경희대학교
Kyung Hee University

Advanced Object Oriented Programming

Classes

Seokhee Jeon

Department of Computer Engineering
Kyung Hee University
jeon@khu.ac.kr

Private and Public

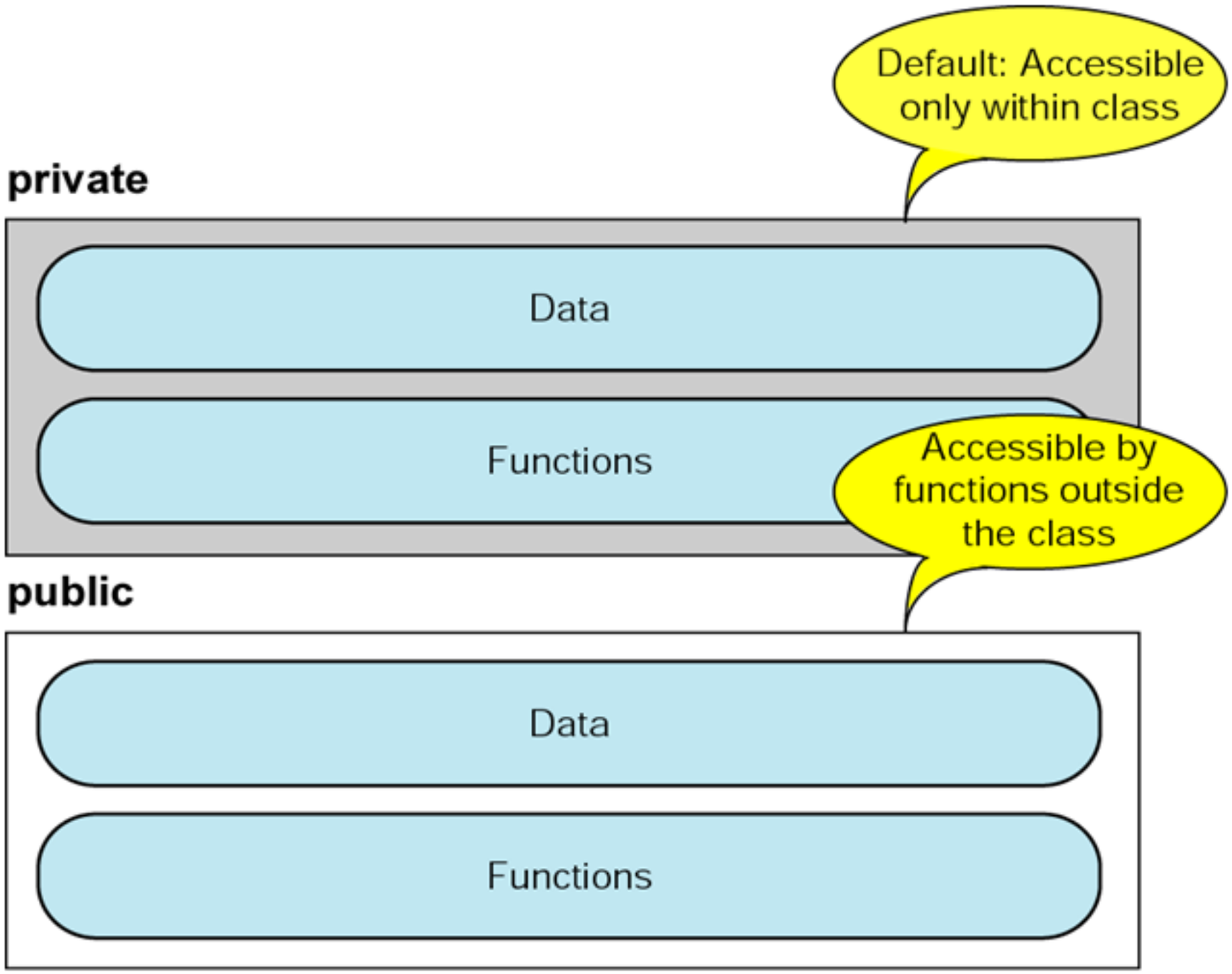
Hidden data and actions
(does not shown to outside)

Private

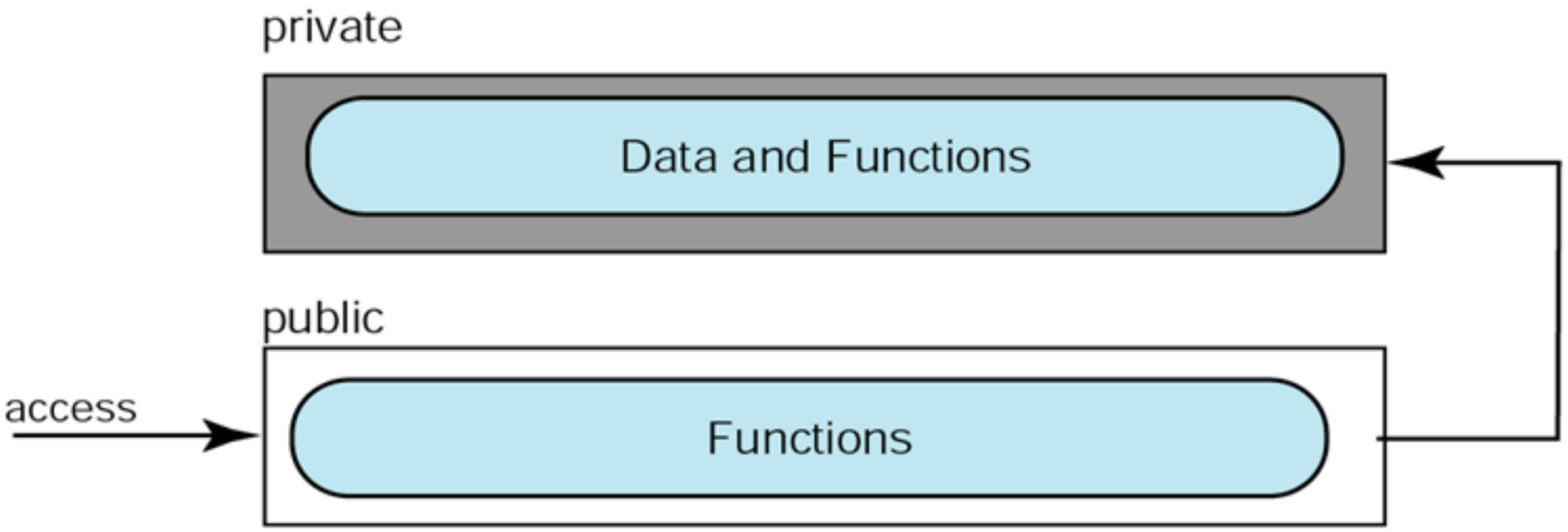
Simplified actions to access hidden data and actions
(only shown to outside, sometime data also)

