

Las clases como tipos de datos definidos por el usuario

1. La clase **Fraccional**
2. Representación en UML de los niveles de acceso
3. Categorías de los objetos que aparecen en los métodos
- 4. El puntero `this`**
- 5. Métodos operadores**

1

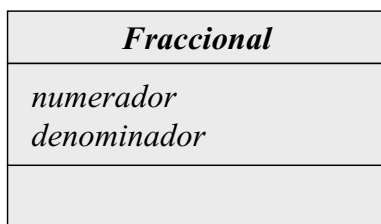
Las clases como tipos de datos definidos por el usuario

La clase **Fraccional**

Vamos a diseñar la clase **Fraccional**, la cual va a permitir modelar un nuevo tipo de datos: los números racionales expresados como fracciones.

$$\frac{\text{Numerador}}{\text{Denominador}}$$

Las fracciones se caracterizan por dos valores de tipo entero que representan el **numerador** y el **denominador**; éstos serán los datos miembro de la clase.



```
class Fraccional
{
    private:
        int numerador;
        int denominador;
};
```

2

Las clases como tipos de datos definidos por el usuario

Operaciones que se pueden realizar

- ➔ Suma, Resta, Producto, División.
- ➔ Las fracciones deben estar simplificadas, por lo tanto necesitamos un método para la operación de simplificación. Este método se ejecuta solo cuando cambia el estado de un objeto. Será un método privado.
- ➔ Además, para cada atributo, habrá un método accedente y un método mutador.
- ➔ Incluiremos un método para mostrar en pantalla el estado del objeto fracción.

3

Las clases como tipos de datos definidos por el usuario

Suma, Resta, Producto y División

<i>Fraccional</i>
<i>numerador</i> <i>denominador</i>
<i>Fraccional(int n, int d)</i> <i>mas(Fraccional): Fraccional</i> <i>menos(Fraccional): Fraccional</i> <i>por(Fraccional): Fraccional</i> <i>entre(Fraccional): Fraccional</i>

Método constructor

```
class Fraccional
{
    private:
        int numerador ;
        int denominador ;

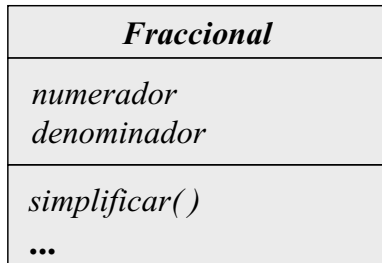
    public:
        Fraccional(int n=0, int d=1);
        Fraccional mas (Fraccional f);
        Fraccional menos(Fraccional f);
        Fraccional por(Fraccional f);
        Fraccional entre(Fraccional f);
};
```

4

Las clases como tipos de datos definidos por el usuario

Un método auxiliar Privado

Necesitamos, a parte de funciones miembro que permitan operar con fracciones (suma, resta, multiplicación, ...) , un método privado que simplifique fracciones. Este método privado solo lo usarán los métodos públicos que modifiquen el estado de un objeto.

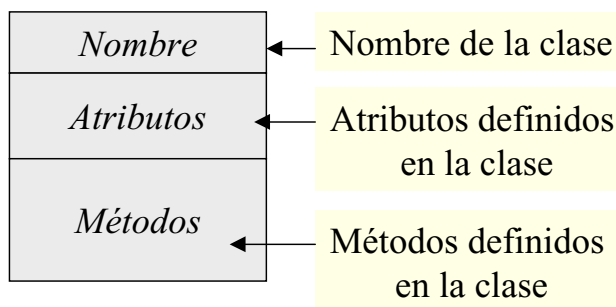


```
class Fraccional
{
    private:
        int numerador ;
        int denominador ;
        void simplificar( ) ;
    public:
        ...
};
```

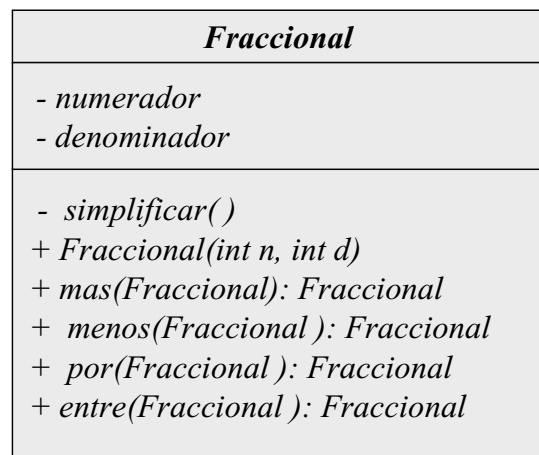
El estado de cada objeto de tipo fraccional representa un número racional simplificado.

