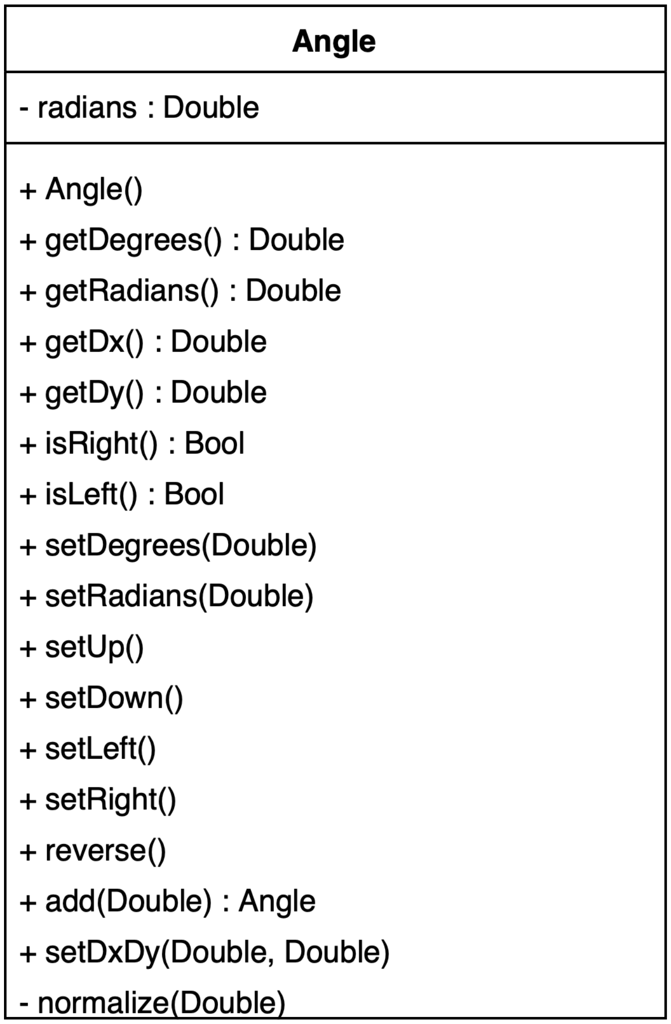
Taden Marston & Mark Van Horn

UML Class Diagrams, Flowcharts, and Pseudocode

Class Diagram: Angle Class



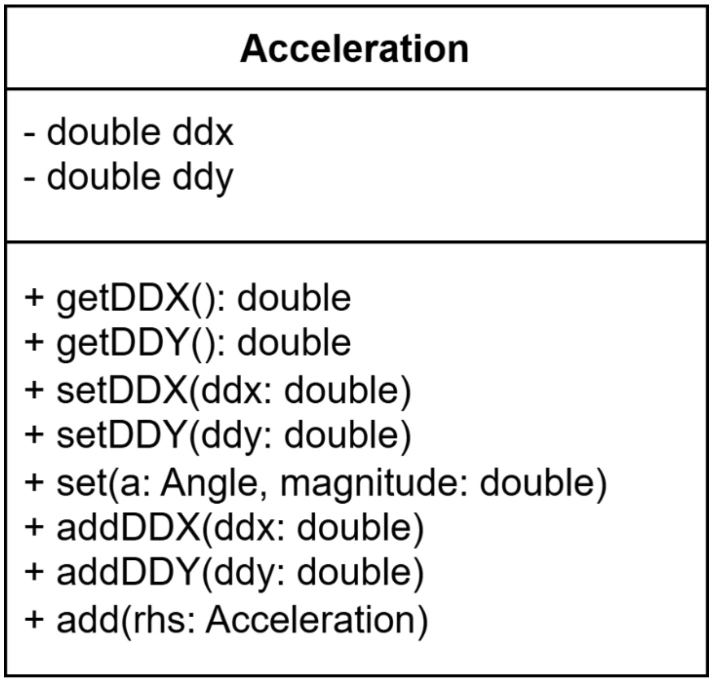
This class diagram has **Complete Fidelity** as it addresses all design concerns, containing methods for all possible use cases.

This class diagram has **Seamless Convenience** as it is perfectly tailored to the needs of the application.

This class diagram has **Porous Abstraction** as there are a few details that would be useful to the user for implementation.

