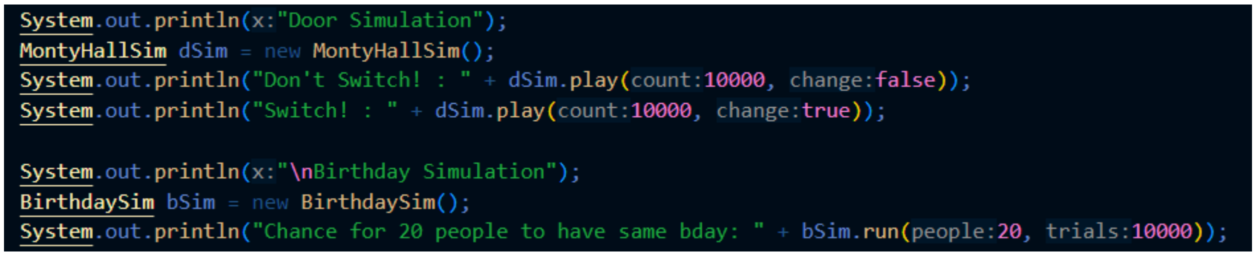
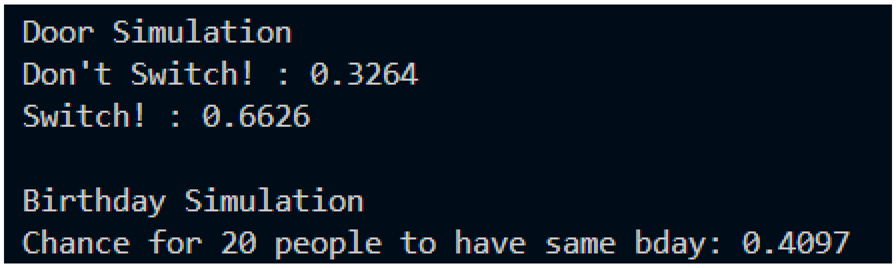
MONTE CARLO SIMULATIONS

Since both the birthday program and the Monty Hall program were basically the same thing, I thought I'd just write one tester for both of them. It goes over "4 lines", but it's for 2 programs.

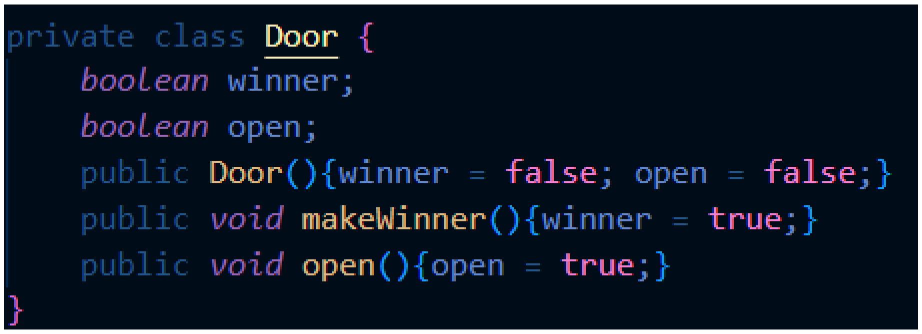


```java
System.out.println(x:"Door Simulation");
MontyHallSim dSim = new MontyHallSim();
System.out.println("Don't Switch! : " + dSim.play(count:10000, change:false));
System.out.println("Switch! : " + dSim.play(count:10000, change:true));

System.out.println(x:"\nBirthday Simulation");
BirthdaySim bSim = new BirthdaySim();
System.out.println("Chance for 20 people to have same bday: " + bSim.run(people:20, trials:10000));
```



```
Door Simulation
Don't Switch! : 0.3264
Switch! : 0.6626

Birthday Simulation
Chance for 20 people to have same bday: 0.4097
```

