

PROGRAMACIÓN ORIENTADA A OBJETOS (PARTE I)

PRÁCTICA II



PROGRAMACIÓN ORIENTADA A OBJETOS (PARTE I) – PRÁCTICA II

GUIA DEL DOCUMENTO

Este documento consta del siguiente contenido:

- Instrucciones previas.
- Ejercicios de **herencia** (3 actividades).
- Ejercicios de **sobrecarga** y **sobrescritura** de método (3 actividades).
- Ejercicios de **Polimorfismo** (2 actividades).
- Importación en Eclipse de las soluciones a las actividades.

NOTA: Estos ejercicios **reutilizan clases** que se han realizado en las prácticas de temas anteriores. Si no has realizado estos ejercicios puedes utilizar la implementación de las clases que aparecen en el solucionario de este tema.

PROGRAMACIÓN ORIENTADA A OBJETOS (PARTE I) – PRÁCTICA II

INSTRUCCIONES PREVIAS

Ya tienes algunas nociones de Java, así que vamos a ponerlas en práctica. A continuación, te presentamos varias **actividades, alguna de ellas guiadas, y sus soluciones correspondientes.**

Para realizar tú mismo estas actividades, primero debes seguir los siguientes pasos:

1. **Instala el software JDK** (Java Development Kit) que provee herramientas de desarrollo para la creación de programas en Java. Podrás encontrar el link a la descarga en la sección inferior **contenido descargable**, con el nombre **"Descarga Java Development Kit"**.
2. **Instala Eclipse**, el editor para programar Java, que encontrarás también en la sección inferior **contenido descargable**, con el nombre **"Descarga de Eclipse"**.

Las **instrucciones para la instalación** de ambos la puedes encontrar igualmente en la sección inferior **contenido descargable** con el nombre **"Manual de Instalación de Eclipse"**.

3. Puedes seguir el documento **primeros pasos en Eclipse**, que encontrarás también en la sección inferior **contenido descargable**, con el nombre **"Primeros pasos en Eclipse"**, que te ayudará a crear el área de trabajo para desarrollar tus prácticas.

Soluciones a las actividades

Te proporcionamos las soluciones a las actividades en un fichero .zip, que encontrarás también en la sección inferior **contenido descargable**, con el nombre **"Solución de las prácticas"**. Si no dispones de ningún software de comprensión/descompresión de datos en tu ordenador, **puedes descomprimir las soluciones** utilizando el **software WinRAR**, que encontrarás también en la sección inferior **contenido descargable** con el nombre **"Descarga WinRAR"**. Para importar las soluciones a Eclipse puedes ayudarte de las instrucciones que te proporcionamos en el último apartado de este documento.

Ten en cuenta que algunas de las instrucciones que se proporcionan en las soluciones puedes no haberlas visto en el curso teórico, pero tienes los conocimientos suficientes para investigar y buscar esta u otras soluciones.

Recuerda que en el mundo de la informática, el cuál evoluciona tan rápidamente, es una práctica muy común tener que investigar y autoformarte, a través de diversas fuentes, para ampliar tus conocimientos.

PROGRAMACIÓN ORIENTADA A OBJETOS (PARTE I) – PRÁCTICA II

EJERCICIOS DE HERENCIA

Actividad 1. Herencia

Vamos a hacer una aplicación o **programa que gestione los empleados de una empresa**.

Para este ejercicio, vamos a **reutilizar la clase Persona** creada en las prácticas del tema anterior. Si no lo has realizado puedes utilizar la clase Persona del solucionario que te proporcionamos.

En primer lugar, vamos a crear una **clase** llamada **Empleado**, que será nuestra **clase hija**, ya que **hereda** de nuestra **clase Persona**.

Esta clase tendrá los siguientes **atributos privados**:

- **antigüedad** de tipo int
- **salario** de tipo double
- **puesto** de tipo String

Se deben crear los siguientes **métodos**:

- **Constructor vacío**
- **Constructor de copia**
- **Constructor de parámetros**
- **Métodos setters/getters** para asignar y obtener los datos de los atributos.
- **Crear el método toString** para que nos muestre la información de los atributos de la clase padre Persona y de la clase Empleado.
- **Método actualizarSalario(double incremento)**: Consiste en aumentar el salario actual del empleado con el valor que se pasa como parámetro. **Este método recibe un parámetro de entrada con la cantidad a sumar y no retorna nada.**

En segundo lugar, vamos a crear una nueva **clase** llamada **Empresa** con un método main para crear la instancia de la clase Empleado y poder probar los métodos creados.

En esta clase, dentro del método **main**, vamos a realizar lo siguiente:

1. Creamos una instancia de tipo Empleado usando el constructor vacío y le asignamos valores a cada uno de sus atributos.

Datos para la nueva instancia: "Rebeca Velasco González", "11.111.111-H", 30, 20, "Jefe", 1800.00.
2. Mostraremos la información de la instancia creada.
3. Realiza una actualización del salario del empleado creado con un incremento de 150.0€ usando el método actualizarSalario.
4. Volvemos a mostrar la información del empleado para comprobar que se ha modificado su salario.

Pasos a seguir:

1. Dentro del proyecto “**FormacionJava**” dentro del paquete “**ClasesObjetos**”, vamos a crear la **clase Empleado** que extiende de la **clase Persona**.

2. Dentro de la **clase Empleado**, lo primero que vamos a hacer, es crear los distintos atributos.

Nota: Acuérdate de usar el modificador de acceso correspondiente, private int antigüedad;

3. A continuación, vamos a crear los **tres constructores**.

Nota: Acuérdate de usar el modificador de acceso correspondiente para los constructores, public. `public Empleado() { }`

Además, en cada constructor debes llamar al constructor de la clase padre con la palabra super, por ejemplo: `super();`

4. Lo siguiente que vamos a hacer, es crear los **métodos setter y getter** para los distintos atributos.

Nota: Acuérdate que estos métodos se pueden generar de forma automática, para ello, con el botón derecho del ratón sobre la parte del código, seleccionar la opción “Source > Generate setters and getters....”.

5. Ahora, vamos a crear o **sobrescribir el método toString** para que nos muestre la información, tanto la información de los atributos de la parte del padre (`super.toString()`) más la información de los propios atributos de la clase.

Nota: Acuérdate que este método se pueden generar de forma automática, para ello, con el botón derecho del ratón sobre la parte del código, seleccionar la opción “Source > Generate toString()....”

6. Ahora, vamos a crear el **método que actualiza el salario** del empleado.

Nota: Acuérdate de usar el modificador de acceso correspondiente para este método, public, y que no tendrá retorno, con lo que tenemos que usar void.

```
public void actualizarSalario(double incremento)
```

7. De nuevo, dentro del proyecto “**FormacionJava**” dentro del **paquete “ClasesObjetos”**, vamos a crear la **clase Empresa** con el **método main()**

8. Dentro de la **clase Empresa**, lo primero que vamos a hacer, es crear la instancia con el constructor vacío.

```
Empleado empleado = new Empleado();
```

9. A continuación, vamos a **dar valores a los atributos de la instancia creada anteriormente**.

Nota: Acuérdate que debes usar los métodos setters para modificar los atributos.

```
empleado.setNombre("Rebeca");
empleado.setPrimerApellido("Velasco");
empleado.setSegundoApellido("Gonzalez");
empleado.setDni("12345678L");
empleado.setEdad(30);
empleado.setAntiguedad(20);
empleado.setPuesto("Jefe");
empleado.setSalario(1800.0);
```

10. A continuación, **mostramos la información del empleado creado**.

Nota: Acuérdate que para mostrar la información podemos usar el `System.out.println` con la instancia creada una vez que hemos sobrescrito el método `toString()`.

```
System.out.println(empleado);
```

11. A continuación, **actualizamos el salario del empleado** con un incremento de 150.00€.

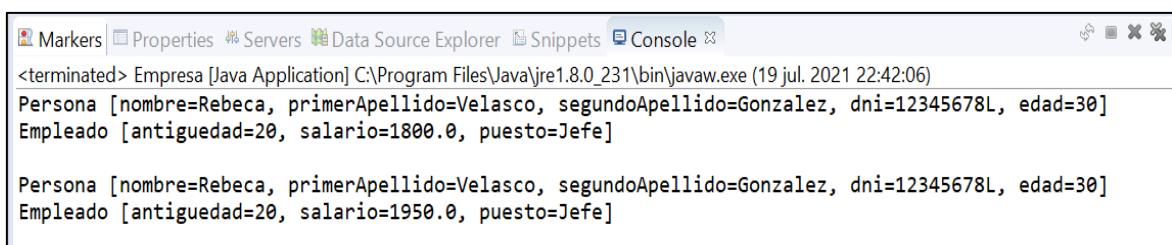
Nota: Acuérdate que este método no retorna un valor, así que puedes llamarle directamente.

