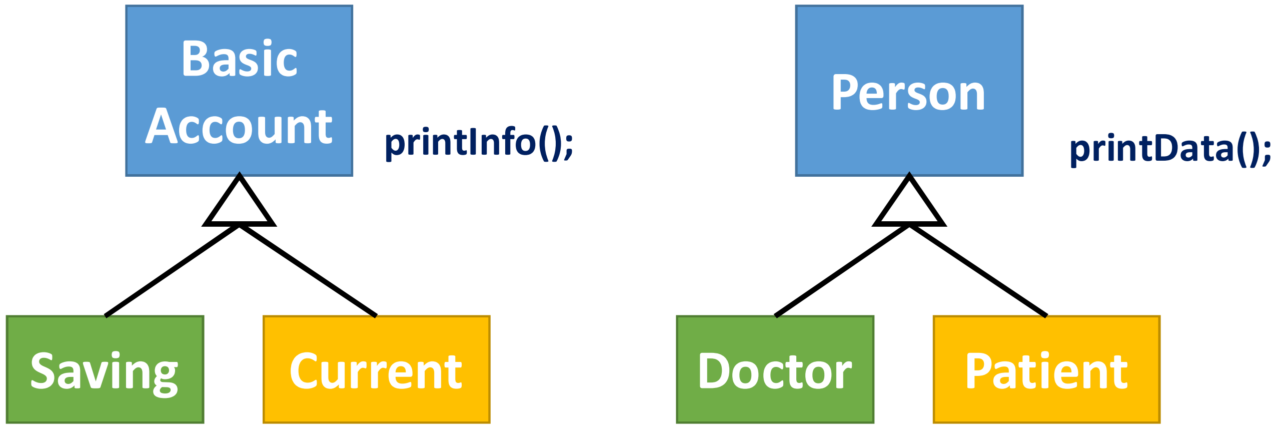
Class/Object Relationships

CS(217) Object Oriented Programming

Abstract Class and Interface

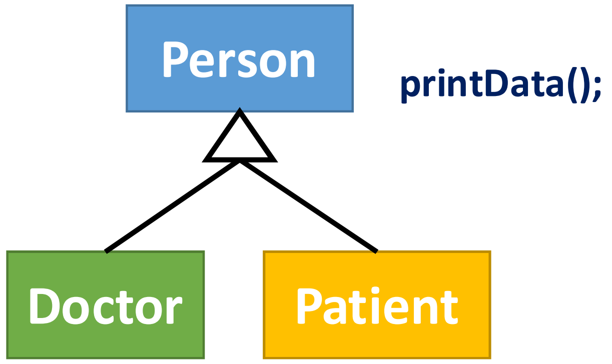
Inheritance (is-a) **Concrete classes**

- Can instantiate or create objects (dynamic and static)
- Can also make pointers and references.
- Implement all functions defined in a concrete class.
- Represent a real time object.



Inheritance (is-a) **Concrete classes**

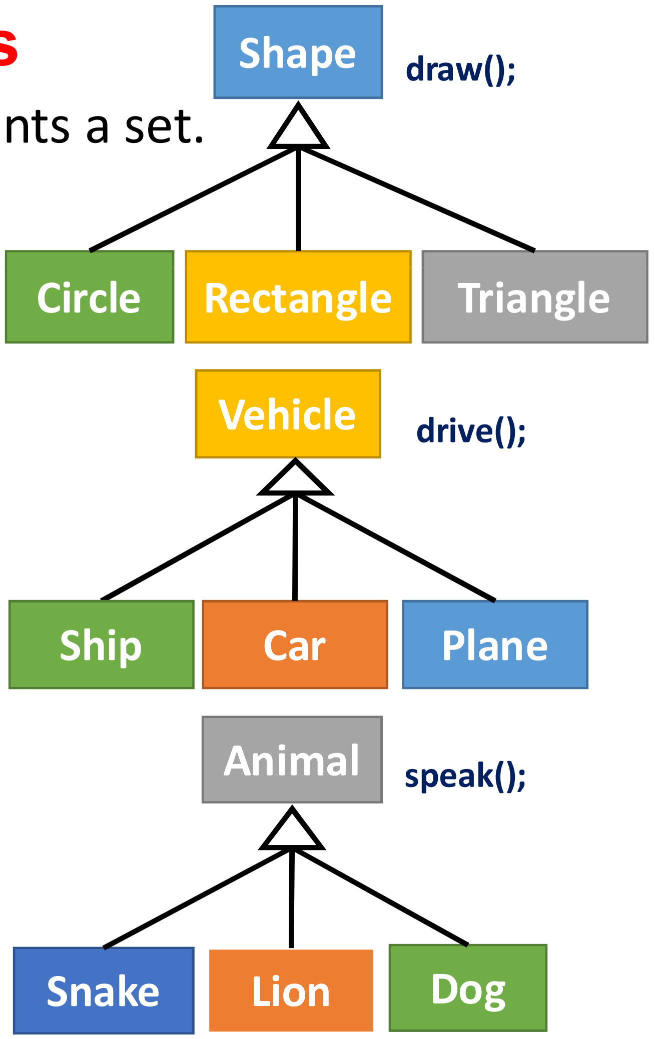
```
void main(){
    Person * p = new Person(1234, “Ali”, “Jamal”, 3, 9, 1989);
    p->printData(); // Print person’s data

    Person * p1 = new Doctor(1234, “Salman”, “Raza”, 3, 9, 1989, “cardiologist”…);
    p1->printData(); // Print Doctors’s data

    Person * p2 = new Patient(1234, “Kamran”, “Shahid”, 3, 9, “Blood Pressure” ..);
    p2->printData(); // Print Patient’s data
}
```



