Encapsulamiento

Acceso a atributos desde funciones “no-miembro”

```
//fraction.h
...
class Fraction
{
  ...
 private:
  int m_numerator;
  int m_denominator;
};


//functions.cpp
...
Fraction mult_fracs(const Fraction& lhs, const Fraction& rhs)
{
    Fraction temp;
    temp.m_numerator = lhs.m_numerator * rhs.m_numerator;
    ...
```

Acceso a atributos desde funciones “no-miembro”

```
//fraction.h
...
class Fraction
{
  ...
 private:
  int m_numerator;
  int m_denominator;
};


//functions.cpp
...
Fraction mult_fracs(const Fraction& lhs, const Fraction& rhs)
{
    Fraction temp;
    temp.m_numerator = lhs.m_numerator * rhs.m_numerator;
    ...
```

Acceso a atributos desde funciones “no-miembro”

```
//fraction.h
...
class Fraction
{
  ...
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//functions.cpp
...
Fraction mult_fracs(const Fraction& lhs, const Fraction& rhs)
{
    Fraction temp;
    temp.m_numerator = lhs.m_numerator * rhs.m_numerator;
    ...
```

Solución 1: “getters & setters”

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

Solución 1: “getters & setters”

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
int Fraction::getNum()
{
    return m_numerator;
}
```

Solución 1: “getters & setters”

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
int Fraction::getDen()
{
    return m_denominator;
}
```

Solución 1: “getters & setters”

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
void Fraction::setNumer(const int numer)
{
    m_numerator = numer;
    return;
}
```

Solución 1: “getters & setters”

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
bool Fraction::setDenom(const int denom)
{
    bool set = false;
    if(denom != 0)
    {
        set = true;
        m_denominator = denom;
    }
    return set;
}
```

Solución 1: “getters & setters” -> inline

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
int Fraction::getNum()
{
    return m_numerator;
}
```

Solución 1: “getters & setters” -> inline

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
int Fraction::getNum()
{
    return m_numerator;
}
```

Solución 1: “getters & setters” -> inline

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum() {                    }
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
int Fraction::getNum()
{
    return m_numerator;
}
```

Solución 1: “getters & setters” -> inline

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum() {                    }
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
int Fraction::getNum()
{
  return m_numerator;
}
```

Solución 1: “getters & setters” -> inline

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum() { return m_numerator; }
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
int Fraction::getNum()
{

}
```

Solución 1: “getters & setters” -> inline

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum() { return m_numerator; }
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
```

Solución 1: “getters & setters” -> inline

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum() { return m_numerator; }
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
int Fraction::getDen()
{
    return m_denominator;
}
```

Solución 1: “getters & setters” -> inline

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum() { return m_numerator; }
  int getDen() { return m_denominator; }
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
int Fraction::getDen()
{
    return m_denominator;
}
```

Solución 1: “getters & setters” -> inline

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum() { return m_numerator; }
  int getDen() { return m_denominator; }
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
```

Solución 1

```
// functions.cpp file
Fraction mult_fracs(const Fraction& lhs, const Fraction& rhs)
{
    Fraction temp;
    temp.setNumer(lhs.getNum() * rhs.getNum());

    if ( !temp.setDenom(lhs.getDen() * rhs.getDen()) )
    {
        cout << "ERROR: denominator is 0 ‟ << endl;
    }

    return temp;
}
```

Solución 1

```
// functions.cpp file
Fraction mult_fracs(const Fraction& lhs, const Fraction& rhs)
{
    Fraction temp;
    temp.setNumer(lhs.getNum() * rhs.getNum());

    if ( !temp.setDenom(lhs.getDen() * rhs.getDen()) )
    {
        cout << "ERROR: denominator is 0 ‟ << endl;
    }

    return temp;
}
```

Solución 1: no compila

```
// functions.cpp file
Fraction mult_fracs(const Fraction& lhs, const Fraction& rhs)
{
    Fraction temp;
    temp.setNumer(lhs.getNum() * rhs.getNum());

    if ( !temp.setDenom(lhs.getDen() * rhs.getDen()) )
    {
        cout << "ERROR: denominator is 0 " << endl;
    }

    return temp;
}
```

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
void Fraction::readin()
{
  cout<<"enter numerator: ";
  cin>>m_numerator;
  cout<<"enter denominator: ";
  cin>>m_denominator;

  return;
}
```

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
void Fraction:readin()
{
  cout<<"enter numerator: ";
  cin>>m_numerator;
  cout<<"enter denominator: ";
  cin>>m_denominator;

  return;
}
```

NOT READ-ONLY

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
void Fraction::print()
{
    cout<<"("<<m_numerator
        <<"/"<<m_denominator<<")";
    return;
}
```

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print() const;
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
void Fraction::print() const
{
    cout<<"("<<m_numerator
        <<"/"<<m_denominator<<")";
    return;
}
```

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print() const;
  Fraction reciprocal();
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
Fraction Fraction::reciprocal()
{
    Fraction returnable;

    returnable.m_numerator =
        m_denominator;
    returnable.m_denominator =
        m_numerator;

    return returnable;
}
```

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print() const;
  Fraction reciprocal() const;
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
Fraction Fraction::reciprocal() const
{
    Fraction returnable;

    returnable.m_numerator =
        m_denominator;
    returnable.m_denominator =
        m_numerator;

    return returnable;
}
```

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print() const;
  Fraction reciprocal() const;
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
void Fraction::unreduce (const int m)
{
    m_numerator*=m;
    m_denominator*=m;
    return;
}
```

