

Procesamiento de archivos

Clases para leer archivos de texto

- `java.io.File`. Una representación de caminos de archivos y directorios.
- `java.util.Scanner`. Un scanner de texto para analizar tipos primitivos y cadenas utilizando expresiones regulares.

Clase File

- Un objeto File permite obtener información sobre un archivo.

// Esto no crea un nuevo archivo en el disco duro

```
File f = new File("example.txt");  
if (f.exists() && f.length() > 1000) {  
    f.delete();  
}
```

Clase File

<code>canRead()</code>	Regresa true si el archivo puede ser leído
<code>delete()</code>	Elimina el archivo del disco
<code>exists()</code>	Regresa true si el archivo existe
<code>getName()</code>	Regresa el nombre del archivo
<code>length()</code>	Regresa el número de bytes en el archivo
<code>renameTo(file)</code>	Cambia el nombre del archivo

Leer archivos

- Para leer un archivo, pasar un objeto File al constructor de Scanner.
- Ejemplo en dos líneas:
`File file = new File("mydata.txt");`
`Scanner input = new Scanner(file);`
- Ejemplo en una línea:
`Scanner input = new Scanner(new File("mydata.txt"));`

Clase Scanner

<code>next()</code>	Regresa el siguiente token como String
<code>nextDouble()</code>	Regresa el siguiente token como doble
<code>nextInt()</code>	Regresa el siguiente token como entero
<code>hasNext()</code>	Regresa true si la entrada tiene un siguiente token
<code>hasNextDouble()</code>	Regresa true si el siguiente token se puede interpretar como un doble
<code>hasNextInt()</code>	Regresa true si el siguiente token se puede interpretar como un entero

Nota: los métodos `hasNext<Tipo>()` no consumen el token.

Tokens

- **Token.** Unidad léxica separada por espacios en blanco.
- Un Scanner divide el contenido de un archivo en tokens.
- Si un archivo contiene lo siguiente:

23	3.14
"John Smith"	

- Un Scanner puede interpretar los tokens como los siguientes tipos:

Token	Tipo
23	int, double, String
3.14	double, String
"John Smith"	String

Cursor de entrada

- Considerar un archivo weather1.txt con el siguiente texto:

```
16.2    23.5
      19.1 7.4   22.8
18.5   -1.8 14.9
```

- Un Scanner ve toda entrada como un flujo de caracteres.

```
16.2    23.5\n19.1 7.4   22.8\n\n18.5   -1.8 14.9\n
```

^

- Cursor de entrada:** es la posición actual del Scanner. Apunta al siguiente token que se va a leer.

Consumiendo tokens

- **Consumir la entrada.** Leer de la entrada y avanzar el cursor.
- Llamar a `nextInt`, etc. mueve el cursor después del token actual.

```
16.2    23.5\n19.1 7.4    22.8\n\n18.5    -1.8  14.9\n
```

^

```
double d = input.nextDouble();    // 16.2
```

```
16.2    23.5\n19.1 7.4    22.8\n\n18.5    -1.8  14.9\n
```

^

```
String s = input.next();          // "23.5"
```

```
16.2    23.5\n19.1 7.4    22.8\n\n18.5    -1.8  14.9\n
```

^

Errores de compilador con archivos

```
import java.io.*;           // para File
import java.util.*;         // para Scanner
public class ReadFile {
    public static void main(String[] args)
    {
        Scanner input = new Scanner(new File("data.txt"));
        String text = input.next();
        System.out.println(text);
    }
}
```

Errores de compilador con Scanner

- El programa no compila por el siguiente error:

`ReadFile.java:6: unreported exception`

`java.io.FileNotFoundException;`

`must be caught or declared to be thrown`

`Scanner input = new Scanner(new File("data.txt"));`

`^`

- Se necesita encerrar esa línea entre try / catch.
- Es un ejemplo de una excepción marcada (*checked exception*).

