Classes in C++

A lot of this stuff is trivia, but it can be hard to discern up front. Classes in C++ are complex and have a ton of rules, most of which I look up as I go

I'll try to point out the core concepts and assign readings / practice

Because there are so many rules, you **have** to do the reading

Coming to class **will not** be enough

Let’s say I wanted to represent a rectangle

```
class Rectangle {
private:
  unsigned int length;
  unsigned int width;

public:
  Rectangle(unsigned int length, unsigned int width)
    : length(length), width(width)
  {
  }

  unsigned int area() const {
    return (length * width);
  }
};
```

```
class Rectangle {
private:
  unsigned int length;
  unsigned int width;

public:
  Rectangle(unsigned int length, unsigned int width)
    : length(length), width(width)
  {
  }

  unsigned int area() const {
    return (length * width);
  }
};
```

Member variables

```
class Rectangle {
private:
  unsigned int length;
  unsigned int width;

public:
  Rectangle(unsigned int length, unsigned int width)
    : length(length), width(width)
  {
  }

  unsigned int area() const {
    return (length * width);
  }
};
```

Constructor

```
class Rectangle {
private:
  unsigned int length;
  unsigned int width;

public:
  Rectangle(unsigned int length, unsigned int width)
    : length(length), width(width)
  {
  }

  unsigned int area() const {
    return (length * width);
  }
};
```

Method

Rectangle myRectangle = Rectangle(12,14);

"Give me an empty **Rectangle** and then call its constructor to fill it in."

This is called a “constructor”

```
Rectangle(unsigned int len, unsigned int wid)
{
  length = len;
  width = wid;
}
```

It is a special method, called whenever a Rectangle is created, to set it up

Here I use an *initialization list*

```
Rectangle(unsigned int length, unsigned int width)
  : length(length), width(width)
{
}
```

In C++, initialization is a special thing, and this method is preferred

(More efficient, for reasons we'll learn later)

Special Constructors

- A **default constructor** is one without any arguments
  - C++ makes this for you if you don't define a constructor
- A **copy constructor** copies an object
  - C++ will generate this too

Initialization by copy-constructor

```
int main() {
  Rectangle myRectangle = Rectangle(12,14);

  Rectangle myOtherRectangle(myRectangle);

  cout << "The area of myOtherRectangle is “
       << myOtherRectangle.area() << endl;
  return 0;
}
```

But you can define it instead, if you want!

Note: must be const…

```
Rectangle(const Rectangle &other)
  : length(other.length), width(other.width)
{
}
```

I can define things outside of classes too…

```
// shapes.h
class CartesianPoint {
public:
  double distanceFrom(CartesianPoint &other) const;
}
// shapes.cc
double CartesianPoint::distanceFrom(CartesianPoint &other) const {
  return
    sqrt((x - other.getX()) * (x - other.getX())
         + (y - other.getY()) * (y - other.getY()));
}
```

Functions smaller than a line or two should usually
go outside of the header file

Why keep things private?

- Reveal as little about your implementation as possible
- Because if someone else assumes details, I'll break their code
- **Almost never** make member vars public

Consider what would happen if we wrote 20k lines of code that used x and y, but then we wanted to change to the top left and bottom right point

We'd waste a ton of time rewriting it

Consider what would happen if we wrote 20k lines of code that used x and y, but then we wanted to change to the top left and bottom right point

This is the **biggest mistake** new programmers make: declare everything public. Hiding things makes things harder, and that's exactly what we want