12. Por último, **mostramos la información del empleado** de nuevo para comprobar su nuevo salario.

Nota: Acuérdate que para mostrar la información podemos usar el `System.out.println` con la instancia creada una vez que hemos sobrescrito el método `toString()`.

```
System.out.println(empleado);
```

Resultado esperado:



```
<terminated> Empresa [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (19 jul. 2021 22:42:06)
Persona [nombre=Rebeca, primerApellido=Velasco, segundoApellido=Gonzalez, dni=12345678L, edad=30]
Empleado [antiguedad=20, salario=1800.0, puesto=Jefe]

Persona [nombre=Rebeca, primerApellido=Velasco, segundoApellido=Gonzalez, dni=12345678L, edad=30]
Empleado [antiguedad=20, salario=1950.0, puesto=Jefe]
```

Actividad 2. Herencia

Vamos a hacer una aplicación o programa que **gestione el precio de una serie de productos de un comercio.**

Para ello vamos a crear una clase **Producto** que represente un producto que podemos tener en un comercio. Además, como tenemos dos tipos de productos, perecedero o no perecedero, vamos a **crear dos clases, *Perecedero* y *NoPerecedero*, ambas van a heredar de la clase *Producto*.**

La **clase *Producto*** tendrá los siguientes **atributos privados**:

- **nombre** de tipo String
- **precio** de tipo double

La **clase *Perecedero*** tendrá el siguiente **atributo privado**:

- **diasACaducar** de tipo int

La **clase *NoPerecedero*** tendrá el siguiente **atributo privado**:

- **tipo** de tipo String

Para las tres clases, se deben crear los siguientes **métodos**:

- **Constructor vacío**
- **Constructor de copia**
- **Constructor de parámetros**
- **Métodos setters/getters** para asignar y obtener los datos de los atributos.
- **Sobrescribir el método *toString* para cada clase** (piensa como aprovechar la herencia para este método en el caso de las clases *Perecedero* y *NoPerecedero*) para que muestre los datos del producto de la siguiente forma:

```
"Datos del producto"
"-----"
"Nombre: " nombre
"Precio: " precio
"Dias a caducar: " diasACaducar (sólo en el caso de la clase
Perecedero)
"Tipo: " tipo (sólo en el caso de la clase NoPerecedero)
```

Para la clase *Producto* además vamos a crear:

- **Método *calcularImporteTotal(double cantidad)***: Consiste en calcular el importe total de un número de productos que le pasemos, es decir, multiplicar el precio del producto por la cantidad de productos pasados. **Este método recibe un parámetro de entrada con la cantidad de ese producto y retorna el importe total para ese número de productos.**

Para la clase **Perecedero** vamos a sobrescribir el **método calcularImporteTotal** que hemos heredado de la **clase Producto**:

- **Método calcularImporteTotal(double cantidad):** Consiste en calcular el importe total de un número de productos que le pasemos, es decir, multiplicar el precio del producto por la cantidad de productos pasados, pero además el precio se reducirá según los días a caducar, de tal forma que:
 - Si le queda 1 día para caducar, se reducirá un 50% el precio final
 - Si le quedan 2 días para caducar, se reducirá un 25% el precio final
 - Si le quedan más de 2 días para caducar, se reducirá un 10% el precio final

Este método recibe un parámetro de entrada con la cantidad de ese producto y retorna el importe total para ese número de productos.

Para la clase **NoPerecedero**, vamos a usar el **método calcularImporteTotal** que hemos heredado de la **clase Producto**, con lo que no deberíamos implementar o sobrescribir este método.

Por último, una vez creadas las tres clases anteriores, vamos a crear una **nueva clase** llamada **Tienda** con un **método main** para poder crear las distintas instancias de las **clases Perecedero** y **NoPerecedero** y así poder probar los métodos creados.

En esta clase, dentro del **método main**, vamos a realizar lo siguiente:

1. Vamos a crearnos un array de tipo Producto de 5 posiciones para guardar cinco instancias de tipo Perecedero o NoPerecedero que vamos a crear a continuación.
2. Creamos una instancia de tipo Perecedero usando el constructor vacío y le asignamos valores a cada uno de sus atributos.

Datos para la nueva instancia: Nombre del producto será "Yogurt", el precio es 0.50, y los días a caducar será 1.

Esta instancia la vamos a guardar en la primera posición del array creado anteriormente.
3. Creamos una nueva instancia de tipo Perecedero usando el constructor de parámetros.

Datos para la nueva instancia: ("Leche", 1.20, 2).

Esta instancia la vamos a guardar en la segunda posición del array creado anteriormente.
4. Creamos una nueva instancia de tipo Perecedero usando el constructor de parámetros.

Datos para la nueva instancia: ("Queso", 5.70, 25).

Esta instancia la vamos a guardar en la tercera posición del array creado anteriormente.

5. Creamos una nueva instancia de tipo NoPerecedero usando el constructor de parámetros.
Datos para la nueva instancia: ("Esponja", 0.90, "Limpieza & hogar").
Esta instancia la vamos a guardar en la cuarta posición del array creado anteriormente
6. Creamos una nueva instancia de tipo NoPerecedero usando el constructor de parámetros.
Datos para la nueva instancia: ("Jabón", 1.90, "Limpieza & hogar").
Esta instancia la vamos a guardar en la quinta y última posición del array creado anteriormente.
7. Ahora vamos a calcular el precio para todos los productos del array usando el método calcularImporteTotal, donde la cantidad de productos será 2. Y además, iremos mostrando la información del producto junto el precio total calculado.
8. Por último, debemos mostrar el precio total de todos los productos, es decir, la suma de todos los precios calculados anteriormente.

Pasos a seguir:

1. Dentro del proyecto “**FormacionJava**”, lo primero que vamos a hacer es crear dentro de la carpeta “**src**” un nuevo **paquete**, donde vamos a agrupar todas las clases para este ejercicio, al que denominaremos “**EjerciciosHerencia**”.
2. Dentro del paquete “**EjerciciosHerencia**” crearemos la **clase Producto**.
3. Dentro de la **clase Producto**, lo primero que vamos a hacer, es crear los distintos atributos.

Nota: Acuérdate de usar el modificador de acceso correspondiente, private String nombre;

4. A continuación, vamos a crear los **tres constructores**.

Nota: Acuérdate de usar el modificador de acceso correspondiente para los constructores, public. public Producto() { }

5. Lo siguiente que vamos a hacer, es **crear los métodos setter y getter** para los distintos atributos.

Nota: Acuérdate que estos métodos se pueden generar de forma automática, para ello, con el botón derecho del ratón sobre la parte del código, seleccionar la opción “Source > Generate setters and getters....”

6. Ahora, vamos a crear o **sobrescribir el método toString** para que nos muestre la información del objeto como nosotros queremos.

Nota: Acuérdate que este método se pueden generar de forma automática, para ello, con el botón derecho del ratón sobre la parte del código, seleccionar la opción “Source > Generate toString()....”

7. Por último, vamos a **crear el método calcularImporteTotal(double cantidad)** que calcula el precio del producto según la cantidad pasada.

