Fundamentos de Programación

Tratamiento de datos leídos desde ficheros CSV

Autor: Mariano González. **Revisores**: José Mª Luna, Toñi Reina. **Última modificación**: 14/3/2023

Al leer las líneas de un fichero de texto, estas se leen como cadenas de caracteres. Si los datos que contiene el fichero representan valores numéricos, fechas, horas, o en general valores que no son cadenas, hay que convertirlos al tipo correspondiente. En este documento se muestra cómo realizar esta transformación con algunos de los tipos de datos más habituales.

En cada caso, suponemos que tenemos almacenada en una variable de tipo `String` el valor que se ha leído del fichero, y que hemos eliminado los caracteres en blanco iniciales y finales de dicha cadena aplicando el método *trim*.

Nota: es recomendable crear métodos auxiliares para realizar las conversiones.

Valores numéricos

Para convertir una cadena que contiene dígitos numéricos en un objeto de tipo `Integer` o `Double`, usamos el método *valueOf* del tipo envoltura:

```
Integer numero = Integer.valueOf(cadena);
Double temperatura = Double.valueOf(cadena);
```

También podemos utilizar el método *parse* del tipo envoltura correspondiente:

```
Integer numero = Integer.parseInt(cadena);
Double temperatura = Double.parseDouble(cadena);
```

Caracteres

Para convertir una cadena que contiene un único carácter en un objeto de tipo `Character`, usamos el método *charAt* del tipo String:

```
Character letra = cadena.charAt(0);
```

Fechas y horas

Para convertir cadenas que representan fechas y horas en objetos de tipos `LocalDate`, `LocalTime` o `LocalDateTime`, se utiliza el método *parse*. Este método recibe dos parámetros: la cadena a convertir y un objeto de tipo `DateTimeFormatter` que representa el formato que tiene la fecha u hora en la cadena.

La cadena a convertir contiene los valores para día, mes, año, hora, minutos y segundos, además de otros caracteres que actúan como separadores. El objeto de tipo

`DateTimeFormatter` se construye a partir de una cadena de formato, que es una cadena formada por unos caracteres que indican lo que representa cada valor de la cadena a convertir: 'd' representa el día, 'M' el mes, 'y' el año, 'h' la hora, 'm' los minutos y 's' los segundos, además de los caracteres separadores mencionados. La referencia completa de códigos se puede ver en https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html.

Suponiendo que la fecha viene expresada en forma "día/mes/año", la hora en la forma "hora:minutos:segundos" y la fecha/hora en la forma "día/mes/año-hora:minutos:segundos", la conversión se haría de la siguiente forma:

```
LocalDate fecha = LocalDate.parse(cadena,
      DateTimeFormatter.ofPattern("d/M/y"));
LocalTime hora = LocalTime.parse(cadena,
      DateTimeFormatter.ofPattern("h:m:s"));
LocalDateTime fechaHora = LocalDateTime.parse(cadena,
      DateTimeFormatter.ofPattern("d/M/y-h:m:s"));
```

Es posible que la fecha o la hora vengan en el fichero en forma de varios números separados por comas. Por ejemplo, el día, el mes y el año, o la hora, los minutos y los segundos. En este caso, al trocear la línea se obtiene un trozo para cada número, y para crear la fecha o la hora a partir de ellos se haría de la siguiente forma:

```
Integer dia = Integer.valueOf(cadena1);
Integer mes = Integer.valueOf(cadena2);
Integer anyo = Integer.valueOf(cadena3);
LocalDate fecha = LocalDate.of(anyo, mes, dia);

Integer hora = Integer.valueOf(cadena1);
Integer minutos = Integer.valueOf(cadena2);
Integer segundos = Integer.valueOf(cadena3);
LocalTime hora = LocalTime.of(hora, minutos, segundos);
```

Valores que representan cantidades de tiempo

Para almacenar una duración (por ejemplo, el tiempo empleado por un atleta en una carrera, o el tiempo que dura una película), Java proporciona el tipo `Duration`. Normalmente, el valor que viene en el fichero es de tipo numérico, y representa un número de horas, minutos o segundos. En este caso, hay que convertir este valor numérico en un objeto de tipo Duration, usando un método u otro según lo que represente el número leído del fichero:

```
Integer s = Integer.valueOf(cadena);
Duration d = Duration.ofSeconds(s);

Integer m = Integer.valueOf(cadena);
Duration d = Duration.ofMinutes(m);

Integer h = Integer.valueOf(cadena);
Duration d = Duration.ofHours(h);
```

De manera similar, podemos tener en el fichero un número que represente un periodo de tiempo en días, meses o años, y queremos almacenarlo en un objeto de tipo `Period`, que representa un periodo de tiempo. Se haría de forma análoga:

```
Integer d = Integer.valueOf(cadena);
Period p = Period.ofDays(d);

Integer m = Integer.valueOf(cadena);
Period p = Period.ofMonths(m);

Integer a = Integer.valueOf(cadena);
Period p = Period.ofYears(a);
```

Valores booleanos

Los valores booleanos pueden aparecer de varias formas en el fichero. Si aparecen en la forma "true" y "false", usamos el método *parseBoolean* para convertir la cadena a los literales de tipo `Boolean`, que son `true` y `false`:

```
Boolean b = Boolean.parseBoolean(cadena);
```

Este método devuelve `true` si recibe como parámetro la cadena "true" (independientemente del uso de mayúsculas o minúsculas), y `false` en cuaquier otro caso.