Exercise: write impl of rectangle that caches area

In class exercise: use top-left coordinate, bottom-right coordinate instead

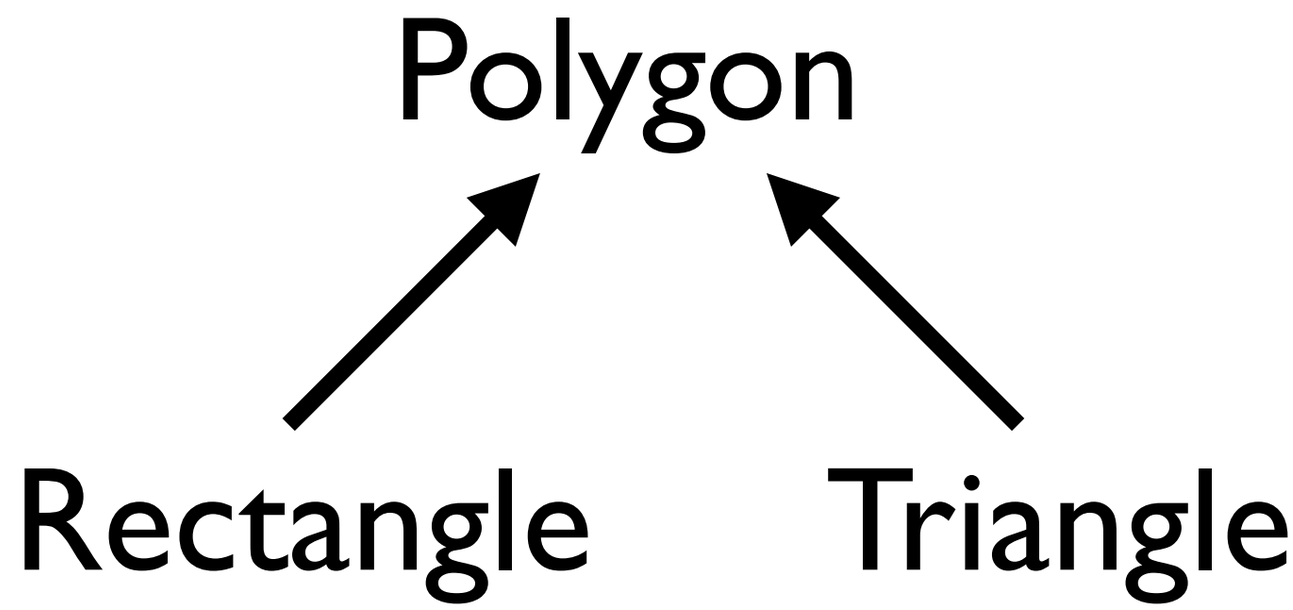
I can define things outside of classes too…

At the end of a function, const means "I don't change anything"

```
// shapes.h
class CartesianPoint {
public:
  double distanceFrom(CartesianPoint &other) const;
}
// shapes.cc
double CartesianPoint::distanceFrom(CartesianPoint &other) const {
  return
    sqrt((x - other.getX()) * (x - other.getX())
         + (y - other.getY()) * (y - other.getY()));
}
```

Functions smaller than a line or two should usually go outside of the header file

Now let's say I want…



Abstract class

```
class Shape {
public:
  virtual double area() const = 0;
};
```

Pure virtual method

Abstract class

```
class Shape {
public:
  virtual double area() const = 0;
};
```

Pure virtual method

I’m not defining it here, but subclasses **must**

Abstract class

```
class Shape {
public:
  virtual double area() const = 0;
};
```

Pure virtual method

I’m not defining it here, but subclasses **must**

Because Shape has a pure virtual method, no concrete Shape can exist, it is an **abstract class**

Abstract class

```
class Shape {
public:
  virtual double area() const = 0;
};
```

Pure virtual method

I’m not defining it here, but subclasses **must**

Because Shape has a pure virtual method, no concrete Shape can exist, it is an **abstract class**

Big note: the `= 0` does **not** mean to return 0, it means nothing defined

This syntax means “Rectangle inherits from Shape”

```
class Rectangle : public Shape {
private:
  unsigned int length;
  unsigned int width;

public:
  Rectangle(unsigned int length, unsigned int width)
  : length(length), width(width)
  { }

  Rectangle(const Rectangle &other)
    : length(other.length), width(other.width)
  { }

  virtual double area() const {
    return (length * width);
  }
};
```

This syntax means “Rectangle inherits from Shape”

```
class Rectangle : public Shape {
private:
  unsigned int length;
  unsigned int width;

public:
  Rectangle(unsigned int length, unsigned int width)
    { }

  Rectangle(const Rectangle &other)
    : length(other.length), width(other.width)
  { }

  virtual double area() const {
    return (length * width);
  }
};
```

https://stackoverflow.com/questions/860339/difference-between-private-public-and-protected-inheritance