Taden Marston & Mark Van Horn

Class Diagram: Acceleration



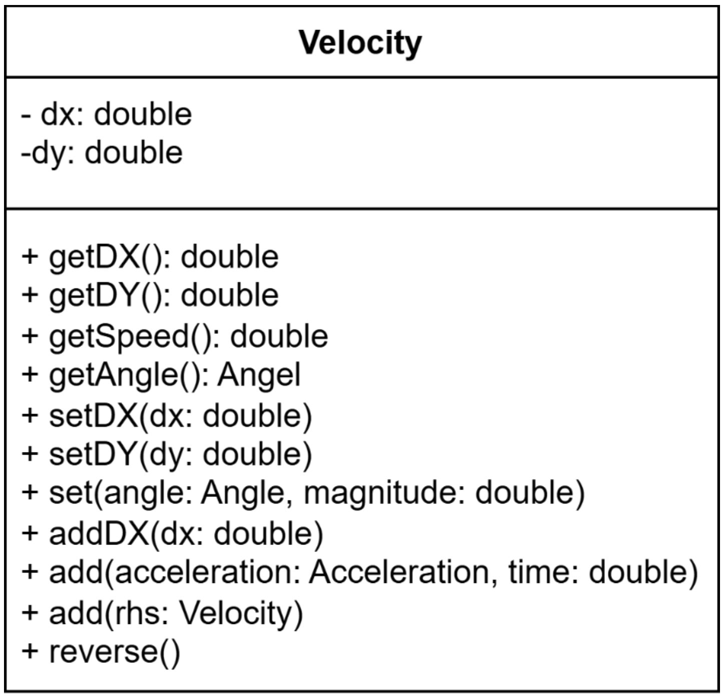
This class diagram has **Complete Fidelity** as it addresses all design concerns, containing methods for all possible use cases.

This class diagram has **Seamless Convenience** as it is perfectly tailored to the needs of the application.

This class diagram has **Porous Abstraction** as there are a few details that would be useful to the user for implementation.

Taden Marston & Mark Van Horn

Class Diagram: Velocity



```
Velocity
- dx: double
-dy: double

+ getDX(): double
+ getDY(): double
+ getSpeed(): double
+ getAngle(): Angel
+ setDX(dx: double)
+ setDY(dy: double)
+ set(angle: Angle, magnitude: double)
+ addDX(dx: double)
+ add(acceleration: Acceleration, time: double)
+ add(rhs: Velocity)
+ reverse()
```

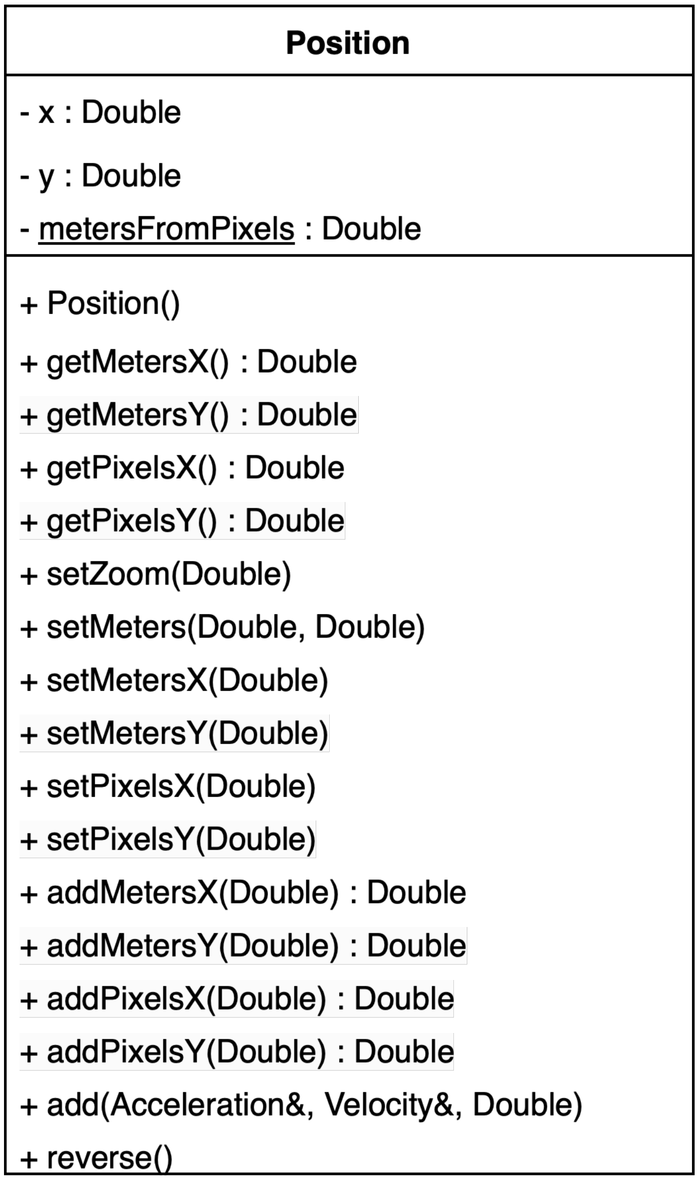
This class diagram has **Complete Fidelity** as it addresses all design concerns, containing methods for all possible use cases.

