

## **Week 5 Homework Q19**

Telmen Enkhbold

San Fransico Bay University

CE480 - Java and Internet Application

Dr. Chang, Henry

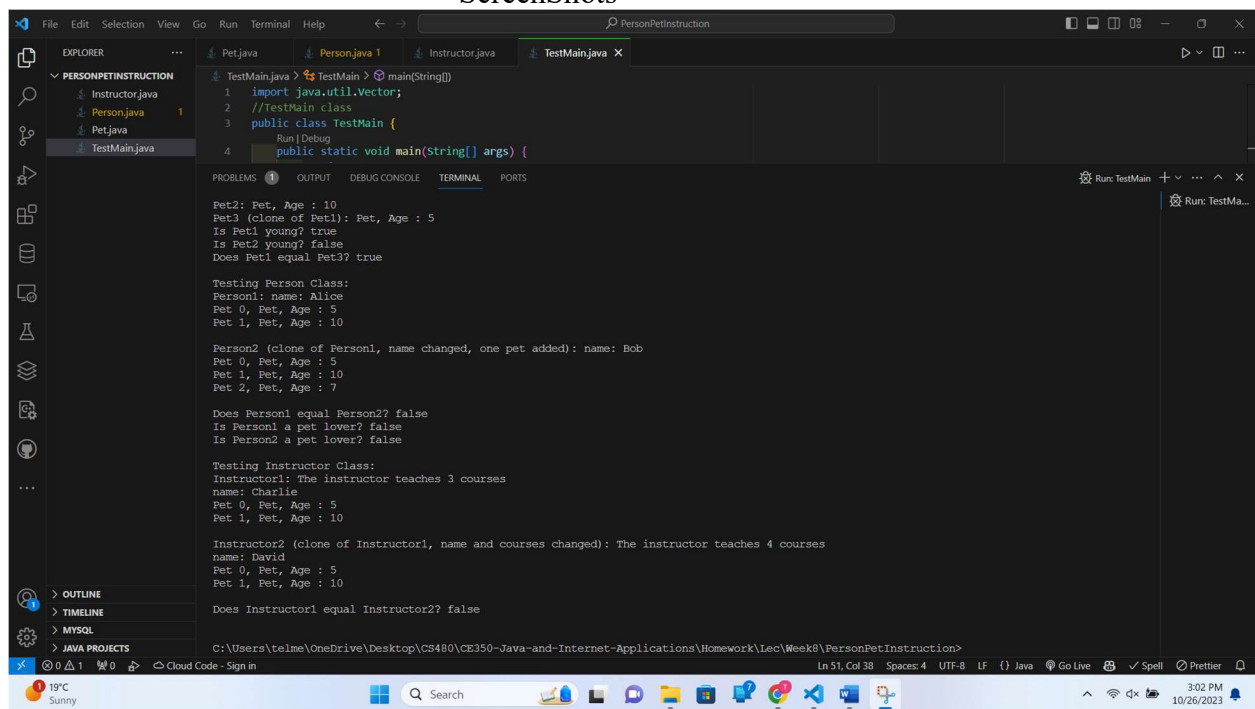
10/12/2023

**Author Note**

## The Question

## Object-Oriented Programming: Person + Pet + Instructore

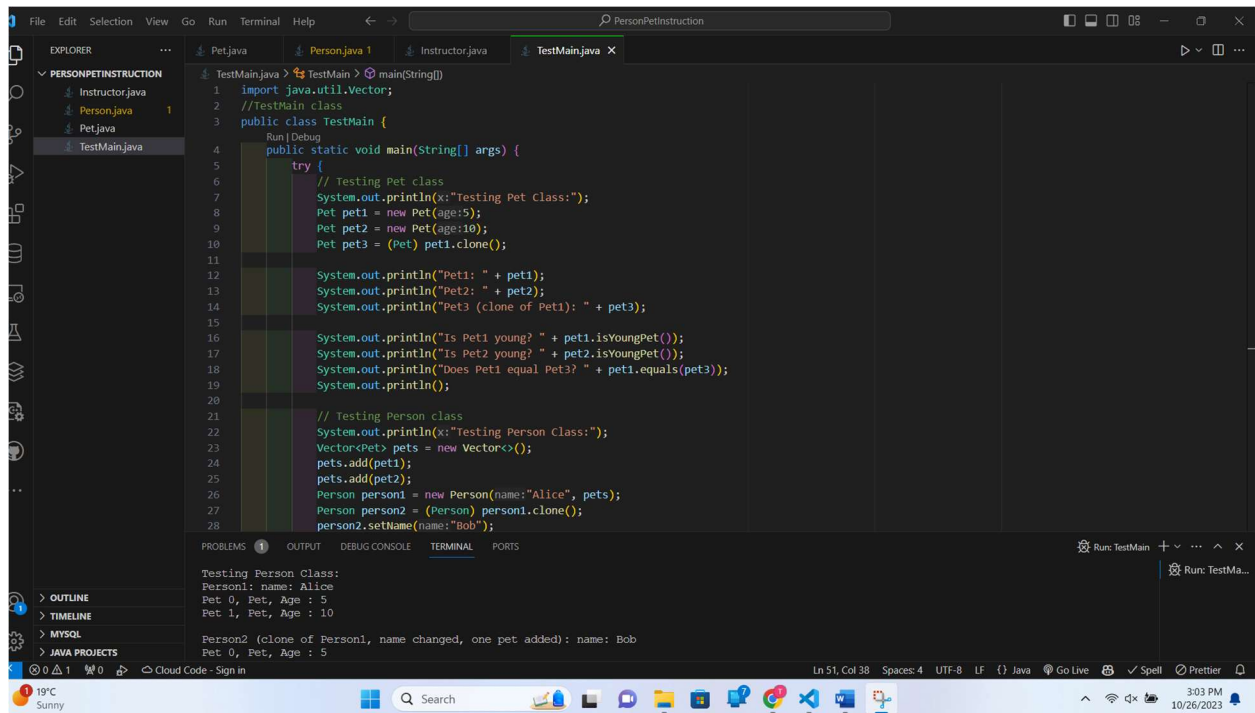
### ScreenShots



The screenshot shows an IDE window titled "PersonPetInstruction" with the following components:

- EXPLORER:** Shows a project named "PERSONPETINSTRUCTION" with files: `Instructor.java`, `Person.java`, `Pet.java`, and `TestMain.java`.
- EDITOR:** Displays the code for `TestMain.java`. The code includes imports for `java.util.Vector` and `TestMain` class, and a `main` method that tests various objects and methods.
- TERMINAL:** Shows the output of the program execution. The output includes:
  - Creation and testing of `Pet` objects: `Pet2: Pet, Age : 10`, `Pet3 (clone of Pet1): Pet, Age : 5`, and comparisons like `Is Pet1 young? true`.
  - Creation and testing of `Person` objects: `Person1: name: Alice`, `Pet 0, Pet, Age : 5`, `Pet 1, Pet, Age : 10`, and comparisons like `Does Person1 equal Person2? false`.
  - Creation and testing of `Instructor` objects: `Instructor1: The instructor teaches 3 courses`, `name: Charlie`, `Pet 0, Pet, Age : 5`, `Pet 1, Pet, Age : 10`, and comparisons like `Does Instructor1 equal Instructor2? false`.

The status bar at the bottom indicates the file is at line 51, column 38, using UTF-8 encoding, and the system clock shows 3:02 PM on 10/26/2023.

