SLIIT
Discover Your Future

Object Oriented Concepts

Lecture 06

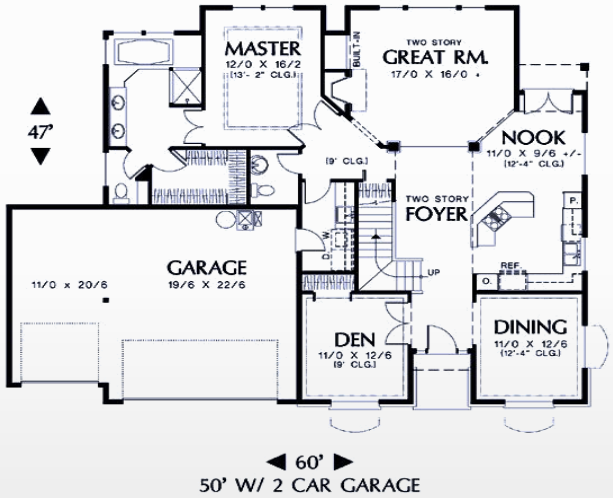
Classes in C++

Learning Outcomes

- At the end of the Lecture students should be able to
  - Have a better understanding of the differences between classes and objects (in C++ coding)
  - Use setters and getters in a Class
  - Write Object Oriented Programs
  - Use header files with classes

SLIIT
Discover Your Future

Classes and Objects



Class House
Blue Print of a House

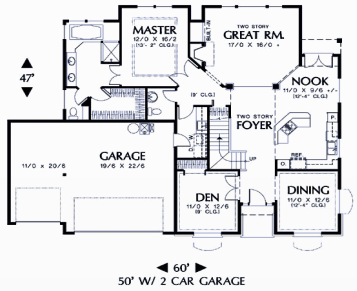
House
- length - width - height - area
+ paint()

```
class House {
  private:
     int length;
     int width;
     int height
     int area;
     ..
  public:
     void paint();
     …
}
```

SLIIT
Discover Your Future

Classes and Objects

Class




```
class House {
  private:
     int length;
     int width;
     int height
     int area;
     ..
  public:
     void paint();
    …
}
```

Objects

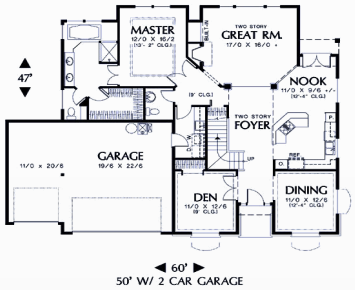
myHouse1

```
int main() {
   House myHouse1;




}
```

SLIIT
Discover Your Future

Classes and Objects

Class




```
class House {
  private:
     int length;
     int width;
     int height
     int area;
     ..
  public:
     void paint();
     …
}
```

Objects

myHouse1

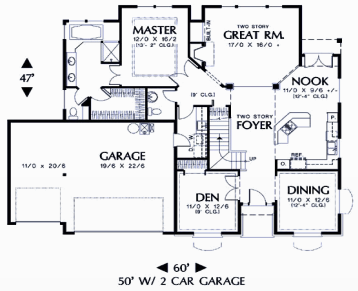
myHouse2

```
int main() {
   House myHouse1;
   House myHouse2;



}
```

SLIIT
Discover Your Future

Classes and Objects

Class




```
class House {
  private:
     int length;
     int width;
     int height
     int area;
     ..
  public:
     void paint();
     …
}
```

Objects

myHouse1

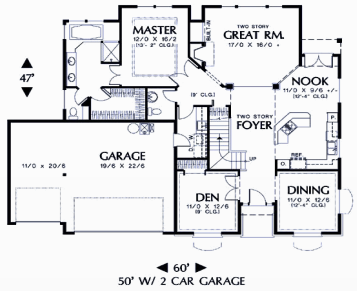
myHouse2

```
int main() {
   House myHouse1;
   House myHouse2;
   myHouse1.paint(green);


}
```

SLIIT
Discover Your Future

Classes and Objects

Class




```
class House {
  private:
     int length;
     int width;
     int height
     int area;
     ..
  public:
     void paint();
    …
}
```

Objects

myHouse1

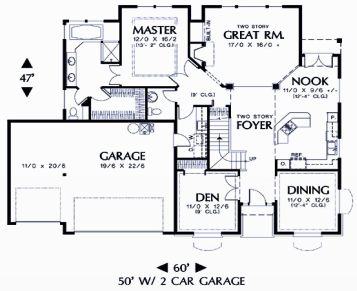
myHouse2

```
int main() {
   House myHouse1;
   House myHouse2;
   myHouse1.paint(green);
   myHouse2.paint(blue);


}
```

SLIIT
Discover Your Future

Classes and Objects

Class




```
class House {
  private:
     int length;
     int width;
     int height
     int area;
     ..
  public:
     void paint();
     …
}
```

These attributes are protected
We can't access these
in the main function()

We interact with objects using
The public methods.