Public

Class access classifiers



Class data access



Structure of class

```
Class Fraction
{
    private:
        int numerator;
        int denominator;

    public:
        void store (int number, int denom);
        void print () const;
}
```


5

Structure of class

Class declaration

Class Fraction — Name of Class

Opening brace

{

private: Private data and function ***declaration***

```
int numerator;
int denominator;
```

public: Private data and function ***declaration***

```
void store (int number, int denom);
void print () const;
```

Closing brace

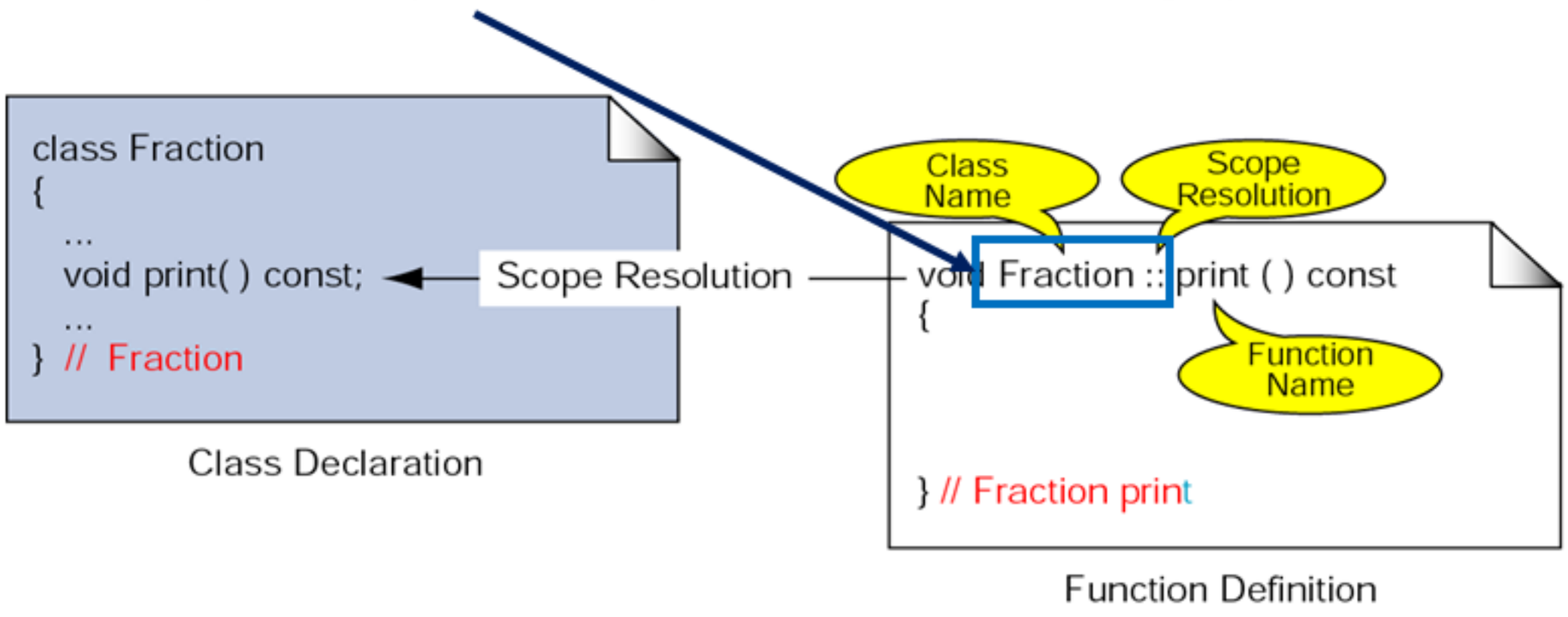
}

Definition of class

- **Function definitions follows after class declaration.**

Similar to function declaration before main(),
and function definition after main().

However, function of Class is defined with Class name as prefix.



Complete class *Fraction* definition

```
class Fraction
{
    private:
        int numerator;
        int denominator;
    public:
        void store (int number, int denom);
        void print () const;
}

void Fraction::store (int numer, int denom)
{
    numerator = number;
    denominator = denom;
    return;
}

void Fraction::print () const
{
    cout << numerator << "/" << denominator;
    return;
}
```

Access private variables freely.

Class instance

- Class is just a type, more advanced type.

int

float *class Fraction*

char

Class instance

- You make a real variable using simple type like this

int a;

float b;

char c;

class Fraction?

Class instance

- Same way!!! Class is just a type!!!

int a

float b

char c

class Fraction f;

Objects of the class

```
class Fraction
{
 private:
   int numerator;

   int denominator;
public:
   void store (int numer,
               int denom);

   void print () const;
} ; // class Fraction
```

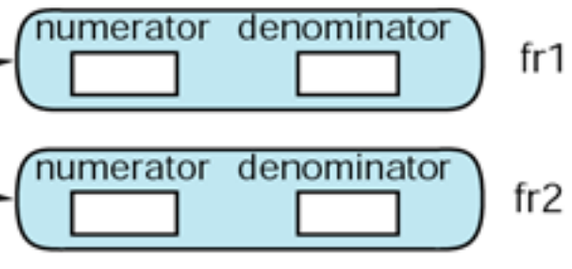
(a) Class Declaration

```
void Fraction :: store
     (int  numer,
      int  denom)
{
     numerator = numer;
```

```
void Fraction :: print() const
{
     cout << numerator
          << "/"
          << denominator;
     return;
} // Fraction print
```

(b) Function Definitions

```
int main ()
{
   Fraction  fr1;
   Fraction  fr2;
   …
} // main
```



(c) Class Instantiation

Invocation of class function

- objectID.memberID

```
Fraction fr;
fr.store(5, 19)
fr.print();
```

OR *to avoid confusion when very many classes are used*

```
fr.Fraction::print();
fr.Fraction::store(5, 19);
```

