

TALLER DE LENGUAJE DE PROGRAMACIÓN I

Entorno de CONSOLA y GRÁFICOS

UNIDAD 2

Programación Orientada a Objetos – POO

Herencia – Polimorfismo – Clases Abstractas – Clases Interfaces

JAVA

OBJETIVO

- Herencia
- Polimorfismo
- Clase Abstracta
- Clases Interfaces

Trabajo Independiente

- Ejercicio

Herencias (POO)

Herencia

Es la forma como una clase permite heredar

- Métodos constructores – Estado.
- Atributos – características
- Métodos de acción - comportamiento de otra clase.

Ventajas

Reutilización de código.

Permite definir los propios atributos y métodos.

Desarrollo de software más eficiente.

Se definen los conceptos

SuperClase → La clase que se hereda

SubClase → La clase que incorpora la herencia

Herencias (POO), continuación

Herencia

Super Clase

ArmarComputador

Sub Clase

Servidor



```
graph BT; Servidor --> ArmarComputador
```

The diagram illustrates inheritance. A red box labeled 'ArmarComputador' is at the top, and a red box labeled 'Servidor' is below it. A red arrow points from the 'Servidor' box up to the 'ArmarComputador' box, indicating that 'Servidor' inherits from 'ArmarComputador'.

Para determinar si la definición de la herencia está bien definida, utilizamos la regla “Es un...”

¿Un ArmarComputador es un Servidor?

¿Un Servidor es un ArmarComputador?

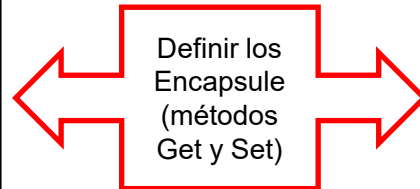
Herencias (POO), continuación

En el proyecto **armarcomputado**, se definen las clases Chasis y DiscoDuro en la carpeta bean

```
package bean;

public class Chasis
{
    private int largoPlaca;
    private int anchoPlaca;
    private int ranuras;
    private boolean adminCables;
    private int ancho;
    private int alto;
    private int profundidad;

    public Chasis ()
    {
        largoPlaca=30;
        anchoPlaca=24;
        ranuras=8;
        adminCables=true;
        ancho=60;
        alto=45;
        profundidad=83;
    }
}
```



```
package bean;

public class DiscoDuro
{
    private String tipoDisco; //Define si es magnetico o SSD
    private String interfaz; //IDE, SATA, SCSI, SAS, SATA Express
    private int capacidad; //Capacidad de almacenamiento en GigaByte

    public DiscoDuro()
    {
        tipoDisco = "Magnetico";
        interfaz = "SATA";
        capacidad = 100;
    }
}
```

