

PROGRAMACIÓN ORIENTADA A OBJETOS (Laboratorio de Prácticas)

Titulaciones de Grado en Ingeniería de Computadores, Ingeniería Informática y Sistemas de Información

Sesión 4 POO Básica en JAVA

En esta sesión realizaremos programas orientados a objetos básicos utilizando el lenguaje Java. Se pretende enseñar cómo realizar aplicaciones OO utilizando varias clases con relaciones de herencia y asociación.

1. POO CON JAVA

A continuación realizaremos unos ejemplos de utilización de las distintas técnicas de POO que nos proporciona Java. Con estos ejemplos aprenderemos la utilización de las interfaces, la herencia y las clases con métodos estáticos.

1. Ejemplo de definición y utilización de una interface:

```
public interface Trabajo {  
    double pagarSueldo(double horas);  
}  
  
public class Empresa1 implements Trabajo {  
    //Esta empresa paga a 12 euros la hora de trabajo  
    public double pagarSueldo(double horas) {  
        return horas*12;  
    }  
}  
  
public class Empresa2 implements Trabajo {  
    //Esta empresa paga a 15 euros la hora de trabajo  
    public double pagarSueldo(double horas) {  
        return horas*15;  
    }  
}
```

2. Construir una interface en Java que contenga los siguientes métodos:

```
public interface ModuloCalculo {  
    double suma(double a, double b);  
    double resta(double a, double b);  
    double multiplica(double a, double b);  
    double divide(double a, double b);  
}
```

3. Construir una clase en Java que implemente la interface anterior:

```
public class ModuloCalculoImpl implements ModuloCalculo {
    public double suma(double a, double b)    {
        return a+b;
    }
    public double resta(double a, double b)    {
        return a-b;
    }
    public double multiplica(double a, double b) {
        return a*b;
    }
    public double divide(double a, double b) {
        return a/b;
    }
}
```

4. Construir una clase en Java que haga uso de la clase anterior y que sume 2+3, y multiplique el resultado por 8:

```
public class PruebaMCI {
    public static void main (String args[])    {
        ModuloCalculoImpl mci = new ModuloCalculoImpl();
        System.out.println( mci.multiplica(mci.suma(2, 3), 8) );
    }
}
```

5. Crear una nueva clase a partir de la clase ModuloCalculoImpl que tenga la operación módulo (%).

```
public class ModuloCalculoImpl2 extends ModuloCalculoImpl {
    public double modulo(double a, double b) {
        return a % b;
    }
}
```

6. Construir una clase en Java que haga uso de la clase anterior y que sume 2+3, y calcule el módulo de la división por 3:

```
public class PruebaMCI2 {
    public static void main (String args[])    {
        ModuloCalculoImpl2 mci2 = new ModuloCalculoImpl2();
        System.out.println( mci2.modulo(mci2.suma(2, 3), 3) );
    }
}
```

7. Construir una clase en Java que contenga métodos estáticos:

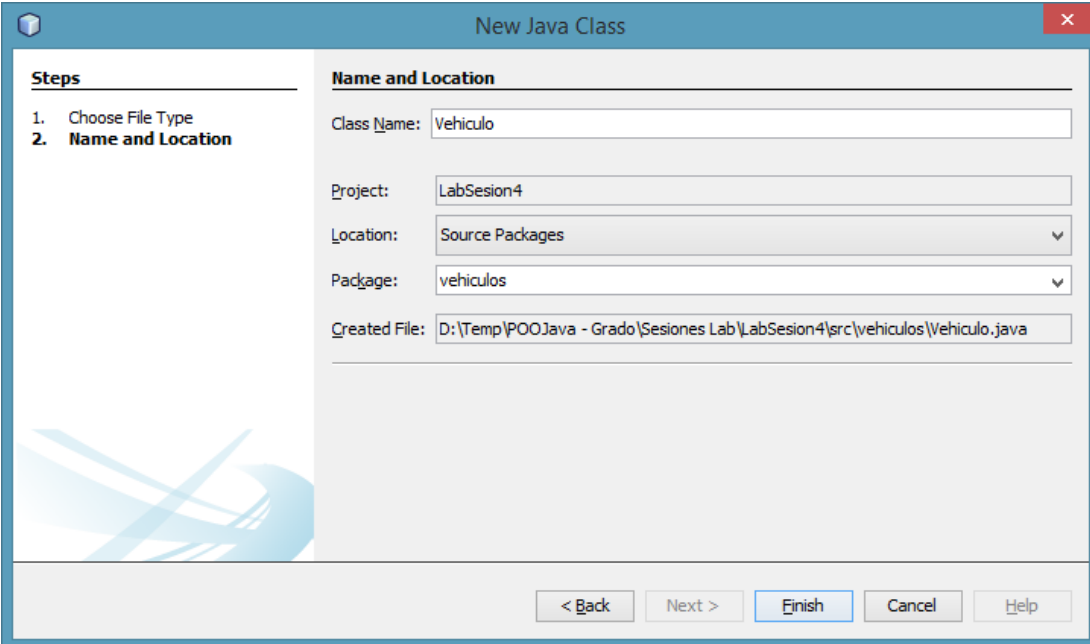
```
public class Operaciones {
    public static double suma(double a, double b)    {
        return a+b;
    }
    public static double resta(double a, double b)    {
        return a-b;
    }
    public static double multiplica(double a, double b) {
        return a*b;
    }
    public static double divide(double a, double b) {
        return a/b;
    }
    public static double modulo(double a, double b) {
        return a % b;
    }
}
```