Inheritance (is-a) **Abstract classes**

- Abstract class is a concept not a real object, represents a set.

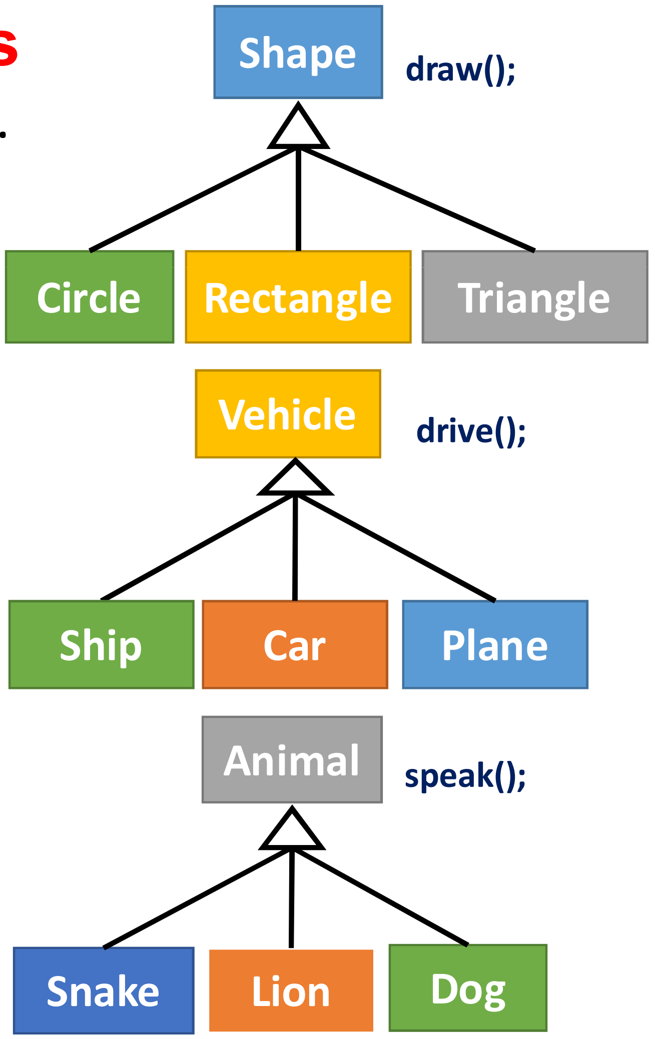
```
class Shape{
    Point p;
public:
    Shape(int x=0, int y=0) :p(x,y){}
    virtual void draw(){// What to draw?}
};
class Vehicle{
public:
    virtual void drive(){// What to drive?}
};
class Animal{
public:
    virtual void speak(){// What to speak?}
};
```



Inheritance (is-a) **Abstract classes**

- System allows us to make objects of all base classes.
- It is useless to make objects of abstract class.

```
void main(){
    Shape * s = new Shape(3, 2);
    s->draw(); // What to draw ?

    Shape * s2 = new Circle(3, 2, 5.5);
    s2->draw(); // Draw a circle
    Shape * s3 = new Rectangle(3, 2, 5.5, 6);
    s3->draw(); // Draw a Rectangle

    Animal * a = new Animal;
    a->speak(); // What to Speak ?

    Animal * a2 = new Lion;
    a2->speak(); // Lion Roars

    Vehicle * v = new Vehicle;
    v->drive(); // What to drive ?
}
```