Nota: Acuérdate de usar el modificador de acceso correspondiente para este método (public) y que tendrá el retorno de tipo double.

```
public double calcularImporteTotal(double cantidad)
```

8. De nuevo, dentro del proyecto “**FormacionJava**” dentro del paquete “**EjerciciosHerencia**”, vamos a crear la **clase Perecedero** que **extiende de la clase Producto**.

9. Dentro de la **clase Perecedero**, lo primero que vamos a hacer, es crear el nuevo atributo.

Nota: Acuérdate de usar el modificador de acceso correspondiente, `private int diasACaducar;`

10. A continuación, vamos a crear los **tres constructores**.

Nota: Acuérdate de usar el modificador de acceso correspondiente para los constructores.

```
public Perecedero() { }
```

11. Lo siguiente que vamos a hacer, es crear los **métodos setter y getter** para el atributo.

Nota: Acuérdate que estos métodos se pueden generar de forma automática, para ello, con el botón derecho del ratón sobre la parte del código, seleccionar la opción "Source > Generate setters and getters...."

12. Ahora, vamos a crear o sobrescribir el **método toString** para que nos muestre la información del objeto como nosotros queremos.

Nota: Acuérdate que este método se pueden generar de forma automática, para ello, con el botón derecho del ratón sobre la parte del código, seleccionar la opción "Source > Generate toString()...."

13. Por último, vamos a sobrescribir el **método calcularImporteTotal(double cantidad)** que hereda de la clase `Producto`, que calcula el precio del producto según la cantidad pasada y aplicará el descuento correspondiente según los días a caducar.

14. De nuevo, dentro del proyecto "**FormacionJava**" dentro del paquete "**EjerciciosHerencia**", vamos a crear la **clase NoPerecedero** que **extiende de la clase Producto**.

15. Dentro de la **clase NoPerecedero**, lo primero que vamos a hacer, es crear el nuevo atributo.

Nota: Acuérdate de usar el modificador de acceso correspondiente, `private int diasACaducar;`

16. A continuación, vamos a crear los **tres constructores**.

Nota: Acuérdate de usar el modificador de acceso correspondiente para los constructores.

```
public NoPerecedero() { }
```

17. Lo siguiente que vamos a hacer, es crear los **métodos setter y getter** para el atributo.

Nota: Acuérdate que estos métodos se pueden generar de forma automática, para ello, con el botón derecho del ratón sobre la parte del código, seleccionar la opción "Source > Generate setters and getters...."

18. Ahora, vamos a crear o **sobrescribir el método toString** para que nos muestre la información del objeto como nosotros queremos.

Nota: Acuérdate que este método se pueden generar de forma automática, para ello, con el botón derecho del ratón sobre la parte del código, seleccionar la opción "Source > Generate toString()...."

19. De nuevo, dentro del proyecto "**FormacionJava**" dentro del paquete "**EjerciciosHerencia**", vamos a crear la **clase Tienda** con el **método main()**.
20. Dentro de la **clase Tienda**, lo primero que vamos a hacer, es **crear el array de tipo Producto de 5 posiciones**.

```
Producto[] listaProductos = new Producto[5];
```

21. Ahora nos creamos las **cinco instancias de tipo Perecedero y NoPerecedero**.

```
Perecedero yogurt = new Perecedero();
yogurt.setNombre("Yogurt");
yogurt.setPrecio(0.50);
yogurt.setDiasACaducar(1);
Perecedero leche = new Perecedero("Leche", 1.20, 2);
Perecedero queso = new Perecedero("Queso", 5.70, 25);

NoPerecedero esponja = new NoPerecedero("Esponja", 0.90, "Limpieza & hogar");
NoPerecedero jabon = new NoPerecedero("Jabon", 1.90, "Limpieza & hogar");
```

22. Guardamos las **instancias creadas en el array**.

```
listaProductos[0] = yogurt;
listaProductos[1] = leche;
listaProductos[2] = queso;
listaProductos[3] = esponja;
listaProductos[4] = jabon;
```

23. Creamos una **variable para ir sumando el precio total de todos los productos**.

```
double precioTotal = 0;
```

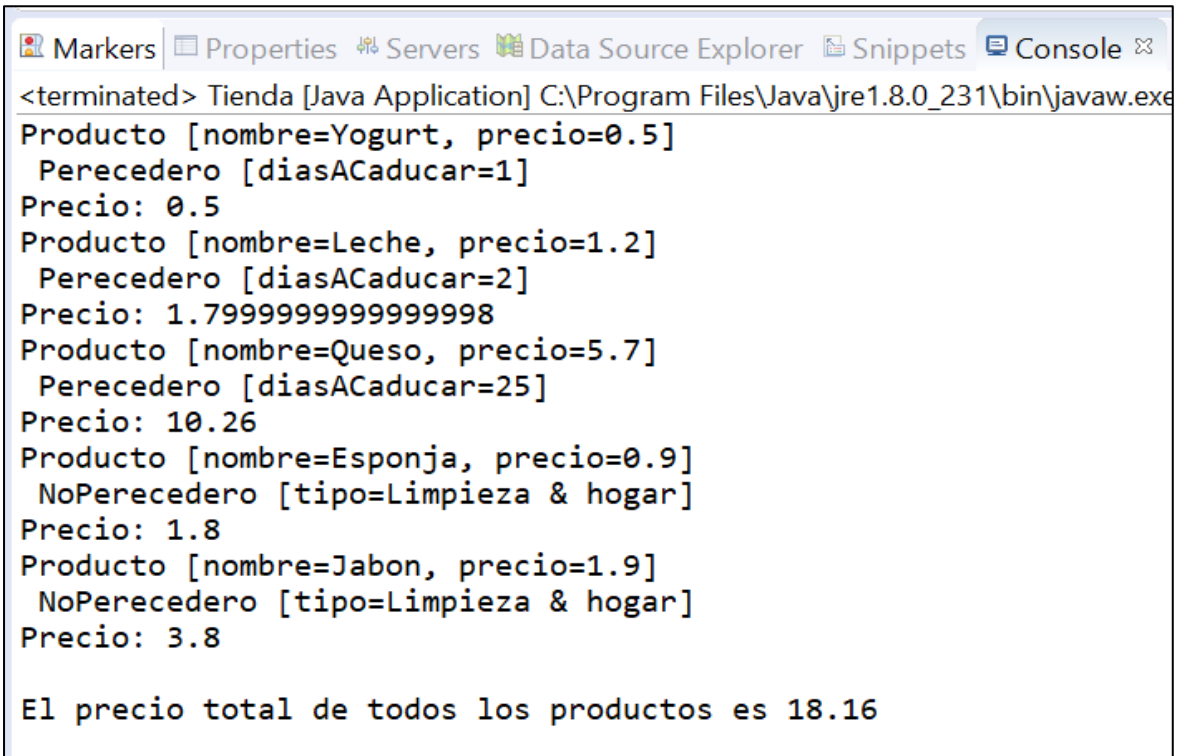
24. Ahora vamos a **calcular el precio para todos los productos del array usando el método calcularImporteTotal**, para ello podemos usar un **forEach** para recorrer el array.

```
for (Producto producto : listaProductos) {
    System.out.println(producto);
    System.out.println("Precio: " + producto.calcularImporteTotal(2));
    precioTotal = precioTotal + producto.calcularImporteTotal(2);
}
```

25. Por último, debemos **mostrar el precio total de todos los productos**, es decir, la suma de todos los precios calculados anteriormente.

```
System.out.println("El precio total de todos los productos es " +  
precioTotal);
```

Resultado esperado:



```
<terminated> Tienda [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe  
Producto [nombre=Yogurt, precio=0.5]  
  Perecedero [diasACaducar=1]  
Precio: 0.5  
Producto [nombre=Leche, precio=1.2]  
  Perecedero [diasACaducar=2]  
Precio: 1.7999999999999998  
Producto [nombre=Queso, precio=5.7]  
  Perecedero [diasACaducar=25]  
Precio: 10.26  
Producto [nombre=Esponja, precio=0.9]  
  NoPerecedero [tipo=Limpieza & hogar]  
Precio: 1.8  
Producto [nombre=Jabon, precio=1.9]  
  NoPerecedero [tipo=Limpieza & hogar]  
Precio: 3.8  
  
El precio total de todos los productos es 18.16
```

Actividad 3. Herencia

Vamos a hacer una aplicación o **programa que gestione el alquiler de películas y series**.

