PROJECT #2
Michael Day

For the second Intro to AI project, we were tasked with creating a genetic algorithm in the hopes to find the best solution to the knapsack problem. This algorithm is meant to begin with an initial population of individuals, containing a "chromosome" of randomly selected items, that would go through multiple generations of evolution creating better individuals with the highest and most efficient utility.

In my implementation of this program, this evolution was facilitated by three class objects. The Gene class is the smallest component, which represents each item that may be selected by defining their utility and weight. This Gene class is utilized within an array list of genes called the "gene pool". The Individual class contains a "chromosome", an array list of boolean values that correspond to a "gene pool" determining which items have or have not been selected for the solution. The individuals contain a fitness score that represents its total utility count. These individuals help make up the Population class, which contains an array of individuals containing the individual solutions to the problem. This population class also contains the gene pool (a list of all the possible items that may make up an individual's chromosome), variables relating to the overall fitness of the population, and an array list containing the average fitness of each generation within the population.

The way the program works is by generating an initial population (the size defined as a constant in the population class), evaluating the fitness of each individual within the population, creating the next generation through tournament selection, crossover, and mutation, finding the average fitness of this next generation, and repeating these steps until the growth of the average fitness of the population has plateaued. When running the program, the population seems to typically go through about 150-200 generations before plateauing, with an initial population of

5000. This seemingly contrasts the professor's findings, where he mentioned that his algorithm typically plateaued around 2500 generations.