Object Oriented Programming
C++ Class and Member Functions

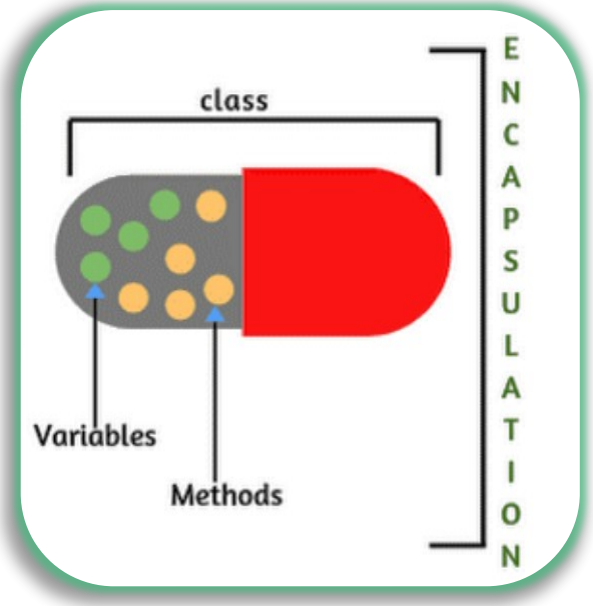
CS(217) Object Oriented Programming

Abeeda Akram

Object Oriented Programming

Implementation of **Abstract Data Type (ADT)**

1. Select a concrete data representation **using built-in data types**
2. ***Implement all relevant functions***



C++ Class (**class** is reserve word in C++)

- Class is used to only ***define*** new data types.
- It is collection of
  - Data called **data members** or **attributes**
  - Functions called **member functions**, **methods** or **behaviors**

Class Member access specifiers

`private:` (reserve word in C++)

- Class members accessible only to member functions of class
- Not accessible outside class

`public:` (reserve word in C++)

- Class members accessible to member functions of class
- Also accessible outside class

`protected:` (reserve word in C++)

- Class members Accessible to member functions and derived classes **(will use and discuss later on)**

By default class member access is private, if no access specifier is mentioned

```
class Point
{
    private:
        int x;
        int y;
};
```

```
class Point
{
    public:
        int x;
        int y;
};
```

```
class Point
{
        int x;
        int y;
};
```

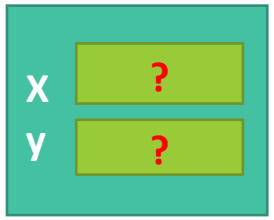
Where is Object Encapsulation?

- **Information** (**Data**) and implementation hiding

```
void main(){
    Point p;

    cout << p.x;
    //object variable name dot member name
    cout << p.y;
    //object variable name dot member name

    p.x = 100;
    cin>>p.y;
    //initialize members
}
```



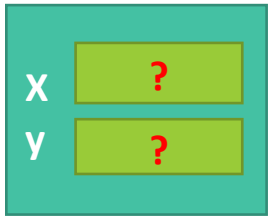
```
class Point
{
   public:
      int x;
      int y;
};
```

Object Encapsulation

- **Information** (**Data**) and implementation hiding

```
void main(){
    Point p;
    cout << p.x;
    //Compiler Error cannot access private member
    outside
    cout << p.y;
    //Compiler Error cannot access private member
    outside

    p.x = 100;  cin>>p.y;
    //Compiler Error cannot access private member
    outside
}
```

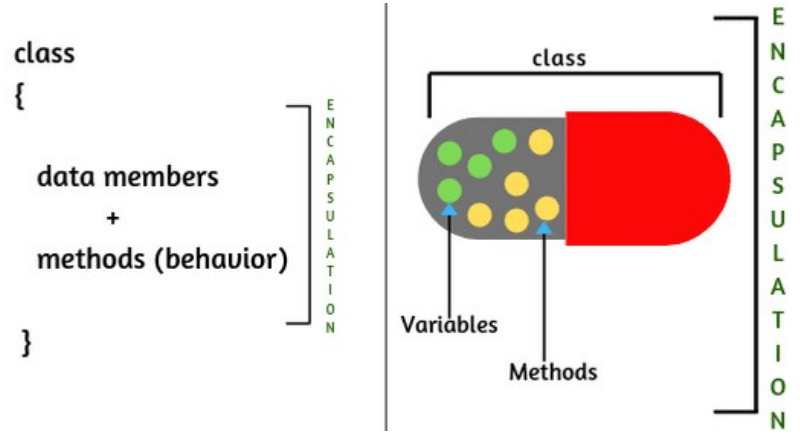



```
class Point
{
        int x;
        int y;
};
```

```
class Point
{
    private:
        int x;
        int y;
};
```

