

CONCEPTOS RELACIONADOS CON LA POO

PRÁCTICA II



CONCEPTOS RELACIONADOS CON LA POO – PRÁCTICA II

GUIA DEL DOCUMENTO

Este documento consta del siguiente contenido:

- Instrucciones previas.
- Ejercicios de **composición** (2 actividades).
- Ejercicios de **genéricos** (2 actividades).
- Ejercicios de **colecciones** (3 actividades).
- Ejercicios de **excepciones** (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.

CONCEPTOS RELACIONADOS CON LA POO – 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.

CONCEPTOS RELACIONADOS CON LA POO – PRÁCTICA II

EJERCICIOS DE COMPOSICIÓN

Actividad 1. Composición

Vamos a crear una **aplicación que gestione las operaciones de cuentas bancarias**.

Para este ejercicio, vamos a crear en un **paquete** nuevo “**EjerciciosComposicion**” y dentro vamos a crear una **clase** llamada **Persona**.

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

- **nombreCompleto** de tipo String
- **nif** de tipo String
- **direccion** de tipo String

Para esta clase se deben crear los siguientes **métodos**:

- Métodos **setters/getters**.
- Sobrescribir el **método toString** para que muestre los datos de la persona de la siguiente forma:

```
"Nombre: " <nombreCompleto>
"D.N.I: " <nif>
"Dirección: " <dirección>
```

Nota: Para poder conseguir el salto de línea en un String, hay que usar el carácter \n, de la siguiente forma: “Datos personales \nNombre: ”

A continuación, en el mismo paquete, vamos a crear una nueva **clase** llamada **Cuenta**.

Esta clase tendrá los siguientes atributos privados:

- **titularCta** de tipo Persona (aquí tenemos la **composición**)
- **numeroCta** de tipo String
- **saldoCta** de tipo double

Se deben crear los siguientes **métodos**:

- Métodos **setters/getters**.
- Sobrescribir el **método toString** para que muestre los datos de la cuenta de la siguiente forma:

```
"Datos de la cuenta"
"-----"
"Datos del titular"
<titularCta>
"Número de cuenta: " numeroCta
"Saldo: " saldoCta
```

- **Método ingreso(double cantidad):** Consiste en aumentar el saldo de la cuenta con la cantidad que se pase. Debemos comprobar que la cantidad no puede ser negativa, si la cantidad introducida es negativa, no se hará nada. **Este método recibe un parámetro de entrada con la cantidad a ingresar y retorna true si se ha realizado la operación o false si no se ha realizado.**
- **Método reintegro(double cantidad):** Consiste en retirar una cantidad a la cuenta, es decir, debemos disminuir el saldo de la cuenta. Debemos comprobar que la cantidad no sea negativa y además que hay saldo suficiente para hacer el reintegro, si la cantidad es negativa o si no hay saldo suficiente, no se hará nada. **Este método recibe un parámetro de entrada con la cantidad a ingresar y retorna true si se ha realizado la operación o false si no se ha realizado.**
- **Método transferencia(Cuenta ctaDestino, double cantidad):** Este método nos debe permitir pasar dinero de una cuenta a otra siempre que en la cuenta de origen haya saldo suficiente. **Este método recibe dos parámetros, uno con una instancia de tipo Cuenta, que será la cuenta destino del importe, y un segundo parámetro con la cantidad a transferir, de tal forma, que disminuimos el saldo de la cuenta y se lo incrementamos a la otra cuenta. Debe retorna true si se ha realizado la operación o false si no se ha realizado.**

Por último, en el mismo **paquete** creamos una nueva **clase OperacionesBancarias** con un **método main** para poder crear las **instancias** de tipo **Cuenta** y **Persona**, y además poder probar los distintos métodos creados.

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

1. Creamos dos instancias de tipo Persona con el constructor vacío.
2. Creamos dos instancias de tipo Cuenta usando el constructor vacío.
3. Ahora, a las instancias de tipo Cuenta creadas en el punto anterior, vamos a ir asignando valores a sus atributos con los métodos setters. Puedes usar los siguientes datos:
 - 1º cta: titularCta = "Federico Garcia", "11.111.111-H", "Calle Nueva, 5"
numeroCta = "01012020030340400505"
saldoCta = 2890.00
 - 2º cta: titularCta = "Alfonso de la Calle", "22.222.222-G", "Calle Real, 5"
numeroCta = "11112222333344445555"
saldoCta = 90.00

Nota: Ten en cuenta que el atributo titularCta es de tipo Persona, así que tendrás que dar valor a sus atributos también, tienes la opción de hacerlo atributo por atributo o dando valor a las instancias de tipo persona y luego asignarlas a este atributo.

4. Realiza un ingreso sobre la segunda cuenta creada anteriormente, si se ha podido realizar la operación se muestra por pantalla un mensaje de "Ingreso realizado correctamente" y la información de dicha cuenta para comprobar que su saldo ha cambiado, y si no ha podido realizarse la operación mostrará el mensaje "No se ha podido realizar la operación". Datos para la operación: 100.00
5. Realiza un reintegro sobre la primera cuenta creada anteriormente, si se ha podido realizar la operación se muestra por pantalla un mensaje de "Reintegro realizado correctamente" y la información de dicha cuenta para comprobar que su saldo ha cambiado, y si no ha podido realizarse la operación mostrará el mensaje "No se ha podido realizar la operación". Datos para la operación: 290.00
6. Realiza una transferencia entre las dos cuentas creadas anteriormente, si se ha podido realizar la operación se muestra por pantalla un mensaje de "Transferencia realizada correctamente" y la información de ambas cuentas para comprobar que su saldo ha cambiado, y si no ha podido realizarse la operación mostrará el mensaje "No se ha podido realizar la operación". Datos para la operación: 1000.00

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 esta clase, al que denominaremos "**EjerciciosComposicion**".
2. Dentro del **paquete "EjerciciosComposicion"** crearemos la **clase Persona**.

```
public class Persona
```

3. Dentro de la **clase Persona** creamos sus **atributos privados**.

```
private String nombreCompleto;  
private String nif;  
private String direccion;
```

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. Por último en esta clase, 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()....".

```
@Override
public String toString() {
    return "\n\t\t\tNombre: " + nombreCompleto +
           "\n\t\t\tD.N.I: " + nif +
           "\n\t\t\tDireccion:" + direccion;
}
```

6. Ahora, dentro del **paquete "EjerciciosComposicion"** crearemos la **clase Cuenta**.
7. Dentro de la **clase Cuenta**, lo primero que vamos a hacer, es crear los distintos **atributos**.

```
private Persona titularCta;
private String numeroCta;
private double saldoCta;
```

8. 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...."

9. 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()...."

```
@Override
public String toString() {
    return "Datos de la cuenta" + "\n-----" +
           "\nDatos del titular" + titularCta +
           "\nNúmero de cuenta: " + numeroCta +
           "\nSaldo: " + saldoCta;
}
```

10. Por último, vamos a crear los tres **métodos que marcan el comportamiento** que va a tener nuestra cuenta.

```
public boolean ingreso(double cantidad)
public boolean reintegro(double cantidad)
public boolean transferencia(Cuenta ctaDestino, double cantidad)
```

11. Ahora, dentro del **paquete “EjercicioCuentaBancaria”**, creamos la **clase OperacionesBancarias** con el **método main()**.

```
public class OperacionesBancarias
```

12. Dentro de la **clase OperacionesBancarias** dentro del **método main()**, lo primero que vamos a hacer, es crear las **instancias** de tipo **Persona** con el constructor vacío.

```
Persona pFederico = new Persona();
Persona pAlfonso = new Persona();
```

13. A continuación, creamos dos **instancias** de tipo **Cuenta** con el constructor vacío.

```
Cuenta cta1 = new Cuenta();
Cuenta cta2 = new Cuenta();
```

14. Tras la creación de las **instancias** de tipo **Cuenta**, vamos a asignar los valores a los atributos de ambas instancias.