8. Construir una clase en Java que haga uso de la clase anterior y que sume 2+3, y multiplique el resultado por 8:

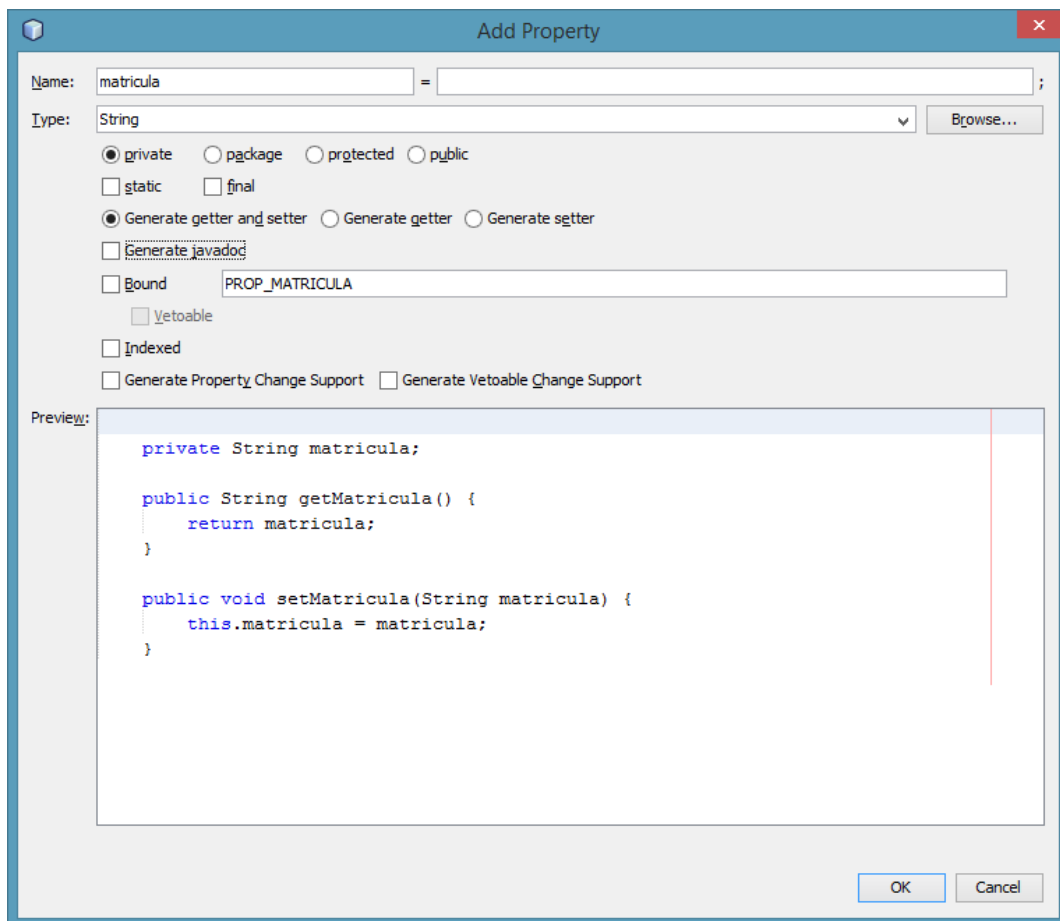
```
public class PruebaOperaciones {  
    public static void main (String args[]) {  
        System.out.println( Operaciones.multiplica(Operaciones.suma(2, 3), 8) );  
    }  
}
```

2. Ejemplo POO Vehículos

A continuación realizaremos un programa orientado a objetos para la instanciación y manejo de objetos de tipo vehículo. Dentro de un proyecto crearemos un nuevo fichero seleccionando la categoría “Java” y dentro “Java Class”, incluyendo todas las clases que creemos dentro del paquete “vehiculos”:



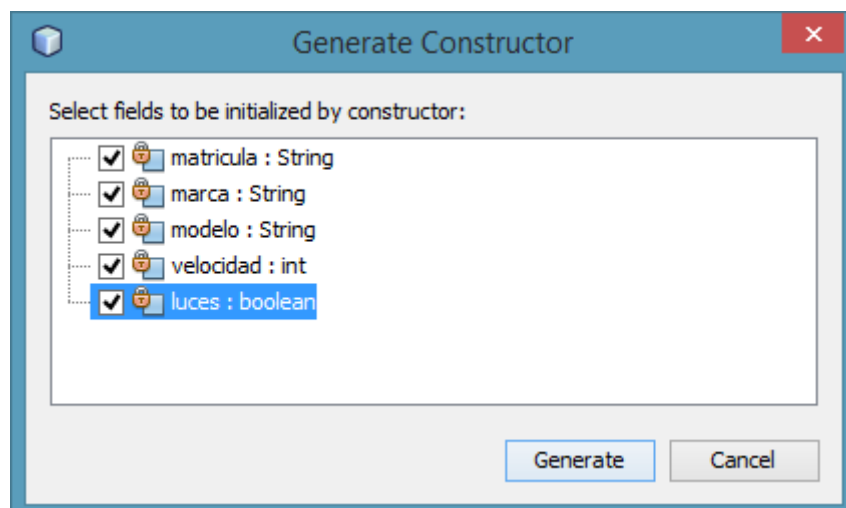
En primer lugar se crean los atributos: matrícula (String), marca (String), modelo (String), velocidad (int) y luces (boolean). Para ello situamos el cursor sobre la zona de código y seleccionamos “Insert Code” y en el menú desplegable que aparece seleccionamos “Add Property”. En el cuadro de diálogo que se muestra se indica el tipo de acceso, el tipo de atributo, el nombre del atributo (también se puede indicar si se quiere inicializar); incluso se pueden generar los “getter” y los “setter”.