Sample code

User created header file

p10-04.h

```
class Fraction
{
    private:
      int numerator;
      int denominator;
    public:
      void store (int numer, int denom);
      void print () const;
} ; // class Fraction

void Fraction :: store (int numer, int denom)
{
    numerator   = numer;
    denominator = denom;
    return;
}// Fraction store

void Fraction :: print () const
{
    cout << numerator << "/" << denominator;
    return;
}// Fraction print

// ============ End Fraction Functions
```

```
#include <iostream>
using namespace std;

#include "p10-04.h"

void getData (int& numer, int& denom);

int main ()
{
        cout << "This program creates a fraction₩n₩n";
        int numer;
        int denom;
        getData  (numer,  denom);

        Fraction fr;
        fr.store (numer,  denom);

        cout << "₩nYour  fraction contains: ";
        fr.print ();

        cout << "₩n₩nThank you for using fractions₩n";
        return 0 ;
} // main

void getData (int& numer, int& denom)
{
        cout << "Please enter the numerator:   ";
        cin  >>  numer;

        cout << "Please enter the denominator: ";
        cin  >>  denom;
        return;
}       // getData

/*      Results
*/
```

Sample code

```
class Fraction
{
      private:
         int numerator;
         int denominator;
      public:
```

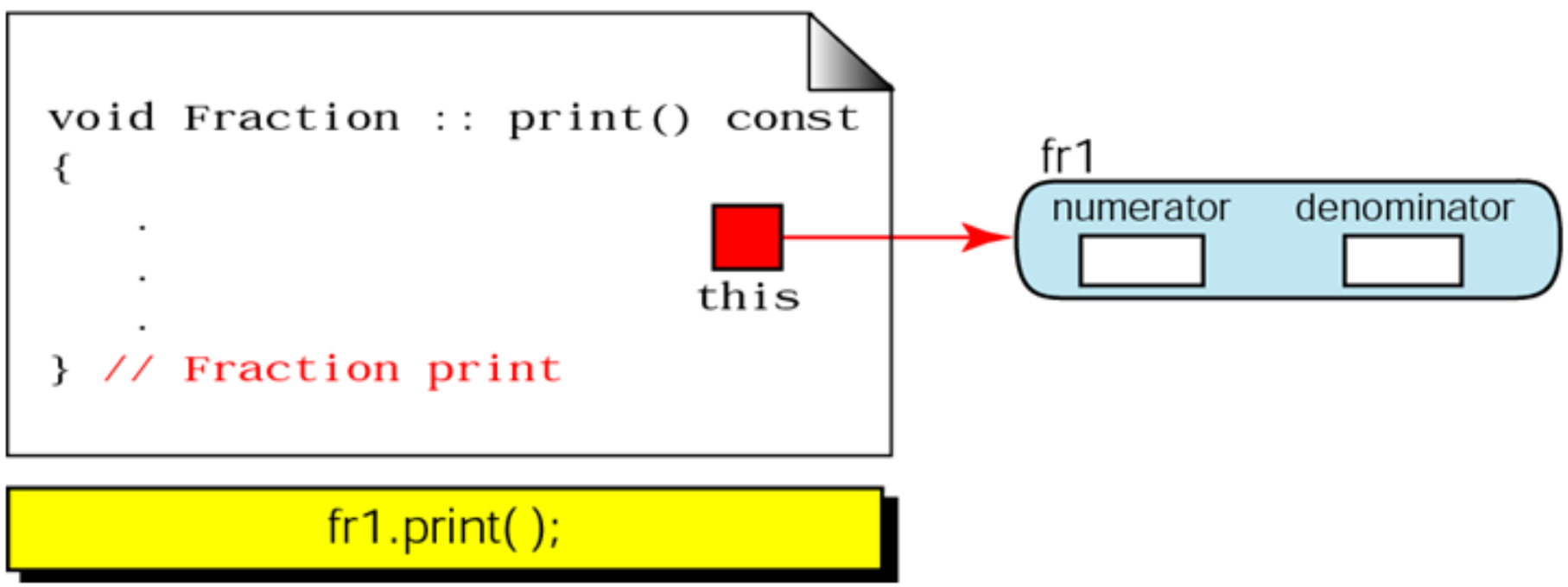
```
This program creates a fraction

Please enter the numerator:   19
Please enter the denominator: 51

Your fraction contains: 19/51

Thank you for using fractions
```

```
      return;
   }// Fraction print

   // ============ End Fraction Functions
```

```
#include <iostream>
using namespace std;

#include "p10-04.h"

void getData (int& numer, int& denom);

int main ()
{
                cout << "This program creates a fraction₩n₩n";

                int numer;
                int denom;
                getData  (numer,  denom);

                Fraction fr;
                fr.store (numer,  denom);

                cout << "₩nYour  fraction contains: ";
                fr.print ();

                cout << "₩n₩nThank you for using fractions₩n";
                return 0 ;
} // main

void getData (int& numer, int& denom)
{
                cout << "Please enter the numerator:   ";
                cin  >>  numer;

                cout << "Please enter the denominator: ";
                cin  >>  denom;
                return;
}               // getData

/*              Results
*/
```

this pointer

- ***this* pointer**
  - Constant pointer
    - contains the address of the invoking objects,
      so that it can refer to data and functions in the class
    - can not be changed
  - Most of the time it is used implicitly (hidden)
  - Can us explicitly like : *this.numerator, *this.store(2,5)



this pointer

- ***this* pointer? Why?**
  - **C++'s implementation principles**

    "C++ keeps only one copy of each member function and the data members are allocated memory for all of their instances. This kind of various instances of data are maintained use this pointer."

Three types of functions in classes

- Manager functions
  - Create, copy or destroy objects
- Mutator functions
- Accessor functions

Constructor function

Note:

A constructor is called every time an object is instantiated.

Constructor function

Note:

The name of a constructor is the same as the name of the class and it may not have a return type.

Constructor function

FOR EXAMPLE

```
Fraction (int numem, int denom);  // constructor declaration in the class
```

```
Fraction::Fraction(int numem, int denom)  // constructor definition
{
        …
}
```

Constructor function

Note:

Multiple constructors with same name, but different input/out type can be provided

Constructor function

```
class Fraction
{
      private:
        int numerator;
        int denominator;
      public:
        Fraction ( );
        Fraction (int numer);
        Fraction (int numer, int denom);
        void store (int numer, int
      denom);
        void print () const;
};    // Fraction
```

```
Fraction :: Fraction ()
{
                numerator   = 0;
                denominator = 1;
}// constructor
```

```
Fraction :: Fraction (int numen)
{
                numerator   = numen;
                denominator = 1;
}// Fraction constructor
```

```
Fraction :: Fraction (int numen, int denom)
{
                numerator   = numen;
                denominator = denom;
}               // constructor
```

