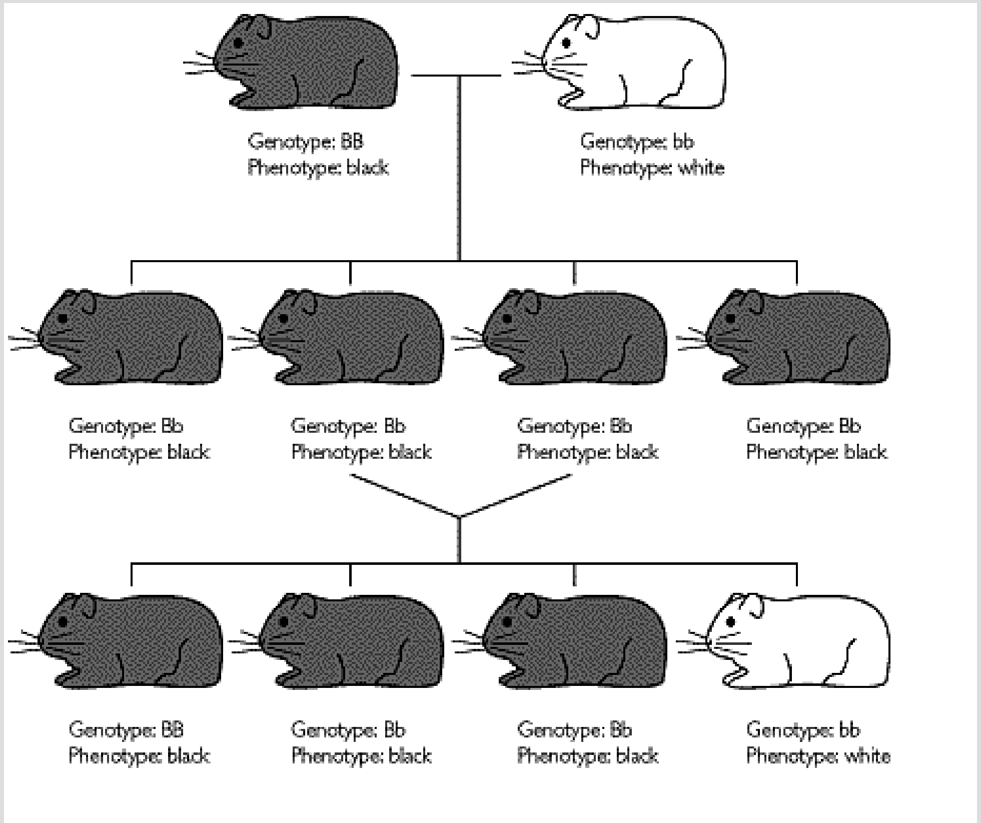
Inheritance

Ch 15.1-15.2



Highlights

- Creating parent/child classes (inheritance)

```
class Parent{
public:
    void foo();
};
```

```
class Child : public Parent {
public:
    Child();
};
```

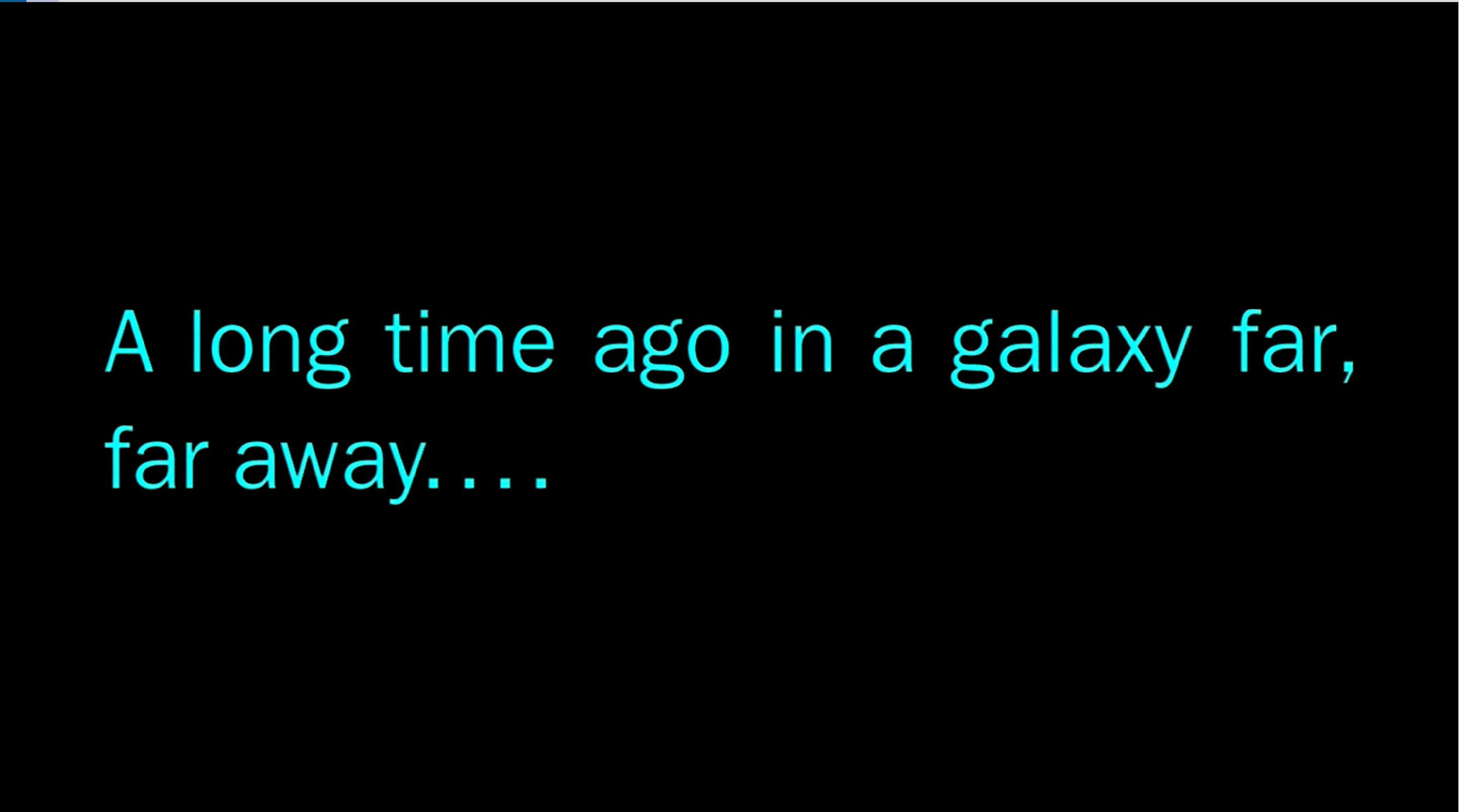
- protected

```
class Parent{
protected:
    int x;
};
```

