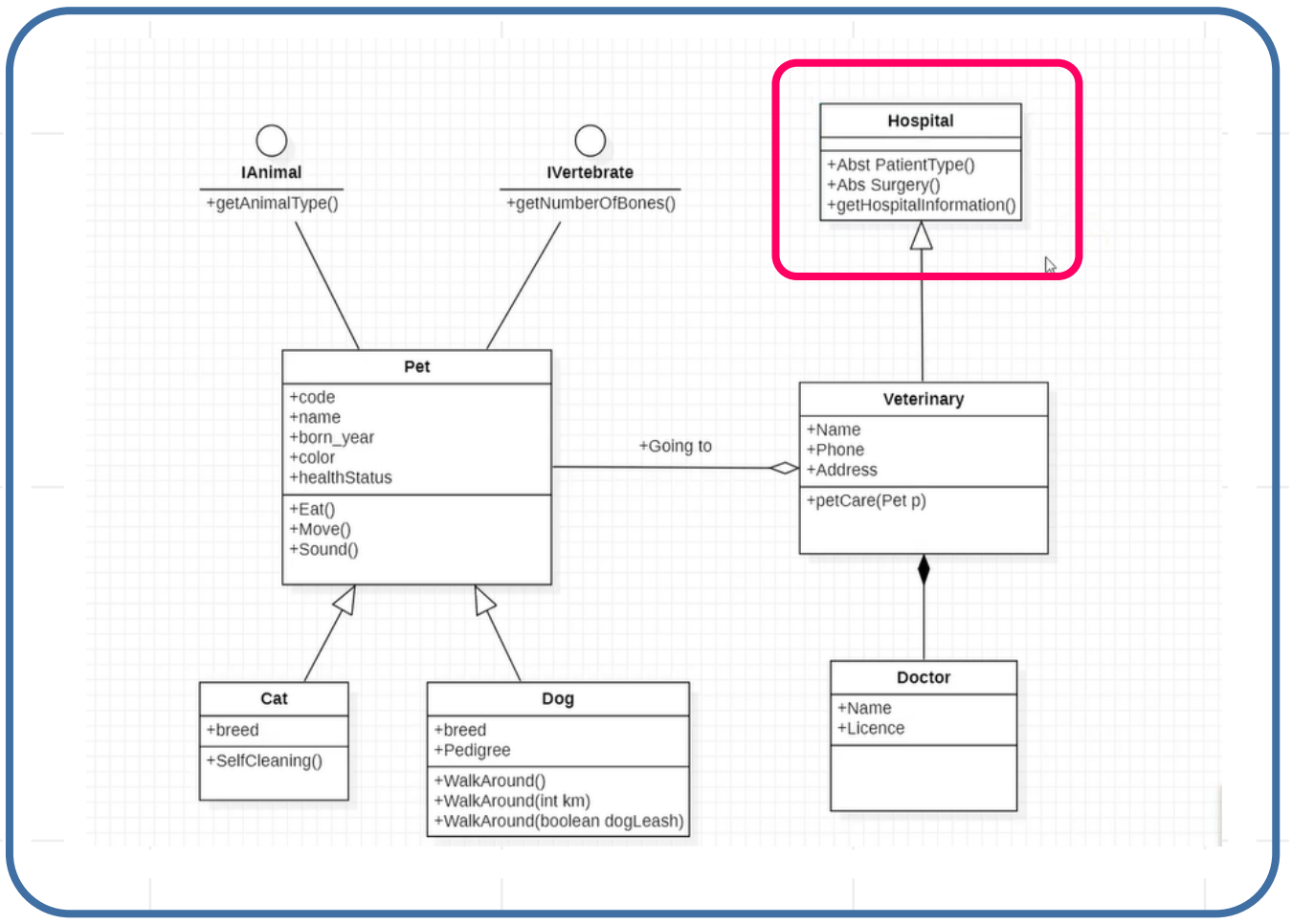
Clase Abstracta

+Abst

Hola:

Ya conocemos los dos (2) tipos de clases, la primera clase donde se debe definir todo y la segunda donde solo se define la firma de los métodos. Es este caso el punto intermedio será la Clase Abstracta y que no representan algo específico y se puede usar para crear otras clases. No pueden ser instanciadas, por lo que no se pueden crear nuevos objetos con ellas y se ubican en un intermedio entre clase e interface, esto debido a que la clase abstracta puede definir solo la firma de un método o implementarlo completamente. Las clases abstractas pueden ser heredadas y las clases hijas deberán implementar los métodos que no tengan cuerpo.

