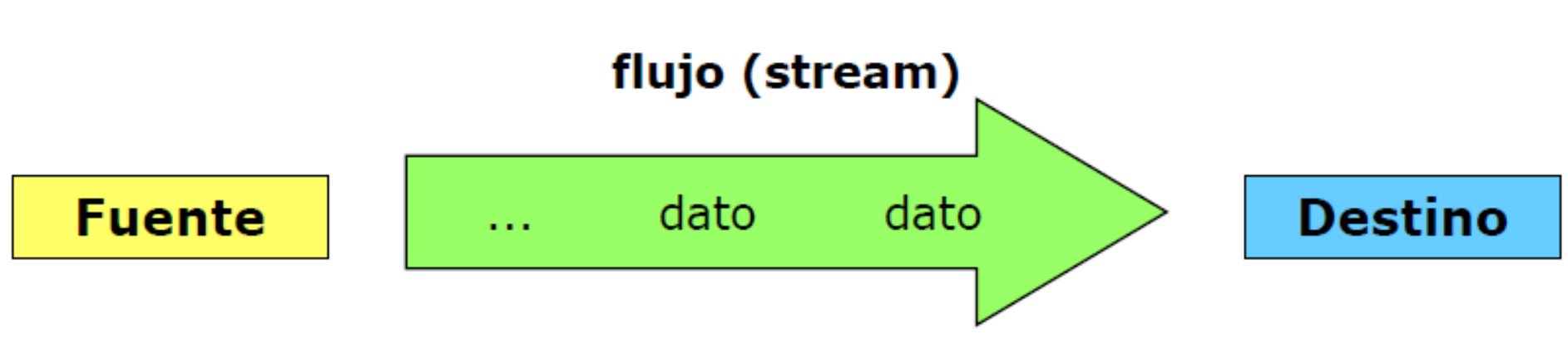
Programación en Java

Flujos y Manejo de Excepciones

Flujo de datos

- Para recibir datos del teclado se requiere un flujo de datos (stream) entre la interfaz y el programa.

- El flujo es independiente del tipo de dato y del dispositivo.

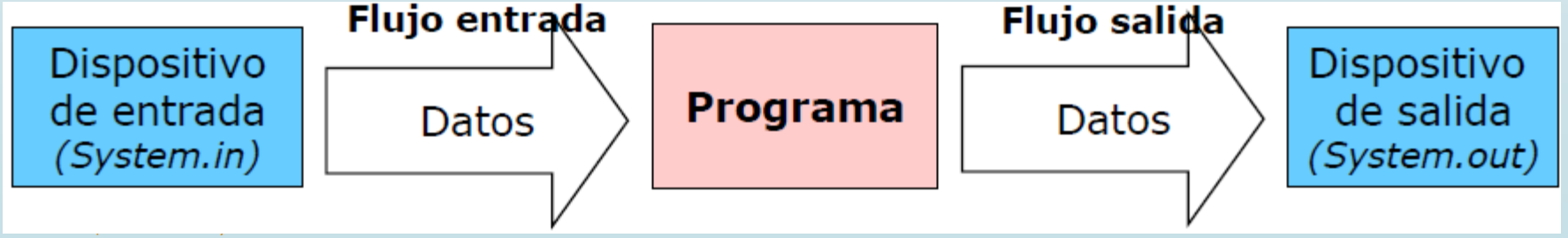


Clase System

- Proporciona acceso a recursos del sistema como los dispositivos de entrada/salida sin importar la plataforma.
- Controla 3 canales o streams:

System.in	Entrada estándar	InputStream
System.out	Salida estándar	PrintStream
System.err	Salida de error estándar	PrintStream

- Métodos: `print(), println, printf(), read()`



Entrada estándar

- **Sólo pueden leerse bytes.**
- El mismo código puede servir para leer del teclado, de un archivo o de otro dispositivo de entrada.
- Al leer la letra **"k" minúscula** se recibe el **valor entero 107** que es el valor del byte leído que corresponde a su representación **ASCII.**
- Para leer palabras o números deben convertirse al tipo de dato necesario.

