https://youtu.be/C2DNR0Ao0VM

Override

"Overriding" a function occurs when a derived class defines the implementation of a `virtual` function that it inherits from a base class.

It is possible, but not required, to specify a function declaration as `override`.

```
class Shape {
public:
  virtual double Area() const = 0;
  virtual double Perimeter() const = 0;
};

class Circle : public Shape {
public:
  Circle(double radius) : radius_(radius) {}
  double Area() const override { return pow(radius_, 2) * PI; } // specified as an override function
  double Perimeter() const override { return 2 * radius_ * PI; } // specified as an override function

private:
  double radius_;
};
```

This specification tells both the compiler and the human programmer that the purpose of this function is to override a virtual function. The compiler will verify that a function specified as `override` does indeed override some other virtual function, or otherwise the compiler will generate an error.

Specifying a function as `override` is **good practice**, as it empowers the compiler to verify the code, and communicates the intention of the code to future users.

Exercise

In this exercise, you will build two **vehicle motion models**, and override the `Move()` member function.

The first motion model will be `class ParticleModel`. In this model, the state is `x`, `y`, and `theta` (heading). The `Move(double v, double theta)` function for this model includes instantaneous steering:

`theta += phi`

`x += v * cos(theta)`

`y += v * cos(theta)`

The second motion model will be `class BicycleModel`. In this model, the state is `x`, `y`, `theta` (heading), and `L` (the length of the vehicle). The `Move(double v, double theta)` function for this model is affected by the length of the vehicle:

`theta += v / L * tan(phi)`

`x += v * cos(theta)`

`y += v * cos(theta)`

You are encouraged to **read more** about vehicle motion, but for the purposes of practicing function overriding, the precise motion models are not so important. What is important is that the two models, and thus to the two `Move()` functions, are *different*.

Instructions

1. Define `class ParticleModel`, including its state and `Move()` function.
2. Extend `class BicycleModel` from `class ParticleModel`.
3. Override the `Move()` function within `class BicycleModel`.
4. Specify `BicycleModel::Move()` as **`override`**.
5. Pass the tests in `main()` by verifying that the two `Move()` functions override each other in different scenarios.