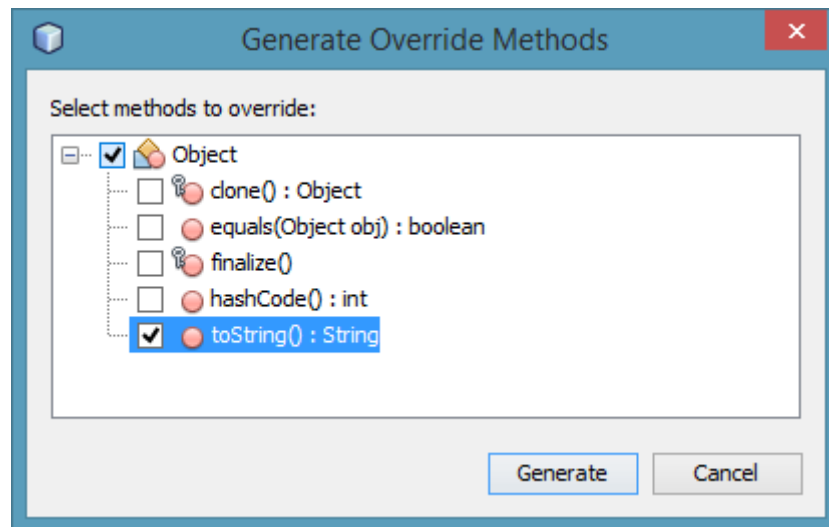
De la misma forma se procedería con los otros atributos. Los métodos get y set se han creado según se iban añadiendo atributos. En caso de no haberlos implementado a la vez que los atributos se podrían hacer de dos formas:

1. Pulsamos con el botón derecho del ratón en el código y seleccionamos “Insert Code” y pulsamos en “Getter and Setter” y seleccionamos los atributos sobre los que queremos crear los get y los set.
2. Pulsamos con el botón derecho del ratón en el código y seleccionamos “Refactor” y “Encapsulate Fields”, seleccionamos los atributos sobre los que queremos crear los get y los set y después pulsamos “Refactor”.

El siguiente paso es crear el constructor, para ello situamos el cursor sobre la zona de código y pulsamos el botón derecho del ratón y seleccionamos “Insert Code” y en el menú desplegable que aparece seleccionamos “Constructor”. En el cuadro de diálogo se marcan los atributos que vayan a ser utilizados:



El último paso es crear un método toString que proporcione información textual de los objetos de tipo vehículo, para hacerlo pulsamos con el botón derecho del ratón en el código y seleccionamos “Insert Code” y pulsamos en “Override Method” seleccionando el método toString():



El código final de la clase es el siguiente:

- Clase Vehículo:

```
package vehiculos;
public class Vehiculo {

    private String matricula;
    private String marca;
    private String modelo;
    private int velocidad;
    private boolean luces;

    public Vehiculo(String matricula, String marca, String modelo, int velocidad,
boolean luces) {
        this.matricula = matricula;
        this.marca = marca;
        this.modelo = modelo;
        this.velocidad = velocidad;
        this.luces = luces;
    }

    public String getMatricula() {
        return matricula;
    }

    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }

    public String getMarca() {
        return marca;
    }

    public void setMarca(String marca) {
        this.marca = marca;
    }

    public String getModelo() {
        return modelo;
    }

    public void setModelo(String modelo) {
```

```

        this.modelo = modelo;
    }

    public int getVelocidad() {
        return velocidad;
    }

    public void setVelocidad(int velocidad) {
        this.velocidad = velocidad;
    }

    public boolean isLuces() {
        return luces;
    }

    public void setLuces(boolean luces) {
        this.luces = luces;
    }

    @Override
    public String toString() {
        String lucestxt = "";
        if (this.luces) {
            lucestxt = "luces encendidas";
        } else {
            lucestxt = "luces apagadas";
        }
        return "Vehiculo{" + "matricula=" + matricula + ", marca=" + marca + ",
modelo=" + modelo
        + ", velocidad=" + velocidad + ", luces=" + lucestxt + '}';
    }
}

```

Para probar la clase Vehículo crearemos una clase con un método main, para hacerlo, pulsamos en el icono de nuevo fichero y seleccionamos la categoría “Java” y dentro “Java Main Class”.

- Clase PruebaVehiculos:

```

package vehiculos;

public class PruebaVehiculos {

    public static void main(String[] args) {
        // TODO code application logic here
        Vehiculo v1 = new Vehiculo("1234 BCD", "Opel", "Astra", 0, false);
        Vehiculo v2 = new Vehiculo("2345 CDE", "BMW", "S1", 80, false);
        Vehiculo v3 = new Vehiculo("3456 DEF", "Audi", "A3", 100, true);

        v1.setVelocidad(50);
        v1.setLuces(true);
        System.out.println(v1.toString());
    }
}

```

2.1 Ejercicios sobre la clase Vehículo

1. Crear un nuevo constructor que solo reciba como parámetros la matrícula, la marca y el modelo, de forma que la velocidad será 0 y las luces estarán apagadas.
2. Incorporar un nuevo atributo llamado marcha (int) que podrá tomar valores comprendidos entre el 0 y el 5 junto a los correspondientes métodos get y set.
3. Cambiar el código del método setMarcha de forma que además de establecer el valor del atributo marcha se encargue de establecer la velocidad del vehículo en 0, 10, 30, 60, 90 y 120 correspondiente a la marcha del vehículo.