Para ello vamos a crear una **clase Ejemplar** que represente un elemento que podemos alquilar. Además, como tenemos dos tipos, series o películas, vamos a crear dos **clases**, **Serie** y **Pelicula**, ambas van a heredar de la **clase Ejemplar**.

La **clase Ejemplar** tendrá los siguientes **atributos privados**:

- **titulo** de tipo String
- **genero** de tipo String
- **disponible** de tipo Boolean

La **clase Serie** tendrá los siguientes **atributos privados**:

- **numTemporadas** de tipo Int
- **tipo** de tipo String

La **clase Pelicula** tendrá el **atributo privado**:

- **director** de tipo String

Para las tres clases, se deben crear los siguientes **métodos**:

- **Constructor vacío**
- **Constructor de copia**
- **Constructor de parámetros**
- **Métodos setters/getters** para asignar y obtener los datos de los atributos.
- **Sobrescribir el método toString para cada clase** (piensa como aprovechar la herencia para este método en el caso de las clases Serie y Pelicula) para que muestre los datos de la siguiente forma:

```
"Datos del alquiler"
"-----"
"Titulo: " titulo
"Genero: " genero
"Tipo: " tipo (sólo en el caso de la clase Serie)
"Nº de temporadas: " numTemporadas (sólo en el caso de la clase Serie)
"Director: " director (sólo en el caso de la clase Pelicula)
"Disponible: " disponible
```

Para la clase Ejemplar además vamos a crear:

- **Método entregar():** Consiste en entregar una serie o película, es decir, sólo debe modificar el atributo disponible con el valor false y mostrar un mensaje si se ha podido realizar la operación o no, teniendo en cuenta que no se podrán alquilar series o películas que no estén disponibles. **Este método no recibe un parámetro de entrada ni retorna nada.**

- **Método devolver():** Consiste en devolver una serie o película, es decir, sólo debe modificar el atributo disponible con el valor true y mostrar un mensaje si se ha podido realizar la operación o no, teniendo en cuenta que no se podrán devolver series o películas que no estén alquiladas. **Este método no recibe un parámetro de entrada ni retorna nada**

Para la clase Serie y Pelicula, vamos a usar los métodos entregar y devolver que hemos heredado de la clase Ejemplar, con lo que no deberíamos implementar o sobrescribir dichos métodos.

Por último, una vez creadas las tres clases anteriores, vamos a **crear** una nueva **clase** llamada **Alquiler** con un método main para poder crear las distintas instancias de las clases **Serie** y **Pelicula** y así poder probar los métodos creados.

En esta clase, dentro del **método main**, vamos a realizar lo siguiente:

1. Vamos a crearnos un array de tipo Ejemplar de 8 posiciones para guardar 8 instancias de tipo Serie o Pelicula que vamos a crear a continuación.
2. Creamos cuatro instancias de tipo Serie usando el constructor de parámetros y se las asignamos o guardamos en el array creado anteriormente. Datos a utilizar:

```
listaEjemplares[0] = new Serie("Juego de tronos", "George R. R. Martin",
true, 8, "Aventuras");
listaEjemplares[1] = new Serie("Los Simpsons", "Matt Groening", false,
25, "Humor");
listaEjemplares[2] = new Serie("Padre de familia", "Seth MacFarlane",
true, 12, "Humor");
listaEjemplares[3] = new Serie("Breaking Bad", "Vince Gilligan", false,
5, "Thriller");
```

3. Creamos cuatro instancias de tipo Pelicula usando el constructor de parámetros y se las asignamos o guardamos en el array creado anteriormente. Datos a utilizar:

```
listaEjemplares[4] = new Pelicula("Memorias de una Geisha", "Drama",
true, "Rob Marshall");
listaEjemplares[5] = new Pelicula("La milla verde", "Drama/Fantástico",
false, "Frank Darabont");
listaEjemplares[6] = new Pelicula("Hermanos por pelotas", "Comedia",
true, "Adam McKay");
listaEjemplares[7] = new Pelicula("El resplandor", "Terror", false,
"Stanley Kubrick");
```

4. Realizar la operación de préstamo sobre uno de los ejemplares, para ello:
 - Mostramos un mensaje en consola solicitando el título de la serie o película a prestar/entregar. *Nota:* Recuerda que puedes usar el objeto Scanner para interactuar con la consola.
 - Si la serie o película se encuentra en nuestro array, procedemos a realizar el préstamo,
 - si la operación se ha realizado correctamente, mostramos un mensaje como: "Se ha prestado " + <título> y la información del alquiler del elemento prestado.
 - pero si no se ha podido prestar, mostramos un mensaje como: "No está disponible + <título> + " para prestar"
 - Si no tenemos dicho ejemplar, mostramos un mensaje "No tenemos ese ejemplar para prestar".
5. Realizar la operación de devolución sobre uno de los ejemplares, para ello:
 - Mostramos un mensaje en consola solicitando el título de la serie o película a devolver.
 - Si la serie o película se encuentra en nuestro array, procedemos a realizar la devolución,
 - si la operación se ha realizado correctamente, mostramos un mensaje como: "Se ha devuelto " + <título> y la información del alquiler del elemento devuelto.
 - pero si no se ha podido devolver, mostramos un mensaje como: "No estaba alquilada + <título> + " para devolver"
 - Si no tenemos dicho ejemplar, mostramos un mensaje "No tenemos ese ejemplar para devolver".
6. Posteriormente, mostrar un listado con el título de las películas y series que se encuentran disponibles.
7. Por último, mostrar un listado con el título de las películas y series que se encuentran alquiladas.

Notas:

- Podemos crearnos los métodos que creamos oportunos dentro de la clase Alquiler para simplificar el código.
- Recuerda que Java es un lenguaje que diferencia entre mayúsculas y minúsculas, para que sean válidos los títulos de los ejemplares escritos tanto en mayúsculas como en minúsculas podemos utilizar el método `equalsIgnoreCase` ya que compara dos cadenas ignorando si está escrito con mayúsculas o minúsculas.

Resultado esperado:

Serie/Película a prestar: "Padre de familia"

Serie/Película a devolver: "La milla verde"

```
Markers Properties Servers Data Source Explorer Snippets Console
<terminated> Alquiler [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe
Por favor, introduzca el título de la serie o pelicula a prestar
padre de familia
Se ha prestado Padre de familia
Datos del alquiler
-----
TituloPadre de familia
Genero=Seth MacFarlane
Nº de temporadas:12
Tipo:Humor
Disponible: false

Por favor, introduzca el título de la novela a devolver
la milla verde
Se ha devuelto La milla verde
Datos del alquiler
-----
TituloLa milla verde
Genero=Drama/Fantástico
Director:Frank Darabont
Disponible: true

Listado de peliculas disponibles
-----
Juego de tronos
Memorias de una Geisha
La milla verde
Hermanos por pelotas

Listado de peliculas alquiladas
-----
Los Simpsons
Padre de familia
Breaking Bad
El resplandor
```

PROGRAMACIÓN ORIENTADA A OBJETOS (PARTE I) – PRÁCTICA II

EJERCICIOS DE SOBRECARGA Y SOBRESCRITURA

Actividad 1. Sobrecarga y Sobrescritura

Vamos a hacer una aplicación o **programa que gestione los empleados de una empresa.**

Para este ejercicio, vamos a **reutilizar la clase Empleado** creada en el paquete **"ClasesObjetos"** que extiende a su vez de la **clase Persona** que se encuentra en el mismo paquete.

En primer lugar, vamos a crear una nueva **clase** llamada **Comercial** que hereda de nuestra **clase Empleado**.

