Methods

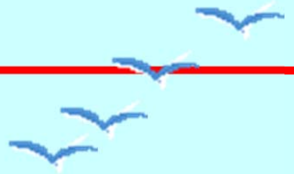
- Methods overloading
- Constructors: default, custom, initializer list

Previous lectures

We have talked about:

- structures having functions as members
- data hiding
- classes and objects

Classes



A **class** is a ***user-defined type*** that contains ***variables (data members)*** as well as the set of ***methods (member functions)*** that manipulate that data.

Encapsulation

A language mechanism for restricting access to some of the object's components.

Encapsulation is to prevent unauthorized parties to use variables or methods hidden inside the private part of a class.

So only the public methods of the class access its private variables (data members) and the other functions/classes call these public methods in order to use objects of the class type—send messages to objects.

Point class example

```
class Point {
public:
  void set(int, int);
  void print();
private:
  int x,y; //coordinates
};
```

member functions

data members

preferred 159.234 style

```
class Point {
  int x,y; //coordinates
public:
  void set(int, int);
  void print();
};
```

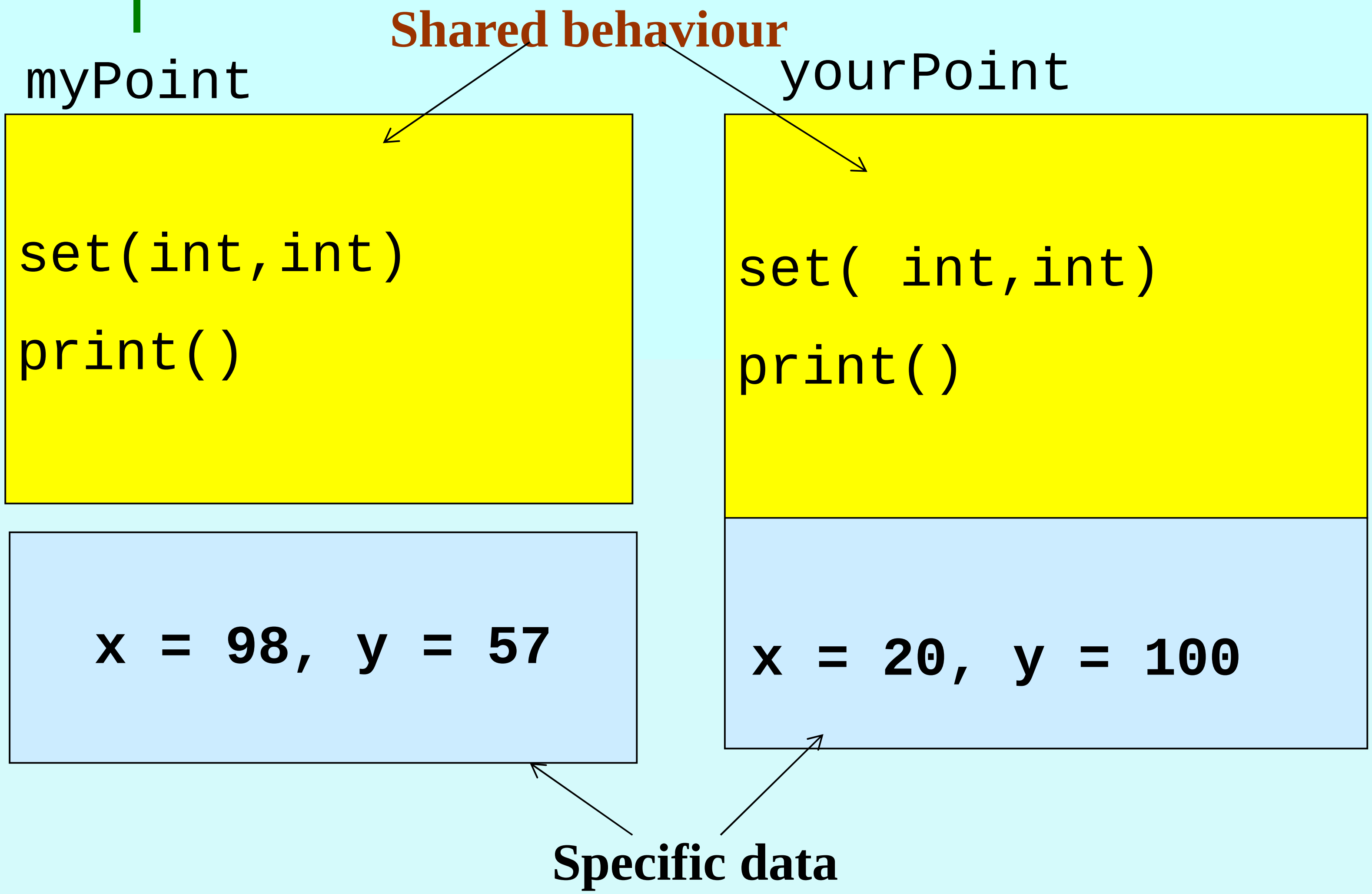
Using objects

```
class Point {
 public:
   void print();
   void set(int u, int v){
      x = u; y = v;
   }
 private:
   int x,y;
};

void Point::print() {
   cout << "(" << x << ", "
        << y << ") ";
}
```

```
int main(){
   Point origin, somePt;
   origin.set(0,0);
   somePt.set(-34,8);
   cout <<"The origin is  at ";
   origin.print();
   cout <<"\nAnd the center is  at ";
   somePt.print();
   cout <<endl;
   return 0;
}
```