Video de clase abstracta implementadas por clases (Clase mascota con método implementado de comer y hacer sonido será abstracto).



PetManager - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Classes;

/**
 *
 * @author USUARIO
 */
public abstract class AbstHospital {
    public abstract String getPatientType();
    public abstract String Surgery();

}
```

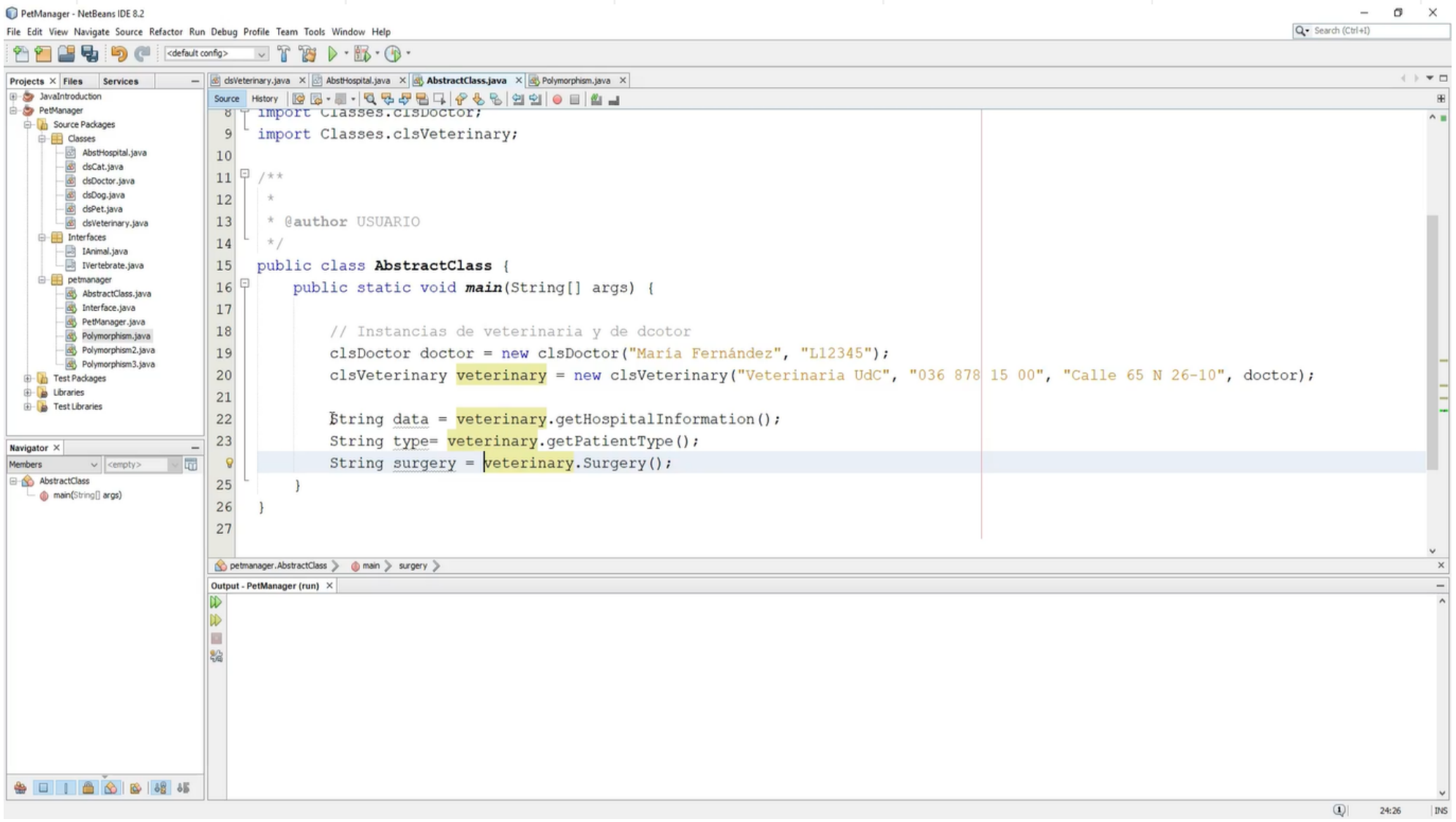
PetManager - NetBeans IDE 8.2