Nota: Ten en cuenta que el atributo titularCta es de tipo Persona, así que tendrás que dar valor a sus atributos también, tienes la opción de hacerlo atributo por atributo o dando valor a las instancias de tipo persona y luego asignarlas a este atributo.

```
cta1.setTitularCta(pFederico);
cta1.getTitularCta().setNombreCompleto("Federico Garcia");
cta1.getTitularCta().setNif("11.111.111-H");
cta1.getTitularCta().setDireccion("Calle Nueva, 5");
cta1.setNumeroCta("01012020030340400505");
cta1.setSaldoCta(2890.00);

pAlfonso.setNombreCompleto("Alfonso de la Calle");
pAlfonso.setNif("22.222.222-G");
pAlfonso.setDireccion("Calle Real, 5");
cta2.setTitularCta(pAlfonso);
cta2.setNumeroCta("1111222333344445555");
cta2.setSaldoCta(90.00);
```


15. Por último, sobre la segunda **instancia** de tipo **Cuenta** creada vamos a realizar un ingreso de 100.00..

Nota: Acuérdate que este método retorna un valor boolean, así que puedes hacer la llamada dentro de un if para saber si se ha realizado o no y así mostrar la información de la cuenta o el mensaje de error correspondiente.

```
if (cta2.ingreso(100.00)) {  
    System.out.println("Ingreso realizado correctamente");  
    System.out.println(cta2);  
} else {  
    System.out.println("No se ha podido realizar la operación");  
}
```

16. Ahora, sobre la primera **instancia** creada de tipo **Cuenta** vamos a realizar un reintegro de 290.00

Nota: Acuérdate que este método retorna un valor boolean, así que puedes hacer la llamada dentro de un if para saber si se ha realizado o no y así mostrar la información de la cuenta o el mensaje de error correspondiente.

```
if (cta1.reintegro(290.00)) {  
    System.out.println("Reintegro realizado correctamente");  
    System.out.println(cta1);  
} else {  
    System.out.println("No se ha podido realizar la operación");  
}
```

17. Por último, sobre la primera **instancia** de tipo **Cuenta** creada vamos a realizar una transferencia hacia la segunda instancia de tipo Cuenta creada de 1000.00

Nota: Acuérdate que este método retorna un valor boolean, así que puedes hacer la llamada dentro de un if para saber si se ha realizado o no y así mostrar la información de la cuenta o el mensaje de error correspondiente

```
if (cta1.transferencia(cta2, 1000.00)) {  
    System.out.println("Transferencia realizada correctamente");  
    System.out.println(cta1);  
    System.out.println(cta2);  
} else {  
    System.out.println("No se ha podido realizar la operación");  
}
```

Resultado esperado:

```
Markers Properties Servers Data Source Explorer Snippets Console
<terminated> OperacionesBancarias (1) [Java Application] C:\Program Files\Java\jre1.8.0
Ingreso realizado correctamente
Datos de la cuenta
-----
Datos del titular
    Nombre: Alfonso de la Calle
    D.N.I: 22.222.222-G
    Direccion:Calle Real, 5
Número de cuenta: 11112222333344445555
Saldo: 190.0

Reintegro realizado correctamente
Datos de la cuenta
-----
Datos del titular
    Nombre: Federico Garcia
    D.N.I: 11.111.111-H
    Direccion:Calle Nueva, 5
Número de cuenta: 01012020030340400505
Saldo: 2600.0

Transferencia realizada correctamente
Datos de la cuenta
-----
Datos del titular
    Nombre: Federico Garcia
    D.N.I: 11.111.111-H
    Direccion:Calle Nueva, 5
Número de cuenta: 01012020030340400505
Saldo: 1600.0
Datos de la cuenta
-----
Datos del titular
    Nombre: Alfonso de la Calle
    D.N.I: 22.222.222-G
    Direccion:Calle Real, 5
Número de cuenta: 11112222333344445555
Saldo: 1190.0
```

Actividad 2. Composición

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

Para este ejercicio vamos a reutilizar la **clase Persona** creada en el ejercicio anterior para gestionar las personas que pueden alquilar nuestros vehículos.

Por otro lado, en el proyecto “**FormacionJava**” dentro del **paquete “EjerciciosComposicion”** nos vamos a crear una **clase** nueva llamada **Vehículo** que define los atributos y métodos comunes para todos los vehículos que podemos tener en nuestra empresa de alquiler.

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

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

Además, esta clase tendrá 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 que muestre los datos de la siguiente forma:

```
Marca: <marca>
Modelo: <modelo>
Matricula: <matricula>
```

Además, en este mismo **paquete**, vamos a crear las distintas **clases Turismo y Furgoneta** para los distintos tipos de vehículos que podemos tener en la empresa, de tal forma que estas dos clases extiendan o hereden de la **clase Vehiculo**.

La **clase Turismo** no tendrá ningún atributo adicional, pero tendrá los siguientes métodos.

- **Constructor vacío**
- **Constructor de copia**
- **Constructor de parámetros**

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

- **carga** de tipo int

Además, esta clase tendrá 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 que muestre los datos de la siguiente forma:

Marca: <marca>
Modelo: <modelo>
Matricula: <matricula>
Carga: <carga>

También es necesario crearnos una nueva **clase** llamada **Alquiler**, dentro del mismo paquete, que se encargue de recoger la gestión de los alquileres de la empresa.

La **clase Alquiler** tendrá los atributos privados:

- **cliente** de tipo Persona (aquí tenemos la **composición**)
- **vehiculo** de tipo Vehículo (aquí tenemos la **composición**)
- **fechaAlquiler** de tipo String

Además, esta clase tendrá los siguientes métodos:

- **Constructor vacío**
- **Constructor de copia**
- **Constructor de parámetros**
- **Métodos setters/getters** para asignar y obtener los datos de los atributos.
- **Método mostrarAlquiler()** para que muestre los datos de la siguiente forma:

Persona: <cliente> → podemos usar el método toString del atributo
Vehículo: <vehiculo> → podemos usar el método toString del atributo
Fecha alquiler: <fechaAlquiler>

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

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

1. Creamos una instancia de tipo Vehiculo pero utilizando la clase Turismo, la damos valores.

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

2. Creamos una instancia de tipo Vehiculo pero utilizando la clase Furgoneta, la damos valores.

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

3. Creamos dos instancias de tipo Persona.

```
Persona cliente1 = "Federico Garcia", "11.111.111-H", "Calle Nueva, 5"
```

```
Persona cliente2 = "Alfonso de la Calle", "22.222.222-G", "Calle Real, 5"
```

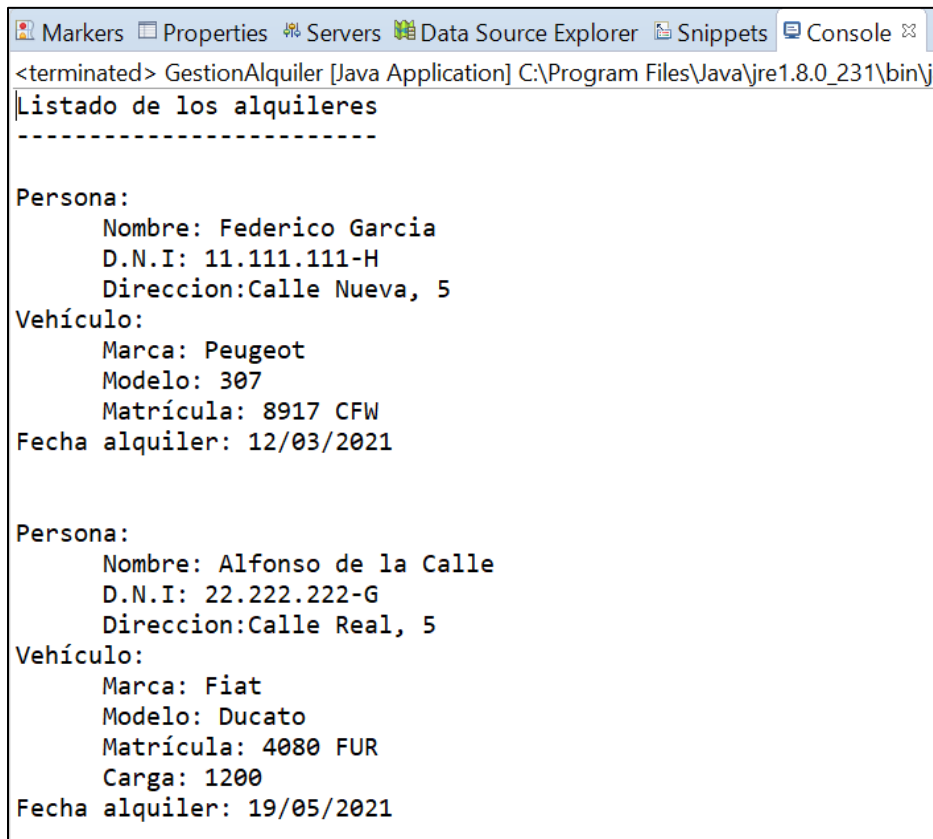
4. Creamos dos instancias de tipo Alquiler con los datos de dos alquileres de los dos vehículos y las dos personas.

```
Alquiler alquiler1 = new Alquiler(cliente1, turismo, "12/03/2021");
```

```
Alquiler alquiler2 = new Alquiler(cliente2, furgoneta, "19/05/2021");
```