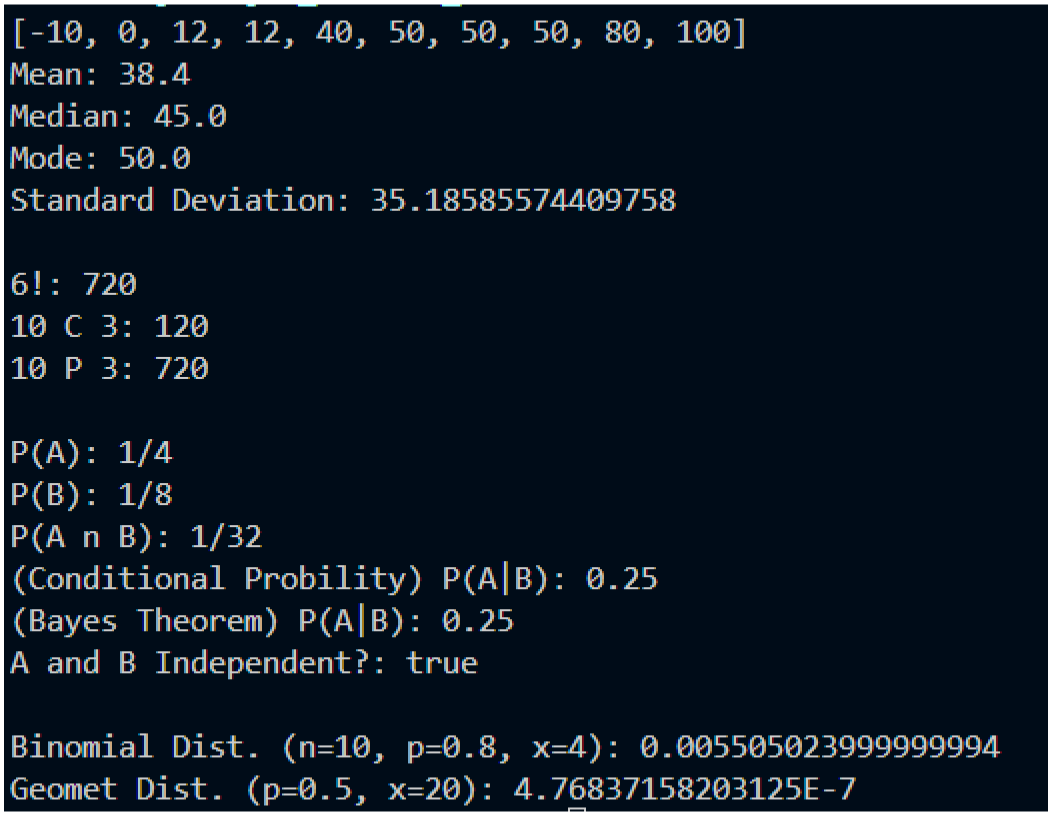
The Monty Hall simulation felt pretty jank. It did the job fine, though, which was all that mattered to me. The jank in question felt worst where I made a Door inner class, but hey, if I wanted to, I could easily edit the simulation to accept a parameter for a different number of doors.



```java
private class Door {
    boolean winner;
    boolean open;
    public Door(){winner = false; open = false;}
    public void makeWinner(){winner = true;}
    public void open(){open = true;}
}
```

