Classes & Objects

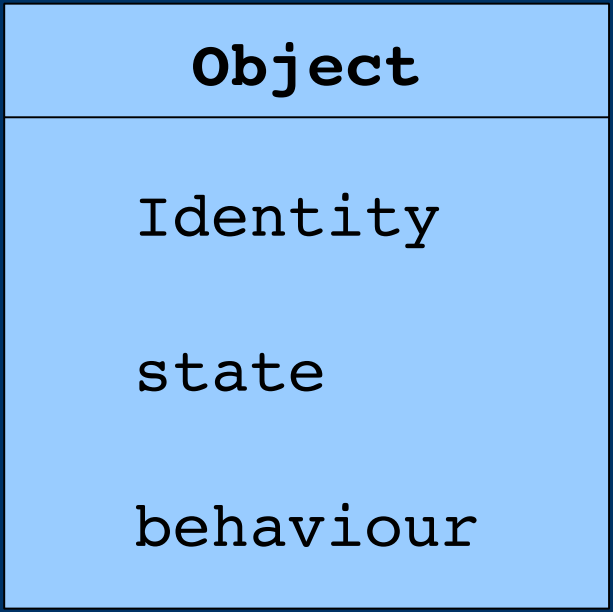
hncharif@bournemouth.ac.uk

nccastaff.bournemouth.ac.uk/hncharif/CA2

What is an Object

- A tangible or visible 'thing' e.g. a person
- 'An object represents an individual identifiable item, unit or entity, either real or abstract, with a well defined role in the problem domain'
- What Defines an Object ?
  - A boundary e.g. a car body or building exterior
  - An internal state e.g. Blood pressure, waiting time
  - Behavior -- what it does
  - A Unique identity (as with instances, entities or roles)

What is an Object

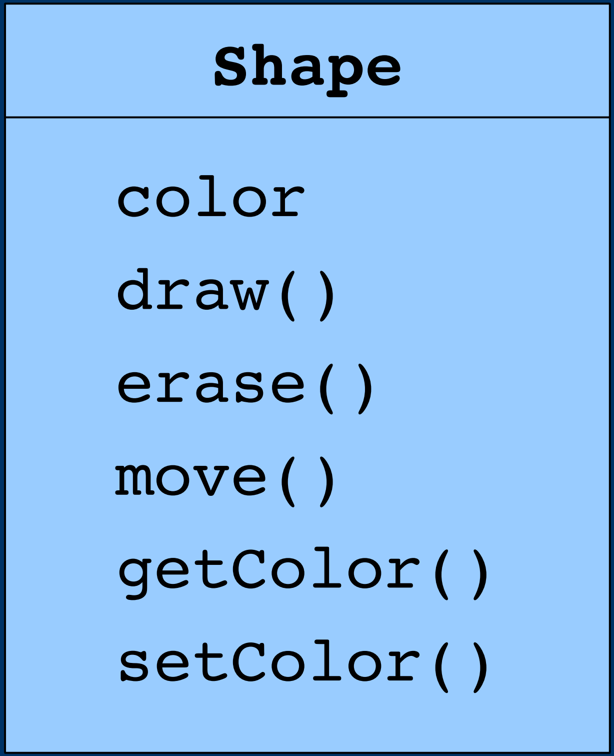


Object

Identity

state

behaviour



Shape

color

draw()

erase()

move()

getColor()

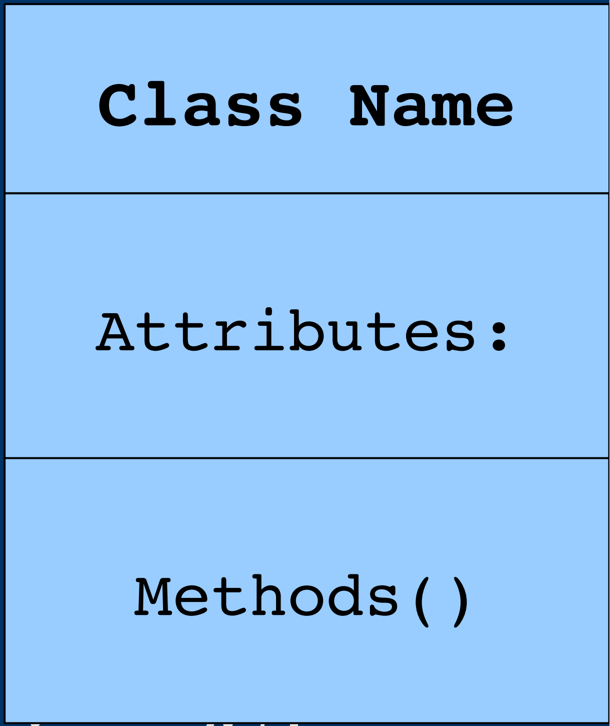
setColor()