Objects

myHouse1

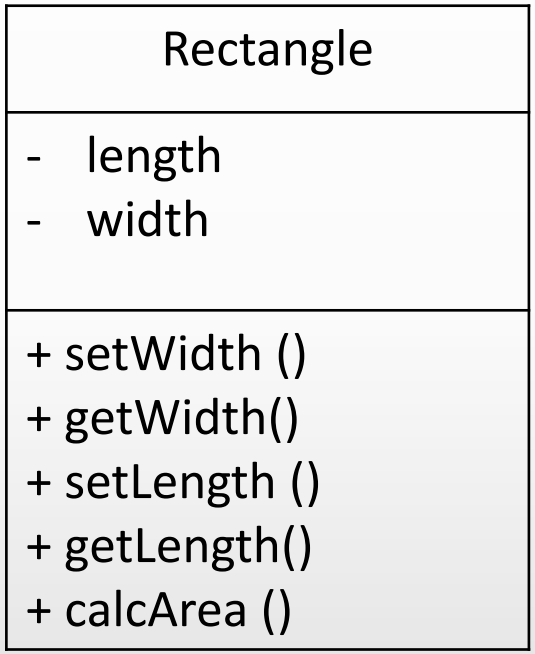
myHouse2

```
int main() {
   House myHouse1;
   House myHouse2;
   myHouse1.paint(green);
   myHouse2.paint(blue);

}
```

We use the dot Operator to
access methods.

SLIIT
Discover Your Future

Rectangle Class




```
class Rectangle {
   private:
      int width;
      int length;
   public:
      void setWidth(int w);
      int getWidth();
      void setLength(int l);
      int getLength()
      int calcArea();
};
```

The diagram on the left side is a UML (Unified Modeling Language) Class Diagram. Here – means private and + means public

Rectangle Class

Rectangle
- length - width
+ setWidth () + getWidth() + setLength () + getLength() + calcArea ()

```
class Rectangle {
   private:
     int width;
     int length;
   public:
     void setWidth(int w);
     int getWidth();
     void setLength(int l);
     int getLength()
     int calcArea();
};
```

A Setter

A Getter

Since the properties length and width are protected and cannot be accessed from the main function we usually write two methods per property. A set method (setters) and a get method (getters).

Setters (Mutators)

```
class Rectangle {
   private:
     int width;
     int length;
   public:
     void setWidth(int w);
     int getWidth();
     void setLength(int l);
     int getLength()
     int calcArea();
};
```

We can do validations
In a setter
Here we are assuming that the default width is 10.
We know a Rectangle's width cannot be zero or negative.
If someone sets a negative width The Rectangle width will be set to 10.

```
// A Setter starts with the word set
// followed by the name of the property
// e.g. setWidth()
// setters are always methods that don't
// return values (void functions)

void Rectangle::setWidth(int w) {
  if (w > 0)
     width = w;
  else
     width = 10;
}
```

Getters (Accessors)

```
class Rectangle {
   private:
     int width;
     int length;
   public:
     void setWidth(int w);
     int getWidth();
     void setLength(int l);
     int getLength()
     int calcArea();
};
```

Getters always contain the Following code.

return property;

```
// A Getter starts with the word get
// followed by the name of the proeprty
// e.g. getWidth()
// getters always have the return type of
// the property.

int Rectangle::getWidth(){
   return width;
}
```

SLIIT
Discover Your Future

Exercise - 1

StopWatch
- minute - second
+ setMinute() + getMinute() + setSecond() + getSecond() + start() + stop()

Write a setter and getter for the property minute.

Exercise – 1 – Class definition (Not part of the answer)

StopWatch
- minute - second
+ setMinute() + getMinute() + setSecond() + getSecond() + start() + stop()

```
class StopWatch {
   private :
         int minute;
         int second;
    public:
          void setMinute(int min);
           int getMinute();
           void setSecond(int sec);
           int getSecond();
           void start();
           void stop();
};
```

Setter and getters for minute property

```
void StopWatch::setMinute(int min) {
    minute= min;
}
// or with validations
void StopWatch::setMinute(int min) {
    if (min >= 0 && min <= 59)
        minute = min;
    else
         minute = 0;
 }
int StopWatch::getMinute() {
     return minute;
}
```

Implementing a Class.. Example..