Las clases como tipos de datos definidos por el usuario

Representación en UML de los niveles de acceso



Privado: -
Público: +



La clase Fraccional

```
class Fraccional
{
    private:
        int numerador ;
        int denominador ;
        void simplificar( ) ;
    public:
        Fraccional(int n=0, int d=1);
        Fraccional mas (Fraccional f);
        Fraccional menos(Fraccional f);
        Fraccional por(Fraccional f);
        Fraccional entre(Fraccional f);
};
```

```
void Fraccional::simplificar( )
{
    int d;
    d= mcd(numerador,denominador);
    numerador = numerador / d;
    denominador = denominador /d;
}
```

```
void Fraccional::Fraccional(int n=0, int d=1 )
{
    numerador = n;
    denominador = d;
    simplificar( );
}
```

Constructor con argumentos implícitos

El método suma de fracciones

```
class Fraccional
{
    private:
        int numerador ;
        int denominador ;
        void simplificar( ) ;
    public:
        Fraccional(int n=0, int d=1);
        Fraccional mas (Fraccional f);
        Fraccional menos(Fraccional f);
        Fraccional por(Fraccional f);
        Fraccional entre(Fraccional f);
};
```

```
Fraccional Fraccional:: mas (Fraccional f)
{
    Fraccional aux;
    aux.numerador = numerador * f.denominador +
                    denominador * f.numerador;
    aux.denominador = denominador * f.denominador;
    aux. simplifica();
    return aux;
}
```

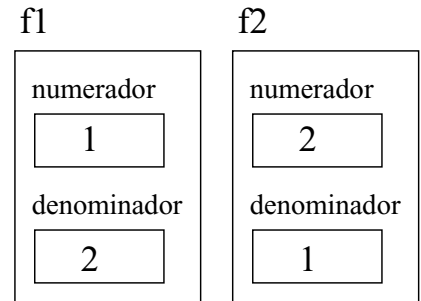
$$\frac{n}{d} + \frac{n'}{d'} = \frac{n \times d' + d \times n'}{d \times d'}$$

Las clases como tipos de datos definidos por el usuario

La clase Fraccional

La operación suma de fracciones, se realiza de forma extraña, ya que su sintaxis es la normal para paso de mensajes.

```
Fraccional Fraccional:: mas (Fraccional f )  
{  
    Fraccional aux;  
    aux.numerador = numerador * f.denominador +  
                    denominador * f.numerador;  
    aux.denominador = denominador * f.denominador;  
    aux.simplifica();  
    return aux;  
}
```



$$\frac{1}{2} + \frac{2}{1} = \frac{5}{2}$$

```
void main()  
{  
    Fraccional f1(1,2), f2(2,1), f3;  
    f3 = f1.mas( f2 );  
}
```

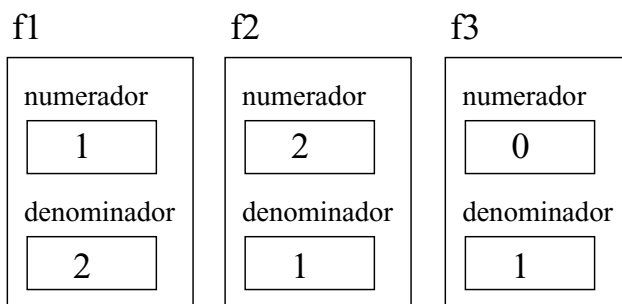
Se envía el mensaje **mas** al objeto **f1** pasando como argumento el objeto **f2**

Las clases como tipos de datos definidos por el usuario

Detalle de la ejecución

```
Fraccional Fraccional:: mas (Fraccional f )  
{  
    Fraccional aux;  
    aux.numerador = numerador * f.denominador +  
                    denominador * f.numerador;  
    aux.denominador = denominador * f.denominador;  
    aux.simplifica();  
    return aux;  
}
```

```
...  
Fraccional f1(1,2), f2(2,1), f3;  
f3 = f1.mas( f2 );  
...
```



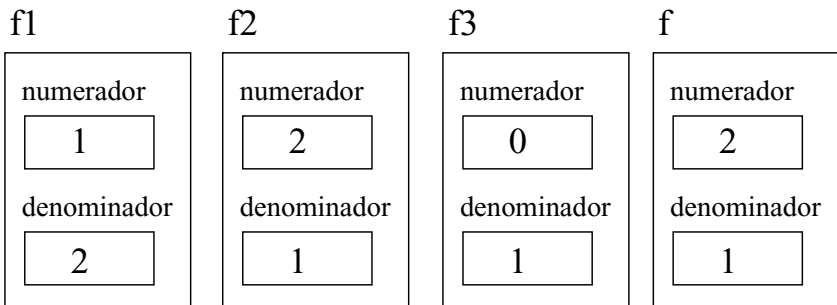
Las clases como tipos de datos definidos por el usuario

Detalle de la ejecución

```
Fraccional Fraccional:: mas (Fraccional f )  
{  
    Fraccional aux;  
    aux.numerador = numerador * f.denominador +  
                    denominador * f.numerador;  
    aux.denominador = denominador * f.denominador;  
    aux.simplifica();  
    return aux;  
}
```

```
...  
Fraccional f1(1,2), f2(2,1), f3;  
f3 = f1.mas(f2);  
...
```

El parámetro **f** es un parámetro por valor, entonces, el compilador crea una copia.