Classes

- The structure data type can be used in both C and C++ However it is only used to store data

- In C++ we store the data and the operation that can be performed on that data together in the same entity.
- This entity is know as a class
- We need the following C++ keywords
  - **class** keyword to define a class (ADT)
  - **private** keyword to define 'hidden' parts of the class
  - **public** keyword to define the visible interface of the class
  - The scope resolution operator (::) used to link together classes and methods

The Anatomy of a class



Class Name

Attributes:

Methods()

- A Class has two parts
  - A private (hidden) part
  - A public interface
- The public part defines the behavior of the object (methods)
- The private part contains the data (attributes)
- It is normal practice to put attributes in the private part of the class where they can only be accessed by methods

A Class Example

```
#ifndef __Colour_H_
#define __Colour_H_
class Colour
{
 private :
   float m_r;
   float m_g;
   float m_b;
   float m_a;
 Public:
     inline void SetRed(float _r)
      {
          m_r=_r
      }
     inline float GetRed()
      {
            return m_r;
      }
};  //end of class
#endif
```

Some observations on the class

- A class definition is terminated with a semicolon (unlike a function definition)
- The members of a class are private by default, so the private keyword could be omitted.
- Methods are usually know as 'member functions' in C++.
- setValue is of void type because it doesn't return a value
- getValue returns and int (the value of i)

Classes and Objects

- A class is a 'blueprint' for all Objects of a certain type
  - class defines the attributes (data)
  - and the operations (methods)
- The class may be considered as an object factory
  - allowing us to instantiate objects of the same type.
- A single class allows us to make as many instances of a class as required

State Identity and Behavior

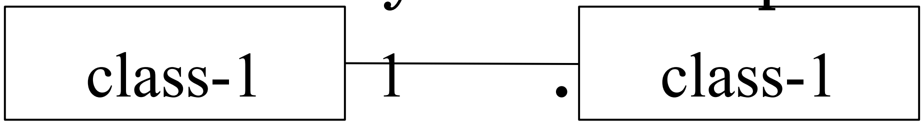
- A class is defined by the following 3 attributes
  - A unique name, Attributes and Methods
- An Object is defined by
  - Identity, State and Behavior
- The properties of the Class relate to the properties of the object
  - Class attributes == Object state data
  - Class Methods == Object behavior

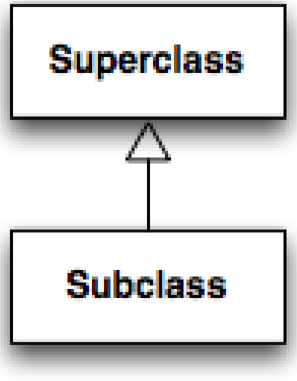
Difference between Classes and Objects

- Once defined, a class exists all the time the program is running, Objects may be created and destroyed
- For a single class there may be any number of objects instantiated at any one time
- A class has a unique name, attributes and methods. An Object has identity, state and behavior
- A class provides no state values for the object attributes. These can only be given values when applied to specific objects