5. Sacamos un listado con los datos de los distintos alquileres que tenemos

Resultado esperado:



```
<terminated> GestionAlquiler [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\j
Listado de los alquileres
-----

Persona:
  Nombre: Federico Garcia
  D.N.I: 11.111.111-H
  Direccion:Calle Nueva, 5
Vehículo:
  Marca: Peugeot
  Modelo: 307
  Matrícula: 8917 CFW
Fecha alquiler: 12/03/2021

Persona:
  Nombre: Alfonso de la Calle
  D.N.I: 22.222.222-G
  Direccion:Calle Real, 5
Vehículo:
  Marca: Fiat
  Modelo: Ducato
  Matrícula: 4080 FUR
  Carga: 1200
Fecha alquiler: 19/05/2021
```

CONCEPTOS RELACIONADOS CON LA POO – PRÁCTICA II

EJERCICIOS DE GENÉRICOS

Actividad 1. Genéricos

Escribe una **clase genérica** **Operaciones**, que declare las cuatro operaciones básicas: **suma, resta, producto y división**.

Para este ejercicio, vamos a crear un nuevo **paquete** “**EjerciciosGenericos**” y dentro vamos a crear una **clase genérica** llamada **Operaciones** con un tipo genérico <N> (public class Operaciones <N extends Number>).

Esta clase genérica va a tener **cuatro métodos** y todos ellos reciben un objeto de tipo N y devuelven un double, y serán los siguientes:

- **sumar(N num):** Este método suma 2 al valor pasado como parámetro y retorna el resultado de dicha operación.
- **restar(N num):** Este método resta 2 al valor pasado como parámetro y retorna el resultado de dicha operación.
- **multiplicar(N num):** Este método multiplica por 2 al valor pasado como parámetro y retorna el resultado de dicha operación.
- **dividir(N num):** Este método divide por 2 al valor pasado como parámetro y retorna el resultado de dicha operación.

Por último, en el mismo **paquete** creamos una **clase** llamada **DemoOperaciones** con un **método main** para probar la clase genérica Operaciones creada anteriormente.

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

1. Creamos una primera instancia de tipo Operaciones pasando un tipo Integer.
2. Sobre esta instancia, realizamos las cuatro operaciones y mostramos el resultado obtenido por consola. Vamos a utilizar el valor 5 de prueba.
3. Creamos una primera instancia de tipo Operaciones pasando un tipo Double.
4. Sobre esta instancia, realizamos las cuatro operaciones y mostramos el resultado obtenido por consola. Vamos a utilizar el valor 12.34 de prueba.

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 esta clase, al que denominaremos “**EjerciciosGenericos**”.

2. Dentro del paquete “**EjerciciosGenericos**” crearemos la **clase Operaciones**.

```
public class Operaciones <N extends Number>
```

3. Dentro de la **clase Operaciones** vamos a crear los cuatro **métodos**.

```
public double suma(N num){
    return num.doubleValue() + 2;
}
public double resta(N num){
    return num.doubleValue() - 2;
}
public double producto(N num){
    return num.doubleValue() * 2;
}
public double division(N num){
    return num.doubleValue() / 2;
}
```

4. Ahora, dentro del mismo **paquete “EjerciciosGenericos”**, creamos la **clase DemoOperaciones** con el **método main()**.

```
public class DemoOperaciones
```

5. Dentro de la **clase DemoOperaciones** dentro del **método main()**, lo primero que vamos a hacer, es crear la primera instancia de la clase genérica pasando como parámetro un Integer.

```
Operaciones<Integer> iOb = new Operaciones<Integer>();
```

6. A continuación, sobre esta **instancia realizamos las cuatro operaciones y mostramos el resultado por consola**.

```
System.out.println("La suma es: " + iOb.suma(5));
System.out.println("La resta es: " + iOb.resta(5));
System.out.println("La multiplicación es: " + iOb.producto(5));
System.out.println("La división es: " + iOb.division(5));
```

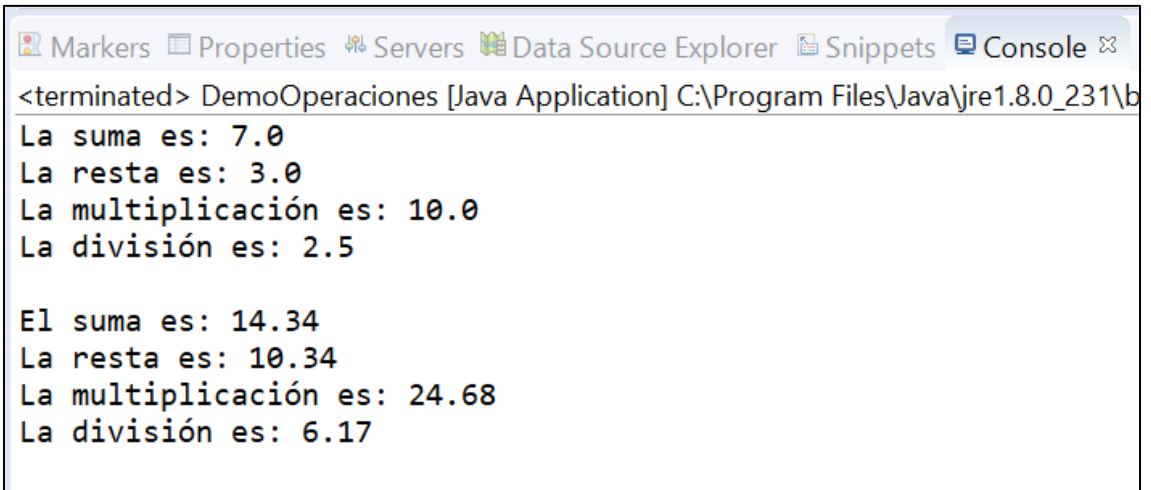
7. Ahora, vamos a **crear la segunda instancia de la clase genérica** pasando como parámetro un Double.

```
Operaciones<Double> dOb = new Operaciones<Double>();
```

8. Por último, sobre esta **instancia realizamos las cuatro operaciones y mostramos el resultado por consola.**

```
System.out.println("El suma es: " + dOb.suma(12.34));  
System.out.println("La resta es: " + dOb.resta(12.34));  
System.out.println("La multiplicación es: " + dOb.producto(12.34));  
System.out.println("La división es: " + dOb.division(12.34));
```

Resultado esperado:



```
<terminated> DemoOperaciones [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\java.exe  
La suma es: 7.0  
La resta es: 3.0  
La multiplicación es: 10.0  
La división es: 2.5  
  
El suma es: 14.34  
La resta es: 10.34  
La multiplicación es: 24.68  
La división es: 6.17
```


Actividad 2. Genéricos

Escribe una **clase** **MetodosGenericosArrays**, que nos permitirá mostrar y contar los elementos de un array.

Este ejercicio lo vamos a crear en el **paquete** “**EjerciciosGenericos**” y dentro vamos a crear una **clase** llamada **MetodosGenericosArrays** con un **método** **main()**.

Esta **clase** va a tener dos **métodos genéricos**:

- **Método genérico mostrarElementosArray**: Este método recibe como parámetro un array de tipo genérico y no tiene retorno. Es método se va a encargar de recorrer el array pasado como parámetro y mostrar por consola los elementos de dicho array.

```
public static <E> void mostrarElementosArray(E[] inputArray)
```

- **Método genérico contarElementosArray**: Este método recibe como parámetro un array de tipo genérico y nos retorna un String con el mensaje con los elementos del array.