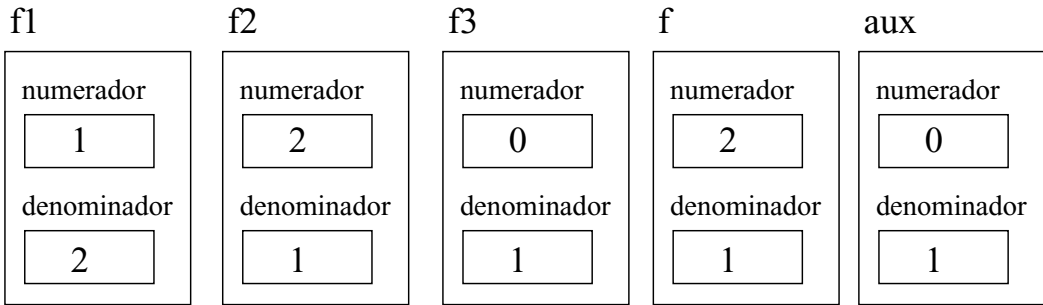
Las clases como tipos de datos definidos por el usuario

Detalle de la ejecución

```
Fraccional Fraccional:: mas (Fraccional f )  
{  
    Fraccional aux;  
    aux.numerador = numerador * f.denominador +  
                    denominador * f.numerador;  
    aux.denominador = denominador * f.denominador;  
    aux.simplifica();  
    return aux;  
}
```

```
...  
Fraccional f1(1,2), f2(2,1), f3;  
f3 = f1.mas(f2);  
...
```

Se crea el objeto local **aux**.



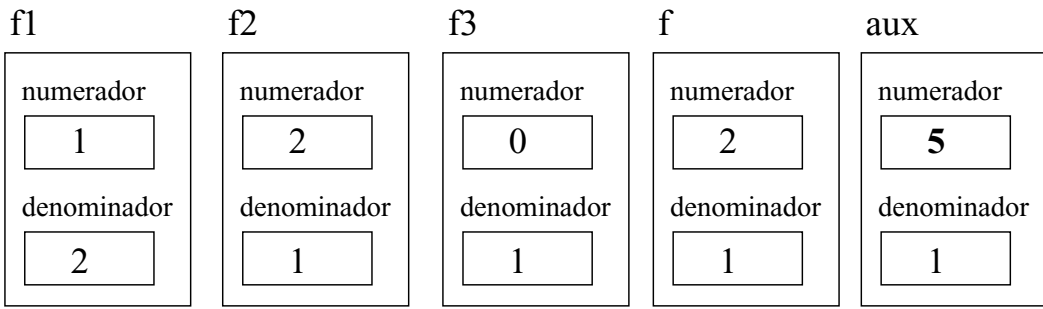
Las clases como tipos de datos definidos por el usuario

Detalle de la ejecución

```
Fraccional Fraccional:: mas (Fraccional f)
{
  Fraccional aux;
  aux.numerador = numerador * f.denominador +
                 denominador * f.numerador;
  aux.denominador = denominador * f.denominador;
  aux.simplifica();
  return aux;
}
```

```
... Fraccional f1(1,2), f2(2,1), f3;
... f3 = f1.mas(f2);
...
```

Se calcula el valor del atributo numerador del objeto **aux**.



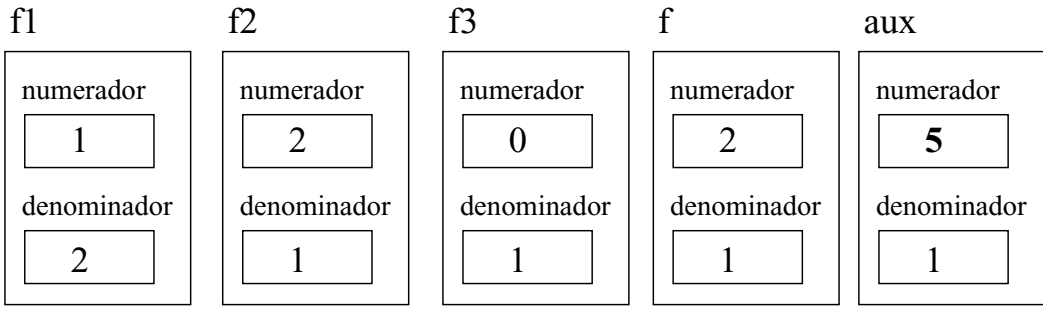
Las clases como tipos de datos definidos por el usuario

Detalle de la ejecución

```
Fraccional Fraccional:: mas (Fraccional f)
{
  Fraccional aux;
  aux.numerador = numerador * f.denominador +
                 denominador * f.numerador;
  aux.denominador = denominador * f.denominador;
  aux.simplifica();
  return aux;
}
```

```
... Fraccional f1(1,2), f2(2,1), f3;
... f3 = f1.mas(f2);
...
```

Se refiere al atributo numerador del **objeto receptor** del mensaje **suma**.