How to store and process the data of objects

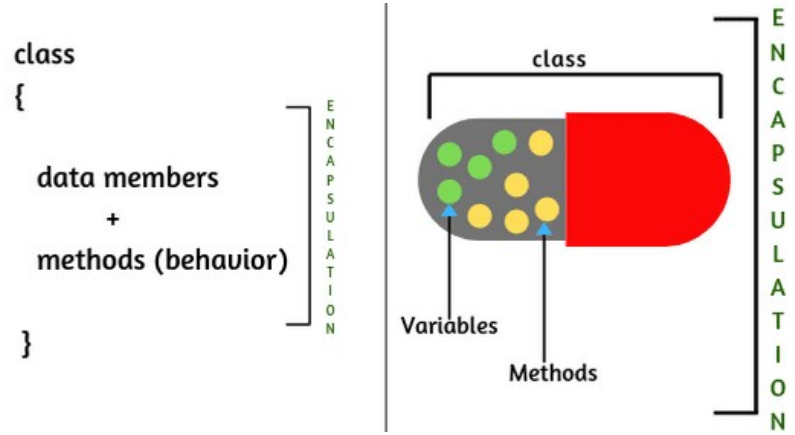
Class Member Functions



- Provide Information hiding and safe processing
- Can directly access all other members of class
  1. Data members / no need to pass in parameters
  2. Implement all member functions before use
- For all member functions only **one copy** is created at class level
  - That copy is used by all objects.
- Member access specifier can be **public** or **private**
  - Should be **public to access and call them from outside**.
  - Should be **private** to make some hidden functions.

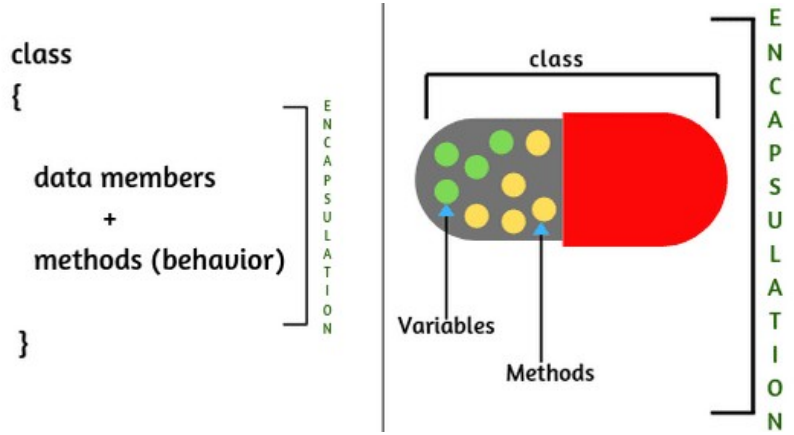
Class Member Functions (TYPES)

1. Setters
2. Getters
3. Mutators or Transformers
4. Accessors or Observers
5. Constructors
6. Destructors
7. Operators
8. Iterators



C++ Class

```
class class-name
{
   //declaration statements here
   //data members defined only not initialized
   //add member functions prototype or complete implementation
};
```




```
class Point
{
    int x;
    int y;
};
```

```
class myTime
{
    int sec;
    int min;
    int hour;
};
```

```
class myDate
{
    int day;
    int month;
    int year;
};
```

```
class Student
{
    int rollNum;
    int courses;
    float marks;
    char name[20];
};
```

Member functions (Setters)

- Used to set or update values of individual **data members** or **complete Object**

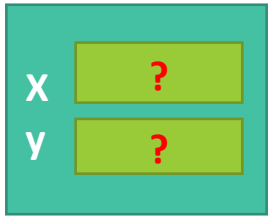
```
class Point {
        int x;
        int y;
    public:
     void setX(int xi){ x = xi;}
     void setY(int yi){ y = yi;}
     void setPoint(int xi, int yi){
        x = xi;
        y = yi;
    }
    void setPoint(Point p){
        x = p.x;
        y = p.y;
    }
    //add member functions complete
    implementation
};
```

```
void main(){
    Point p;
    //Objects created but not initialized

    p.setX(5);
    p.setY(5);
    p.setPoint(5,8);

    Point p2;
    p2.setPoint(p);

}
```



Member functions (Getters)

