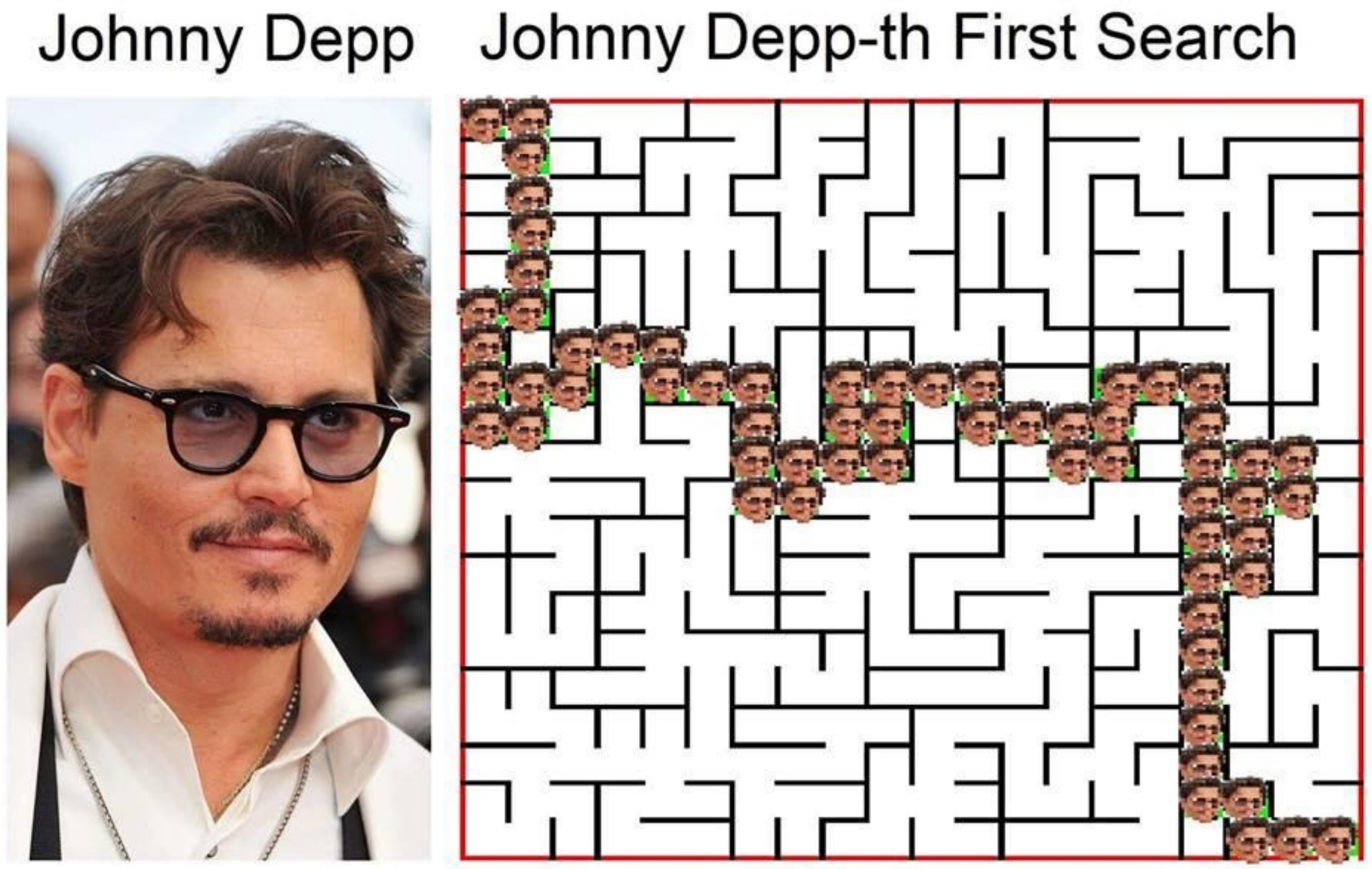
CS 32 - Discussion 2D/3D

Week 5 - Inheritance & Recursion

Unrelated meme



Inheritance

Basics

```
class Shape // (Base class)
{
public:
…
private:
double x; // x-coord of center
double y; // y-coord of center
};

// Circle is a kind of Shape and so has the general
// properties of a Shape + the more specific properties
// of a Circle.
class Circle : public Shape // (Derived class)
{
public:
double getRadius() {return r;} // specifically for Circles
…
private:
double r; // Circle inherits x,y as well
};
```

- Derived class inherits all members of the base class
- Automatic conversion of Circle* to Base* and Circle& to Base&.