Basic Class Diagrams

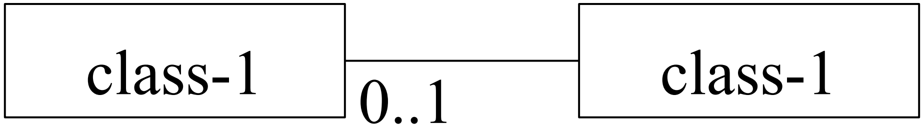
1 to many relationship

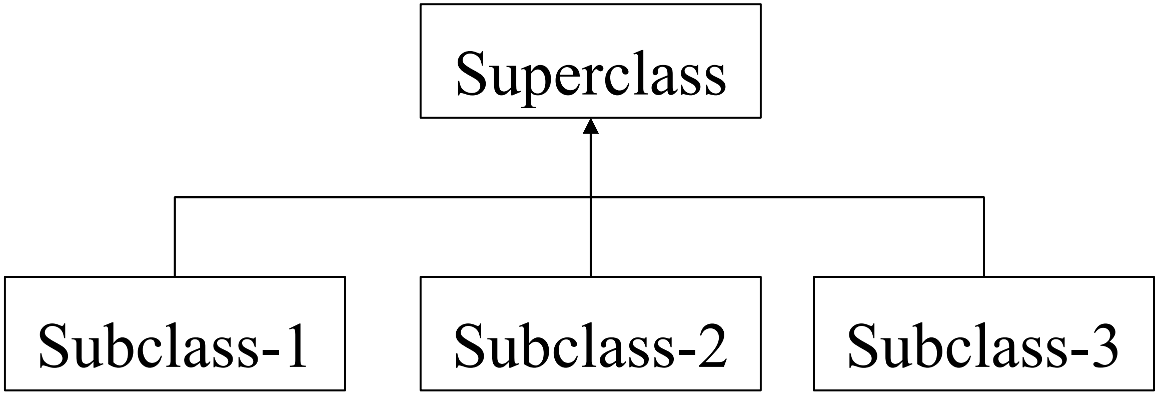




A Class hierarchy using inheritance

0 or 1 to Many relationship





Specifying Attributes & Methods

- Attributes are defined as follows

```
 [visibility symbol]   name   :     [data type]
```

- For methods we use the syntax

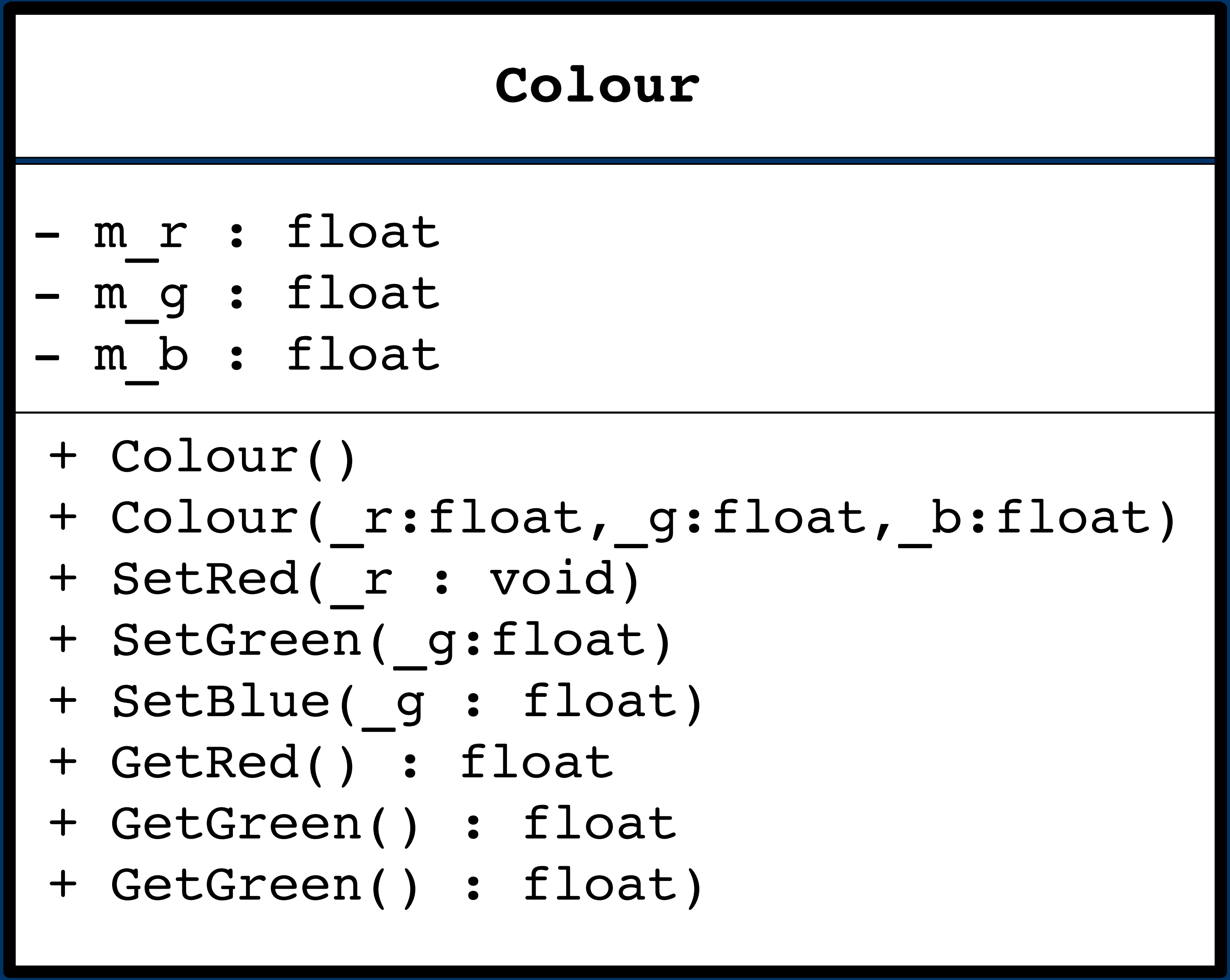
```
[visibility symbol] name ([param]): [return type]
```