Herencias (POO), continuación

Se define la composición en la clase ArmarComputador

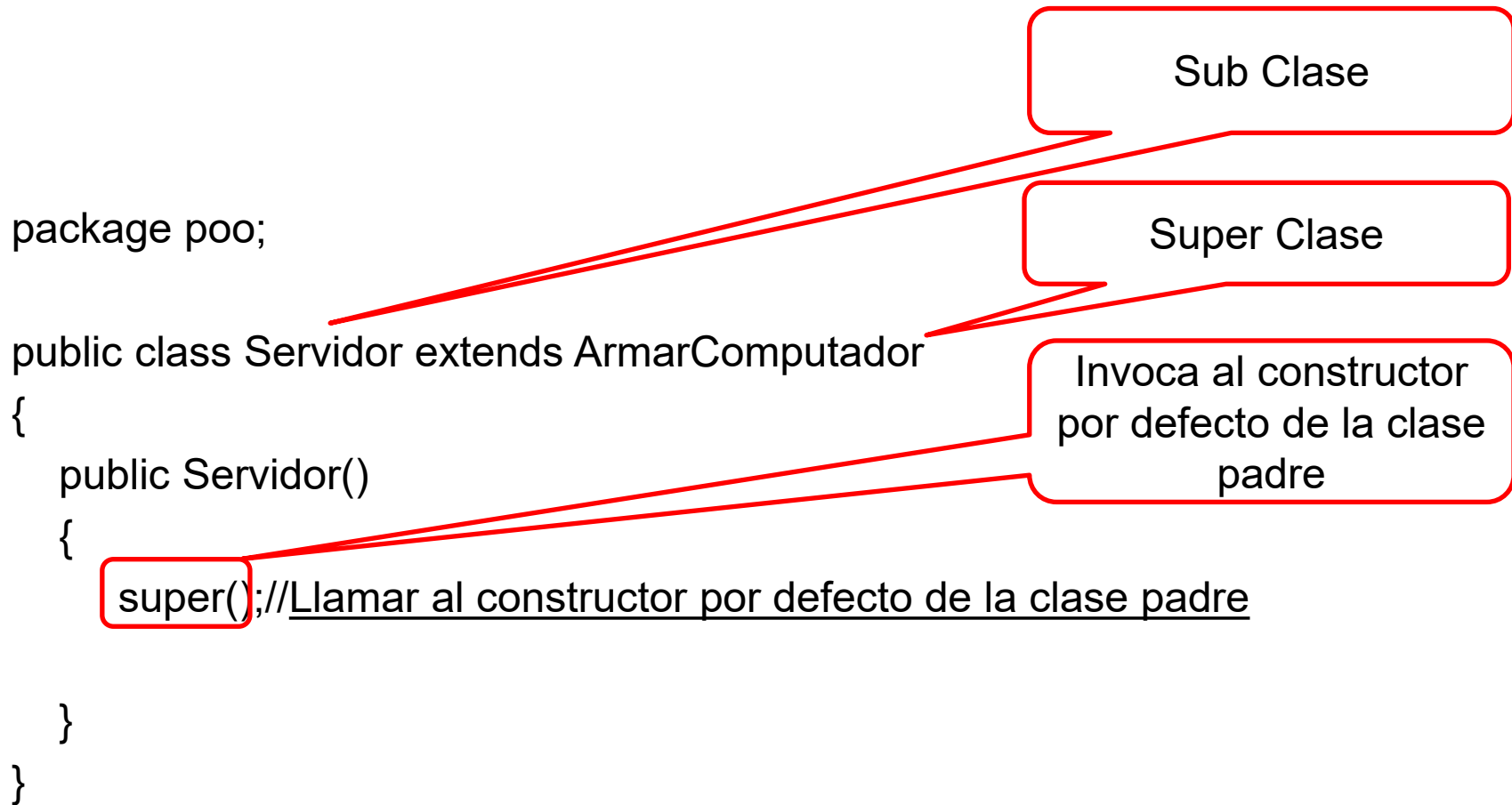
```
package bean;

public class ArmarComputador {
    private Chasis chasis;
    private DiscoDuro dd;

    public ArmarComputador(Chasis chasis, DiscoDuro dd) {
        this.chasis = chasis;
        this.dd = dd;
    }
    public Chasis getChasis() {
        return chasis;
    }
    public void setChasis(Chasis chasis) {
        this.chasis = chasis;
    }
    public DiscoDuro getDd() {
        return dd;
    }
    public void setDd(DiscoDuro dd) {
        this.dd = dd;
    }
}
```

Herencias (POO), continuación

Se define la HERENCIA en la clase Servidor extendiendo la clase ArmarComputador



Herencias (POO), continuación

Definir la clase Inicio

```
package poo;
public class Inicio
{
    public static void main(String[] args)
    {
        Chasis cmtatx = new Chasis();
        DiscoDuro dd = new DiscoDuro();
        Servidor s = new Servidor();
        s.setChasis(cmtatx);
        s.setDd(dd);
        System.out.println("El servidor tiene las siguientes características: ");
        System.out.println("    Dimensión de la Placa base: " + s.getChasis().getLargoPlaca() + "cm X " +
            s.getChasis().getAnchoPlaca() + "cm");
        System.out.println("    Medidas: " + s.getChasis().getAlto() + "cm");
        System.out.println("Disco Duro");
        System.out.println("    Interfaz: " + s.getDd().getInterfaz());
    }
}
```


Polimorfismo (POO)

Es la capacidad de un objeto de adquirir varias formas, en la POO lo más común es cuando se utiliza la referencia de una clase padre, para referirse al objeto de la clase hijo.

Veamos desde el ejemplo.