Subclasses

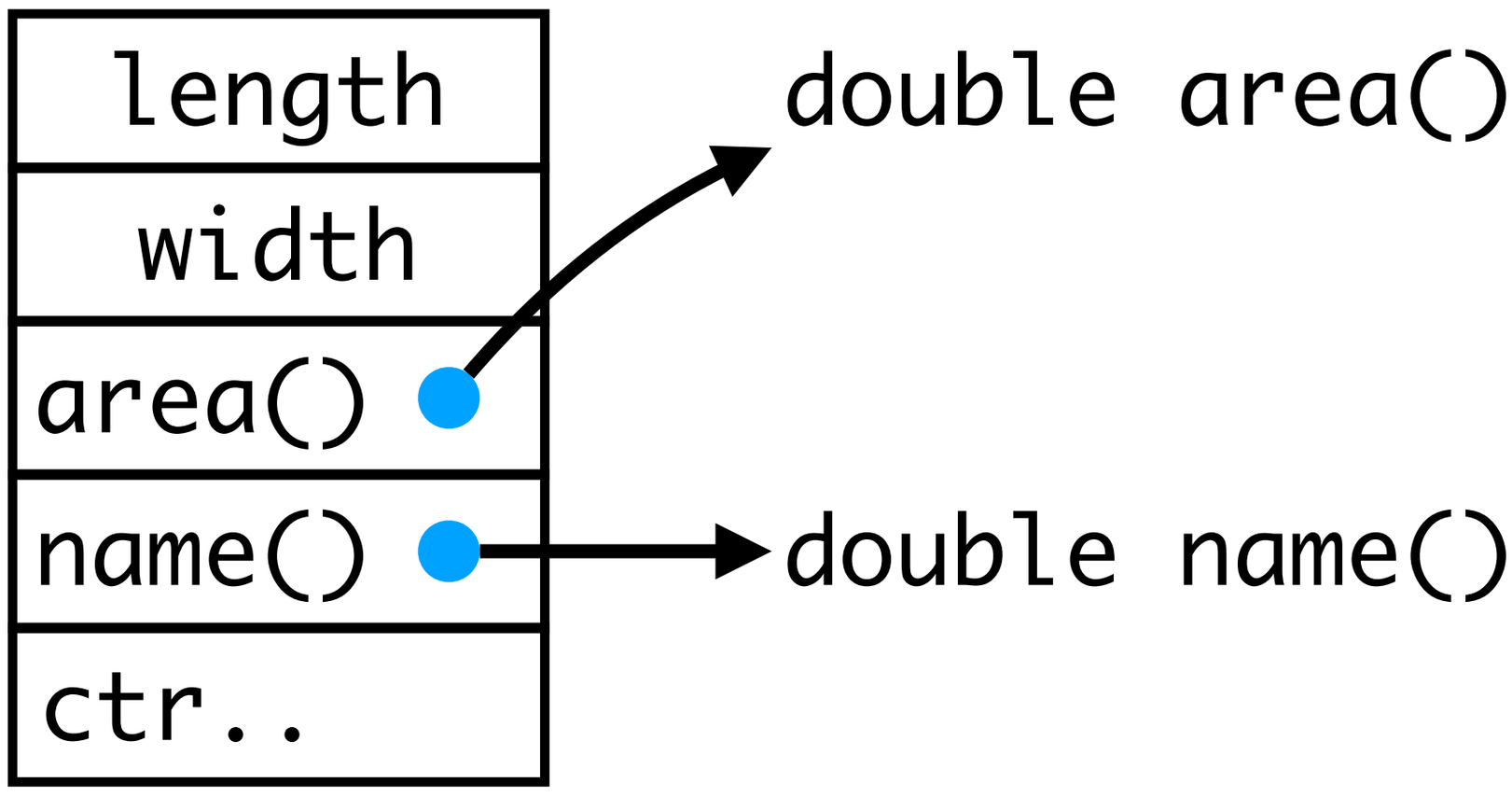
- Inherit all member variables, can modify public / protected ones
- Can redefine **virtual** methods
  - It may help to think of these as "redefinable" methods