This class diagram has **Seamless Convenience** as it is perfectly tailored to the needs of the application.

This class diagram has **Porous Abstraction** as there are a few details that would be useful to the user for implementation.

Taden Marston & Mark Van Horn

Class Diagram: Position

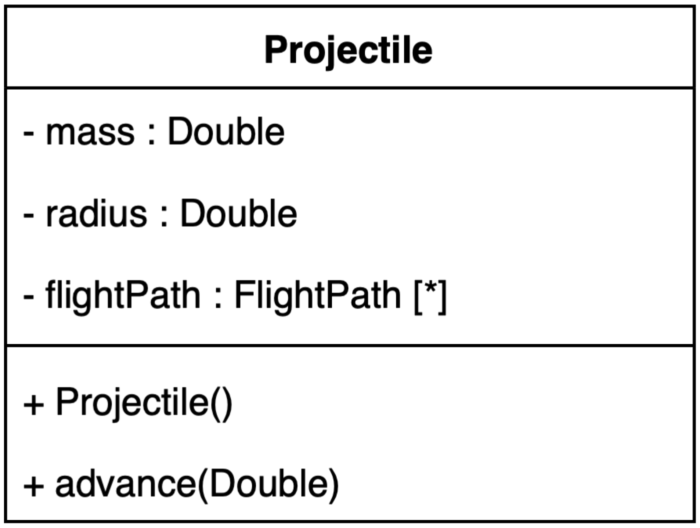


This class has **Complete Fidelity** since the class definition closely matches the intent of representing a position with x and y coordinates and includes methods for manipulating these coordinates.

This class has **Seamless Convenience** as it provides various constructors, getters, and setters. It also includes methods for adding coordinates and interaction with other objects (Acceleration and Velocity), which makes it easier to work with positions in the context of other classes.

This class has **Porous Abstraction**. Internal details such as the coordinates (x, y) and the conversion between meters and pixels are encapsulated within the class, providing a clear interface for the user. However, the class could be further abstracted by defining more complex behaviors and interactions with other classes, such as Acceleration and Velocity, in more detail.

Class Diagram: Projectile



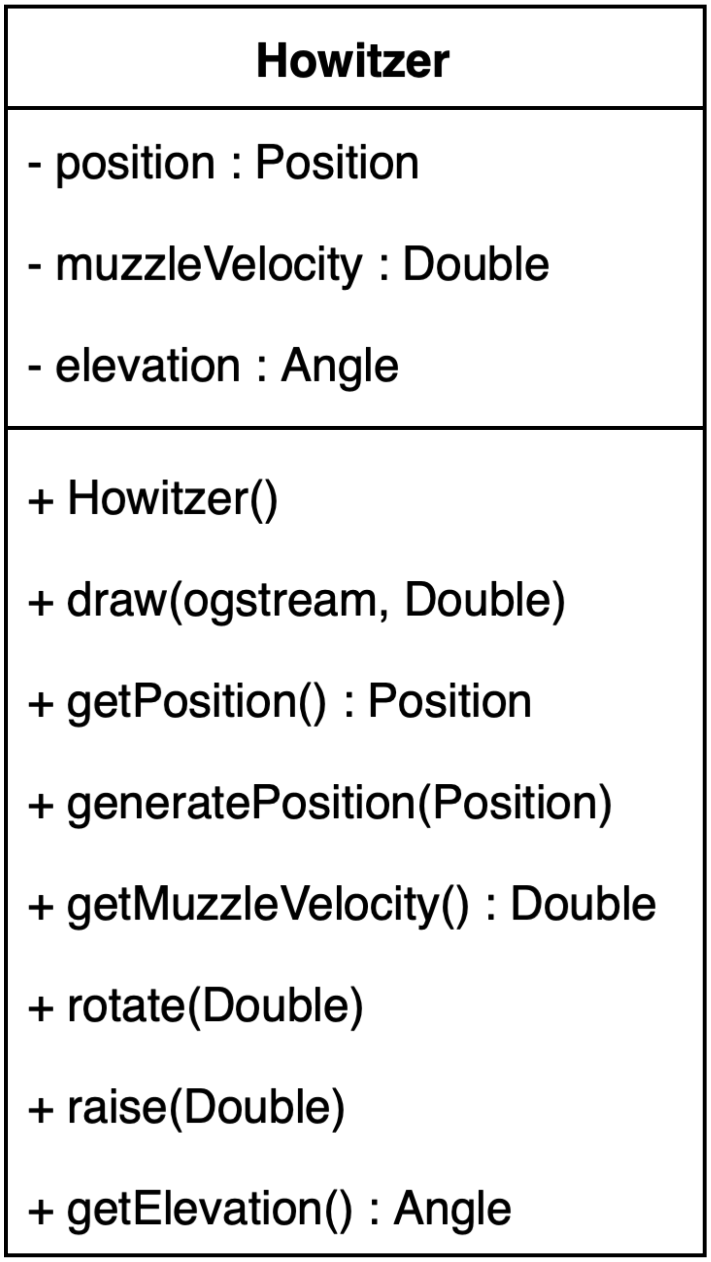
This class has **Complete Fidelity** as it is very simple as addresses the design concern.

