Inheritance

CSCI 201
Principles of Software Development

Jeffrey Miller, Ph.D.
jeffrey.miller@usc.edu

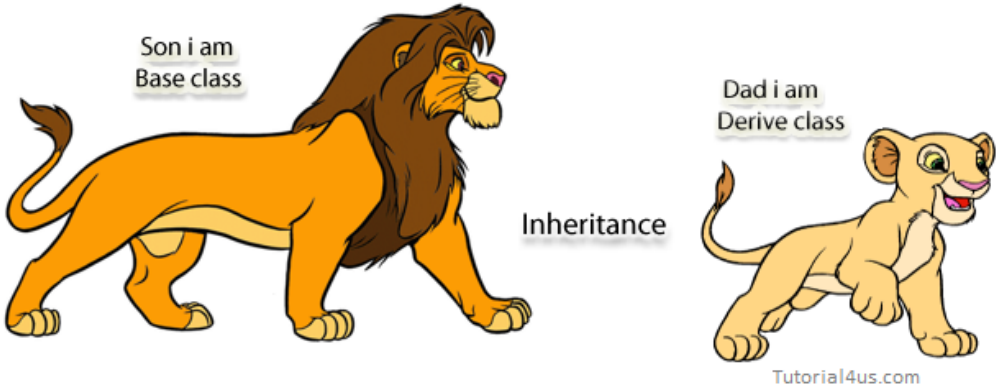
USC Viterbi
School of Engineering

Outline

- Inheritance

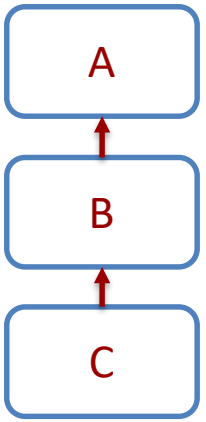
Inheritance

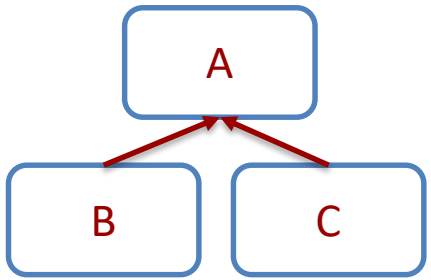
- Inheritance is a form of software reuse in which you create a class that absorbs an existing class's data and behaviors and enhances them with new capabilities
- When creating a class, you can designate that the new class should inherit the members of an existing class
  - The existing class is called the base class (or parent class)
  - The new class is called the derived class (or child class)



Single vs Multiple Inheritance

- Single inheritance allows a derived class to inherit from only one base class
  - Java supports single inheritance
- Multiple inheritance allows a derived class to inherit from more than one base class
  - C++ supports multiple inheritance
  - What if more than one parent implements the same function?





Multiple Inheritance in C++

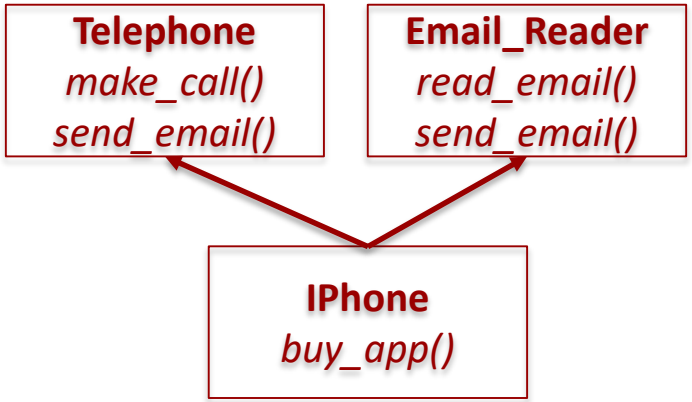
```
1 #include <iostream>
2 using namespace std;
3
4 class Email_Reader {
5   public:
6     void read_email() {
7       cout << "reading email" << endl;
8     }
9 };
10
11 class Telephone {
12  public:
13    void make_call() {
14      cout << "making call" << endl;
15    }
16 };
17
```

```
18 class IPhone : public Telephone, public Email_Reader {
19   public:
20     void buy_app() {
21       cout << "buying app" << endl;
22     }
23 };
24
25 void main() {
26   IPhone ip;
27   ip.buy_app();
28   ip.make_call();
29   ip.read_email();
30 }
```

Multiple Inheritance in C++

```
1 #include <iostream>
2 using namespace std;
3
4 class Email_Reader {
5   public:
6     void read_email() {
7       cout << "reading email" << endl;
8     }
9     void send_email() {
10      cout << "Email sending email";
11    }
12 };
13
14 class Telephone {
15   public:
16     void make_call() {
17       cout << "making call" << endl;
18     }
19     void send_email() {
20       cout << "Telephone sending email";
21     }
22 };
```

