DAW
Desarrollo de Aplicaciones Web
2º Curso

DWES
Desarrollo Web Entorno Servidor

UD 4
Iniciación a Java
(Parte 2)

IES BALMIS
Dpto Informática
Curso 2025-2026
Versión 1 (11/2025)

UD4 – Iniciación a Java

ÍNDICE

13. Tipo Date

Las fechas en JAVA suelen almacenarse en variable de tipo Date, pero para mostrarse como un String se deben convertir asignando el formato.

En el siguiente ejemplo se puede comprobar cómo se crea una fecha con el día y hora actual, y como se puede almacenar en un String y mostrarse.

DateToISO

```
import java.text.SimpleDateFormat;
import java.util.Date;

public class DateToISO {
  public static void main(String[] args) {
    Date fecha;
    fecha = new Date();

    String strFecha = new SimpleDateFormat("dd/mm/YYYY").format(fecha);
    System.out.println(strFecha);

    String strHora = new SimpleDateFormat("HH:mm:ss").format(fecha);
    System.out.println(strHora);

    // ISO 8601 Format
    String strFechaISO =
        new SimpleDateFormat("YYYY-MM-dd'T'HH:mm:ss'Z'").format(fecha);
    System.out.println(strFechaISO);
  }
}
```

Si deseamos crear una variable de tipo Date a partir de una cadena String con una fecha, podemos usar la clase Calendar.

StringToDate

```
import java.util.Calendar;
import java.util.Date;

public class StringToDate {

   public static void main(String[] args) {
        Date fecha;

        String strFecha = "05/11/2028";
        System.out.println(stringToDate(strFecha));

        String strFechaMySQL = "2028-11-16";
        System.out.println(stringToDateMySQL(strFechaMySQL));

        String strFechaISO = "2028-10-25T15:27:57Z";
        System.out.println(stringToDateISO(strFechaISO));
    }
```

```
    public static Date stringToDateMySQL(String strFecha) {
        // ***********************
        // strFecha = 'AAAA-MM-DD'
        // ***********************

        // year - AAAA
        // month - the month between 0-11.
        // date - the day of the month between 1-31.
        Calendar calendar = Calendar.getInstance();
        calendar.set(Integer.parseInt(strFecha.substring(0, 4)),
                Integer.parseInt(strFecha.substring(5, 7)) - 1,
                Integer.parseInt(strFecha.substring(8, 10)),
                0,0,0);
        return calendar.getTime();
    }

    public static Date stringToDate(String strFecha) {
        // ***********************
        // strFecha = 'dd/mm/AAAA'
        // ***********************

        // year - AAAA
        // month - the month between 0-11.
        // day - the day of the month between 1-31.
        Calendar calendar = Calendar.getInstance();
        calendar.set(
                Integer.parseInt(strFecha.substring(6, 10)),
                Integer.parseInt(strFecha.substring(3, 5)) - 1,
                Integer.parseInt(strFecha.substring(0, 2)),
                0,0,0
                );
        return calendar.getTime();
    }


    public static Date stringToDateISO(String strFecha) {
        // ***********************
        // strDia = 'AAAA-MM-DD';
        // strHora = 'HH:mm:ss';
        // strFecha = strDia+'T'+strHora+'Z';
        // ***********************
        Calendar calendar = Calendar.getInstance();
        calendar.set(Integer.parseInt(strFecha.substring(0, 4)),
                Integer.parseInt(strFecha.substring(5, 7)) - 1,
                Integer.parseInt(strFecha.substring(8, 10)),
                Integer.parseInt(strFecha.substring(11, 13)),
                Integer.parseInt(strFecha.substring(14, 16)),
                Integer.parseInt(strFecha.substring(17, 19)));
        return calendar.getTime();

    }

}
```

14. Recorrer Arrays y Colecciones

Los Arrays y Colecciones son las estructuras más usadas para el almacenamiento en memoria, pero hay varias dependiendo de cómo se desee almacenar o recorrer.

Los Arrays tienen un tamaño estático definido al crearse, mientras que las listas aumentan de tamaño mientras se van añadiendo elementos.

Lo primero que debemos conocer son las diferentes formas que tenemos de recorrer un conjunto de datos. En el siguiente ejemplo usaremos la clase ArrayList, pero podríamos utilizar estos recorridos en la mayoría del resto de clases.