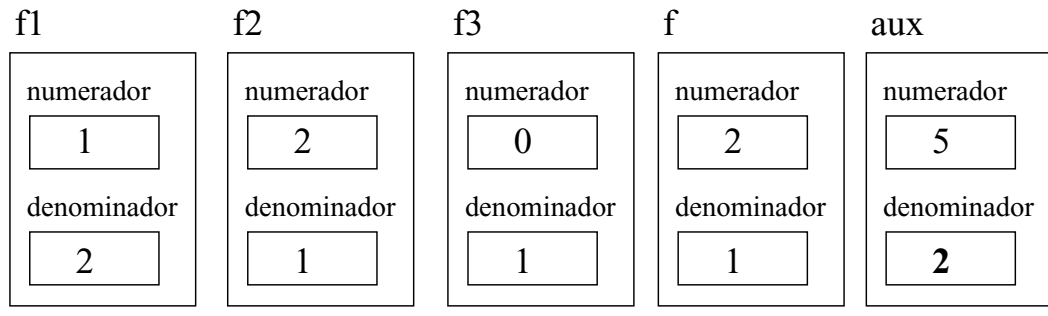
Las clases como tipos de datos definidos por el usuario

Detalle de la ejecución

```
Fraccional Fraccional:: mas (Fraccional f )  
{  
    Fraccional aux;  
    aux.numerador = numerador * f.denominador +  
                    denominador * f.numerador;  
    aux.denominador = denominador * f.denominador;  
    aux.simplifica();  
    return aux;  
}
```

```
...  
Fraccional f1(1,2), f2(2,1), f3;  
f3 = f1.mas( f2 );  
...
```

Se calcula el valor del atributo denominador del objeto **aux**.



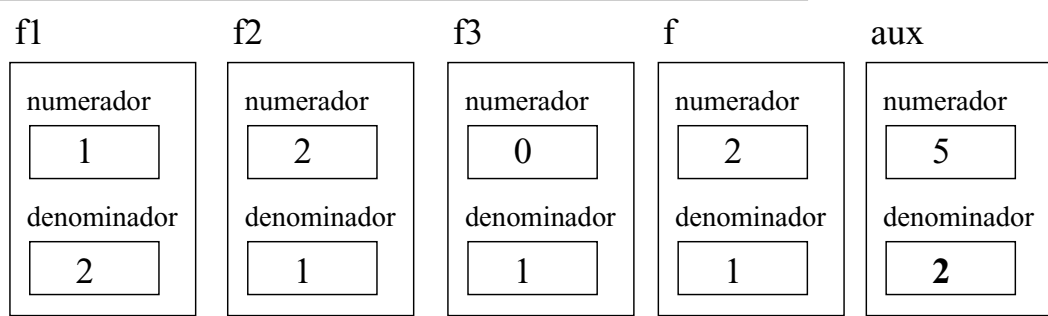
Las clases como tipos de datos definidos por el usuario

Detalle de la ejecución

```
Fraccional Fraccional:: mas (Fraccional f )  
{  
    Fraccional aux;  
    aux.numerador = numerador * f.denominador +  
                    denominador * f.numerador;  
    aux.denominador = denominador * f.denominador;  
    aux.simplifica();  
    return aux;  
}
```

```
...  
Fraccional f1(1,2), f2(2,1), f3;  
f3 = f1.mas( f2 );  
...
```

Se envía el mensaje simplifica() al objeto **aux**.