Entrada estándar

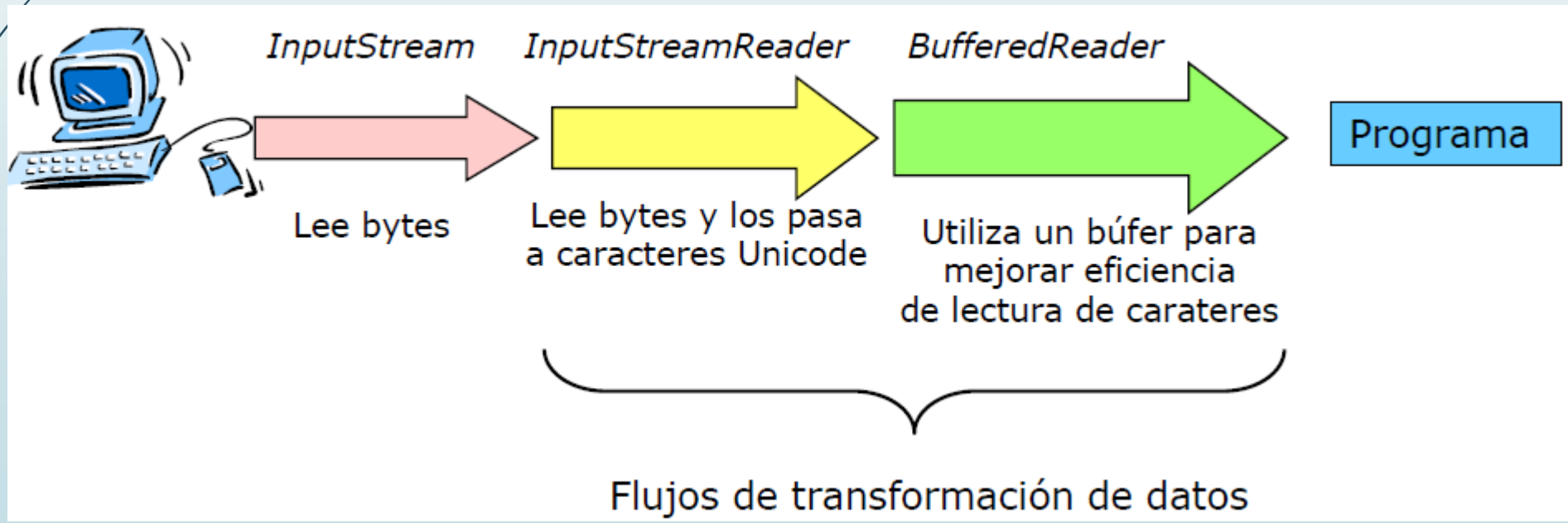
Clase	Definición	Qué hace?
InputStream (System.in)	`System.in.read();`	Lee bytes de teclado, archivo u otro dispositivo de entrada.

Reader	`InputStreamReader isr = new InputStreamReader(System.in);`
	
	`int c=0;`
	`c= isr.read();`
		Toma del flujo de entrada los bytes y convierte a caracteres sueltos si no se dice cuantos ni limite.

BufferedReader	`InputStreamReader isr = new InputStreamReader(System.in);`
	`BufferedReader br = new BufferedReader (isr);`
	
	`String texto = br.readLine();`
		Toma una línea completa de caracteres a partir del Reader anterior hasta el fin o salto

Combinación de flujos

- Leer cada byte del flujo de la entrada estándar (*System.in - InputStream*), se convierten a caracteres Unicode (InputStreamReader) y se procesan almacenando en un búfer y se entregan como una cadena (BufferedReader)



Excepción

- Son el mecanismo por el cual pueden controlarse en un programa Java las condiciones de error que se producen.
- Cuando sucede un evento anormal en la ejecución de un programa y lo detiene decimos que se ha producido una excepción.
- Pueden ser errores en la lógica del programa o errores disparados por los propios objetos que denuncian algún tipo de estado no previsto, o condición que no pueden manejar.
- Deberá ser capturada por el que le llamó o por alguien más.
- Contiene información del error que se ha producido.

