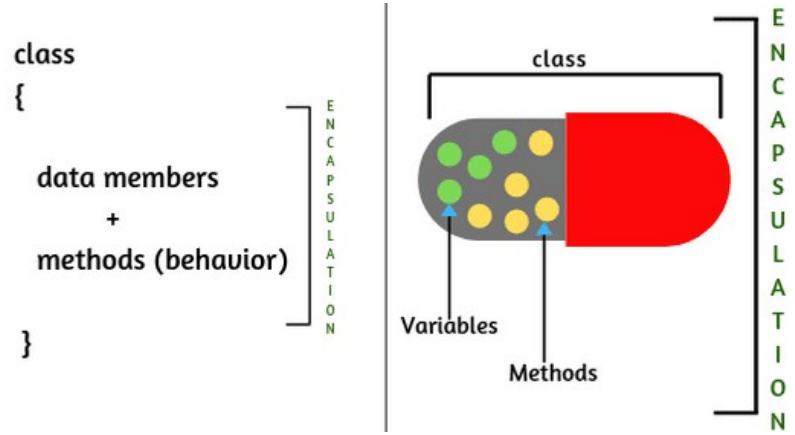
Object Oriented Programming
Constructors and Destructor

CS(217) Object Oriented Programming

Abeeda Akram

Class Member Functions (TYPES)

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



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 (Setters)

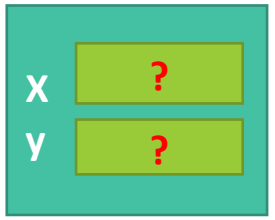
- Used to set or update values of individual **data members** or **complete Object**

```
void main(){

    Point p;
    //Objects created but not initialized

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

}
```



How to initialize data members of objects at declaration Time

Member functions (Constructors)

Special Member functions

- Used to **Initialize** data members **at creation time of object**
- Name should be same as of class name
- Have no return type
- Can overload constructors by changing
  - Parameters type
  - Number of Parameters
- **No need to call explicitly**
- **Automatically called by system when object is created**

Member functions (Constructors)

Types of Constructors

1. **Default Constructor** (has no parameters)
2. **Parameterized Constructors** (have one or more parameters)
3. **Copy Constructor** (has class object by reference only)

In entire life time of the object only one constructor is called at its creation time either Default, Parameterized or Copy

If no constructor is implemented in the class

- **then system calls the dummy default constructor**
- **which does nothing (does not initialize data members)**

All type of Constructors should have public member access specifier

Member functions (Default Constructor)

- Used to initialize objects data members with **some default values**
- Can add **only one default** constructor in a class
- Has **no** parameters

```
class Point {
        int x;
        int y;
    public:
    //Default constructor
    Point();
    //add member functions prototype
};
Point::Point(){
    x = 0;
    y = 0;
}
```

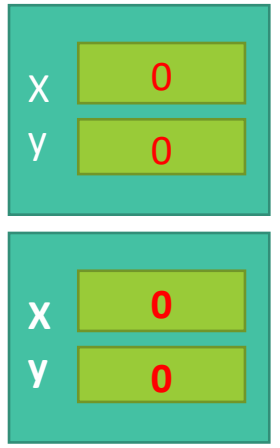
```
void main(){

    Point p;
    Point p2;

    Point p3();
    //Compiler Error

    p.point();
    // Compiler Error Cannot call more than
    once

    p.setpoint(3,2);
    //Call Setters for object further updates
}
```



Member functions (Parameterized Constructors)

- Used to initialize objects data members with **input values**
- Can add **more than one** parameterized constructors in a class

```
class Point {
        int x;
        int y;
    public:
    //Default constructor
    Point();
    //Parameterized constructors
    Point(int x);
    Point(int x, int y);
};
Point::Point(){ x = 0; y = 0;}
Point::Point(int x){ this->x = x;}
Point::Point(int x, int y){
    this->x = x;
    this->y = y;
    }
```

