<u>Part I</u>

- Create a class called **Animal** that implements the **Runnable** interface.
- In the main method create 2 instances of the Animal class, one called **hare** and one called **tortoise**. Make them "user" threads, as opposed to daemon threads.
- Some detail about the Animal class: it has instance variables **name**, **position**, **speed**, and **restMax**. It has a static boolean **winner** which is initialized to false. **position** represents the position in the race for this Animal object. **restMax** represents how long the Animal rests between each time it runs.
  - The hare rests longer than the tortoise, but the hare has a higher speed.
- Let's make up some values to make this simulation more concrete.
  - The race is 120 yards. The initial position is 0. Suppose the speed of the hare is 9, and its maxRest is 220. Suppose the speed of the tortoise is 5, and its maxRest is 165.
- In the main method start both the hare and the tortoise and see which one wins the races.
  - Run a few races, and adjust the values so that the hare wins sometimes, and the tortoise wins sometimes.
- Here is the behavior of the run method in the Animal class.
  - Loop until the position is >= 120 or there is a winner. Each time through the loop, sleep() some random number of milliseconds. This random number will be between 0 and **maxRest**. Advance the position of the Animal by its **speed**.
  - Print who is running, what their **position** is, each time through the loop.

    - When someone wins, set the static variable **winner** to true, and both threads will finish their run method, and thus stop.
    - The winner is announced from inside the run method.

Part II

- The objective here is to demonstrate the behavior of threads that share data, and use synchronized methods. You do NOT use wait / notify/ notifyAll in this exercise.
- When the above race working, add to it in the following way:
- Create a class called Food. It is not a Thread, and does not run. It's just a class that represents some data that will be shared by multiple threads.
- Simulating an animal eating, this simply means that the thread will sleep for some length of time. This is analogous to the "resting" that the tortoise and hare did in Part I.
- There is one instance of the Food class that is shared by both of the animals. Pass it to the constructor of the Animal class for both the tortoise and the hare.
- There is a method in the Food class called eat(). This method is synchronized, i.e., only one Animal can be eating at a time.
- The hare eats the food (the thread will sleep) for a longer time than the tortoise, thus giving an advantage to the tortoise.
- But, the tortoise must wait until the hare is done eating until it can eat, so the advantage is reduced.
- Print out the message inside the eat method when the animal begins to eat, and when it is done eating. Indicate which animal it is that starts to eat.
- Try making the eat method not synchronized, and observe the different behavior if the eat method allows the hare to begin eating before the tortoise is done eating.

- Note that this program may require exception handling. Make sure that all exceptions are handled according to standard programming practices.