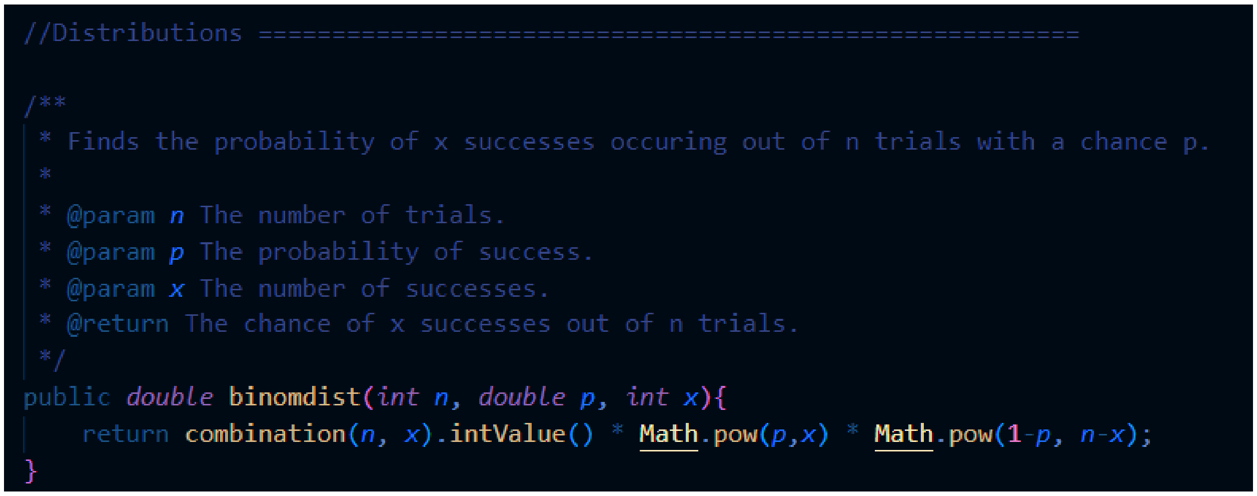
STATS LIBRARY + SET OPERATIONS

StatsLibrary went smoothly across everything. Everything has Java Documentation as well. I did have to integrate BigInteger into factorial() and the combinatorial methods.



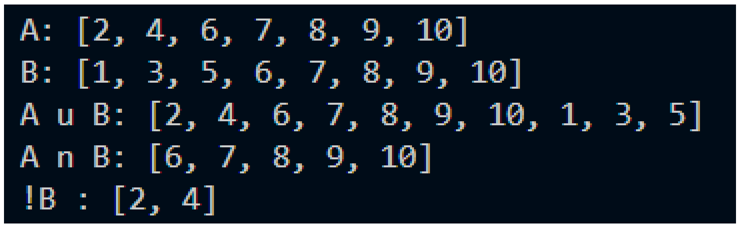
```
[-10, 0, 12, 12, 40, 50, 50, 50, 80, 100]
Mean: 38.4
Median: 45.0
Mode: 50.0
Standard Deviation: 35.18585574409758

6!: 720
10 C 3: 120
10 P 3: 720

P(A): 1/4
P(B): 1/8
P(A n B): 1/32
(Conditional Probility) P(A|B): 0.25
(Bayes Theorem) P(A|B): 0.25
A and B Independent?: true

Binomial Dist. (n=10, p=0.8, x=4): 0.005505023999999994
Geomet Dist. (p=0.5, x=20): 4.76837158203125E-7
```

Code Snippet of binomial distribution



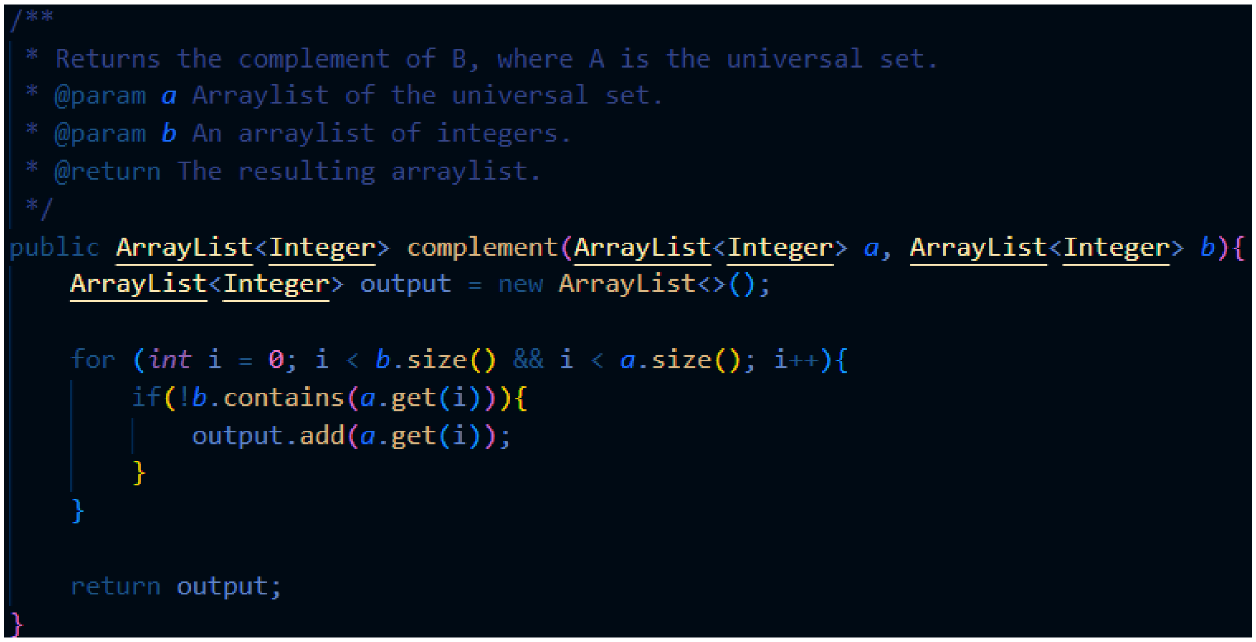
```java
//Distributions ==========================================================

/**
 * Finds the probability of x successes occuring out of n trials with a chance p.
 *
 * @param n The number of trials.
 * @param p The probability of success.
 * @param x The number of successes.
 * @return The chance of x successes out of n trials.
 */
public double binomdist(int n, double p, int x){
    return combination(n, x).intValue() * Math.pow(p,x) * Math.pow(1-p, n-x);
}
```