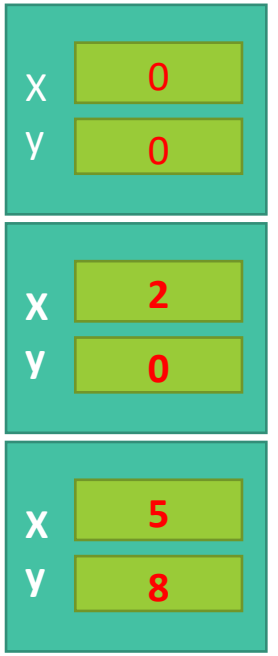
```
void main(){

    Point p;

    Point p2(2);

    Point p3(5,8);

}
//which one to call depends on number
of arguments
//C++ conversion if type not
matched //Ambiguous then compile time
error
```



Member functions (Constructor with Default Parameters)

```
class Point {
        int x;
        int y;
    public:
    //Default and Parameterized
    Point(int x = 0, int y = 0);
    Point(int x);
};
//With Default Parameters
Point::Point(int x = 0, int y = 0)
{
    this->x = x;
    this->y = y;
}
Point::Point(int x){ this->x = x;}
```

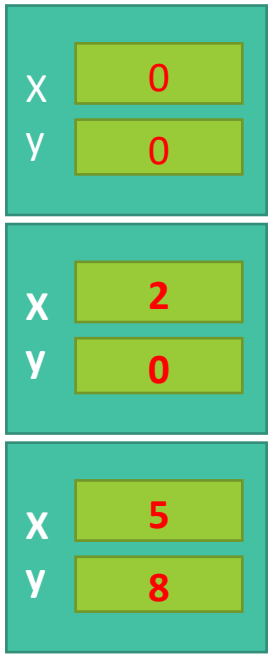
```
void main(){

    Point p;

    Point p2(2);

    Point p3(5,8);

}
//which one to call depends on number
of arguments
//C++ conversion if type not
matched //Ambiguous then compile time
error
```



Member functions (Parameterized Constructors)

- Used to initialize objects data members with **input values**
- Can add **more than one** parameterized constructors in a class

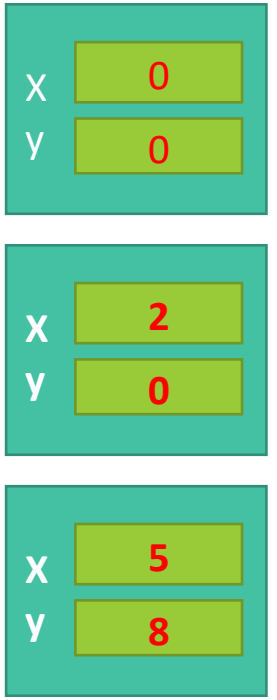
```
class Point {
        int x;
        int y;
    public:
    //Default and Parameterized
    Point(int x = 0, int y = 0);
};
//With Default Parameters
Point::Point(int x = 0, int y = 0)
{
    this->x = x;
    this->y = y;
}
```

```
void main(){
    //Dynamic Objects
    Point * p = new Point;
    p->printPoint();

    Point * p2 = new Point(2);
    p2->printPoint();

    Point * p3 = new Point(5,8);
    p3->printPoint();

    delete p;
    delete p2;
    delete p3;
}
```



Member functions (Copy Constructor)

Used to **create copy** of data members of different objects

- Name should be same as of class name
- Have no return type
- Takes same class object as parameter (**by constant reference only)**

No need to call explicitly Automatically called by system

1. **When an object is initialized using another object**
2. **Object is passed in a function by value**
3. **Object is returned from a function by value**

If no copy constructor is implemented in the class

- **then system calls the built-in copy constructor**
- **which member wise copy data using assignment operation**

Member functions (Copy Constructor)

