Inheritance

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

(More) Concretely

- the Base class is the Parent
- the Derived class is the Child
- a __________ is a(n) __________

What is not inherited?

Everything in a Parents Private

What is inherited?

Everything in Parent's Public/Protected is inherited.

How does privacy interact with inheritance?

It provides a layer of inheritance protection

- Public
- Protected
- Private

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;  – turtle turtle
std::cout << gopher.MakeSound() << std::endl; – fuzzy fuzz
std::cout << cow->MakeSound() << std::endl;  – moo
```

What is the output of the above code?

Would the below code work? why/why not? Yes

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

1 Calls parent implementation

Dynamic Dispatch

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

the process of selecting w/ implementation of a polymorphic operation to call at run time.

virtual base class is nested innerclass whose functions and attr. can be over ridden & redefined by subclasses of an outer class

```
// Now, let's instantiate some more objects as follows:
Animal * t2 = new Turtle(); - turtle is an animal ✓
Animal * m2 = new Mammal(); - mammal is an animal ✓
Animal * r2 = new Reptile("hiss"); - Rep is an animal
```

Would the below code work? why/why not? Yes, all animals

Answer:

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

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

↳ turtle, turtle
↳ fuzzy fuzz
↳ hiss rawr

• make sound()
↳ parent

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

↳ 0
↳ 2
↳ 1

• Get Speed() - derived
b/c virtual keyword

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

we would get parent calls to Get Speed.

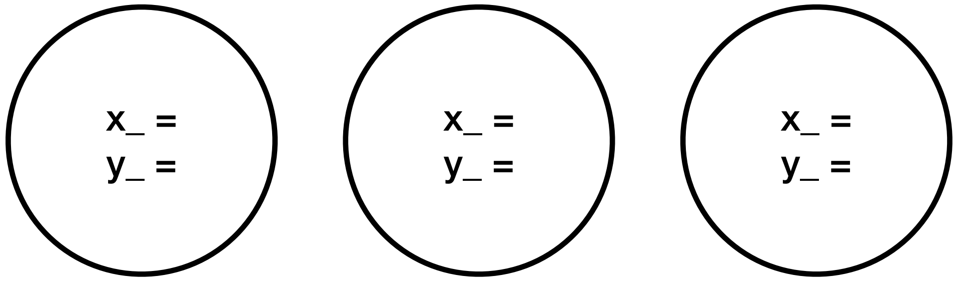
out = 0, 0, 0

Non static fields

```
Point.h

int x_;
int y_;
```

Point instances



x_ =
y_ =

x_ =
y_ =

x_ =
y_ =

Static fields

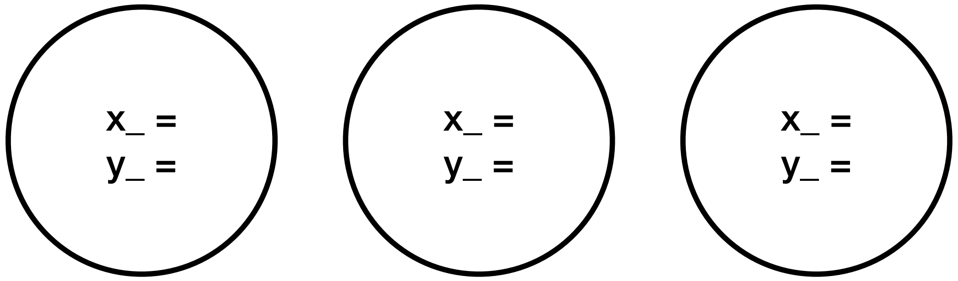
```
Point.h

static int x_;
static int y_;
```

```
Point.cpp

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

Point instances



x_ =
y_ =

x_ =
y_ =

x_ =
y_ =

Non static methods

```
Point.h

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

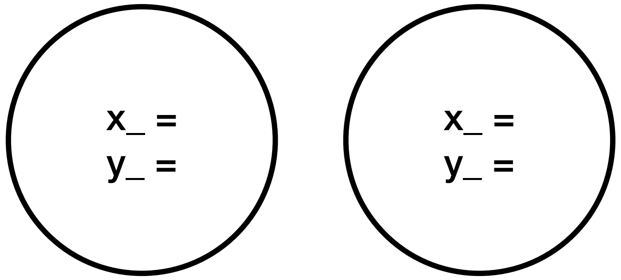
x_ =
y_ =

x_ =
y_ =

Static methods

```
Point.h

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



x_ =
y_ =

x_ =
y_ =