Caleb Starkey, Dylan Stump

Inheritance

- the base (derived/base) class is the parent (parent/child)
- the derived (derived/base) class is the child (parent/child)
- a child (parent/child) has an is-a relationship with the parent (parent/child)

(More) Concretely

- the base class is the parent
- the derived class is the child
- a child is a(n) parent

What is not inherited?

constructors/de-constructors, private fields, and operator overloads, unless specified specifically

What is inherited?

Everything else

How does privacy interact with inheritance?

If unspecified all public methods/fields other than constructors/de-constructors are inherited as private. If public, all public methods/fields are inherited as public. If protected, all public methods/fields are inherited as protected.

Animal

```
class Animal {
public:
    Animal(string sound): sound_(sound) {}
    string MakeSound() {return sound_; }
    virtual int GetSpeed() {return 0; }
private:
    std::string sound_;
}
```

Reptile

```
class Reptile : public Animal {
public:
     Reptile(std::string sound):
     Animal(sound + "rawr") {}

     int GetSpeed() {return 2; }
}
```

Mammal

```
class Mammal : public Animal {
public:
     Mammal():
     Animal("fuzzy fuzz") {}
     int GetSpeed() {return 3; }
}
```

Turtle

```
class Turtle : public Reptile {
public:
     Turtle(): Reptile("turtle turtle") {}
     int GetSpeed() {return 1; }
}
```

```
// We could instantiate some Animals as follows:
Turtle t;
Mammal gopher;
Animal *cow = new Animal("moo");

std::cout << t.MakeSound() << std::endl;
std::cout << gopher.MakeSound() << std::endl;
std::cout << cow->MakeSound() << std::endl;
```

What is the output of the above code?

turtle turtle
fuzzy fuzz
moo

Would the below code work? why/why not?

Yes, they all have the parent type Animal

```
std::vector<Animal> vec = {t, gopher, *(cow)};
```

Dynamic Dispatch

What is dynamic dispatch? How does it relate to the `virtual` keyword?

dynamic dispatch allows for children to overwrite methods designated as virtual from the parent

```
// Now, let's instantiate some more objects as follows:
Animal * t2 = new Turtle();
Animal * m2 = new Mammal();
Animal * r2 = new Reptile("hiss");
```

Would the below code work? why/why not?

```
std::vector<Animal *> vec = {t2, m2, r2};
```

Answer:

This should work because all variables are animal pointers

What method(s) are called in the following code?

```
// which method is being called for these function calls?
for (int i = 0; i < vec.size(); i++) {
   std::cout << vec[i]->MakeSound() << std::endl;
}
```

method(s) called

It calls MakeSound() from the Animal class

What method(s) are called in the following code?

```
// which method is being called for these function calls?
for (int i = 0; i < vec.size(); i++) {
   std::cout << vec[i]->GetSpeed() << std::endl;
}
```

method(s) called

it class GetSpeed() from the Animal class which is dynamically dispatched to the respective child call

What would happen if `GetSpeed()` had not been marked `virtual`?

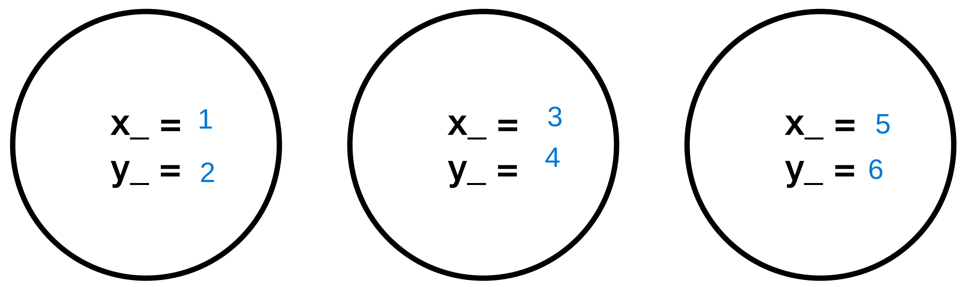
Then it would just call it in the Animal Class

Non static fields

```
Point.h

int x_;
int y_;
```

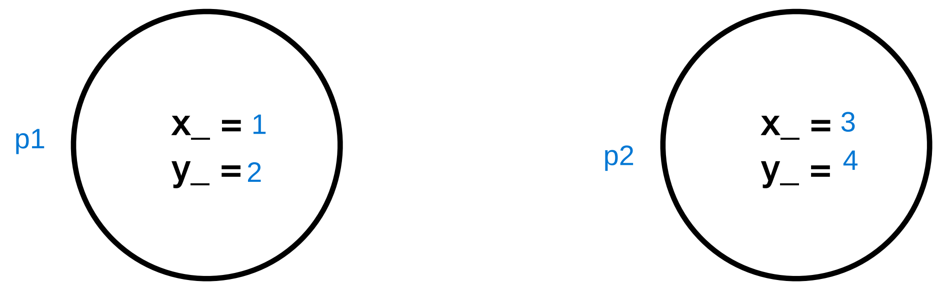
Point instances



Non static methods

```
Point.h

double Distance(const Point & other) const;
```



p1.Distance(p2);

Static fields

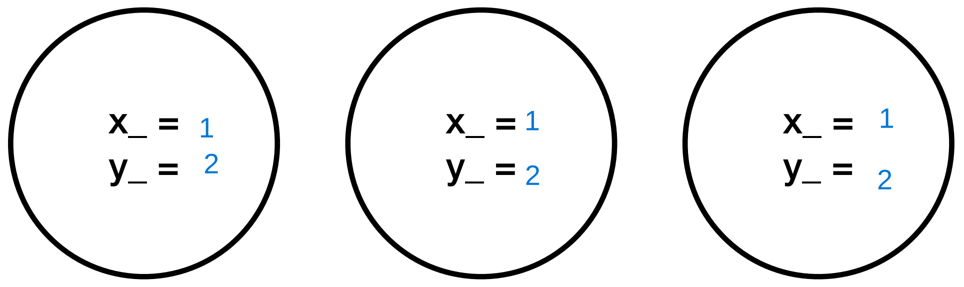
```
Point.h

static int x_;
static int y_;
```

```
Point.cpp

int Point::x_ =  1 ;
int Point::y_ =  2 ;
```

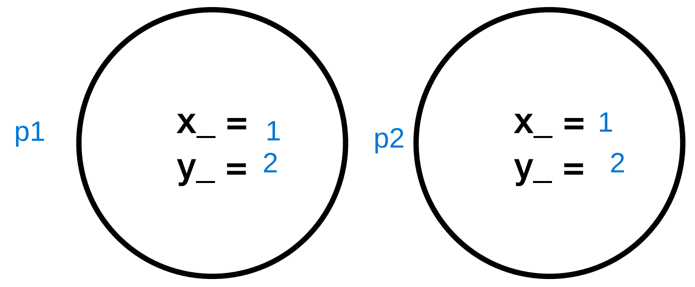
Point instances



Static methods

```
Point.h

static double Distance(const Point & p1, const Point & p2);
```



Point::Distance(p1,p2)