- The visibility symbols are.

scope symbol	meaning
-	Private
+	Public
#	Protected

An Example: Class Diagram

- We can now generate a class diagram



```
Colour

- m_r : float
- m_g : float
- m_b : float

+ Colour()
+ Colour(_r:float,_g:float,_b:float)
+ SetRed(_r : void)
+ SetGreen(_g:float)
+ SetBlue(_g : float)
+ GetRed() : float
+ GetGreen() : float
+ GetGreen() : float)
```

Single File Instruction

- header files define interfaces for functions, structures and classes
- They may also define variables, however if this header file is then included in more than one module the linker will complain as the same variables may be defined twice.
- To overcome this problem we can use Single File Inclusion
- The traditional way of doing this is as follows
  - Use the name of the Header file as an identifier for the pre-processor and create a unique **`#define`** name
  - place a **`#include`** directive in the code to define the module name
  - If the **`#define`** does not exist then it will be included if it does there is no need to include it again.

A Class Example

```
#ifndef __Colour_H_
#define __Colour_H_
class Colour
{
 private :
   float m_r;
   float m_g;
   float m_b;
   float m_a;
 Public:
      inline void SetRed(float _r)
       {
          m_r=_r
       }
      inline float GetRed()
       {
             return m_r;
       }
};  //end of class
#endif
```

Some observations on the class

- A class definition is terminated with a semicolon (unlike a function definition)
- The members of a class are private by default, so the private keyword could be omitted. The float variables m_**r**,m_g m_g and m_a are private hence inaccessible from outside the class, except indirectly via the methods setRed getRed
- Methods are usually know as 'member functions' in C++. The member functions 'setRed' and 'getRed' are defined 'inline' (inside the class definition).
- setRed is of void type because it doesn't return a value
- getRed returns and int (the value of i)

More on declaring Methods

- In the previous example the inline pre-fix is used to tell the compiler that the Colour constructor methods are part of the header file
- Inline methods can have a major speed advantage over non inlined methods
- Most accessor and mutator methods are defined in this way
- Most modern compilers will only allow small '**`inline`**' functions

Classes

- It is standard practice to split the class into two separate files.
- A .h Header file is used to define the class and prototype the methods and data for this class.
- A .cpp file is used to contain the actual class code and algorithmic elements.
- To link these two elements together we need to tell the compiler which class the methods in the .cpp file belong to.

