Objetos y Clases

Conceptos

- En un objeto se agrupan:
  - Tipos de datos -> ***Atributos***
  - Funcionalidad -> ***Métodos***
- Encapsulamiento:
  - Los atributos se utilizan para describir el estado del objeto
  - Los métodos dictaminan la interfaz y la forma de interactuar con otras entidades

Conceptos

- Por analogía con la programación modular:
  - Estructuras de datos -> **Atributos**
  - Funciones -> **Métodos**
- Principales diferencias:
  - Los objetos no solo contienen lo análogo a los campos de una estructura (atributos), sino las operaciones permitidas sobre los mismos (métodos)
  - Los objetos *encapsulan* sus atributos y solo podemos acceder a ellos a través de sus métodos

Objetos y Clases

- Por analogía con la programación modular:
  - Variables -> **Objetos**
  - Tipos -> **Clases**
- Decimos que un objeto es una ***instancia*** de una clase
- En nuestros programas, declararemos las distintas clases en ficheros .h y usaremos objetos pertenecientes a dichas clases en ficheros .cpp

Sintaxis

```
class name
{
  public:
    // function prototypes here
  private:
    // member data here
};
```

Sintaxis: ejemplo

```
class name
{
  public:
    // function prototypes here
  private:
    // member data here
};
```

```
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};
```

Sintaxis: ejemplo

```
class name
{
  public:
    // function prototypes here
  private:
    // member data here
};
```

```
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};
```

Sintaxis: ejemplo

```
class name
{
 public:
    // function prototypes here
  private:
    // member data here
};
```

```
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};
```

Sintaxis: ejemplo

```
class name
{
  public:
    // function prototypes here
  private:
    // member data here
};
```

```
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};
```

Sintaxis: ejemplo

```
class name
{
  public:
    // function prototypes here
  private:
    // member data here
};
```

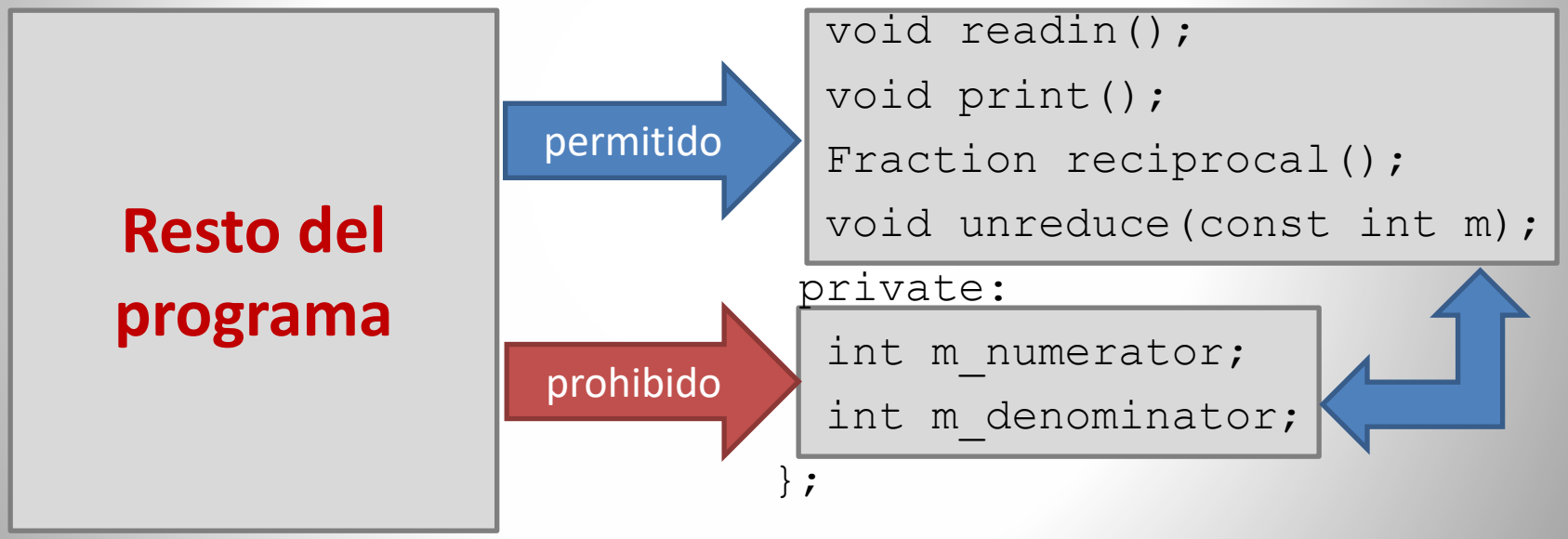
```
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};
```