- Used to fetch values of individual **data members** or **complete Object**

```
class Point {
        int x;
        int y;
    public:
    //setters
     void setX(int xi){ x = xi;}
     void setY(int yi){ y = yi;}
    //getters
     int getX(){ return x;}
     int getY(){ return Y;}
     void getPoint(int & xi, int & yi){
        xi = x;
        yi = y;
    }
    //add member functions complete implementation
};
```

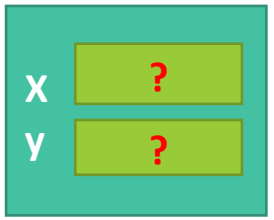
```
void main(){
    Point p;

    p.setPoint(5,8);

    cout << p.getX();
    cout << p.getY();

    int a, b;
    cout << p. getPoint(a, b);
    cout << a << b;

}
```



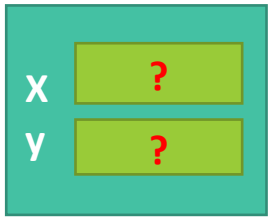
Member functions (Setters and this pointer)

- Used to set or update values of individual **data members** or **complete Object**

```
class Point
{
        int x;
        int y;
    public:
     void setX(int x){ this->x = x;}
     void setY(int y){ this->y = y;}
     void setPoint(int x, int y){
        this->x = x;
        this->y = y;
    }
    void setPoint(Point p){
        x = p.x;
        y = p.y;
    }
    //add member functions complete
    implementation
};
```

```
void main(){
    Point p;

    p.setX(5);
    p.setY(5);
    p.setPoint(5,8);
}
```



this is a pointer

- Single copy of this pointer is maintained at class level accessible in member functions only.

1. **Used when to keep same member functions parameters name as data members.**

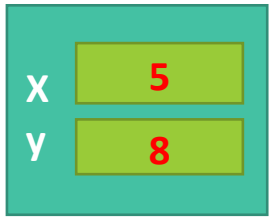
Member functions (Getters and this pointer)

- Used to fetch values of individual **data members** or **complete Object**

```
class Point
{
        int x;
        int y;
    public:
    //setters
     void setX(int xi){ x = xi;}
     void setY(int yi){ y = yi;}
    //getters
     int getX(){ return x;}
     int getY(){ return Y;}
     Point getPoint(){return *this;}
    //add member functions complete implementation
};
```

```
void main(){
    Point p;
    p.setPoint(5,8);

    cout << p.getX();
    cout << p.getY();
    Point p2 = p.getPoint();
}
```



this is a pointer

- Single copy of this pointer is maintained at class level accessible in member functions only.

2. **Used when to return complete object from member functions.**

Member functions (Setters)

- Implement outside the class

```
class Point
{
        int x;
        int y;
    public:
     int setX();
     int setY();
     void setPoint(int x, int y);
     void setPoint(Point p);
    //add member functions prototype

};
```

```
//implement outside class definition
void setX(int x){ this->x = x;}
//Compiler Error undefined variable this

void setY(int y){ this->y = y;}
//Compiler Error undefined variable this

void setPoint(int x, int y){
        this->x = x;
        this->y = y;
}
//Compiler Error undefined variable this
void setPoint(Point p){
    x = p.x;
    y = p.y;
}
//How to resolve membership issue?
```

Member functions (Getters)

- Implement outside the class

```
class Point
{
        int x;
        int y;
    public:
     int getX();
     int getY();
     Point getPoint();
    //add member functions prototype

};
```

```
//implement outside class definition
int getX(){ return x;}
//Compiler Error undefined variable x

int getY(){ return Y;}
//Compiler Error undefined variable Y

Point getPoint(){return *this;}
//Compiler Error undefined variable this

//How to resolve membership issue?
```

Class Members Scope resolution

- Use Scope resolution operator **(::)**

  Class name :: class member name

```
class Point
{
        int x;
        int y;
    public:
     int setX();
     int setY();
     void setPoint(int x, int y);
     void setPoint(Point p);

    //add member functions prototype

};
```

```
//implement outside class definition
void Point:: setX(int x){ this->x = x;}

void Point:: setY(int y){ this->y = y;}

void Point:: setPoint(int x, int y){
        this->x = x;
        this->y = y;
}

void Point:: setPoint(Point p){
    x = p.x;
    y = p.y;
}
```

Class Members Scope resolution

- Use Scope resolution operator **(::)**

  Class name :: class member name

```
class Point
{
        int x;
        int y;
    public:
     int getX();
     int getY();
     Point getPoint();
    //add member functions prototype

};
```

```
//implement outside class definition
int Point::getX(){ return x;}

int Point::getY(){ return y;}

Point Point::getPoint(){
    return *this;
}
```

Member functions (Mutators)