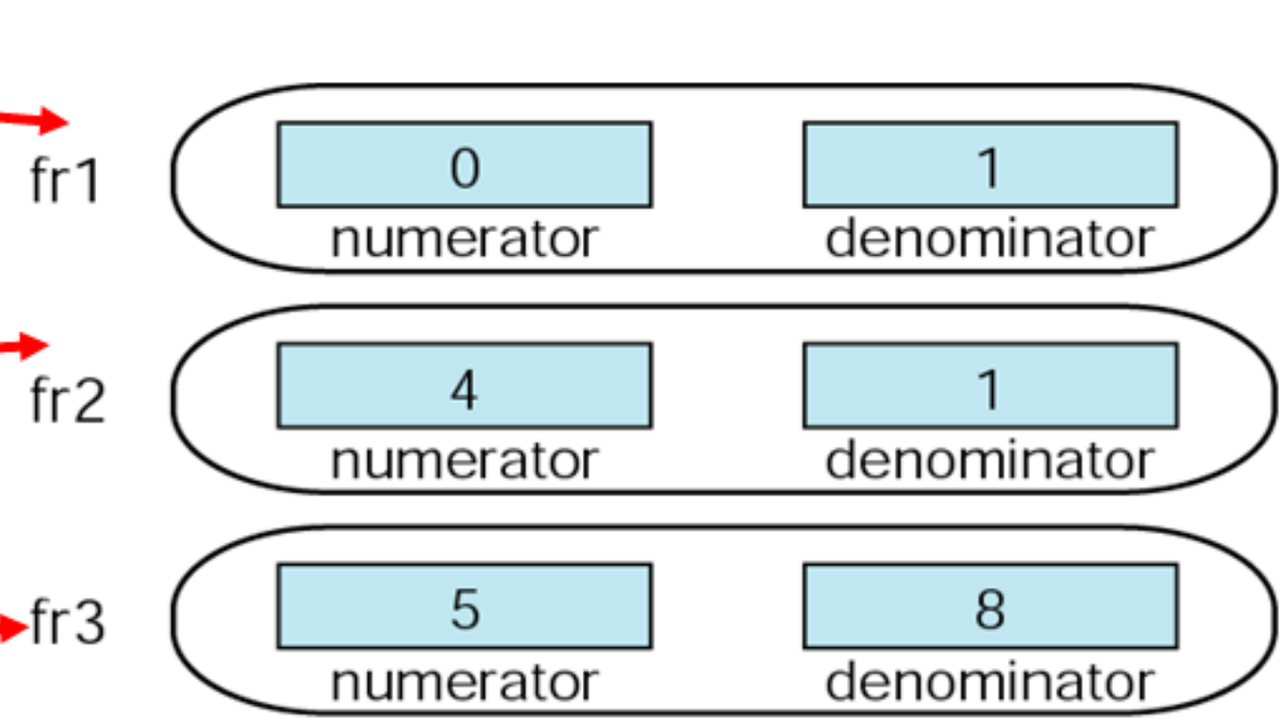
Sample code

```
#include <iostream>
using namespace std;

#include "p10-06.h"              // Fraction class
#include "p10-03.h"              // print function

int main ()
{
      Fraction fr1;
      cout << "fr1 contains: ";
      fr1.print ();
      cout << endl;

      Fraction fr2 (4);
      cout << "fr2 contains: ";
      fr2.print ();
      cout << endl;

      Fraction fr3 (5, 8);
      cout << "fr3 contains: ";
      fr3.print ();
      cout << endl;
      return 0;
} // main
```

```
fr1 contains: 0/1
fr2 contains: 4/1
fr3 contains: 5/8
```

Sample code

```
#include <iostream>
using namespace std;

#include "p10-06.h"          // Fraction class
#include "p10-03.h"          // print function

int main ()
{
    Fraction fr1;
    cout << "fr1 contains: ";
    fr1.print ();
    cout << endl;

    Fraction fr2 (4);
    cout << "fr2 contains: ";
    fr2.print ();
    cout << endl;

    Fraction fr3 (5, 8);
    cout << "fr3 contains: ";
    fr3.print ();
    cout << endl;
    return 0;
} // main
```



Constructor required

Note:

A class must have at least one constructor, either defined by the program or by the compiler.

Default constructor

Note:

The default constructor is called whenever an object is crated without passing any arguments.

The default constructor is a constructor that can be called without any arguments.

System-defined Default Constructor

- If the programmer doesn't provide any constructors, the compiler provides a default constructor
- The provided constructor is very primitive and does nothing

```
class Fun
{
        …                       // No constructor
} // Fun
int main()
{
        Fun fun;                // No errors
        …
} // main
```

Default constructor

Note:

If we define any type of constructor, we must also define a default constructor if it is needed.

Default constructor

```
#include <iostream>
using namespace std;

class Funny
{
      private:
        int num;
      public:
        Funny (int x);   // A constructor
};    // Funny

Funny :: Funny (int x)
{
      num = x;
}     // Initializtion Constructor

int main ()
{
      Funny funny1 (10);            // Constructor called
      Funny funny2;       // Error: no default constructor
      return 0 ;
} // main
```

Error: function call '[Funny].Funny()' does not match
...
temptest.cpp line 23 Funny funny2;

Constructor with default arguments

- Programmer defined constructor can also be used as a default constructor if all of the arguments have default values

Fraction :: Fraction (*int numen = 0, int denom = 1*)

```
{
    numerator = numen;
    denominator = denom;
} // Fraction default constructor
```

