La Sénia
Institut d'Educació Secundària
Paiporta

GENERALITAT VALENCIANA
CONSELLERIA D'EDUCACIÓ, INVESTIGACIÓ, CULTURA I ESPORT

Fondo Social Europeo

Programación

UD 3: Entrada y salida de información

Ciclo Formativo de Grado Superior
Desarrollo de Aplicaciones Web

Jose Chamorro Molina

Introducción a la Programación

ORDEN 60/2012, de 25 de septiembre, de la Conselleria de Educación, Formación y Empleo por la que se establece para la Comunitat Valenciana el currículo del ciclo formativo de Grado Superior correspondiente al título de Técnico Superior en Desarrollo de Aplicaciones Web. [2012/9149]

Contenidos:

5.- Lectura y escritura de información:

5.1.− Programación de la consola: entrada y salida de información.

5.2.− Concepto de flujo.

5.3.− Tipos de flujos. Flujos de bytes y de caracteres.

5.4.− Flujos predefinidos.

5.5.− Clases relativas a flujos.

5.6.− Utilización de flujos.

5.7.− Entrada desde teclado.

5.8.− Salida a pantalla.

Real Decreto 686/2010, de 20 de mayo, por el que se establece el título de Técnico Superior en Desarrollo de Aplicaciones Web y se fijan sus enseñanzas mínimas.

Resultados de aprendizaje:

5. Realiza operaciones de entrada y salida de información, utilizando procedimientos específicos del lenguaje y librerías de clases.

Criterios de evaluación:

5.a) Se ha utilizado la consola para realizar operaciones de entrada y salida de información.

5.b) Se han aplicado formatos en la visualización de la información.

5.c) Se han reconocido las posibilidades de entrada / salida del lenguaje y las librerías asociadas.

Competencias profesionales, personales y sociales:

p) Adaptarse a las nuevas situaciones laborales, manteniendo actualizados los conocimientos científicos, técnicos y tecnológicos relativos a su entorno profesional, gestionando su formación y los recursos existentes en el aprendizaje a lo largo de la vida y utilizando las tecnologías de la información y la comunicación.

Entrada y salida de información

1.– Programación de la consola: entrada y salida de información

2.– Concepto de flujo

3.– Tipos de flujos. Flujos de bytes y de caracteres

4.– Flujos predefinidos

5.– Clases relativas a flujos

6.– Utilización de flujos

7.– Entrada desde teclado

8.– Salida a pantalla

1.– Programación de la consola

1.– Programación de la consola

Una aplicación de consola es un programa informático diseñado para ser utilizado a través de una interfaz de solo texto, como un terminal de texto, la interfaz de línea de comando de algunos sistemas operativos (Unix, DOS, etc.), o la consola Win32 en Microsoft Windows y la terminal en MacOS.

Un usuario generalmente interactúa con una aplicación de consola usando solo un teclado y una pantalla, a diferencia de las aplicaciones de IGU, que normalmente requieren el uso de un mouse u otro dispositivo señalador.

Muchas aplicaciones de consola, como los intérpretes de línea de comandos, son herramientas de línea de comandos, pero también existen numerosos programas de interfaz de usuario basada en texto.

1.– Programación de la consola

A medida que la velocidad y la facilidad de uso de las aplicaciones de IGU han mejorado con el tiempo, el uso de las aplicaciones de consola ha disminuido en gran medida, pero no ha desaparecido.

Algunos usuarios simplemente prefieren las aplicaciones basadas en consola, mientras que algunas organizaciones aún dependen de las aplicaciones de consola existentes para manejar las tareas clave de procesamiento de datos.

1.– Programación de la consola

La capacidad de crear aplicaciones de consola se mantiene como una característica de los entornos de programación modernos porque simplifica enormemente el proceso de aprendizaje de un nuevo lenguaje de programación al eliminar la complejidad de una interfaz gráfica de usuario.

Para tareas de procesamiento de datos y administración de computadoras, es posible que no haya necesidad de una interfaz de usuario bastante gráfica, dejando la aplicación más ágil, más rápida y más fácil de mantener.

Coloreado de texto

El texto que se muestra por pantalla se puede colorear (*únicamente en un terminal de Linux*), para ello es necesario insertar unas secuencias de caracteres - que indican el color con el que se quiere escribir – justo antes del propio texto.

```
String rojo    = "\033[31m";
String verde   = "\033[32m";
String naranja = "\033[33m";
String azul    = "\033[34m";

System.out.print(rojo + " rojo " + verde + " verde");
System.out.print(naranja + " naranja " + azul + " azul");
```

2.– Concepto de flujo

2.– Concepto de flujo

✓ En Java se define la abstracción de stream (flujo) para tratar la comunicación de información entre el programa y el exterior.