4. Crear un nuevo constructor que reciba como parámetros la matrícula, la marca, el modelo y la marcha, de forma que dependiendo del valor de la marcha se establezca su velocidad (las luces estarán apagadas).
5. Modificar el método toString para que muestre también la marcha.
6. Realizar una nueva clase PruebaVehiculos2 de forma que se instancien nuevos objetos con los constructores creados en los pasos anteriores y se pruebe el método setMarcha.

3. Ejercicio Lámpara (ejercicio obligatorio para II y SI, voluntario para IC)

Crear una clase Lámpara como representación de un sistema de iluminación. Nuestra lámpara es capaz de dar distintos niveles de iluminación. Para eso crearemos la clase con dos atributos y sus correspondientes métodos get y set:

- boolean encendida: que indicará si la luz está encendida o no.
- int intensidad: que indicara el nivel de luz (de 0 a 100 %).

Podemos cambiar la intensidad de la luz mediante el método setIntensidad, indicando la cantidad de luz que queremos recibir. También nos gustaría poder encender la luz indicando el voltaje. El voltaje será un número de tipo double con valores entre 1.5 y 12.5. Para eso crearemos el método: *public void setIntensidad(double voltaje)*. Si el parámetro voltaje es inferior a 1.5, la intensidad está al 0 %. Si el valor es mayor que 12.5, la intensidad está al 100%. Los otros valores del parámetro dependen del voltaje y se pueden calcular mediante una regla de 3.

Por último queremos conocer el estado actual de nuestra luz. Para realizar esto, sobrescribiremos el método toString() para que devuelva un mensaje de tipo: Luz: ON, Intensidad: 45%

Cuando creemos un objeto de tipo Lámpara, asignaremos a los atributos los valores por defecto:

encendida = false

intensidad = 0

Como podéis ver, en este ejercicio hemos sobrecargado el método setIntensidad con diferentes tipos de parámetros.

Codificar una clase con método main para probar que la sobrecarga funciona correctamente.

4. Ejemplo de manejo de fechas

En el siguiente ejemplo podemos ver un ejemplo del manejo de fechas con la clase LocalDateTime que representa una fecha en formato completo.

```
import java.time.LocalDateTime;
import java.time.Month;
import java.time.format.DateTimeFormatter;

public class FechaJavaTime {
    public static void main(String[] args) {
        LocalDateTime hoy = LocalDateTime.now();
        System.out.println("La fecha de hoy completa es: " + hoy);
        // Cambiamos el formato
        DateTimeFormatter formatoCorto =
DateTimeFormatter.ofPattern("dd/MM/yyyy:HH:mm");
        System.out.println("Hoy es (formato corto) " + hoy.format(formatoCorto));
        //Establecemos una fecha
        LocalDateTime diaHora = LocalDateTime.of(2010, Month.FEBRUARY, 25, 10, 59, 59);
        System.out.println("La fecha establecida: " + diaHora);
        diaHora = diaHora.plusMonths(1).minusDays(5).minusHours(3);
        System.out.println("La fecha modificada: " + diaHora);
        //diferencia entre fechas
        long dias = ChronoUnit.DAYS.between(diaHora, hoy);
        System.out.println("días: " + dias);
    }
}
```

5. Ejemplo de ArrayList

En el siguiente ejemplo podemos ver un ejemplo del manejo de una estructura de datos de tipo ArrayList que permite gestionar objetos de forma dinámica.

Métodos más utilizados: add() para añadir un objeto y remove() para borrarlos. Remplazar un elemento con el método set().

Se puede buscar un elemento en concreto utilizando los métodos contains(), indexOf() o lastIndexOf().

