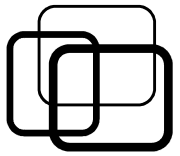
Object Life Cycle

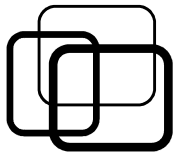
Inst. Nguyễn Minh Huy

Contents



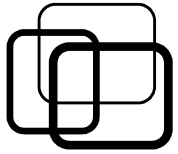
- Constructors.
- Destructor.
- Static members.
- Class template.

Contents



- **Constructors.**
- Destructor.
- Static members.
- Class template.

Constructors



- Object initialization:
  - How to initialize object attributes?

```
class Fraction
{
private:
        int         m_num;
        int         m_den;
};
```

```
int main()
{
        Fraction   p;
        //  Value of p??
}
```

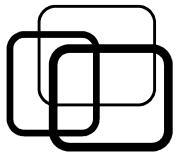
  - Use setters:

```
class Fraction
{
public:
        void setNum( int num );
        void setDen( int den );
};
```

```
int main()
{
        Fraction   p;
        p.setNum( 1 );
        p.setDen( 3 );
}
```

Forget to call?!

Constructors

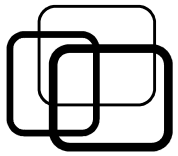


- Role of constructors:
  - “Birth certificate” for object!!
  - Mandatory when declaring object.
  - Can be overloaded.
  - Has no return type.
  - Name: <class name> (C++).

```
class Fraction
{
private:
        int         m_num;
        int         m_den;
public:
        Fraction( int num, int den );
        Fraction( int value );
};
```

```
int main()
{
        Fraction   p1(1, 2);
        Fraction   p2( 5 );
        Fraction   *p3 = new Fraction( 2, 3 );
}
```

Constructors



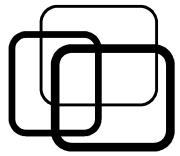
- Default constructor:
  - Default initialization.
  - Has no argument.
  - If class has no constructor:

    ➔ Compiler will provide default constructor.

```
class Fraction
{
private:
        int         m_num;
        int         m_den;
public:
        Fraction();
};
```

```
int main()
{
        Fraction   p;
        Fraction   *q = new Fraction;
}
```

Constructors



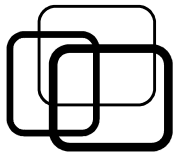
- Copy constructor:
  - Initialization by copying another object.
  - Take object of same class as argument.
  - If class has no copy constructor:

    ➔ Compiler will provide one.

```
class Fraction
{
private:
        int         m_num;
        int         m_den;
public:
        Fraction(const Fraction &p);
};
```

```
int main()
{
        Fraction   p1(1, 2);

        // Copy p1…
        Fraction   p2(p1);
        // Copy p2…
        Fraction   p3 = p2;
}
```

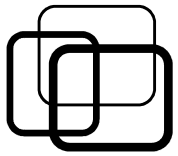
Constructors



- Dr. Guru advises:
  - A class should have at least 3 constructors:
    - Default constructor.
    - Copy constructor.
    - Constructor to initialize all attributes.

```
class Fraction
{
private:
        int         m_num;
        int         m_den;
public:
        Fraction();
        Fraction( const Fraction &p );
        Fraction( int num, int den );
};
```

Constructors



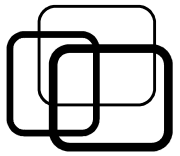
- Constructor implementation:
  - Syntax:

<class name>::**<class name>**(<arguments>) :
        // Simple initialization.
        <attribute 1>(<value 1>) ,
        <attribute 2>(<value 2>) ,
        …
{
        // Complex initialization.
}

```
Fraction::Fraction( int value ) :
    m_num( value ),
    m_den( 1 )
{
}
```

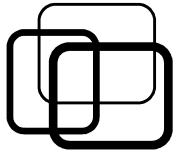
```
Fraction::Fraction( int num, int den ) {
    if ( den == 0 )
        throw std::invalid_argument("Zero den.");
    m_num = num;
    m_den = den;
}
```

Contents



- Constructors.
- **Destructor.**
- Static members.
- Class template.

Destructor



- Memory leak problem:
  - Memory allocated to pointer must be deleted.

```
class Student
{
private:
        char       *m_name;
};
```

```
int main()
{
        Student   s;
}
// Memory leak: s.m_name!!
```

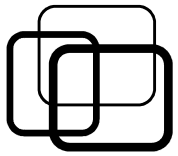
  - Use delete method:

```
class Student
{
private:
        char       *m_name;
public:
        void deleteMemory() {
                delete [ ]m_name;  }
};
```

```
int main()
{
        Student   s;
        s.deleteMemory();
}
```

Forget to call?!