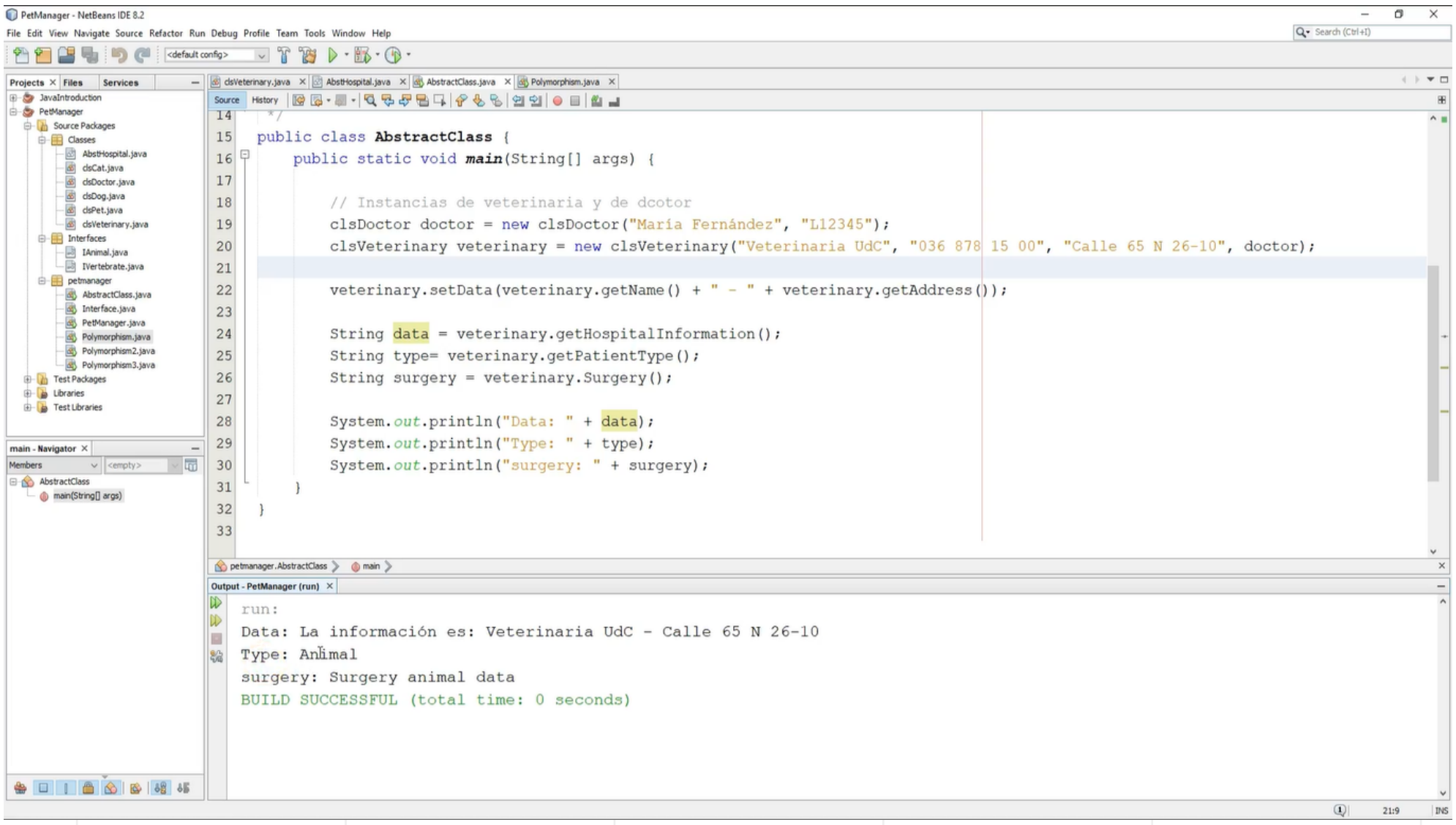
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Classes;

/**
 *
 * @author USUARIO
 */
public class clsVeterinary extends AbstHospital {
    private String name;
    private String phone;
    private String address;
    private clsDoctor doctor;

    public clsVeterinary(String name, String phone, String address, clsDoctor doctor) {
        this.name = name;
        this.phone = phone;
```