RecorrerArrayList

```
// Carga de datos
ArrayList<String> lista = new ArrayList<>();
lista.add("uno");
lista.add("dos");

// Recorrer con for y contador
for (int i=0; i<lista.size(); i++) {
    String str = lista.get(i);
    System.out.print(str+" ");
}
System.out.println();

// Recorrer con iterator
Iterator<String> iter   = lista.iterator();
while (iter.hasNext()) {
    String str = iter.next();
    System.out.print(str+" ");
}
System.out.println();

// Recorrer con for de contenido
for (String str: lista) {
    System.out.print(str+" ");
}
System.out.println();

// Recorrer con foreach y una expresión lambda
lista.forEach((str) -> {
    System.out.print(str+" ");
});
System.out.println();
```

```
// Recorrer con stream y map
List<String> collect1 = lista.stream()
                             .map(String::toUpperCase)
                             .collect(Collectors.toList());
System.out.println(collect1); // [UNO, DOS]

// Recorrer con stream y map
List<String> collect2 = lista.stream()
                  .map(String::toUpperCase)
                  .collect(Collectors.toCollection(ArrayList::new));
System.out.println(collect2); // [UNO, DOS]

// Recorrer con stream y map
List<Integer> num = Arrays.asList(1,2,3,4,5); // Carga de datos


List<Integer> collect3 = num.stream()
                            .map((n) -> n * 2)
                            .collect(Collectors.toList());
System.out.println(collect3); //[2, 4, 6, 8, 10]

// Recorrer con stream y map
List<String> collect4 = num.stream()
                           .map((n) -> n.toString())
                           .collect(Collectors.toList());
System.out.println(collect4); //["1", "2", "3", "4", "5"]
```

Las **expresiones lambda** suelen utilizarse para lo siguiente:

- Como **argumentos** que son pasados a otras funciones de orden superior.
- Para **construir el resultado** de una función de orden superior que necesita retornar una función.

Una expresión lambda puede ser sintácticamente más simple que una función nombrada, si solo se utiliza una vez o un número limitado de veces.

Las funciones lambda son muy comunes en la **programación funcional** y otros lenguajes con funciones de primera clase.

Las expresiones Lambda simplifican mucho el código, pero también lo hacen menos entendible en algunas ocasiones.

Veamos un ejemplo donde se muestran dos ejemplos para obtener el mismo resultado, uno con una notación Lambda integrada y el otro con un Lambda Consumer:

Lambda

```
import java.util.ArrayList;
import java.util.function.Consumer;
```

```
public class Lambda {
    public static void main(String[] args) {

        ArrayList<Integer> numbers = new ArrayList<Integer>();
        numbers.add(5);
        numbers.add(9);
        numbers.add(8);
        numbers.add(1);

        // Lambda Direct
        numbers.forEach((n) -> {
            System.out.print(n + " ");
        });
        System.out.println();

        // Lambda Consumer
        Consumer<Integer> method = (n) -> {
            System.out.print(n + " ");
        };
        numbers.forEach(method);
        System.out.println();
    }
}
```

Las conversiones entre tipos de datos son necesarias para poder procesar la información adecuadamente.

Veamos a continuación como convertir de **String a Array** y de **Array a String:**

ConversionesArrays

```
  //*************************
  // String -> Arrays
  String   strColores;
  String[] arrColores;

  strColores="rojo,amarillo,verde,azul";
  arrColores=strColores.split(",");
  for (String color: arrColores) {
      System.out.print(color+"-");
  }
  System.out.println();

  strColores="rojo|amarillo|verde|azul";
  arrColores=strColores.split("\\|");
  for (String color: arrColores) {
     System.out.print(color+"-");
  }
  System.out.println();

  //*************************
  // Arrays -> String
  String[] arrColores2={"rojo","amarillo","verde","azul"};
  String   strColores2=String.join("|",arrColores2);
  System.out.println(strColores2);
```

Las Colecciones más simples son las listas que almacenan un conjunto de un mismo tipo dato. Usaremos generalmente **ArrayList** ya que **List** es un Interface, no una clase de datos.