Inheritance

Basics

```
class Shape // (Base class)
{
public:
…
private:
double x; // x-coord of center
double y; // y-coord of center
};

// Circle is a kind of Shape and so has the general
// properties of a Shape + the more specific properties
// of a Circle.
class Circle : public Shape // (Derived class)
{
public:
double getRadius() {return r;} // specifically for Circles
…
private:
double r; // Circle inherits x,y as well
};
```

- Derived class inherits all members of the base class
- Automatic conversion of Circle* to Base* and Circle& to Base&.

I.e…

```
void f(Shape& s) { … }

Circle c;

f(c);

// this is legal
```

Inheritance

Basics

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
… // Using base class constructor to initialize inherited data members …
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```

- Derived class inherits all members of the base class
- Automatic conversion of Circle* to Base* and Circle& to Base&.

I.e…

```
void f(Shape& s) { … }

Circle c;

f(c);

// this is legal
```

Inheritance - Virtual Functions

Static vs Dynamic Binding

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
void move(double x_new, double y_new) {xc = x_new; yc = y_new;}
// all shapes move in the same way, so we can use static binding (not virtual)

private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```

Inheritance - Virtual Functions

Static vs Dynamic Binding

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
void move(double x_new, double y_new) {xc = x_new; yc = y_new;}
void draw() const {… draw a cloud centered at (xc,yc) …}
// problem: We want this function to be different for different Shapes…
// but, suppose we still want to be able to draw a Shape without knowing what kind of Shape it is.
private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```

Inheritance - Virtual Functions

Static vs Dynamic Binding

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
void move(double x_new, double y_new) {xc = x_new; yc = y_new;}
virtual void draw() const {… draw a cloud centered at (xc,yc) …}
// fix: use virtual keyword and overload in Circle class
private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
virtual void draw() const {…draw a circle of radius r centered at (xc,yc)…}
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```

Inheritance - Virtual Functions

Static vs Dynamic Binding

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
void move(double x_new, double y_new) {xc = x_new; yc = y_new;}
virtual void draw() const {… draw a cloud centered at (xc,yc) …}
// fix: use virtual keyword and overload in Circle class
private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
virtual void draw() const {…draw a circle of radius r centered at (xc,yc)…}
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```

- Basic idea:
  - Declaring a function as virtual makes it dynamically bound.
    - dynamic means the derived version of the function is called
  - Otherwise, function is statically bound.
    - static means the base version of the function is called by default

Inheritance - Virtual Functions

Static vs Dynamic Binding

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
void move(double x_new, double y_new) {xc = x_new; yc = y_new;}
virtual void draw() const {… draw a cloud centered at (xc,yc) …}
// fix: use virtual keyword and overload in Circle class
private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
virtual void draw() const {…draw a circle of radius r centered at (xc,yc)…}
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```

```
Circle c(0,0,1);

c.draw();

// calls Circle's draw function for c

Shape s(1,1);

s.draw();

// calls Shape's draw function for s
```

Inheritance - Pure Virtual Functions

“Abstract” base classes

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
void move(double x_new, double y_new);
virtual void draw() const {… draw a cloud centered at (xc,yc) …}
virtual double getArea() const;
// problem: we want getArea() function to be able to be called for any derived shape (Circle,
// Rectangle, Triangle, etc…), but not for “just a Shape and nothing more”.
private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
virtual void draw() const {…draw a circle of radius r centered at (xc,yc)…}
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```