Destructor

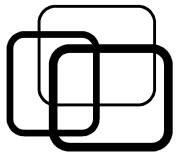


- Role of destructor:
  - “Living will” for object.
  - Automatically called when object is dead.
  - A class has only ONE destructor.
  - Name: ~<class name> (C++).

```
class Student
{
private:
        char       *m_name;
public:
        ~Student() { delete [ ]m_name; }
};
```

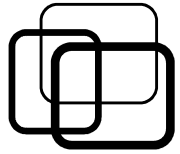
```
int main()
{
        Student   s;
        Student   *p = new Student;
        delete p;
}
```

Contents



- Constructors.
- Destructor.
- **Static members.**
- Class template.

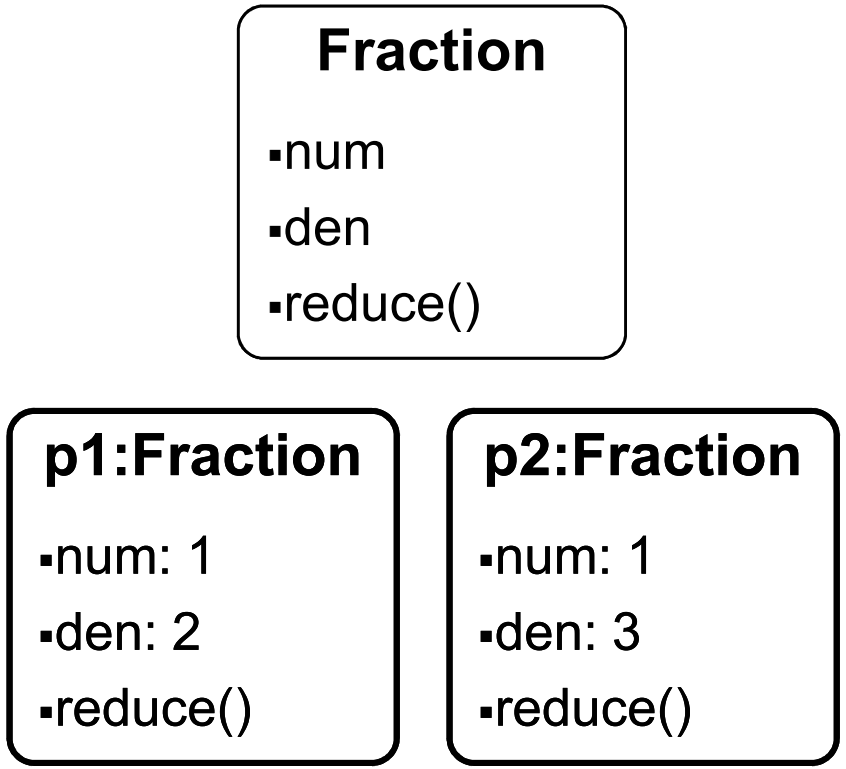
Static members



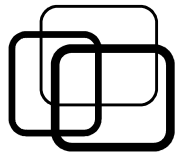
- Object sharing:
  - Each object has its own:
    - Attributes.
    - Methods.

    ➔ Object members.
  - How to share information with other objects?

    ➔ Static members.



Static members

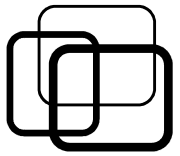


- Static members:
  - Class-level attributes and methods.
  - Shared among objects of same class.
  - Usage in C++:
    - Keyword **"static"**.
    - Initialization: outside class.
    - Use **::** to access.

```
class Fraction
{
private:
        static    int m_maxValue;
public:
        static    int getMaxValue();
};
```

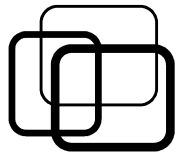
```
int Fraction::m_maxValue = 10000;

int main()
{
    int   x = Fraction::getMaxValue();
}
```

Contents



- Constructors.
- Destructor.
- Static members.
- **Class template.**

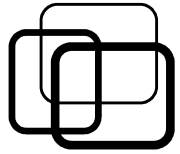
Class template



- Consider class Array:
  - Elements of integer.
  - Abstraction: elements of any type.
    - Parameterize type of elements.
    - Parameterize prototype of methods related to elements.

    ➔ Class Template.

Class template

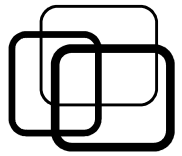


- Class template usage:

```
template <class T>
class Array
{
private:
        T   *m_data;
        int   m_size;
public:
        Array( int size );
        T&  getElement( int  i );
        T  findMax( );
};
```

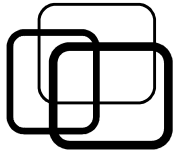
```
int main()
{
        Array<int>          m1( 10 );
        int   a = m1.getElement( 5 );
        int   max1 = m1.findMax( );

        Array<Fraction> m2( 5 );
        Fraction   p = m2.getElement( 2 );
        Fraction   max2 = m2.findMax( );
}
```

Summary



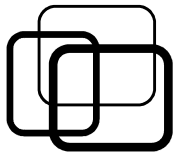
- Constructors:
  - “Birth certificate” of object.
  - Mandatory when declaring object.
  - Can be overloaded.
- Destructor:
  - “Living will” of obejct.
  - Automatically called when deposing object.
  - Has only one.