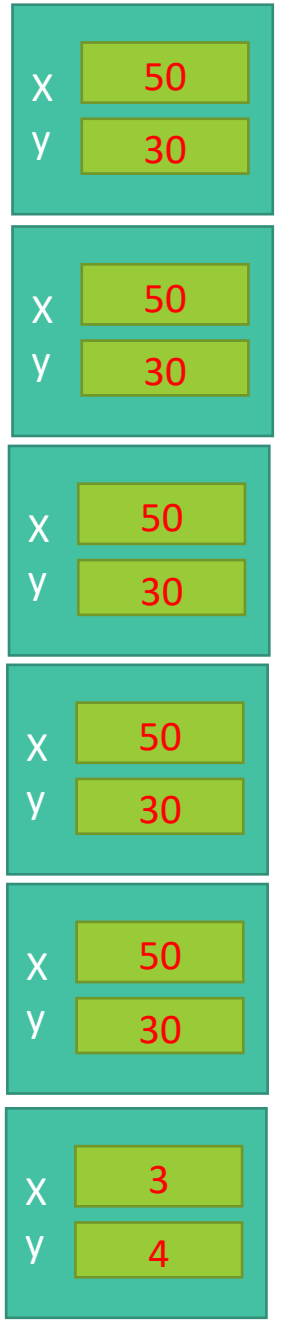
```
class Point {
        int x;
        int y;
    public:
    //Copy Constructor
    Point(const Point & p);
};
//Copy Constructor
Point::Point(const Point & p)
        this->x = p.x;
        this->y = p.y;
}
```

```
void main(){
    Point p(50,30);
//When an object is initialized using another object
    Point p2(p);
    Point p3 = p2;
//not assignment when object is created

    Point * p4 = new Point(p3);
    Point * p5 = new Point(*p4);

//not called for anonymous objects
    Point p6(point(3,4));
//Changed to
    Point p6(3,4);

    delete p4;
    delete p5;
}
```

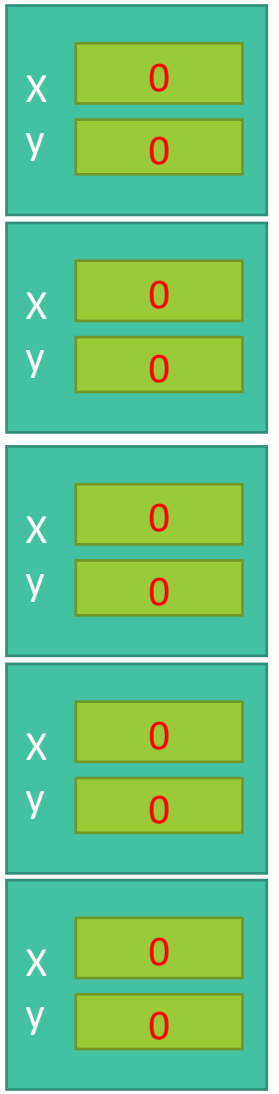


Member functions (Copy Constructor)

```
class Point {
        int x;
        int y;
    public:
    float calculateDistance(Point p)
    const;
    //Copy Constructor
    Point(const Point & p);
};
//Copy Constructor
Point::Point(const Point & p)
        this->x = p.x;
        this->y = p.y;
}
```

```
void main(){
    Point p;
    Point p2(p);
//Object is passed in  function by value
    cout << equal (p1, p2);
    cout << p.calculateDistance(p3);
}

//objects pass by value or copy
bool equal(Point p, Point q){
    if((p.x == q.x)&&(p.y == q.y))
        return true;
    else
        return false;
}
```