Se puede extraer un objeto de una posición específica utilizando el método get().

```
import java.util.ArrayList;
import java.util.Collections;

public class PruebaArraylist {

    public static void main(String[] args) {
        // ArrayList de cadenas
        ArrayList<String> colores = new ArrayList<>();
        colores.add("rojo");
        colores.add("verde");
        colores.add("azul");
        colores.add("amarillo");

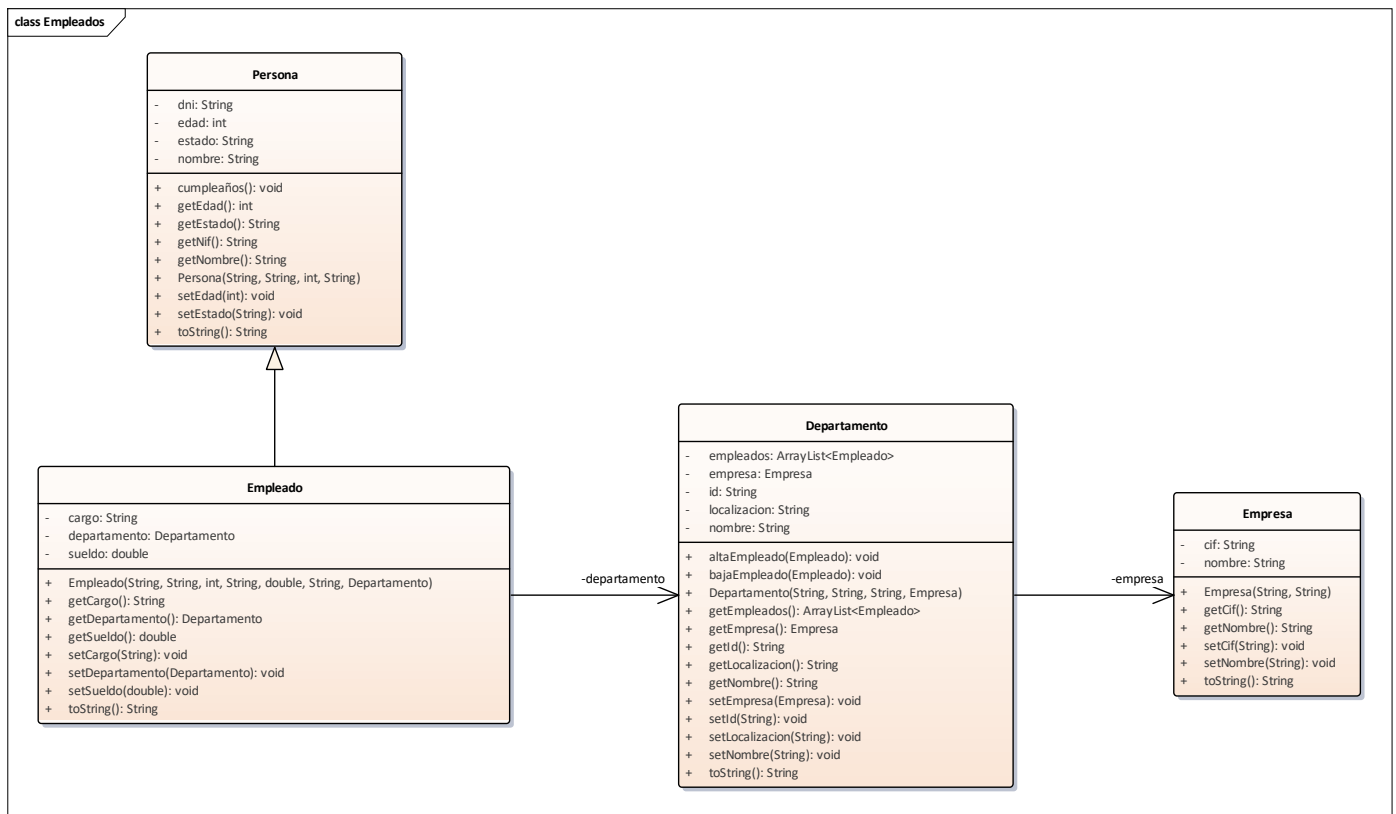
        // primer y último elemento
        System.out.println("Primer elemento: " + colores.get(0));
        System.out.println("Último elemento: " + colores.get(colores.size() - 1));

        colores.set(2, "azul claro");

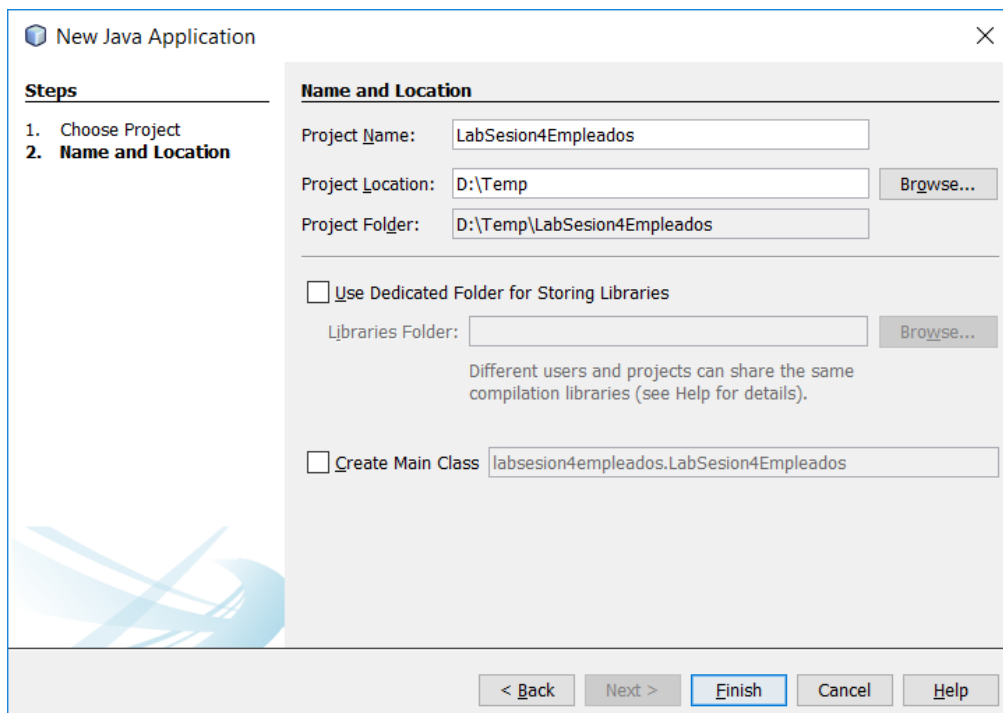
        if (colores.contains("rojo")) {
            System.out.println("\n\"rojo\" encontrado en el índice "
                               + colores.indexOf("rojo") + "\n");
        } else {
            System.out.println("\n\"rojo\" no encontrado\n");
        }
        // borrar un elemento
        colores.remove("amarillo");
        // ordenamos el array
        Collections.sort(colores);
        // imprimimos el contenido
        System.out.println(colores.toString());
        for (String c : colores) {
            System.out.println(c);
        }
    }
}
```


