|  |
| --- |
| Animal  class Animal { public:  Animal(string sound): sound\_(sound) {} string MakeSound() {return sound\_; } **virtual** int GetSpeed() {return 0; } private: std::string sound\_;  } |

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 \_\_\_\_animal\_\_\_\_\_\_\_\_ class is the \_\_\_parent\_\_\_\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | 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; } } | |

* the \_\_\_\_reptile\_\_\_\_\_\_\_\_ class is the \_\_\_\_child\_\_\_\_\_\_

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

* a \_\_\_\_\_reptile\_\_\_\_\_\_\_ is a(n) \_\_\_\_\_animal\_\_\_\_\_\_\_\_\_ What is not inherited?
* Private attributes,

What is inherited?

Public attributes, and protected

|  |
| --- |
| // 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; |

How does privacy interact with inheritance?

It stops the inheritance

What is the output of the above code? “turtle turtle rawr, fuzzy fuzz, moo”

Would the below code work? why/why not? Yes since they are all the same type

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

# Dynamic Dispatch

What is dynamic dispatch? How does it relate to the virtual keyword? That the lowest, most derived, method is called at run time to find the lowest, the act of choosing

// 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? Answer: yes they all inherit from the animal type

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

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

What method(s) are called in the following code? The ones is the most derived class

method(s) called

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

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

hjjjjjjj

method(s) called: the getSpeed method from the specific class

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

The parent animal clas method would have been called

|  |  |
| --- | --- |
| Non static fields | Static fields |

|  |  |  |
| --- | --- | --- |
| Point.h  static int x\_; static int y\_; |  | Point.cpp  int Point::x\_ = ; int Point::y\_ = ; |

Point.h

int x\_; int y\_;

Point instances Point instances

x\_ =

y\_ =

x\_ =

y\_ =

x\_ =

y\_ =

x\_ =

y\_ =

x\_ =

y\_ =

x\_ =

y\_ =

|  |  |
| --- | --- |
| Non static methods | Static methods |

# Point.h Point.h

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

x\_ =

y\_ =

x\_ =

y\_ =

x\_ =

y\_ =

x\_ =

y\_ =