Veamos unos ejemplos de convertir Arrays en ArrayList y vicersa:

ConversionesArrays

```
//*************************
// Arrays -> ArrayList
String[]          arrCiudades  = {"Alicante","Valencia","Castellón"};
ArrayList<String> listCiudades =
                    new ArrayList<String>(Arrays.asList(arrCiudades));

listCiudades.forEach((s) -> System.out.print(s+"-"));
System.out.println();

//*************************
// ArrayList -> Arrays
ArrayList<Integer>  listNumeros = new ArrayList<Integer>();
Collections.addAll(listNumeros, 1, 2, 3, 4);
System.out.println(listNumeros.toString());

Integer[] arrNumeros = new Integer[listNumeros.size()];
listNumeros.toArray(arrNumeros);
for (Integer num: arrNumeros) {
   System.out.print(num+"-");
}
System.out.println();
```

Las Colecciones que implementan List son diferentes de las que implementan Set.

La interfaz List:

- es una interfaz secundaria de Collection
- permite almacenar la colección ordenada
- es una colección **ordenada** de objetos en los que se permite almacenar valores **duplicados**
- conserva el orden de inserción
- permite el acceso posicional y la inserción de elementos.

La interfaz Set:

- es una interfaz Collection extendida
- es una colección **desordenada** de objetos en los que **no se pueden almacenar valores duplicados**

	List	Set
Acceso posicional	SI	NO
Duplicados	SI	NO
Orden	Ordenados	Depende de la implementación

La mayoría de métodos para el tratamiento de los datos existen igual en las dos clases. Vemos unos ejemplos:

ConversionesArrays

```
// +------------+--------------+--------------+
// | interface  | -> interface | -> class     |
// +------------+--------------+--------------+
// | Collection |    List      | ArrayList    |
// | Collection |    Set       | TreeSet      |
// +------------+--------------+--------------+

ArrayList<String> aList = new ArrayList<>();
aList.add("verde");
aList.add("azul");
aList.add("rojo");
aList.forEach((s) -> System.out.print(s+"-"));
System.out.println();

TreeSet<String> tSet = new TreeSet<>();
tSet.add("verde");
tSet.add("azul");
tSet.add("rojo");
tSet.forEach((s) -> System.out.print(s+"-"));
System.out.println();
```

Las clases de datos que implementan el interface **Map** permiten almacenarlos mediante una referencia o clave, es decir, mediante una combinación de **clave** (Key) y **valor** (value).

Las dos clases más usadas son HashMap y Properties. Veamos unos ejemplos:

ConversionesArrays

```
// +------------+--------------+
// | interface  | -> class     |
// +------------+--------------+
// | Map        | HashMap      |
// | Map        | Properties   |
// +------------+--------------+

// HashMap
HashMap<String,Integer> hMap = new HashMap<>();
hMap.put("uno",1);
hMap.put("dos",2);
hMap.put("tres",3);

System.out.println("Keys..: " + hMap.keySet());
System.out.println("Values: " + hMap.values());

hMap.forEach((key,value) -> System.out.println(key+"-"+value));

for (String key: hMap.keySet()) {
   System.out.println(key+"-"+hMap.get(key));
}
System.out.println();
```

```
// Properties
Properties prop = new Properties();
prop.setProperty("nombre","Alberto");
prop.setProperty("apellido","Gómez");
prop.setProperty("dni","21578488");

prop.forEach((key,value) -> System.out.println(key+"-"+value));

System.out.println("Keys..: " + prop.keySet());
System.out.println("Values: " + prop.values());

for (Object key: prop.keySet()) {
  System.out.println(key.toString()+"-"+
                     prop.getProperty(key.toString()));
}
System.out.println();
```

15. Record Class

En ocasiones, necesitamos almacenar diferentes tipos de datos en una variable.

Podríamos crear una clase y definir sus métodos, pero también tenemos la clase **Record** que es mucho más simple aunque tiene limitaciones:

- Los Record son clases de datos inmutables que solo requieren el tipo y el nombre de los campos
- Solo se crea con el constructor y luego se lee su contenido
- No tiene ni getters ni setters
- Se puede modificar el constructor
- Se pueden crear métodos pero estos no pueden actualizar los datos del Record

Por lo tanto, la principal diferencia entre **Class** y **Record** en Java es que un registro tiene el propósito principal de almacenar datos, mientras que una clase define métodos para el tratamiento dinámico de datos.