6. Ejemplo POO Empleados

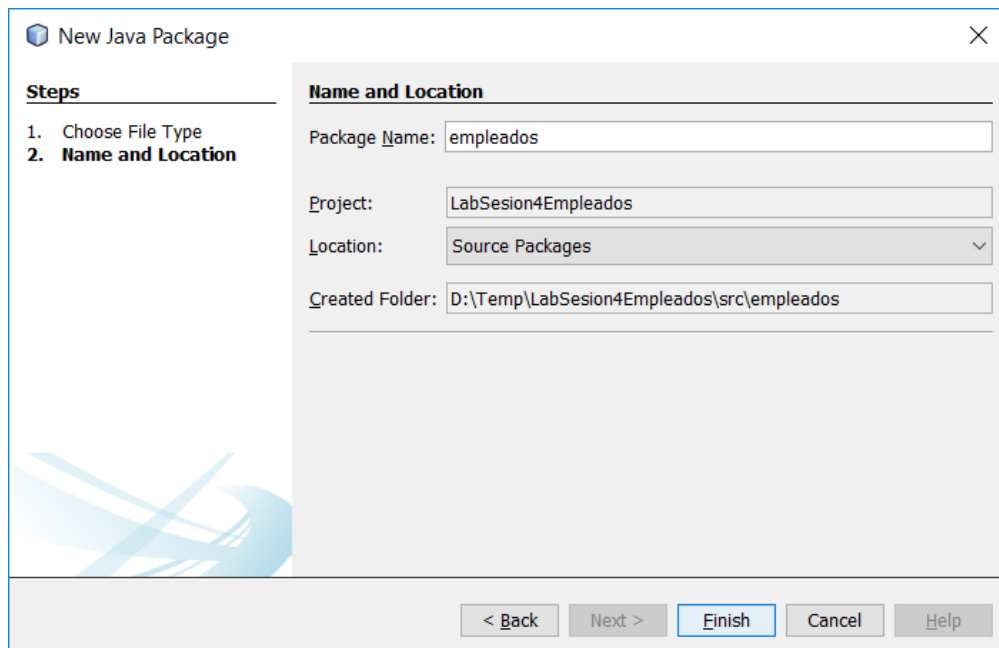
El siguiente ejemplo de POO consiste en la creación de una serie de clases que nos permiten gestionar los empleados de una empresa.



Crearemos un nuevo proyecto llamado LabSesion4Empleados:



Creamos un paquete llamado “empleados” donde meteremos las clases:



Una vez creado el paquete desarrollamos las siguientes clases: Persona, Empleado, Empresa y Departamento.

Para crear estas clases pulsamos el icono de nuevo fichero y seleccionamos la categoría “Java Classes” y dentro “Java Class”, incluyendo todas las clases dentro del paquete “empleados”.

El código de las clases es el siguiente:

- Clase Persona:

```
package empleados;

public class Persona {

    private String dni;
    private String nombre;
    private int edad;
    private String estado;

    //constructor
    public Persona(String dni, String nombre, int edad, String estado) {
        this.dni = dni;
        this.nombre = nombre;
        this.edad = edad;
        this.estado = estado;
    }
    public void cumpleaños() {
        edad++;
    }
    public String getDni() {
        return dni;
    }
    public String getNombre() {
        return nombre;
    }
    public int getEdad() {
        return edad;
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }
    public String getEstado() {
```

```

        return estado;
    }
    public void setEstado(String estado) {
        this.estado = estado;
    }
    //información textual de la persona
    @Override
    public String toString() {
        return "Persona{" + "dni=" + dni + ", nombre=" + nombre + ", edad=" + edad + ",
estado=" + estado + '}';
    }
}

```

- Clase Empleado:

```

package empleados;

public class Empleado extends Persona {
    private double sueldo;
    private String categoria;
    private Departamento departamento; //asociación

    //constructor
    public Empleado(String dni, String nombre, int edad, String estado, double sueldo,
String categoria, Departamento departamento) {
        super(dni, nombre, edad, estado);
        this.sueldo = sueldo;
        this.categoria = categoria;
        this.departamento = departamento;
        this.departamento.altaEmpleado(this); //tenemos que incorporar el empleado al
departamento
    }
    public Departamento getDepartamento() {
        return departamento;
    }
    public void setDepartamento(Departamento departamento) {
        this.departamento = departamento;
    }
    public String getCategoria() {
        return categoria;
    }
    public void setCategoria(String categoria) {
        this.categoria = categoria;
    }
    public double getSueldo() {
        return sueldo;
    }
    public void setSueldo(double sueldo) {
        this.sueldo = sueldo;
    }
    //información textual del empleado compuesta por la información de la persona +
empleado + departamento
    @Override
    public String toString() {
        return super.toString() + " # Empleado{" + "sueldo=" + sueldo + ", categoria="
+ categoria + ", departamento=" + departamento + '}';
    }
}