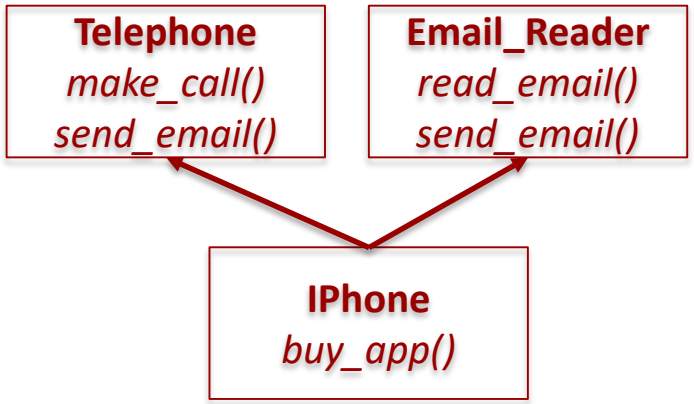
```
23 class IPhone : public Telephone, public Email_Reader {
24   public:
25     void buy_app() {
26       cout << "buying app" << endl;
27     }
28 };
29
30 void main() {
31   IPhone ip;
32   ip.buy_app();
33   ip.make_call();
34   ip.read_email();
35   ip.send_email(); // does this line compile?
36 }
```



Multiple Inheritance in C++

```
1 #include <iostream>
2 using namespace std;
3
4 class Email_Reader {
5   public:
6     void read_email() {
7       cout << "reading email" << endl;
8     }
9     void send_email() {
10      cout << "Email sending email";
11    }
12 };
13
14 class Telephone {
15   public:
16     void make_call() {
17       cout << "making call" << endl;
18     }
19     void send_email() {
20       cout << "Telephone sending email";
21     }
22 };
```

```
23 class IPhone : public Telephone, public Email_Reader {
24   public:
25     void buy_app() {
26       cout << "buying app" << endl;
27     }
28 };
29
30 void main() {
31   IPhone ip;
32   ip.buy_app();
33   ip.make_call();
34   ip.read_email();
35   ip.Telephone::send_email();
36   ip.Email_Reader::send_email();
37 }
```



Inheritance vs Composition

- is-a Relationship **Inheritance**
  - If an object has an “is-a” relationship with another object, inheritance will be used
  - Vehicle, Car, Truck, Motorcycle
- has-a Relationship **Composition**
  - If an object has a “has-a” relationship with another object, composition will be used
  - Car, Steering Wheel, Brake Pedal, Speedometer

Access Methods

- `public`
  - Any other class has access to public member variables and methods
- `protected`
  - Subclasses and classes within the same package have access to protected member variables and methods
- <package>
  - Other classes within the same package have access to member variables and (which is the default access)
- `private`
  - Only the current class has access to the member variables and methods

Instantiating a Child Class

- To inherit from another class, use the keyword `extends` immediately following the name of the class, followed by the name of the class from which you would like to inherit
- When a child class is instantiated, the parent class must be instantiated **first** in the child class's constructor
  - This will happen automatically by the compiler calling the parent class's default constructor unless we explicitly instantiate the parent
  - Note that if there is no default constructor in the parent, we MUST explicitly call the parent class's constructor from the child
- When we call the parent class's constructor from the child, it must be the first line of code in the child class's constructor

```
1 public class Shape {
2   protected char name;
3   public Shape(char n) {
4       name = n;
5   }
6 }
```

```
1 public class TwoDShape extends Shape {
2   public TwoDShape(char name){
3     super(name);
4   }
5 }
```

Inheritance Example

```
1   class Parent {
2     private int num;
3     public Parent(int num) {
4       this.num = num;
5     }
6     public int meth() {
7       return num;
8     }
9   }
10
11  public class Child extends Parent {
12    public Child() {
13
14    }
15    public static void main(String [] args) {
16      Child c = new Child();
17      System.out.println(c.meth());
18    }
19  }
```

Inheritance Solution #1

```
1   class Parent {
2     private int num;
3     public Parent(int num) {
4       this.num = num;
5     }
6     public int meth() {
7       return num;
8     }
9   }
10
11  public class Child extends Parent {
12    public Child() {
13      super(10);
14    }
15    public static void main(String [] args) {
16      Child c = new Child();
17      System.out.println(c.meth());
18    }
19  }
```