SetOperations was also relatively simple thanks to ArrayList’s .contains() method.



```
A: [2, 4, 6, 7, 8, 9, 10]
B: [1, 3, 5, 6, 7, 8, 9, 10]
A u B: [2, 4, 6, 7, 8, 9, 10, 1, 3, 5]
A n B: [6, 7, 8, 9, 10]
!B : [2, 4]
```

Code snippet:

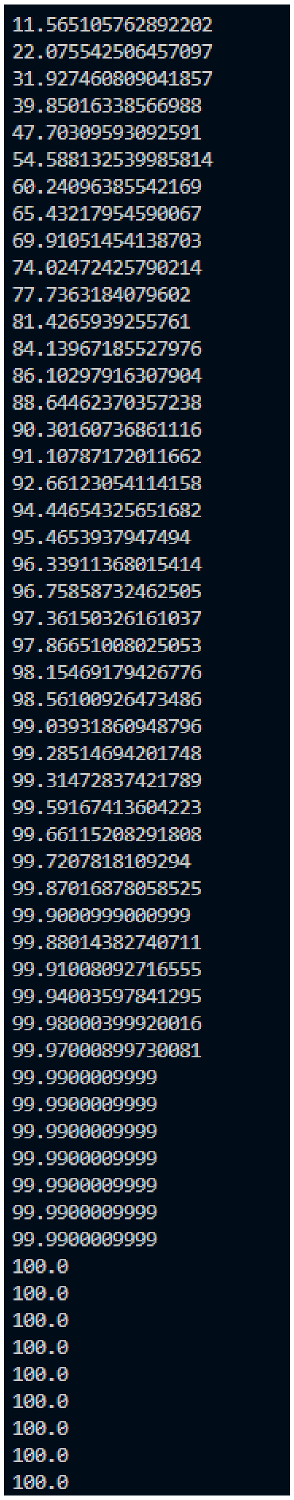


```java
/**
 * Returns the complement of B, where A is the universal set.
 * @param a Arraylist of the universal set.
 * @param b An arraylist of integers.
 * @return The resulting arraylist.
 */
public ArrayList<Integer> complement(ArrayList<Integer> a, ArrayList<Integer> b){
    ArrayList<Integer> output = new ArrayList<>();

    for (int i = 0; i < b.size() && i < a.size(); i++){
        if(!b.contains(a.get(i))){
            output.add(a.get(i));
        }
    }

    return output;
}
```

POKEMON MONTE CARLO

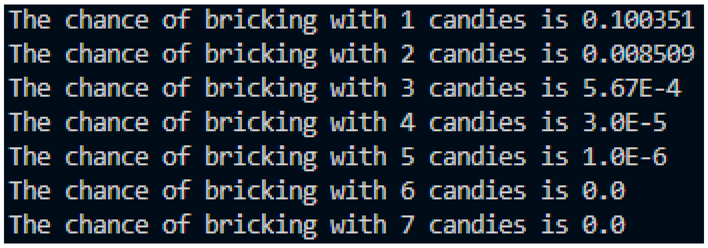
Coding was straight forward. I made a Player class and ran trials off of it drawing a valid hand, or bricking a deck. Java documentation is available.