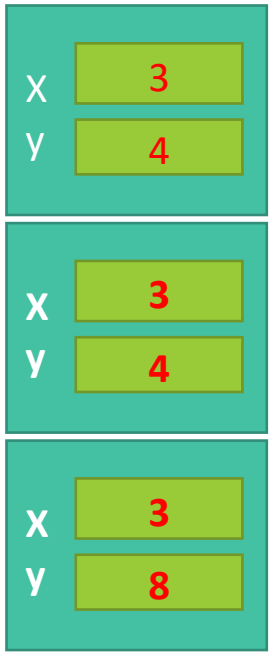
Member functions (Copy Constructor)

```
class Point {
        int x;
        int y;
    public:
    float calculateDistance(Point p)
    const;
    Point find_closest(Point &p1,
    Point &p2) const;
    //Copy Constructor
    Point(const Point & p);
};
//Copy Constructor
Point::Point(const Point & p)
        this->x = p.x;
        this->y = p.y;
}
```

```
void main(){
    Point p(3,4);
    Point p2(p);
    Point p3(3,8);
//Object is returned from  function by value
   p.find_closest(p2,p3).printPoint();

    Point p4 = p.find_closest(p2,p3);
    //not called for anonymous objects
}

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



Member functions (Destructor)

Special Member functions

- Used to **deallocate** data members memory **at destruction time of object**
- Name should be same as of class name prefix with telda (**~**)
- Takes no parameters
- Has no return type
- Cannot overload destructors

No need to call explicitly automatically called by system when

- **Object is out of scope**
- **Dynamic object is deallocated**

Destructor should have public member access specifier always

If no destructor is implemented in the class

- **then system calls the dummy default destructor which does nothing**

Member functions (Destructor)

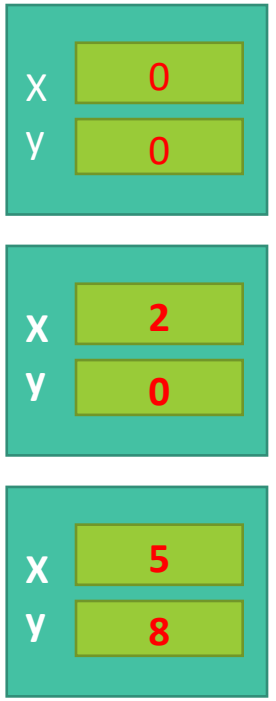
- Can add **only one destructor** in a class

```
class Point {
        int x;
        int y;
    public:
    //Constructors
    Point(int x = 0, int y = 0);
    Point(const Point & p);
    //Destructor
    ~Point();
};

Point::~Point(){
    cout << "Nothing to do";
}
```

```
void main(){

    {
     Point p;
     Point p2(2);
     Point p3(5,8);
     }
     //Destructor called
     //Destructor called
     //Destructor called
     cout << "out of block";
}
```



Member functions (Destructor)

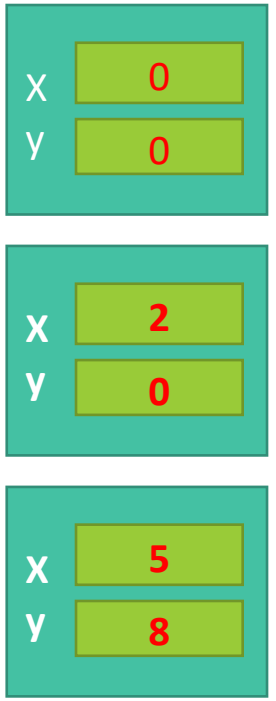
- Can add **only one destructor** in a class

```
class Point {
        int x;
        int y;
    public:
    //Constructors
    Point(int x = 0, int y = 0);
    Point(const Point & p);
    //Destructor
    ~Point();
};

Point::~Point(){
    cout << "Nothing to do";
}
```

```
void main(){
    fun();
}

void fun(){
    Point p;
    Point p2(2);
    Point p3(5,8);
     //Destructor called
    //Destructor called
    //Destructor called
}
```



Member functions (Destructor)

- Can add **only one destructor** in a class

```
class Point {
        int x;
        int y;
    public:
    //Constructors
    Point(int x = 0, int y = 0);
    Point(const Point & p);
    //Destructor
    ~Point();
};

Point::~Point(){
    cout << "Nothing to do";
}
```

```
void main(){

    //Dynamic Objects
    Point * p = new Point;
    Point * p2 = new Point(2,8);
    Point * p3 = new Point(p2);

    delete p;
    //Destructor called
    delete p2;
    //Destructor called
    delete p3;
    //Destructor called

}
```