Las clases como tipos de datos definidos por el usuario

Detalle de la ejecución

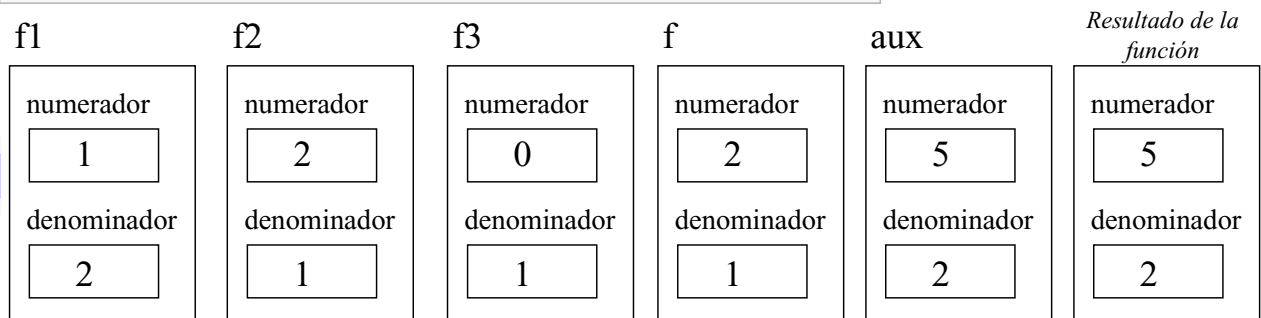
```

Fraccional Fraccional:: mas (Fraccional f)
{
    Fraccional aux;
    aux.numerador = numerador * f.denominador +
                    denominador * f.numerador;
    aux.denominador = denominador * f.denominador;
    aux.simplifica();
    return aux;
}
    
```

```

...
Fraccional f1(1,2), f2(2,1), f3;
f3 = f1.mas(f2);
...
    
```

Devolución por valor
Se devuelve una copia de **aux**.



Las clases como tipos de datos definidos por el usuario

Detalle de la ejecución

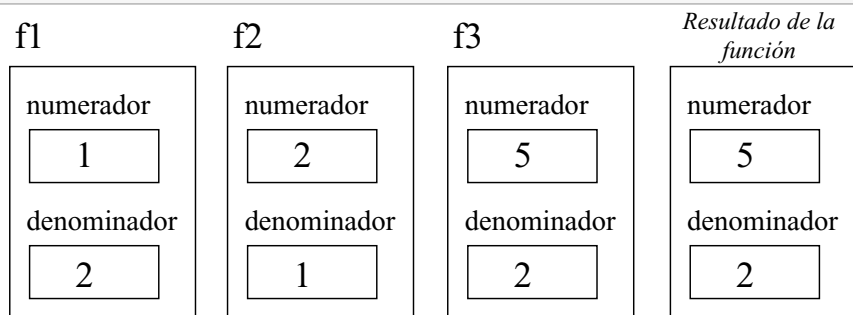
```

Fraccional Fraccional:: mas (Fraccional f)
{
    Fraccional aux;
    aux.numerador = numerador * f.denominador +
                    denominador * f.numerador;
    aux.denominador = denominador * f.denominador;
    aux.simplifica();
    return aux;
}
    
```

```

...
Fraccional f1(1,2), f2(2,1), f3;
f3 = f1.mas(f2);
...
    
```

Se destruyen los objetos temporales **aux** y **f**



Se copia el objeto devuelto por la función **mas** en el objeto **f3**.

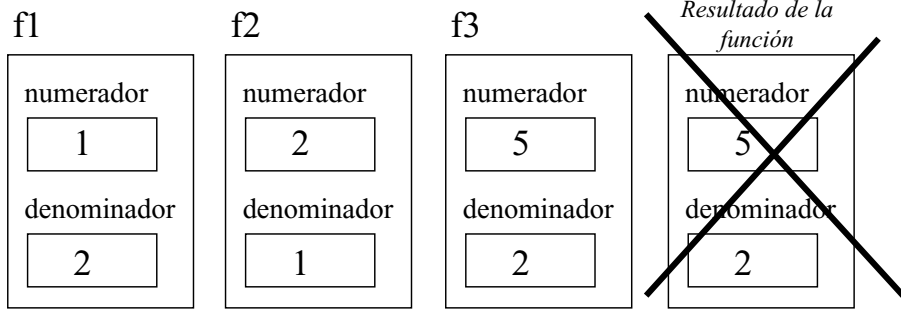
Las clases como tipos de datos definidos por el usuario

Detalle de la ejecución

```
Fraccional Fraccional::mas (Fraccional f )  
{  
    Fraccional aux;  
    aux.numerador = numerador * f.denominador +  
                    denominador * f.numerador;  
    aux.denominador = denominador * f.denominador;  
    aux.simplifica();  
    return aux;  
}
```

```
...  
Fraccional f1(1,2), f2(2,1), f3;  
f3 = f1.mas(f2);  
...
```

Ya no se necesita el objeto devuelto, por lo que se destruye.