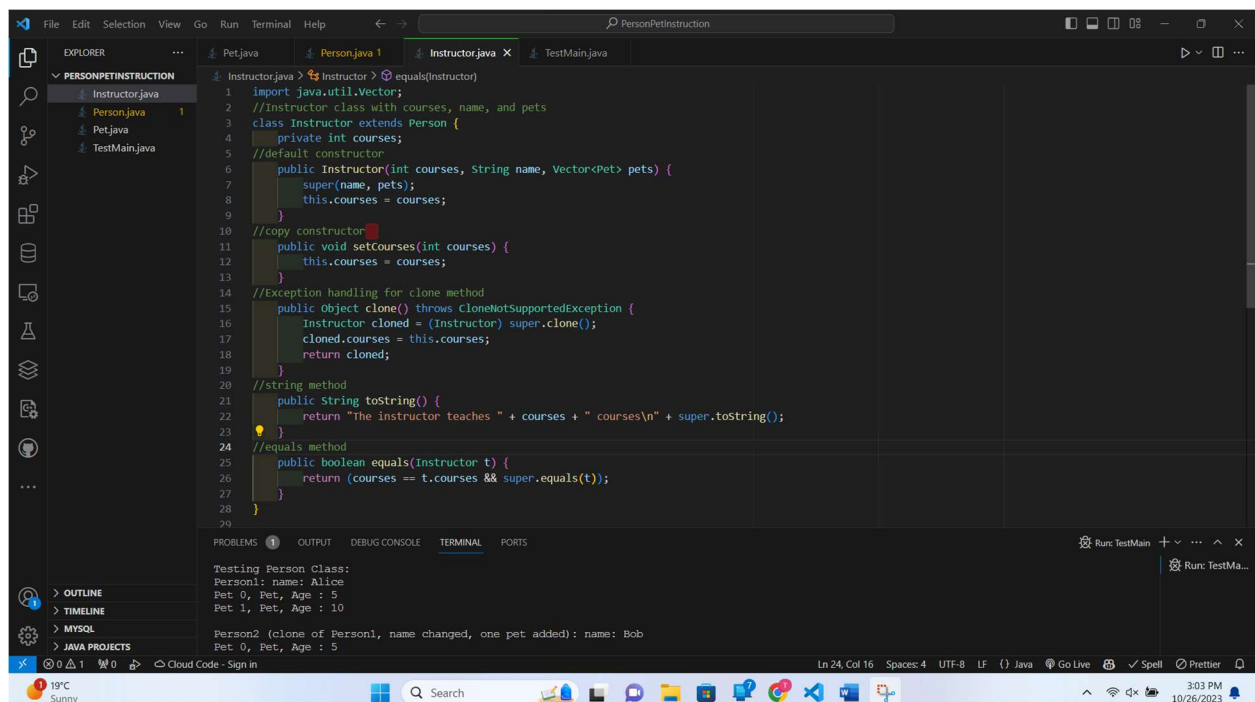


This screenshot shows the VS Code editor with the `TestMain.java` file open. The Explorer sidebar on the left shows the project structure: `PERSONPETINSTRUCTION` containing `Instructor.java`, `Person.java`, `Pet.java`, and `TestMain.java`. The `TestMain.java` file contains the following code:

```
1 import java.util.Vector;
2 //TestMain class
3 public class TestMain {
4     public static void main(String[] args) {
5         try {
6             // Testing Pet class
7             System.out.println("Testing Pet Class:");
8             Pet pet1 = new Pet(age:5);
9             Pet pet2 = new Pet(age:10);
10            Pet pet3 = (Pet) pet1.clone();
11
12            System.out.println("Pet1: " + pet1);
13            System.out.println("Pet2: " + pet2);
14            System.out.println("Pet3 (clone of Pet1): " + pet3);
15
16            System.out.println("Is Pet1 young? " + pet1.isYoungPet());
17            System.out.println("Is Pet2 young? " + pet2.isYoungPet());
18            System.out.println("Does Pet1 equal Pet3? " + pet1.equals(pet3));
19            System.out.println();
20
21            // Testing Person class
22            System.out.println("Testing Person Class:");
23            Vector<Pet> pets = new Vector<>();
24            pets.add(pet1);
25            pets.add(pet2);
26            Person person1 = new Person(name:"Alice", pets);
27            Person person2 = (Person) person1.clone();
28            person2.setName(name:"Bob");
```

The Output window at the bottom shows the following text:

```
Testing Person Class:
Person1: name: Alice
Pet 0, Pet, Age : 5
Pet 1, Pet, Age : 10
Person2 (clone of Person1, name changed, one pet added): name: Bob
Pet 0, Pet, Age : 5
```



This screenshot shows the VS Code editor with the `Instructor.java` file open. The Explorer sidebar on the left shows the project structure: `PERSONPETINSTRUCTION` containing `Instructor.java`, `Person.java`, `Pet.java`, and `TestMain.java`. The `Instructor.java` file contains the following code:

```
1 import java.util.Vector;
2 //Instructor class with courses, name, and pets
3 class Instructor extends Person {
4     private int courses;
5     //default constructor
6     public Instructor(int courses, String name, Vector<Pet> pets) {
7         super(name, pets);
8         this.courses = courses;
9     }
10    //copy constructor
11    public void setCourses(int courses) {
12        this.courses = courses;
13    }
14    //Exception handling for clone method
15    public Object clone() throws CloneNotSupportedException {
16        Instructor cloned = (Instructor) super.clone();
17        cloned.courses = this.courses;
18        return cloned;
19    }
20    //string method
21    public String toString() {
22        return "The instructor teaches " + courses + " courses\n" + super.toString();
23    }
24    //equals method
25    public boolean equals(Instructor t) {
26        return (courses == t.courses && super.equals(t));
27    }
28 }
29
```