C++ Scope Resolution Operator ::

- The :: (scope resolution) operator is used to qualify hidden names so that you can still use them.
- This is how C++ allows us to have different classes with the same member function names (polymorphism)
- We use the :: to imply membership to a particular class and differentiate the different methods / class relationships

Constructors

Colour.h

```
#ifndef __Colour_H_
#define __Colour_H_
class Colour
{
 private :
    float m_r;
   float m_g;
   float m_b;
   float m_a;
 Public:
   Colour();
   Colour(float,float,float,float);
   void SetRed(float _r);
    float GetRed();

}; //end of class
#endif
```

Colour.cpp

```
#include “Colour.h”
Colour::SetRed(float _r)
      {
          m_r=_r;
      }
float Color::GetRed()
      {
            return m_r;
      }
```

More on Scope Resolution

- In the above example the methods are declared as part of the class
  - however there is no code defined in the class
- The methods are then prototyped after the class definition using the following

-
  ```
  return_type Class_name::method_name(parameterlist)
  ```
- This has the effect that only one version of the method exists at run-time
  - This method is available to all instances of objects of this class
  - Thus saving memory and improving efficiency

Constructors

- When an object is created there are certain processes which must take place
- Instantiation always involves the allocation of memory for the objects state data.
- The methods do not require any memory as these are consistent for all objects of the class and are handled in the class itself.
- The special method which allocates the memory for an object is know as the 'constructor'
- There are three basic types of constructor
  - The default constructor
  - User defined Constructor
  - The Copy Constructor

Coding a Constructor

- A C++ constructor has three important aspects
  - It takes the same name as the class
  - It may take arguments
  - It cannot return a value
- For example the Colour Class constructor would be

```
class Colour
{
................
 Public:
   Colour();
   Colour(float,float,float,float);
...........

};
```

```
#include “Colour.h”
Colour::Colour()
{
    m_r = 0.0;
    m_g = 0.0;
    m_b = 0.0;
    m_a = 0.0;
}
```

The Default Constructor

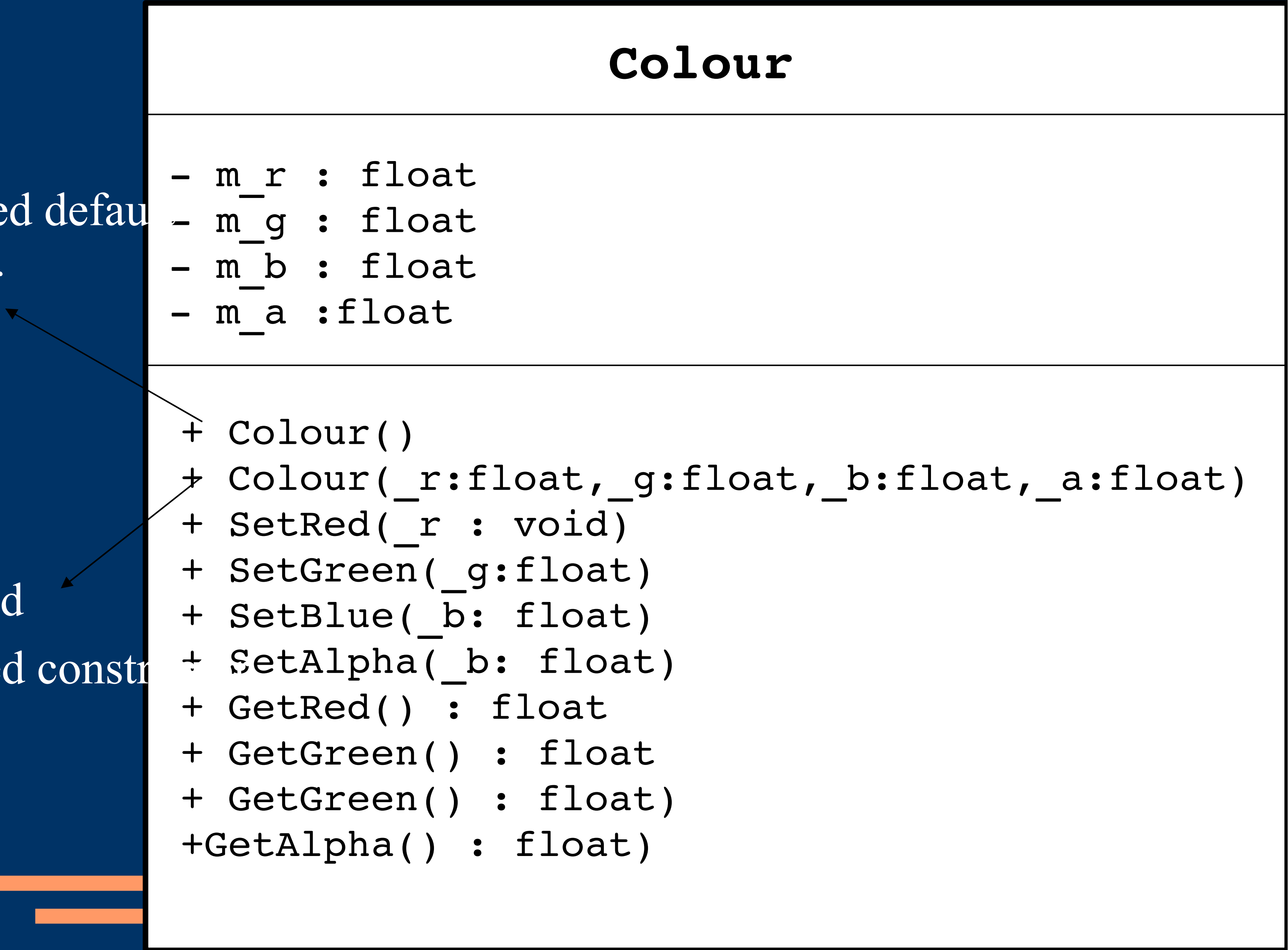
- The default constructor takes no parameters (or return type)
- It performs no processing on the object data just memory allocation
- It will always be called by the compiler if no user defined constructor is provided
- The default constructor is not referred to by the programmer in the class definition