Objects



Shared behaviour

myPoint

set(int,int)

print()

x = 98, y = 57

yourPoint

set(int,int)

print()

x = 20, y = 100

Specific data

Overloading methods

Member functions can also be **overloaded**.

```
class Point {
 public:
   void set(int u, int v) {
      x = u; y = v;
   }
   void print();
   void print(string s);

 private:
   int x,y;
};
```

```
void Point::print() {
   cout << “(” << x << “,” << y << “)”;
}

void Point::print(string s) {
   cout << s;
   print(); //No scope operator is required here.
}
```

```
int main(){
   Point w;  //w is an object of Point type
   w.set(4,7);
   w.print();
   cout << endl;
   w.print(“our point = ");
 }
```

```
Output:                      (4,7)
                             our point = (4,7)
```

Class scope

```
void Point::print(string s) {
   cout << s;
   print();
}
```

Object initialization

Constructor

```
class Point {
 public:
   Point(int  i, int j);
   int x,y;
};

Point::Point(int  i, int j) {
   x = i;
   y = j;
}
```

```
int main{
  Point p(4,5);
  //..more code..
}
```

Object initialization

Is this correct ?

```
    Point t;
```

Yes, because the C++ system provides a **system default constructor** in case you do not provide any constructors for your class.

In some cases (pointer variables) this constructor is not good enough.

```
    Point a[100];
```

Array of objects can only be initialised using the default constructor.

No default constructor no **arrays.**

Object initialization

Several constructor functions:

```
class Point {
 public:
   Point();
   Point(int  i, int j);
 private:
   int x,y;
};
```

Our default constructor

```
Point::Point(){  x = 0; y = 0;}

Point::Point(int  i, int j) {x =  i; y = j;}
```

Object initialization

This is OK

```
class Point {
 public:
   Point(int i=0, int j=0);
private:
   int x,y;
};


Point::Point(int i, int j)
{x = i; y = j;}
```

```
class Point {
public:
  Point(int i=0, int j=0);
private:
  int x,y;
};

 Point::Point(int i=0, int j=0)
 {x = i; y = j;}
```

This is WRONG

Initialiser lists

```
Point::Point() : x(0), y(0){}

Point::Point(int i,int j):x(i), y(j) {}
```

Constructor initialiser lists is the preferred way.

Can we write a constructor to be at the same custom and default constructor?

Initialiser lists

```
Point::Point() : x(0), y(0){}

Point::Point(int i,int j):x(i), y(j) {}
```

Constructor initialiser lists is the preferred way.

