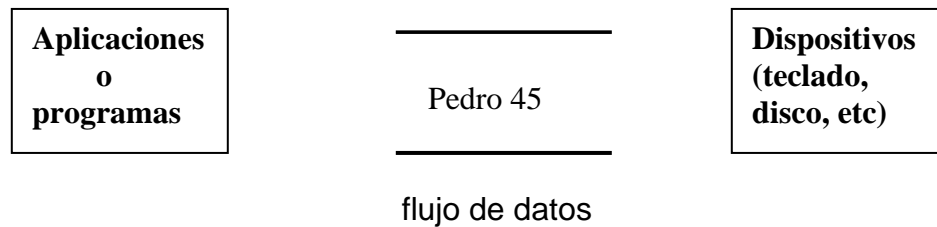


III. Entrada, salida y manejo de excepciones.

3.1 Flujo de datos

En Java la entrada/salida de datos en los programas se realiza mediante **flujos de datos**. Estos flujos son objetos que actúan de intermediario entre el programa y el dispositivo físico (teclado, pantalla, archivo en disco, una conexión a red o un buffer en memoria como origen o destino de datos para la entrada o salida de los mismos), por lo que las mismas clases y métodos de E/S se pueden aplicar a cualquier tipo de dispositivo.



3.1.1 Tipos de flujos de datos:

Los tipos de flujos que existen dependen del código utilizado para representar un carácter. Actualmente se puede utilizar dos formas para E/S de datos a través de flujos, estos son:

- **Flujo de bytes (8 bits , para representar un carácter en ASCII):** Originalmente, la única manera para E/S de datos y todavía se utiliza, pero se considera obsoleto.
- **Flujo de caracteres (16 bits, para representar un carácter en UNICODE):** Estos se diseñaron después del flujo de bytes para permitir la internacionalización de los programas porque permite trabajar todos los caracteres de los lenguajes y dialectos en el mundo.

3.1.2 Clases utilizadas para el manejo de flujo de datos:

Desde que inició Java, ofrece diferentes **clases** para trabajar los flujos de datos:

1. **Clase System** del paquete **java.lang** maneja **flujo de bytes** o predefinidos o standard. El paquete **java.lang** se importa automáticamente.

```

public final Class System
{
    ...
}

```

public: Permite ser usada en cualquier paquete

final: Permite referenciar a los miembros de la clase a través de System como si fuera el objeto.

```

public final static InputStream in = null; /*objeto del tipo o instanciado a la
                                           clase InputStream*/
public final static PrintStream out = null; /*objeto del tipo o instanciado a la
                                           clases PrintStream*/
public final static PrintStream err = null; //objeto instanciado a la PrintStream
...
}

```

Recuerde una clase tiene datos y métodos.

La clase **System** tiene como datos 3 objetos de flujos predefinidos llamados **in**, **out** y **err**.

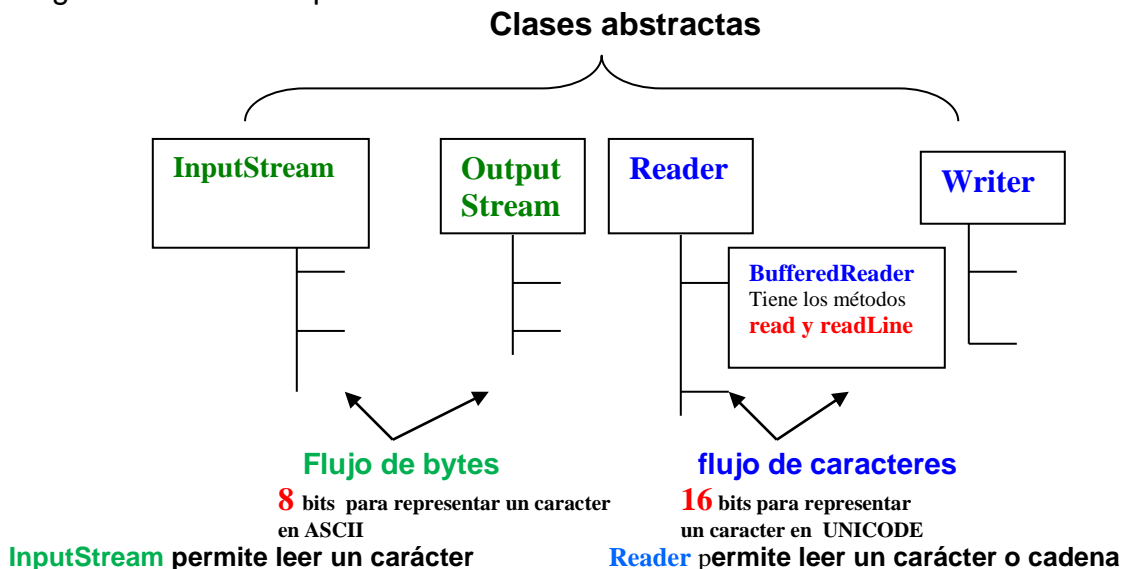
in: Permite el acceso método `read()` para la lectura de un solo carácter por el teclado.