A veces, los valores aparecen de otra forma, por ejemplo "1" y "0", "Sí" y "No", etc. Si queremos convertir estos valores a los literales `true` y `false`, hemos de usar una estructura condicional. Por ejemplo, si los valores son "Sí" y "No", haremos lo siguiente:

```
Boolean b = false;
if (cadena.equals("Sí")) {
    b = true
}
```

Valores de tipos enumerados

Para convertir una cadena en un valor de un tipo enumerado, usamos el método *valueOf* del tipo enumerado:

```
Categoria cat = Categoria.valueOf(cadena);
```

Para que esta conversión funcione, la cadena debe tener el mismo formato que el valor enumerado. Por ejemplo, podemos convertir la cadena "PRINCIPAL" en el valor enumerado Categoria.PRINCIPAL. Si la cadena fuese, por ejemplo, "Principal", entonces habría que pasarla a mayúsculas antes de realizar la conversión.

Valores que son objetos

Puede ocurrir que una propiedad del objeto sea otro objeto de un tipo definido por nosotros. Por ejemplo, el tipo `Avistamiento` tiene una propiedad ubicación que es de tipo `Coordenadas`, que a su vez tiene dos propiedades latitud y longitud de tipo `Double`. Si leemos

del fichero dos cadenas, cadena1 y cadena2, con los valores de la latitud y la longitud respectivamente, hemos de convertir cada cadena a un valor de tipo `Double`, y usar el constructor con parámetros del tipo `Coordenadas` para crear el objeto:

```
Double latitud = Double.valueOf(cadena1);
Double longitud = Double.valueOf(cadena2);
Coordenadas ubicacion = new Coordenadas(latitud, longitud);
```

Valores que son listas

La cadena puede estar formada por varios elementos separados por un carácter dado. En este caso, hay que trocear la cadena en varias subcadenas y construir una lista o conjunto con ellas.

Por ejemplo, supongamos que tenemos una cadena formada por varios nombres separados por comas: “Ana, Juan, María”. Para crear una lista de cadenas con los tres nombres, creamos un método auxiliar que realice el parseo de la cadena:

```
List<String> nombres = parsearNombres(cadena);

private static List<String> parsearNombres(String nombres) {
	String[] trozos = nombres.split(",");
	List<String> res = new ArrayList<>();
	for (String t: trozos) {
		res.add(t.trim());
	}
	return res;
}
```

Puede ocurrir que los elementos de la lista no sean cadenas; en ese caso, una vez troceada la cadena, hay que convertir cada trozo al tipo correspondiente antes de añadirlo a la lista o conjunto.

Por ejemplo, supongamos que tenemos una cadena formada por varios números separados por comas: “21, 19, 20”. Para crear una lista de enteros con los tres números, creamos un método auxiliar que realice el parseo de la cadena y convierta cada trozo al tipo entero:

```
List<Integer> numeros = parsearNumeros(cadena);

private static List<Integer> parsearNumeros(String numeros) {
	String[] trozos = numeros.split(",");
	List<Integer> res = new ArrayList<>();
	for (String t: trozos) {
		res.add(Integer.valueOf(t.trim());
	}
	return res;
}
```

Transformaciones previas

En algunas ocasiones hay que realizar una transformación previa de la cadena antes de convertirla. Esto puede suceder cuando la cadena contiene caracteres que no son parte del valor, o cuando no está escrita de la forma que deseamos. Veamos algunos ejemplos.

Caso 1: una temperatura está expresada en la forma "10 ºC", y hay que eliminar la subcadena " ºC" para quedarse con la parte numérica de la cadena. Para ello, disponemos del método *replace* del tipo `String`, que permite sustituir un grupo de caracteres por otro, o eliminarlo si lo sustituimos por una cadena vacía. Sería así:

```
Integer temperatura = Integer.valueOf(cadena.replace(" ºC", ""));
```

Caso 2: un valor enumerado está escrito en minúsculas en el fichero, y hay que convertirlo a mayúsculas, ya que los valores enumerados se escriben en mayúsculas según la convención de Java. Para ello, disponemos del método *toUpperCase* del tipo `String`. Sería así:

```
Categoria cat = Categoria.valueOf(cadena.toUpperCase());
```