Inheritance Solution #2

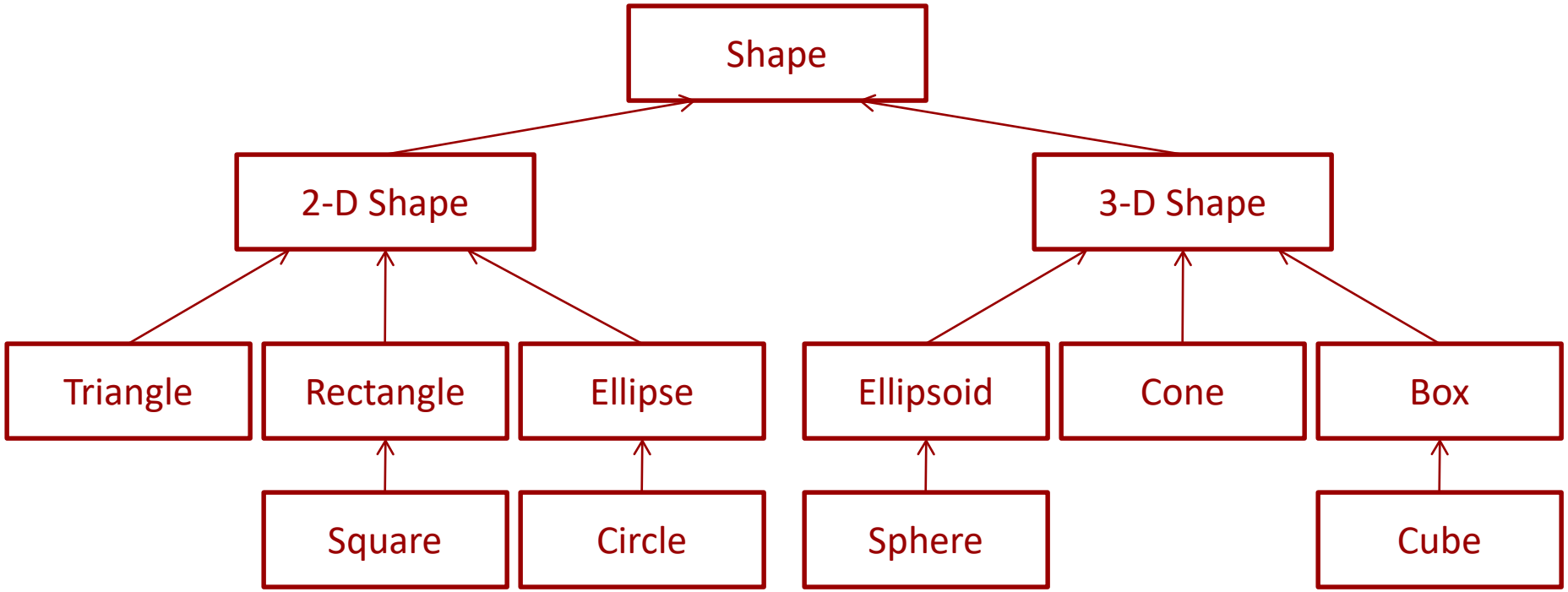
```
1   class Parent {
2     private int num;
3     public Parent() {
4
5     }
6     public Parent(int num) {
7       this.num = num;
8     }
9     public int meth() {
10      return num;
11    }
12  }
13
14  public class Child extends Parent {
15    public Child() {
16
17    }
18    public static void main(String [] args) {
19      Child c = new Child();
20      System.out.println(c.meth());
21    }
22  }
```

Inheritance Solution #3

```
1   class Parent {
2     private int num;
3     // public Parent(int num) {
4     //   this.num = num;
5     // }
6     public int meth() {
7       return num;
8     }
9   }
10
11  public class Child extends Parent {
12    public Child() {
13
14    }
15    public static void main(String [] args) {
16      Child c = new Child();
17      System.out.println(c.meth());
18    }
19  }
```

Inheritance Hierarchy

- To show a child class and a parent class in a diagram, we draw a line connecting the child class to the parent class where the parent class is above the child
- Assume the following hierarchy for the rest of this lecture



Inheritance Example

```
1   class Shape {
2     protected char name;
3     public Shape(char n) {
4       name = n;
5     }
6     public void printName() {
7       System.out.println(name);
8     }
9   }
```

```
1   class TwoD extends Shape {
2     public TwoD(char name) {
3       super(name);
4     }
5   }
```

```
1   class Square extends Rectangle {
2     public Square(char nm, float s) {
3       super(nm, s, s);
4     }
5   }
```

```
1   class Triangle extends TwoD {
2     private float base;
3     private float height;
4     public Triangle(char nm, float b, float h) {
5       super(nm);
6       base = b;
7       height = h;
8     }
9     public float getArea() {
10      return 0.5f * base * height;
11    }
12  }
```

```
1   class Rectangle extends TwoD {
2     protected float width, length;
3     public Rectangle(char nm, float w, float l) {
4       super(nm);
5       width = w;
6       length = l;
7     }
8     public float getArea() {
9       return width * length;
10    }
11  }
```

Inheritance Example (cont.)

```
1   public class Test {
2     public static void main() {
3     Triangle t = new Triangle('t', 3.0f, 4.0f);
4     t.printName();
5     System.out.println(t.getArea());
6     Rectangle r = new Rectangle('r', 5.0f, 6.0f);
7     r.printName();
8     System.out.println(r.getArea());
9     Square sq = new Square('s', 3);
10    sq.printName();
11    System.out.println(sq.getArea());
12  }
```