Polimorfismo (POO), continuación

Crea el proyecto poonomina, que permitirá capturar N empleados y alguno de ellos serán jefes los que gozaran de una bonificación al salario base.

Definiremos una clase con los atributos propios de un empleado llamada BEmpleado, sin método main.

IdEmp → Entero

nombreEmp → Cadena de caracteres

sueldoEmp → Entero grande

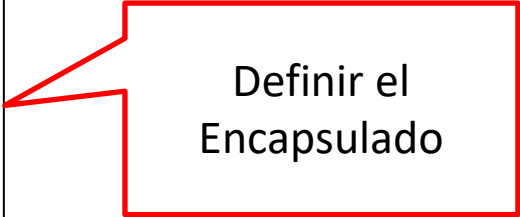
fechaNacEmp → Date



Los Atributos se encapsulan

Polimorfismo (POO), continuación

```
package poonominas;  
import java.util.*;  
public class BEmpleado  
{  
    private int    idEmp;  
    private String  nombreEmp;  
    private long    sueldoEmp;  
    private Date    fechaNacEmp;  
  
    public BEmpleado() {};  
}
```



Definir el
Encapsulado

Polimorfismo (POO), continuación

Se crea otra clase LEmpleado, donde se definen los métodos constructores y las acciones del empleado, generando una herencia de la clase BEmpleado.

Con tres métodos:

Método constructor vacío	→ LEmpleado()
Método Registrar Empleados	→ LEmpleadoRegistrar()
Método Aumentar el sueldo	→ LaumentoSueldo()

sin método main.

Polimorfismo (POO), continuación

```
package poo_nomina;
import java.util.GregorianCalendar;
import javax.swing.JOptionPane;

public class LEmpleado extends BEmpleado
{
    public LEmpleado() {}

    public LEmpleado(int id,String nombre, long sueldo, int annio, int mes, int dia) {
        super.setIdEmp(id);
        super.setNombreEmp(nombre);
        super.setSueldoEmp(sueldo);
        super.setFechaNacEmp(LocalDate.of(annio,mes,dia));
    }

    public void LEmpleadoRegistrar(int id,String nombre, long sueldo, int annio, int mes, int dia) {
        super.setIdEmp(id);
        super.setNombreEmp(nombre);
        super.setSueldoEmp(sueldo);
        super.setFechaNacEmp(LocalDate.of(annio,mes,dia));
    }

    public void LEmpleadoRegistrar(BEmpleado empleado, int annio, int mes, int dia)
    {
        super.setIdEmp(empleado.getIdEmp());
        super.setNombreEmp(empleado.getNombreEmp());
        super.setSueldoEmp(empleado.getSueldoEmp());
        super.setFechaNacEmp(LocalDate.of(annio,mes,dia));
    }

    public void LAumentoSueldo(double porcentaje) {
        long aumento = (long)(getSueldoEmp() * porcentaje/100);
        super.setSueldoEmp(super.getSueldoEmp() + aumento);
    }
}
```

Polimorfismo (POO), continuación

Crear la clase LEmpleadoJefe, que hereda la clase LEmpleado, que define un empleado jefe, con la acción de asignarle una bonificación.

Sin método main.

Polimorfismo (POO), continuación

```
package poo_nomina;
import bean.BEmpleado;
public class LEmpleadoJefe extends LEmpleado{

    private long bonificacion;

    public LEmpleadoJefe(int id,String nombre, long sueldo, int annio, int mes, int dia) {
        super(id,nombre,sueldo,annio,mes,dia);
    }

    public void LEmpleadoJefeRegistrar(BEmpleado empleado, int annio, int mes, int dia){
        try{
            super.IEmpleadoRegistrar(empleado, annio, mes, dia);

        }catch(Exception e) {System.out.println("Problemas con el acceso al archivo de datos.");}
    }

    public void estableceBonificacion(long valor) {
        bonificacion = valor;
    }

    public long getSueldoEmp() {
        long salariojefe = super.getSueldoEmp();
        return (salariojefe + bonificacion);
    }
}
```