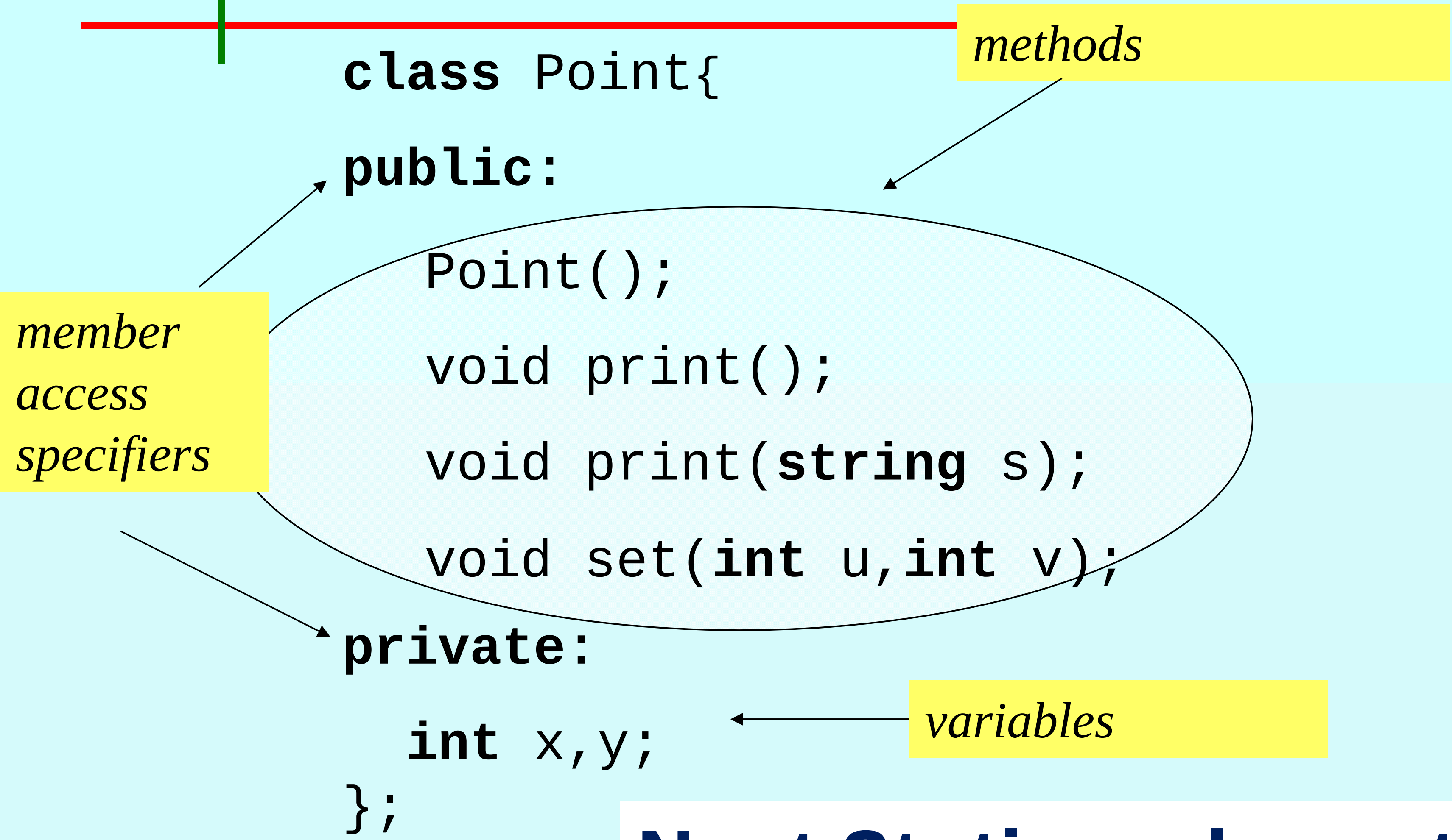
Can we write a constructor to be at the same custom and default constructor?

```
Point::Point(int i=0,int j=0):x(i), y(j) {}
```

Error:

```
Point::Point(int i(0),int j(0)):x(i), y(j) {}
Point::Point(int i{0},int j{0}):x(i), y(j) {}
```

Summary



```
class Point{
public:
   Point();
   void print();
   void print(string s);
   void set(int u,int v);
private:
  int x,y;
};
```

methods

member access specifiers

variables

Next Static and const