User Defined Constructor

- These constructors may be used to pass parameter values to the object
- These may be used to set default object values
- It is possible to have more than one constructor in a class passing different parameters
- This is known as overloading and gives more flexibility to the way the object can be instantiated

An Example: Class Diagram

- We can now generate a class diagram



User defined defau constructor

User defined parametrised constr

```
Colour

- m_r : float
- m_g : float
- m_b : float
- m_a :float

+ Colour()
+ Colour(_r:float,_g:float,_b:float,_a:float)
+ SetRed(_r : void)
+ SetGreen(_g:float)
+ SetBlue(_b: float)
+ SetAlpha(_b: float)
+ GetRed() : float
+ GetGreen() : float
+ GetGreen() : float)
+GetAlpha() : float)
```

Constructors

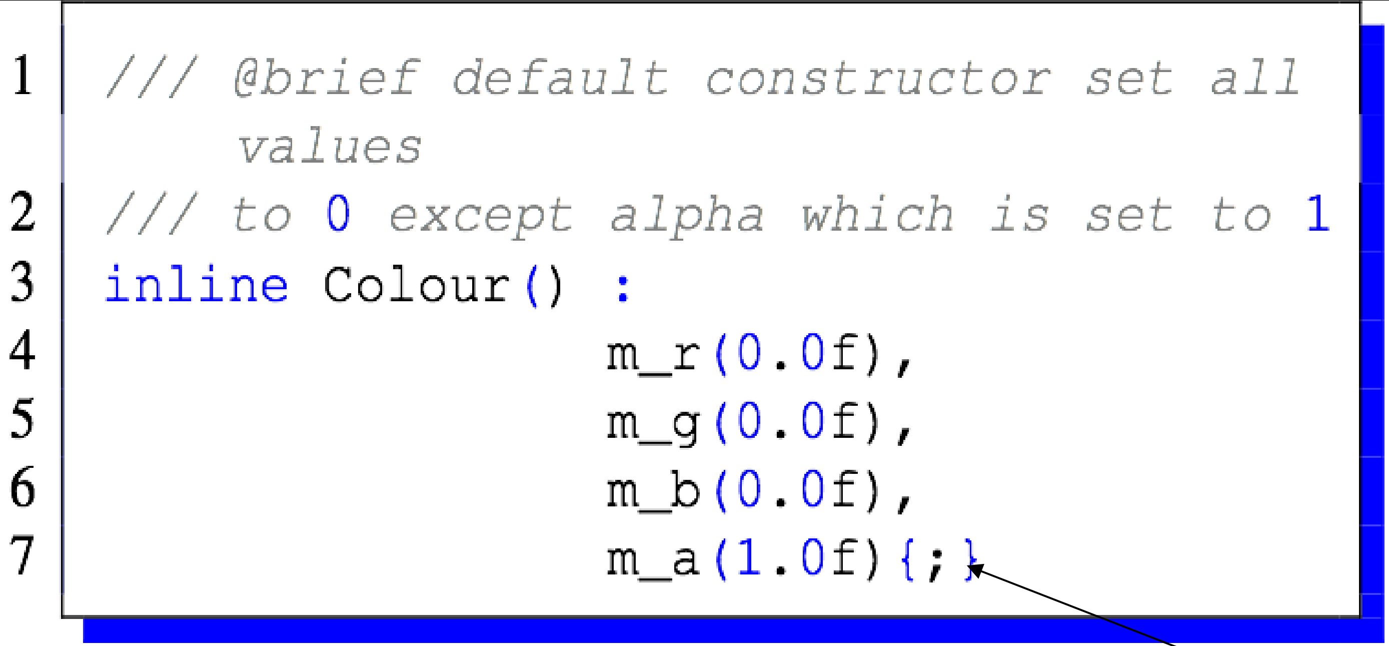
Colour.h

```
#ifndef __Colour_H_
#define __Colour_H_
class Colour
{
 private :
    float m_r;
    float m_g;
    float m_b;
    float m_a;
 Public:
    Colour();
    colour(float,float,float,float);
    void SetRed(float _r);
    float GetRed();

};  //end of class
#endif
```

