import java.util.Random; //Imports the Random type  
//Contender class, is going to implement the Runnable interface. Creates the contenders that will  
//fight against each other  
public class Contender implements Runnable {  
 private String name; //name of the contender  
 private int hotDogs ; // Represents how many hot dogs are eaten at a time  
 private int maxRest; // Maximum rest time in milliseconds  
 private int totalHotDogsEaten; // Total hot dogs eaten  
 private Water water; // Shared water object  
 private static volatile boolean *winner* = false;  
  
 //Constructor  
 public Contender(String name, int hotDogs, int maxRest, Water water) {  
 this.name = name;  
 this.hotDogs = hotDogs;  
 this.totalHotDogsEaten = 0;  
 this.maxRest = maxRest;  
 this.water = water;  
 }  
  
 //Run method  
 @Override  
 public void run() {  
 Random random = new Random(); //Creates a new random number generator  
  
 while (!*winner*) { //while loop that iterates until winner = true  
 try { // try block starts  
 // Rest for a random time within the maxRest limit  
 int restTime = random.nextInt(maxRest);  
 System.*out*.println(name + " is resting for " + restTime + "ms.");  
 Thread.*sleep*(restTime);  
  
 // Eat hot dogs  
 totalHotDogsEaten += hotDogs;  
 System.*out*.println(name + " ate " + hotDogs +" hot dogs. Total: " + totalHotDogsEaten);  
  
 // Check if the contender has won (if they have eaten 200 or more hot dogs)  
 if (totalHotDogsEaten >= 200) {  
 *winner* = true;  
 System.*out*.println(name + " has won the battle!");  
 break;  
 }  
  
 // Take a water break  
 water.drink(name);  
  
 }//try block ends  
 catch (InterruptedException e) { //catch block starts  
 e.printStackTrace();  
 }// catch block ends  
 }  
 }  
}

// Class Water, represents some data that will be shared by multiple threads, and it simulates  
// the action of taking a water break.  
// When the contender is "drinking", the thread will be asleep.  
public class Water {  
 // Synchronized drink method to ensure only one contender drinks at a time  
 public synchronized void drink(String contenderName) {  
 try { //try block starts  
 System.*out*.println(contenderName + " is drinking water.");  
 // Simulate the drinking time, where one contender drinks longer than the other  
 if (contenderName.equals("Blue")) {  
 Thread.*sleep*(500); // Blue drinks for 500ms  
 } else {  
 Thread.*sleep*(300); // Red drinks for 300ms  
 }  
 System.*out*.println(contenderName + " has finished drinking water.");  
 } //try block ends  
 catch (InterruptedException e) { //catch block starts  
 e.printStackTrace();  
 } //catch block ends  
 }  
}

/\*  
Marla Peraza Ravelo, COP-3330C-17193, 9/29/2024  
The objective of this program is to demonstrate the use of threads and concurrency, by creating two  
threads which will run independently and go into a battle, only one of them being the winner.  
The user will not need to input any values, and everything will be printed out in the screen according to  
the values provided.  
First, each contestant will rest for a certain amount of time. Then, they will perform an action (in this  
case eating an amount of hot dogs). The program will evaluate if the contestant has won (if they have eaten  
200 hot dogs or more). If there is a winner, there will be a message printed out, and if not, the contestants  
will drink water, and the iteration will begin again.  
 \*/  
//class UltimateBattleApp, the main class where we will instantiate the Contender and Water classes, and where  
//all the actions take place  
class UltimateBattleApp {  
 public static void main(String[] args) {  
 // Create a shared Water object  
 Water water = new Water();  
  
 // Create two contenders with different rest times and actions per iteration  
 Contender blue = new Contender("Blue", 5, 200, water);  
 Contender red = new Contender("Red", 10, 400, water);  
  
 // Create and start threads for the contenders  
 Thread thread1 = new Thread(blue);  
 Thread thread2 = new Thread(red);  
  
 thread1.start();  
 thread2.start();  
  
 try {//try block starts  
 // Wait for both threads to complete  
 thread1.join();  
 thread2.join();  
 }//try block ends  
 catch (InterruptedException e) {//catch block starts  
 e.printStackTrace();  
 }//catch block ends  
  
 //prints out a message signaling that the battle is over  
 System.*out*.println("The battle is over!");  
 }  
}