Programación 2

Strings y Entrada/Salida Básica en Java

Profesores:
Ismael Figueroa - ifigueroap@gmail.com
Eduardo Godoy - eduardo.gl@gmail.com

Strings en Java

Los strings, o en español *cadenas*, representan secuencias de caracteres. En Java los strings siempre son objetos de clase `String`. Existen muchas maneras de crear un String en Java[1]:

- Asignar directamente un string literal
- Usar alguno de los constructores en base a arreglos de bytes o de caracteres.
- Usar alguno de los constructores en base a otros strings y clases similares.

Las operaciones básicas son la concatenación, con el operador +, y también conocer el largo del string, con la propiedad `length`

[1] `https://docs.oracle.com/javase/8/docs/api/java/lang/String.html`

Inmutabilidad de los Strings

Inmutables

Por eficiencia, en Java los objetos `String` son **inmutables**. Una vez creados, su valor no puede cambiar.

```
public class Hola {
  public static void main(String[] args) {
    String s1 = "Hola!";
    String s2 = s1;
    System.out.println(s1 + " / " + s2);

    s1 = "Ayudaaa";
    System.out.println(s1 + " / " + s2);
  }
}
```

Igualdad de Strings

¿Qué imprime por pantalla este programa?

```
public class Hola {
  public static void main(String[] args) {
    String s1 = new String("Hola!");
    String s2 = "Hola!";
    String s3 = s2;

    if(s1 == s2) {
      System.out.println("JAVA");
    }

    if(s2 == s3) {
      System.out.println("ES LO MEJOR");
    }
  }
}
```

Igualdad de Strings

Igualdad por referencia o identidad

En Java el operador == compara la ***igualdad de referencias***: ¿están las dos referencias apuntando a un mismo objeto? En el fondo este concepto está demasiado ligado a la *identidad del objeto*.

Igualdad por valor

En general es más práctico programar con la ***igualdad por valor***. Para esto se debe utilizar el método `equals` o bien `equalsIgnoreCase`.

```
if(s1.equals(s2)) {
  System.out.println("JAVA ES LO MEJOR");
}
```

Comparación de Strings

Además de querer ver la igualdad entre strings, muchas necesitamos:

- Ver si un string comienza o termina con determinado prefijo/sufijo, con los métodos `startsWith` y `endsWith`
- Comparar dos strings para ver cuál es "mayor.º "menor", según orden lexicográfico, con los métodos `compareTo` y `compareToIgnoreCase`
- Determinar si un string calza con una *expresión regular*, con el método `matches`

Strings de Formato

Como alternativa a la concatenación, donde es algo engorroso abrir y cerrar comillas para cada elemento, en Java se pueden crear *strings de formato*, que trabajan de forma similar a `printf`.

```
public class Hola {
  public static void main(String[] args) {
    String nombre = "Juanito";
    int edad = 32;
    // Concatenacion
    String msg1 = "Hola soy " + nombre + " y tengo " + edad + " años";

    // String de formato
    String msg2 = String.format(
        "Hola soy %s y tengo %d años", nombre, edad);

    System.out.println(msg1);
    System.out.println(msg2);
  }
}
```

Conversión de String a número

Java provee 5 clases numéricas, que representan como objetos los tipos numéricos primitivos:

- La clase `Byte`
- La clase `Integer`
- La clase `Double`
- La clase `Float`
- La clase `Long`
- La clase `Short`

Cada una de estas clases tiene el método `valueOf` que convierte un string, asumiendo que tiene un número válido, en un objeto numérico.

Manipulación de Caracteres

A veces es necesario considerar los caracteres individuales de un string. Para esto tenemos los siguientes métodos:

- `charAt`: retorna el caracter en la posición dada
- `indexOf`/`lastIndexOf`: retorna la primera/última posición del caracter en el string, o -1 si es que no se encuentra
- `contains`: retorna verdader o falso según el caracter dado está o no en el string
- `substring`: retorna el string entre las posiciones indicadas

Otros métodos de String

docs.oracle.com/javase/8/docs/api/java/lang/String.html

Flujos de Entrada/Salida

La programación de entrada/salida en Java se basa en la manipulación de *flujos de entrada/salida*

Un flujo de entrada/salida representa una fuente de entrada o un destino de salida. Este concepto general puede representar diversos tipos de fuentes y destinos: archivos, dispositivos, otros programas, arreglos, etc.

