Inheritance

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

(More) Concretely

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

What is not inherited?

private methods/attributes, constructor/destructors, and assignment operators.

What is inherited?

public/protected methods/attributes

How does privacy interact with inheritance?

Only public methods/attributes are passed from the base class to the derived class whereas the private methods/attributes are not. However, the private members can be accessed through public methods of the base class that the derived class inherits.
Protected members can also be accessed by the derived class.

Animal

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

Reptile

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

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

Mammal

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

Turtle

```
class Turtle : public Reptile {
public:
     Turtle(): Reptile("turtle turtle") {}
     int GetPower() {return 7; }
}
```

```
// 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 roar
fuzzy fuzz
moo

Would the below code work? why/why not? **No, the types would conflict. They are not all of 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 is late binding and the virtual keyword allows the correct method to be used at runtime.

```
// 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:

Yes, because they are all of type Animal.

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

The animal make sound function is called.

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]->GetPower() << std::endl;
}
```

method(s) called

It will call the get power method of the corresponding derived classes.

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

It would call the base class' GetPower