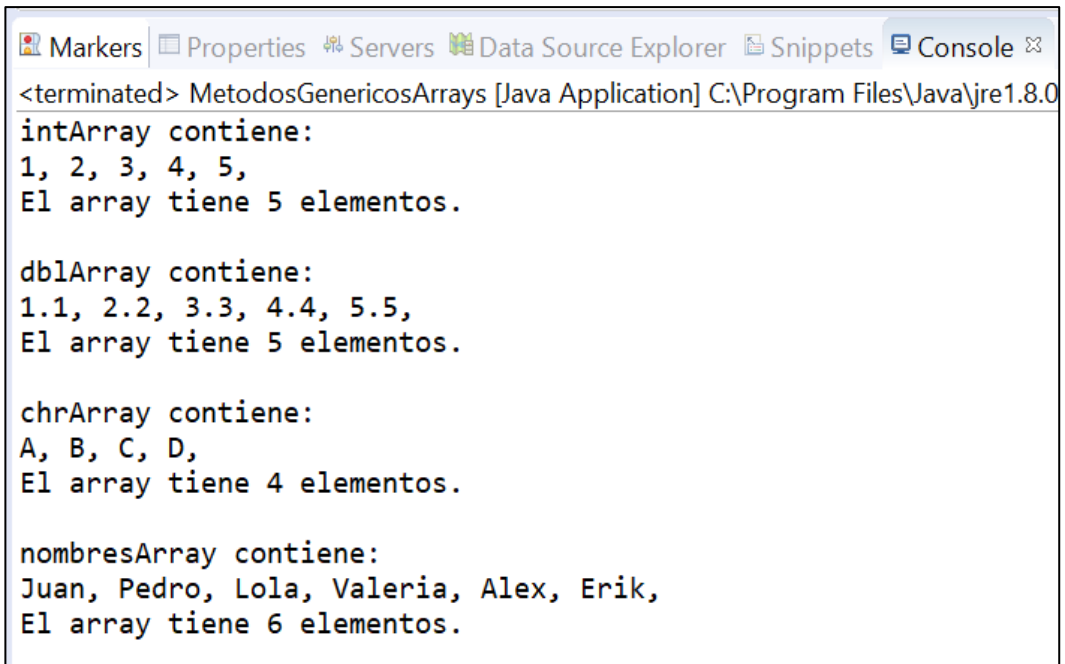
```
public static <T> String contarElementosArray(T[] inputArray)
```

Por último, en esta **clase** dentro del **método main** vamos a realizar lo siguiente para probar los métodos creados anteriormente:

1. Creamos un array de tipo Integer: `Integer[] intArray = { 1, 2, 3, 4, 5 };`
2. Haciendo uso de los métodos anteriores, mostramos los elementos de este array y después el número de elementos que tiene.
`mostrarElementosArray(intArray);`
`System.out.println();`
`System.out.println(contarElementosArray(intArray));`
`System.out.println();`
3. Ahora creamos un array de tipo Double con los valores { 1.1, 2.2, 3.3, 4.4, 5.5 }
4. Haciendo uso de los métodos anteriores, mostramos los elementos de este array y después el número de elementos que tiene.
5. Ahora creamos un array de tipo Character con los valores { 'A', 'B', 'C', 'D' }
6. Haciendo uso de los métodos anteriores, mostramos los elementos de este array y después el número de elementos que tiene.

7. Ahora creamos un array de tipo String con los valores { "Juan", "Pedro", "Lola", "Valeria", "Alex", "Erik" }
8. Haciendo uso de los métodos anteriores, mostramos los elementos de este array y después el número de elementos que tiene.

Resultado esperado:



```
<terminated> MetodosGenericosArrays [Java Application] C:\Program Files\Java\jre1.8.0
intArray contiene:
1, 2, 3, 4, 5,
El array tiene 5 elementos.

dblArray contiene:
1.1, 2.2, 3.3, 4.4, 5.5,
El array tiene 5 elementos.

chrArray contiene:
A, B, C, D,
El array tiene 4 elementos.

nombresArray contiene:
Juan, Pedro, Lola, Valeria, Alex, Erik,
El array tiene 6 elementos.
```

CONCEPTOS RELACIONADOS CON LA POO – PRÁCTICA II

EJERCICIOS DE COLECCIONES

Actividad 1. Colecciones

Vamos a crear una **aplicación que gestione un listado de personas y las ordene de menor a mayor**.

Para este ejercicio, podemos utilizar la **clase Persona** creada en ejercicios anteriores que se encuentra en el **paquete "ClasesObjetos"** o podemos crear un nuevo paquete **"EjerciciosColecciones"** y dentro creamos una **clase** nueva llamada **Persona**. En ambos casos, la **clase Persona** debe **implementar de la interface Comparable** para crear un criterio de ordenación por el atributo edad.

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

- **nombre** de tipo String
- **primerApellido** de tipo String
- **segundoApellido** de tipo String
- **dni** de tipo String
- **edad** de tipo int

Para esta clase se deben crear los siguientes **métodos**:

- Crear los tres **constructores**
- **Métodos setters/getters** para asignar y obtener los datos de los atributos.
- **Sobrescribir** el **método toString** para que muestre los datos de la persona de la siguiente forma:

```
"Nombre: " <nombreCompleto>
"D.N.I: " <nif>
"Edad: " <edad>
```

Nota: Para poder conseguir el salto de línea en un String, hay que usar el carácter \n, de la siguiente forma: "Datos personales \nNombre: "

- **Sobrescribir** o generar el **método equals** y **hashCode** para poder comparar objetos de este tipo.
- **sobrescribir** el **método compareTo** dado por la **interface Comparable**. Este método debe ordenar las personas por el atributo edad de menor a mayor.

Por último, en el mismo paquete creamos una nueva **clase ListadoPersonas** con un **método main** para poder crear las **instancias** de tipo **Persona**, la lista de personas y ordenarla.

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

1. Creamos cuatro instancias de tipo **Persona** con el constructor de parámetros. Puedes usar los siguientes datos:
 - Primera persona: "Alex", "Muñoz", "Velasco", "12.345.678-L", 25
 - Segunda persona: "Ana", "Gil", "Gil", "12.345.678-L", 55
 - Tercera persona: "Erik", "Muñoz", "Velasco", "12.345.678-L", 34
 - Cuarta persona: "Sara", "Sanz", "Perez", "12.345.678-L", 48
2. Creamos una List usando la implementación ArrayList donde vamos a guardar objetos de tipo **Persona**.
3. Una vez creada la lista, vamos a añadir en ella las distintas instancias de tipo **persona** que hemos creado en el primer punto.
4. Sacamos por consola un listado de las personas que tenemos sin ordenar.
5. Después, podemos ordenar la lista creada por edad de menor a mayor.
6. Por último, volvemos a mostrar por consola el listado de personas, pero esta vez ordenados por la edad.

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 esta clase**, al que denominaremos "**EjerciciosColecciones**".
2. Dentro del paquete "**EjerciciosColecciones**" crearemos la **clase Persona** que implementa la **interface Comparable**.

```
public class Persona implements Comparable <Persona>
```

3. Dentro de la **clase Persona** creamos sus **atributos privados**.

```
private String nombre;  
private String primerApellido;  
private String segundoApellido;  
private String dni;  
private int edad;
```

4. Lo siguiente que vamos a hacer, es crear los **tres constructores**.

```
// Constructor vacío
public Persona() {
}
// Constructor de copia
public Persona(Persona p) {
    this.nombre = p.nombre;
    this.primerApellido = p.primerApellido;
    this.segundoApellido = p.segundoApellido;
    this.dni = p.dni;
    this.edad = p.edad;
}
// Constructor de parámetros
public Persona(String nombre, String primerA, String segundoA,
String dni, int edad) {
    this.nombre = nombre;
    this.primerApellido = primerA;
    this.segundoApellido = segundoA;
    this.dni = dni;
    this.edad = edad;
}
```

5. A continuación, vamos a 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. También 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()...."

```
@Override
public String toString() {
    return "Persona [nombre=" + nombre + ", primerApellido=" +
primerApellido +
", segundoApellido=" + segundoApellido + ", dni=" + dni + ", edad="
+ edad + "]";
}
```

7. También vamos a crear o sobrescribir los **métodos equals y hashCode** para que nos permita comparar objetos de este tipo.

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 hashCode() and equals()...."