Time
- hour : int - minute : int - second : int
+ setTime (h : int, m : int , s : int) : void + printTimeUniversal () : void + printTimeStandard () : void

We can represent datatypes and parameters in UML class diagrams as shown above. The datatype or return type is given after the property or method. A colan is used as a seperator

Exercise - 2

Time
- hour : int - minute : int - second : int
+ setTime (h : int, m : int , s : int) : void + printTimeUniversal () : void + printTimeStandard () : void

Implement this class. Here the time is represented in a 24 hour clock Format. Universal time is a 24 hour clock. Standard time is a 12 hour Clock, we also need to print AM, PM. In this class for simplicity we have not included setters and getters.

Time class in C++

```
class Time {
   private :
          int hours;
          int minute;
          int second;
    public:
          void setTime(int h, int m, int s);
          void printTimeUniversal();
          void printTimeStandard();
};
```

Implement methods

```
void Time::setTime(int h, int m, int s)
{
   hour = h;
   minute = m;
   second = s;
}

void Time::printTimeUniversal()
{
     cout<<setfill('0')<<setw(2) <<hour<<":"<<setw(2)
<<minute<< ":"<<setw(2)<<second;
}
```

Setfill is used to have leading zeros

```
void Time:: printTimeStandard()
{
   cout<<setfill('0')<<setw(2);
   if( hour == 0 || hour == 12)
        cout<<12;
   else
        cout<<hour%12;

    cout<<":"<<setw(2)<<minute<<":"<<setw(2)<<second;
    if ( hour < 12)
          cout << " AM" <<endl;
    else
          cout << " PM" << endl;
}
```

SLIIT
Discover Your Future

Client Program – Exercise 2

OUTPUT :

Input Hour :13
Input Minutes :27
Input seconds :6
13:27:06
01:27:06 PM

Write a main program to input values for hours, minutes and seconds in a 24 hour clock format and print the time both in universal time and standard time

Client Program

```
int main()
{
      Time t;   // static object
      int hou, min, sec;

      cout<<"Input Hour :";
      cin >> hou;
      cout<<"Input Minutes :";
      cin >> min;
      cout<<"Input seconds :";
      cin >> sec;

      t.setTime (hou, min, sec);
      t.printTimeUniversal ();
      t.printTimeStandard ();
}
```

OUTPUT :

```
Input Hour :13
Input Minutes :27
Input seconds :6
13:27:06
01:27:06 PM
```