Out: Permite el acceso al método `print()` o `println ()` para imprimir por **pantalla**.

err: Referencia la salida de errores.

2. Clases abstractas **InputStream**, **OutputStream** (para el flujo de bytes) y **Reader** y **Writer** (para el flujo de caracteres) del paquete **java.io**. Este paquete requiere importarse usando la instrucción **import java.io.***;

Cada una de estas clases tienen varias subclases que implementan los métodos que permiten leer y escribir caracteres en cualquier dispositivo. A continuación, el diagrama de clases aplicando la herencia:



3.2 Flujo de caracteres y la clase `BufferedReader` para leer del teclado:

En este curso utilizaremos para la entrada **el flujo de caracteres**, y lo primero que hacemos es crear un objeto instanciado a la clase `BufferedReader` asociada a la `InputStreamReader` y ligada al teclado mediante `System.in` con el siguiente formato.

Formato:

```
BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
```

`br` es un objeto que representa el flujo de caracteres

Lo segundo que hay que hacer es definir que se quiere leer si un carácter o una cadena. A continuación, formatos para la lectura de un carácter o una cadena:

Lectura de un solo carácter: Para leer un caracter se usa el método `read`

Método	Ejemplo	Descripción
<code>public int read () throws IOException</code>	<pre>char character; character = (char) br.read(); br.skip(1);</pre>	<p>El método <code>read</code> retorna el carácter en código Unicode en decimal por lo que se requiere del casting a <code>char</code>.</p> <p>Por ejemplo si el usuario introduce por el teclado el carácter <code>d</code> el método <code>read</code> retorna el código decimal en Ascii equivalente a 100, por lo que se requiere convertir a <code>char</code> con el casting o conversión explícita.</p> <p><code>\r\n</code> son los caracteres originados al pulsar enter por esto se hace <code>skip</code> de 1 para limpiar del flujo esos caracteres</p>

Lectura de cadena: Para leer una cadena se usa el siguiente método.

Método	Ejemplo	Descripción
<code>public String readLine () throws IOException</code>	<pre>String cadena; cadena = br.readLine();</pre>	<p>retorna un String que contiene los caracteres leídos o retorna null si hace un intento de lectura al final del flujo.</p>

3.3 Clases y métodos de conversión de datos:

En esta asignatura se les enseña a leer un carácter o una cadena como explique antes. Para tratar esta clase de necesidad Java, provee un conjunto de clases que permiten convertir cadena con contenidos numéricos a un tipo de dato simple o primitivo.

Estas clases pertenecen al paquete **java.lang**. Las clases son Double, Float, Long, Integer, Short, Byte y los métodos de conversión parseDouble (), parseFloat (), parseLong (), parseInt (), parseShort (), parseByte ().

A continuación, se presenta un cuadro con las clases y sus correspondientes métodos de conversión:

Clases	Métodos de conversión
Double	static double parseDouble (String str) throws NumberFormatException
Float	static float parseFloat (String str) throws NumberFormatException
Long	static long parseLong (String str) throws NumberFormatException
Integer	static int parseInt (String str) throws NumberFormatException
Short	static short parseShort (String str) throws NumberFormatException
Byte	static byte parseByte (String str) throws NumberFormatException

Ejemplo:

Las clases Double, Float, Long, Integer, Short y Byte son **public final**, por lo que no se necesita instanciar un objeto a la clase.