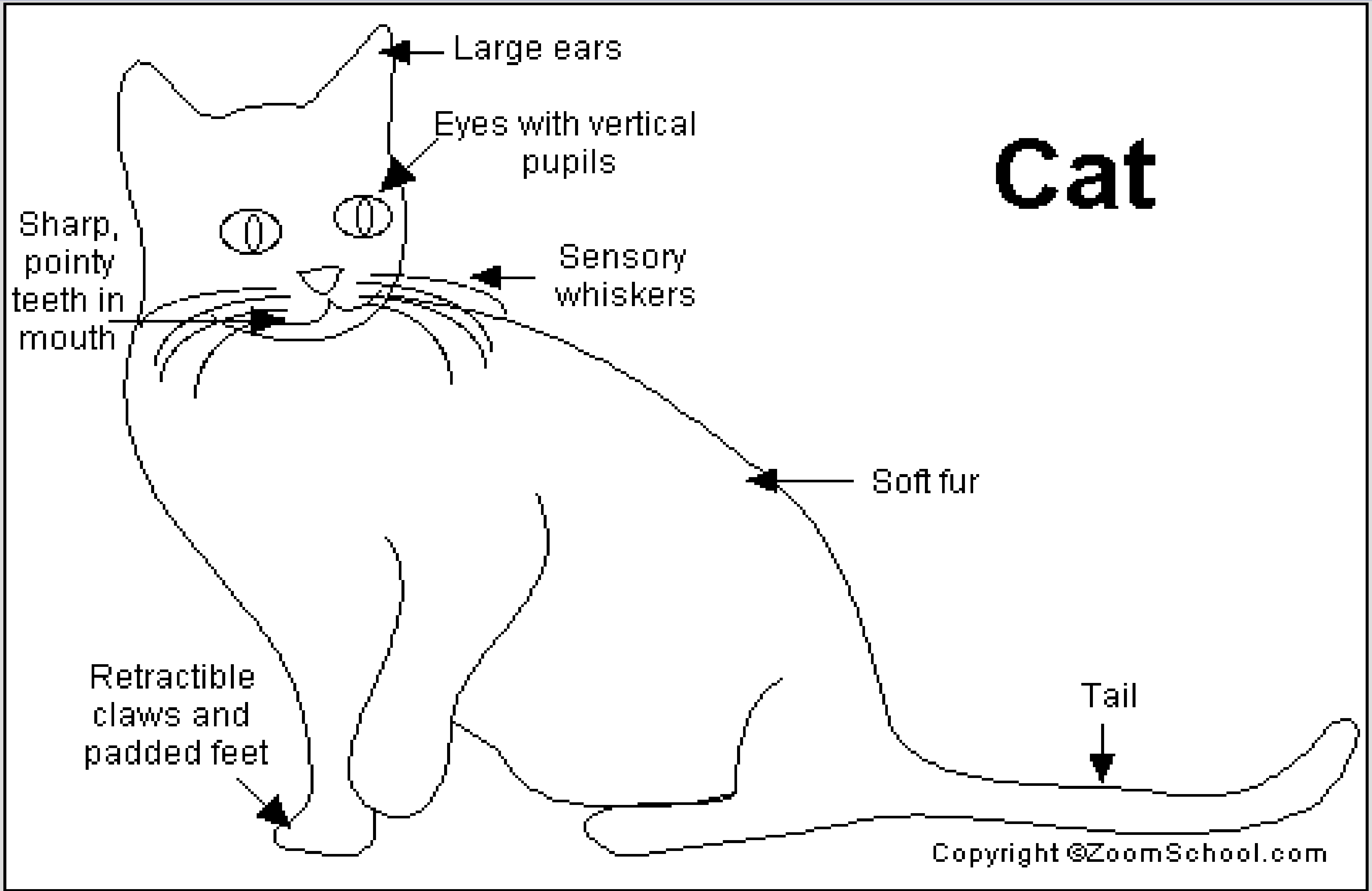
- reuse constructors

```
Child::Child() : Parent()
{
    // runs parent default constructor before itself
}
```