Copy constructor

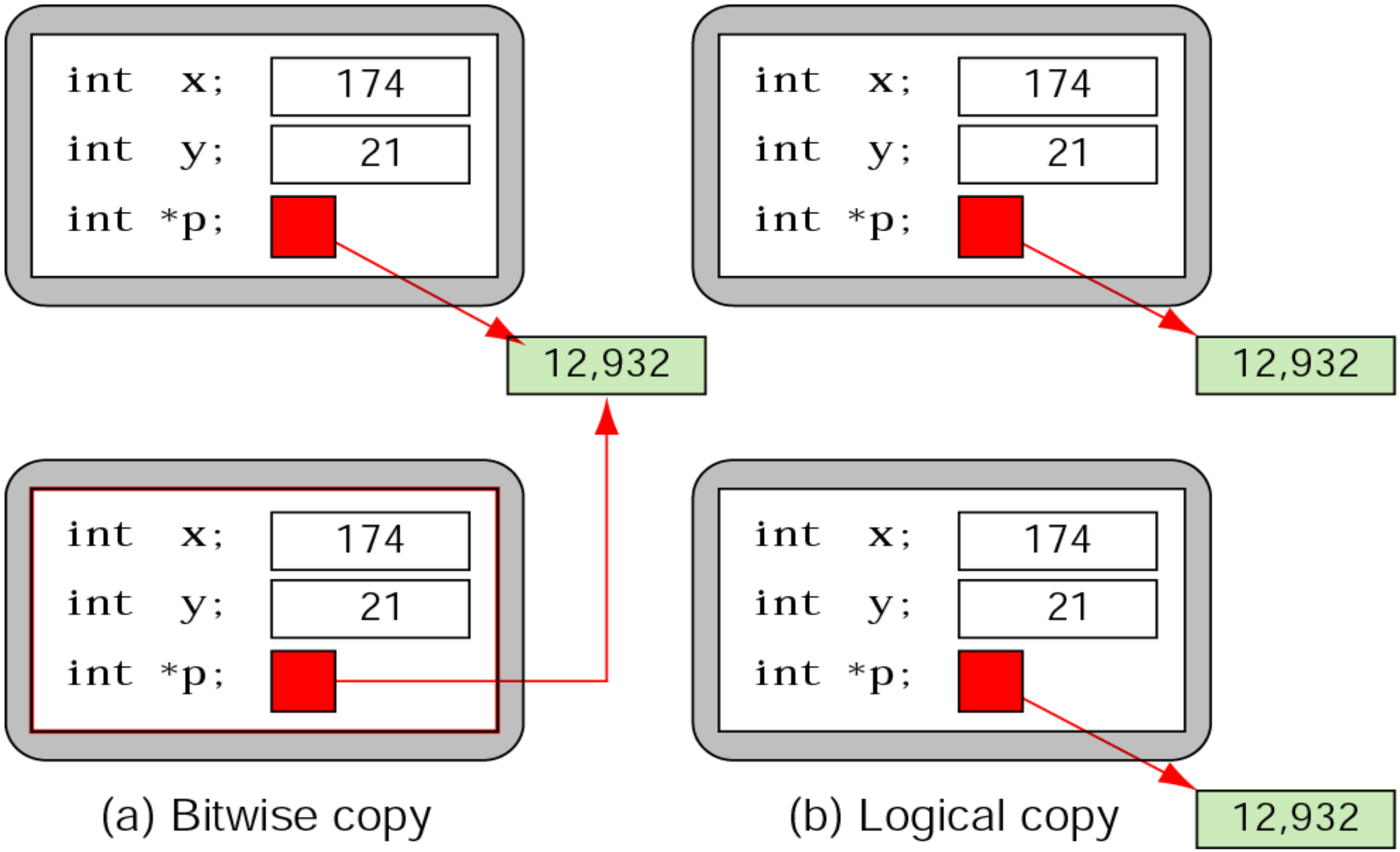
- called whenever a copy of an existing instance needs to be created

```
Fraction fr1;

Fraction fr2(fr1);
```

- Defined by using *call by reference*

```
Fraction (const Fraction& fr);// Declaration

Fraction :: Fraction (const Fraction& fr) // Definition
{
    numerator = fr.numerator;
    denominator = fr.denominator;
} // Fraction copy constructor
```

Bitwise and logical copies



(a) Bitwise copy

(b) Logical copy

Cases copy constructors are called

- Instantiated Objects
  - An object is instantiated using another object as a parameter
- Passed Objects
  - An object is passed to a function by value creating a local copy
- Returned Objects
  - An object is returned from a function

Pass objects by reference

Recommendation

Always pass objects to a function by reference; when the object cannot be changed, pass it as a constant.

```
int fun (Fraction& fr);

int fun (const Fraction& fr);
```

Anonymous objects

- **Return Class Object as a function output**
  - Called ***anonymous object***

```
Fraction func (int x )
{
        ...
        return Fraction (2, x);
        // explicit call to constructor, to return class object
}
```

Destructor

Note:

Destructor come into play when an object dies

Destructor

- **Destructor structure**
  - The name of the destructor is the name of the class preceded by a tilde(~)
  - Can not have return value

```
Fraction::~Fraction()
{
        // de-allocate memory
        // close files
        // others
}
```

Destructor

Note:

A class can have one, and only one, destructor.

Some terms on C++

- **Just remember!!!**

  - ***Mutator*** functions
    - Functions that actually change the state (or variables) of a (class) object

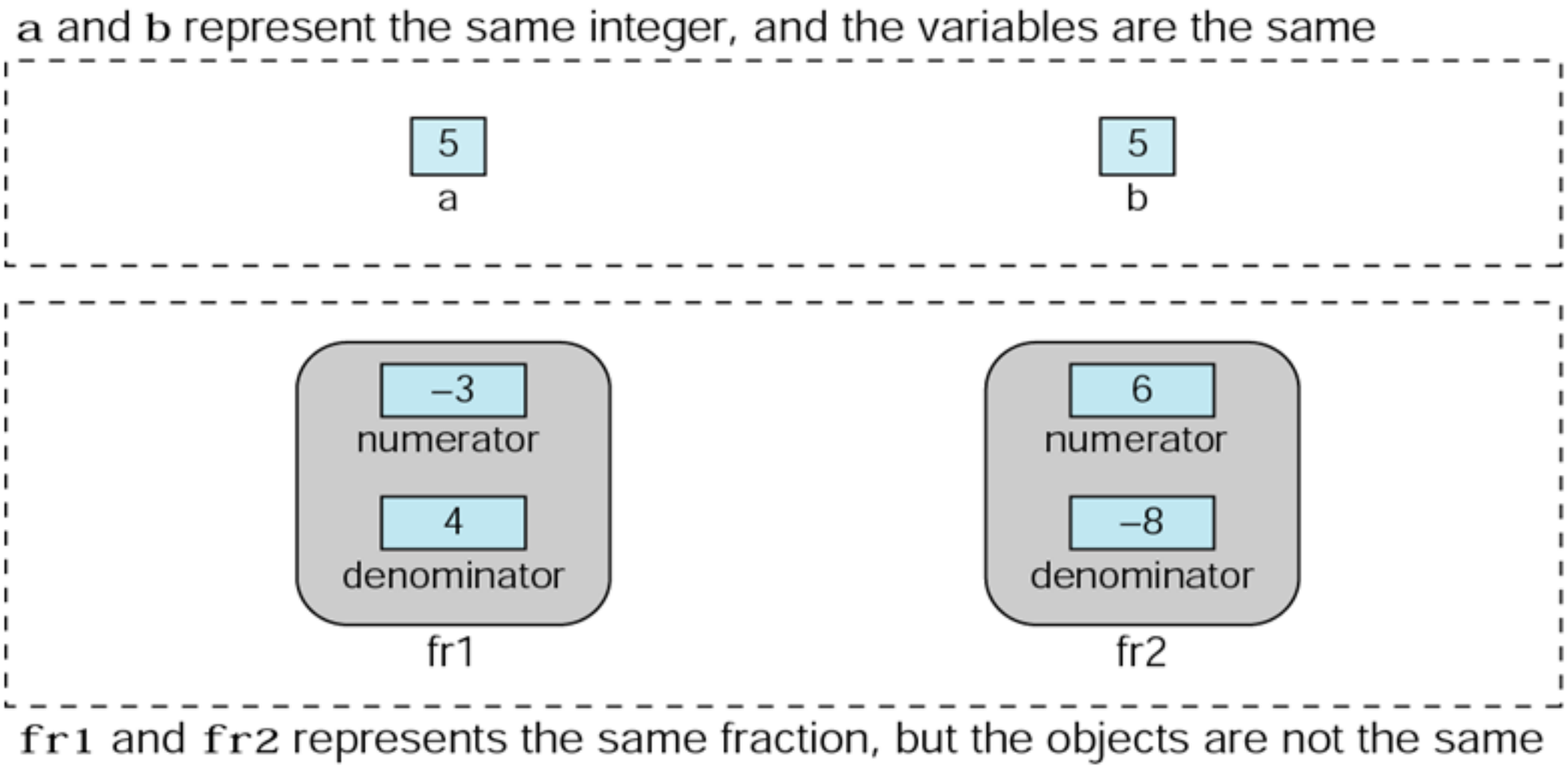
      fr1.store (4,7) // changes its private variables to 4 and 7