- Entre una fuente y un destino fluye una secuencia de datos .

✓ Los flujos actúan como interfaz con el dispositivo o clase asociada.

- Operación independiente del tipo de datos y del dispositivo.

- Mayor flexibilidad (p.e. redirección, combinación).

- Diversidad de dispositivos (fichero, pantalla, teclado, red, ...).

- Diversidad de formas de comunicación:

  • Modo de acceso: secuencial, aleatorio
  • Información intercambiada: binaria, caracteres, líneas

2.– Concepto de flujo

3.– Tipos de flujos:
Flujos de bytes y de caracteres.

3.– Tipos de flujos

java.io

Flujos de bytes:

clases InputStream y OutputStream

Flujos de caracteres:

clases Reader y Writer

Se puede pasar de un flujo de bytes a uno de caracteres con InputStreamReader y OutputStreamWriter

4.– Flujos predefinidos

4.– Flujos predefinidos

System.in

Instancia de la clase InputStream: flujo de bytes de entrada
Métodos:

- read() permite leer un byte de la entrada como entero
- skip(n) ignora n bytes de la entrada
- available() número de bytes disponibles para leer en la entrada

System.out

Instancia de la clase PrintStream: flujo de bytes de salida
Métodos para impresión de datos:

- print(), println(), printf()
- flush() vacía el buffer de salida escribiendo su contenido

System.err

Funcionamiento similar a System.out
Se utiliza para enviar mensajes de error (por ejemplo a un fichero de log o a la consola)

5.– Clases relativas a flujos

5.– Clases relativas a flujos

Las clases del paquete java.io relativas a flujos:

BufferedInputStream: permite leer datos a través de un flujo con un buffer intermedio.

BufferedOutputStream: implementa los métodos para escribir en un flujo a través de un buffer.

FileInputStream: permite leer bytes de un fichero.

FileOutputStream: permite escribir bytes en un fichero o descriptor.

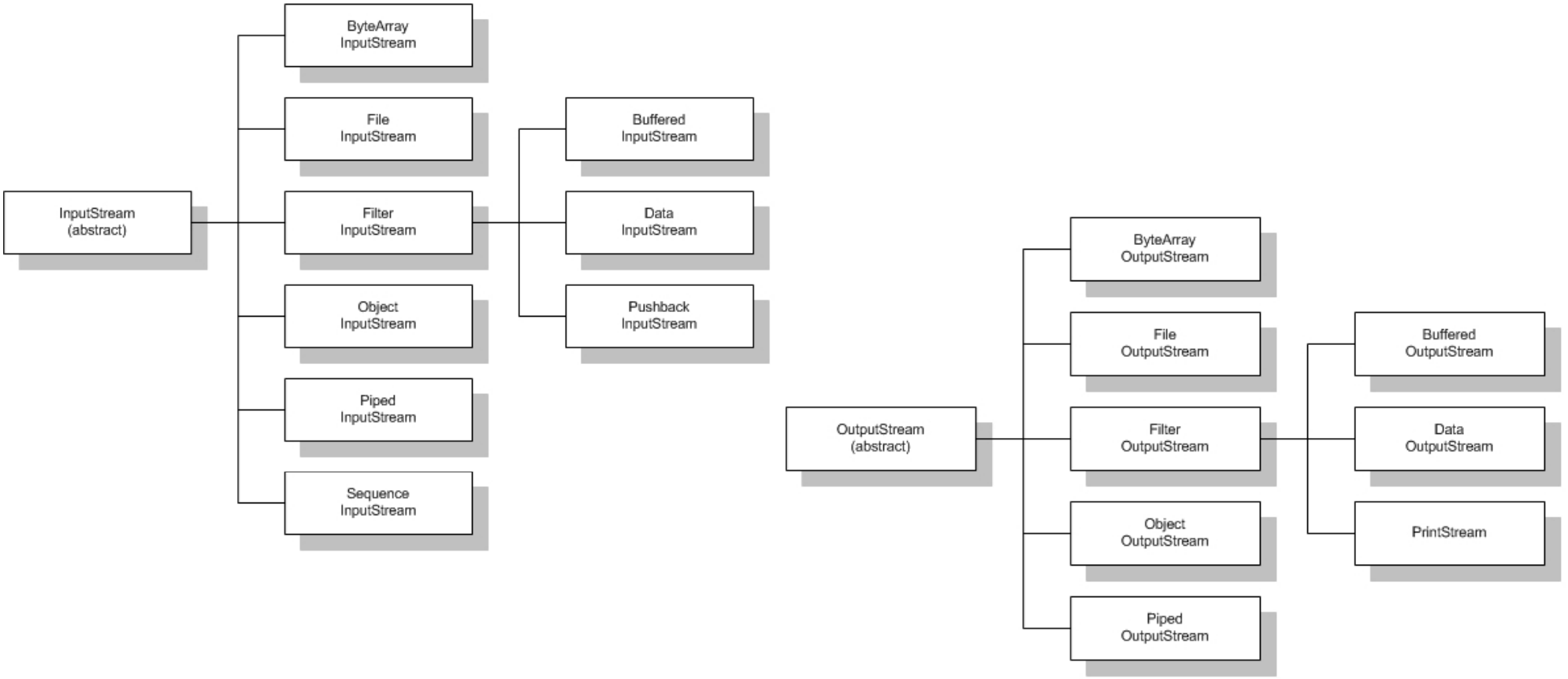
StreamTokenizer: esta clase recibe un flujo de entrada, lo analiza (parse) y divide en diversos pedazos (tokens), permitiendo leer uno en cada momento.