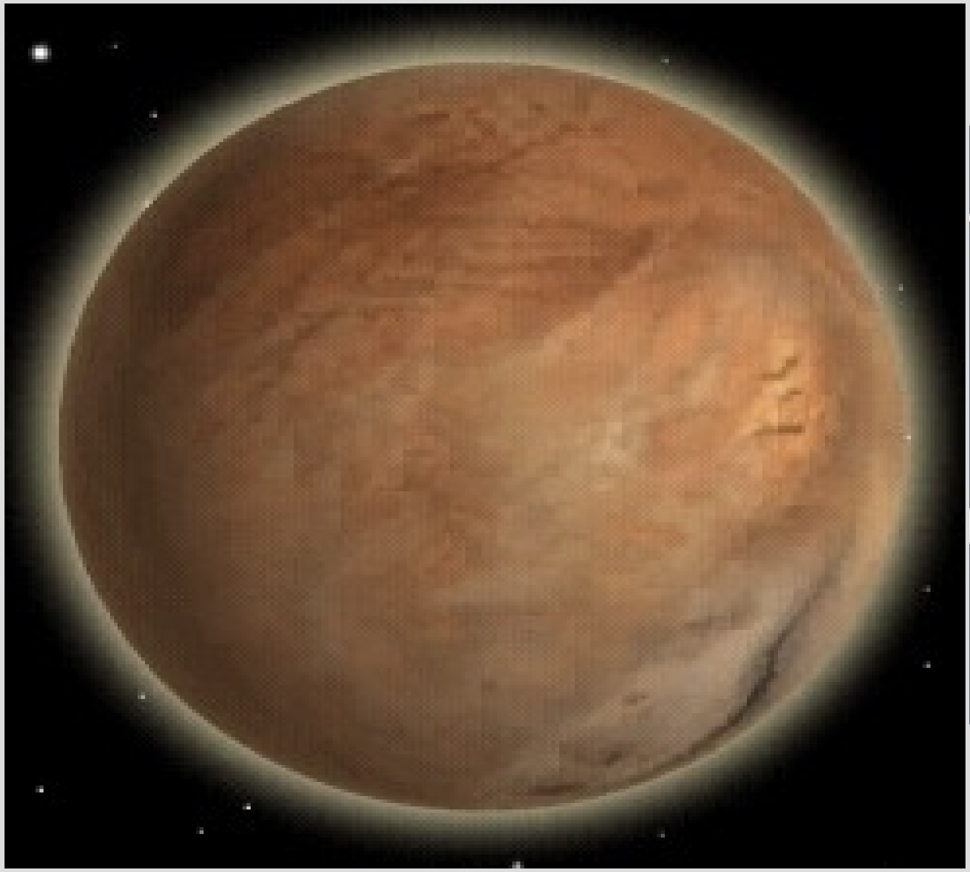
Story time


A long time ago in a galaxy far, far away....

Story time


Large ears
Eyes with vertical pupils
Cat
Sharp, pointy teeth in mouth
Sensory whiskers
Soft fur
Retractible claws and padded feet
Tail
Copyright ©ZoomSchool.com