La **clase Comercial** tendrá el **atributo privado**:

- **comision** de tipo double

Para las tres clases, se deben crear los siguientes **métodos**:

- **Constructor vacío**
- **Constructor de copia**
- **Constructor de parámetros**
- **Métodos setters/getters** para asignar y obtener los datos de los atributos.
- **Método toString()** para que muestre toda la información, tanto de las clases padre como de la nueva clase.

Para la **clase Comercial** además vamos a **crear dos métodos** llamados **calcularComision**, es decir, vamos a **sobrecargar** el método para hacer dos implementaciones distintas de la siguiente forma:

- **1º Implementación del Método calcularComision(double incremento):** Este método se encargará de modificar el atributo comision incrementándolo con el incremento que se le pase como parámetro. **Este método recibe un parámetro de entrada con la cantidad a incrementar a la comision y no tendrá retorno.**
- **2º Implementación del Método calcularComision(double incremento, double plus):** Este método se encargará de modificar el atributo comision incrementándolo con el incremento que se le pase como parámetro más el plus de convenio pasado como parámetro. **Este método recibe dos parámetros de entrada con la cantidad y el plus a incrementar a la comisión y no tendrá retorno.**

Además, para la **clase Comercial** además vamos a **sobrescribir el método actualizarSalario** que ha heredado de la **clase Empleado** de la siguiente forma:

- **Método actualizarSalario(double incremento):** Este método se encargará de incrementar el atributo salario con el incremento que pasamos como parámetro pero además debemos sumarle la comisión correspondiente, para ello debemos calcularla con alguno de los métodos anteriores (calcularComision) en función de lo siguiente:
 - Si la antigüedad es ≤ 5 años \rightarrow La comisión se incrementará en 25.0
 - Si la antigüedad es > 5 años \rightarrow La comisión se incrementará en un 25.0 y un plus de convenio de 100.0 más.

Este método recibe un parámetro de entrada con la cantidad a incrementar al salario y no tendrá retorno.

Por último, una vez creada la clase anterior, vamos a **crear una nueva clase llamada AppEmpresa** con un **método main** para poder crear las distintas instancias de la **clase Comercial** y así poder probar los métodos creados.

En esta clase, dentro del **método main**, vamos a realizar lo siguiente:

1. Creamos varias instancias de tipo Comercial.
Datos: Comercial jefeZona = new Comercial("Lucas", "Guerrero", "Arjona", "12345678A", 42, 12, 2400.00, "Jefe de zona", 100.00);
Comercial comercial = new Comercial("Francisco", "Perez", "Antón", "11111111H", 24, 2, 1200.00, "Comercial", 50.00);
2. Una vez creadas las instancias para los dos empleados, vamos a actualizar el salario de ambos con un incremento de 200.00€, para ello usaremos el método actualizarSalario.
3. Mostrar por consola los datos de ambos empleados para comprobar el nuevo salario para ambos empleados.

Pasos a seguir:

1. Dentro del proyecto "**FormacionJava**" en el **paquete "ClasesObjetos"** vamos a crear la nueva **clase Comercial que extienda de la clase Empleado** que tenemos en este mismo paquete.
2. Dentro de la **clase Comercial**, lo primero que vamos a hacer, es **crear el nuevo atributo**.

```
private double comision
```

3. A continuación, vamos a crear los **tres constructores**.

Nota: Acuérdate de usar el modificador de acceso correspondiente para los constructores. `public Comercial() { }`

4. Lo siguiente que vamos a hacer, es **crear los métodos setter y getter** para los distintos atributos.

Nota: Acuérdate que estos métodos se pueden generar de forma automática, para ello, con el botón derecho del ratón sobre la parte del código, seleccionar la opción "Source > Generate setters and getters...."

5. Ahora, vamos a crear o **sobrescribir el método toString** para que nos muestre la información del objeto como nosotros queremos.

Nota: Acuérdate que este método se pueden generar de forma automática, para ello, con el botón derecho del ratón sobre la parte del código, seleccionar la opción "Source > Generate toString()...."

```
public String toString() {  
    return super.toString() + "\nComercial [comision=" + comision + "];  
}
```

6. A continuación, vamos a **crear el método calcularComision(double incremento)** que se encargará de modificar el atributo comisión incrementándolo con el incremento que se le pase como parámetro.

Nota: Acuérdate de usar el modificador de acceso correspondiente para este método (public) y que no tendrá el retorno.

```
public void calcularComision (double incremento)
```

7. A continuación, vamos a **crear el método calcularComision(double incremento, double plus)** que se encargará de modificar el atributo comision incrementándolo con el incremento que se le pase como parámetro más el plus de convenio pasado como parámetro.

Nota: Acuérdate de usar el modificador de acceso correspondiente para este método (public) y que no tendrá el retorno.

```
public void calcularComision (double incremento, double plus)
```

8. Por último, vamos a **sobrescribir el método actualizarSalario(double incremento)** que hereda de la **clase Empleado**, que se encargará de incrementar el atributo **salario con el incremento que pasamos como parámetro** pero **además debemos sumarle la comisión correspondiente**, para ello debemos calcularla con alguno de los métodos anteriores (calcularComision) en función de lo siguiente:
- Si la antigüedad es ≤ 5 años \rightarrow La comisión se incrementará en 25.0
 - Si la antigüedad es > 5 años \rightarrow La comisión se incrementará en un 25.0 y un plus de convenio de 100.0 más.
9. De nuevo, dentro del proyecto "**FormacionJava**" dentro del paquete "**ClasesObjetos**", vamos a crear la **clase AppEmpresa** con el **método main()**.
- Nota: Acuérdate que estos métodos se pueden generar de forma automática, para ello, con el botón derecho del ratón sobre la parte del código, seleccionar la opción "Source > Generate setters and getters...."*
10. Dentro de la **clase AppEmpresa**, lo primero que vamos a hacer es crear las instancias para los dos empleados de tipo Comercial.

```
Comercial jefeZona = new Comercial("Lucas", "Guerrero",  
"Arjona","12345678A", 42, 12, 2400.00, "Jefe de zona", 100.00);  
Comercial comercial = new Comercial("Francisco", "Perez",  
"Antón","11111111H ", 24, 2, 1200.00, "Comercial",50.00);
```

11. Ahora vamos a **actualizar el salario a los dos empleados creados anteriormente**.

```
jefeZona.actualizarSalario(200.00);  
comercial.actualizarSalario(200.00);
```

12. Por último, debemos **mostrar la información de ambos empleados** para comprobar su nuevo salario.

Resultado esperado:

```
Markers Properties Servers Data Source Explorer Snippets Console ×  
<terminated> AppEmpresa [Java Application] C:\Program Files\Java\jdk-17.0.2\bin\javaw.exe (10 mar 2022 12:26:28 – 12:26:30)  
Persona [nombre=Lucas, primerApellido=Guerrero, segundoApellido=Arjona, dni=12345678A, edad=42]  
Empleado [antigüedad=12, salario=2825.0, puesto=Jefe de zona]  
Comercial [comision=225.0]  
  
Persona [nombre=Francisco, primerApellido=Perez, segundoApellido=Antón, dni=11111111H , edad=24]  
Empleado [antigüedad=2, salario=1475.0, puesto=Comercial]  
Comercial [comision=75.0]
```

Actividad 2. Sobrecarga

Vamos a hacer una aplicación o **programa que realice la conversión de diferentes tipos a int.**

Para ello vamos a crear una **clase Conversion** con un método `main()` y con los siguientes métodos llamados **convertirAEntero**, es decir, vamos a sobrecargar el método para hacer varias implementaciones distintas de la siguiente forma:

- **1º** Implementación del **método convertirAEntero(double numero)**: Este método se encargará de convertir a entero el valor recibido y retornarlo. **Este método recibe un parámetro de entrada con el número a convertir y retorna el número convertido.**
- **2º** Implementación del **método calcularComision(char numero)**: Este método se encargará de convertir a entero el valor recibido y retornarlo. **Este método recibe un parámetro de entrada con el carácter a convertir y retorna el número correspondiente.**