  - ***Accessor*** functions
    - Functions that does not change the state (or variables) of a (class) object

      fr1.print () // just print its private variables

Class invariants

"Two or more instances of a class that represent the same value must have the same data members"

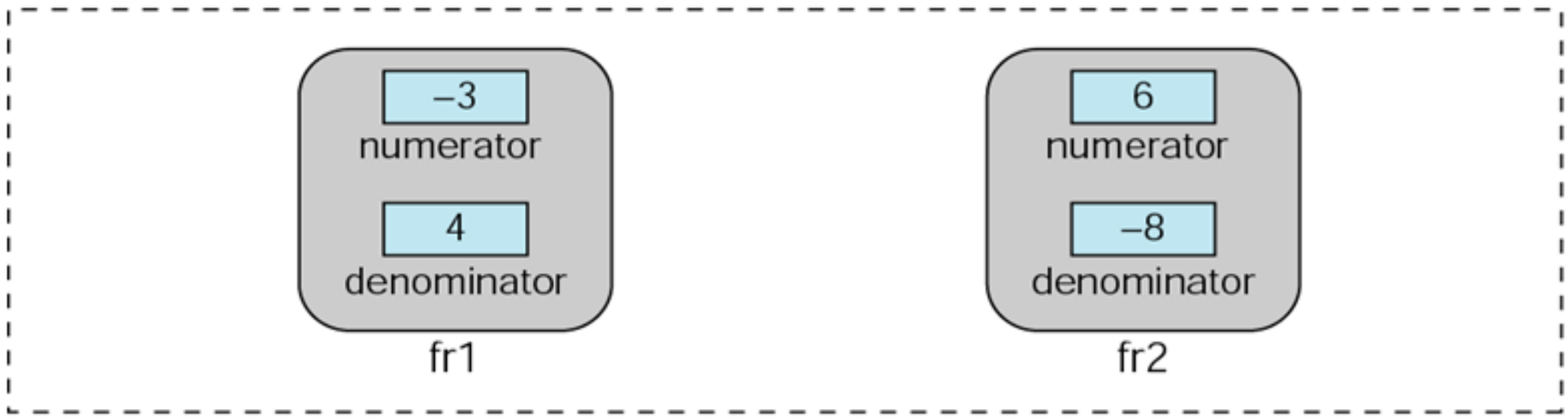


Class invariants

- ***Normalization !!!***

$$\gcd(x, y) = \begin{cases} x & \text{if } y = 0 \\ \gcd(y, x \bmod y) & \text{otherwise} \end{cases}$$

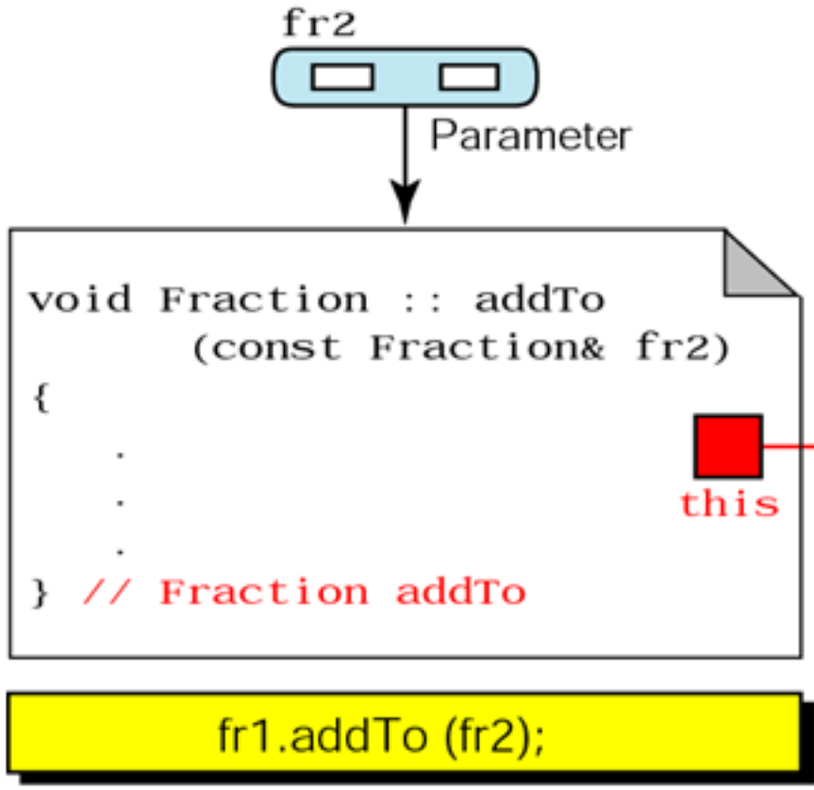
Greatest Common Divisor applied to make them same value as (-3/4).



