```java
 * $Id: DogWalkingSimulator.java 406 2009-02-19 19:44:41Z msiegen $

package edu.cornell.cs4410.hw2;

public class DogWalkingSimulator {

    private static String[] GOOD_DOGS = new String[] { "Biscuit", "Coco", "Flower",
                                                       "Blue", "Simcha" };
    private static String[] BAD_DOGS = new String[] { "Rocky", "Butch" };

    public DogWalkingSimulator() {
        System.out.print("Starting...\n");
        Park bryantPark = new Park();
        for (String dogName : GOOD_DOGS) {
            GoodDog dog = new GoodDog(dogName);
            Porch porch = new Porch(dog, bryantPark);
            dog.setPorch(porch);
            dog.start();
        }
        for (String dogName : BAD_DOGS) {
            BadDog dog = new BadDog(dogName);
            Porch porch = new Porch(dog, bryantPark);
            dog.setPorch(porch);
            dog.start();
        }
    }

    public static void main(String[] args) {
        new DogWalkingSimulator();
    }

}
```

```
 * $Id: Dog.java 406 2009-02-19 19:44:41Z msiegen $

package edu.cornell.cs4410.hw2;

import java.util.Random;

public class Dog extends Thread {

    protected String dogName;
    protected Random generator = new Random();
    protected Porch porch;

    public Dog(String dogName) {
        this.dogName = dogName;
    }

    // This method will be automatically called in a new thread
    // when the programmer invokes the start() method.
    public void run() {
        try {
            while (true) { // Loop forever...
                takeNap();
                goForWalk();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    private void takeNap() throws InterruptedException {
        int napTime = (int)(generator.nextDouble()*5000);
        System.out.println(dogName + " is napping for " + napTime + " ms");
        Thread.sleep(napTime);
    }

    private void goForWalk() throws InterruptedException {
        System.out.println(dogName + " wants to go for a walk");
        porch.RFID_Detect(); // Wait for green light
        int walkTime = (int)(generator.nextDouble()*2000);
        System.out.println(dogName + " is taking a walk for " + walkTime + " ms");
        porch.RFID_Offline();
        try {
            Thread.sleep(walkTime); // walk for a while
        } finally {
            // Make sure we leave the park, even in the even of an error
            System.out.println(dogName + " has gone home");
            porch.RFID_Detect(); // go back onto the porch
            porch.RFID_Offline(); // go back into the house
        }
    }

    public void setPorch(Porch porch) {
        this.porch = porch;
    }
}
```

```
 * $Id: GoodDog.java 406 2009-02-19 19:44:41Z msiegen $

package edu.cornell.cs4410.hw2;

public class GoodDog extends Dog {

    public GoodDog(String dogName) {
        super(dogName);
    }

}
```

```
 * $Id: BadDog.java 406 2009-02-19 19:44:41Z msiegen $

package edu.cornell.cs4410.hw2;

public class BadDog extends Dog {

    public BadDog(String dogName) {
        super(dogName);
    }
}
```

```java
 * $Id: Porch.java 406 2009-02-19 19:44:41Z msiegen $

package edu.cornell.cs4410.hw2;

public class Porch {

    public enum LightColor {
        BLACK, RED, GREEN
    }

    private Dog dog;
    private Park park;
    private boolean cameFromHouse = true;
    private LightColor light = LightColor.BLACK;

    public Porch(Dog dog, Park park) {
        this.dog = dog;
        this.park = park;
    }

    public void RFID_Detect() throws InterruptedException {
        if (cameFromHouse) {
            // The dog wants to walk
            LIGHT_Set(LightColor.RED);
            park.EnterPark(this.dog); // wait if necessary
            LIGHT_Set(LightColor.GREEN);
            // The dog is now allowed to enter the park
        } else {
            // The dog has finished walking
            park.LeavePark(this.dog);
        }
    }

    public void RFID_Offline() {
        if (cameFromHouse)
            LIGHT_Set(LightColor.BLACK);
        // Assume the dog never changes its mind while on the porch...
        cameFromHouse = !cameFromHouse;
    }

    public void LIGHT_Set(LightColor color) {
        light = color;
        System.out.println("Light for " + dog.dogName + " set to " + light.toString());
    }

}
```

```
 * $Id: Park.java 406 2009-02-19 19:44:41Z msiegen $

package edu.cornell.cs4410.hw2;

import java.util.concurrent.locks.*;

public class Park {

    private int goodDogsInPark = 0;
    private int badDogsInPark = 0;
    private int goodDogsWaiting = 0;
    private int badDogsWaiting = 0;

    // Create a lock for mutual exclusion
    private final Lock lock = new ReentrantLock();

    // Create condition variables that are associated with the above lock
    private final Condition goodDogsWaitingCond = lock.newCondition();
    private final Condition badDogsWaitingCond = lock.newCondition();

    public void EnterPark(Dog dog) throws InterruptedException {

        lock.lock(); // protect the logic in this critical region
        try {

            if (dog instanceof GoodDog) {

                if (badDogsInPark == 0 && badDogsWaiting == 0) {
                    goodDogsInPark++;
                    return; // Successfully entered park
                } else {
                    // Need to wait for some bad dogs to finish their turn
                    goodDogsWaiting++;
                    try {
                        // Wait until this dog is allowed to walk. While
                        // waiting, the lock is released so that other
                        // threads can enter the critical region.
                        goodDogsWaitingCond.await();
                        goodDogsInPark++;
                        return; // Successfully entered park
                    } finally {
                        goodDogsWaiting--;
                    }
                }

            } else if (dog instanceof BadDog) {

                if (goodDogsInPark == 0 && goodDogsWaiting == 0 && badDogsInPark == 0) {
                    badDogsInPark++;
                    return; // Successfully entered park
                } else {
                    // Need to wait for some good dogs to finish their turn
                    badDogsWaiting++;
                    try {
                        badDogsWaitingCond.await();
                        badDogsInPark++;
                        return; // Successfully entered park
                    } finally {
                        badDogsWaiting--;
```

```
                    }
                }

            } else {
                System.out.println("Error: dog is neither good nor bad!");
            }

        } finally {
            // This lock gets released even if an exception occurred
            lock.unlock();
        }
    }

    public void LeavePark(Dog dog) {

        lock.lock();
        try {

            if (dog instanceof GoodDog) {

                goodDogsInPark--;
                if (goodDogsInPark == 0 && badDogsWaiting > 0) {
                    // Allow ONE bad dog to enter the park
                    badDogsWaitingCond.signal();
                }

            } else if (dog instanceof BadDog) {

                badDogsInPark--;
                if (goodDogsWaiting > 0) {
                    // Allow ALL waiting good dogs to enter the park
                    goodDogsWaitingCond.signalAll();
                }

            } else {
                System.out.println("Error: dog is neither good nor bad!");
            }

        } finally {
            // This lock gets released even if an exception occurred
            lock.unlock();
        }
    }

}
```