Sintaxis: ejemplo

```
class name
{
  public:
    // function prototypes here
  private:
    // member data here
};
```

```
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};
```

Sintaxis: ejemplo




```
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};
```

Por defecto: atributos privados

```
class Fraction
{
  int m_numerator;
  int m_denominator;
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
};
```

```
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};
```

Por defecto: atributos privados

```
class Fraction
{
  int m_numerator;
  int m_denominator;
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
};
```

```
class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};
```

Declaración de clase y uso de objetos

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H

class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};

#endif
```

```
#include “fraction.h”

int main()
{
  Fraction f, g;
  f.m_numerator = 7;
  f.readin();
  f.print();
  f.unreduce(5);

  return 0;
}
```

Declaración de clase y uso de objetos

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H

class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};

#endif
```

```
#include "fraction.h"

int main()
{
  Fraction f, g;
  f.m_numerator = 7;
  f.readin();
  f.print();
  f.unreduce(5);

  return 0;
}
```

Declaración de clase y uso de objetos

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H

class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};

#endif
```

```
#include "fraction.h"

int main()
{
  Fraction f, g;
  f.m_numerator = 7; //no puedo acceder!
  f.readin();
  f.print();
  f.unreduce(5);

  return 0;
}
```

Declaración de clase y uso de objetos

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H

class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};

#endif
```

```
#include "fraction.h"

int main()
{
  Fraction f, g;
  f.m_numerator = 7;
  f.readin();
  f.print();
  f.unreduce(5);

  return 0;
}
```

Declaración de clase y uso de objetos

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H

class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};

#endif
```

```
#include "fraction.h"

int main()
{
  Fraction f, g;
  f.m_numerator = 7;
  f.readin();
  f.print();
  f.unreduce(5);

  return 0;
}
```

Declaración de clase y uso de objetos

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H

class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};

#endif
```

```
#include "fraction.h"

int main()
{
  Fraction f, g;
  f.m_numerator = 7;
  f.readin();
  f.print();
  f.unreduce(5);

  return 0;
}
```

Implementación de Métodos

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H

class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};

#endif
```

Implementación de Métodos

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H

class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};

#endif
```

```
//fraction.cpp
#include “fraction.h”
#include <iostream>
using namespace std;
```

Implementación de Métodos

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H

class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};

#endif
```

```
//fraction.cpp
#include "fraction.h"
#include <iostream>
using namespace std;


void Fraction::readin()
{
}
```

Implementación de Métodos

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H

class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};

#endif
```

```
//fraction.cpp
#include "fraction.h"
#include <iostream>
using namespace std;


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

  return;
}
```

Implementación de Métodos

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H

class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};

#endif
```

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

Implementación de Métodos

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H

class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};

#endif
```

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

Implementación de Métodos

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H

class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};

#endif
```

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

Fraction Fraction::reciprocal()
{
    Fraction returnable;
    returnable.m_numerator = m_denominator;
    returnable.m_denominator = m_numerator;
    return returnable;
}
```

Implementación de Métodos

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H

class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};

#endif
```

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

Fraction Fraction::reciprocal()
{
    Fraction returnable;
    returnable.m_numerator = m_denominator;
    returnable.m_denominator = m_numerator;
    return returnable;
}

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

Uso de Métodos