Static Object

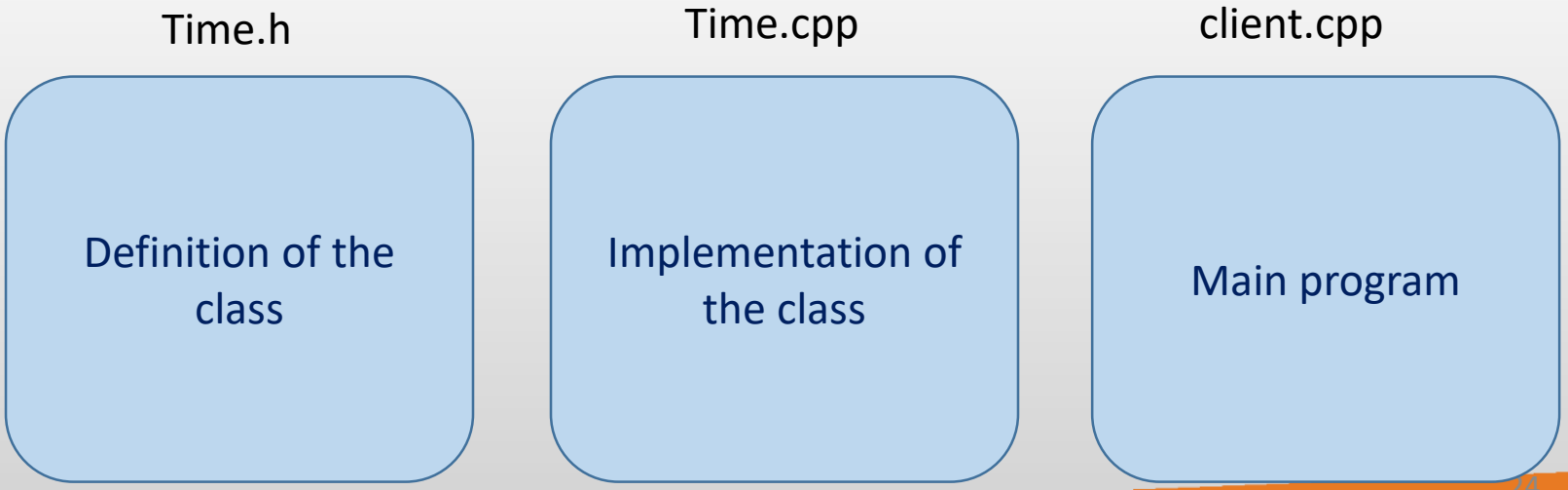
```
Time   t;
```

Methods are accessed using dot (.) operator

```
t.setTime ( 13, 27, 6);
```

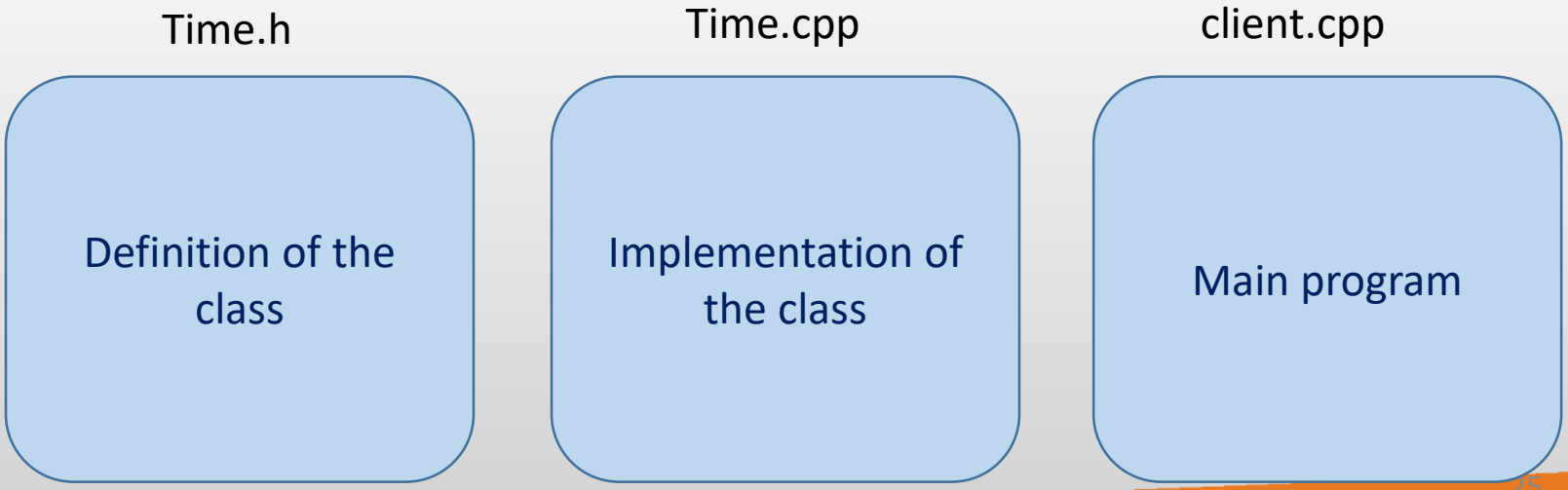
How Classes are implemented

- In C++ we generally separate each class implementation into two files.
- A Header file containing the class definitions. e.g. Time.h
- A .cpp file containing the implementation of the methods of the class e.g. Time.cpp
- The client program is the main program that is used to create objects of the classes we have previously implemented



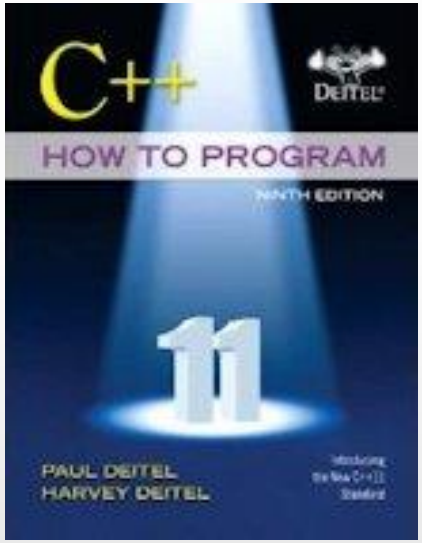
How Classes are implemented

- This approach allows us to reuse a class in many applications.
- This is a standard practice when writing C++ code.
- The header file only contain the definitions of the class, including the interfaces (public methods)



SLIIT
Discover Your Future

Reference



Chapter 09

Deitel & Deitel's (2016), C++ How to Program, 9th Edition