Ahora, dentro del **método main**, vamos a realizar lo siguiente para probar los métodos anteriores:

1. Vamos a crearnos varias variables de diferentes tipos:
 - a. `double numDouble = 2145.56;`
 - b. `char caracter = 'H';`
2. Por último, vamos a mostrar la conversión de cada una de estas variables a entero usando los métodos creados anteriormente

Pasos a seguir:

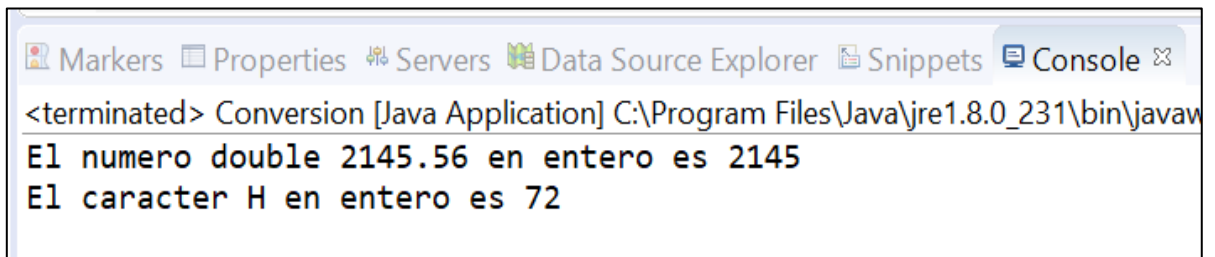
1. Dentro del proyecto "**FormacionJava**" creamos un **paquete** nuevo llamado "**EjerciciosSobrecargaSobrescritura**" vamos a crear la nueva **clase Conversion** con método **main()**.
2. A continuación, vamos a **crear el método convertirAEntero(double numero)** que se encargará de convertir el valor double a entero y retornarlo.
3. A continuación, vamos a **crear el método calcularComision(char numero)** que se encargará de convertir el valor char a entero y retornarlo.
4. El siguiente paso, dentro del método **main**, vamos a crearnos las **distintas variables con los valores** correspondientes:

```
double numDouble = 2145.56;  
char caracter = 'H';
```

5. Por último, **mostramos por consola la conversión de esas variables** usando los métodos anteriores.

```
System.out.println("El numero double " + numDouble + " en entero es " +  
convertirAEntero(numDouble));
```

Resultado esperado:



Actividad 3. Sobrescritura

Vamos a hacer una aplicación o **programa que realice la implementación de los diferentes sonidos de los animales.**

Para este ejercicio, vamos a crear una **clase** nueva llamada **Animal** en el **paquete "EjerciciosSobrecargaSobrescritura"**. Además, en este mismo paquete vamos a crear otras **tres clases** llamadas **Perro, Gato y Vaca** que **extienda de la clase Animal**.

La **clase Animal** tendrá los siguientes **atributos privados**:

- **tipo** de tipo String
- **nPatas** de tipo int
- **sonido** de tipo String

Y se deben crear los siguientes **métodos**:

- **Constructor vacío**
- **Métodos setters/getters** para asignar y obtener los datos de los atributos.

Para la **clase Animal** además vamos a crear el **método emitirSonido** de la siguiente forma:

- **Método emitirSonido ()**: Este método simplemente mostrará por consola el sonido que hace el animal, en este clase simplemente muestra el mensaje "Sonido del animal". **Este método no recibe parámetros ni tiene retorno.**

Además, para la **clase Perro, Gato y Vaca** además vamos a **sobrescribir** el **método emitirSonido** que **han heredado de la clase Animal** de la siguiente forma:

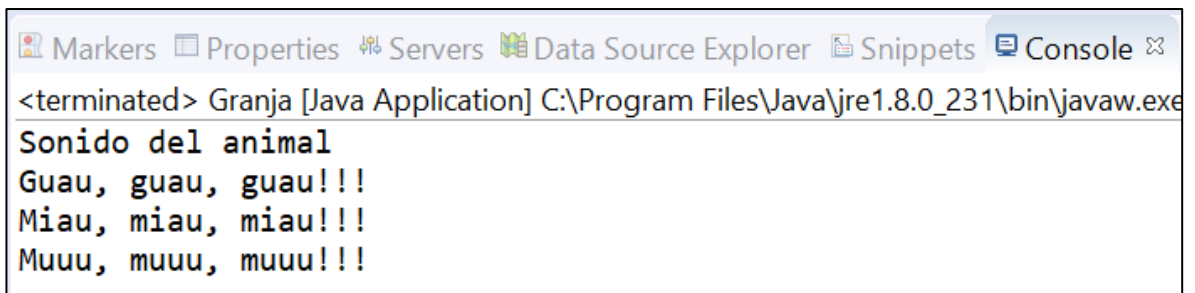
- Para la **clase Perro: método emitirSonido()**: Este método simplemente mostrará por consola el sonido que hace el animal, en este clase simplemente muestra el mensaje "Guau, guau, guau!!". **Este método no recibe parámetros ni tiene retorno.**
- Para la **clase Gato: método emitirSonido()**: Este método simplemente mostrará por consola el sonido que hace el animal, en este clase simplemente muestra el mensaje "Miau, miau, miau!!". **Este método no recibe parámetros ni tiene retorno.**
- Para la **clase Vaca: método emitirSonido()**: Este método simplemente mostrará por consola el sonido que hace el animal, en este clase simplemente muestra el mensaje "Muuu, muuu, muuu!!". **Este método no recibe parámetros ni tiene retorno.**

Por último, una vez creada la clase anterior, vamos a crear una nueva **clase** llamada **Granja** con un **método main** para poder crear las distintas instancias y así poder probar los métodos creados.

En esta clase, dentro del **método main**, vamos a realizar lo siguiente:

1. Creamos **varias instancias**, una de tipo Animal, otra de tipo Perro, otra de tipo Gato y otra de tipo Vaca.
2. Una vez creadas las instancias llamaremos para cada una de ellas al **método emitirSonido** para que **muestre por pantalla el sonido de cada instancia o cada animal**.

Resultado esperado:



```
<terminated> Granja [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe
Sonido del animal
Guau, guau, guau!!!
Miau, miau, miau!!!
Muuu, muuu, muuu!!!
```

PROGRAMACIÓN ORIENTADA A OBJETOS (PARTE I) – PRÁCTICA II

EJERCICIOS DE POLIMORFISMO

Actividad 1. Polimorfismo

Vamos a hacer una aplicación o **programa que realice la implementación de los diferentes sonidos de los animales utilizando el Polimorfismo.**

Para este ejercicio, vamos a **reutilizar las clases** creadas en el ejercicio anterior (**Animal**, **Perro**, **Gato** y **Vaca**) del **paquete “EjerciciosSobrecargaSobrescritura”**.

Lo único que tenemos que hacer, es crear una nueva **clase** en el mismo **paquete** llamada **SonidosAnimales** con un método **main** para poder crear las distintas instancias y así poder probar los métodos creados.

En esta clase, dentro del método **main**, vamos a realizar lo siguiente:

1. Creamos una instancia de tipo **Animal** y mostramos su sonido con el método **emitirSonido**.
`Animal animal = new Animal();`
2. Ahora, a la variable referencia anterior de tipo **Animal** le damos la forma de **Perro**, es decir, a la variable de referencia anterior de tipo **Animal** ahora la usamos para crear la instancia usando la clase **Perro** (`animal = new Perro();`). Mostraremos su sonido con el método **emitirSonido**.
3. Ahora, a la variable referencia anterior de tipo **Animal** le damos la forma de **Gato**, es decir, a la variable de referencia anterior de tipo **Animal** ahora la usamos para crear la instancia usando la clase **Gato** (`animal = new Gato();`). Mostraremos su sonido con el método **emitirSonido**.
4. Ahora, a la variable referencia anterior de tipo **Animal** le damos la forma de **Vaca**, es decir, a la variable de referencia anterior de tipo **Animal** ahora la usamos para crear la instancia usando la clase **Vaca** (`animal = new Vaca();`). Mostraremos su sonido con el método **emitirSonido**.