Las clases como tipos de datos definidos por el usuario

El método resta de fracciones

```
class Fraccional  
{  
    private:  
        int numerador ;  
        int denominador ;  
        void simplificar( ) ;  
    public:  
        Fraccional(int n=0, int d=1);  
        Fraccional mas (Fraccional f );  
        Fraccional menos(Fraccional f);  
        Fraccional por(Fraccional f);  
        Fraccional entre(Fraccional f);  
};
```

```
Fraccional Fraccional::menos (Fraccional f )  
{  
    Fraccional aux;  
    aux.numerador = numerador * f.denominador -  
                    denominador * f.numerador;  
    aux.denominador = denominador * f.denominador;  
    aux.simplifica();  
    return aux;  
}
```

$$\frac{n}{d} - \frac{n'}{d'} = \frac{n \times d' - d \times n'}{d \times d'}$$

El método producto de fracciones

```
class Fraccional
{
private:
    int numerador ;
    int denominador ;
    void simplificar( ) ;
public:
    Fraccional(int n=0, int d=1);
    Fraccional mas (Fraccional f);
    Fraccional menos(Fraccional f);
    Fraccional por(Fraccional f);
    Fraccional entre(Fraccional f);
};
```

```
Fraccional Fraccional::por (Fraccional f)
{
    Fraccional aux;
    aux.numerador = numerador * f.numerador;
    aux.denominador = denominador * f.denominador;
    aux.simplifica();
    return aux;
}
```

$$\frac{n}{d} \times \frac{n'}{d'} = \frac{n \times n'}{d \times d'}$$

El método división de fracciones

```
class Fraccional
{
private:
    int numerador ;
    int denominador ;
    void simplificar( ) ;
public:
    Fraccional(int n=0, int d=1);
    Fraccional mas (Fraccional f);
    Fraccional menos(Fraccional f);
    Fraccional por(Fraccional f);
    Fraccional entre(Fraccional f);
};
```

```
Fraccional Fraccional::entre (Fraccional f)
{
    Fraccional aux;
    aux.numerador = numerador * f.denominador;
    aux.denominador = denominador * f.numerador;
    aux.simplifica();
    return aux;
}
```

$$\frac{n}{d} \div \frac{n'}{d'} = \frac{n \times d'}{d \times n'}$$

Método accedente y mutador para cada atributo

<i>Fraccional</i>
- <i>numerador</i> - <i>denominador</i>
+ <i>Fraccional(int n, int d)</i> - <i>simplificar()</i> + <i>mas(Fraccional): Fraccional</i> + <i>menos(Fraccional): Fraccional</i> + <i>por(Fraccional): Fraccional</i> + <i>entre(Fraccional): Fraccional</i> <i>//accedentes ----</i> + <i>ac_numerador () : int</i> + <i>ac_denominador (): int</i> <i>// mutadores ----</i> + <i>mu_numerador (int n)</i> + <i>mu_denominador (int d)</i>

```
class Fraccional
{
  private:
    int numerador ;
    int denominador ;
    void simplificar ( ) ;
  public:
    Fraccional(int n=0, int d=1);
    Fraccional mas (Fraccional f);
    Fraccional menos(Fraccional f);
    Fraccional por(Fraccional f);
    Fraccional entre(Fraccional f);
    int ac_numerador( );
    int ac_denominador( );
    void mu_numerador (int n);
    void mu_denominador ( int d);
};
```

Método accedente y mutador para cada atributo

```
class Fraccional
{
  private:
    int numerador ;
    int denominador ;
    void simplificar ( ) ;
  public:
    ...
    ...
    int ac_numerador( );
    int ac_denominador( );
    void mu_numerador (int n);
    void mu_denominador ( int d);
};
```

```
int Fraccional:: ac_numerador ( )
{
  return numerador;
}
```

```
int Fraccional:: ac_denominador ( )
{
  return denominador;
}
```

```
void Fraccional::mu_denominador ( int d )
{
  denominador = d;
  simplificar( );
}
```

Las clases como tipos de datos definidos por el usuario

Método para mostrar el estado de un objeto

```
class Fraccional
{
private:
    int numerador ;
    int denominador ;
    void simplificar( ) ;
public:
    ...
    ...
    int ac_numerador( ) ;
    int ac_denominador( ) ;
    void mu_numerador (int n);
    void mu_denominador ( int d);
    void mostrar ( ) ;
};
```

```
void Fraccional::mostrar( )
{
    cout << numerador << "/" << denominador ;
}
```

25

Las clases como tipos de datos definidos por el usuario

Detalle de ejecución de la clase Fraccional

```
class Fraccional
{ ... }

void main()
{
    Fraccional f1, f2, f3;
    f1.mu_numerador (2);
    f1.mu_denominador(3);
    f1.mostrar();
    cout << endl;
    f2.mu_numerador (5);
    f2.mu_denominador(7);
    f2.mostrar();
    cout << endl;
    ...
    ...
}
```

```
2/3      // f1
5/7      // f2
```

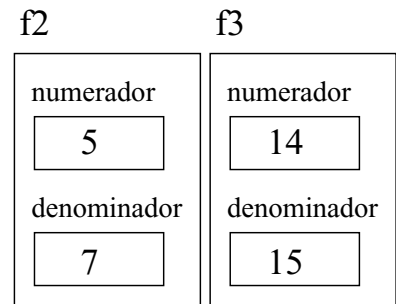
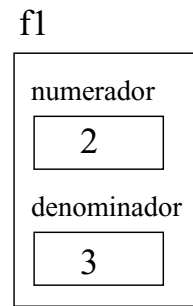
26