Storage

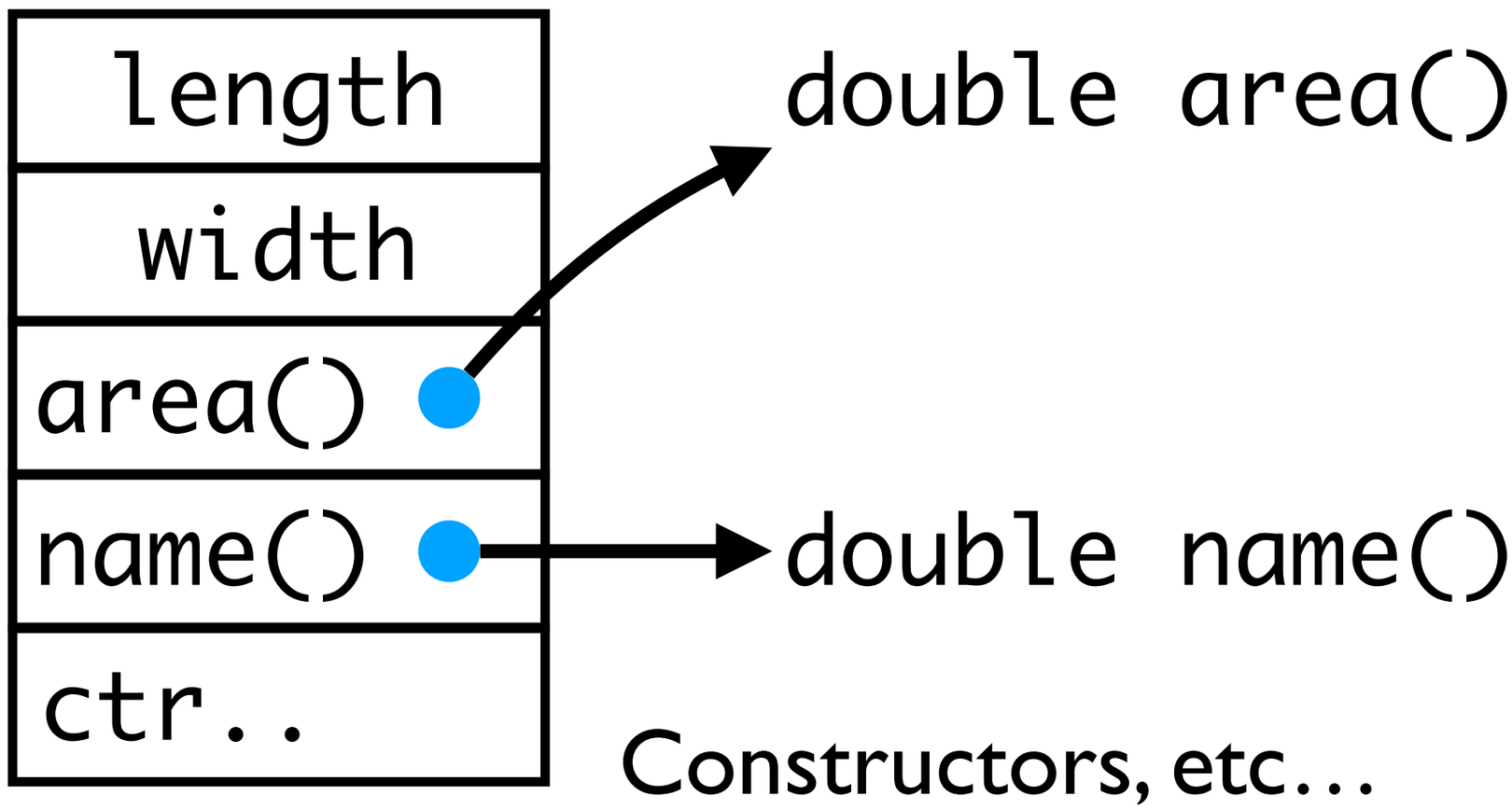
Storage

We’re not going to talk about pointers yet, so don’t think about that today