Inheritance - Pure Virtual Functions

“Abstract” base classes

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
void move(double x_new, double y_new);
virtual void draw() const {… draw a cloud centered at (xc,yc) …}
virtual double getArea() const = 0;
// fix: make getArea() a “pure virtual function” by adding “= 0”
// tells compiler:
    // (1) all kinds of Shapes have this function
    // (2) this function will never be called by “only a Shape and nothing more”
private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
virtual void draw() const {…draw a circle of radius r centered at (xc,yc)…}
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```

Inheritance - Pure Virtual Functions

“Abstract” base classes

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
void move(double x_new, double y_new);
virtual void draw() const {… draw a cloud centered at (xc,yc) …}
virtual double getArea() const = 0;
// fix: make getArea() a “pure virtual function” by adding “= 0”
// tells compiler:
    // (1) all kinds of Shapes have this function
    // (2) this function will never be called by “only a Shape and nothing more”
private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
virtual void draw() const {…draw a circle of radius r centered at (xc,yc)…}
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```

- When a class contains a pure virtual function it becomes “abstract”.
- An abstract class cannot be instantiated

Shape s(0,0); // now a compiler error!

Inheritance - Pure Virtual Functions

"Abstract" base classes

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
void move(double x_new, double y_new);
virtual void draw() const = 0;
virtual double getArea() const = 0;
// fix: make getArea() a "pure virtual function" by adding "= 0"
// tells compiler:
    // (1) all kinds of Shapes have this function
    // (2) this function will never be called by "only a Shape and nothing more"
private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
virtual void draw() const {…draw a circle of radius r centered at (xc,yc)…}
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```

- When a class contains a pure virtual function it becomes "abstract".
- An abstract class cannot be instantiated

Shape s(0,0); // now a compiler error!

… note that we might as well make draw() pure virtual at this point…

Inheritance - Pure Virtual Functions

“Abstract” base classes

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
void move(double x_new, double y_new);
virtual void draw() const = 0;
virtual double getArea() const = 0;
private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
virtual void draw() const {…draw a circle of radius r centered at (xc,yc)…}
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```

Q. What is the problem with the current state of our code?

Inheritance - Pure Virtual Functions

“Abstract” base classes

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
void move(double x_new, double y_new);
virtual void draw() const = 0;
virtual double getArea() const = 0;
private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
virtual void draw() const {…draw a circle of radius r centered at (xc,yc)…}
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```

Q. What is the problem with the current state of our code?

A. Circle inherits the pure virtual function “getArea()” and thus Circle becomes abstract.

Inheritance - Pure Virtual Functions

“Abstract” base classes

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
void move(double x_new, double y_new);
virtual void draw() const = 0;
virtual double getArea() const = 0;
private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
virtual double getArea() { return pi*(r**2) };
virtual void draw() const {…draw a circle of radius r centered at (xc,yc)…}
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```

