



SLIIT

Discover Your Future


IT1050-Object Oriented Concepts

Lecture-06 - Classes in C++

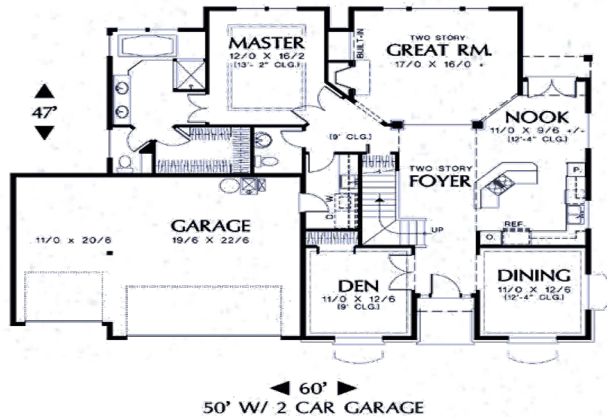




Learning Outcomes

- 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
- 

Classes and Objects



Class House
Blue Print of a House

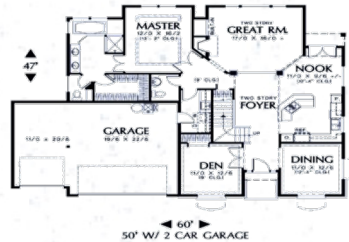
House
- length - width - height - area
+ paint()

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

Classes and Objects

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

Class



Objects



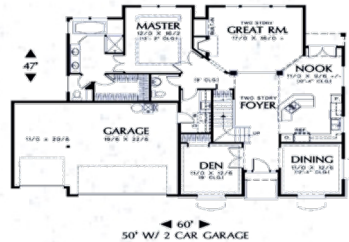
myHouse1

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

Classes and Objects

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

Class



Objects



myHouse1



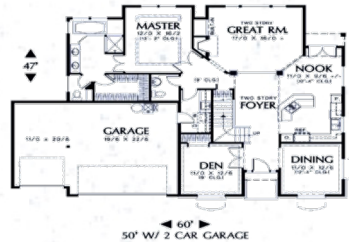
myHouse2

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

Classes and Objects

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

Class



Objects



myHouse1



myHouse2

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

Classes and Objects

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



myHouse1



myHouse2

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

Classes and Objects

```
class House {  
    private:
```

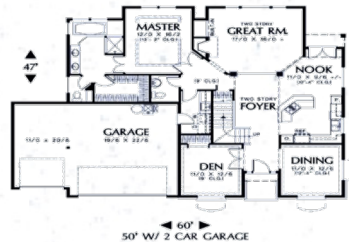
```
        int length;  
        int width;  
        int height  
        int area;
```

```
        ..
```

```
    public:
```

```
        void paint();
```

```
    ...  
}
```



Class



myHouse1



myHouse2

Objects

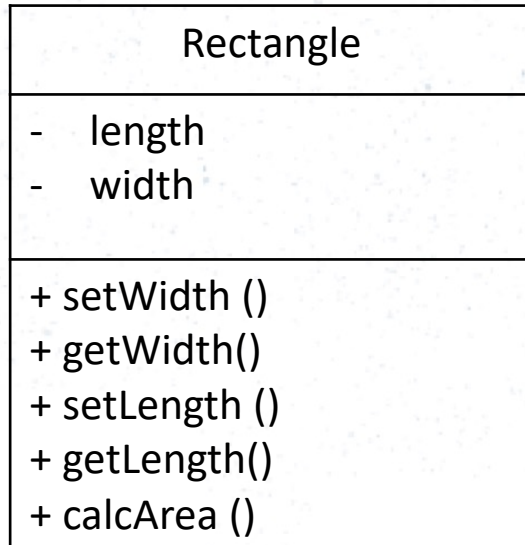
These attributes are protected
We can't access these
in the main function()

We interact with objects using
The public methods.

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

We use the dot Operator to
access methods.