Valid Hand results:



```
11.565105762892202
22.075542506457097
31.927460809041857
39.85016338566988
47.70309593092591
54.588132539985814
60.24096385542169
65.43217954590067
69.91051454138703
74.02472425790214
77.7363184079602
81.4265939255761
84.13967185527976
86.10297916307904
88.64462370357238
90.30160736861116
91.10787172011662
92.66123054114158
94.44654325651682
95.4653937947494
96.33911368015414
96.75858732462505
97.36150326161037
97.86651008025053
98.15469179426776
98.56100926473486
99.03931860948796
99.28514694201748
99.31472837421789
99.59167413604223
99.66115208291808
99.7207818109294
99.87016878058525
99.9000999000999
99.88014382740711
99.91008092716555
99.94003597841295
99.98000399920016
99.97000899730081
99.9900009999
99.9900009999
99.9900009999
99.9900009999
99.9900009999
99.9900009999
99.9900009999
100.0
100.0
100.0
100.0
100.0
100.0
100.0
100.0
100.0
```

Rare candy sim was interesting. The chance got extremely small, so instead of the typical 10,000s to run, I ran 1,000,000. I stopped testing at 7 candies since there's only 6 prize cards and there's no point testing further. Past 6 candies is 0%.

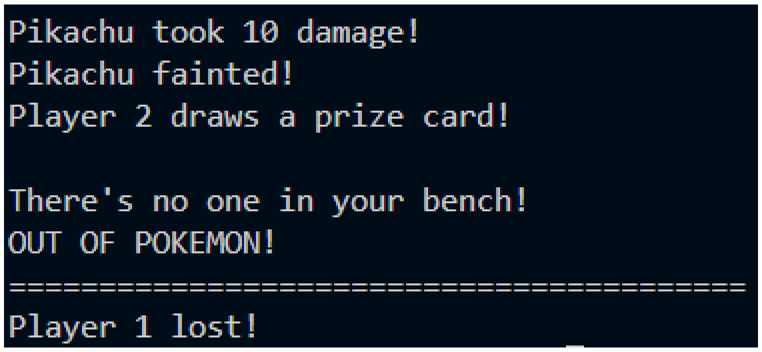


```
The chance of bricking with 1 candies is 0.100351
The chance of bricking with 2 candies is 0.008509
The chance of bricking with 3 candies is 5.67E-4
The chance of bricking with 4 candies is 3.0E-5
The chance of bricking with 5 candies is 1.0E-6
The chance of bricking with 6 candies is 0.0
The chance of bricking with 7 candies is 0.0
```

POKEMON GAME

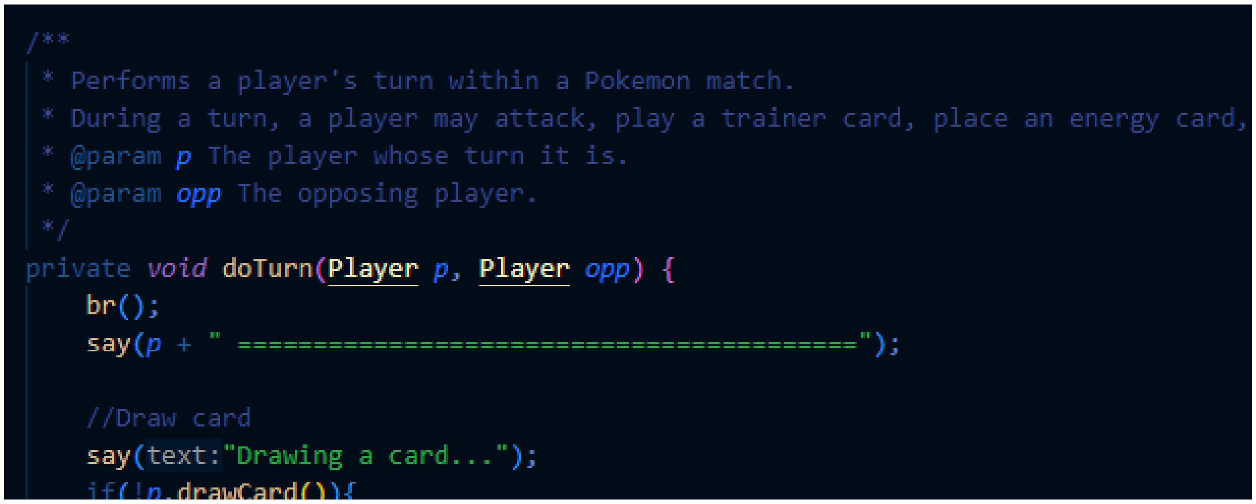
The game was done in a way where both players had to be controlled manually with Scanner.