This is what a `Rectangle` looks like to C++

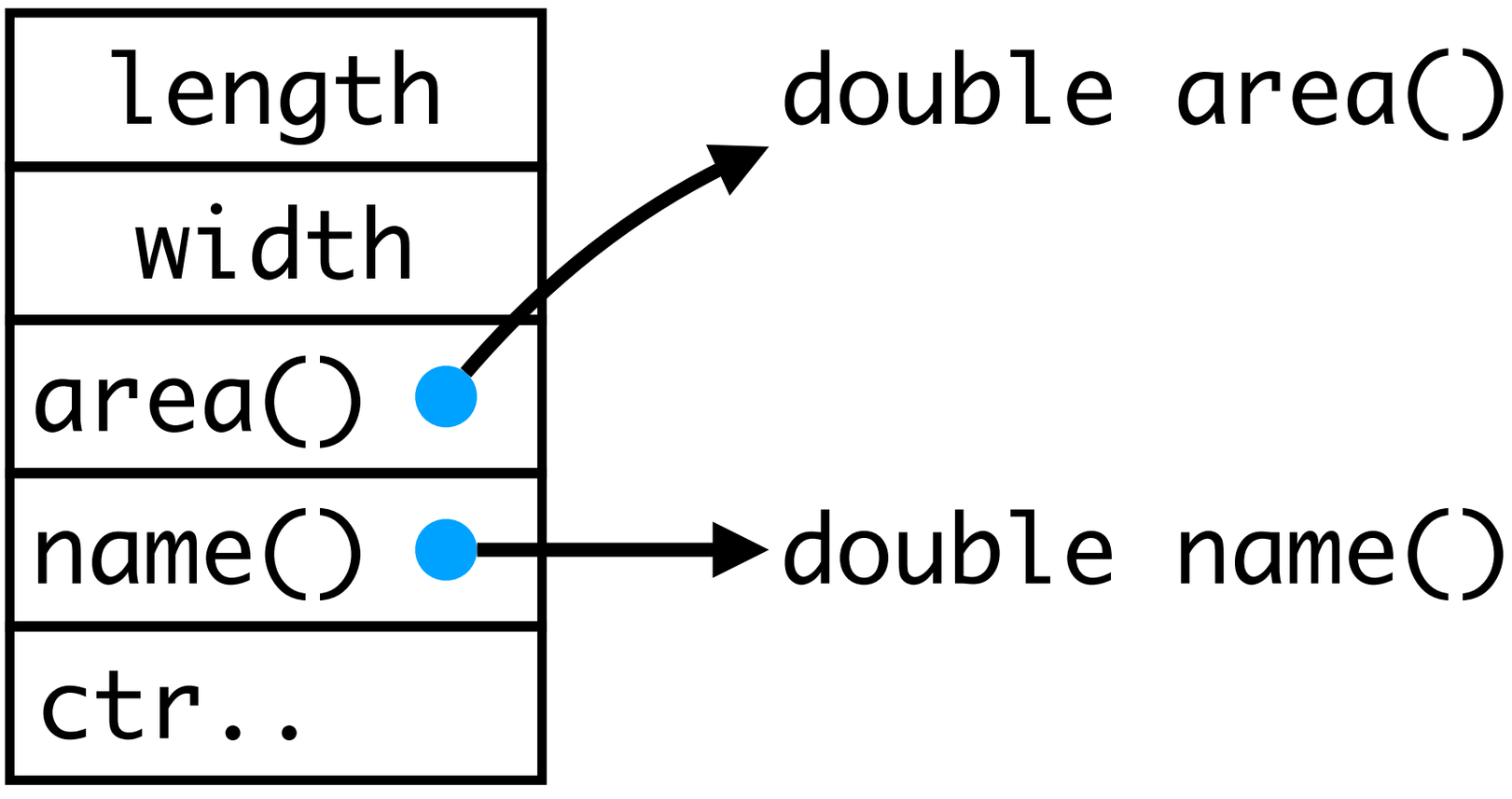


This is what a `Rectangle` looks like to C++

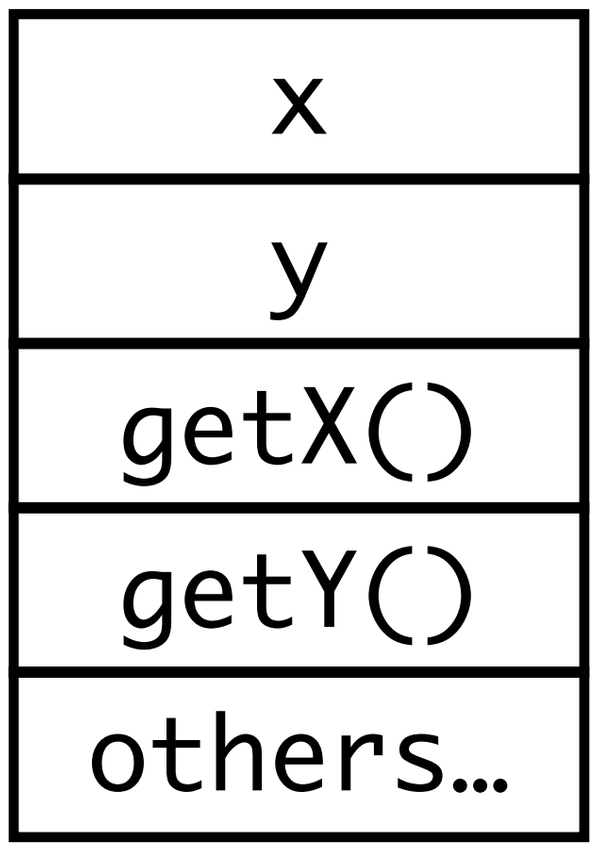