Summary



- Static members:
  - Attributes/methods shared among objects.
  - Keyword “static”.
- Class Template:
  - Parameterization class attributes and types.

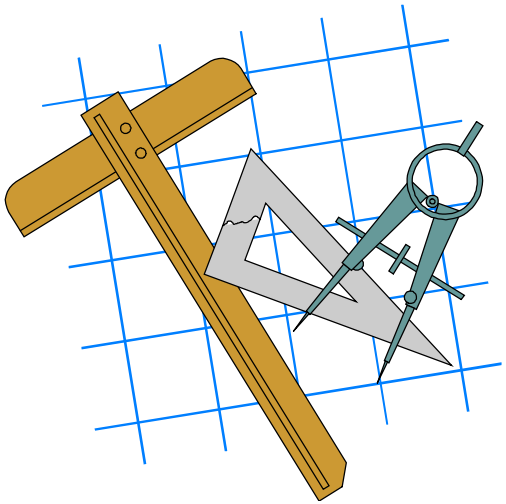
Practice



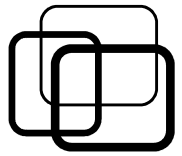
- Practice 3.1:

  Provide class **Fraction** with the following constructors to:

  - Default initialize a fraction = 0.
  - Initialize a fraction with num and den.
  - Initialize a fraction = integer value.
  - Initialize a fraction from another fraction.



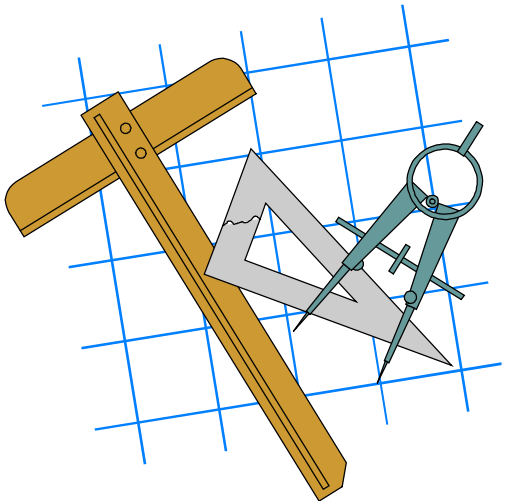
Practice



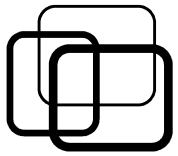
- Practice 3.2:

  Construct class **Student** with constructors and destructor to:

  - Initialize a student with name, math, literature points.
  - Initialize a student with name, math = literature = 0.
  - Initialize a student from another student.
  - Dispose a student without memory leak.



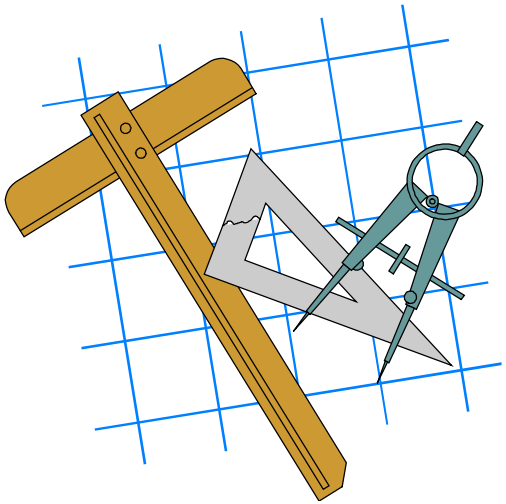
Practice



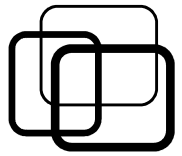
- Practice 3.3:

  Construct class **Array** (of integers) with the following constructors and destructor to:

  - Default initialize an array with zero-length.
  - Initialize an array with length, all elements = 0.
  - Initialize an array with int [ ] and length.
  - Initialize an array from another array.
  - Depose an array without memory leak.

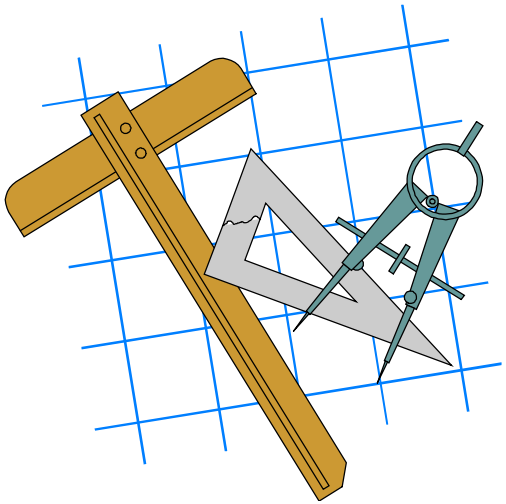


Practice

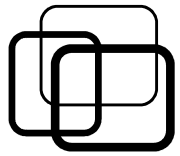


- ## Practice 3.4:

  Based on practice 3.3 construct class **Array** with elements of any type.



Practice



- Practice 3.5 (*):

  Provide class **Fraction** an ability to count number of fraction objects created in main( ) function.