In terms of organization, the two seesaw riders, Fred and Wilma, are 2 separate classes that inherit from a SeeSawRider base class, since both Fred and Wilma use the same logic to push up when it's their turn, with the only real difference being in the force they push up with. The program then uses 2 seperate threads: 1 for Fred and 1 for Wilma, which are created and launched by the SeeSawSimulator class. In order to insure that Fred was not pushing the see saw up while Wilma was, I used 2 binary semaphores: 1 for Fred and 1 for Wilma. I assumed that Fred would go first, so I set his semaphore to 1 in the Fred class constructor, and Wilma's to 0 in the Wilma constructor. Inside the pushUp() method (located at line 95 in the SeeSawRider class), both Fred and Wilma call mySemaphore.aqquire() (line 101 in the SeeSawRider class) before attempting to push up. When Fred is done pushing up (which means that Wilma is within 1 ft of the ground), Fred signals over to Wilma's Semaphore by doing otherRider.getMySemaphore().release() (see line 138 in SeeSawRider), which allows Wilma to push up. Since I call mySemaphore.aqquire() before allowing Fred to push up, then Fred's semaphore value is 0 by the time he signals over to Wilma's semaphore, effectively preventing Fred from pushing up until Wilma is done pushing up and signals back to Fred's semaphore. This is how the two Binary semaphores were used to provide the needed synchronization; the picture I drew below helps to illustrate this usage of semaphores, which is very similar to the diagram given in the lecture slides and mentioned in the textbook, with the main differences being in the names of the methods acquire(), which is like wait(), release(), which is like signal(), and the method Fred and Welma execute once they get permission to do so: pushUp().