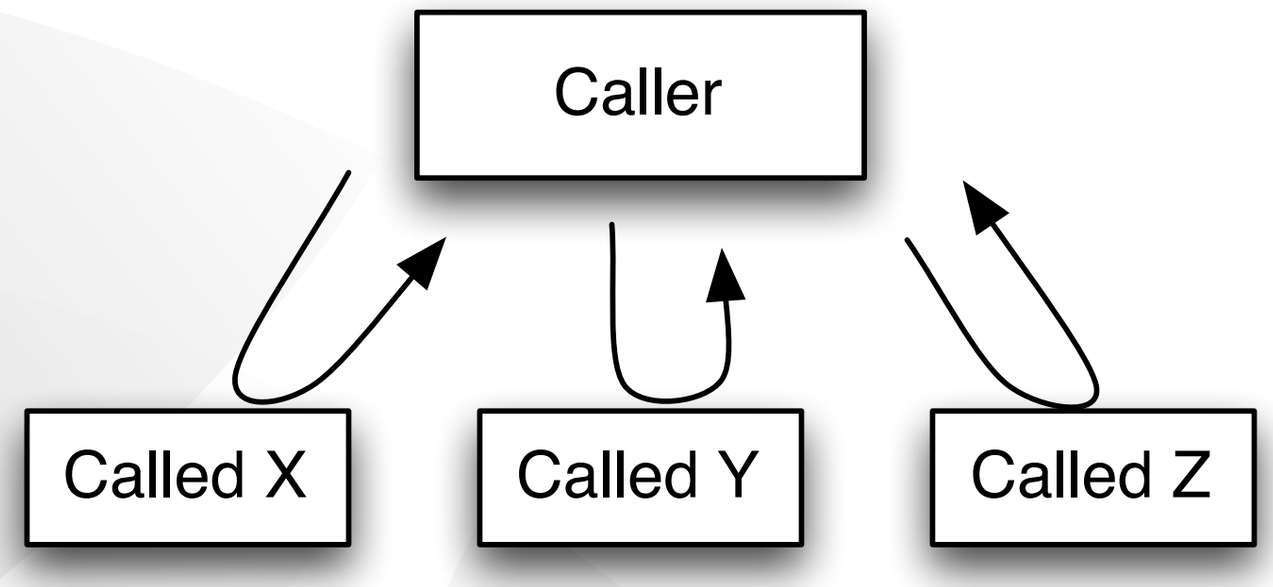
Polymorphism vs S.R.

- Subroutine
  - make commonly called function.

- Polymorphism
  - make a common way to call function.



Common Subroutine



Polymorphism

Polymorphism

```
   Class CharFromFileReader {
private:
     // the number of the currently accessed file.
     int fileNO;

public:

     // open the specified file
     void open(const string filename); // openFile

     // close the currently opened file
     void close();                          // closeFile

     // read one char from the currently opened file
     char read();                           // readChar
}
```

Polymorphism

```
Class CharFromNetworkReader {
public:
    // open the specified network
    void open();

    // close the currently opened network
    void close();

    // read one char from the currently opened network
    char read();
}
```

Polymorphism

```
Class CharReader {
public:
    // open the specified network
    void open();

    // close the currently opened network
    void close();

    // read one char from the currently opened network
    char read();
}
```

Polymorphism

```
class CharFromFileReader : CharReader {
...
}
class CharFromNetworkReader : CharReader{
...
```

```
int countChar(CharReader reader) {
    int count = 0;
    while (true) {
        char c = reader.read();
        count++;
    }
    return count;
}
```

Inheritance

- make a common class
  - group commonly used variables/functions.
- use them in derived class
  - derived ....inheritance