Story time



Story time

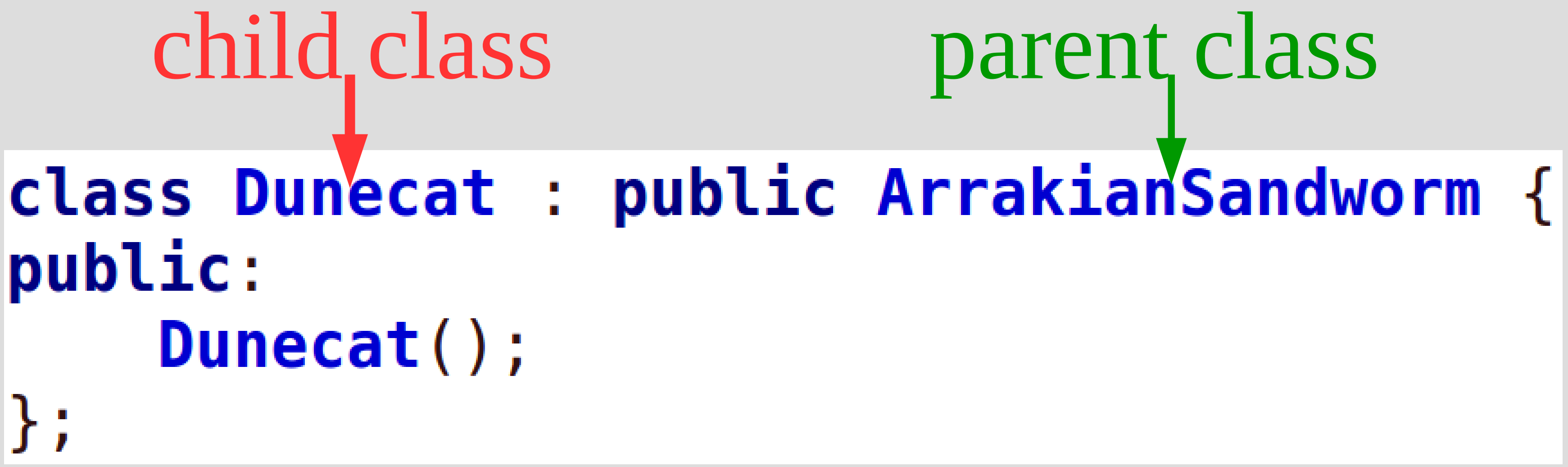
Story time

I ARE DUNECAT
I CONTROLS THE SPICE,
I CONTROLS THE UNIVERSE.
DUNE CAT
haz no fear, fear iz mindkillerz
motifake.com

Derived classes

Let's make this story into code!

To create create a <u>child</u> class from a <u>parent</u> class, use a : in the (child) class declaration



```
class Dunecat : public ArrakianSandworm {
public:
    Dunecat();
};
```

(See: dunecat.cpp)

Derived classes

In a parent/child class relationship, the child gets all variables and functions of the parent

This allows you to build off previous work, even if you need to modify it slightly

This also makes it easier to maintain code, as changing it in the parent class can effect all children (and the children's children)

Derived classes

Typically you use classes when you have multiple objects that are somewhat similar

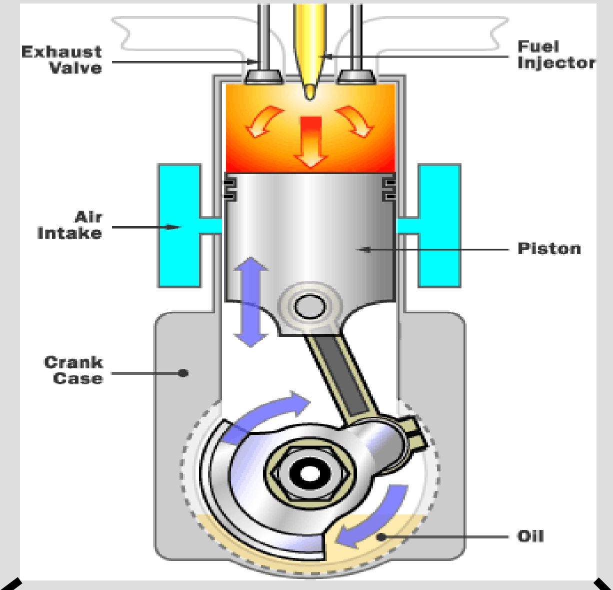
You group the similar parts into a parent class and the different parts into children classes

For examples all chairs have a flat surface to sit on, but they come in different designs (folding types that you are sitting on)
(or rolling types)

Derived classes

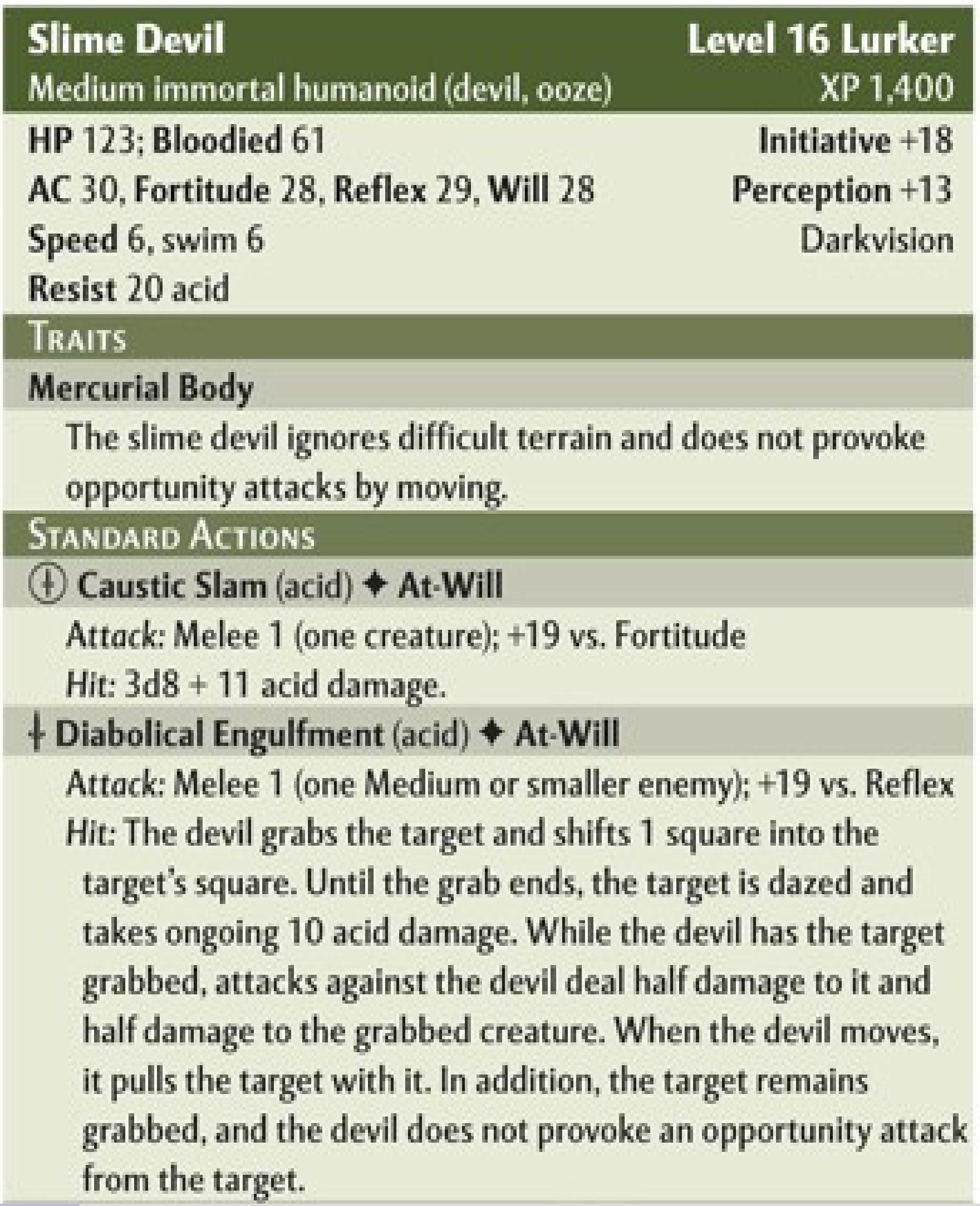
Parent:

(Internal combustion engine)



Children:

AD&D example



Slime Devil — **Level 16 Lurker**
Medium immortal humanoid (devil, ooze) — XP 1,400

HP 123; **Bloodied** 61 — **Initiative** +18
AC 30, **Fortitude** 28, **Reflex** 29, **Will** 28 — **Perception** +13
Speed 6, swim 6 — Darkvision
Resist 20 acid

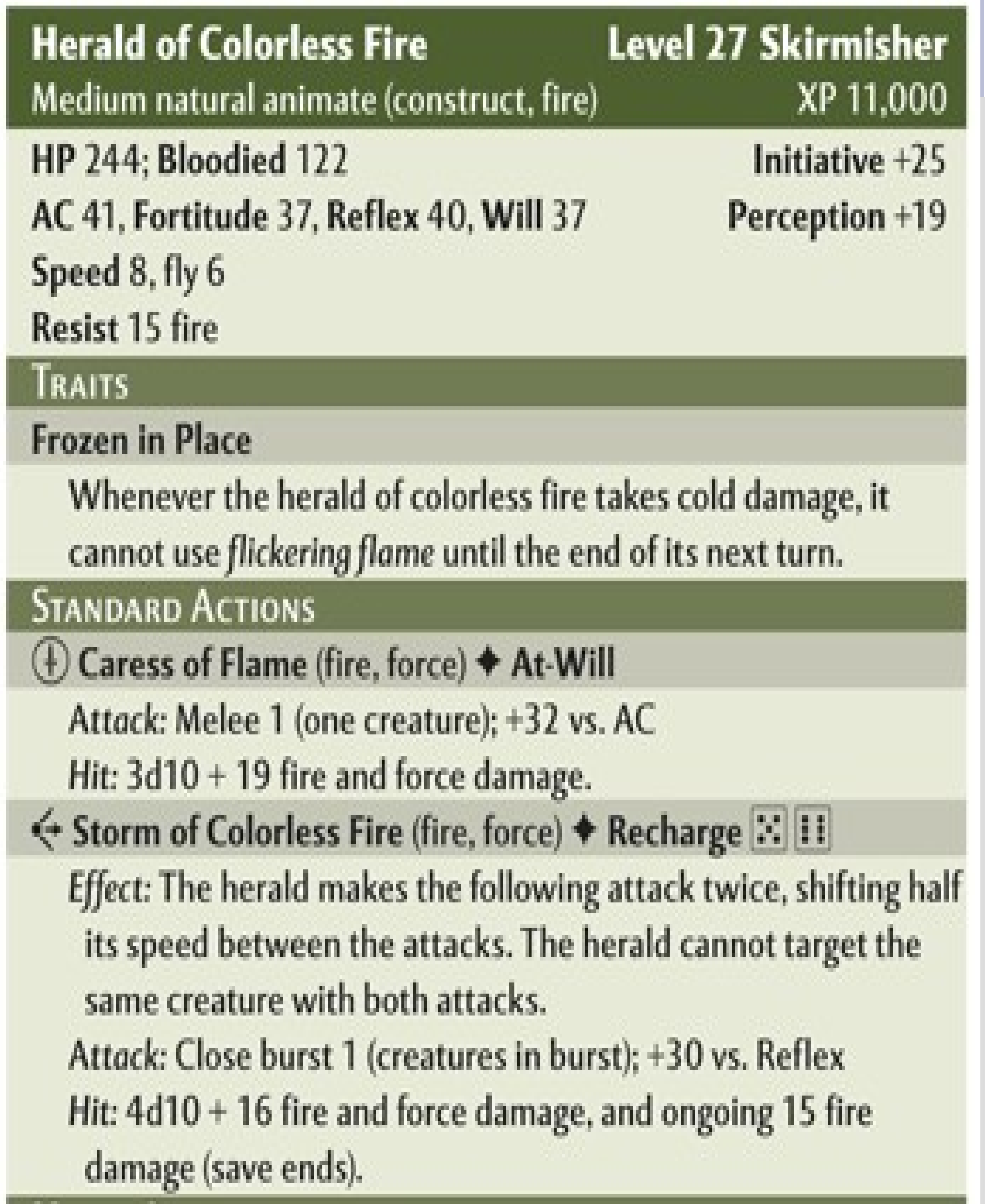
Traits

Mercurial Body
The slime devil ignores difficult terrain and does not provoke opportunity attacks by moving.