Colour.cpp

```
#include “Colour.h”

Colour::Colour( )
       {
          m_r=0.0;
          m_g=0.0;
          m_b=0.0;
          m_a=0.0;
       }
Colour::Colour(folat _r,
               Float _g,
               Float _b,
               Float _a)
       {
            m_r = _r;
            m_g = _g;
            m_b = _b;
            m_a = _a;
       }
```

User Defined Default Contractor

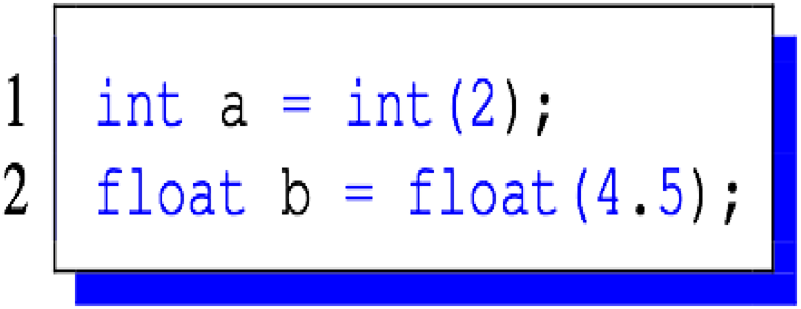


```cpp
/// @brief default constructor set all values
/// to 0 except alpha which is set to 1
inline Colour() :
                    m_r(0.0f),
                    m_g(0.0f),
                    m_b(0.0f),
                    m_a(1.0f){;}
```

