ESCOLA SUPERIOR DE ENXEÑARÍA INFORMÁTICA

Universidade Vigo

TEMA 4: Entrada/Salida

Programación II - 2017/2018

Pedro Cuesta Morales, Baltasar García Pérez-Schofield,
Encarnación González Rufino (Dpto. de Informática)

Índice

1. Introducción

Programa JAVA: necesita datos de entrada para generar unos determinados datos de salida

Entrada/salida estándar:
Por consola (teclado + pantalla)

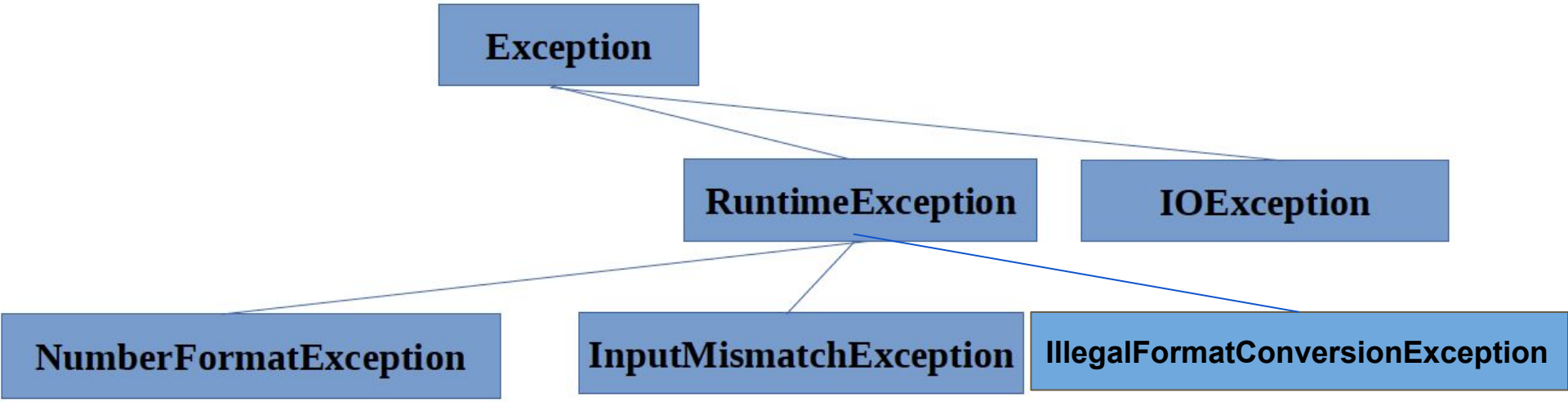
Clases envolventes:

métodos para manejar
tipos primitivos

Excepciones:

gestión de errores en E/S

2. Excepciones relacionadas



NumberFormatException: se lanza para indicar que la aplicación ha intentado convertir una cadena a uno de los tipos numéricos, pero que la cadena no tiene el formato apropiado.

InputMismatchException: lanzado por una instancia de Scanner para indicar que el token recuperado no coincide con el patrón del tipo esperado o que el token está fuera de rango para el tipo esperado (p.e. nextInt() se introduce un carácter)

IOException (controladas): la utiliza el programador para indicar que se ha producido una excepción de E/S de algún tipo (fallo o interrupción en las operaciones de E/S).

3. Clases envolventes

Se pueden utilizar tipos primitivos: int, double, char, ...

JAVA: Lenguaje Orientado a Objetos

Clases especiales que proporcionan una serie de funcionalidades (métodos) sobre estos tipos primitivos (convertirlos en clases)

Tipo primitivo	Clase asociada	Explicación
int	**Integer**	Números enteros.
double	**Double**	Números reales.
char	**Character**	Caracteres Unicode.
boolean	**Boolean**	Booleanos, **true** o **false**.

3. Clases envolventes

Miembro	¿Es estático?	Explicación
`TRUE`	Sí	Constante **true**.
`FALSE`	Sí	Constante **false**.
`parseBoolean(s)`	Sí	Transforma una cadena en un booleano.
`toString()`	No	Devuelve el booleano como una cadena.

Boolean

Miembro	¿Es estático?	Explicación
`MAX_VALUE`	Sí	Constante máximo valor entero soportado.
`MIN_VALUE`	Sí	Constante mínimo valor entero soportado.
`parseInt(s)`	Sí	Transforma una cadena en un número entero.
`toString()`	No	Devuelve el número como una cadena.
`doubleValue()`	No	Devuelve el entero como **double**.
`intValue()`	No	Devuelve el valor guardado como **int**.

Integer

3. Clases envolventes

- Convertir una cadena en un valor primitivo (estático): *Integer.parseInt(s)*, *Boolean.parseBoolean(s)*, *Double.parseDouble(s)*
  Pueden lanzar la excepción *NumberFormatException*
- Convertir en un String (no estático): *toString()*
- Devolver el valor como tipo primitivo (no estático): *intValue(), doubleValue()*

https://ideone.com/eS3wbD

```
int x = Integer.parseInt( "5" );
Integer intX = new Integer( x );
int copiaX = intX.intValue();
System.out.println( x + " " + intX + " " + copiaX );

double d = Double.parseDouble( "5.6" );
Double doubleD = new Double( d );
double copiaD = doubleD.doubleValue();
System.out.println( d + " " + doubleD + " " + copiaD);
```

String.format(s, x...)

Permite crear una cadena a partir de una serie de datos, aportando una plantilla con formato y campos tipo

Campo tipo de formato	Explicación
%d	Número entero.
%f	Número real.
%s	Cadena de texto.

