#Simple Cache

#Introduction
The purposed of this project is to give the user a good idea of how caches work in different settings. While the code likely won't give a full picture of how caching really works in a computer system, it should give a simple, limited, yet reasonable display of how caching works under different mapping techniques and provide some stats at the end of the program. By doing so, hopefully the user can get some insight into caching and how caching can be very helpful in mutliple applications.

#Table of Contents

#Requirements
Any decent java compiler should do.

#How to run and what to expect
- Step 1: After compiling and running the program, there will be 4 options that correspond to how the cache will be mapped. Entering "0" will go to direct mapping, "1" goes to 2-set associative mapping, "2" goes to 4-set associative mapping, "3" goes to fully associative mapping.
- Step 2: From here, the user and enter any integer value within 0 to 15. User can repeatedly enter any of these values over and over and notice that the program notifies whether there has been a "hit" or a "miss" and what is in the cache after entering an integer value.
- Step 3: When user is done, simply enter an integer value outside of numbers 0-15. For example you can enter "-1". By doing so, the program will stop repeating step 2 and move on to step 4
- Step 4: Program will go on to display some stats and the final content of the caching. The following stats are "total attempts", "hit count", "miss count", and "miss rate". How the content of the caching is displayed will depend on what mapping technique is selected.

#Notes, Tips, Extra Info, Final thoughts
- Try not to enter any non integer values or strings. The program will basically crash and force the user to recompile and start over.
- The actual addresses should be 32 bits long and caching is much more expansive and advanced, but I've shortened them to 4 bits and the corresponding 16 numbers (0-15) to keep it simple. Direct mapping always has 4 cache slots. 2-set associative always has 4 sets with 2 slots per set, 4-set associative always has 4 sets with 4 slots each, full associative has 4 slots.
- I'm sure there are countless ways to improve the code so that makes more sense, is higher quality, and even add more capabilities that actual caching should have, but this is what I got so...yeah...
- This project is a very simple display of caching compared to real world examples, I can only imagine what caching in a real computer system would be like.
- Not gonna lie I kinda feel proud of this one since I've pretty much programmed this from scratch with a little help getting some ideas and inspirations to point me in the right direction.

#Credits
- Brandon Nguyen (original creator)