```

- Clase Departamento:

```
package empleados;

import java.util.ArrayList;
public class Departamento {
    private String nombre;
    private String id;
    private String localizacion;
    private Empresa empresa; //asociación
    private ArrayList<Empleado> empleados; //estructura de datos dinámica para
    almacenar a los empleados del departamento

    //constructor
    public Departamento(String nombre, String id, String localizacion, Empresa empresa)
    {
        this.nombre = nombre;
        this.id = id;
        this.localizacion = localizacion;
        this.empresa = empresa;
        empleados = new ArrayList<>(); //array vacío para ir añadiendo los empleados
    }
    public ArrayList<Empleado> getEmpleados() {
        return empleados;
    }

    //alta de un empleado en el departamento
    public void altaEmpleado(Empleado emp) {
        if (!empleados.contains(emp)) {
            empleados.add(emp);
            emp.setDepartamento(this); //el empleado tiene que reflejar el alta
        }
    }
    //baja de un empleado del departamento
    public void bajaEmpleado(Empleado emp) {
        if (empleados.contains(emp)) {
            empleados.remove(emp);
            emp.setDepartamento(null); //el empleado tiene que reflejar la baja
        }
    }

    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public Empresa getEmpresa() {
        return empresa;
    }
    public void setEmpresa(Empresa empresa) {
        this.empresa = empresa;
    }
    public String getLocalizacion() {
        return localizacion;
    }
    public void setLocalizacion(String localizacion) {
        this.localizacion = localizacion;
    }
    //información textual del departamento
    public String toString() {
```

```

        return "Departamento{" + "nombre=" + nombre + ", id=" + id + ", localizacion="
+ localizacion + '}';
    }
}

```

- Clase Empresa:

```

package empleados;

public class Empresa {
    private String nombre;
    private String cif;

    //constructor
    public Empresa(String nombre, String cif) {
        this.nombre = nombre;
        this.cif = cif;
    }
    public String getCif() {
        return cif;
    }
    public void setCif(String cif) {
        this.cif = cif;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    //información textual de la empresa
    @Override
    public String toString() {
        return "Empresa{" + "nombre=" + nombre + ", cif=" + cif + '}';
    }
}

```

Para probar las clases vamos a realizar una clase “Main” llamada “PruebaEmpleados” como sigue:

```

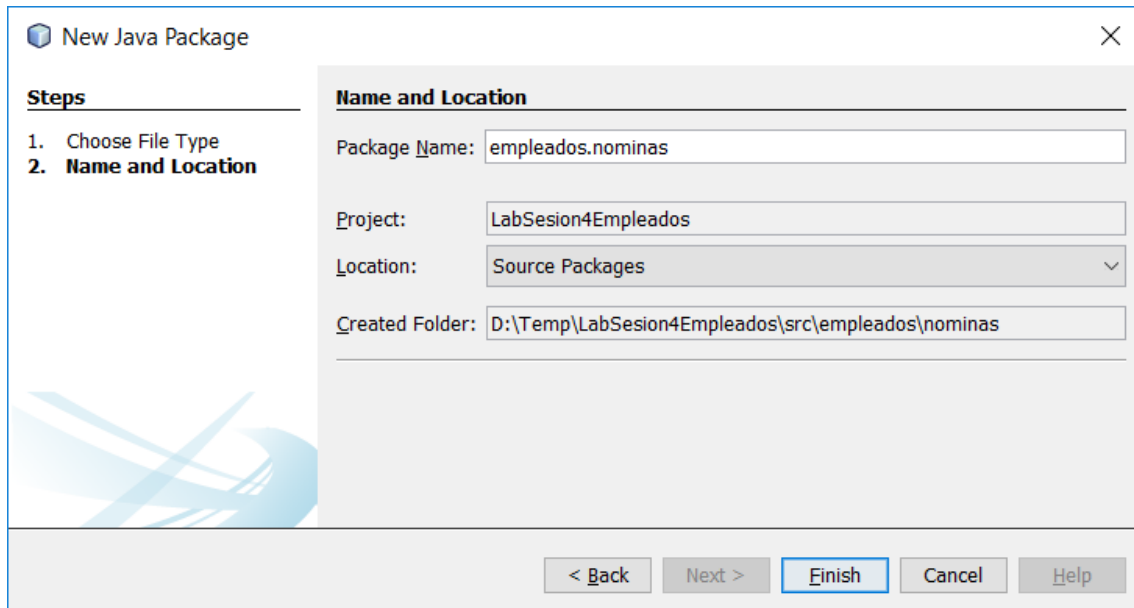
package empleados;

public class Main {
    public static void main(String[] args) {
        //1 - creamos la empresa
        Empresa e1 = new Empresa("Indra", "1234567");
        //2 - creamos los departamentos
        Departamento d1 = new Departamento("Informática", "1", "Madrid", e1);
        Departamento d2 = new Departamento("Personal", "2", "Barcelona", e1);
        //3 - creamos los empleados que asignamos a los departamentos
        Empleado emp1 = new Empleado("12345678Z", "Pepe", 25, "soltero", 1500,
"programador", d1);
        Empleado emp2 = new Empleado("45673419T", "Laura", 35, "casada", 2000,
"analista", d1);
        Empleado emp3 = new Empleado("56782345F", "Marta", 40, "casada", 2500,
"gerente", d2);
        //modificamos los datos de los empleados
        emp1.cumpleaños();
        emp1.setSueldo(2000);
        System.out.println(emp1.toString());
        emp2.setCategoria("jefe proyecto");
        System.out.println(emp2.toString());
        //imprimimos los elementos del ArrayList
        System.out.println("Empleados departamento: " + d1.getEmpleados().toString());
    }
}

```

6.1 Ejercicio Nóminas de empleados

Se pretende modificar el proyecto anterior para que incluya la gestión de las nóminas de los empleados. Para realizarlo hay que crear un nuevo subpaquete llamado “nominas”:



Dentro desarrollaremos la clase Nómina que se encargará de almacenar las nóminas de los empleados. Esta clase debe tener los siguientes atributos (tipo – nombre) con sus correspondientes métodos get y set (exceptuando el salarioNeto que solo tendrá método get):

1. Empresa – empresa.
2. Departamento – departamento.
3. Empleado – empleado.
4. LocalDate – fecha.
5. double – salarioBruto.
6. double – retencion.
7. double – salarioNeto.