Sete ach of the attributes to a default value by calling its own constructor

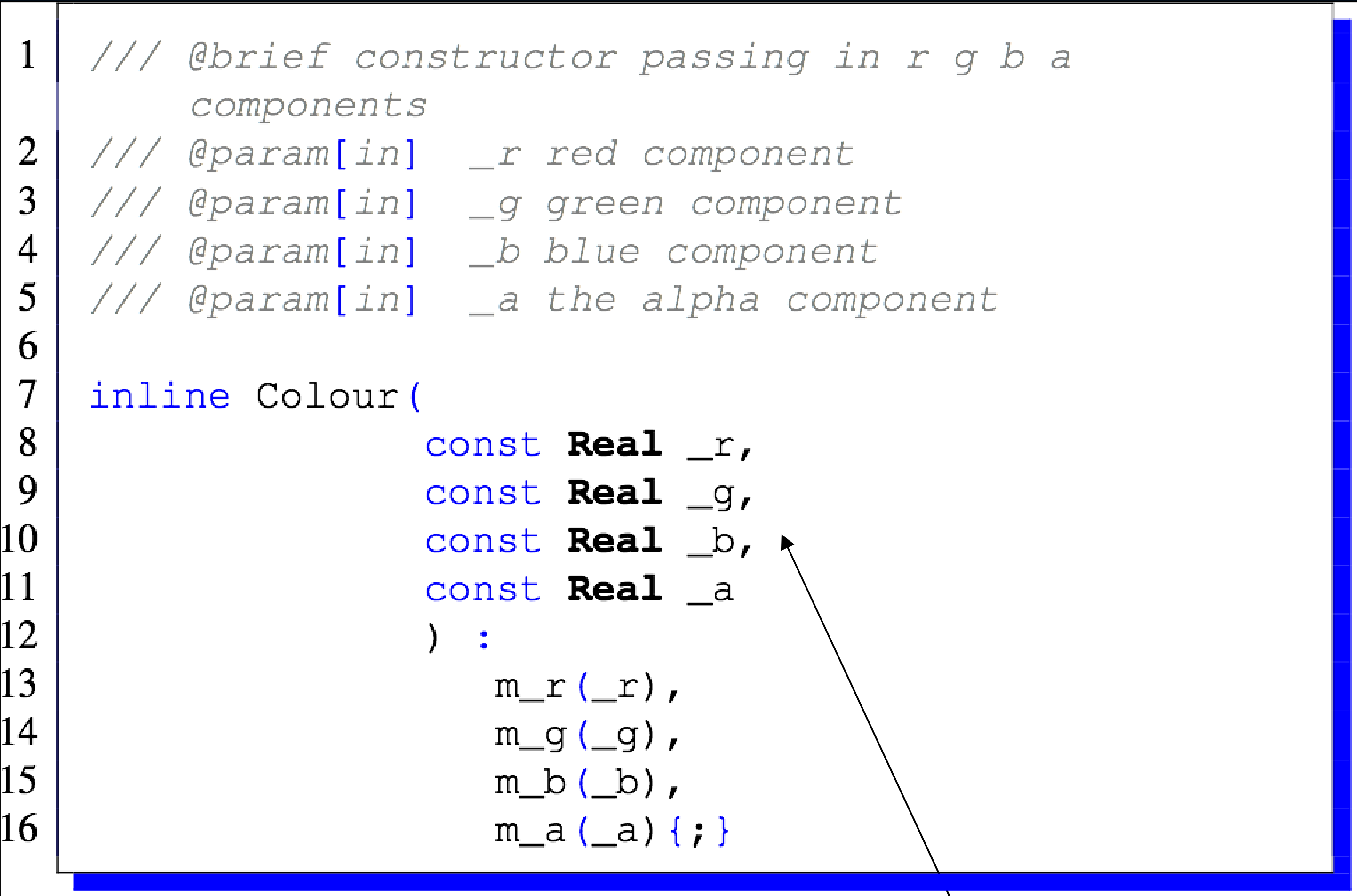
No other code

Float has a constructor as doall C++ data types



```cpp
int a = int(2);
float b = float(4.5);
```

User Defined Contractor



```
/// @brief constructor passing in r g b a components
/// @param[in]  _r red component
/// @param[in]  _g green component
/// @param[in]  _b blue component
/// @param[in]  _a the alpha component

inline Colour(
              const Real _r,
              const Real _g,
              const Real _b,
              const Real _a
              )  :
                 m_r(_r),
                 m_g(_g),
                 m_b(_b),
                 m_a(_a){;}
```

Here we are passing in individual values to set the class attributes

The Copy Constructor

- The copy constructor allows a new object to be created as a copy of an existing object. e.g.

```
ExampleClass object2 = object1;
```

- Therefore all of the objects state data is copied to the new object
- This object will be identical to the first except for it's identity
- This remains until methods are called on the new object to change the data

User Defined Copy Constructor

- A programmer may also create a copy constructor by passing a reference to the object to be copied for example

```
Colour(const Colour &colour0);
```

- & denotes pass by reference and the const prefix indicates that the object being passed in must not be modified
- The code for the copy constructor may look like this

```
Colour::Colour(const Colour &colour0
{
   m_r = colour0.m_r;
   m_g = colour.m_g;
  m_b = colour0.m_b;
   m_a = colour.m_a;
}
```

Clean-up and Garbage Collection

- The default constructor is used to allocate memory for the creation of the Object
- The default destructor is used to de-allocate this memory
- With a static object this is done implicitly however for dynamic objects this must be done by the programmer
- This now allows us to determine exactly the life-cycle of the object by
  - being able to control both the creation and destruction of the object