

Constructores y Destrucciones

Creación/inicialización de Objetos

```
#include "fraction.h"
```

```
int main()
```

```
{
```

```
    float x;
```

```
    float y = 6.7;
```

```
    float z(7.2);
```

```
    Fraction f;
```

```
    Fraction g(4, 5);
```

```
    ...
```

Creación/inicialización de Objetos

```
#include "fraction.h"
```

```
int main()
```

```
{
```



```
float x;
```

```
float y = 6.7;
```

```
float z(7.2);
```

```
Fraction f;
```

```
Fraction g(4, 5);
```

```
...
```


Creación/inicialización de Objetos

```
#include "fraction.h"
```

```
int main()
```

```
{
```

```
    float x;
```

```
     float y = 6.7;
```

```
    float z(7.2);
```

```
    Fraction f;
```

```
    Fraction g(4, 5);
```

```
    ...
```

Creación/inicialización de Objetos

```
#include "fraction.h"
```

```
int main()
```

```
{
```

```
    float x;
```

```
    float y = 6.7;
```

```
     float z(7.2);
```

```
    Fraction f;
```

```
    Fraction g(4, 5);
```

```
    ...
```

Creación/inicialización de Objetos

```
#include "fraction.h"
```

```
int main()
```

```
{
```

```
    float x;
```

```
    float y = 6.7;
```

```
    float z(7.2);
```

```
     Fraction f;
```

```
    Fraction g(4, 5);
```

```
    ...
```

Creación/inicialización de Objetos

```
#include "fraction.h"
```

```
int main()
```


```
{
```

```
    float x;
```

```
    float y = 6.7;
```

```
    float z(7.2);
```

```
    Fraction f;
```

```
     Fraction g(4, 5);    // debemos hacer un constructor adecuado para esto
```

```
    ...
```

Constructores

- Tienen el mismo nombre que la clase
- No tienen tipo de retorno (tampoco `void`), por lo tanto no tienen `return`
- Pueden existir varios constructores con el mismo nombre (sobrecarga de funciones, ver tema 1)
- El constructor apropiado se llama automáticamente cuando creamos un objeto de una clase
- El compilador proporciona un constructor por defecto para cada clase *si no creamos ninguno*; pero si creamos cualquier constructor, desaparece el constructor por defecto

Constructores: Ejemplos


```
class Fraction
{
    public:
        Fraction(const int num, const int den);
        void readin();
        void print();
        ...
};
```

Constructores: Ejemplos

```
class Fraction
{
    public:
        Fraction(const int num, const int den);
        void readin();
        void print();
        ...
};

Fraction::Fraction(const int num, const int den)
{
    m_numerator = num;
    m_denominator = den;
}
```

Constructores: Ejemplos



```
class Fraction
{
    public:
        int Fraction(const int num, const int den);
        void readin();
        void print();
        ...
};
```

No tienen tipo de retorno

```
Fraction::Fraction(const int num, const int den)
{
    m_numerator = num;
    m_denominator = den;
}
```

Constructores: Ejemplos

```
class Fraction
```

```
{
```

```
public:
```

```
    Fraction(const int num, const int den);
```

```
    void readin();
```

```
    void print();
```

```
};
```

No tienen tipo de retorno

```
void Fraction::Fraction(const int num, const int den)
```

```
{
```

```
    m_numerator = num;
```

```
    m_denominator = den;
```

```
}
```

Constructores: Ejemplos

```
class Fraction
{
public:
    Fraction(const int num, const int den);
    void readin();
    void print();
    ...
};

Fraction::Fraction(const int num, const int den)
{
    m_numerator = num;
    m_denominator = den;
    return;
}
```

No!

No tienen sentencia de
retorno (return)

Constructores: Ejemplos

```
// main.cpp
#include "fraction.h"
```

```
int main()
{
    float x;
    float y = 6.7;
    float z(7.2);
    Fraction f;
    Fraction g(4, 0);
    ...
}
```

```
// fraction.cpp
Fraction::Fraction(const int num, const int den)
{
    m_numerator = num;
    m_denominator = den;
}
```

Constructores: Ejemplos

```
// main.cpp
#include "fraction.h"
```

```
int main()
{
```

```
    float x;
```

```
    float y = 6.7;
```

```
    float z(7.2);
```

```
    Fraction f;
```