A **class** is like a recipe to make a rectangle

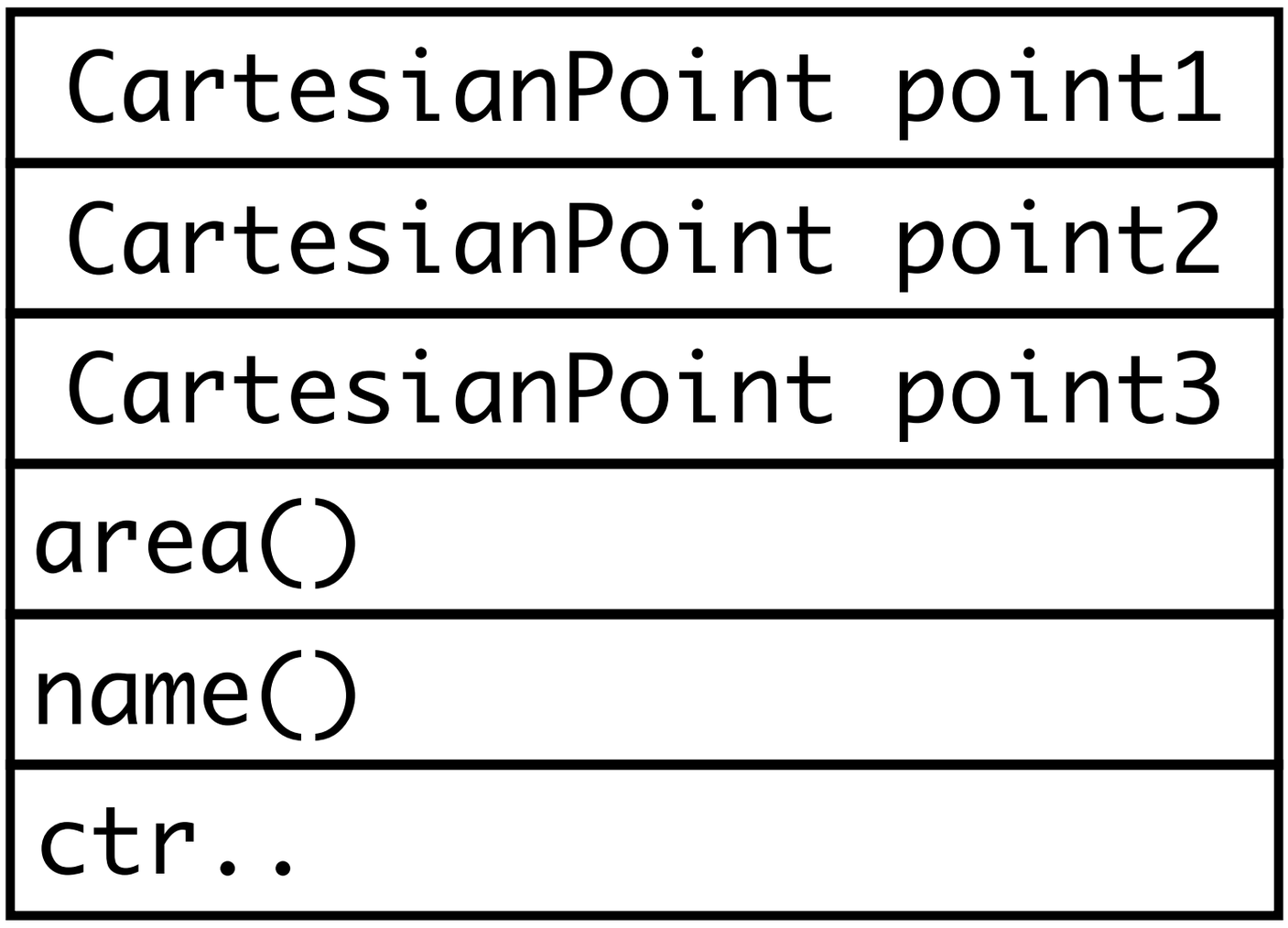
"To make a rectangle, I need to fill in length / width"