Rectangle Class- private & public



```
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); ← A Setter  
        int getWidth(); ← A Getter  
        void setLength(int l);  
        int getLength()  
        int calcArea();  
};
```

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 property  
// e.g. getWidth()  
// getters always have the return type of  
// the property.
```

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

Activity - 1

Item
<ul style="list-style-type: none">- itemCode- Name- price
<ul style="list-style-type: none">+ setItemDetails()+ setPrice()+ getItemCode()+ getPrice()

- Write a class for Item and implement the setters and getters for the class.

Activity – 1 – Class definition

Item
<ul style="list-style-type: none">- itemCode- name- price
<ul style="list-style-type: none">+ setItemDetails()+ setPrice()+ getItemCode()+ getPrice()

```
class Item {  
    private :  
        int itemCode;  
        char name[20];  
        float price;  
    public:  
        void setItemDetails(int no, char pName[]);  
        void setPrice(float pPrice);  
        int getItemCode();  
        float getPrice();  
};
```

Setters of the item class

```
void Item::setItemDetails(int no, char pName[])
{
    itemCode = no;
    strcpy(name, pName);
}

void Item::setPrice(float pPrice)
{
    price= pPrice;
}
```

Getters of the item class

```
int Item::getItemCode()
{
    return itemCode;
}
```

```
float Item::getPrice()
{
    return price;
}
```



Activity- 2

- Write a client program to input the itemCode, name and price of an item and print the itemCode and price.



Activity-2 – Client Program

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
    Item itm1;

    int i_code;
    char i_name[20];
    float i_price;

    cout << "Input Item code : ";
    cin >> i_code;
    cout << "Input Item Name : ";
    cin >> i_name;
    cout << "Input Item price : ";
    cin >> i_price;

    itm1.setItemDetails(i_code, i_name);
    itm1.setPrice(i_price);

    cout << "Item Code : " << itm1.getItemCode() << endl;
    cout << "Item price : " << itm1.getPrice() << endl;

    return 0;
}
```

Activity - 3

Implement a class to set the day, month, year and print the date.

Eg : 11/08/2020

Date
<ul style="list-style-type: none">- day : int- month : int- year : int
<ul style="list-style-type: none">+ setDay (d : int) : void+ setMonth(m : int) : void+ setYear(y : int) : void+ getDay() : int+ getMonth() : int+ getYear() : int

Activity-3

Date
<ul style="list-style-type: none">- day : int- month : int- year : int
<ul style="list-style-type: none">+ setDay (d : int) : void+ setMonth(m : int) : void+ setYear(y : int) : void+ getDay() : int+ getMonth() : int+ getYear() : int

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 colon is used as a separator

```
class Date {  
    private :  
        int day;  
        int month;  
        int year;  
    public:  
        void setDay(int d);  
        void setMonth( int m);  
        void setYear( int y);  
        int getDay();  
        int getMonth();  
        int getYear();  
};
```

Date class in C++

Implement methods

Setters

```
void Date::setDay(int d)
{
    day = d;
}
void Date::setMonth(int m)
{
    month = m;
}
void Date::setYear( int y)
{
    year = y;
}
```

Getters

```
int Date::getDay()
{
    return day;
}
int Date::getMonth()
{
    return month;
}

int Date: getYear()
{
    return year;
}
```

Client Program – Activity-3

OUTPUT :

Input Day : 11

Input month : 08

Input year : 2020

Date : 11/08/2020

Write a main program to input values for day, month and year and print the date in the given format.

Client Program

```
int main()
{
    Date d1; // static object
    int d_day, d_month, d_year;

    cout<<"Input Day:";
    cin >> d_day;
    cout<<"Input Month :";
    cin >> d_month;
    cout<<"Input Year:";
    cin >> d_year;

    d1.setDay(d_day);
    d1.setMonth( d_month);
    d1.setYear(d_year);
}
```

Client Program

```
cout<<d1.getDay()<<"/"<<d1.getMonth()<<"/"<<getYear()<<endl;  
  
return 0;  
  
} // end main
```