NOTA. Fíjate que la **variable de referencia “animal”** toma **distinto comportamiento (polimorfismo)** dependiendo de la clase que estamos usando para su creación.

Pasos a seguir:

1. Dentro del proyecto **“FormacionJava”** en el **paquete** llamado **“EjerciciosSobrecargaSobrescritura”** vamos a crear la nueva **clase SonidosAnimales** con método **main()**.

2. Dentro del **método main**, vamos a crearnos la instancia de tipo **Animal** y llamamos al **método emitirSonido()**.

```
Animal animal = new Animal();  
animal.emitirSonido();
```

3. Ahora, a la variable de referencia “animal” creada antes de tipo **Animal**, la usamos para **crear la instancia** de tipo **Perro** y llamamos al **método emitirSonido()**.

```
animal = new Perro();  
animal.emitirSonido();
```

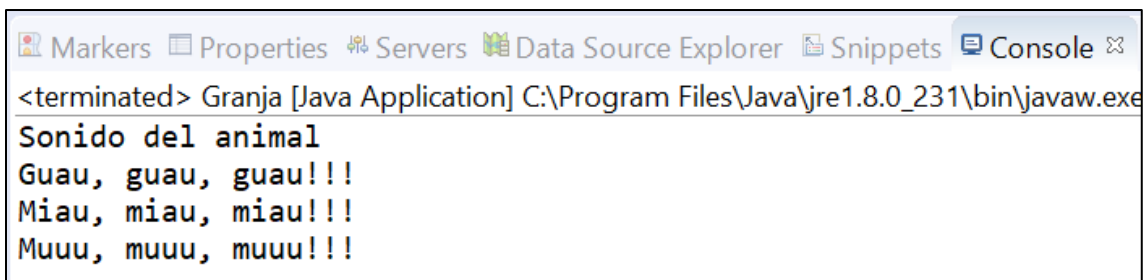
4. Ahora, a la variable de referencia “animal” creada antes de tipo **Animal**, la usamos para **crear la instancia** de tipo **Gato** y llamamos al **método emitirSonido()**.

```
animal = new Gato();  
animal.emitirSonido();
```

5. Ahora, a la variable de referencia “animal” creada antes de tipo **Animal**, la usamos para **crear la instancia** de tipo **Vaca** y llamamos al **método emitirSonido()**.

```
animal = new Vaca();  
animal.emitirSonido();
```

Resultado esperado:



```
Markers Properties Servers Data Source Explorer Snippets Console  
<terminated> Granja [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe  
Sonido del animal  
Guau, guau, guau!!!  
Miau, miau, miau!!!  
Muuu, muuu, muuu!!!
```

Actividad 2. Polimorfismo

Vamos a hacer una aplicación o **programa que gestiona los tipos de vehículos que tiene una empresa de alquiler de vehículos.**

En primer lugar, en el proyecto “**FormacionJava**” nos vamos a crear un nuevo **paquete “EjercicioPolimorfismo”** y dentro de este paquete nos vamos a **crear una clase** nueva llamada **Vehiculo** que define los atributos y métodos comunes para todos los vehículos (**Turismo**, **Deportivo** y **Furgoneta**). Estas tres clases heredan de la clase **Vehículo**.

La **clase Vehiculo** tendrá los siguientes **atributos privados**:

- **marca** de tipo String
- **modelo** de tipo String
- **matricula** de tipo String

La **clase Deportivo** tendrá además el siguiente **atributo privado**:

- **cilindrada** de tipo int

La **clase Furgoneta** tendrá el **atributo privado**:

- **carga** de tipo int

Para las tres clases, se deben crear los siguientes **métodos**:

- **Constructor vacío**
- **Constructor de copia**
- **Constructor de parámetros**
- **Métodos setters/getters** para asignar y obtener los datos de los atributos.

Para la clase Vehiculo vamos a crear un **método**:

- **Método mostrarAtributos():** Este método simplemente mostrará los valores de los atributos de la clase de la siguiente forma:

```
Datos del vehículo
-----
Marca: <marca>
Modelo: <modelo>
Matricula: <matricula>
```

Este método no tiene parámetros de entrada ni tiene ningún retorno.

Para la clase Turismo, Deportivo y Furgoneta vamos a crear **sobrescribir el método** de la **clase Vehiculo** de la siguiente forma:

- **Clase Turismo: Método mostrarAtributos():** Este método simplemente mostrará los valores de los atributos de la clase de la siguiente forma:

```
Datos del turismo
-----
Marca: <marca>
Modelo: <modelo>
Matricula: <matricula>
```

Este método no tiene parámetros de entrada ni tiene ningún retorno.

- **Clase Deportivo: Método mostrarAtributos():** Este método simplemente mostrará los valores de los atributos de la clase de la siguiente forma:

```
Datos del deportivo
-----
Marca: <marca>
Modelo: <modelo>
Matricula: <matricula>
Cilindrada: <cilindrada>
```

Este método no tiene parámetros de entrada ni tiene ningún retorno.

- **Clase Furgoneta : Método mostrarAtributos():** Este método simplemente mostrará los valores de los atributos de la clase de la siguiente forma:

```
Datos de la furgoneta
-----
Marca: <marca>
Modelo: <modelo>
Matricula: <matricula>
Carga: <carga>
```

Este método no tiene parámetros de entrada ni tiene ningún retorno.

Por último, vamos a **crear** una nueva **clase** llamada **GestionVehiculos** con un **método main** dentro del mismo paquete para poder crear las distintas instancias y así poder probar los métodos creados.

En esta clase, dentro del **método main**, vamos a realizar lo siguiente:

1. **Creamos una instancia** de tipo **Vehículo** pero utilizando la clase **Turismo**, la damos valores y mostramos sus atributos.

```
Vehiculo vehiculo = new Turismo("Peugeot","307","8917 CFW");
```

2. Ahora, **a la variable referencia anterior de tipo Vehiculo le damos la forma de Deportivo**, es decir, a la variable de referencia anterior de tipo Vehiculo ahora la usamos para crear la instancia usando la clase Deportivo y mostramos sus atributos.

```
vehiculo = new Deportivo("Ford","Mustang","4070 DEP",150);
```

3. Ahora, **a la variable referencia anterior de tipo Vehiculo le damos la forma de Furgoneta**, es decir, a la variable de referencia anterior de tipo Vehiculo ahora la usamos para crear la instancia usando la clase Furgoneta y mostramos sus atributos.

```
vehiculo = new Furgoneta("Fiat","Ducato","4080 FUR",1200);
```

NOTA. Fíjate que la **variable de referencia “vehiculo”** toma **distinto comportamiento (polimorfismo)** dependiendo de la clase que estamos usando para su creación.