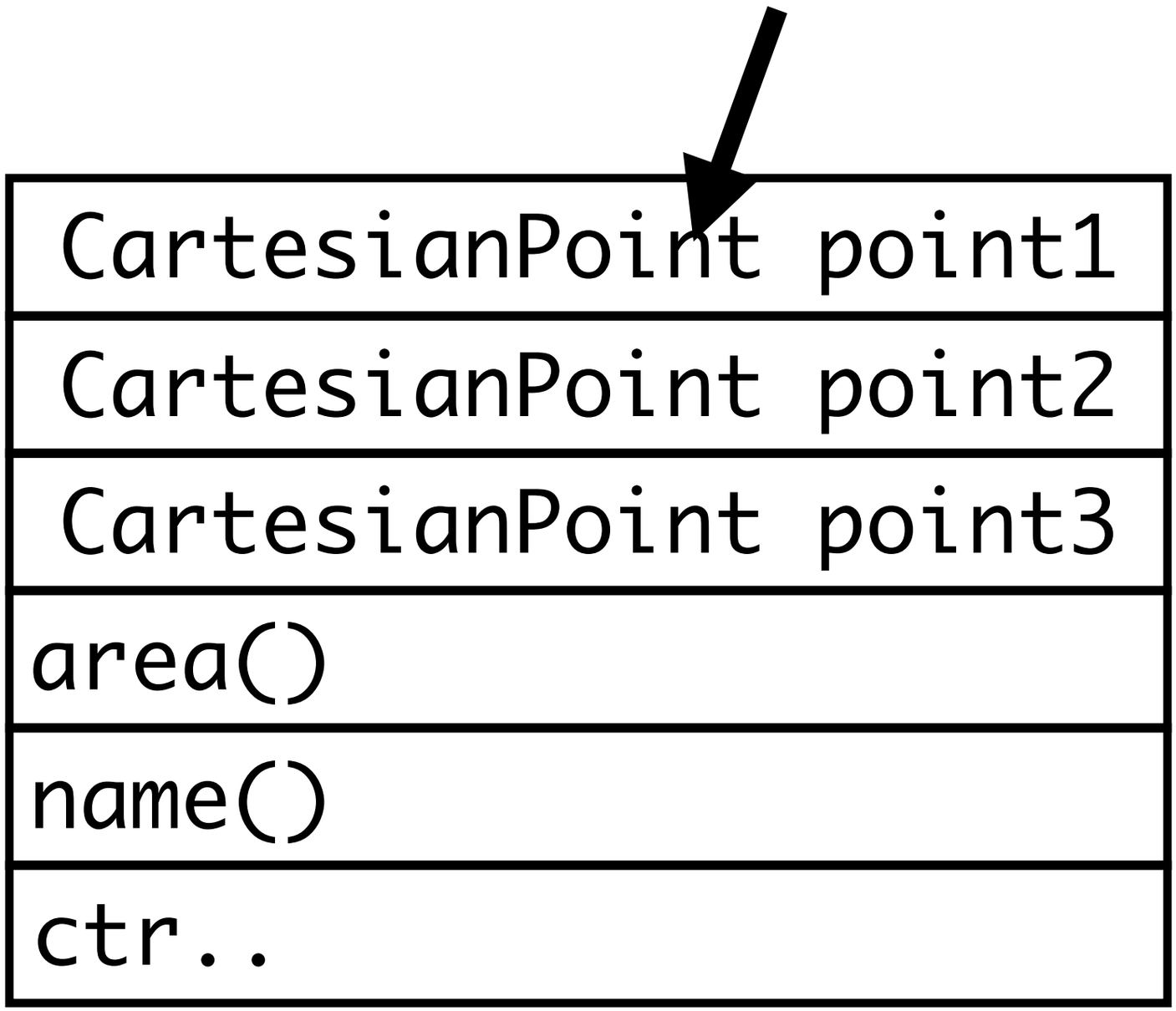
This is what a `CartesianPoint` looks like