El uso de **Record** es ideal para cuando queremos crear clases de datos cuyos datos no requieran ser modificados, sino simplemente leídos para ser mostrados o procesados.
En caso de necesitar modificar los datos usaremos **Class** con los métodos setters correspondientes.

Un **Data Transfer Object (DTO)** es un patrón de diseño de objetos que contiene las propiedades del objeto y sirven únicamente para transportar datos. Los **DTO** suelen implementarse con **Record**.

Veamos un ejemplo de definición de **Record** para almacenar los datos **nombre**, **apellidos** y **edad** de una persona:

PersonaDTO → Record (PersonaDTODefault)

```
public record PersonaDTODefault (
    String nombre,
    String apellidos,
    int edad)
{
}
```

En los Record se puede modificar el constructor por defecto y añadir métodos que no modifiquen los datos.

Veamos un ejemplo:

PersonaDTO → Record (PersonaDTO)

```
public record PersonaDTO (
    String nombre,
    String apellidos,
    int edad)
{
    public PersonaDTO(String nombre, String apellidos, int edad) {
        this.nombre = nombre.toUpperCase();
        this.apellidos = apellidos.toUpperCase();
        this.edad = edad;
    }

    public String getNombreLower() {
        return this.nombre.toLowerCase();
    }

    public boolean isMayorEdad() {

        return (this.edad >= 18);
    }
}
```

Ahora podremos utilizar estos **Record** para crear estructuras de datos y mostrar su contenido:

PersonaDTO

```
  // PersonaDTO con constructor por defecto
  PersonaDTODefault p = new PersonaDTODefault("pepe", "perez", 20);

  System.out.println("Nombre...: " + p.nombre());
  System.out.println("Apellidos: " + p.apellidos());
  System.out.println("Edad.....: " + p.edad());
  System.out.println();

  // PersonaDTO con constructor definido por usuario
  PersonaDTO p1 = new PersonaDTO("pepe", "perez", 20);
  PersonaDTO p2 = new PersonaDTO("pepe", "perez", 20);

  System.out.println(p1);

  System.out.println("Nombre...: " + p1.nombre());
  System.out.println("Apellidos: " + p1.apellidos());
  System.out.println("Edad.....: " + p1.edad());

  if (p1.isMayorEdad()) {
     System.out.println(p1.getNombreLower()+" es mayor de edad");
  } else {
     System.out.println(p1.getNombreLower()+" es menor de edad");
  }

  if (p1.equals(p2)) {
     System.out.println("p1 es igual a p2");
  } else {
      System.out.println("p1 NO es igual a p2");
  }
```

16. Apache Commons Libraries

Los desarrolladores están creando continuamente nuevas funcionalidades para Java. En muchos casos, estas librerías son libres y gratuitas.

Apache Commons Libraries son un conjunto de librerías que nos añaden funcionalidad a muchas de las clases que ya tenemos.

Apache Commons Libraries
https://commons.apache.org/

Nosotros vamos a destacar en esta parte de iniciación las de Apache Commons Lang

Apache Commons Lang
https://commons.apache.org/proper/commons-lang/javadocs/api-release/index.html

Como ejemplo podemos explorar los métodos disponibles en las clases:

- SystemUtils
- StringUtils
- NumberUtils
- BooleanUtils
- DateUtils
- DateFormatUtils
- ArrayUtils
- RandomUtils
- WordUtils

En el siguiente ejemplo donde usaremos la clase **StringUtils** deberemos añadir la librería **commons-lang3-3.12.0.jar**

Como la clase es **static**, llamaremos a los métodos sin instanciar un objeto:

ApacheCommons

```
String saludo="BIENVENIDO";
System.out.println(StringUtils.reverse(saludo));
System.out.println(StringUtils.left(saludo, 4));
System.out.println(StringUtils.contains(saludo,"I"));
System.out.println(StringUtils.contains(saludo,"J"));
System.out.println(StringUtils.repeat("ali",5));
System.out.println(StringUtils.replace(saludo, "EN", "en"));
System.out.println(StringUtils.remove(saludo,"EN"));
```