The Output window at the bottom shows the following text:

```
Testing Person Class:
Person1: name: Alice
Pet 0, Pet, Age : 5
Pet 1, Pet, Age : 10
Person2 (clone of Person1, name changed, one pet added): name: Bob
Pet 0, Pet, Age : 5
```

```

1  Person.java > Person > removePets()
2  import java.util.Vector;
3
4  //Person class with name and pets
5  class Person implements Cloneable {
6      String name;
7      private Vector<Pet> pets;
8      //default constructor
9      public Person(String name, Vector<Pet> pets) {
10         this.name = name;
11         this.pets = pets;
12     }
13     //Exception handling for clone method
14     public Object clone() throws CloneNotSupportedException {
15         Person cloned = (Person) super.clone();
16         cloned.pets = (Vector<Pet>) this.pets.clone();
17         return cloned;
18     }
19     //getters and setters
20     public Pet getPet(int i) {
21         return pets.get(i);
22     }
23     public int getPetAge(int i) {
24         return pets.get(i).getAge();
25     }
26     public Vector<Pet> getPets() {
27
28
29

```

Testing Person Class:  
Person1: name: Alice  
Pet 0, Pet, Age : 5  
Pet 1, Pet, Age : 10  
Person2 (clone of Person1, name changed, one pet added): name: Bob  
Pet 0, Pet, Age : 5

```

1  Pet.java > Pet > isYoungPet()
2  //Pet class for use in the Pet class hierarchy
3  class Pet implements Cloneable {
4      private int age;
5      //default constructor
6      public Pet(int age) {
7         this.age = age;
8     }
9     //Exception handling for clone method
10    public Object clone() throws CloneNotSupportedException {
11        return super.clone();
12    }
13    //getters and setters
14    public int getAge() {
15        return age;
16    }
17    public void setAge(int age) {
18        this.age = age;
19    }
20    //methods: isYoungPet, toString, equals
21    public boolean isYoungPet() {
22        return age < 10;
23    }
24    public String toString() {
25        return "Pet, Age : " + age;
26    }
27    public boolean equals(Pet k) {
28
29

```

Testing Person Class:  
Person1: name: Alice  
Pet 0, Pet, Age : 5  
Pet 1, Pet, Age : 10  
Person2 (clone of Person1, name changed, one pet added): name: Bob  
Pet 0, Pet, Age : 5

## The Code

### Instructor.java

```

import java.util.Vector;
//Instructor class with courses, name, and pets

```

```
class Instructor extends Person {
    private int courses;
    //default constructor
    public Instructor(int courses, String name, Vector<Pet> pets) {
        super(name, pets);
        this.courses = courses;
    }
    //copy constructor
    public void setCourses(int courses) {
        this.courses = courses;
    }
    //Exception handling for clone method
    public Object clone() throws CloneNotSupportedException {
        Instructor cloned = (Instructor) super.clone();
        cloned.courses = this.courses;
        return cloned;
    }
    //string method
    public String toString() {
        return "The instructor teaches " + courses + " courses\n" +
super.toString();
    }
    //equals method
    public boolean equals(Instructor t) {
        return (courses == t.courses && super.equals(t));
    }
}
```

### Person.java

```
import java.util.Vector;

//Person class with name and pets
class Person implements Cloneable {
    String name;
    private Vector<Pet> pets;
    //default constructor
    public Person(String name, Vector<Pet> pets) {
        this.name = name;
        this.pets = pets;
    }
    //Exception handling for clone method
    public Object clone() throws CloneNotSupportedException {
        Person cloned = (Person) super.clone();
        cloned.pets = (Vector<Pet>) this.pets.clone();
    }
}
```

```
        return cloned;
    }

//getters and setters
    public Pet getPet(int i) {
        return pets.get(i);
    }

    public int getPetAge(int i) {
        return pets.get(i).getAge();
    }

    public Vector<Pet> getPets() {
        return pets;
    }

    public int getNumOfPets() {
        return pets.size();
    }

    public void setPet(int i, int age) {
        pets.get(i).setAge(age);
    }

    public void setPets(Vector<Pet> pets) {
        this.pets = pets;
    }

    public void setName(String name) {
        this.name = name;
    }

//methods: isPetLover, removeAllPets, addPet, insertPet, firstPet,
lastPet, removePet, toString, equals
    public boolean isPetLover() {
        return pets.size() > 3;
    }

    public void removeAllPets() {
        pets.clear();
    }

    public void addPet(int age) {
        pets.add(new Pet(age));
    }
}
```

```
public void insertPet(int age, int index) {
    pets.add(index, new Pet(age));
}

public Pet firstPet() {
    return pets.firstElement();
}

public Pet lastPet() {
    return pets.lastElement();
}

public void removePet(int i) {
    pets.remove(i);
}

//toString and equals methods
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("name: ").append(name).append("\n");
    for (int i = 0; i < pets.size(); i++) {
        sb.append("Pet ").append(i).append(",
").append(pets.get(i).toString()).append("\n");
    }
    return sb.toString();
}

//equals method
public boolean equals(Person t) {
    if (!name.equals(t.name)) {
        return false;
    }
    if (getNumOfPets() != t.getNumOfPets()) {
        return false;
    }
    for (int i = 0; i < getNumOfPets(); i++) {
        if (!getPet(i).equals(t.getPet(i))) {
            return false;
        }
    }
    return true;
}

//Destructor
public void removePets() {
    pets.clear();
}
```

```
}
```