Polimorfismo (POO), continuación

```
package poo_nomina;

import java.util.Scanner;

public class Inicio {

    public static void main(String[] args)
    {
        Scanner captura = new Scanner(System.in);

        System.out.println("Especifique el número de empleados para ingresar.");
        int nroEmpleados = 0;
        String tipoEmpleado;

        nroEmpleados = captura.nextInt();

        if(nroEmpleados > 0)
        {
            LEmpleado[] lEmpleado = new LEmpleado[nroEmpleados];
            for(int i=0; i < (lEmpleado.length); i++) {

                lEmpleado[i] = new LEmpleado();

                System.out.println("Especifique el Id del empleado " + (i+1));
                lEmpleado[i].setIdEmp(captura.nextInt());
                System.out.println("Especifique el Nombre del empleado " + (i+1));
                lEmpleado[i].setNombreEmp(captura.next());
                lEmpleado[i].setSueldoEmp(1000000);

                System.out.println("El Empleado es Jefe (S/N)?");
                tipoEmpleado = captura.next();

                if (tipoEmpleado.contentEquals("S")) {
```


Polimorfismo (POO), continuación

Polimorfismo

```
LEmpleadoJefe(jempleado[i].getIdEmp(),jempleado[i].getNombreEmp(),jempleado[i].getSueldoEmp(),1998,8,28);
jempleado[i].estableceBonificacion(1000000);
jempleado[i] = jefe_sistemas;

for(LEmpleado e : jempleado ) {
    e.aumentoSueldo(3.0);
}

for(LEmpleado e : jempleado) {
    System.out.println("Empleado: " + e.getIdEmp() + " Nombre: " + e.getNombreEmp() + "
    //e.IEmpleadoRegistrar(e, 1998,8,28);
}

}
else {
    System.out.println("Canceló la captura de empleados.");
}
captura.close();
}
```

La aplicación del Polimorfismo

La aplicación se denomina Principio de Sustitución

Enlazado Dinámico

Clases Abstractas (POO)

Este Tipo de Clases nos permiten crear “método generales”, que recrean un comportamiento común, pero sin especificar cómo lo hacen.

A nivel de código tienen por particularidad que:

- Algunos de sus métodos no tienen “cuerpo de declaración”, no tienen las llaves { } ni código dentro de ellos.
- Deben estar precedidos por la palabra clave **abstract**.
- Si una clases contiene uno o más métodos abstractos está clase **debe** ser abstracta.

Estas clases como son generalidades no pueden ser instanciadas por ningún objeto (se dice que su nivel de abstracción es demasiado alto), entonces su único fin es ser **heredado/extendido** por otras clases.

