Polymorphism...

many shapes

- Static vs. Dynamic type.
  ```cpp
class A
{...

class B : public A
{...
  ```

void main
{
  FarmAnimal* Farm[4];
  Farm[0] = new Cow;
  Farm[1] = new Chicken;
  Farm[2] = new Pig;
  Farm[3] = new Fox;
}

Polymorphism: the ability of an object of a static type, to take “many shapes” i.e. to behave differently, according to its dynamic type.
also known as “Dynamic dispatch”

**EXAMPLE:**

```cpp
class FarmAnimal
{
  virtual void speak() { cout << "..."; }
}

class Cow : public FarmAnimal
{
  void main
  {
    FarmAnimal* Farm[4];
    Farm[0] = new Cow;
    Farm[1] = new Chicken;
    Farm[2] = new Pig;
    Farm[3] = new Fox;
  }
}
```
```cpp
class Cow : public FarmAnimal
{
    void speak()
    { cout << "Moo"; }
};

class Chicken : public FarmAnimal
{
    void speak()
    { cout << "Cluck"; }
};

class Pig : public FarmAnimal
{
    void speak()
    { cout << "Oink!"; }
};

class Fox : public FarmAnimal
{
};

// version 2
class FarmAnimal
{
    virtual void speak() = 0;
};

FarmAnimal Bob;
Fox Dan;

void main
{
    FarmAnimal* Farm[4];
    Farm[0] = new Cow;
    Farm[1] = new Chicken;
    Farm[2] = new Pig;
    Farm[3] = new Fox;

    for(int k=0; k<4; k++)
    { Farm[k]->speak(); }
}
```

- By default, which function to call is decided by static type
- `virtual` = decide which function to use based on dynamic type
- The `virtual` modifier is inherited

- A class with a pure virtual function is called an abstract class
- Abstract classes cannot be instantiated
- An abstract class that consists of only pure virtual functions is called an Interface.
The virtual quality of a function is inherited.

```cpp
class FarmAnimal
{
    virtual void speak() = 0;
    virtual ~FarmAnimal() {}
};

class Cow : public FarmAnimal
{
    virtual void speak()
    { cout << "Moo"; }
}

class Chicken : public FarmAnimal
{
    virtual void speak()
    { cout << "Cluck"; }
}

class CrazyCow : public Cow
{
    virtual void speak()
    { cout << "HaHaHA!"; }
}

void main
{
    FarmAnimal* Farm[3];
    Farm[0] = new Cow;
    Farm[1] = new Chicken;
    Farm[2] = new Pig;

    for(int k=0; k<3; k++)
    { 
        Farm[k]->speak();
    }

    for(int k=0; k<3; k++)
    { 
        delete Farm[k];
    }
}
```
FarmAnimal *p = new Pig;

Pig* q = p;  // Red
Pig* q = dynamic_cast<Pig>( p );  // NULL if not successful
Cow* r = static_cast<Cow >( p );  // works but dangerous..