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print() const;
  Fraction reciprocal() const;
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
void Fraction:unreduce (const int m)
{
  m_numerator*=m;
  m_denominator*=m;
  return;
```

NOT READ-ONLY

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print() const;
  Fraction reciprocal() const;
  void unreduce(const int m);
  int getNum();
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
int Fraction::getNum()
{
    return m_numerator;
}
```

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print() const;
  Fraction reciprocal() const;
  void unreduce(const int m);
  int getNum() const;
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
int Fraction::getNum() const
{
    return m_numerator;
}
```

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print() const;
  Fraction reciprocal() const;
  void unreduce(const int m);
  int getNum() const;
  int getDen();
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
int Fraction::getDen()
{
    return m_denominator;
}
```

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print() const;
  Fraction reciprocal() const;
  void unreduce(const int m);
  int getNum() const;
  int getDen() const;
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
int Fraction::getDen() const
{
    return m_denominator;
}
```

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print() const;
  Fraction reciprocal() const;
  void unreduce(const int m);
  int getNum() const;
  int getDen() const;
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
void Fraction::setNumer(const int numer)
{
    m_numerator = numer;
    return;
}
```

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print() const;
  Fraction reciprocal() const;
  void unreduce(const int m);
  int getNum() const;
  int getDen() const;
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
void Fraction::setNumer(const int numer)
{
  m_numerator = numer;
  return;
}
```

NOT READ-ONLY

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print() const;
  Fraction reciprocal() const;
  void unreduce(const int m);
  int getNum() const;
  int getDen() const;
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
bool Fraction::setDenom(const int denom)
{
    bool set = false;
    if(denom != 0)
    {
        set = true;
        m_denominator = denom;
    }
    return set;
}
```

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print() const;
  Fraction reciprocal() const;
  void unreduce(const int m);
  int getNum() const;
  int getDen() const;
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

```
//fraction.cpp
...
bool Fraction::setDenom(const int denom)
{
    bool set = false;
    if denom != 0)
    {
        set = true;
        m_denominator = denom;
    }
    return set;
}
```

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print() const;
  Fraction reciprocal() const;
  void unreduce(const int m);
  int getNum() const;
  int getDen() const;
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

Métodos const: no modifican atributos

```
//fraction.h
...
class Fraction
{
 public:
  void readin();
  void print() const;
  Fraction reciprocal() const;
  void unreduce(const int m);
  int getNum() const;
  int getDen() const;
  void setNumer(const int numer);
  bool setDenom(const int denom);
 private:
  int m_numerator;
  int m_denominator;
};
```

DIFERENTE SIGNIFICADO !!!

Solución 1: ya compila

```
// functions.cpp file
Fraction mult_fracs(const Fraction& lhs, const Fraction& rhs)
{
    Fraction temp;
    temp.setNumer(lhs.getNum() * rhs.getNum());

    if ( !temp.setDenom(lhs.getDen() * rhs.getDen()) )
    {
        cout << "ERROR: denominator is 0 ‟ << endl;
    }

    return temp;
}
```

Solución 2: funciones friend

- Una función friend es una función “no-miembro” a la que se le proporcionan derechos de acceso a la parte privada de una clase.
- La palabra reservada *friend* **siempre** se usa dentro de la definición de la clase que proporciona los derechos de acceso.
- La palabra reservada *friend* **nunca** se usa fuera de una definición de clase.

Solución 2: funciones friend

```
//fraction.h
...
class Fraction
{
   ...
   friend Fraction mult_fracs(const Fraction& lhs, const Fraction& rhs);
 private:
   int m_numerator;
   int m_denominator;
};
```

```
//fraction.cpp
...
Fraction mult_fracs(const Fraction& lhs, const Fraction& rhs)
{
    Fraction temp;
    temp.m_numerator = lhs.m_numerator * rhs.m_numerator;
    ...
```

Solución 2: funciones friend

```
//fraction.h
...
class Fraction
{
   ...
   friend Fraction mult_fracs(const Fraction& lhs, const Fraction& rhs);
 private:
   int m_numerator;
   int m_denominator;
};
```

```
//fraction.cpp
...
Fraction mult_fracs(const Fraction& lhs, const Fraction& rhs)
{
    Fraction temp;
    temp.m_numerator = lhs.m_numerator * rhs.m_numerator;
    ...
```

Solución 2: funciones friend

```
//fraction.h
...
class Fraction
{
    ...
    friend Fraction mult_fracs(const Fraction& lhs, const Fraction& rhs);
    ...
};
```

```
//fraction.cpp
...
Fraction mult_fracs(const Fraction& lhs, const Fraction& rhs)
{
    Fraction temp;
    temp.m_numerator = lhs.m_numerator * rhs.m_numerator;
    temp.m_denominator = lhs.m_denominator * rhs.m_denominator;
    return temp;
}
```

Lo que NO debemos hacer

```
//fraction.h
...
class Fraction
{
    ...
    friend Fraction mult_fracs(const Fraction& lhs, const Fraction& rhs);
    ...
};
```

```
//fraction.cpp
...
friend Fraction mult_fracs(const Fraction& lhs, const Fraction& rhs)
{
    Fraction temp;
    temp.m_numerator = lhs.m_numerator * rhs.m_numerator;
    temp.m_denominator = lhs.m_denominator * rhs.m_denominator;
    return temp;
}
```