Standard Actions

Caustic Slam (acid) ✦ **At-Will**
Attack: Melee 1 (one creature); +19 vs. Fortitude
Hit: 3d8 + 11 acid damage.

Diabolical Engulfment (acid) ✦ **At-Will**
Attack: Melee 1 (one Medium or smaller enemy); +19 vs. Reflex
Hit: The devil grabs the target and shifts 1 square into the target's square. Until the grab ends, the target is dazed and takes ongoing 10 acid damage. While the devil has the target grabbed, attacks against the devil deal half damage to it and half damage to the grabbed creature. When the devil moves, it pulls the target with it. In addition, the target remains grabbed, and the devil does not provoke an opportunity attack from the target.



Herald of Colorless Fire — **Level 27 Skirmisher**
Medium natural animate (construct, fire) — XP 11,000

HP 244; **Bloodied** 122 — **Initiative** +25
AC 41, **Fortitude** 37, **Reflex** 40, **Will** 37 — **Perception** +19
Speed 8, fly 6
Resist 15 fire

Traits

Frozen in Place
Whenever the herald of colorless fire takes cold damage, it cannot use *flickering flame* until the end of its next turn.

Standard Actions

Caress of Flame (fire, force) ✦ **At-Will**
Attack: Melee 1 (one creature); +32 vs. AC
Hit: 3d10 + 19 fire and force damage.

Storm of Colorless Fire (fire, force) ✦ **Recharge** ⚄ ⚅
Effect: The herald makes the following attack twice, shifting half its speed between the attacks. The herald cannot target the same creature with both attacks.
Attack: Close burst 1 (creatures in burst); +30 vs. Reflex
Hit: 4d10 + 16 fire and force damage, and ongoing 15 fire damage (save ends).

Phone

TELEPHONE

SONY
SAMSUNG
htc

SMARTPHONES
Picard uses Android
VERY DEMOTIVATIONAL .com

Finding similarities

Consider these two sports:

If you were going to create a C++ class for these, what data would you store in them?

Finding similarities

Consider two classes you have made already:
Point
Complex

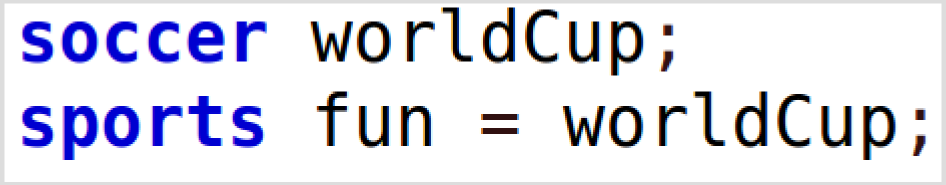
You can have a single parent of both of these that stores the similar parts

This means you only need to type the code once for both classes
(See: complexPoint.cpp)

Types + inheritance

What type of object is “soccer”?

It is (obviously) a “soccer”, but could it also be classified as “sports”?
In fact, yes... both of these are legal:



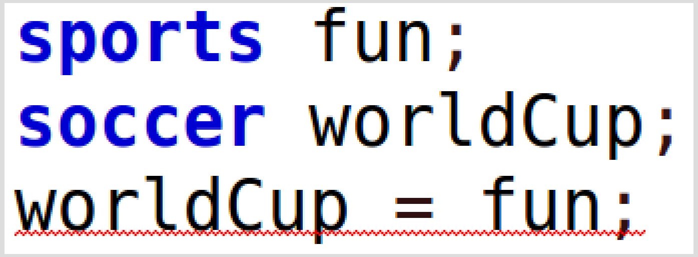
```
soccer worldCup;
sports fun = worldCup;
```

“soccer” have more functionality than “sports” (extra stuff), so they can act as one (just pretend some boxes aren't there)

Types + inheritance

The reverse is not true (as we are using them):