- Mutator functions can
  - **read** data members
  - **write** into class data members
- **Example:**
  - Setters are also Mutators

Member functions (Accessors)

- Accessor functions can only
  - **read** data members
  - **Cannot write** into class data members
- Constant functions are used to make accessors

```
returntype functionName (parameters list) const;
```

- **Example:**
  - Getters should be accessors as they only read data

Member functions (Mutators and Accessors)

```
class Point
{
        int x;
        int y;
    public:
    //getters
     int getX() const { return x;} //Accessor
     int getY() const { return Y;} //Accessor
     Point getPoint() const {return *this;} //Accessor
    //setters
     void setX(int x){ this->x = x;} //Mutator
     void setY(int y){ this->y = y;} //Mutator
     void setPoint(int x, int y){ this->x = x; this->y = y;} //Mutator
     void setPoint(Point p){//Mutator
        this->x = p.x;
        this->y = p.y;
    }
    //add member functions complete implementation inside class definition
};
```

Member functions (Setters and Getters)

```
class Point
{
        int x;
        int y;
    public:
    //getters
     int getX() const; //Accessor
     int getY() const; //Accessor
     Point getPoint() const; //Accessor
    //setters
     void setX(int x); //Mutator
     void setY(int y); //Mutator
     void setPoint(int x, int y); //Mutator
     void setPoint(Point p); //Mutator
    //add member functions Prototype inside class definition
};
```

Member functions (Mutators and Accessors)

```
//implement outside class definition
void Point::setX(int x){
    this->x = x;
}

void Point::setY(int y){
    this->y = y;
}

void Point::setPoint(int x, int y){
        this->x = x;
        this->y = y;
}

void Point::setPoint(Point p){
    x = p.x;
    y = p.y;
}
```

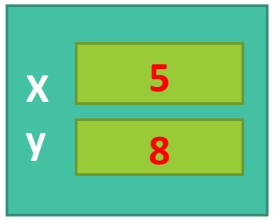
```
int Point::getX() const { return x;}

int Point::getY() const { return y;}

Point Point::getPoint() const
{
    return *this;
}
```

```
void main(){
    Point p;
    p.setPoint(5,8);
    cout << p.getX();
    cout << p.getY();

}
```



Member functions (Other functions)

```
class Point {
        int x;
        int y;
    public:
    //getters
     int getX();
     int getY();
     Point getPoint();
    //setters
     void setX(int x);
     void setY(int y);
     void setPoint(int x, int y);
     void setPoint(Point p);
    //Other functions
     void printPoint() const;
    //Accessor
};
```

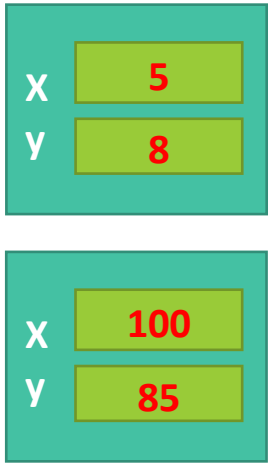
```
void Point::printPoint() const {
    cout<< " X: " << this->x;
    cout<< " Y: " << this->y;
}
```

```
void main(){

    Point p;
    p.setPoint(5,8);
    p.printPoint();

    Point p2;
    p2.setPoint(100,85);
    p2.printPoint();

}
```



Member functions (Other functions)

```
class Point {
        int x;
        int y;
        float calculateDistance(Point &p) const;
    public:
    //getters
     int getX();
     int getY();
     Point getPoint();
    //setters
     void setX(int x);
     void setY(int y);
     void setPoint(int x, int y);
     void setPoint(Point p);
    //Other functions //Accessors
     void printPoint() const;
     Point find_closest (Point &p1, Point &p2) const;
};
```

Member functions (Other functions)

```
float Point::calculateDistance(Point &p)
const
{
    int d1 = p.x - x;
    int d2 = p.y - y;
    float temp = ( (d1*d1) + (d2*d2) );
    return sqrt(temp);
}

Point Point::(Point &p1, Point &p2)const
{
    float d1 = Calculate_Distance(p1);
    float d2 = Calculate_Distance(p2);
    if(d1<=d2)
        return p1;
    else
        return p2;
}
```

```
void main(){
    Point p; p.setPoint(5,8);
    p.printPoint();

    Point p2;
    p2.setPoint(100,85);
    p2.printPoint();

    Point p3;
    p3.setPoint(10,89);
    p3.printPoint();

    cout<< p.calculateDistance(p2);

    p.calculateDistance(p2,p3).printPoint();
}
```