Las clases como tipos de datos definidos por el usuario

Detalle de ejecución de la clase Fraccional

```
...  
cout << "Numerador: " << f2.acc_numerador ();  
cout << "Denominador: " << f2.acc_denominador();  
f3 = f1.mas(f2);  
f3.mostrar();  
cout << endl;  
f3.= f1.menos (f2);  
f3.mostrar();  
cout << endl;  
f3.= f1.por (f2);  
f3.mostrar();  
cout << endl;  
return;  
}
```

```
2/3  
5/7  
Numerador: 5  
Denominador: 7  
29/21 // f1+f2  
-1/10  
10/21  
14/15
```



Al ejecutarse la instrucción **return**, se destruyen todos los objetos del programa

27

Las clases como tipos de datos definidos por el usuario

Categorías de objetos que aparecen en los métodos

- Objeto receptor del mensaje
 - Es el que provoca la ejecución del método.
 - El método se ejecuta sobre él.
 - Acceso total a sus atributos y métodos sin el operador punto (•).
- Objetos parámetros
 - Hacen referencia a los objetos argumentos. Se pasan por copia o por referencia.
 - Si son de la misma clase, se tiene acceso a lo privado.

Se crean cuando comienza la ejecución del método y se destruyen cuando termina.

- Objetos locales
 - Declarados en el cuerpo de los métodos.
 - Si son de la misma clase, se tiene acceso a lo privado.

28

Las clases como tipos de datos definidos por el usuario

this: puntero al objeto receptor del mensaje

- * Los métodos tienen definido un **parámetro secreto** o **fantasma** que no aparece en la lista de parámetros.
- * Este parámetro fantasma es un puntero a un objeto de la clase, su nombre es **this**.
- * El parámetro fantasma es un puntero al objeto receptor del mensaje.
- * Dentro de los métodos de la clase, en el código, se puede utilizar ese parámetro para acceder a los atributos del objeto receptor o hacer que el objeto receptor se pase mensajes a sí mismo.

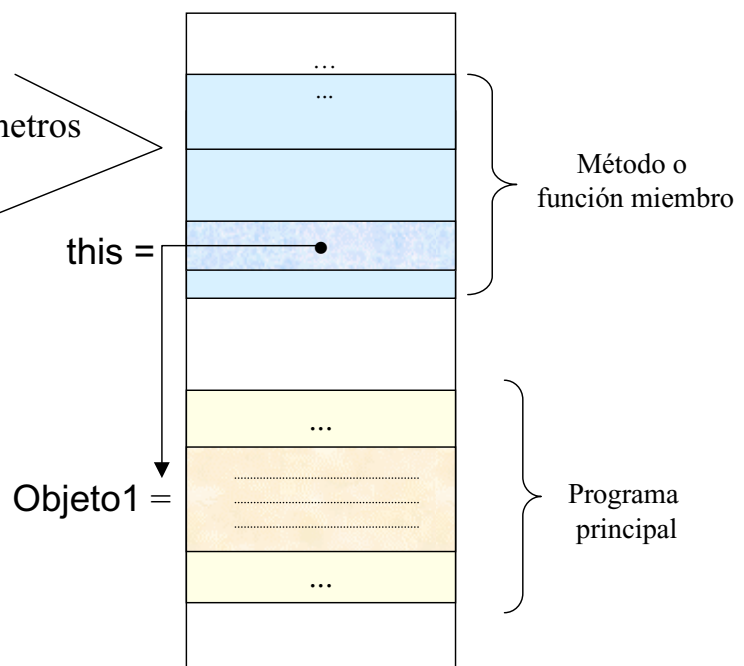
29

Las clases como tipos de datos definidos por el usuario

this: puntero al objeto receptor del mensaje

- Variables locales
- Variables de la lista de parámetros
- El parámetro fantasma: **this**

```
void main( )  
{  
    ....  
    objeto1.metodo ();  
    ....  
}
```