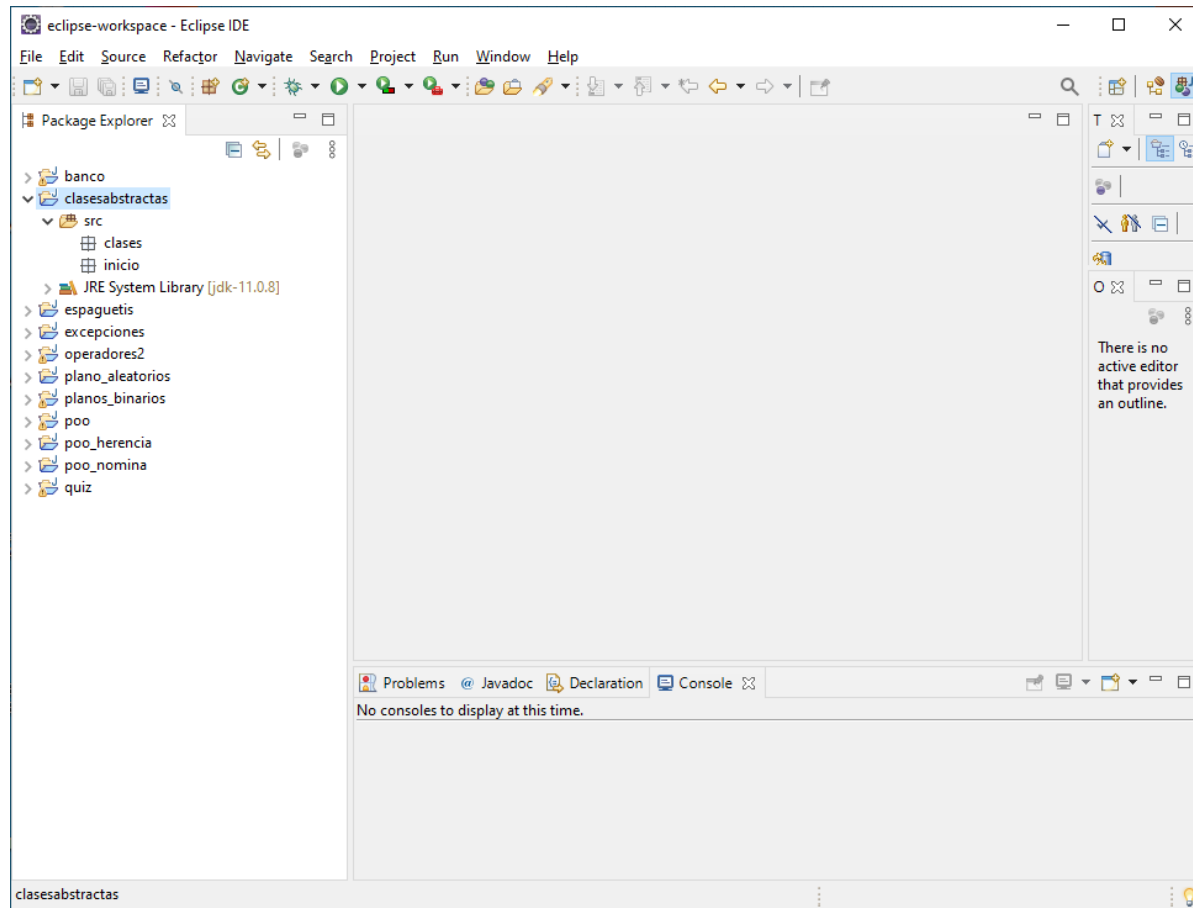
Clases Abstractas (POO), continuacion

Veamos el concepto con un ejemplo.

Crearemos un proyecto llamado `clasesabstractas`, dentro de este definiremos dos paquetes, uno desde donde se ejecutará el programa, llamado `inicio`, y otro paquete que contendrá las clase de definición de clases.

