Miembros static

Atributos static

- Pertenecen globalmente a la clase, representando a **todos** los objetos de la clase
- Por tanto poseen un único valor (al contrario que los que no son “static”, que poseen un valor diferente para cada objeto de la clase)

Atributos static: Ejemplo

```
// fraction.h
class Fraction
{
  public:
    Fraction(const int n=0, const int d=1) : m_numerator(n) { setDenom(d); }
    void readin();
    ...
    static double zero_tolerance;
  private:
    int m_numerator;
    int m_denominator;
};

// fraction.cpp
double Fraction::zero_tolerance = .0001;
```

```
// fraction.h
class Fraction
{
  public:
    Fraction(const int n=0, const int d=1) : m_numerator(n) { setDenom(d); }
    ...
    static double zero_tolerance;
  private:
    int m_numerator;
    int m_denominator;
};

// fraction.cpp
double Fraction::zero_tolerance = .0001;

// main.cpp
  ...
  Fraction f(3, 100000);
  if (static_cast<double>(f.getNum())/f.getDen() <= Fraction::zero_tolerance)
      f.setNumer(0);
```

```
// fraction.h
class Fraction
{
  public:
    static double zero_tolerance;
    ...
  private:
    void zeroize_if_close();
    ...
};

// fraction.cpp
double Fraction::zero_tolerance = .0001;

void Fraction::zeroize_if_close()
{
   ...
}
```

```
// fraction.cpp
void Fraction::zeroize_if_close()
{
    if(static_cast<double>(m_numerator)/m_denominator <= zero_tolerance)
        m_numerator = 0;
    return;
}

void Fraction::readin()
{
    cout << "enter numerator: ";
    cin >> m_numerator;
    cout << "enter denominator: ";
    cin >> m_denominator;

    zeroize_if_close();
    return;
}
```

Métodos static

- Pertenecen globalmente a la clase, representando a **todos** los objetos de la clase
- Por tanto no pertenecen a ningún objeto de la clase en particular

Métodos static: Ejemplo

```
// fraction.h
class Fraction
{
  public:
    static double get_zero_tolerance(){ return zero_tolerance; }
    static void set_zero_tolerance(const double tol){zero_tolerance = tol;}
    ...
};

// fraction.cpp
double Fraction::zero_tolerance = .0001;

// main.cpp
    ...
    cout << Fraction::get_zero_tolerance() << endl;
    Fraction::set_zero_tolerance(.000001);
    cout << Fraction::get_zero_tolerance() << endl;
```