StringReader: es un flujo de caracteres cuya fuente es una cadena de caracteres o string.

StringWriter: es un flujo de caracteres cuya salida es un buffer de cadena de caracteres, que puede utilizarse para construir un string.

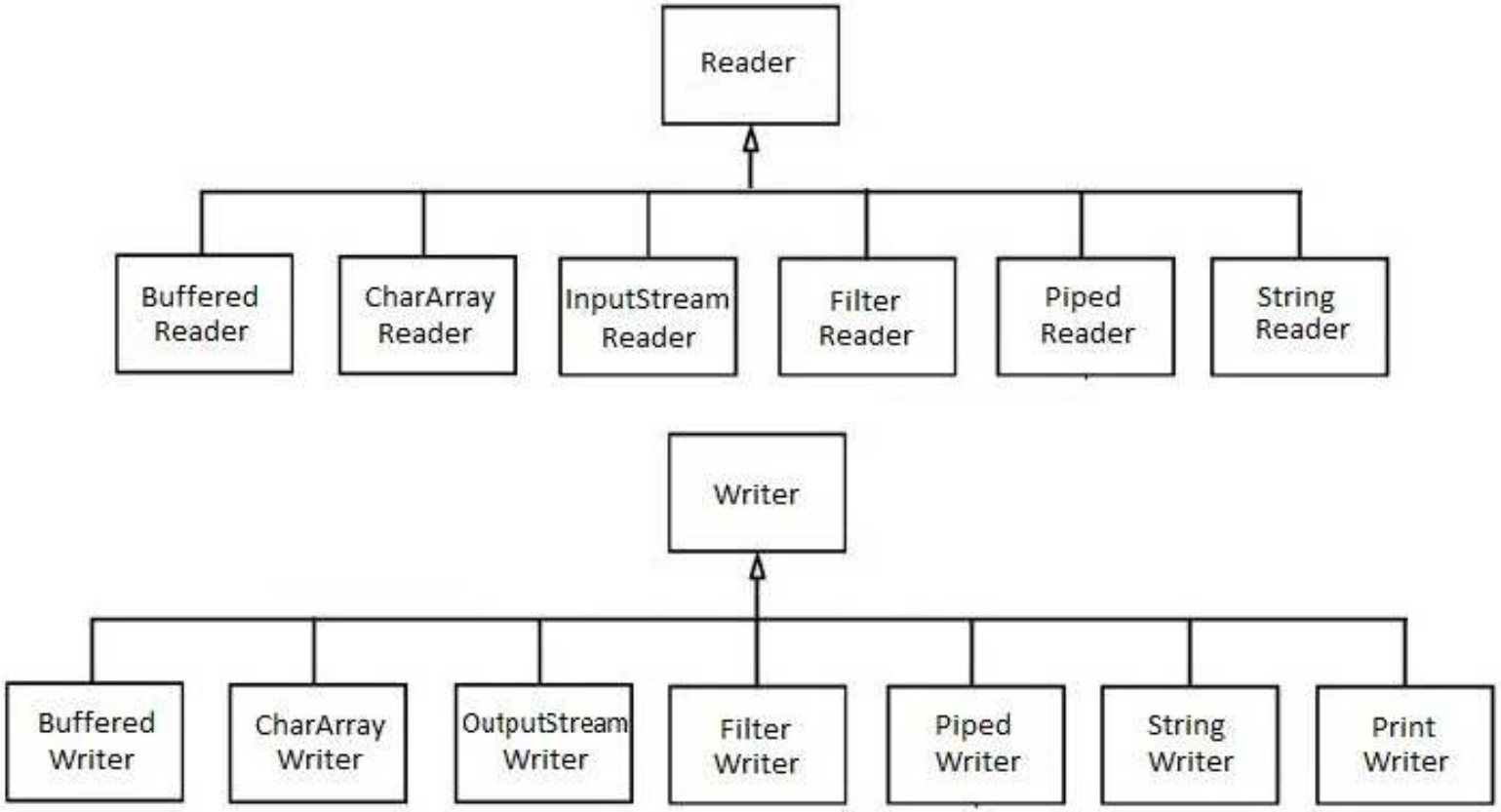
5.– Clases relativas a flujos

Flujos de bytes



5.– Clases relativas a flujos

Flujos de caracteres



6.– Utilización de flujos

6.– Utilización de flujos

Lectura

1. Abrir un flujo a una fuente de datos (creación del objeto stream)
    - Teclado
    - Fichero
    - Socket remoto
2. Mientras existan datos disponibles
    - Leer datos
3. Cerrar el flujo (método close)

Escritura

1. Abrir un flujo a una fuente de datos (creación del objeto stream)
    - Pantalla
    - Fichero
    - Socket local
2. Mientras existan datos disponibles
    - Escribir datos
3. Cerrar el flujo (método close)

***Nota**: para los flujos estándar ya se encarga el sistema de abrirlos y cerrarlos*

6.– Utilización de flujos

Ejemplo:

```
try {
        BufferedReader reader = new BufferedReader(new FileReader("nombrefichero"));

        String linea = reader.readLine();

        while(linea != null) {
                // procesar el texto de la línea
                linea = reader.readLine();
        }
        reader.close();
}
catch(FileNotFoundException e) {
        // no se encontró el fichero
}
catch(IOException e) {
        // algo fue mal al leer o cerrar el fichero
}
```

*Este punto se verá de manera más detallada en la UD10 - Lectura y escritura de información

7.– Entrada desde teclado

7.– Entrada desde teclado

La clase Scanner de Java provee métodos para leer valores de entrada de varios tipos y está localizada en el paquete java.util.

Los valores de entrada pueden venir de varias fuentes, incluyendo valores que se entren por el teclado o datos almacenados en un archivo.

Tenemos que crear un objeto de la clase Scanner asociado al dispositivo de entrada. Si el dispositivo de entrada es el teclado escribiremos:

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

Se ha creado el objeto teclado asociado al teclado representado por *System.in*

Una vez hecho esto podemos leer datos por teclado.

7.– Entrada desde teclado

Principales constructores y métodos de la clase Scanner	
public **Scanner** (InputStream source)	Crea un nuevo Scanner a partir de un flujo de entrada de datos como es el caso de System.in (para poder leer desde teclado).
public String **next** () public String **next** (String pattern)	Obtiene el siguiente elemento leído del teclado como un String (si coincide con el patrón especificado). Lanza NoSuchElementException si no quedan más elementos por leer.
public String **nextLine** ()	Se lee el resto de línea completa, descartando el salto de línea. Devuelve el resultado como un String. Lanza NoSuchElementException si no quedan más elementos por leer.
public int **nextInt** () public long **nextLong** () public short **nextShort** () public byte **nextByte** () public float **nextFloat** () public double **nextDouble** () public boolean **nextBoolean** ()	Devuelve el siguiente elemento como un int siempre que se trate de un int. Ídem para long, short, byte, float, double y boolean. Lanza InputMismatchException en caso de no poder obtener un valor del tipo apropiado. Lanza NoSuchElementException si no quedan más elementos por leer.
public boolean **hasNext** ()	Devuelve true si queda algún elemento por leer.
public boolean **hasNextLine** ()	Devuelve true si queda alguna línea por leer.
public boolean **hasNextInt** () public boolean **hasNextLong** () public boolean **hasNextShort** () public boolean **hasNextByte** () public boolean **hasNextFloat** () public boolean **hasNextDouble** () public boolean **hasNextBoolean** ()	Devuelve true si el siguiente elemento a obtener se puede interpretar como un int. Ídem para long, short, byte, float, double y boolean.
public Scanner **useLocale** (Locale l)	Establece la configuración local del Scanner a la configuración especificada por el Locale l.