Ejemplo en forma simplificada	Ejemplo en forma detallada
<pre>int num; double saldo; Byte edad; : num = Integer.parseInt (br.readLine()); saldo = Double.parseDouble (br.readLine()); edad = Byte.parseByte (br.readLine()) ;</pre>	<pre>int num; String cadena; : //otra forma de hacerlo cadena= br.readLine(); num = Integer.parseInt (cadena) ;</pre>

3.4 Manejo de excepciones:

Las excepciones son clases que se usan para atrapar diferentes tipos errores. Las excepciones que se pueden utilizar al leer un caracter, cadenas y al convertir cadenas a un tipo de dato primitivo (int, float, double, etc) están en las siguientes clases:

Clases	Descripción	Paquete
Exception	Atrapa todos los errores (I/O, archivos, sql, ...)	java.lang
IOException	Atrapa los errores de entrada/salida generados al utilizar por ejemplo los métodos read (), readLine(), etc.	java.io
NumberFormatException	Atrapa errores de conversión de tipo de dato generados al utilizar los métodos parseInt (), parseFloat() , etc.	java.lang

Existen tres formas de control de excepciones:

- **Claúsula throws:** el mensaje de error sale generado por el sistema, narrado de una forma muy técnica en inglés, de tal manera poco entendible inclusive para personas que no son desarrolladoras de software y después se **detiene** la ejecución del programa. Esta opción de controlar excepciones no es eficiente.

Formato:

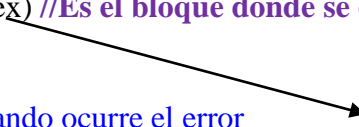
tipo_dato nombre_metodo() throws nombre_exception

[Ver ejemplo1 ConversionHoras1p](#)
[Ver ejemplo2 ConversionHoras2p](#)
[Ver ejemplo3 ConversionHoras3p](#)

- **try y catch():** utiliza un bloque try para colocar el código de entrada capaz de lanzar las excepciones y un bloque catch para capturarla e imprimir el mensaje, con esto **no se detiene** la ejecución del programa (la ejecución del programa continua mientras el usuario lo desee). Entonces el programador puede tener mejor control al desarrollar la lógica al programar y enviar los mensajes entendibles por cualquier usuario del programa y hasta el mensaje técnico que atrapa el objeto.

Formato:

```
try //Es el bloque donde se lanzan
{
    // sentencias de entrada/salida o de conversión
}
catch (nombre_excepcion ex) //Es el bloque donde se capturan las excepciones
{
    //sentencias a realizar cuando ocurre el error
}
```



Es el objeto que atrapa la **cadena** con el mensaje de error técnico que ocurrió

Recuerde que los nombres de los objetos los define el programador, por lo que puede ser cualquier nombre que cumpla con las reglas de nombre para un identificador.

Ejemplo:

```
try
{
    caracter=(char)br.read();
    br.skip(2);
}
catch (IOException ex)
{
    System.out.println("\n\nHa ocurrido un error en la lectura"+ex);
}
```

Ejemplo:

```
String str=" 12 ";
int numero;
try{
    numero=Integer.parseInt(str);
}
catch(NumberFormatException ex)
{
    System.out.println("\n\nNo es un número entero ha ocurrido un error de
    conversión de tipo de dato");
}
```

Ejemplo:

```
int num;
String cadena;

try
{
    System.out.println("Entre un numero entero");
    cadena= br.readLine();

    num = Integer.parseInt (cadena) ;
}

catch (IOException ex)
{ System.out.println("\n\nHa ocurrido un error en la lectura"+ex); }

catch(NumberFormatException ex)
{ System.out.println("\n\nNo es un número entero ha ocurrido un error de
                        conversión de tipo de dato");
}
```

Ejemplo:

```
int num;
String cadena;

try
{
    System.out.println("Entre un numero entero");
    cadena= br.readLine();

    num = Integer.parseInt (cadena) ;
}

catch (Exception ex)
{ System.out.println("\n\nHa ocurrido un error en la lectura o de conversión
de tipo de dato "+ex); }
```

Ver ejemplo4 Suma1p

Ver ejemplo5 Suma2p

- **throws, try y catch:** La forma más eficiente de capturar errores. En este curso no la explicamos. Se explica en el curso de desarrollo de software III de la carrera de desarrollo de software.