```
#include "fraction.h"

int main()
{
  Fraction f1, f2, f3;
  f1.readin();
  f1.print();
  f2.readin();
  f2.print();
  f2.unreduce(2);
  f2.print();
  f3 = f1.reciprocal();
  f3 = f1 + f2;
  ...
```

Uso de Métodos

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H

class Fraction
{
 public:
  void readin();
  void print();
  Fraction reciprocal();
  void unreduce(const int m);
 private:
  int m_numerator;
  int m_denominator;
};

#endif
```

```
#include "fraction.h"

int main()
{
  Fraction f1, f2, f3;
  f1.readin();
  f1.print();
  f2.readin();
  f2.print();
  f2.unreduce(2);
  f2.print();
  f3 = f1.reciprocal();
  f3 = f1 + f2;
  ...
```

Uso de Métodos

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

  return;
}
```

```
#include "fraction.h"

int main()
{
  Fraction f1, f2, f3;
  f1.readin();
  f1.print();
  f2.readin();
  f2.print();
  f2.unreduce(2);
  f2.print();
  f3 = f1.reciprocal();
  f3 = f1 + f2;
  ...
```

Uso de Métodos

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

```
#include "fraction.h"

int main()
{
  Fraction f1, f2, f3;
  f1.readin();
  f1.print();
  f2.readin();
  f2.print();
  f2.unreduce(2);
  f2.print();
  f3 = f1.reciprocal();
  f3 = f1 + f2;
  ...
```

Uso de Métodos

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

  return;
}
```

```
#include "fraction.h"

int main()
{
  Fraction f1, f2, f3;
  f1.readin();
  f1.print();
  f2.readin();
  f2.print();
  f2.unreduce(2);
  f2.print();
  f3 = f1.reciprocal();
  f3 = f1 + f2;
  ...
```

Uso de Métodos

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

```
#include "fraction.h"

int main()
{
  Fraction f1, f2, f3;
  f1.readin();
  f1.print();
  f2.readin();
  f2.print();
  f2.unreduce(2);
  f2.print();
  f3 = f1.reciprocal();
  f3 = f1 + f2;
  ...
```

Uso de Métodos

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

```
#include "fraction.h"

int main()
{
  Fraction f1, f2, f3;
  f1.readin();
  f1.print();
  f2.readin();
  f2.print();
  f2.unreduce(2);
  f2.print();
  f3 = f1.reciprocal();
  f3 = f1 + f2;
  ...
```

Uso de Métodos

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

```
#include "fraction.h"

int main()
{
  Fraction f1, f2, f3;
  f1.readin();
  f1.print();
  f2.readin();
  f2.print();
  f2.unreduce(2);
  f2.print();
  f3 = f1.reciprocal();
  f3 = f1 + f2;
  ...
```

Uso de Métodos

```
Fraction Fraction::reciprocal()
{
    Fraction returnable;

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

    return returnable;
}
```

```
#include "fraction.h"

int main()
{
  Fraction f1, f2, f3;
  f1.readin();
  f1.print();
  f2.readin();
  f2.print();
  f2.unreduce(2);
  f2.print();
  f3 = f1.reciprocal();
  f3 = f1 + f2;
  ...
```

Uso de Métodos

```
??? Fraction::operator+(???)
{
    ???
}
```

```
#include "fraction.h"

int main()
{
  Fraction f1, f2, f3;
  f1.readin();
  f1.print();
  f2.readin();
  f2.print();
  f2.unreduce(2);
  f2.print();
  f3 = f1.reciprocal();
  f3 = f1 + f2;  //no puedo sumar!!!
  ...
```

Uso de Métodos

La asignación sí está permitida

```
#include "fraction.h"

int main()
{
  Fraction f1, f2, f3;
  f1.readin();
  f1.print();
  f2.readin();
  f2.print();
  f2.unreduce(2);
  f2.print();
  f3 = f1.reciprocal();
  // f3 = f1 + f2;
  f3.print();
  ...
```