8. Por último, vamos a crear o **sobrescribir** el **método compareTo** dado por la **interface Comparable** para marcar el criterio de ordenación por la edad.

```
@Override
public int compareTo(Persona p) {
    return this.edad - p.edad;
}
```

9. Ahora, dentro del **paquete "EjercicioColecciones"**, creamos la **clase ListadoPersonas** con el **método main()**.

```
public class ListadoPersonas
```

10. Dentro de la **clase ListadoPersonas** dentro del **método main()**, lo primero que vamos a hacer, es crear las **instancias** de tipo **Persona** con el **constructor de parámetros**.

```
Persona alex = new Persona("Alex", "Muñoz", "Velasco", "12.345.678-L", 25);
Persona ana = new Persona("Ana", "Gil", "Gil", "12.345.678-L", 55);
Persona erik = new Persona("Erik", "Muñoz", "Velasco", "12.345.678-L", 34);
Persona sara = new Persona("Sara", "Sanz", "Perez", "12.345.678-L", 48);
```

11. Tras la **creación** de las **instancias** de tipo **Persona**, vamos a crearnos la lista para guardar objetos de tipo **Persona**.

```
List<Persona> listadoPersonas = new ArrayList<Persona>();
```

12. Ahora, podemos añadir a la lista creada anteriormente las cuatro **instancias** de tipo **Persona** que hemos creado en el punto 1.

```
listadoPersonas.add(alex);
listadoPersonas.add(ana);
listadoPersonas.add(erik);
listadoPersonas.add(sara);
```

13. Una vez creada la lista, vamos a **mostrar por consola esta lista sin ordenar**.

```
System.out.println("Listado de personas sin ordenar");
System.out.println("-----");
for (Persona persona : listadoPersonas) {
    System.out.println(persona);
}
```

14. Ahora, vamos a **ordenar la lista** por el criterio creado, es decir, por edad de menor a mayor.

```
Collections.sort(listadoPersonas);
```

15. Por último, volvemos a sacar por consola el listado de las personas pero esta vez ordenadas por la edad.

```
System.out.println("Listado de personas ordenadas por edad");
System.out.println("-----");
for (Persona persona : listadoPersonas) {
    System.out.println(persona);
}
```

Resultado esperado:

```
<terminated> ListadoPersonas [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (26 jul. 2021 18:55:22)
Listado de personas sin ordenar
-----
Persona [nombre=Alex, primerApellido=Munoz, segundoApellido=Velasco, dni=12.345.678-L, edad=25]
Persona [nombre=Ana, primerApellido=Gil, segundoApellido=Gil, dni=12.345.678-L, edad=55]
Persona [nombre=Erik, primerApellido=Munoz, segundoApellido=Velasco, dni=12.345.678-L, edad=34]
Persona [nombre=Sara, primerApellido=Sanz, segundoApellido=Perez, dni=12.345.678-L, edad=48]

Listado de personas ordenadas por edad
-----
Persona [nombre=Alex, primerApellido=Munoz, segundoApellido=Velasco, dni=12.345.678-L, edad=25]
Persona [nombre=Erik, primerApellido=Munoz, segundoApellido=Velasco, dni=12.345.678-L, edad=34]
Persona [nombre=Sara, primerApellido=Sanz, segundoApellido=Perez, dni=12.345.678-L, edad=48]
Persona [nombre=Ana, primerApellido=Gil, segundoApellido=Gil, dni=12.345.678-L, edad=55]
```

Actividad 2. Colecciones

Para este ejercicio vamos a intentar poner en práctica el uso de alguna de las interfaces de Colecciones y ordenación, para ello vamos a **crear un listado de los distintos integrantes de un grupo musical ordenados por edad y altura.**

Para este ejercicio, dentro del paquete “**EjerciciosColecciones**”, vamos a crear una **clase** nueva llamada **Integrante** que debe implementar de la **interface Comparable** para crear un criterio de ordenación por el atributo edad.

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

- **nombre** de tipo String
- **edad** de tipo int
- **altura** de tipo double
- **instrumento** de tipo String
- **grupo** de tipo String

Para esta clase se deben crear los siguientes métodos:

- Crear los **tres constructores**
- **Métodos setters/getters** para asignar y obtener los datos de los atributos.
- **Sobrescribir** el **método toString** para que muestre los datos de la persona de la siguiente forma: <nombre> (<edad> y <altura>) - <instrumento> - <grupo>
- **Sobrescribir** o generar el **método equals** y **hashCode** para poder comparar objetos de este tipo.
- **sobrescribir** el **método compareTo** dado por la interface Comparable. Este método debe ordenar las personas por el atributo edad de menor a mayor.

Ahora vamos a crear un segundo criterio de ordenación, para ello, en el mismo **paquete** creamos una nueva **clase OrdenarPorAltura** que implemente de la **interfaz Comparator**.

Para esta clase se debe:

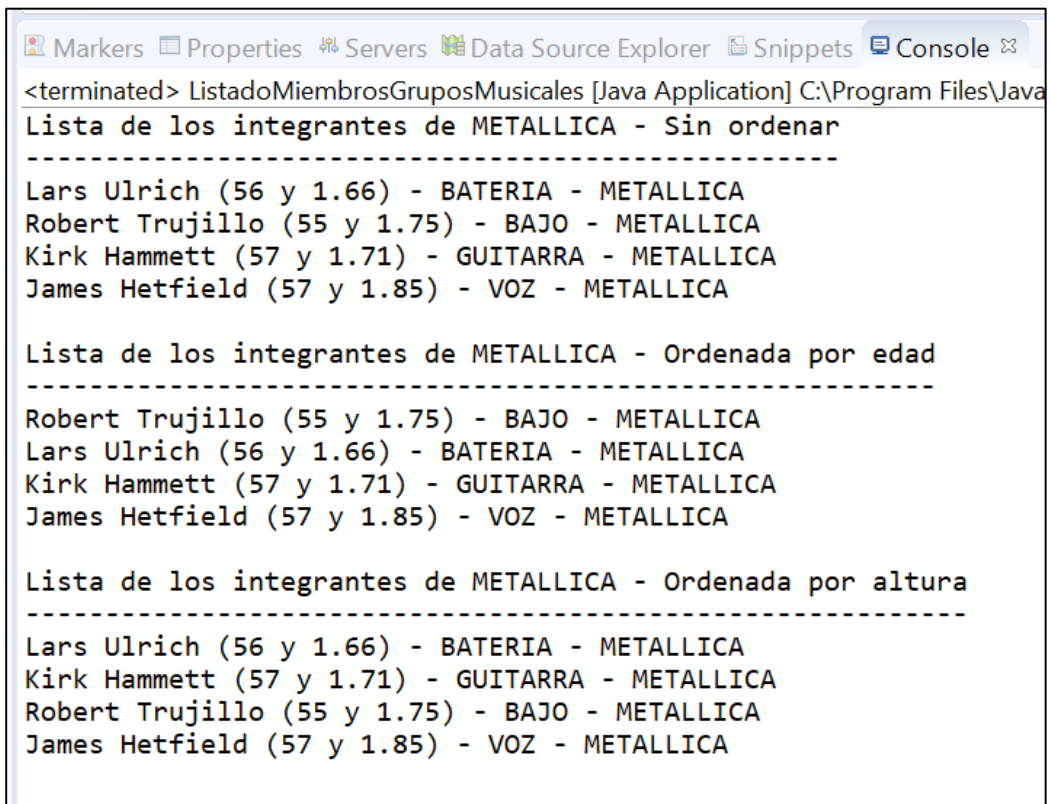
- **sobrescribir** el **método compare** dado por la **interface Comparator**. Este método debe ordenar las personas por el atributo altura de menor a mayor.

Por último, dentro del mismo paquete vamos a crearnos una nueva **clase ListadoMiembrosGruposMusicales** con un **método main** para poder crear las instancias de tipo Integrante, una lista con los integrantes de un grupo de música y lo vamos a ordenar por edad y por altura.