Realizar un constructor que reciba los siguientes parámetros: empresa, departamento, empleado y retención (representando un %). La fecha se extrae de la fecha actual del sistema. El salarioBruto se extrae del empleado mediante el método getSuelto(). En la instanciación de los objetos nómina se tendrá que realizar el cálculo del atributo salarioNeto mediante la resta al salario bruto de la retención. Crear también el método toString para que imprima la información de una nómina (se pueden utilizar los métodos toString de las clases asociadas para ir componiendo la cadena final).

Ya que a partir de la clase empleado se puede deducir la información del departamento y la empresa, realizar un nuevo constructor que solo reciba el empleado y la retención y extraer los otros datos del empleado.

Una vez realizada la clase Nómina crear una clase con un método main para probar la creación de varias nóminas de los empleados.

7. Ejercicio Taller

Se desea realizar una aplicación que simule un taller donde se reparan vehículos. El taller debe permitir el cálculo de la reparación de un vehículo en base al coste de las piezas utilizadas y las horas dedicadas a la reparación. La aplicación debe tener tres clases: Taller, Vehículo y Pieza:

- La clase Pieza representa una pieza del vehículo reparado y contendrá su nombre (String) y su precio (double).
- El Vehículo contendrá la matrícula (String), marca (String), modelo (String) y un ArrayList con las piezas que se han arreglado.
- La clase Taller tendrá un nombre (String), teléfono (String) y el precio hora (double). El taller debe tener un método repararVehiculo() que recibirá dos parámetros: un objeto de tipo vehículo y las horas dedicadas a la reparación; a partir del objeto vehículo sabremos las piezas utilizadas y su precio y con el coste de la hora debe ser capaz de devolver el importe total de la reparación.

Crear las clases y los métodos necesarios para poder implementar el método repararVehiculo() y una clase PruebaTaller con método main para poder probarlas.

8. Ejercicio Aparcamiento (ejercicio obligatorio para II y SI, voluntario para IC)

Se pretende realizar una aplicación en Java que simule un aparcamiento donde se pueden estacionar vehículos por un tiempo limitado. Existen dos tipos de vehículos que pueden estacionar en el aparcamiento: automóviles y camiones. De todos los vehículos hay que guardar su matrícula, la fecha/hora de entrada en el aparcamiento y si se trata de un vehículo con abono, de los automóviles el tipo: turismo, todoterreno y furgoneta, y de los camiones el número de ejes.

La clase Vehículo tendrá un método abstracto llamado calcularImporte() que tendrán que implementar las clases hijas (polimorfismo) y que será el encargado de determinar los minutos que han transcurrido y mediante la siguiente fórmula calculará el importe del tiempo de aparcamiento:

- Automóvil:
 - turismo $\rightarrow \text{minutos} * 1.5 \text{ €} / 60$
 - todoterreno $\rightarrow \text{minutos} * 2.5 \text{ €} / 60$
 - furgoneta $\rightarrow \text{minutos} * 3.5 \text{ €} / 60$
- Camión:
 - Ejes $\leq 3 \rightarrow \text{minutos} * 4.5 \text{ €} / 60$
 - Ejes $> 3 \rightarrow \text{minutos} * 6.5 \text{ €} / 60$

Si se trata de un vehículo con abono se le aplicará un descuento del 40% sobre el importe final.

Para facilitar la prueba del importe del estacionamiento se tienen que crear dos constructores en la clase Vehículo, uno donde no se pasa la fecha de entrada en el aparcamiento y por lo tanto se extrae del sistema y otra en la que se le pasa una fecha concreta. Estos constructores serán implementados en las clases hijas.

Hay que desarrollar una clase Aparcamiento que se encargará de almacenar los vehículos que tiene estacionados, para hacerlo usará un ArrayList de vehículos. El aparcamiento tiene un número de plazas determinada que se establece a través de un atributo denominado capacidad (int). La clase contará con los siguientes métodos:

- `introducirVehiculo(Vehiculo v)`: Se encargará de añadir un nuevo vehículo al aparcamiento comprobando que no está dentro y que hay capacidad suficiente. También decrementará la capacidad del aparcamiento.
- `sacarVehiculo(Vehiculo v) / sacarVehiculo(String matricula)`: Se encargará de sacar un vehículo del aparcamiento comprobando que está dentro y calculará cuanto tiene que cobrar haciendo uso del método `calcularImporte()` del vehículo correspondiente. También incrementará la capacidad del aparcamiento. Se realizarán dos versiones de este método utilizando sobrecarga, de esta forma se podrá sacar el vehículo con el objeto vehículo o mediante su matrícula.

Por último, para poder probar las clases desarrolladas crear una clase con método `main` para poder instanciar un conjunto de vehículos, introducirlos en el aparcamiento y sacarlos para ver los importes de estacionamiento.