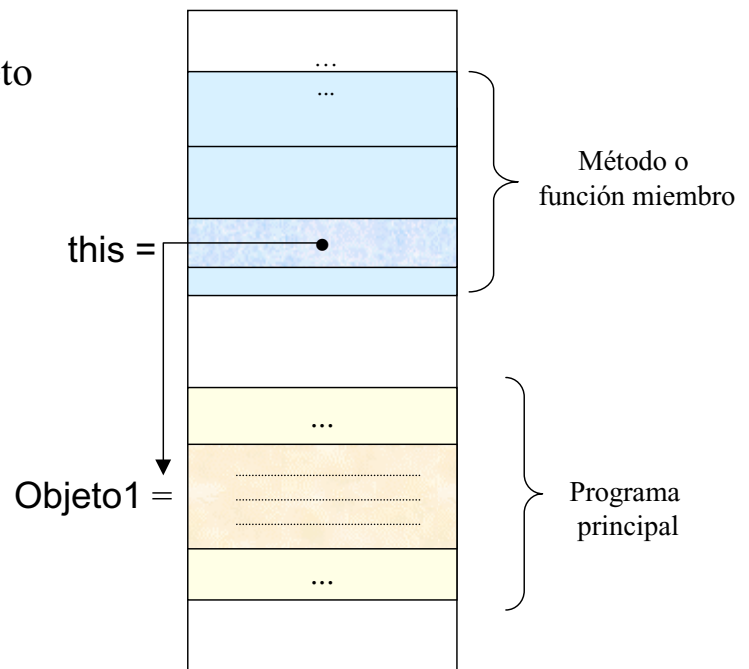
30

Las clases como tipos de datos definidos por el usuario

this: puntero al objeto receptor del mensaje

Como **this** es un puntero al objeto receptor del mensaje, podemos acceder a sus miembros con el operador flecha (\rightarrow).

```
void metodo ()  
{  
    ....  
    this -> atributo;  
    ....  
}
```



Las clases como tipos de datos definidos por el usuario

this: puntero al objeto receptor del mensaje

```
void Fraccional::mu_denominador ( int d )  
{  
    denominador = d;  
    simplificar( );  
}
```

```
void Fraccional::mu_denominador ( int d )  
{  
    this->denominador = d;  
    this->simplificar( );  
}
```

En éste caso, se puede omitir **this->**

Aunque para acceder a los atributos o para pasarse mensajes no sea necesario el uso de **this**, hay ocasiones en las que resulta imprescindible: cuando el método tiene que devolver una copia del objeto receptor:

```
return *this;
```

Las clases como tipos de datos definidos por el usuario

Métodos operadores

Las operaciones que realizamos con los objetos de la clase fraccional, se realizan de forma extraña, ya que su sintaxis es la normal para paso de mensajes.

```
...  
f3 = f1.mas(f2);  
...
```

Sería mucho más natural expresar la suma como:

```
...  
f3 = f1 + f2 ;  
...
```

Es decir, sería más cómodo usar el operador + predefinido en el lenguaje para usarlo con otro tipo de datos. A esto se le llama **sobrecarga de operadores**.

Las clases como tipos de datos definidos por el usuario

Métodos operadores

Para poder usar el **operador +** para sumar fracciones, podemos definir un método al que llamaremos método operador o función miembro operadora.

Prototipo

```
Fraccional operator + ( Fraccional f );
```

operator es
palabra reservada

La implementación de éstos métodos es exactamente la misma.

```
Fraccional Fraccional :: operator + (Fraccional f )  
{  
    Fraccional aux;  
    aux.numerador = numerador * f.denominador + denominador * f.numerador;  
    aux.denominador = denominador * f.denominador;  
    aux.simplifica();  
    return aux;  
}
```

Las clases como tipos de datos definidos por el usuario

Métodos operadores

La clase Fraccional con métodos operadores :

Cuando los métodos se implementan como funciones operadoras, podemos enviar los mensajes +, -, * y / a los objetos de la clase Fraccional de forma más natural (Notación infija)

```
...  
f3 = f1 + f2 ;  
...
```

```
class Fraccional  
{  
private:  
    int numerador ;  
    int denominador ;  
    void simplificar( ) ;  
public:  
    Fraccional(int n=0, int d=1);  
    Fraccional operator + ( Fraccional f );  
    Fraccional operator - (Fraccional f);  
    Fraccional operator * (Fraccional f);  
    Fraccional operator / (Fraccional f);  
    int ac_numerador( );  
    int ac_denominador( );  
    void mu_numerador (int n);  
    void mu_denominador ( int d);  
};
```

Al objeto **f1** se le envía el mensaje + pasándole como argumento el objeto **f2**.

Las clases como tipos de datos definidos por el usuario

Métodos operadores

Esto no es más que una forma abreviada del envío de mensajes habitual:

```
...  
f3 = f1. operator + ( f2 ) ;  
...
```



```
...  
f3 = f1 + f2 ;  
...
```

```
class Fraccional  
{  
private:  
    int numerador ;  
    int denominador ;  
    void simplificar( ) ;  
public:  
    Fraccional(int n=0, int d=1);  
    Fraccional operator + ( Fraccional f );  
    Fraccional operator - (Fraccional f);  
    Fraccional operator * (Fraccional f);  
    Fraccional operator / (Fraccional f);  
    int ac_numerador( );  
    int ac_denominador( );  
    void mu_numerador (int n);  
    void mu_denominador ( int d);  
};
```