`fr1` and `fr2` represents the same fraction, but the objects are not the same

Binary class function

- **Class object can be used as an input parameter for function call**




```
void Fraction :: addTo (const Fraction& fr2)
{
        numerator =
            (numerator     * fr2.denominator)
          + (fr2.numerator * denominator);
        denominator *= fr2.denominator;
        *this = Fraction (numerator, denominator);
        return;
}       // Fraction addTo
```

Friend function

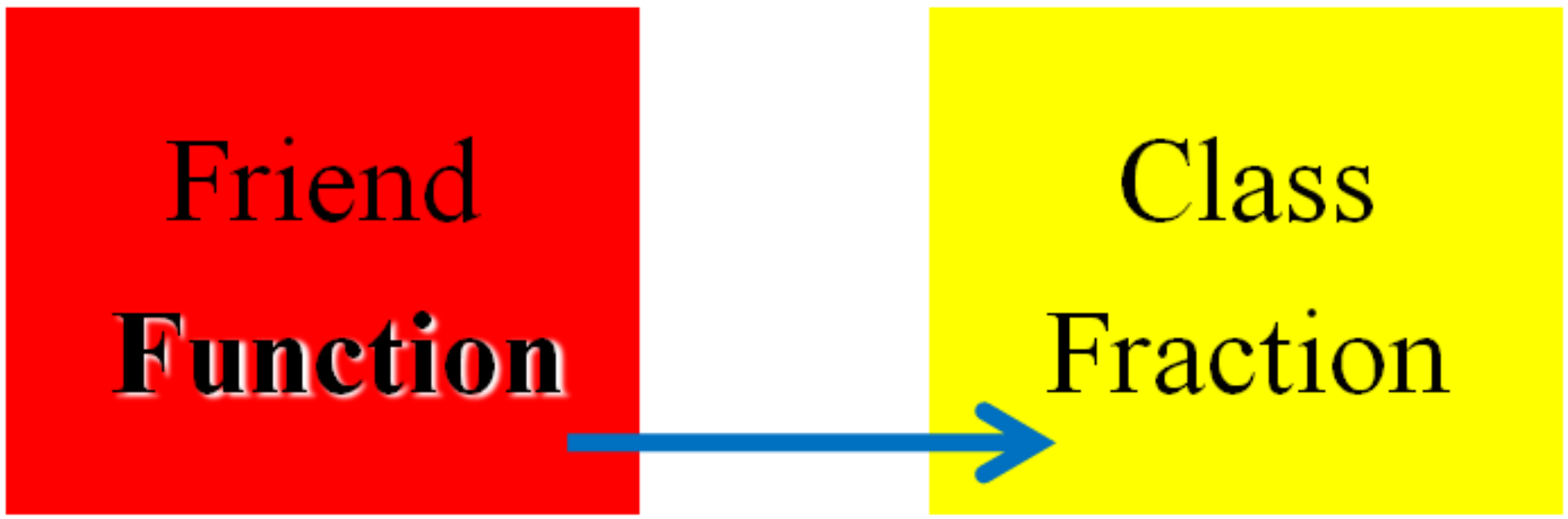
Location: Outside of the class
(not a member of class)

Friend
Function

Class
Fraction

Friend function

Location: Outside of the class
(not a member of class)



Privilege: can access private members

Friend function

Note:

Friend functions are not members of a class, but are associated with it.

Friend function

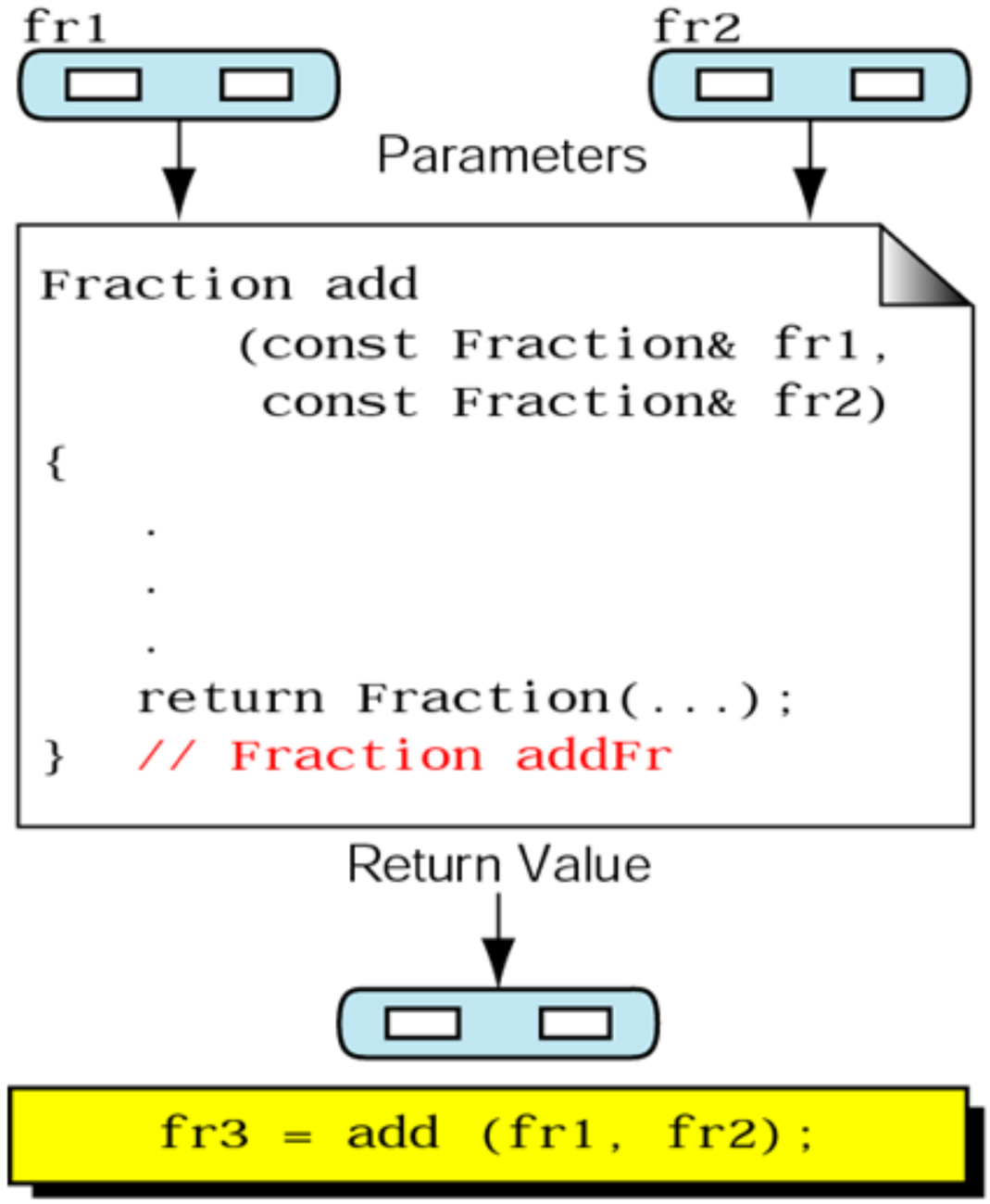
AddFraction Function Definition

```
Fraction add (const Fraction& fr1, const Fraction& fr2)
{
        int numen = (fr1.numerator   * fr2.denominator)
                + (fr2.numerator   * fr1.denominator);
        int denom =  fr1.denominator * fr2.denominator;
        return Fraction (numen, denom);
}       // friend Fraction addFr
```