Inheritance (is-a) **Abstract classes**

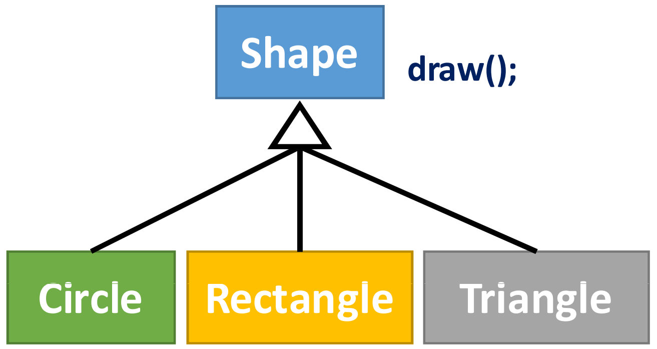
- Sole purpose of abstract class is to be a base class
  - Place common attributes and functions in abstract base class.
- Implementation is Incomplete for some functions
  - Derived classes should fill in "missing pieces“ according to their type.
- Cannot create objects (dynamic or static) of abstract class
  - Compiler will show error message, if create object of abstract class.
  - However, can create pointers and references of abstract classes.
    - Abstract class pointers are still useful for polymorphism.
    - We can still make pointer arrays of abstract base classes to store different type of derived objects.

Shape

Vehicle

Animal

Inheritance (is-a) **Abstract classes**



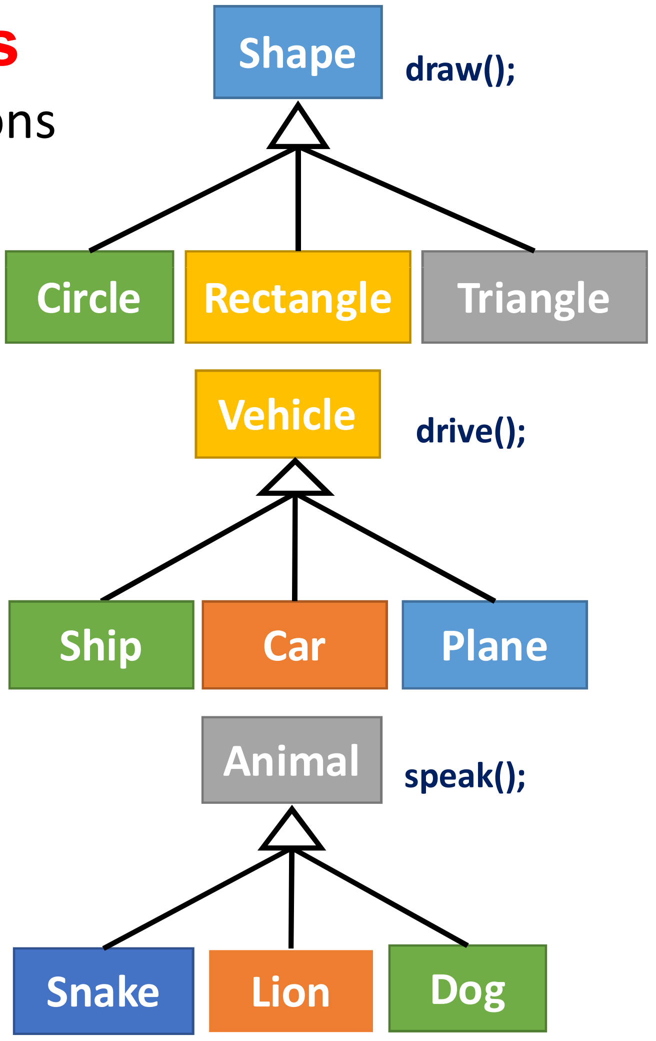
- How to make a class abstract?
  - Add one or more "pure" virtual functions in the class.
    - Declare function with initializer of 0

      ```
      virtual void draw() = 0;
      ```
    - **Pure virtual functions**, have no implementation, just add prototype in base class.
    - Must be overridden in derived classes according to their type.
    - If a derive class do not override a pure virtual function, it also becomes an abstract class.
  - Normal virtual functions have implementations, overriding is optional
  - Abstract classes can still have data and concrete functions
    - Just required to have one or more pure virtual functions
  - Must add virtual destructor in abstract class.
  - Abstract class can has constructors

Inheritance (is-a) **Abstract classes**

- Make abstract classes by adding pure virtual functions
- Must be overridden by derived classes.

