Software Workshop No. 5

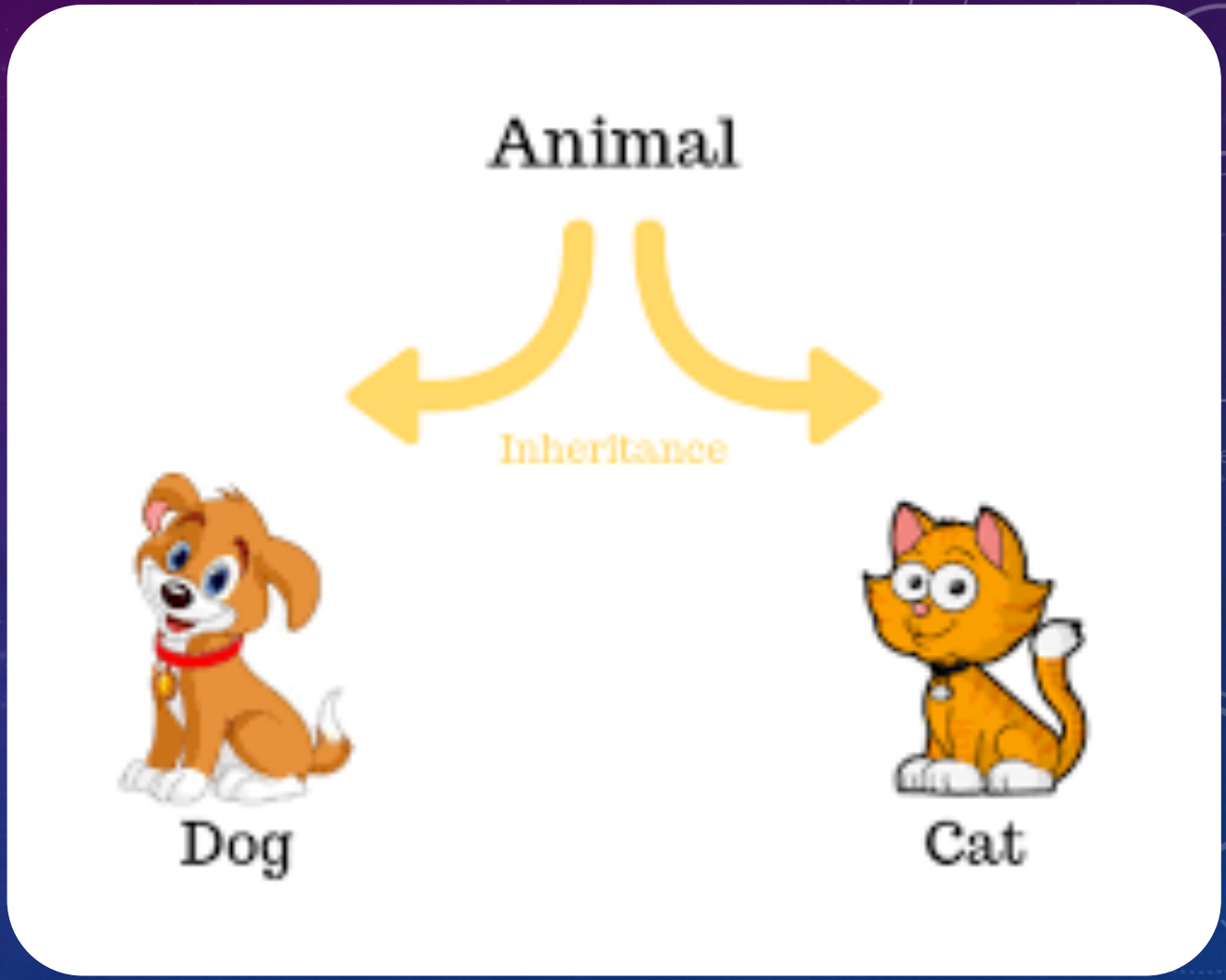
UCSB Robotics, Fall 2020 | Alex Mei

LAST TIME...