Excepciones comunes de Scanner

- NoSuchElementException
- Se leyó después del fin del archivo.
- InputMismatchException
- Se leyó en tipo erróneo de token (e.g. leyó “hi” como un int).

Ejemplo de archivos

- Dado el archivo de entrada weather1.txt:

16.2	23.5	
19.1	7.4	22.8
18.5	-1.8	14.9

- Escribir un programa que imprima el cambio de temperatura entre cada par de días vecinos.
- 16.2 a 23.5, cambio = 7.3
- 23.5 a 19.1, cambio = -4.4
- 19.1 a 7.4, cambio = -11.7
- 7.4 a 22.8, cambio = 15.4
- Etc.

Solución

// Despliega los cambios de temperatura de los datos de un archivo de entrada

```
import java.io.*;           // for File
import java.util.*;        // for Scanner
public class Temperatures {
    public static void main(String[] args) throws FileNotFoundException
    {
        Scanner input = new Scanner(new File("weather1.txt"));
        double prev = input.nextDouble();    // fencepost
    }
}
```

Solución

```
while (input.hasNextDouble()) {  
    double next = input.nextDouble();  
    System.out.println(prev + " a " + next + ", cambio = " +  
        (next - prev));  
    prev = next;  
}  
}
```


Ejemplo de archivos 2

- Modificar el programa de temperatura para manejar archivos que contienen tokens no numéricos (saltándolos).
- Por ejemplo, debe producir la misma salida que antes cuando se le da este archivo de entrada, weather2.txt:

```
16.2    23.5  
Tuesday    19.1    Wed 7.4    THURS. TEMP: 22.8  
  
18.5    -1.8    <-- Marty here is my data!    --Kim  
    14.9    :-)
```

- Puede suponerse que el archivo comienza con un número.

Solución 2

// Despliega los cambios de temperatura de los datos de un archivo de entrada

```
import java.io.*;           // for File
import java.util.*;         // for Scanner
public class Temperatures {
    public static void main(String[] args) throws FileNotFoundException
    {
        Scanner input = new Scanner(new File("weather2.txt"));
        double prev = input.nextDouble();           // fencepost
    }
}
```

Solución 2

```
while (input.hasNext()) {  
    if (input.hasNextDouble()) {  
        double next = input.nextDouble();  
        System.out.println(prev + " a " + next + ", cambio = " +  
            (next - prev));  
        prev = next;  
    } else {  
        input.next(); // tira los tokens no deseados  
    }  
}
```

Leer del teclado

// System.in es la entrada standard

```
Scanner console = new Scanner(System.in);
System.out.print("¿Cuál es tu edad? ");
if (console.hasNextInt()) {
    int age = console.nextInt();
    System.out.println("Hey, " + age + " es mucho");
}
else {
    System.out.println("No escribiste un entero.");
}
```


Ejemplo horas

- Dado un archivo hours1.txt con el siguiente contenido:
123 Kim 12.5 8.1 7.6 3.2
456 Eric 4.0 11.6 6.5 2.7 12
789 Stef 8.0 8.0 8.0 8.0 7.5
- Considerar la tarea de calcular las horas trabajadas por cada persona:
Kim (ID#123) trabajó 31.4 horas (7.85 horas/día)
Eric (ID#456) trabajó 36.8 horas (7.36 horas/día)
Stef (ID#789) trabajó 39.5 horas (7.9 horas/día)
- Intentemos resolver este problema token por token...