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

1. Creamos cuatro instancias de tipo Integrante con el constructor de parámetros. Puedes usar los siguientes datos:
 - Lars Ulrich, 56, 1.66, BATERIA, METALLICA;
 - Robert Trujillo, 55, 1.75, BAJO, METALLICA;
 - Kirk Hammett, 57, 1.71, GUITARRA, METALLICA;
 - James Hetfield, 57, 1.85, VOZ, METALLICA;
2. Creamos una List usando la implementación ArrayList para guardar los integrantes del grupo Metallica, es decir, creamos una lista donde vamos a guardar objetos de tipo Integrante y añadimos los cuatro integrantes del grupo con las instancias creadas anteriormente.
3. Tras crearla mostrar el listado de los integrantes del grupo sin orden, es decir, según se han introducido.
4. A continuación, vamos a ordenar el listado anterior por edad y lo mostramos por consola.
5. Por último, mostrar el listado de los integrantes ordenados por la altura.

Resultado esperado:



```
<terminated> ListadoMiembrosGruposMusicales [Java Application] C:\Program Files\Java
Lista de los integrantes de METALLICA - Sin ordenar
-----
Lars Ulrich (56 y 1.66) - BATERIA - METALLICA
Robert Trujillo (55 y 1.75) - BAJO - METALLICA
Kirk Hammett (57 y 1.71) - GUITARRA - METALLICA
James Hetfield (57 y 1.85) - VOZ - METALLICA

Lista de los integrantes de METALLICA - Ordenada por edad
-----
Robert Trujillo (55 y 1.75) - BAJO - METALLICA
Lars Ulrich (56 y 1.66) - BATERIA - METALLICA
Kirk Hammett (57 y 1.71) - GUITARRA - METALLICA
James Hetfield (57 y 1.85) - VOZ - METALLICA

Lista de los integrantes de METALLICA - Ordenada por altura
-----
Lars Ulrich (56 y 1.66) - BATERIA - METALLICA
Kirk Hammett (57 y 1.71) - GUITARRA - METALLICA
Robert Trujillo (55 y 1.75) - BAJO - METALLICA
James Hetfield (57 y 1.85) - VOZ - METALLICA
```

Actividad 3. Colecciones

Para este ejercicio vamos a intentar **poner en práctica el uso de alguna de las interfaces de Colecciones y ordenación.**

Para este ejercicio, vamos a utilizar la clase creada anteriormente Integrante, pero además, dentro del **paquete "EjerciciosColecciones"**, vamos a crearnos una nueva clase **GestionGruposMusicales** con un **método main** para poder crear las instancias de tipo Integrante con los integrantes de dos grupos, un mapa que guarde todos estos integrantes donde la clave va a ser las cuatro primeras letras del grupo + un número, además vamos a sacar un listado de todos los integrantes de los dos grupos y por último un listado de los integrantes del grupo AC/DC ordenados por edad.

Para ello, en esta clase, dentro del método main, vamos a realizar lo siguiente:

1. Creamos las instancias de tipo Integrante con el constructor de parámetros con los datos de todos los integrantes para los dos grupos de Metallica y AC/DC. Puedes usar los siguientes datos:
 - Lars Ulrich", 56, 1.66, "BATERIA", "METALLICA"
 - "Robert Trujillo", 55, 1.75, "BAJO", "METALLICA"
 - "Kirk Hammett", 57, 1.71, "GITARRA", "METALLICA"
 - "James Hetfield", 57, 1.85, "VOZ", "METALLICA"
 - "Brian Johnson", 73, 1.65, "VOZ", "AC/DC"
 - "Angus Young", 66, 1.57, "GITARRA", "AC/DC"
 - "Malcolm Young", 64, 1.60, "GITARRA", "AC/DC"
 - "Cliff Williams", 71, 1.70, "BAJO", "AC/DC"
 - "Phil Rudd", 67, 1.68, "BATERIA", "AC/DC"
2. Creamos una Map usando la implementación hashMap para guardar los integrantes de los dos grupos, es decir, creamos un mapa donde la clave va a ser de tipo String ("META1", "META2", ..., "ACDC1", "ACDC2", ...) y los valores o elementos serán de tipo Integrante con las instancias creadas anteriormente.

```
Map<String, Integrante> integrantesGruposMusicales = new HashMap<>();
integrantesGruposMusicales.put("META1", lars);
.....
.....
```

3. Tras crear el mapa anterior, vamos a sacar por consola un listado con todos los integrantes.

Pista: Ten en cuenta que puedes recoger en un Set todas las claves del Map usando el método `keySet()` y recorrer dicho Set para recuperar los elementos con el método `get(key)`.

4. Por último, debemos mostrar el listado de los integrantes de AC/DC ordenados por la altura

Pista: Puedes crear una lista (List) de tipo `Integrante` donde guardes los componentes de aquellos elementos del Map donde la clave sea "ACDC1", "ACDC2", etc, para ello puedes volver a recorrer el Map usando el Set anterior con todas las claves e ir añadiendo a la lista aquellos elementos donde coincida la clave. Una vez que tengas la lista, puedes ordenarla usando el método `sort()` como en el ejercicio anterior para que ordene por edad y después recorrer la lista y mostrarla por consola

Resultado esperado:

```
Markers Properties Servers Data Source Explorer Snippets Console
<terminated> GestionGruposMusicales [Java Application] C:\Program Files\Java\jre1.8.0_2
Listado de todos los integrantes
-----
Cliff Williams (71 y 1.7) - BAJO - AC/DC
Lars Ulrich (56 y 1.66) - BATERIA - METALLICA
Malcolm Young (64 y 1.6) - GUITARRA - AC/DC
Robert Trujillo (55 y 1.75) - BAJO - METALLICA
Angus Young (66 y 1.57) - GUITARRA - AC/DC
Kirk Hammett (57 y 1.71) - GUITARRA - METALLICA
Brian Johnson (73 y 1.65) - VOZ - AC/DC
Phil Rudd (67 y 1.68) - BATERIA - AC/DC
James Hetfield (57 y 1.85) - VOZ - METALLICA

Listado de todos los integrantes de AC/DC ordenados por edad
-----
Malcolm Young (64 y 1.6) - GUITARRA - AC/DC
Angus Young (66 y 1.57) - GUITARRA - AC/DC
Phil Rudd (67 y 1.68) - BATERIA - AC/DC
Cliff Williams (71 y 1.7) - BAJO - AC/DC
Brian Johnson (73 y 1.65) - VOZ - AC/DC
```

CONCEPTOS RELACIONADOS CON LA POO – PRÁCTICA II

EJERCICIOS DE EXCEPCIONES

Actividad 1. Excepciones

Vamos a crear una **aplicación que realice la división de dos números dados por consola y controle si la división es por 0 para que no se pare la ejecución y muestre un mensaje.**

Para este ejercicio, creamos un nuevo **paquete “EjerciciosExcepciones”** y dentro creamos una **clase** nueva llamada **Division** con un **método main** para poder realizar la operación.

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

1. Nos creamos tres variables, dos de tipo entero (numerador y denominador) y una de tipo double (resultado).
2. Creamos una instancia de tipo Scanner para solicitar los números de la operación por consola.
3. Solicitamos por consola los dos números para realizar la operación.
4. Una vez que tenemos los números vamos a realizar la división de ellos, pero tenemos que controlar la posibilidad de que se produzca una excepción al dividir.
5. Si la operación se ha podido realizar, sacamos por consola el resultado de la operación
6. Si no se ha podido realizar la operación, mostramos un mensaje de error con la excepción que se ha producido.