Clasificación

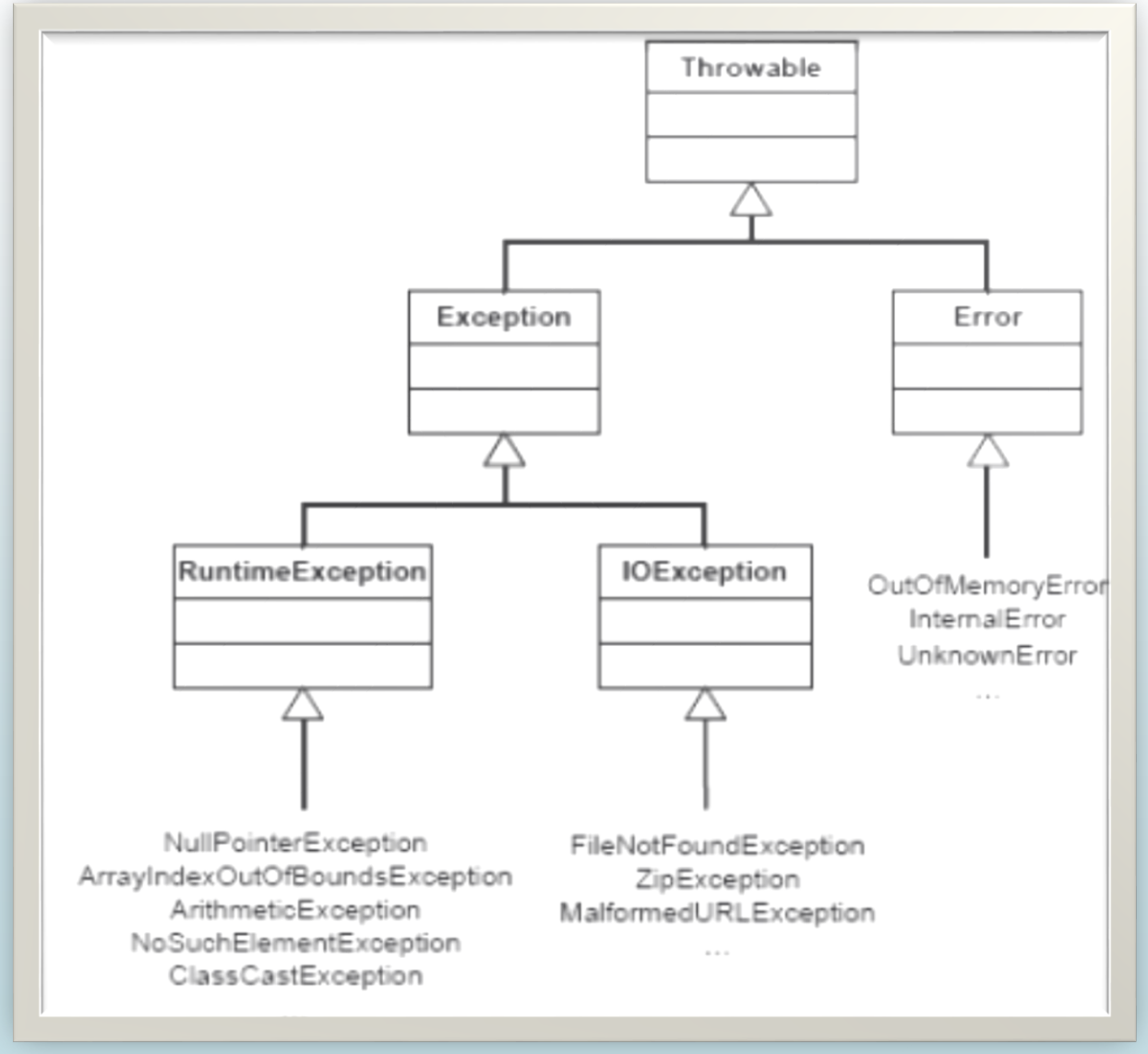
EXCEPCIÓN	Error	Manejo
De la máquina virtual (error)	Falla en memoria, no se puede cargar una clase.	Ninguno
De aplicación (checked)	Son generadas por la aplicación y detectadas en **tiempo de compilación.** • **Lectura del teclado con read() o readLine()**	Capturarlas en try-catch relanzarlas con throw
Del sistema (unchecked)	No es obligatorio atraparlas al no saber si se producirán o no en **tiempo de ejecución.** • ***acceso a objeto que no existe,*** • ***acceso a una posición de un arreglo que no existe,*** • ***división por cero***	Capturarlas en try-catch relanzarlas con throw

Hecho por Huicho :)