Solución defectuosa

// ¡Esta solución no funciona!

```
import java.io.*;           // para File
import java.util.*;         // para Scanner
public class HoursWorked {
    public static void main(String[] args)
        throws FileNotFoundException
    {
        Scanner input = new Scanner(new File("hours1.txt"));
    }
}
```

Solución defectuosa

```
while (input.hasNext()) {  
    int id = input.nextInt();    // procesa una persona  
    String name = input.next();  
    double totalHours = 0.0; int days = 0;  
    while (input.hasNextDouble()) {  
        totalHours += input.nextDouble();  
        days++;  
    }  
    System.out.println(name + " (ID#" + id + ") trabajó " + totalHours +  
        " horas (" + (totalHours / days) + " hours/day)");  
} } }
```

Solución defectuosa

Kim (ID#123) worked 487.4 hours (97.48 hours/day)

Exception in thread "main"

java.util.InputMismatchException

at java.util.Scanner.throwFor(Scanner.java:840)

at java.util.Scanner.next(Scanner.java:1461)

at java.util.Scanner.nextInt(Scanner.java:2091)

at HoursWorked.main(HoursBad.java:9)

- El ciclo interno while toma la identificación de la siguiente persona.
- Se quiere procesar los tokens, pero los saltos de línea importan porque marcan el final de los datos de una persona.

Scanner basado en líneas

<code>nextLine()</code>	Devuelve la siguiente línea completa de entrada (desde el cursor hasta <code>\n</code>)
<code>hasNextLine()</code>	Devuelve <code>true</code> si hay más líneas de entrada para leer (siempre es cierto para la entrada de la consola)

```
Scanner input = new Scanner(new File("file name"));
while (input.hasNextLine()) {
    String line = input.nextLine();
    process this line;
}
```


Consumiendo líneas de entrada

23 3.14 John Smith "Hello" world
45.2 19

- El Scanner lee las líneas como sigue:

```
23\t3.14 John Smith\t"Hello" world\n\t\t45.2 19\n^
```

```
String line = input.nextLine();
```

```
23\t3.14 John Smith\t"Hello" world\n\t\t45.2 19\n^
```

```
String line2 = input.nextLine();
```

```
23\t3.14 John Smith\t"Hello" world\n\t\t45.2 19\n^
```

- Cada caracter \n se consume pero no se regresa.

Scanner sobre strings

- Un Scanner puede tokenizar el contenido de un String:
`Scanner name = new Scanner(String);`

Scanner sobre strings

- Ejemplo:

```
String text = "15 3.2 hello 9 27.5";
```

```
Scanner scan = new Scanner(text);
```

```
int num = scan.nextInt();
```

```
System.out.println(num); // 15
```

```
double num2 = scan.nextDouble();
```

```
System.out.println(num2); // 3.2
```

```
String word = scan.next();
```

```
System.out.println(word); // hello
```



Ejemplo horas

- Con esto, ya se puede arreglar el ejemplo.

Solución horas correcta

// Procesa un archivo de empleados e imprime las horas de cada empleado

import java.io.*; **// for File**

import java.util.*; **// for Scanner**

public class Hours {

 public static void main(String[] args) throws FileNotFoundException {

 Scanner input = new Scanner(new File("hours1.txt"));

 while (**input.hasNextLine()**) {

 String line = input.nextLine();

Scanner lineScan = new Scanner(line);

 int id = **lineScan.nextInt()**; **// e.g. 456**

 String name = **lineScan.next()**; **// e.g. Eric**

Solución horas correcta

```
double sum = 0.0;
int count = 0;
while (lineScan.hasNextDouble()) {
    sum = sum + lineScan.nextDouble();
    count++;
}
double average = sum / count;
System.out.println(name + " (ID#" + id + ") worked " +
    sum + " hours (" + average + " hours/day)");
}
}
```

Ejemplo contar palabras

Archivo input.txt	Salida en la consola
The quick brown fox jumps over the lazy dog.	La línea tiene 6 palabras La línea tiene 3 palabras

Ejemplo contar palabras

// Cuenta las palabras de cada línea de un archivo

```
Scanner input = new Scanner(new File("input.txt"));
```

```
while (input.hasNextLine()) {
```

```
    String line = input.nextLine();
```

```
    Scanner lineScan = new Scanner(line);
```

// Procesa el contenido de la línea

```
    int count = 0;
```

```
    while (lineScan.hasNext()) {
```

```
        String word = lineScan.next();
```

```
        count++;
```

```
    }
```

```
    System.out.println("La línea tiene " + count + " palabras");
```

```
}
```


Salida a archivos

- **PrintStream.** Un objeto en el paquete java.io que permite imprimir la salida a un destino como un archivo.
- Cualquier método utilizado en System.out (como print, println) funciona en un PrintStream.

