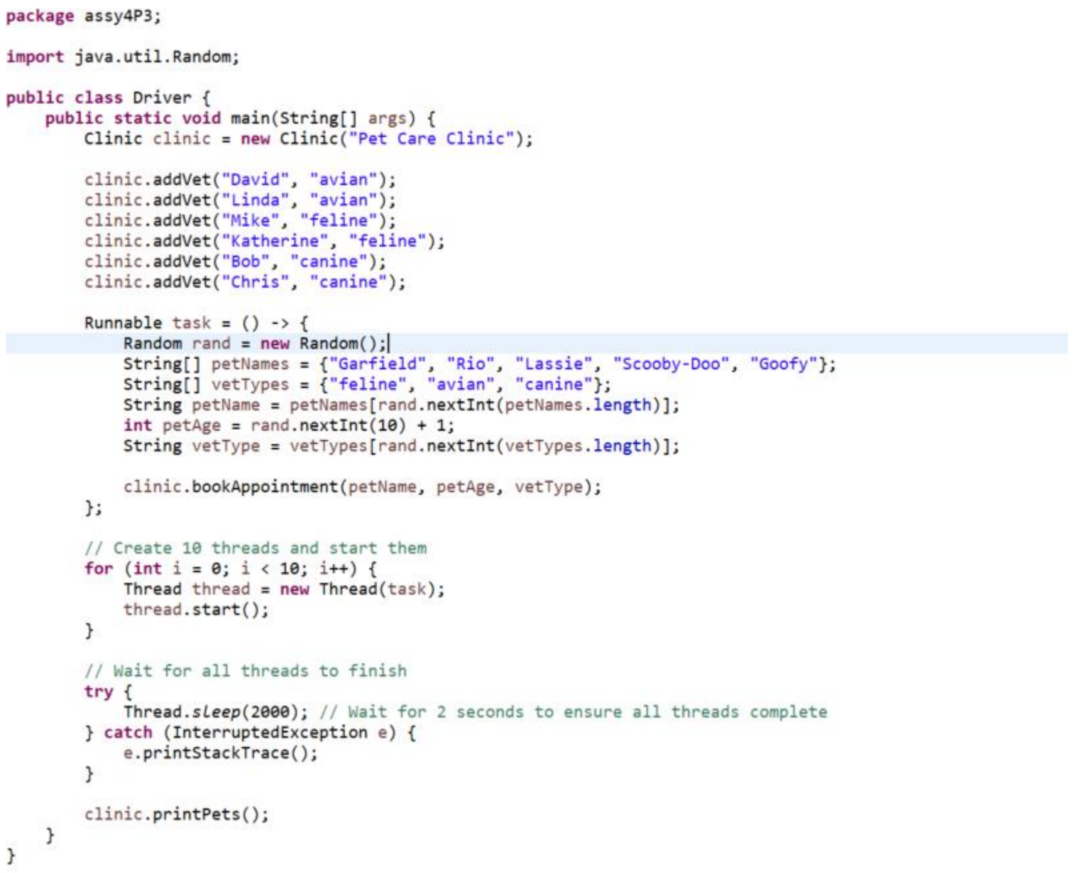
Create a use-case and run the current code in the Driver using multiple threads to double-check that the code is not thread-safe. Ensure that each time you run it, you get a different result. Include the screenshots.

Use-Case Demonstration:

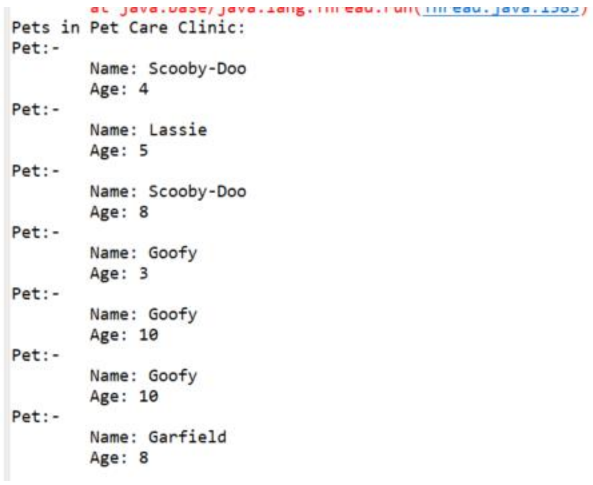
```java
package assy4P3;

import java.util.Random;

public class Driver {
public static void main(String[] args) {
Clinic clinic = new Clinic("Pet Care Clinic");

clinic.addVet("David", "avian");
clinic.addVet("Linda", "avian");
clinic.addVet("Mike", "feline");
clinic.addVet("Katherine", "feline");
clinic.addVet("Bob", "canine");
clinic.addVet("Chris", "canine");

Runnable task = () -> {
Random rand = new Random();
String[] petNames = {"Garfield", "Rio", "Lassie", "Scooby-Doo", "Goofy"};
String[] vetTypes = {"feline", "avian", "canine"};
String petName = petNames[rand.nextInt(petNames.length)];
int petAge = rand.nextInt(10) + 1;
String vetType = vetTypes[rand.nextInt(vetTypes.length)];

clinic.bookAppointment(petName, petAge, vetType);
};

// Create 10 threads and start them
for (int i = 0; i < 10; i++) {
Thread thread = new Thread(task);
thread.start();
}

// Wait for all threads to finish
try {
Thread.sleep(2000); // Wait for 2 seconds to ensure all threads complete
} catch (InterruptedException e) {
e.printStackTrace();
}
clinic.printPets();
}
}
```



```
package assy4P3;

import java.util.Random;

public class Driver {
    public static void main(String[] args) {
        Clinic clinic = new Clinic("Pet Care Clinic");

        clinic.addVet("David", "avian");
        clinic.addVet("Linda", "avian");
        clinic.addVet("Mike", "feline");
        clinic.addVet("Katherine", "feline");
        clinic.addVet("Bob", "canine");
        clinic.addVet("Chris", "canine");

        Runnable task = () -> {
            Random rand = new Random();
            String[] petNames = {"Garfield", "Rio", "Lassie", "Scooby-Doo", "Goofy"};
            String[] vetTypes = {"feline", "avian", "canine"};
            String petName = petNames[rand.nextInt(petNames.length)];
            int petAge = rand.nextInt(10) + 1;
            String vetType = vetTypes[rand.nextInt(vetTypes.length)];

            clinic.bookAppointment(petName, petAge, vetType);
        };

        // Create 10 threads and start them
        for (int i = 0; i < 10; i++) {
            Thread thread = new Thread(task);
            thread.start();
        }

        // Wait for all threads to finish
        try {
            Thread.sleep(2000); // Wait for 2 seconds to ensure all threads complete
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        clinic.printPets();
    }
}
```

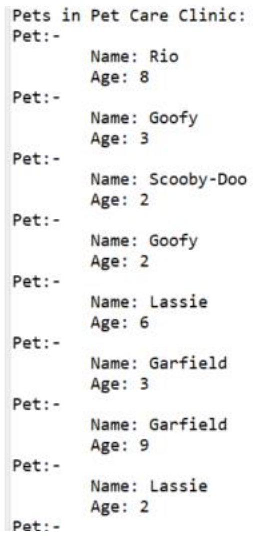
The fact that we get so many random outputs is the perfect indication that the code is not thread-safe

Output1:



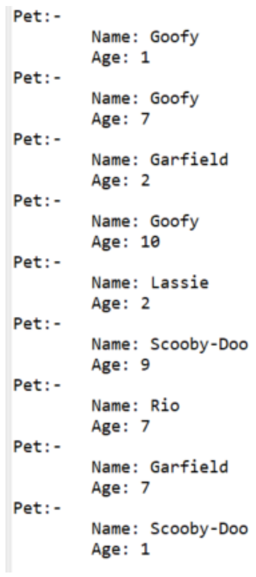
```
        at java.base/java.lang.Thread.run(Thread.java:1583)
Pets in Pet Care Clinic:
Pet:-
        Name: Scooby-Doo
        Age: 4
Pet:-
        Name: Lassie
        Age: 5
Pet:-
        Name: Scooby-Doo
        Age: 8
Pet:-
        Name: Goofy
        Age: 3
Pet:-
        Name: Goofy
        Age: 10
Pet:-
        Name: Goofy
        Age: 10
Pet:-
        Name: Garfield
        Age: 8
```

Output2:



```
Pets in Pet Care Clinic:
Pet:-
        Name: Rio
        Age: 8
Pet:-
        Name: Goofy
        Age: 3
Pet:-
        Name: Scooby-Doo
        Age: 2
Pet:-
        Name: Goofy
        Age: 2
Pet:-
        Name: Lassie
        Age: 6
Pet:-
        Name: Garfield
        Age: 3
Pet:-
        Name: Garfield
        Age: 9
Pet:-
        Name: Lassie
        Age: 2
Pet:-
```

Output3:



```
Pet:-
        Name: Goofy
        Age: 1
Pet:-
        Name: Goofy
        Age: 7
Pet:-
        Name: Garfield
        Age: 2
Pet:-
        Name: Goofy
        Age: 10
Pet:-
        Name: Lassie
        Age: 2
Pet:-
        Name: Scooby-Doo
        Age: 9
Pet:-
        Name: Rio
        Age: 7
Pet:-
        Name: Garfield
        Age: 7
Pet:-
        Name: Scooby-Doo
        Age: 1
```