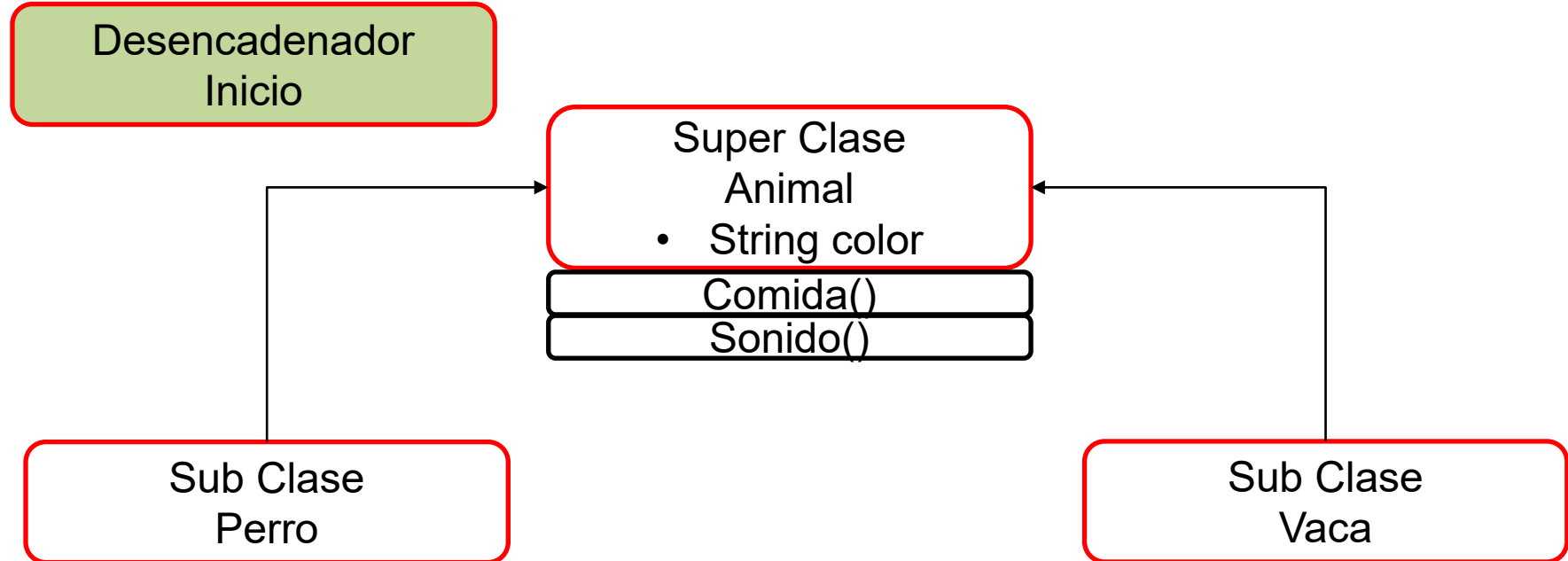
Clases Abstractas (POO), continuacion

Clases Abstractas



Clases Abstractas (POO), continuacion

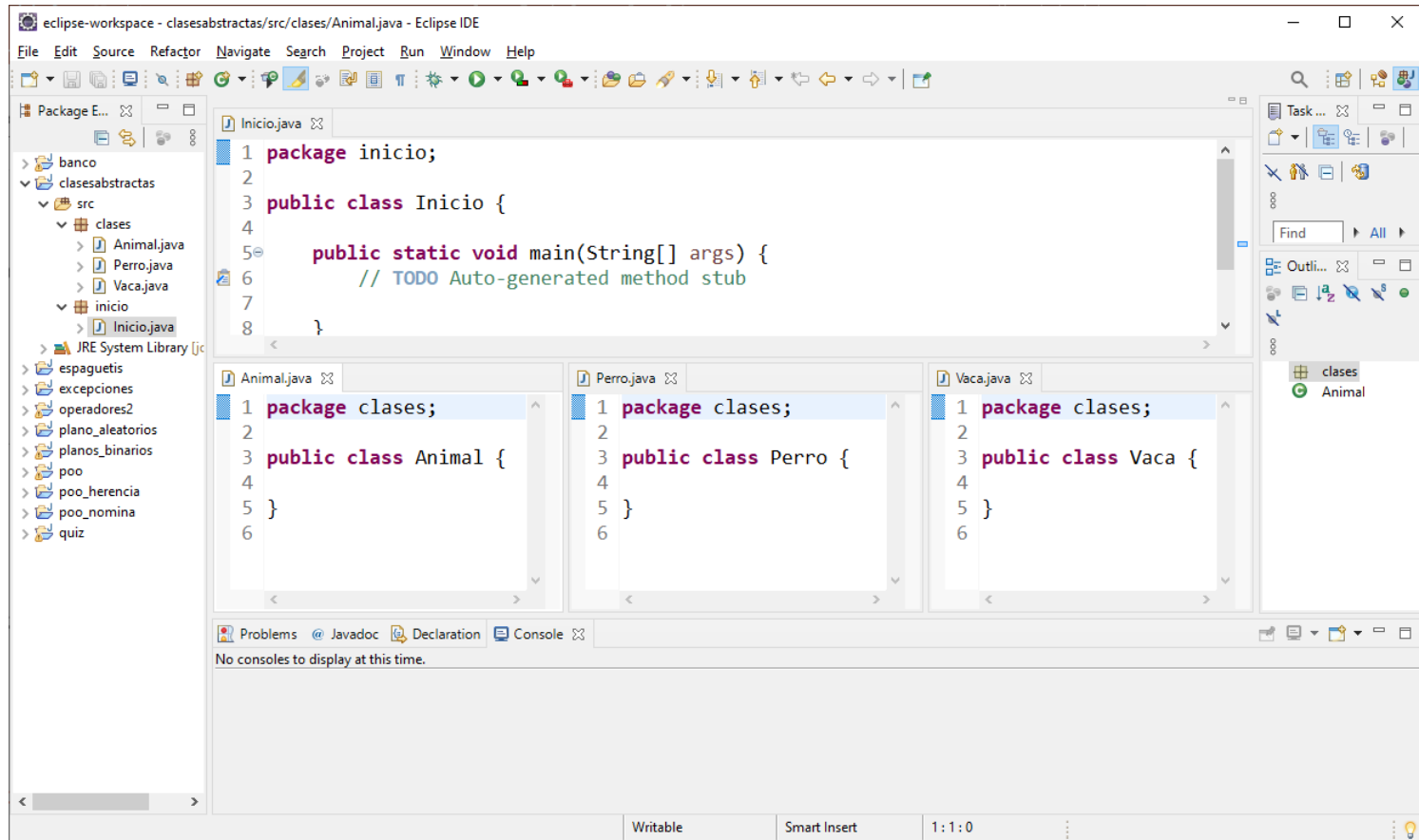
Dentro del paquete clases, crearemos tres clase:



Sin el método main

Clases Abstractas (POO), continuacion

Clases Abstractas



Clases Abstractas (POO), continuacion

Definíos las características del animal, sus constructores y métodos.

```
package clases;

public abstract class Animal {
    private String color;

    public Animal(String color) {
        this.color = color;
    }

    public abstract void comida();
    public abstract void sonido();

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }
}
```

Si uno de los métodos de la clase Padre se Define Abstracto, la clase se debe definir como abstracta.

El comportamiento de los métodos comida() y sonido(), serán definidos en las clase que **Hereden** la clase Animal

Una clase abstracta, puede tener métodos No abstractos.

Clases Abstractas (POO), continuacion

Definíos las características de la clase Perro, hereda la clase Animal.

```
package clases;
```

```
public class Perro extends Animal {
```

Heredamos las características de Animal al Perro, ya que un Perro es un Animal

```
    public Perro() {  
        super("Café");  
    }
```

```
    public void comida() {  
        System.out.println("El Perro come carne");  
    }
```

Por ser Animal una clase Abstracta, estamos Obligados a definir los métodos abstractos que tenga la clase Heredada, con el fin de Identificar cada uno de esos comportamiento específicos de cada clase Sub clase.

```
    public void sonido() {  
        System.out.println("El Perro suena WAU");  
    }  
}
```


Clases Abstractas (POO), continuacion

Definimos el funcionamiento del programa.

```
package inicio;

import clases.Perro;
import clases.Vaca;

public class Inicio {

    public static void main(String[] args) {

        Perro perro = new Perro();
        Vaca vaca = new Vaca();

        perro.comida();
        vaca.sonido();

        perro.setColor("Amarillo");

        System.out.println(perro.getColor());

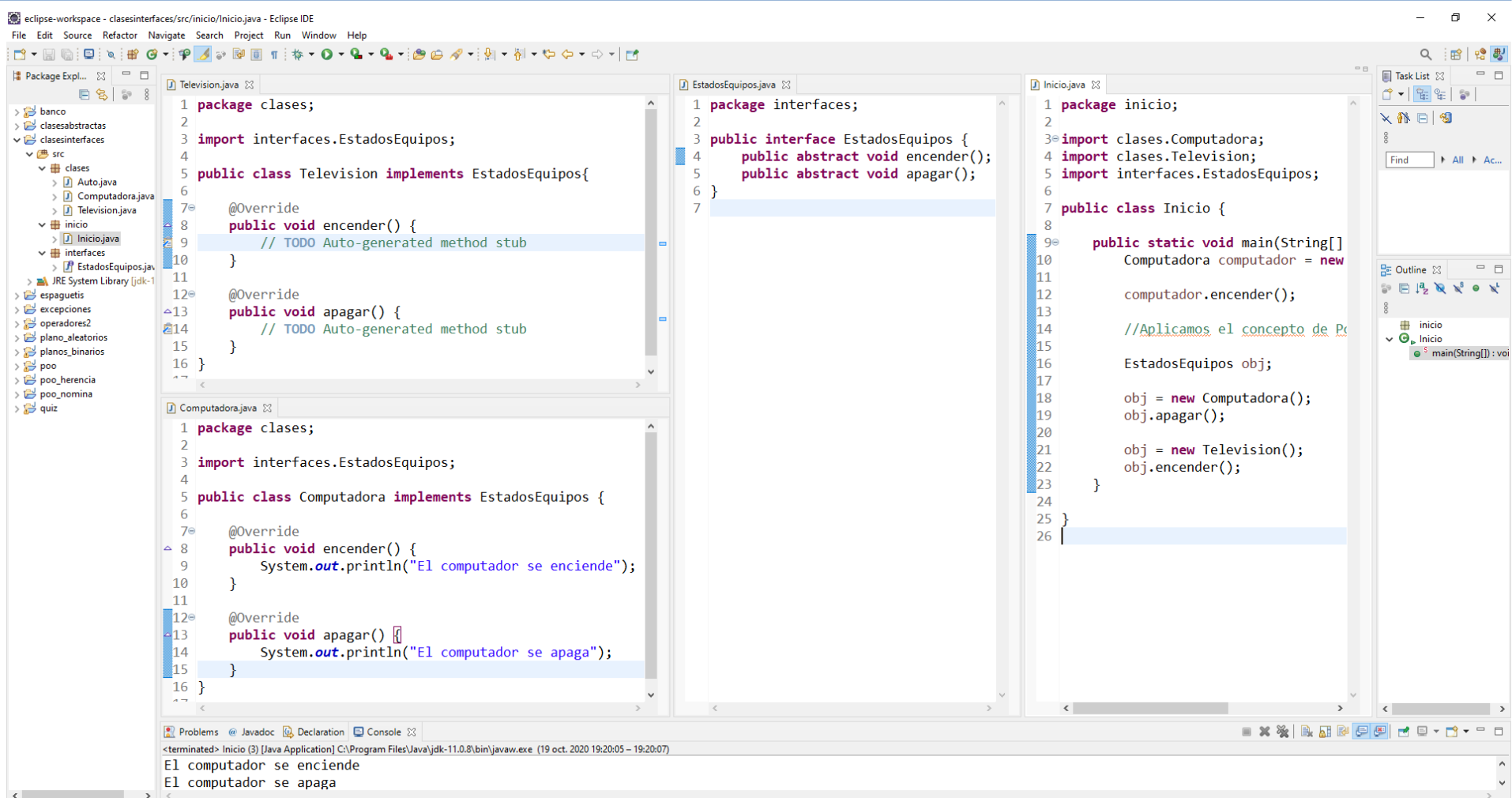
    }
}
```