PetManager - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Files Services
JavaIntroduction
PetManager
Source Packages
Classes
AbstHospital.java
dsCat.java
clsDoctor.java
clsDog.java
clsPet.java
clsVeterinary.java
Interfaces
IAnimal.java
IVertebrate.java
petmanager
AbstractClass.java
Interface.java
PetManager.java
Polymorphism.java
Polymorphism2.java
Polymorphism3.java
Test Packages
Libraries
Test Libraries
Navigator
Members
AbstractClass
main(String[] args)
clsVeterinary.java
AbstHospital.java
AbstractClass.java
Polymorphism.java
Source History
import Classes.clsDoctor;
import Classes.clsVeterinary;
/**
*
* @author USUARIO
*/
public class AbstractClass {
public static void main(String[] args) {
// Instancias de veterinaria y de dcotor
clsDoctor doctor = new clsDoctor("María Fernández", "L12345");
clsVeterinary veterinary = new clsVeterinary("Veterinaria UdC", "036 878 15 00", "Calle 65 N 26-10", doctor);
String data = veterinary.getHospitalInformation();
String type= veterinary.getPatientType();
String surgery = veterinary.Surgery();
}
}
petmanager.AbstractClass
main
surgery
Output - PetManager (run)
24:26
INS


PetManager - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Files Services
JavaIntroduction
PetManager
Source Packages
Classes
AbstHospital.java
dsCat.java
clsDoctor.java
clsDog.java
clsPet.java
clsVeterinary.java
Interfaces
IAnimal.java
IVertebrate.java
petmanager
AbstractClass.java
Interface.java
PetManager.java
Polymorphism.java
Polymorphism2.java
Polymorphism3.java
Test Packages
Libraries
Test Libraries
main - Navigator
Members
AbstractClass
main(String[] args)
clsVeterinary.java
AbstHospital.java
AbstractClass.java
Polymorphism.java
Source History
*/
public class AbstractClass {
public static void main(String[] args) {
// Instancias de veterinaria y de dcotor
clsDoctor doctor = new clsDoctor("María Fernández", "L12345");
clsVeterinary veterinary = new clsVeterinary("Veterinaria UdC", "036 878 15 00", "Calle 65 N 26-10", doctor);
veterinary.setData(veterinary.getName() + " - " + veterinary.getAddress());
String data = veterinary.getHospitalInformation();
String type= veterinary.getPatientType();
String surgery = veterinary.Surgery();
System.out.println("Data: " + data);
System.out.println("Type: " + type);
System.out.println("surgery: " + surgery);
}
}
petmanager.AbstractClass
main
Output - PetManager (run)
run:
Data: La información es: Veterinaria UdC - Calle 65 N 26-10
Type: Animal
surgery: Surgery animal data
BUILD SUCCESSFUL (total time: 0 seconds)
21:9
INS

Con esta temática finalizamos la introducción a toda esta conceptualización, tanto teórica como práctica en el paradigma de programación orientada a objetos. Los invitamos a que pongan en práctica este nuevo conocimiento mediante el reto de la semana.

Universidad de Caldas