Resultado esperado:

```
<terminated> GestionVehiculos [Java Application] C:\Program Files\Java\jre1.8.0_231\bin
Datos del turismo
-----
Marca: Peugeot
Modelo: 307
Matrícula: 8917 CFW

Datos del deportivo
-----
Marca: Ford
Modelo: Mustang
Matrícula: 4070 DEP
Cilindrada: 150

Datos de la furgoneta
-----
Marca: Fiat
Modelo: Ducato
Matrícula: 4080 FUR
Carga: 1200
```

PROGRAMACIÓN ORIENTADA A OBJETOS (PARTE I) – PRÁCTICA II

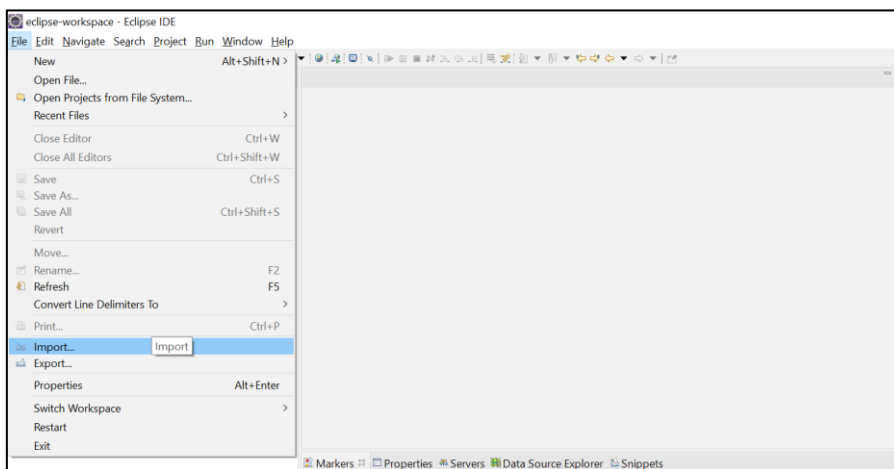
SOLUCIONES A LAS ACTIVIDADES

Una vez instalado el **software necesario (JDK y Eclipse)** y creado el **área de trabajo en Eclipse** (documento **“Primeros pasos en Eclipse”** en contenido descargable del curso) podemos importar las soluciones a estas actividades en nuestro Eclipse.

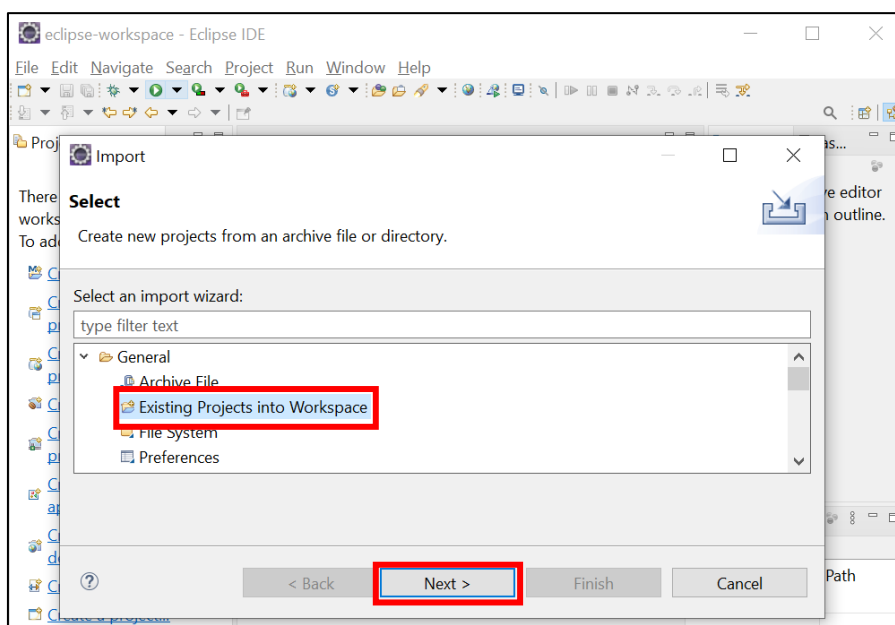
Para ello descomprime en tu directorio local el fichero .zip, que encontrarás también en la sección inferior **contenido descargable**, con el nombre **“Solución de las prácticas”**.

Una vez descomprimido el fichero .zip, abre tu Eclipse y continua con los siguientes pasos:

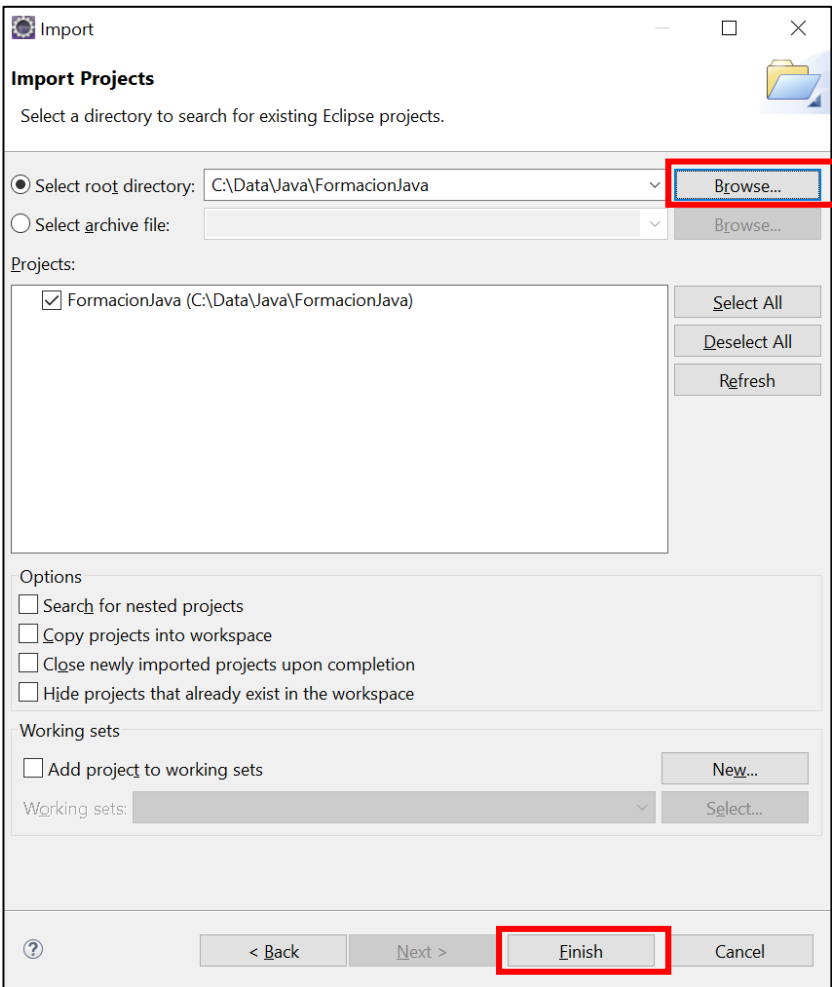
1. Abre en el **menú File → Import**.



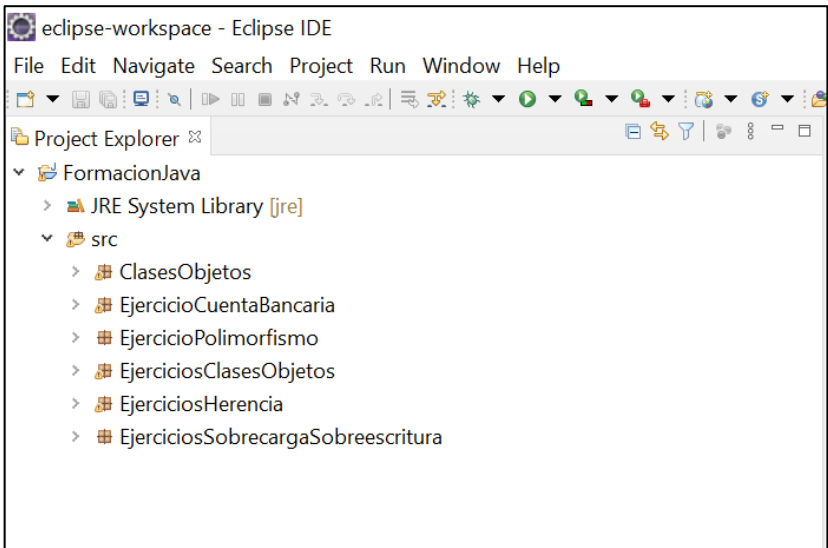
2. Aparece la siguiente ventana en la que **seleccionamos** la **opción “Existing Projects into Workspaces”** y **pulsamos “Next”**.



3. Selecciona mediante el **botón “Browse”**, el **directorio “FormacionJava”** que acabas de descomprimir en tu ordenador y **pulsa** después el **botón “Finish”**.



4. En tu workspace ya tienes el Project importado con las **soluciones a las actividades**.



**FUNDACIÓN
ACCENTURE**

accenture