Modification for Class Declaration

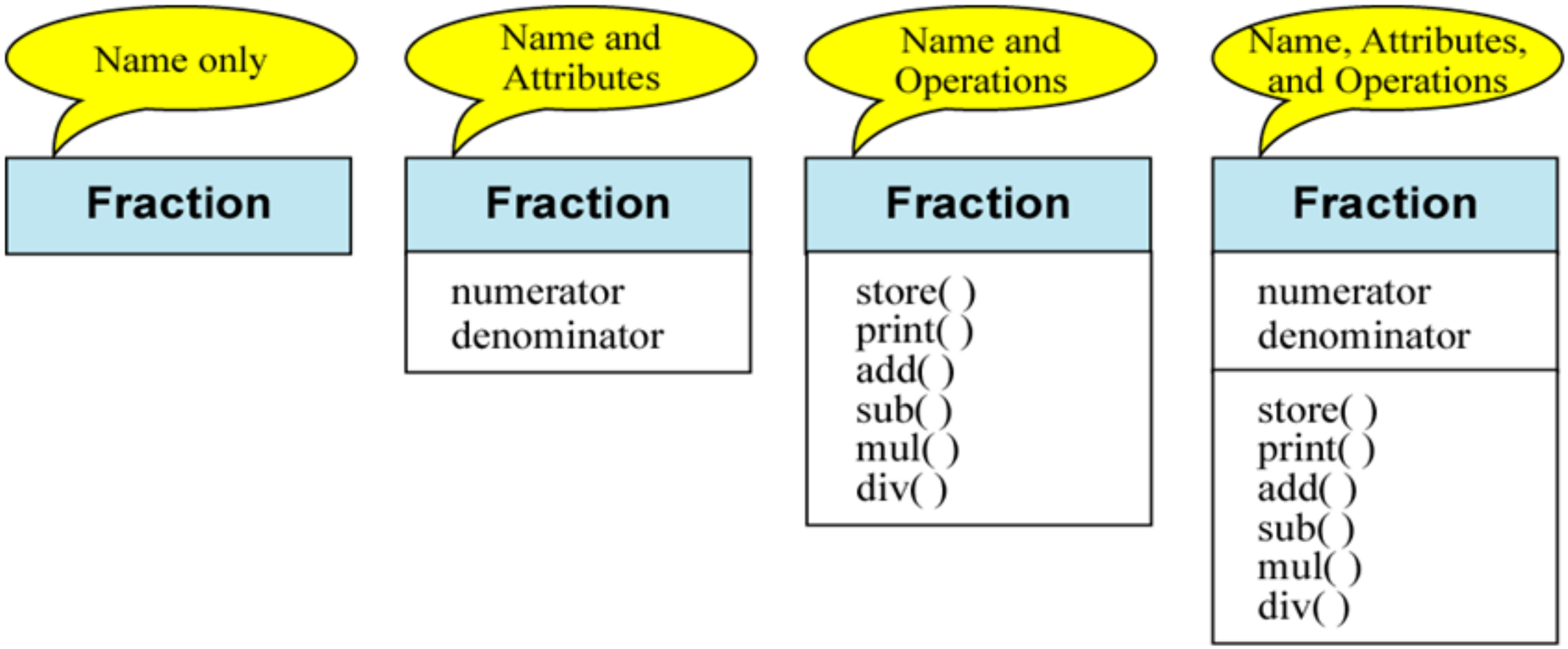
```
class Fraction
{
        private:
                …
        public:
                friend Fraction add (const Fraction& fr1,
                                     const Fraction& fr2);
}
```

Friend function



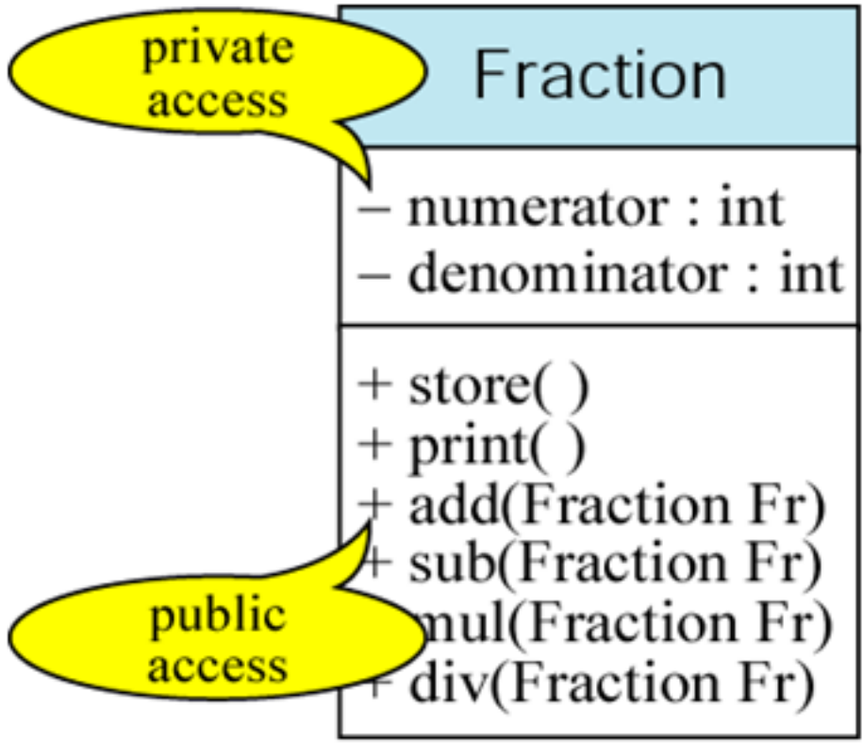
Class diagram (UML : Unified Modeling Language)

- **Promise between programmers to describe Class**



Expanded class diagram

- **Promise between programmers to describe Class**



- http://www.slideshare.net/jdyang54/ss-32523257

Questions?