- Stacks
- Queues
- Command Line
- Github

TODAY'S AGENDA

- Inheritance
- Polymorphism
- Abstract Classes



INHERITANCE

- Definition: a class can inherit members from a parent class
- Example: a motorbike is a child class of a bike class and has all the properties of a regular bike + more

```
class Bicycle{
    public:
        Bicycle(int r, int s);
        int getWheelRadius() const;
        int getBikeSize() const;

    protected:
        int wheelRadius;
        int bikeSize;
};
```

```
class MotorBike : public Bicycle{
    public:
        MotorBike(int r, int s);
        int getBikeSize() const;
        void setVelocity(double v);
        double getVelocity() const;

    private:
        double velocity;
};
```

INHERITED MEMBERS

- Protected: keyword to signify that members of the class are accessible to the class and all children
- The visibility denoted after the colon for the parent class defines how members are inherited
  - public: all public and protected members from the parent are inherited with their respective visibility
  - protected: all public and protected members from the parent are inherited with protected visibility
  - private: all public and protected members from the parent are inherited with private visibility

INHERITED CLASSES

- Constructor: a call needs to be made to the parent class to initialize the properties of the parent
- Members: children can utilize parent members, but parents cannot use children members

```
Bicycle::Bicycle(int r, int s){
    wheelRadius = r;
    bikeSize = s;
}

MotorBike::MotorBike(int r, int s) : Bicycle(r, s){
    velocity = 0;
}
```

MORE ON CONSTRUCTORS

- Default Constructor: a constructor with zero parameters (one is generated automatically if no constructors defined)
- Parameterized Constructor: a constructor with parameters
  - Note: a class has no limit on the number of constructors if each one takes a different set of arguments
- Copy Constructor: a constructor that takes an instance of the same class as a parameter

INHERITED FUNCTIONS

- Overloading: a child class can overload a function defined in the parent class
- Note: child objects will use the child function definition, while parents will use the parent function definition

```
int Bicycle::getBikeSize() const{
    return bikeSize;
}

int MotorBike::getBikeSize() const{
    return bikeSize * 2;
}
```

```
Bicycle b(5, 10);
MotorBike m(5, 10);
cout << "Bike Size: " << b.getBikeSize() << endl;
cout << "MotorBike Size: " << m.getBikeSize() << endl;
```

```
Bike Size: 10
MotorBike Size: 20
```

YOUR TURN...

- Design the prototype for a Fish class which has members swim speed and name.
- Design the prototype for a Lizard class which evolved/inherits from the Fish class with members swim speed, run speed, and name.
- Design the prototype for a Monkey class which evolved/inherits from the Lizard class with members run speed and name.

POLYMORPHISM

- Definition: a child class can take the form of its parent
- Example: a motorbike is also a bicycle

```
Bicycle b(5, 10);
MotorBike m(5, 10);
Bicycle mb = m;
cout << "Bike Size: " << b.getBikeSize() << endl;
cout << "MotorBike Size: " << m.getBikeSize() << endl;
cout << "MotorBike turned Bike Size: " << mb.getBikeSize() << endl;
```

```
Bike Size: 10
MotorBike Size: 20
MotorBike turned Bike Size: 10
```

ABSTRACT CLASSES

- Definition: an abstract class which cannot be instantiated and only intended to be inherited from
- Pure Virtual Functions: functions that are required to be implemented by inherited classes
  - Syntax: virtual returnType functionName(params) = 0;
- Example: there are no 'shape' objects, but triangles and squares inherit from the shape class with common properties such as area and vertices

ABSTRACT CLASSES

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

class Triangle : public Shape {
    public:
        Triangle(double base, double height);
        virtual double area();
        virtual int getVertices() const;
    private:
        double base;
        double height;
        double vertices;
};
```

```
Triangle::Triangle(double b, double h){
    base = b;
    height = h;
    vertices = 3;
}

double Triangle::area(){
    return base * height / 2;
}

int Triangle::getVertices() const{
    return vertices;
}
```

YOUR TURN...

- Design an abstract Animal class with functions cry( ) and getName( ).
- Implement a Cat class that inherits from Animal. The Cat constructor should take a name and a speed. Its cry should print "Meow!". Cats can also run( ) which prints its horizontal velocity.

  Implement a Duck class that inherits from Animal. The Duck constructor should take a name and a speed. Its cry should print "Quack!". Ducks can also fly( ) which prints its vertical velocity.