7.– Entrada desde teclado

Ejemplo:

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

int numClase;
String nombre;
double nota;

System.out.println("Introduce el número de clase:");
numClase = sc.nextInt();

//NOTA: esta línea es para capturar el retorno de carro
sc.nextLine();

System.out.println("Introduce el nombre del alumno:");
nombre = sc.nextLine();

System.out.println("Introduce la nota del exámen:");
nota = sc.nextDouble();
```

8.– Salida a pantalla

8.– Salida a pantalla

Salida por pantalla

*System.**out**.*println

*System.**out**.*print

Salida por pantalla formateada

*System.**out**.*printf

8.– Salida a pantalla

Imprimir números enteros con *System.**out**.*printf

```
…

//Declaración de variables
int a = 8;
int b = 3;
int resultado = 0;

//%d se sustituye por la variable entera, resultado
//%n indica un salto de línea
resultado = (a + b);
System.out.printf(“La suma es: %d %n”, resultado);


resultado = (a - b);
System.out.printf("La resta es: %d %n", resultado);

…
```

8.– Salida a pantalla

Imprimir números decimales con *System.**out**.*printf

```
…

//%f se sustituye por la variable decimal, res
//2.6666666666666667
res = (double) a / b;
System.out.printf("La división es: %f\n", res);

//Se pueden imprimir diferentes variables en una misma instrucción
System.out.printf("La división entre %d y %d es igual a %f \n", a, b, res);

//2,67
System.out.printf("%.2f %n",  res);
// 2,67
System.out.printf("%5.2f %n", res);
//  2,667
System.out.printf("%7.3f %n", res);
//002,667
System.out.printf("%07.3f %n", res);
//    2,6667
System.out.printf("%10.4f %n", res);
//2,667
System.out.printf ("%5.3f %n", res);
//   2,66667
System.out.printf ("%10.5f %n", res);
//0000000003
System.out.printf("%010.0f %n", res);

…
```

8.– Salida a pantalla

Imprimir texto con *System.**out**.*printf

```
…

//%s se sustituye por la variable de texto, imprime en minúsculas
//%S se sustituye por la variable de texto, imprime en mayúsculas
//El salto de línea se puede indicar con \n o %n

String texto = "Mayor";

//Imprime: El resultado es Mayor
System.out.printf("El resultado es: %s \n", texto);

//Imprime: El resultado es MAYOR
System.out.printf("El resultado es: %S %n", texto);

…
```

8.– Salida a pantalla

Uso de la clase DecimalFormat

```
…
DecimalFormat formateador = new DecimalFormat("####.####");
//Imprime el número pasado como parámetro con cuatro decimales, es decir: 7,1234
System.out.println(formateador.format(7.12342383));

formateador = new DecimalFormat("0000.0000");
//Imprime con 4 cifras enteras y 4 decimales: 0001,8200
System.out.println(formateador.format (1.82));

//Redondeo
double aa = 1.2345;
double bb = 1.2356;

formateador = new DecimalFormat(“#.##”);

System.out.println(formateador.format( aa ));   // La salida es 1,23
System.out.println(formateador.format( bb ));   // La salida es 1,24

//Porcentajes
formateador = new DecimalFormat(“###.##%”);
// Imprime: 68,44%
System.out.println (formateador.format(0.6844));

//Simbolos
DecimalFormatSymbols simbolos = new DecimalFormatSymbols();
simbolos.setDecimalSeparator(‘.’);
formateador = new DecimalFormat(“####.####”, simbolos);
// Imprime: 3.4324
System.out.println (formateador.format (3.43242383));
…
```

Bibliografía

Bibliografía

✓ Aprende JAVA con ejercicios. Edición 2018. Luis José Sánchez.

✓ Empezar a programar usando Java. 2ª edición. Universitat Politècnica de València

✓ Apuntes de la asignatura Ingeniería del Software de la Universitat Politècnica de València.

✓ https://github.com/statickidz/TemarioDAW

✓ https://es.stackoverflow.com

✓ https://en.wikipedia.org/wiki/Console_application