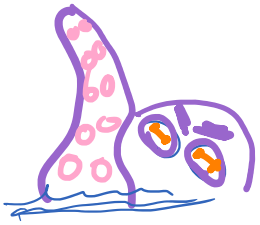
Polymorphism…

Monday, October 14, 2019 5:19 PM

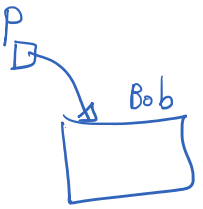


many - shapes

- Static vs. Dynamic type.

```
class A
{...};

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

```
B bob;

A* p = &bob;
```



what is the type of the object pointed by p?

Statically: A

Dynamically: B

Polymorphism: the ability of an object of a static type, to take "Many shapes" i.e. to behave differently, according to it's dynamic type.

also known as "Dynamic dispatch"

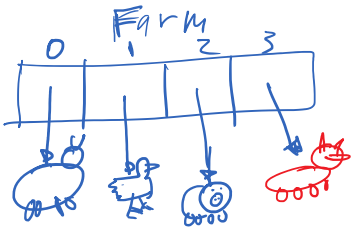
EXAMPLE:

```
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;
```

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

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

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

```
   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 with function to use based on dynamic type

the virtual modifier is inherited

```
class Fox : public FarmAnimal
{
}
```

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

pure virtual function.

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

```
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();
   }

}
```

- The virtual quality of a function is inherited.

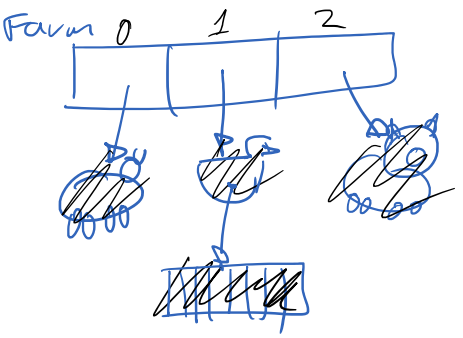
```
class FarmAnimal
{
   virtual void speak() = 0;
}

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!"; }
}
```

- by convention write "virtual" on all virtual functions.

- make Destructors Virtual!!!

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

class Chicken : public FarmAnimal
{
   Egg* nest;

   Chicken() {
        nest = new Egg[8];
   }

   virtual void speak()
   { cout << "Cluck"; }

   ~Chicken() {
      delete [] nest;
   }
}
```

```
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;

Pig* q = dynamic_cast<Pig>( p );  // NULL if not successful

Cow* r = static_cast< Cow >( p ); // works but dangerous..
```