Static Object

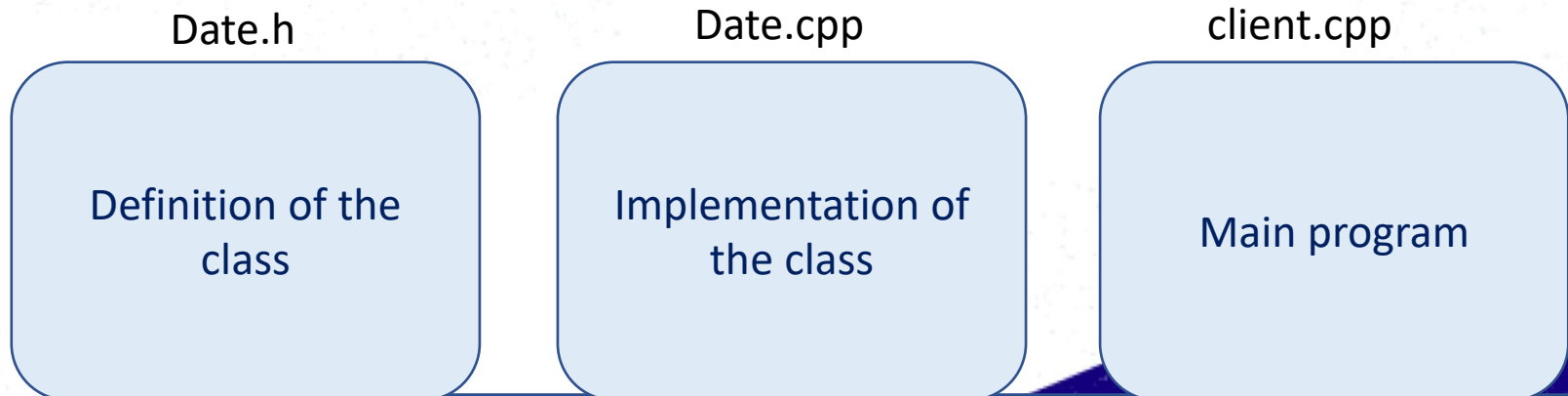
```
Date d1;
```

Methods are accessed using dot (.) operator

```
d1.setDay (11);
```

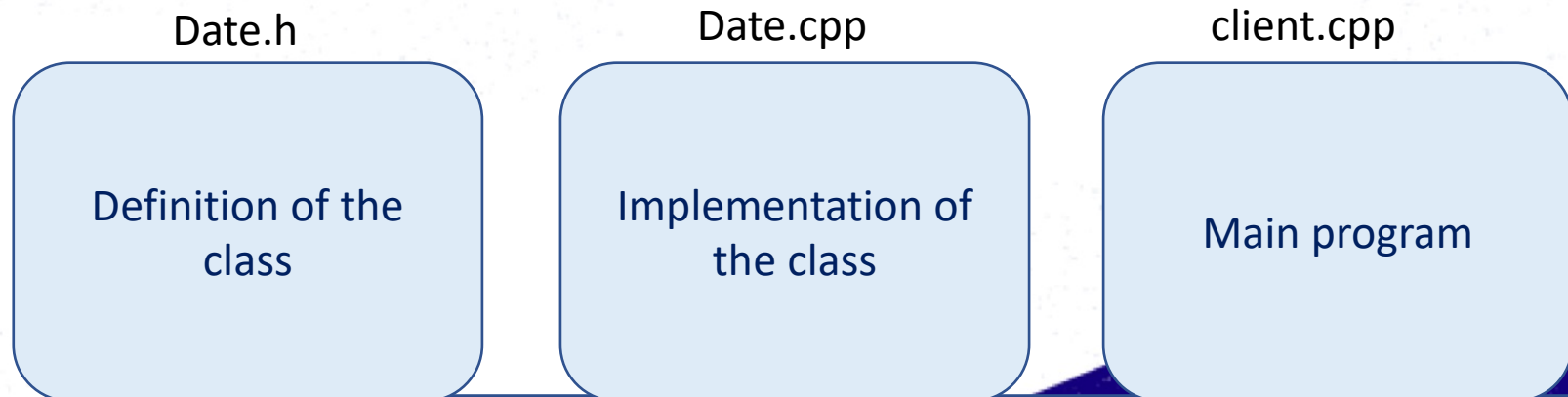
How Classes are implemented

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

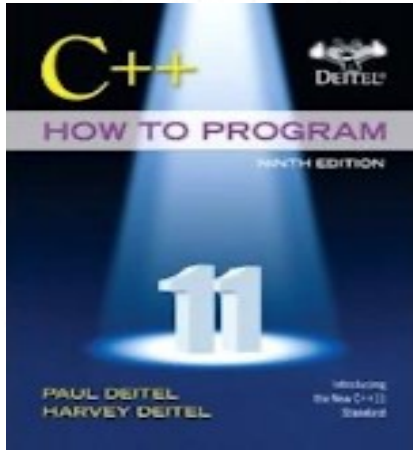


How Classes are implemented

- 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)



Reference



Chapter 03

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