This class has **Seamless Convenience** as it is incredibly easy to use and requires no modification.

This class has **Complete Abstraction** as it only addresses the what is listed in the design concern and all other functions are handled elsewhere.

Taden Marston & Mark Van Horn

Class Diagram: Howitzer



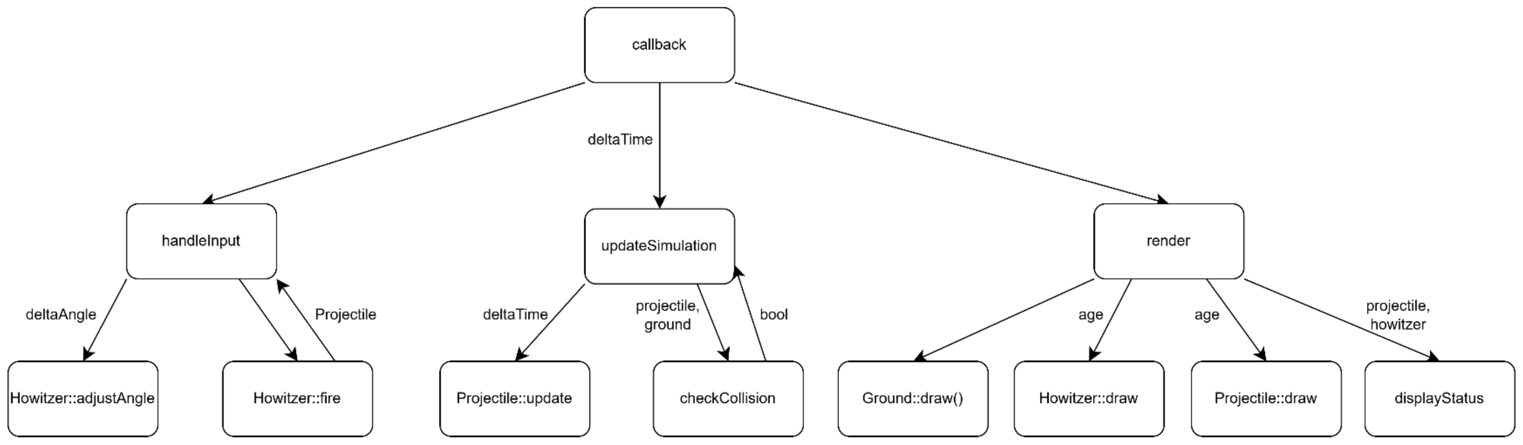
This class has **Complete Fidelity** because it addresses all the design concerns.

This class has **Seamless Convenience** because no modifications are needed to utilize the class.

This class has **Porous Abstraction** because there are a few elements that are useful to know in order to properly use the class.

Taden Marston & Mark Van Horn

Structure Chart: Callback()



Taden Marston & Mark Van Horn

Pseudocode: advance(Double)

```
callback():

    handleInput()

    updateSimulation(deltaTime)

    render()
```

Pseudocode: Projectile update

```
Projectile::update(deltaTime):
    IF projectile is active:
        velocity.dx += acceleration.getDDX() * deltaTime
        velocity.dy += acceleration.getDDY() * deltaTime

        position.x += velocity.getDX() * deltaTime
        position.y += velocity.getDY() * deltaTime

        IF checkCollision(self, ground):
            self.deactivate()
```