Clasificación

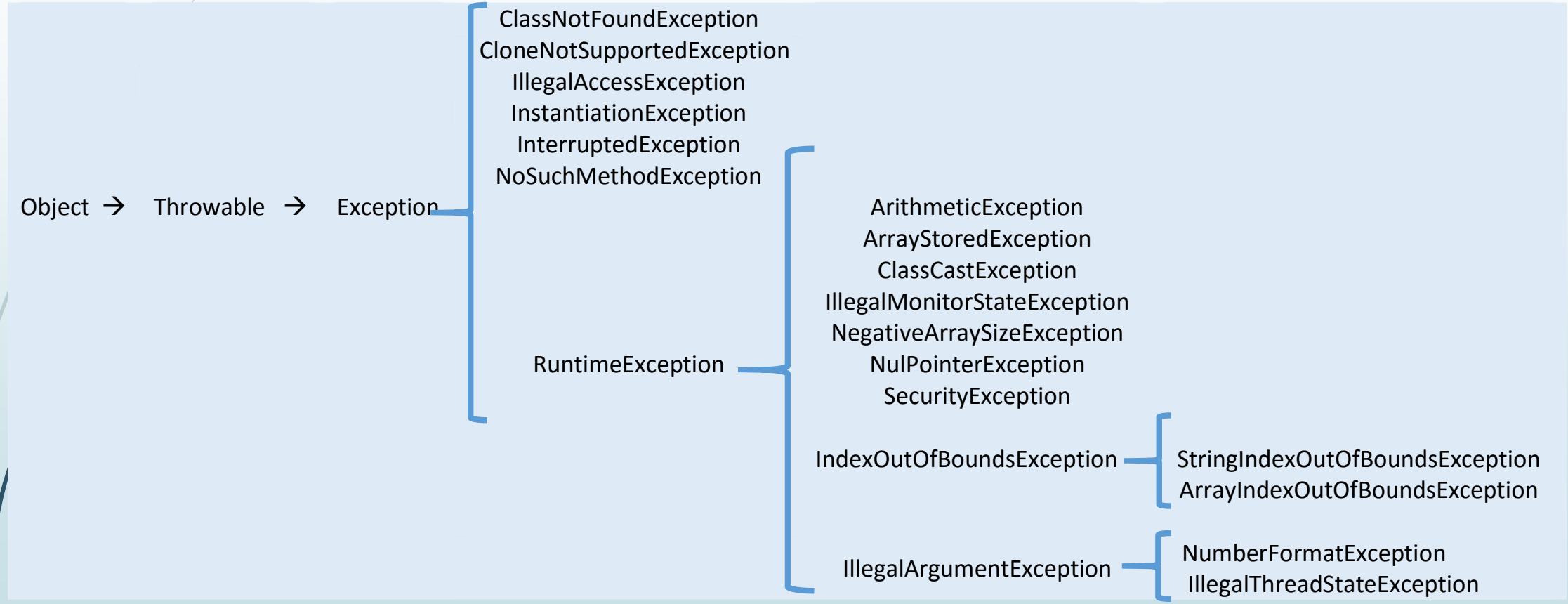
- En Java las excepciones forman una **completa jerarquía de clases**, cuya clase base es **Throwable**.
- La clase **Throwable** tiene dos subclases: **Error** y **Exception**.
- Un ***Error*** indica que se ha producido un fallo del que **no se puede recuperar la ejecución normal del programa**, por lo tanto, este tipo de errores no se pueden tratar.
- Una ***Exception*** indicará una **condición anormal que puede ser resuelta** para evitar la terminación de la ejecución del programa. Hay varias **subclases** de la clase **Exception** conocidas como *excepciones predefinidas*, y una de ellas *RuntimeException*, a su vez, tiene numerosas subclases.

Jerarquía de excepciones



Hecho por Huicho :)

Jerarquía de excepciones



Try – catch – finally

- Maneja la excepción dentro del método para evitar que termine la ejecución repentinamente.
- El bloque finally se ejecuta sin importar si la excepción se produjo o no. Se usa para cerrar archivos.
- **Caso 1:** Poner todo el de código susceptible a presentar excepciones en un bloque try y debajo colocar todos los catch posibles para manejarlas.
- **Caso 2:** Separar las instrucciones susceptibles de presentar excepción en su respectivo bloque try y el catch que lo maneje.