Clases Interfaces (POO)

Una interfaces es un conjunto de métodos **abstractos** y de constantes cuya funcionalidad es la de determinar el funcionamiento de una clase.

Todos los métodos de una interface deben ser abstractos.

Clases Interfaces (POO)



```
1 package classes;
2
3 import interfaces.EstadosEquipos;
4
5 public class Television implements EstadosEquipos{
6
7     @Override
8     public void encender() {
9         // TODO Auto-generated method stub
10    }
11
12    @Override
13    public void apagar() {
14        // TODO Auto-generated method stub
15    }
16 }
```

```
1 package interfaces;
2
3 public interface EstadosEquipos {
4     public abstract void encender();
5     public abstract void apagar();
6 }
7
```

```
1 package inicio;
2
3 import classes.Computadora;
4 import classes.Television;
5 import interfaces.EstadosEquipos;
6
7 public class Inicio {
8
9     public static void main(String[] args) {
10         Computadora computador = new Computadora();
11
12         computador.encender();
13
14         //Aplicamos el concepto de POO
15
16         EstadosEquipos obj;
17
18         obj = new Computadora();
19         obj.apagar();
20
21         obj = new Television();
22         obj.encender();
23     }
24 }
25
26
```

```
1 package classes;
2
3 import interfaces.EstadosEquipos;
4
5 public class Computadora implements EstadosEquipos {
6
7     @Override
8     public void encender() {
9         System.out.println("El computador se enciende");
10    }
11
12    @Override
13    public void apagar() {
14        System.out.println("El computador se apaga");
15    }
16 }
```

Problems @ Javadoc Declaration Console

<terminated> Inicio (3) [Java Application] C:\Program Files\Java\jdk-11.0.8\bin\javaw.exe (19 oct. 2020 19:20:05 - 19:20:07)

El computador se enciende
El computador se apaga

Trabajo Independiente

- Desarrollar los ejercicios del Taller de modelamiento e identificar y crear los proyectos por cada uno de ellos, definiendo clases, objetos, métodos constructores, métodos modificadores de datos, construir composiciones, Encapsulamiento, Herencia, Polimorfismo, Abstracciones e Interfaces.