La cadena de formato también puede contener constantes de formato: \n \t \\

%{ancho}d
%{ancho}{.posiciones_decimales}f

Si el ancho se precede de un 0, se utiliza este dígito para rellenar los espacios sobrantes (en vez del " "), por la izquierda

Si el valor aportado no se corresponda con el campo tipo, se producirá una excepción: *IllegalFormatConversionException*

String.format(s, x...)

http://ideone.com/zGrh5v

```
String cad;
cad = String.format( "'%6d'\n", 5 );
System.out.print(cad);
cad =String.format( "'%6.2f'\n", 5.6 );
System.out.print(cad);
cad =String.format( "'%06d'\n", 5 );
System.out.print(cad);
cad = String.format( "'%06.2f'\n", 5.6 );
System.out.print(cad);
String asignatura = "PROII";
String lenguaje = "Java";
int anho = 2015;
cad = String.format( "Impartimos %s en %s desde %d.", asignatura,
lenguaje, anho );
System.out.println(cad);
```

stdout

```
'     5'
'  5.60'
'000005'
'005.60'
Impartimos PROII en Java desde 2015.
```

String.format(s, x...)

http://ideone.com/fmw05O

```
class Persona {
    private String nombre;
    private int dni;
    private int anhoNacimiento;
```

```
/** @return info de la persona como una cadena */
public String toString()
{
    return String.format( "%s (%8d): %2d años",getNombre(), getDni(),
                          getEdad());
}
```

4.1. Salida por consola

Clase *System*:

- objeto *out* (*PrintStream*): salida estándar - pantalla
- objeto *err* (*PrintStream*): salida estándar de errores - pantalla
- Métodos:

Miembro	Explicación
`print(x)`	Muestra *x* por consola.
`println()`	Cambia de línea.
`println(x)`	Muestra *x* por consola, y cambia de línea.
`format(s, x...)`	Muestra datos por consola, con un formato determinado.

format()

Similar *String.format()*

http://ideone.com/Us9131

```
System.out.format( "'%6d'\n", 5 );
System.out.format( "'%6.2f'\n", 5.6 );
System.out.format( "'%06d'\n", 5 );
System.out.format( "'%06.2f'\n", 5.6 );
String asignatura = "PROII";
String lenguaje = "Java";
int anho = 2015;
System.out.format( "Impartimos %s en %s desde %d.", asignatura,
                   lenguaje, anho );
```

stdout

```
'     5'
'  5.60'
'000005'
'005.60'
Impartimos PROII en Java desde 2015.
```

4.2. Entrada por consola

clase ***Scanner*** (java.util.Scanner)

- Crea objeto: *Scanner entrada = new Scanner(system.in);*
- Leer cadenas de texto: *nextLine()*
- Leer número entero: *nextInt()*
- Leer número real: *nextFloat()* o *nextDouble()*

→ excepción: *InputMismatchException*

```
public static void main (String[] args)
{
	String nombre;
	int edad;
	Persona[] v;
	Scanner scan = new Scanner( System.in );
	int maxPersonas = 0;

	// Leer el max. y crear vector
	try {
		maxPersonas = Integer.parseInt( scan.nextLine() );
		v = new Persona[ maxPersonas ];

		// Pedir los datos
		for(int i = 0; i < v.length; ++i) {
			System.out.print( "Dame un nombre: " );
			nombre = scan.nextLine();

			System.out.print( "Dame una edad: " );
			edad = Integer.parseInt( scan.nextLine() );

			v[ i ] = new Persona( nombre, edad );
		}

		// Mostrar los datos
		for(int i = 0; i < v.length; ++i) {
			System.out.println( v[ i ] );
		}
	} catch(NumberFormatException exc)
	{
		System.err.println( "\nERROR: No se ha introducido un número." );
	}
}
```

Sería interesante tener un método para leer números más robusto, que volviese a pedir el dato en el caso de que no se introdujera un número

Ejemplo: class Persona (nombre y edad)

```
String nombre;
int edad;
Persona[] v;
Scanner scan = new Scanner( System.in );
int maxPersonas = 0;
// Leer el max. y crear vector
maxPersonas = leeNum2( scan, "Max. personas: " );
v = new Persona[ maxPersonas ];
// Pedir los datos
for(int i = 0; i < v.length; ++i) {
    System.out.print( "Dame un nombre: " );
    nombre = scan.nextLine();
    edad = leeNum( scan, "Dame una edad: " );
    v[ i ] = new Persona( nombre, edad );
}
// Mostrar los datos
for(int i = 0; i < v.length; ++i) {
    System.out.println( v[ i ] );
}
```

```java
public static int leeNum2(Scanner scan, String msg)
{
    int toret = 0;
    boolean repite;
    do {
        repite = false;
        System.out.print( "\n" + msg );
        try {
            toret = scan.nextInt();
        } catch(InputMismatchException exc)
          {
                repite = true;
          }
        finally {
                scan.nextLine();
        }
    } while( repite );
    return toret;
}
```

http://ideone.com/1gT4tQ

```java
public static int leeNum(Scanner scan, String msg)
{
    int toret = 0;
    boolean repite;
    do {
        repite = false;
        System.out.print( "\n" + msg );
        try {
            toret = Integer.parseInt( scan.nextLine() );
        } catch(NumberFormatException exc)
          {
                repite = true;
          }
    } while( repite );
    return toret;
}
```

Necesario para evitar que el cambio de línea que se queda en el escáner de entrada ('\n') afecte a subsiguientes lecturas