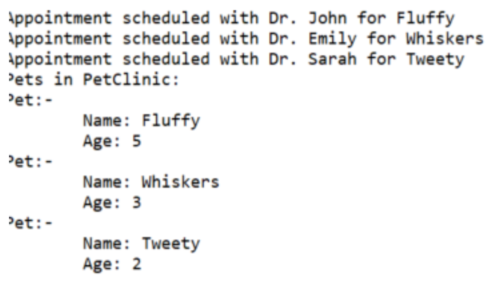
after applying thread safety mechanisms, the code runs well with no errors

```java
package postTHreadSafety;


import java.util.*;

class Canine extends Veterinarian {
public Canine(String name){
super(name);
}
@Override
public String getTitle() {
return "Canine";
}
}

class Feline extends Veterinarian {
public Feline(String name) {
super(name);
}
@Override
public String getTitle() {
return "Feline";
}
}

class Avian extends Veterinarian {
public Avian(String name) {
```

```
super(name);
}
@Override
public String getTitle() {
return "Avian";
}
}

abstract class Veterinarian {
private String name;
private boolean availability;
public Veterinarian(String name) {
this.name = name;
availability = true;
}
/* Getters */
public String getName() {
return name;
}

public boolean getAvailability() {
return availability;
}
// Thread safety: synchronized method to ensure only one thread can set availability
at a time
public synchronized void setAvailability(boolean status) {
availability = status;
}
public String toString() {
StringBuilder sb = new StringBuilder();
sb.append(getTitle()).append(":\n\tName:
").append(this.getName()).append("\n\tAvailable: ").append(this.getAvailability());
return sb.toString();
}
abstract public String getTitle();
}

class Pet {
private String name;
private int age;
private int code;
public static int totalPets; // maintain count of pets

public Pet(String name, int age, int code) {
this.name = name;
this.age = age;
this.code = code;
totalPets++;
}

/* Getters */
public String getName() {
return name;
}

public int getAge() {
return age;
}

public int getCode() {
return code;
}

public String toString() {
return ("Pet:-\n\tName: "+ name+ "\n\tAge: "+age);
}
}

class Clinic {
private String name;
private List<Pet> pets;
private Map<String, PriorityQueue<Veterinarian>> vetMap;

public Clinic(String name) {
this.name = name;
this.vetMap = new HashMap<>();
this.pets = new ArrayList<>();
}

// Thread safety: synchronized method to ensure only one thread can add a vet at a
time
public synchronized void addVet(String name, String dep) {
Veterinarian vet = createDoctor(name, dep);
if (vet != null) {
PriorityQueue<Veterinarian> vets = vetMap.getOrDefault(dep, new
PriorityQueue<>(Comparator.comparing(Veterinarian::getAvailability)));
vets.add(vet);
vetMap.put(dep, vets);
}
}
```

```
private Veterinarian createDoctor(String name, String dep) {
switch (dep) {
case "avian":
return new Avian(name);
case "feline":
return new Feline(name);
case "canine":
return new Canine(name);
default:
System.out.println("Invalid department specified.");
return null;
}
}

/* Booking an appointment */
// Thread safety: synchronized method to ensure only one thread can book an
appointment at a time
public synchronized void bookAppointment(String name, int age, String vetType) {
PriorityQueue<Veterinarian> vets = vetMap.get(vetType);
if (vets != null && !vets.isEmpty()) {
Veterinarian vet = vets.peek();
if (vet != null && vet.getAvailability()) {
System.out.println("Appointment scheduled with " + vet.getName() + " for " + name);
vet.setAvailability(false);
Pet pet = new Pet(name, age, Pet.totalPets + 1);
addPet(pet);
vets.poll();
} else {
System.out.println("No available doctor for the specified type:" + vetType + " for "
+ name);
}
} else {
System.out.println("No doctor available for the specified type:" + vetType + " for "
+ name);
}
}
// Thread safety: synchronized method to ensure only one thread can add a pet at a
time
public synchronized void addPet(Pet pet) {
this.pets.add(pet);
}
// Thread safety: synchronized method to ensure only one thread can print pets at a
time
public synchronized void printPets() {
System.out.println("Pets in " + this.name + ":");
for (Pet pet : pets) {
System.out.println(pet);
}
}
}
```

Output :



```
Appointment scheduled with Dr. John for Fluffy
Appointment scheduled with Dr. Emily for Whiskers
Appointment scheduled with Dr. Sarah for Tweety
Pets in PetClinic:
Pet:-
        Name: Fluffy
        Age: 5
Pet:-
        Name: Whiskers
        Age: 3
Pet:-
        Name: Tweety
        Age: 2
```