**Pet.java**

```
//Pet class for use in the Pet class hierarchy
class Pet implements Cloneable {
    private int age;
    //default constructor
    public Pet(int age) {
        this.age = age;
    }
    //Exception handling for clone method
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
    //getters and setters
    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
    //methods: isYoungPet, toString, equals
    public boolean isYoungPet() {
        return age < 10;
    }

    public String toString() {
        return "Pet, Age : " + age;
    }

    public boolean equals(Pet k) {
        return this.age == k.getAge();
    }
}
```

**TestMain.java**

```
import java.util.Vector;
//TestMain class
public class TestMain {
    public static void main(String[] args) {
        try {
```



```
// Testing Pet class
System.out.println("Testing Pet Class:");
Pet pet1 = new Pet(5);
Pet pet2 = new Pet(10);
Pet pet3 = (Pet) pet1.clone();

System.out.println("Pet1: " + pet1);
System.out.println("Pet2: " + pet2);
System.out.println("Pet3 (clone of Pet1): " + pet3);

System.out.println("Is Pet1 young? " + pet1.isYoungPet());
System.out.println("Is Pet2 young? " + pet2.isYoungPet());
System.out.println("Does Pet1 equal Pet3? " +
pet1.equals(pet3));
System.out.println();

// Testing Person class
System.out.println("Testing Person Class:");
Vector<Pet> pets = new Vector<>();
pets.add(pet1);
pets.add(pet2);
Person person1 = new Person("Alice", pets);
Person person2 = (Person) person1.clone();
person2.setName("Bob");
person2.addPet(7);

System.out.println("Person1: " + person1);
System.out.println("Person2 (clone of Person1, name changed,
one pet added): " + person2);
System.out.println("Does Person1 equal Person2? " +
person1.equals(person2));
System.out.println("Is Person1 a pet lover? " +
person1.isPetLover());
System.out.println("Is Person2 a pet lover? " +
person2.isPetLover());
System.out.println();

// Testing Instructor class
System.out.println("Testing Instructor Class:");
Instructor instructor1 = new Instructor(3, "Charlie", pets);
Instructor instructor2 = (Instructor) instructor1.clone();
instructor2.setCourses(4);
instructor2.setName("David");

System.out.println("Instructor1: " + instructor1);
```

```
        System.out.println("Instructor2 (clone of Instructor1, name  
and courses changed): " + instructor2);  
        System.out.println("Does Instructor1 equal Instructor2? " +  
instructor1.equals(instructor2));  
        System.out.println();  
    }  
    //Exception handling for clone method  
    catch (CloneNotSupportedException e) {  
        System.out.println("Clone not supported: " + e.getMessage());  
    }  
}  
}
```

## Reference

<https://hc.labnet.sfbu.edu/~henry/sfbu/course/introjava/inheritance/slide/exercises4.html>  
Links to an external site.

Q19 ==> Object-Oriented Programming: Person + Pet + Instructor

## Github

[https://github.com/Georgycas/CE350-Java-and-Internet-](https://github.com/Georgycas/CE350-Java-and-Internet-Applications/tree/main/Homework/Lec/Week8/PersonPetInstruction)

[Applications/tree/main/Homework/Lec/Week8/PersonPetInstruction](https://github.com/Georgycas/CE350-Java-and-Internet-Applications/tree/main/Homework/Lec/Week8/PersonPetInstruction)