Results: I was player 2, pinky promise.

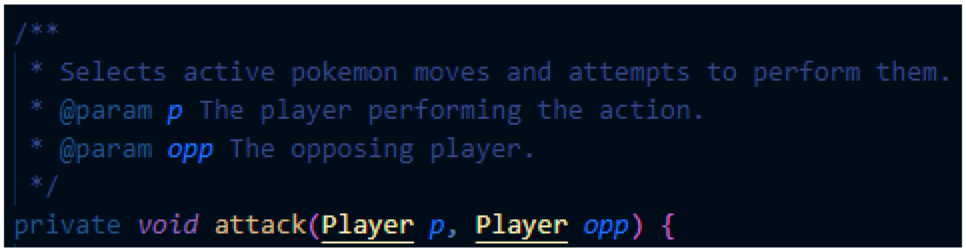


```
Pikachu took 10 damage!
Pikachu fainted!
Player 2 draws a prize card!

There's no one in your bench!
OUT OF POKEMON!
==========================================
Player 1 lost!
```

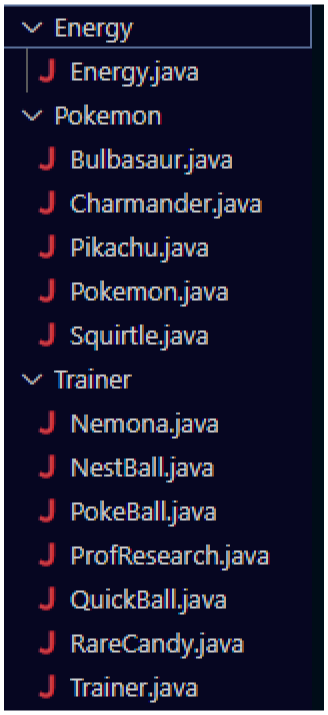
I used documentation comments for practically every class and method.
Code snippet:



```java
/**
 * Performs a player's turn within a Pokemon match.
 * During a turn, a player may attack, play a trainer card, place an energy card,
 * @param p The player whose turn it is.
 * @param opp The opposing player.
 */
private void doTurn(Player p, Player opp) {
    br();
    say(p + " ==========================================");

    //Draw card
    say(text:"Drawing a card...");
    if(!p.drawCard()){
```

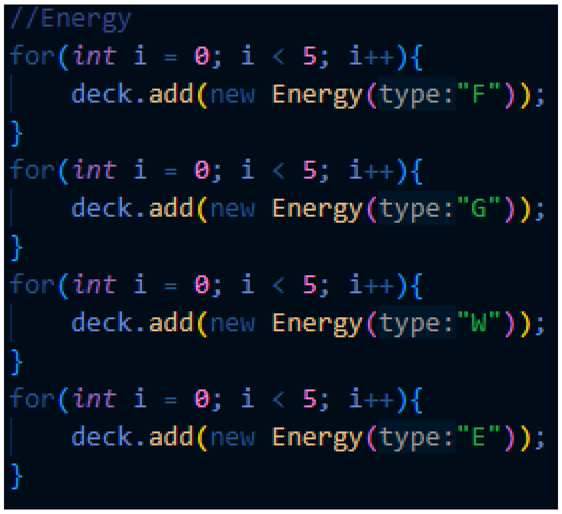


```java
/**
 * Selects active pokemon moves and attempts to perform them.
 * @param p The player performing the action.
 * @param opp The opposing player.
 */
private void attack(Player p, Player opp) {
```



```
Energy
    Energy.java
Pokemon
    Bulbasaur.java
    Charmander.java
    Pikachu.java
    Pokemon.java
    Squirtle.java
Trainer
    Nemona.java
    NestBall.java
    PokeBall.java
    ProfResearch.java
    QuickBall.java
    RareCandy.java
    Trainer.java
```