Los flujos soportan distintos tipos de datos: bytes, tipos de datos primitivos, u objetos. No obstante, *todos los flujos tienen el mismo modelo de programación*

Modelo de Flujos Entrada/Salida

Para el programador todos los flujos de entrada/salida funcionan de manera similar: los flujos son construcciones secuenciales de datos.

- Un *flujo de entrada* se usa en un programa para *leer datos desde la fuente*, de a un elemento cada vez.
- Un *flujo de salida* se usa para *escribir datos en un destino*, también de un elemento a la vez.

Tipos Básicos de Flujos

- **Bytes Streams**: se usan para trabajar directamente con bytes, o sea items de 8 bits. Se usan en general para trabajar con datos/archivos binarios.

- **Character Streams**: se usan para trabajar con caracteres de texto, con soporte Unicode directamente implementado. Es lo más conveniente para trabajar con datos/archivos de texto.

Buffered Streams

El uso directo de los flujos de entrada o salida es poco eficiente porque cada invocación se delega directamente al sistema operativo, elemento por elemento. Java implementa un mecanismo de *buffered streams* que acumula datos de entrada/salida en un área de memoria denominada *buffer*. Entonces se invoca al sistema operativo cuando ya se tiene bastante información en el buffer, mejorando así la eficiencia.

Flujos Estándar del Sistema

Los flujos estándar son una característica de muchos sistemas operativos. Por defecto se usan para leer entrada desde el teclado y imprimir texto en la consola. Por defecto existen 3 streams disponibles automáticamente en Java:

- `System.in`: entrada estándar, para lectura de datos desde el teclado.
- `System.out`: salida estándar, imprime en la consola.
- `System.err`: error estándar, también imprime en la consola, pero el sistema operativo puede diferenciar entre salida y error estándar.

System.out

Hasta ahora hemos usado bastante el objeto `System.out`. Algunos métodos clave son:

- `print`: imprime un valor dado, que puede ser primitivo, o un objeto. En caso que sea un objeto se usa el método `toString` implícitamente para la conversión
- `println`: igual que `print` pero agrega un salto de línea al final
- `printf`: similar a la función de C, usa un string de formato y luego los argumentos respectivos
- `format`: básicamente lo mismo que `printf`

¿Cúal clase usar para leer?

En Java existen al menos 3 formas distintas para leer desde el teclado:

- `new BufferedReader(new InputStreamReader(System.in))`
- Usando la clase `Console`, disponible desde Java 6
- Usando la clase `Scanner`, disponible desde Java 5

Lo recomendado es usar siempre `Scanner`, excepto dos casos donde es mejor usar `Console`: lectura de passwords, y lectura caracter por caracter.

Hola Mundo leyendo datos!

- Se debe importar `java.util.Scanner`
- En su constructor recibe `System.in` para leer desde teclado.
- `nextLine()` lee una línea como String
- `nextInt()` lee el siguiente entero

```
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    String nombre;
    int edad;

    nombre = s.nextLine();
    System.out.println(nombre);

    edad = s.nextInt();
    System.out.println(edad);
  }
}
```

Cómo funciona `Scanner`

- La clase `Scanner` toma la entrada desde teclado
- Considera como *delimitador* el espacio en blanco
- La entrada se separa según el delimitador, en una secuencia de *tokens*
- Se intenta interpretar los tokens según el tipo de dato pedido: `nextInt`, `nextDouble`, etc. Funciona con tipos primitivos y Strings.
- Si no se puede, ocurre una excepción!
- Por lo tanto se debe conocer (o determinar) de antemano la estructura de los datos que se van a leer desde teclado, para que la operación siempre sea correcta

Métodos de Lectura de Scanner

Lectura de Strings

Para leer Strings desde `Scanner`, basta usar el método `next`. También, el método `nextLine` toma la entrada hasta el siguiente salto de línea.

Lectura de Tipos Primitivos

Para cada tipo primitivo, como `int` o `double`, existe un método `nextInt`, `nextDouble`, etc.

Finalizar la lectura

El método `hasNext()` retorna falso si ya no hay más elementos por leer.

Ejemplo a desarrollar: Lectura datos de alumnos

Problema

Un usuario ingresará por teclado un listado con los siguientes datos: nombre, apellido, nombre de asignatura, nota 1, nota 2 y nota 3. Primeramente el usuario ingresará la cantidad de alumnos, y luego esa cantidad de filas con la información mencionada.

- **Escriba un programa en Java para imprimir los promedios de nota de cada alumno, indicando a qué asignatura corresponden.**
- **Escriba un programa en Java que además de lo anterior, imprime un ranking de alumnos, de mejor a peor nota.**

Preguntas

Preguntas?