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 esta clase, al que denominaremos “**EjerciciosExcepciones**”.
2. Dentro del paquete “**EjerciciosExcepciones**” crearemos la **clase Division**.

```
public class Division
```

3. Dentro de la **clase Division** dentro del **método main()**, lo primero que vamos a hacer, es declarar las variables a utilizar.

```
int numerador, denominador;  
double resultado;
```

4. Tras la creación de las variables, creamos la instancia de tipo Scanner para poder solicitar e introducir los datos por consola.

```
Scanner consola = new Scanner(System.in);
```

5. Ahora, **solicitamos los datos** o los números para realizar la operación.

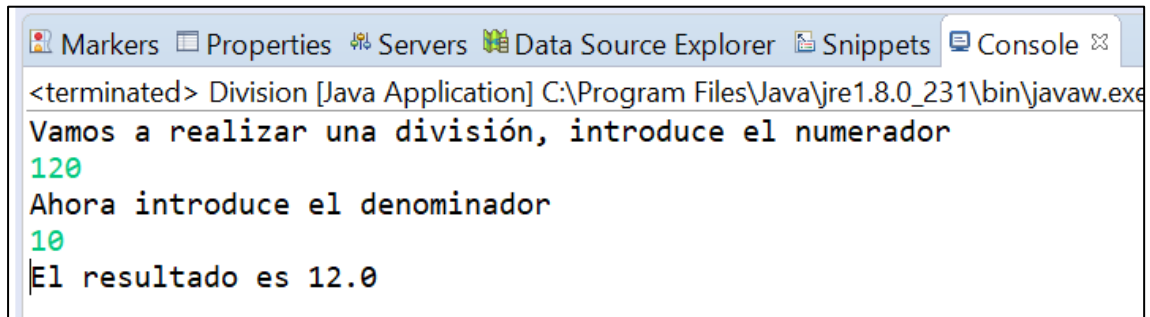
```
System.out.println("Vamos a realizar una división, introduce el numerador");  
numerador = consola.nextInt();  
System.out.println("Ahora introduce el denominador");  
denominador = consola.nextInt();
```

6. Una vez que tenemos los números, vamos a realizar la operación, pero teniendo en cuenta que esto puede fallar según los números que nos hayan introducido. Puedes usar el bloque try-catch

```
try {  
    resultado = numerador / denominador;  
    //Si va bien, mostramos el resultado  
    System.out.println("El resultado es " + resultado);  
} catch (Exception e) {  
    //Si salta una excepción al intentar dividir, mostramos el mensaje con el  
    error producido  
    System.out.println("No se ha podido realizar la división - " +  
        e.getMessage());  
}
```

Resultado esperado:

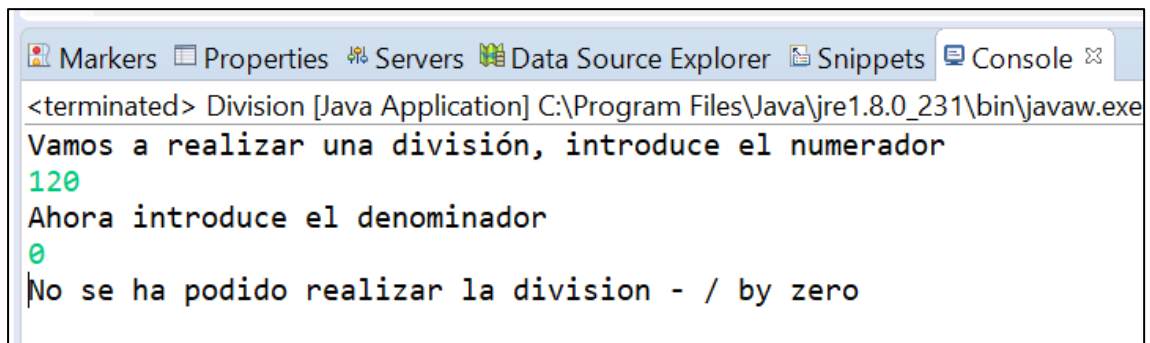
Caso de que se pueda realizar la operación de 120/10



The screenshot shows an IDE console window with the following content:

```
<terminated> Division [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe
Vamos a realizar una división, introduce el numerador
120
Ahora introduce el denominador
10
El resultado es 12.0
```

Caso en que no se puede realizar la operación de 120/0



The screenshot shows an IDE console window with the following content:

```
<terminated> Division [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe
Vamos a realizar una división, introduce el numerador
120
Ahora introduce el denominador
0
No se ha podido realizar la division - / by zero
```

Actividad 2. Excepciones

Para este ejercicio vamos a intentar **poner en práctica el uso y la creación de nuestras propias excepciones**.

Para ello, vamos a utilizar la **clase Integrante** creada en el ejercicio de Colecciones que tenemos en el **paquete “EjerciciosColecciones”** (podemos modificar la clase directamente o copiarnos esta clase en el paquete nuevo **“EjerciciosExcepciones”** para no crearla de nuevo).

Ahora, en el **paquete “EjerciciosExcepciones”** vamos a crear nuestra nueva **clase** de Excepciones llamada **ExcepcionesPropias** que va a extender de la **clase RuntimeException**.

Esta **clase ExcepcionesPropias** simplemente tendrá:

- Una **constante ERM_001** con el mensaje de error “Edad incorrecta!!”
- Un **constructor** para que nos permita crear el objeto con el mensaje que nosotros le pasemos como parámetro

Sobre la **clase Integrante** vamos a modificar el **método setEdad** para controlar y comprobar el valor a asignar a este atributo, de tal forma, que sólo nos permita asignar a este atributo un valor entre 18 y 99 (ambos incluidos).

Por último, dentro del mismo paquete vamos a crearnos una nueva **clase AltaIntegrante** con un **método main** para poder dar de alta un integrante.

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

1. Creamos una instancia de tipo Integrante con el constructor vacío.
2. Creamos una instancia de tipo Scanner para solicitar los datos por consola.
3. Solicitamos los datos por consola y se los vamos asignando a cada uno de los atributos de la instancia de tipo Integrante creada anteriormente.

Nota: Ten en cuenta que para la edad tenemos una validación del dato antes de asignarlo, de tal forma que si es incorrecto saltará o debería saltar nuestra excepción para indicarlo al usuario.

4. Una vez introducidos todos los datos, mostramos por consola los datos del nuevo integrante.

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 esta clase, al que denominaremos “**EjerciciosExcepciones**”.
2. Dentro del **paquete “EjerciciosExcepciones”** creamos la **clase ExcepcionesPropias** que **extienda** de la **RuntimeException**.

```
public class ExcepcionesPropias extends RuntimeException
```

3. Creamos la **constante con el mensaje de error**.

```
public static final String ERM_001 = "Edad incorrecta!!";
```

4. Creamos el **constructor de parámetros** para que reciba un mensaje y pueda crear el objeto con dicho mensaje que reciba como parámetro

```
public ExcepcionesPropias(String mensaje) {  
    super(mensaje);  
}
```

5. Ahora **modificamos** la **clase Integrante** (podemos modificar la clase creada anteriormente en el **paquete “EjerciciosColecciones”** (o copiarnos esta clase en este paquete y modificarla) para añadir en el **método setEdad(int edad)** la validación correspondiente, de tal forma que si el valor a asignar al parámetro es correcta (entre 18 y 99) la asignamos si no, lanzamos nuestra excepción con el mensaje de error creado.

```
public void setEdad(int edad) {  
    if (edad >= 18 && edad <= 99) {  
        this.edad = edad;  
    } else {  
        throw new ExcepcionesPropias(ExcepcionesPropias.ERM_001);  
    }  
}
```

6. Por último, creamos la **clase AltaIntegrante** con el **método main()**.

7. Creamos una **instancia** de tipo **Integrante** con el **constructor vacío**.

```
Integrante nuevoIntegrante = new Integrante();
```

8. Creamos una **instancia** de tipo **Scanner** para solicitar los datos por consola.

```
Scanner consola = new Scanner(System.in);
```

9. Solicitamos los **datos por consola y vamos asignando a cada uno de los atributos de la instancia de tipo Integrante** creada antes los valores introducidos anteriormente.

Nota: Ten en cuenta que para la edad tenemos una validación del dato antes de asignarlo, de tal forma que si es incorrecto saltará o debería saltar nuestra excepción para indicarlo al usuario.

```
System.out.println("Datos del nuevo integrante");
System.out.println("Nombre");
nuevoIntegrante.setNombre(consola.nextLine());
System.out.println("Edad");
try {
    nuevoIntegrante.setEdad(consola.nextInt());
} catch (Exception e) {
    System.out.println(e.getMessage());
}

System.out.println("Introduce su altura");
alt = teclado.next();
altura = Double.parseDouble(alt);

System.out.println("Instrumento");
nuevoIntegrante.setInstrumento(consola.next());

System.out.println("Grupo");
nuevoIntegrante.setGrupo(consola.next());
```

11. Una vez introducidos todos los datos, **mostramos por consola los datos** del nuevo integrante.

```
System.out.println("Datos introducidos para el nuevo integrante");
System.out.println(nuevoIntegrante);
```

Resultado esperado:

Caso en que la **edad es correcta**.

```
Markers Properties Servers Data Source Explorer Snippets Console
<terminated> AltaIntegrante [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\jav
Datos del nuevo integrante
Nombre
Klaus Meine
Edad
73
Altura
1,63
Instrumento
VOZ
Grupo
SCORPIONS
|
Datos introducidos para el nuevo integrante
Klaus Meine (73 y 1.63) - VOZ - SCORPIONS
```