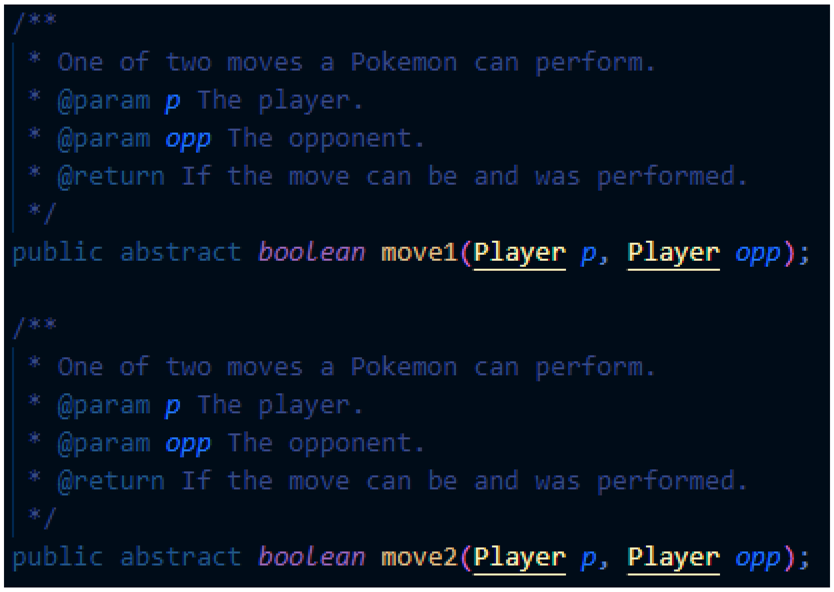
Here are all the cards I have implemented. Rather than make 15 classes of energy cards, instead I opted to initialize an Energy object with a parameter of what type it is.



```java
//Energy
for(int i = 0; i < 5; i++){
    deck.add(new Energy(type:"F"));
}
for(int i = 0; i < 5; i++){
    deck.add(new Energy(type:"G"));
}
for(int i = 0; i < 5; i++){
    deck.add(new Energy(type:"W"));
}
for(int i = 0; i < 5; i++){
    deck.add(new Energy(type:"E"));
}
```

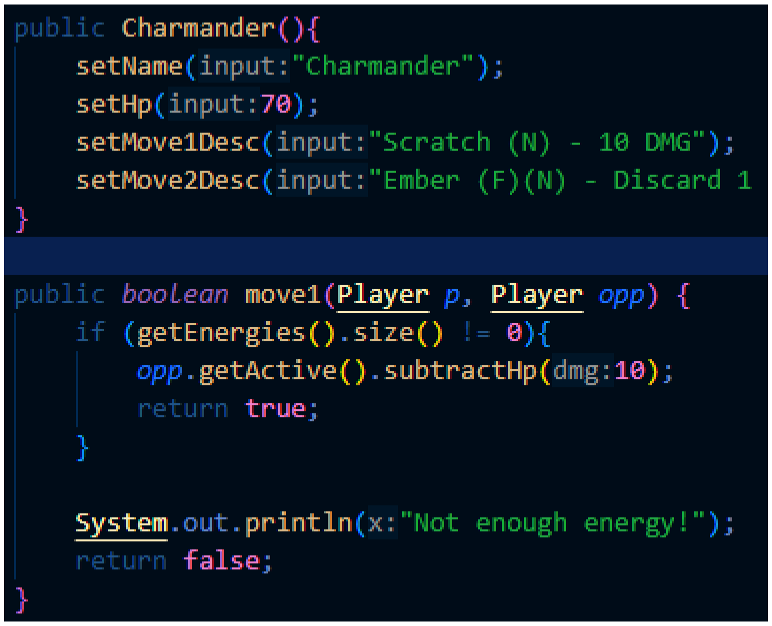
The Pokemon class feels a little jank, though. There's no implementation for weakness, resistances, abilities, nor evolutions, but the general consensus was that those were optional features. For each Pokemon class, I hardcoded the moves--every pokemon had a method for move1 and move2. I realize that that was kind of a bad move and I should've made a Move class and Pokemon should've held Move objects.
That seems to be what happens in the actual video game, too--moves are separate objects from the pokemon since pokemon can learn the same moves and forget them.