Q. What is the problem with the current state of our code?

A. Circle inherits the pure virtual function “getArea()” and thus Circle becomes abstract.

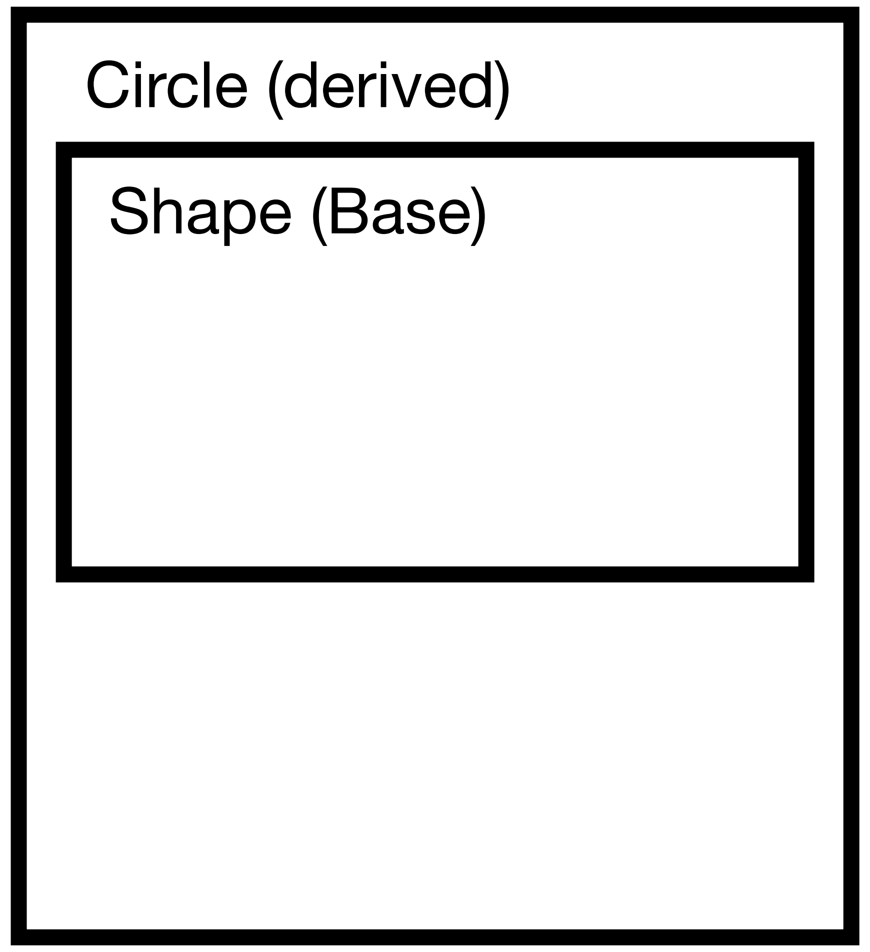
Fix: overload getArea() in Circle class.

Inheritance - closing remarks

Construction & Destruction

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
void move(double x_new, double y_new);
virtual void draw() const = 0;
virtual double getArea() const = 0;
private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
virtual double getArea() { return pi*(r**2) };
virtual void draw() const {…draw a circle of radius r centered at (xc,yc)…}
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```