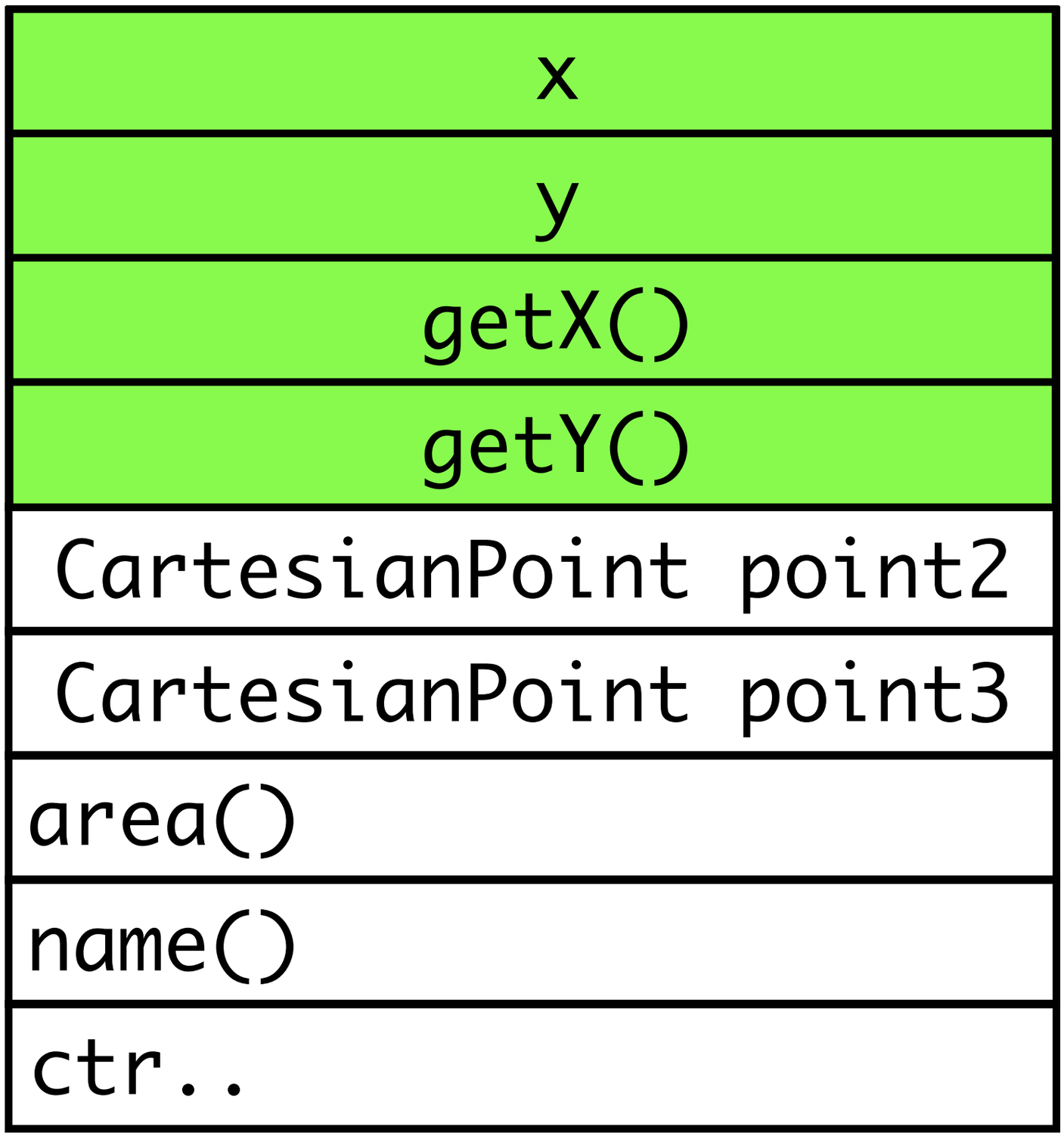


So what does a `Triangle` look like?


CartesianPoint point1
CartesianPoint point2
CartesianPoint point3
area()
name()
ctr..

Stuff the box for `CartesianPoint` here





point1

Same for these

So the rule is, if you see a class as a member variable of another class, it's like stuffing a box inside a box

Next class, we'll see that these boxes are laid out in memory