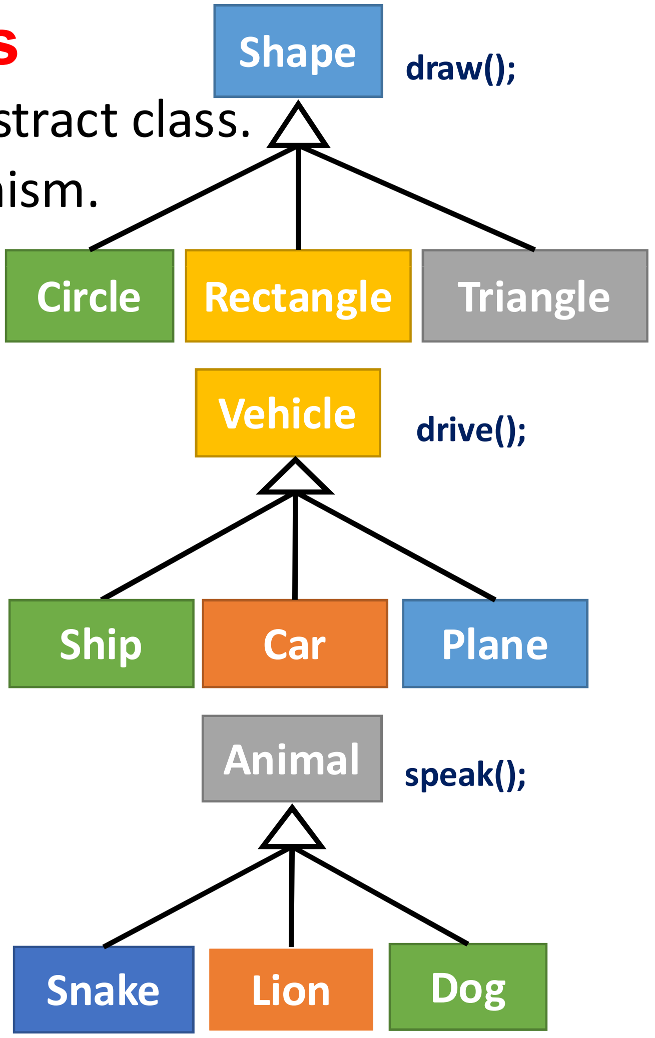
```
class Shape{
    Point p;
public:
    Shape(int x=0, int y=0) :p(x,y){}
    virtual void draw() = 0; // Pure virtual function
};
class Vehicle{
public:
    virtual void drive() = 0; // Pure virtual function
};
class Animal{
public:
    virtual void speak() = 0; // Pure virtual function
};
```



Inheritance (is-a) **Abstract classes**

- Now system will not allow us to make objects of Abstract class.
- Abstract class pointers are still useful for polymorphism.

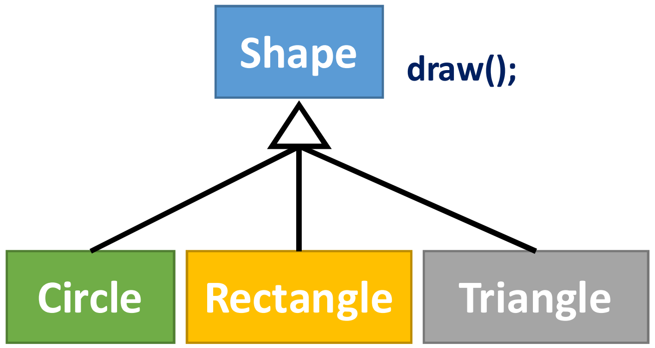
```
void main(){
    Shape * s = new Shape(3, 2);
    // Error: cannot create object of abstract class.
    // Can still point to derived class objects.
    Shape * s2 = new Circle(3, 2, 5.5);
    s2->draw(); // Draw a circle
    Shape * s3 = new Rectangle(3, 2, 5.5, 6);
    s3->draw(); // Draw a Rectangle

    Animal * a = new Animal;
    // Error: cannot create object of abstract class.

    Animal * a2 = new Lion;
    a2->speak(); // Lion Roars

    Vehicle * v = new Vehicle;
    // Error: cannot create object of abstract class.
}
```



Inheritance (is-a) **Abstract classes**

- We can still make pointer arrays of abstract base classes to store different type of derived objects.

```
void main(){
    Shape ** shapes = new Shape*[10];
    shapes[0] = new Circle(2,9,5);
    shapes[1] = new Rectangle(3,8,9,7);
    shapes[2] = new Triangle(3,2,6,5);
    ….
//Draw different shapes according to type
    for(int i=0; i<10 ;i++)
        shapes [i]->draw();


//Deallocate list
    for(int i=0; i<10 ;i++)
        delete shapes[i];
    delete [] shapes;
}
```



Inheritance (is-a) **Interface**

- A special type of class, which has no data members.
- All functions are pure virtual.
- Derived class must implement all pure virtual functions.
  - If it skip implementation of any function, then it will become an abstract class, its object cannot be created.
- Must add virtual destructor in interface class.
- Name of Interface classes usually begin with letter “I”.
  - Just to differentiate it from other classes.

Inheritance (is-a) **Interface**

```
class IExample{
public:
// Pure virtual functions
    virtual void Function1() = 0;
    virtual void Function2() = 0;
    virtual void Function3() = 0;
    virtual ~IExample();
};
class A: public IExample{
    int a;
public:
    A(int a=0){ this->a=a;}
    virtual void print(){ cout<<a;}
    virtual ~A(){}
    // Must implement all Pure virtual functions.
};
```