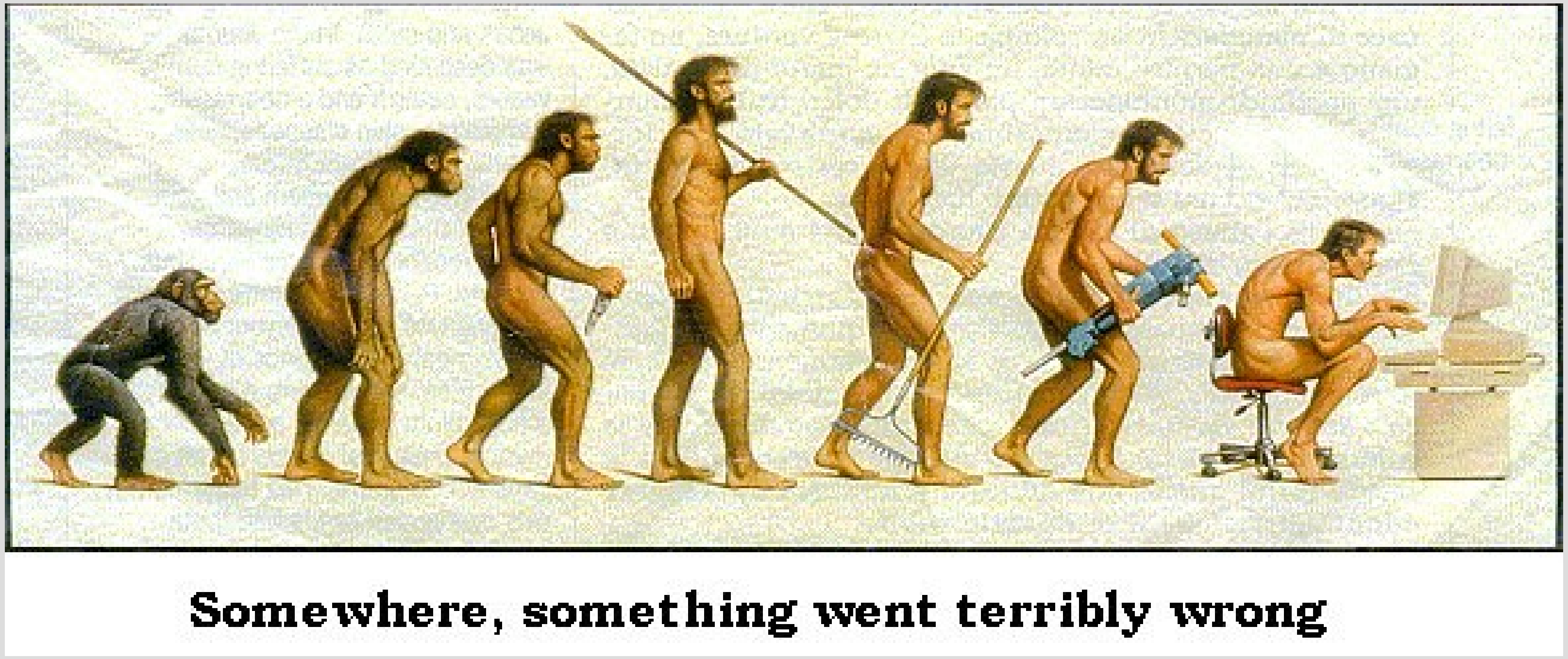
You cannot say:



```
sports fun;
soccer worldCup;
worldCup = fun;
```

As the “worldCup” variable has more info than the “fun” variable (the computer refuses to just guess at the missing functions/data)
(see: convertClassTypes.cpp)

Break



Derived classes

The way data is stored in inherited classes is a bit more complex

Children objects have both a “child” class part and a “parent” class part in their box

While the “parents” only have the “parent” part

(See: childParent.cpp)

Constructors + inheritance

Constructors need to be run every time you make an object...

Now that objects have multiple types what constructors are being run?

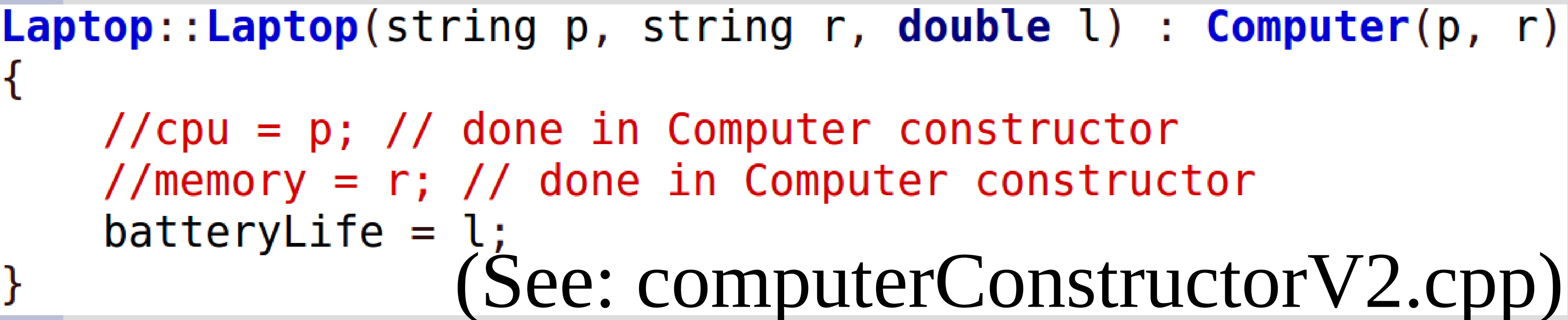
Both actually (again)

(See: computerConstructor.cpp)

Constructors + inheritance

If you do not specify what constructor to use, it will use the default constructor
(or give an error if this does not exist)

You can also specify a non-default constructor by using a “:” after the child's constructor



```
Laptop::Laptop(string p, string r, double l) : Computer(p, r)
{
    //cpu = p; // done in Computer constructor
    //memory = r; // done in Computer constructor
    batteryLife = l;
}
```

(See: computerConstructorV2.cpp)

protected

We know about two scopes for variables:

1. public (anyone, anywhere can use)
2. private (only my class can use)

But there is a third:

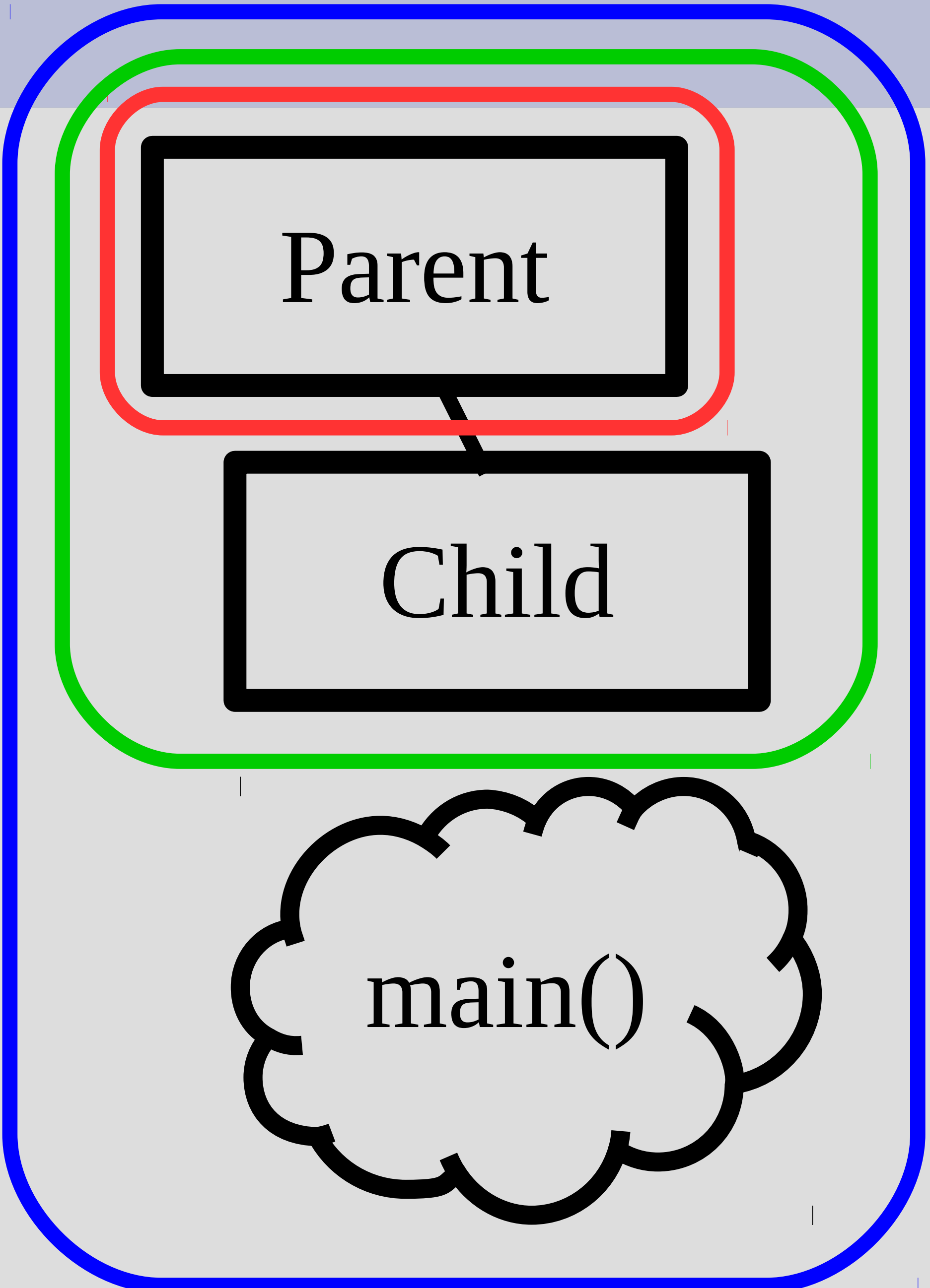
3. protected (me or my children can use)

If you think your children will modify/use a variable, make it protected
(See: classScopes.cpp)

protected

Picture:
Red = private
Green = protected
Blue = public

Variables should be either private or protected



protected

While children technically inherit the private variables/functions, they cannot use them

So effectively, they do not inherit these

It is not considered bad practice to make variables protected (unlike public)

Does access matter?
Yes, because computer viruses

Redefine functions

As children add functionality to a parent class, they may want to redefine some functions

This is different than overloading, where you create multiple versions with the same name

When you redefine, you are basically replacing an old function with a new version

(See: computerRedefine.cpp)

Redefine functions

After you have redefined a function,
the default name will go to the child's version

However, you can still access the parent's version by using “::” (class affiliation)

```
Laptop rightHere = Laptop("2.7 GHz i5", "8 GB DDR3", 3);
rightHere.displaySpecs();
// runs Laptop's version of displaySpecs
rightHere.Computer::displaySpecs();
// runs Computer's version of displaySpecs
```

Not inherited

As we saw before, constructors are not really inherited (though they are called)

overloading operators will also not be inherited (as computer cannot convert parent into child class)

Destructors are also not inherited, but the parent's version of the destructor will always run   (See: childDestructor.cpp)