Code snippet of Pokemon (some abstraction going on)



```
/**
 * One of two moves a Pokemon can perform.
 * @param p The player.
 * @param opp The opponent.
 * @return If the move can be and was performed.
 */
public abstract boolean move1(Player p, Player opp);

/**
 * One of two moves a Pokemon can perform.
 * @param p The player.
 * @param opp The opponent.
 * @return If the move can be and was performed.
 */
public abstract boolean move2(Player p, Player opp);
```

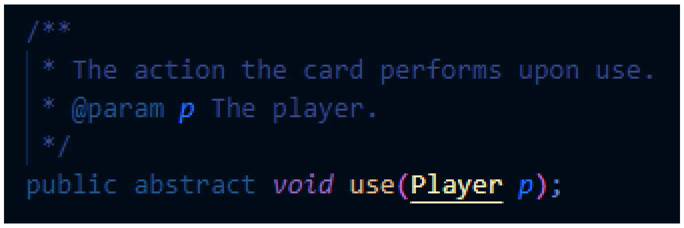
Code snippet of Charmander();



```
public Charmander(){
    setName(input:"Charmander");
    setHp(input:70);
    setMove1Desc(input:"Scratch (N) - 10 DMG");
    setMove2Desc(input:"Ember (F)(N) - Discard 1
}

public boolean move1(Player p, Player opp) {
    if (getEnergies().size() != 0){
        opp.getActive().subtractHp(dmg:10);
        return true;
    }

    System.out.println(x:"Not enough energy!");
    return false;
}
```

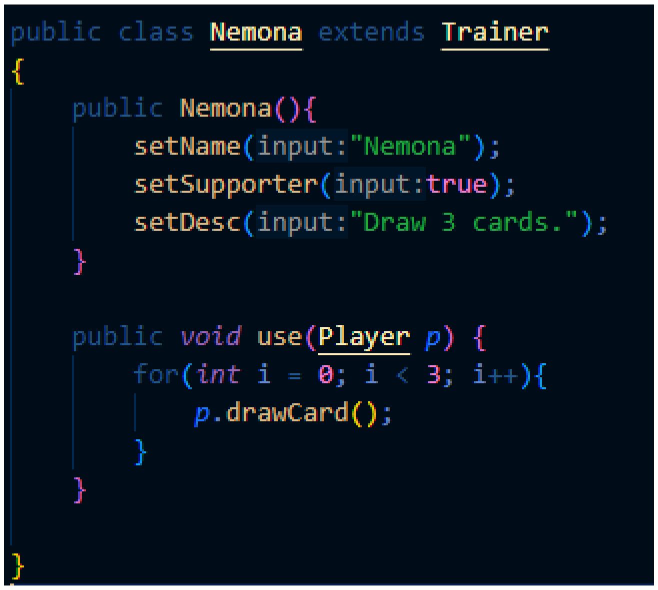
The Trainer class works fine, though. I’m able to track if it’s a supporter or not, too, so only 1 supporter can be played per turn.

Code snippet of Trainer()



```java
/**
 * The action the card performs upon use.
 * @param p The player.
 */
public abstract void use(Player p);
```

Code snippet of Nemona(), probably the most boring Trainer card



```java
public class Nemona extends Trainer
{
    public Nemona(){
        setName(input:"Nemona");
        setSupporter(input:true);
        setDesc(input:"Draw 3 cards.");
    }

    public void use(Player p) {
        for(int i = 0; i < 3; i++){
            p.drawCard();
        }
    }


}
```