Caso en el que la **edad no es correcta**.

```
Markers Properties Servers Data Source Explorer Snippets Console
<terminated> AltaIntegrante [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\jav
Datos del nuevo integrante
Nombre
Klaus Meine
Edad
15
Edad incorrecta!!
Altura
1,63
Instrumento
VOZ
Grupo
SCORPIONS
|
Datos introducidos para el nuevo integrante
Klaus Meine (0 y 1.63) - VOZ - SCORPIONS
```

CONCEPTOS RELACIONADOS CON LA POO – PRÁCTICA II

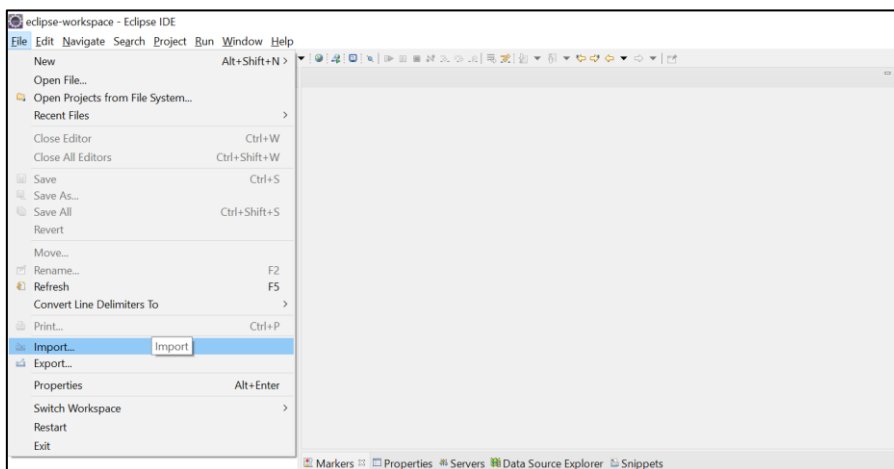
SOLUCIONES A LAS ACTIVIDADES

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.

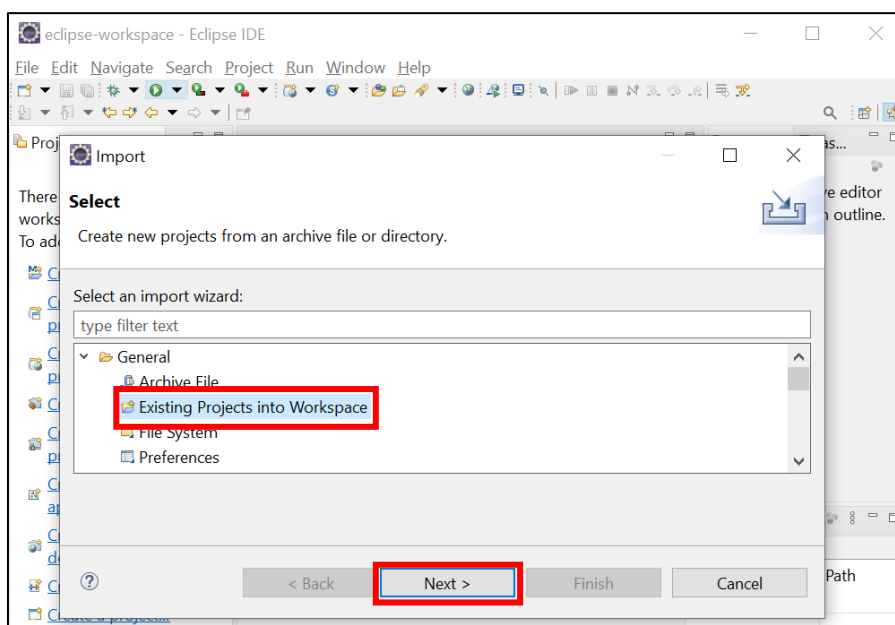
Para ello descomprime en tu directorio local el fichero .zip, que encontrarás también en la sección inferior **contenido descargable**, con el nombre **“Solución de las prácticas”**.

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

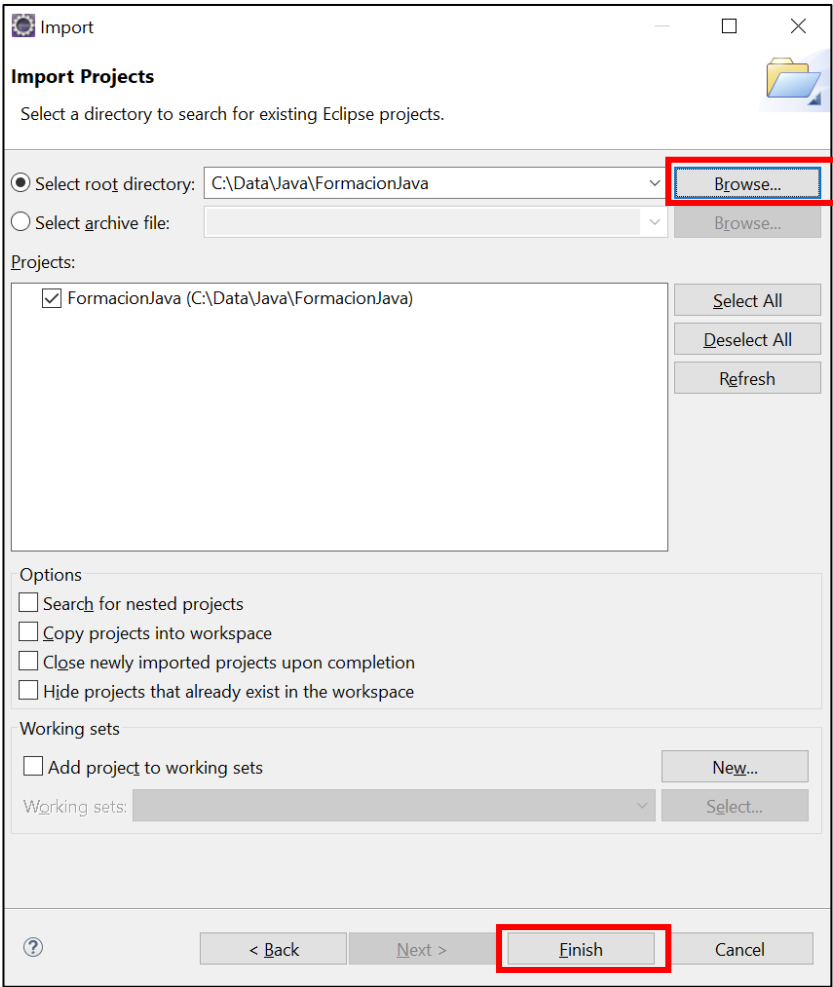
1. Abre en el **menú File → Import**.



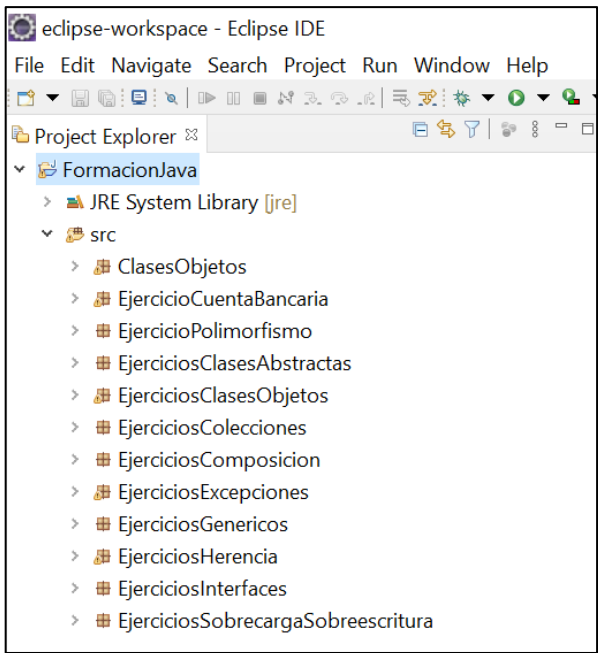
2. Aparece la siguiente ventana en la que **seleccionamos** la **opción “Existing Projects into Workspaces”** y **pulsamos “Next”**.



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