 `Fraction g(4, 0);` // **DENOMINADOR INICIALIZADO A CERO !!**

```
    ...
```

```
// fraction.cpp
```

```
Fraction::Fraction(const int num, const int den)
```

```
{
```

```
    m_numerator = num;
```

```
    m_denominator = den;                      // permite cualquier valor, incluso cero!
```

```
}
```

Constructores: Ejemplos

```
// main.cpp
#include "fraction.h"
```

```
int main()
{
    float x;
    float y = 6.7;
    float z(7.2);
    Fraction f;
    Fraction g(4, 0);
    ...
}
```

```
// fraction.cpp
Fraction::Fraction(const int num, const int den)
{
    setNumer(num);
    setDenom(den);    // dejamos que la función SETTER controle los
                     // valores pasados como argumentos
}
```


Constructores: Ejemplos

```
// main.cpp
#include "fraction.h"
```

```
int main()
{
    float x;
    float y = 6.7;
    float z(7.2);
    Fraction f;
    Fraction g(4, 0);
    ...
}
```

```
// fraction.cpp
Fraction::Fraction(const int num, const int den):
    m_numerator(num), m_denominator(den) {}
```



```
// debemos incluir el cuerpo de la función cuando usamos una  
// lista de inicialización siempre, aunque esté vacío
```

Constructores: Ejemplos

```
// main.cpp
#include "fraction.h"
```

```
int main()
{
    float x;
    float y = 6.7;
    float z(7.2);
    Fraction f;
    Fraction g(4, 0);
    ...
}
```

```
// fraction.cpp
Fraction::Fraction(const int num, const int den):
    m_numerator(num), m_denominator(den)
{
    if (m_denominator == 0)
        cout << "error: 0 passed in as denominator" << endl;
}
```

Constructores: Ejemplos

```
#include "fraction.h"
```

```
int main()
```

```
{
```

```
    float x;
```

```
    float y = 6.7;
```

```
    float z(7.2);
```

```
    ➡ Fraction f;
```

```
    Fraction g(4, 5);
```

```
    ...
```

Cuando implementamos un constructor, el compilador deja de proporcionar el constructor por defecto (constructor sin argumentos)

Constructor por defecto (*Default constructor*)

```
// main.cpp
```

```
...
```

```
int main()
```

```
{
```

```
    Fraction f;
```

```
    ...
```

```
// fraction.h
```

```
...
```

```
class Fraction
```

```
{
```

```
    Fraction();    // default constructor
```

```
    ...
```

```
// fraction.cpp
```

```
Fraction::Fraction() : m_numerator(0), m_denominator(1) {}    // defaults to 0/1
```

Constructor Copia (*Copy constructor*)

```
#include "fraction.h"
```

```
int main()
```

```
{
```

```
    float x;
```

```
    float y = 6.7;
```

```
    float z(7.2);
```

```
    Fraction f;
```

```
    Fraction g(4, 5);
```



```
    Fraction h(g);
```

```
    // crea un objeto que es una copia de otro
```

```
    ...
```

Constructor Copia

```
// main.cpp
...
int main()
{
    Fraction g(4, 5);
    Fraction h(g);
    ...

// fraction.h
...
class Fraction
{
    Fraction(const Fraction & source);
    ...
```

Constructor Copia

```
// main.cpp
```

```
...
```

```
int main()
```

```
{
```

```
    Fraction g(4, 5);
```

```
    Fraction h(g);
```

```
    ...
```

```
// fraction.h
```

```
...
```

```
class Fraction
```

```
{
```

```
    Fraction(const Fraction& source);
```

```
    ...
```

```
// fraction.cpp
```

```
Fraction::Fraction(const Fraction& source) :
```

```
    m_numerator(source.m_numerator), m_denominator(source.m_denominator) {}
```

Lo que **NO** debemos hacer

```
#include "fraction.h"
```

```
int main()
```

```
{
```

```
    Fraction f();
```

```
    ...
```

```
// El compilador entiende que queremos declarar  
// una función llamada "f" que no tiene parámetros  
// y que devuelve un objeto de clase Fraction
```


Destruyores

- Tienen el mismo nombre que la clase, precedido del carácter ~
- No tienen tipo de retorno (tampoco `void`), por lo tanto no tienen `return`
- Sólo puede existir como máximo un destructor para una clase, y no tiene parámetros
- El destructor se llama **automáticamente** cuando un objeto de una clase se sale del *scope* del programa
- Los destructores son útiles cuando tenemos objetos que solicitan memoria dinámica, para liberarla