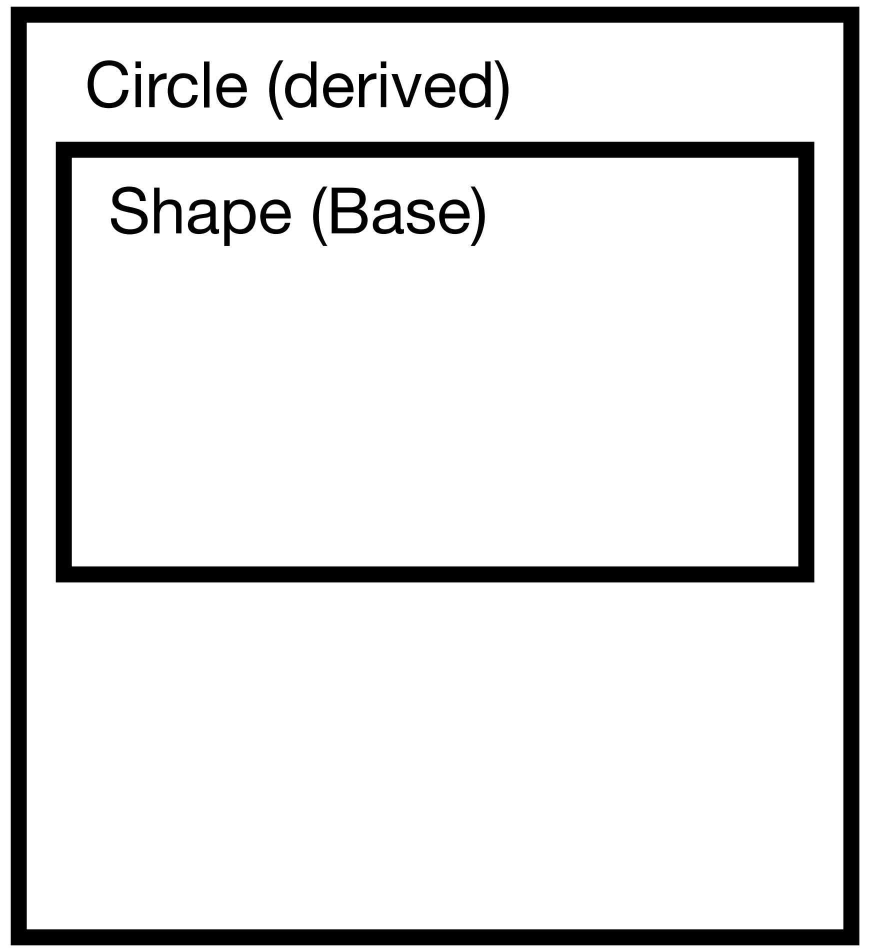


Inheritance - closing remarks

Construction & Destruction

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
void move(double x_new, double y_new);
virtual void draw() const = 0;
virtual double getArea() const = 0;
private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
virtual double getArea() { return pi*(r**2) };
virtual void draw() const {…draw a circle of radius r centered at (xc,yc)…}
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```



Order of Construction:

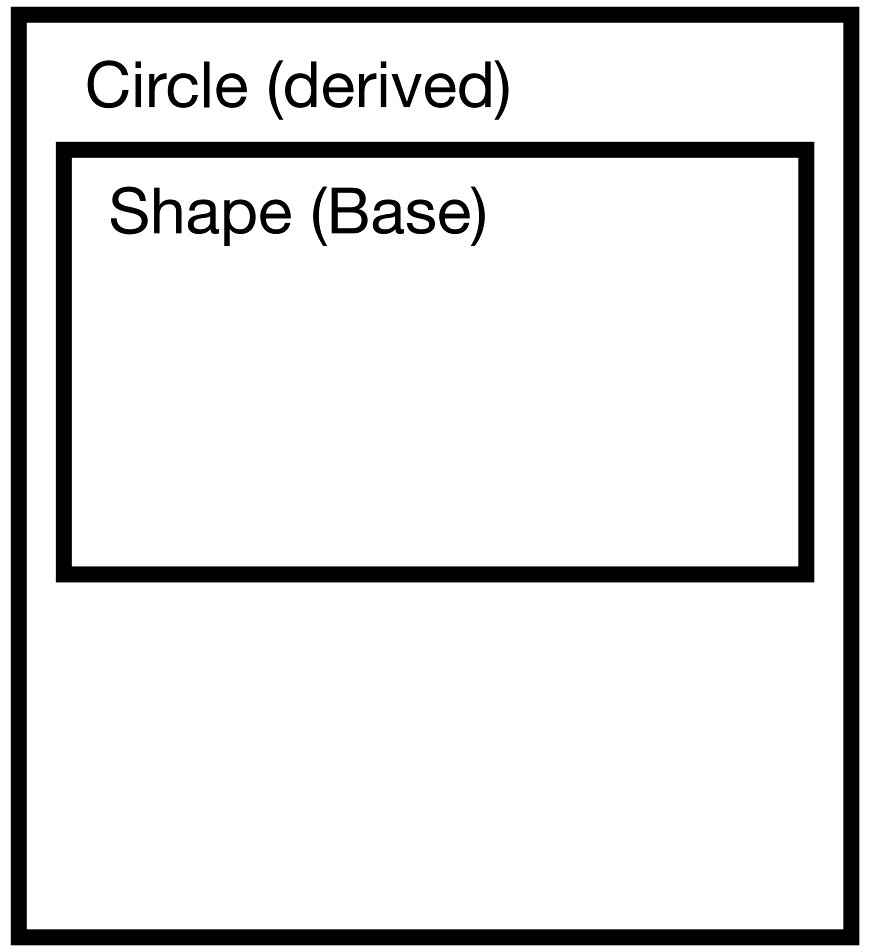
1) Construct base part.
2) Construct data members.
3) Execute constructer body.

Inheritance - closing remarks

Construction & Destruction

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
void move(double x_new, double y_new);
virtual void draw() const = 0;
virtual double getArea() const = 0;
private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
virtual double getArea() { return pi*(r**2) };
virtual void draw() const {…draw a circle of radius r centered at (xc,yc)…}
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```



Order of Construction:

1) Construct base part.
2) Construct data members.
3) Execute constructer body.

