Video 17 : Simulating a Fight with Objects

In this tutorial we will have some fun with Object Oriented Programming, while simulating a fight between Thor and Loki. Sample output will look like this :

```
Thor attacks Loki and deals 9 damage
Loki is down to 10 health
Loki attacks Thor and deals 7 damage
Thor is down to 7 health
Thor attacks Loki and deals 19 damage
Loki is down to -9 health
Loki has Died and Thor is Victorious
Game Over
```

We will create classes for both a Warrior and a Battle class. The Warrior class will simulate both the attributes and capabilities of a Warrior. The Battle class will however simulate the actions that occur in a battle such as starting the fight and getting the results.

CODE

```
# We will create classes for both a Warrior and a Battle class
# The Warrior class will simulate both the attributes and capabilities of a Warrior
# The Battle class will however simulate the actions that occur in a battle such as starting the
fight and getting the results

import random
import math

# Warriors will have names, health, and attack and block maximums
# They will have the capabilities to attack and block random amounts
class Warrior:
    def __init__(self, name="warrior", health=0, attk_max=0, block_max=0):
        self.name = name
        self.health = health
        self.attk_max = attk_max
        self.block_max = block_max

    def attack(self):
        # Randomly calculate the attack amount
        # random() returns a value from 0.0 to 1.0
        attk_amt = self.attk_max * (random.random() + .5)
        return attk_amt

    def block(self):
        # Randomly calculate how much of the attack was blocked
        block_amt = self.block_max * (random.random() + .5)
        return block_amt

# The Battle class will have the capability to loop until 1 Warrior dies
# The Warriors will each get a turn to attack each turn
class Battle:
    def start_fight(self, warrior1, warrior2):
        # Continue looping until a Warrior dies switching back and
```

```
        # forth as the Warriors attack each other
        while True:
            if self.get_attack_result(warrior1, warrior2) == "Game Over":
                print("Game Over")
                break

            if self.get_attack_result(warrior2, warrior1) == "Game Over":
                print("Game Over")
                break

    # A function will receive each Warrior that will attack the other
    # Have the attack and block amounts be integers to make the results clean
    # Output the results of the fight as it goes
    # If a Warrior dies return that result to end the looping in the
    # above function

    # Make this method static because we don't need to use self
    @staticmethod
    def get_attack_result(warriorA, warriorB):
        warrior_a_attk_amt = warriorA.attack()
        warrior_b_block_amt = warriorB.block()
        damage_2_warrior_b = math.ceil(warrior_a_attk_amt - warrior_b_block_amt)
        warriorB.health = warriorB.health - damage_2_warrior_b

        print("{} attacks {} and deals {} damage".format(warriorA.name, warriorB.name,
damage_2_warrior_b))
        print("{} is down to {} health".format(warriorB.name,
warriorB.health))

        if warriorB.health <= 0:
            print("{} has Died and {} is Victorious".format(warriorB.name, warriorA.name))
            return "Game Over"
        else:
            return "Fight Again"

def main():
    # Create 2 Warriors
    thor = Warrior("Thor", 50, 20, 10)
    loki = Warrior("Loki", 50, 20, 10)
    # Create Battle object
    battle = Battle()
    # Initiate Battle
    battle.start_fight(thor, loki)

main()
```

I hope you enjoyed that tutorial. It was fun to make! In the next video I'll cover inheritance, operator overloading and polymorphism.