- Sintaxis:

```
PrintStream name = new PrintStream(new File("file name"));
```

- Ejemplo:

```
PrintStream output = new PrintStream(new File("out.txt"));
```

```
output.println("Hello, file!");
```

```
output.println("This is a second line of output.");
```

Detalles sobre PrintStream

- Si el archivo no existe, se crea.
- Si el archivo ya existe, se sobrescribe.
- La salida aparece en el archivo, no en la consola.
- No se debe abrir el mismo archivo para leer (Scanner) y escribir (PrintStream) al mismo tiempo.
- El archivo se va a sobrescribir con un archivo vacío (0 bytes).

System.out y PrintStream

- El objeto de salida de la consola, System.out, es un PrintStream.
PrintStream out1 = System.out;
PrintStream out2 = new PrintStream(new File("data.txt"));
out1.println("Hello, console!"); // va a la consola
out2.println("Hello, file!"); // va al archivo
- Se puede guardar una referencia a System.out en una variable PrintStream.
- La impresión en la variable hace que aparezca la salida en la consola.
- Se puede pasar System.out a un método que reciba un PrintStream Como parámetro.

Ejemplo con PrintStream

- Modificar el programa de horas para usar un PrintStream para enviar la salida al archivo hours_out.txt.
- El archivo hours_out.txt se crea con el texto:
Kim (ID#123) trabajó 31.4 horas (7.85 horas/día)
Eric (ID#456) trabajó 36.8 horas (7.36 horas/día)
Stef (ID#789) trabajó 39.5 horas (7.9 horas/día)

Solución con PrintStream

// Procesa un archivo de empleados e imprime las horas de cada empleado

```
import java.io.*;    // for File
import java.util.*;  // for Scanner
public class Hours {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner input = new Scanner(new File("hours.txt"));
        PrintStream output = new PrintStream(new File("hours_out.txt"));
        while (input.hasNextLine()) {
            String line = input.nextLine();
            Scanner lineScan = new Scanner(line);
            int id = lineScan.nextInt();           // e.g. 456
            String name = lineScan.next();         // e.g. Eric
        }
    }
}
```

Solución con PrintStream

```
double sum = 0.0;
int count = 0;
while (lineScan.hasNextDouble()) {
    sum = sum + lineScan.nextDouble();
    count++;
}
double average = sum / count;
output.println(name + “ (ID#” + id + “) worked ” +
                sum + “ hours (” + average + “ hours/day)”);
    }
}
```

Mezclando tokens y líneas

- Usar `nextLine()` junto con los métodos `next<Type>()` en el mismo Scanner puede generar resultados no esperados.

Ejemplo

```
Scanner console = new Scanner(System.in);  
System.out.print("Enter your age: ");  
int age = console.nextInt();  
System.out.print("Now enter your name: ");  
String name = console.nextLine();  
System.out.println(name + " is " + age + " years old.");
```

Log of execution (user input underlined):

```
Enter your age: 12  
Now enter your name: Sideshow Bob  
is 12 years old.
```


Ejemplo

```
Overall input:      12\nSideshow Bob
After nextInt() :  12\nSideshow Bob
                   ^
After nextLine() :  12\nSideshow Bob
                   ^
```

- Conclusión: No hay que leer tokens y líneas del mismo Scanner.

Resumen

- Conceptos y clases importantes.
- Clase File.
- Clase Scanner.
- Scanners con archivos o con strings.
- No leer tokens y líneas con el mismo scanner.
- Uso de `next<Tipo>()` y `hasNext<Tipo>()`.
- Excepciones marcadas (checked exceptions).
- Clase `PrintStream`.