Order of Destruction:

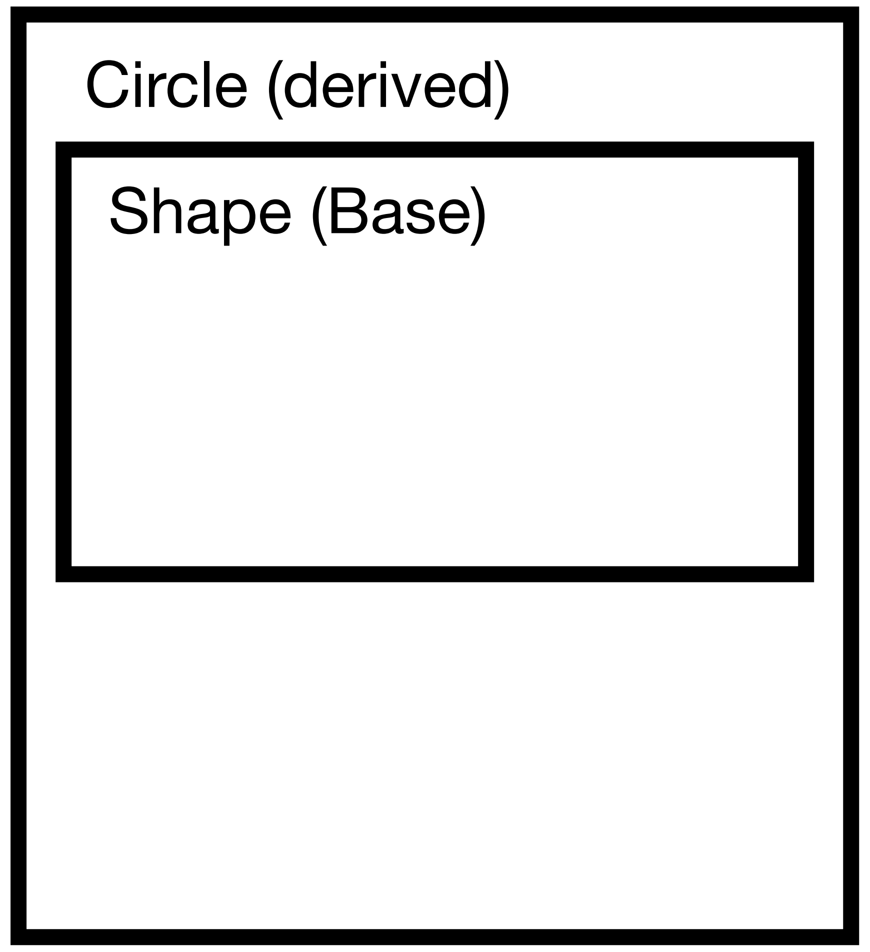
1) Execute destructor body.
2) Destroy data members.
3) Destroy base part.

Inheritance - closing remarks

Construction & Destruction

```
class Shape // (Base class)
{
public:
Shape(double x, double y) : xc(x), yc(y) { }
void move(double x_new, double y_new);
virtual void draw() const = 0;
virtual double getArea() const = 0;
virtual ~Shape(); // Always need to declare Base class destructor as virtual! (and implement it)
private:
double xc; // x-coord of center
double yc; // y-coord of center
};

class Circle : public Shape // (Derived class)
{
public:
virtual double getArea() { return pi*(r**2) };
virtual void draw() const {…draw a circle of radius r centered at (xc,yc)…}
Circle(double x, double y, double r) : Shape(x,y), r(r) { }
double getRadius() {return r;}
private:
double r; // inherits x,y as well
};
```



Order of Construction:
1) Construct base part.
2) Construct data members.
3) Execute constructer body.

Order of Destruction:
1) Execute destructor body.
2) Destroy data members.
3) Destroy base part.

Recursion time.

Recursion

When a function makes a call to itself, usually
on one or more smaller “subproblems”.

Recursion

When a function makes a call to itself, usually
on one or more smaller “subproblems”.

Confusing at first…

but often leads to extremely elegant code…

Recursion

When a function makes a call to itself, usually on one or more smaller "subproblems".

Confusing at first…

but often leads to extremely elegant code…

and memes.

DAD! THERE IS A RECURSION UNDER MY BED!
DAD! THERE IS A RECURSION UNDER MY BED!