try – catch – finally

Caso 1

```
try {
    // Pueden producirse una o varias excepciones
}
catch( TipoExcepción1 nombreVariable ) {
    // se ejecuta para TipoExcepción1
}
catch( TipoExcepcion2 nombreVariable ) {
     // se ejecuta para TipoExcepción2
}
finally {
    // código que se ejecuta con o sin excepción
}
```

```java
public class Excepciones {

    public static void main(String[] args) {

        int valor=5, cero=0, numero=0;
        int arreglo[] = new int[]{1, 2, 3};
        String cad= " 123 ";
        String s= null;

        try {
            valor = valor/cero; //división por cero
            arreglo[4]= 5; //acceso a una posición no disponible
            numero= Integer.parseInt(cad); //convierte cadena a entero
            s.equals("casa"); //compara cadena vacia con casa
        }
        catch( ArithmeticException e ) {
            System.out.println( "Division por cero: " + e );
        }
        catch( ArrayIndexOutOfBoundsException ex ) {
            System.out.println( "Error en arreglo: " + ex );
        }
        catch( NumberFormatException en) {
            System.out.println( "Falla al convertir cadena: " + en);
        }
        catch( NullPointerException nu) {
            System.out.println( "Falla al comparar cadena nula: " + nu);
        }
        finally {
            System.out.println("\nSe manejaron varias excepciones");
        }
    }
}
```

```
<terminated> Excepciones [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (13/
Division por cero: java.lang.ArithmeticException: / by zero

Se manejaron varias excepciones
```

Si hay varias instrucciones que causen excepción se detiene en la primera y no hace el resto del código del bloque try

try – catch – finally

Caso 2

```
try {
    // bloque de código con excepción tipo1
}
catch( TipoExcepción1 nombreVariable ) {
    // se ejecuta para TipoExcepción1
}
try {
     // bloque de código con excepción tipo2
}
catch( TipoExcepcion2 nombreVariable ) {
     // se ejecuta para TipoExcepción2
}
finally {
    // código que se ejecuta con o sin excepción
}
```

```java
public class ExcepcionesV2 {

    public static void main(String[] args) {

        int valor=5, cero=0, numero=0;
        int arreglo[] = new int[]{1, 2, 3};
        String cad= " 123 ";

        try {
            valor = valor/cero; //división por cero
        }
        catch( ArithmeticException e ) {
            System.out.println( "Division por cero: " + e );
        }

        try {
            arreglo[4]= 5; //acceso a una posición no disponible
        }
        catch( ArrayIndexOutOfBoundsException ex ) {
            System.out.println( "Error en arreglo: " + ex );
        }

        try {
            numero= Integer.parseInt(cad); //convierte cadena a entero
        }
        catch( NumberFormatException en) {
            System.out.println( "Falla al convertir cadena: " + en);
        }
        finally {
            System.out.println("\nSe manejaron varias excepciones");
        }
    }
}
```

```
Division por cero: java.lang.ArithmeticException: / by zero
Error en arreglo: java.lang.ArrayIndexOutOfBoundsException: 4
Falla al convertir cadena: java.lang.NumberFormatException: For input string: " 123 "

Se manejaron varias excepciones
```

Si hay varias instrucciones que causen excepción se validan todas en diferentes bloques try y manda a sus respectivos catch

throws

- Le digo al compilador que tomaré en cuenta la excepción para que me deje proceder pero no le doy ningún manejo.
- Se relanza para ser manejada por otro método.
- En la definición del método se agrega después la palabra throws y el nombre del tipo de excepción.

throws

```
tipoDevuelto nombreMetodo(argumentos) throws listaExcepciones {
    /* cuerpo del método */
 }

public void ejemploExcep () throws ArithmeticException {
    /* cuerpo del método */
 }

public static void main(String args[]) throws IOException {
    /* cuerpo del método */
 }
```

throw

- Se lanza explícitamente una excepción o un objeto de tipo Throwable.

- Cuando se lanza la excepción se sale del bloque de código actual y busca si tiene asociada una clausula catch que la maneje, de lo contrario sale del método hasta encontrarla o en su defecto termina la ejecución.

throw

```
try {
    //se crea un objeto de la subclase ArithmeticException() y se
    // lanza por throw para manejar la excepcion
    throw new ArithmeticException();
}


catch( ArithmeticException cero_real)
{
    System.out.println("Falla al dividir un flotante entre 0: " +
                                cero_real);
}
```

throw

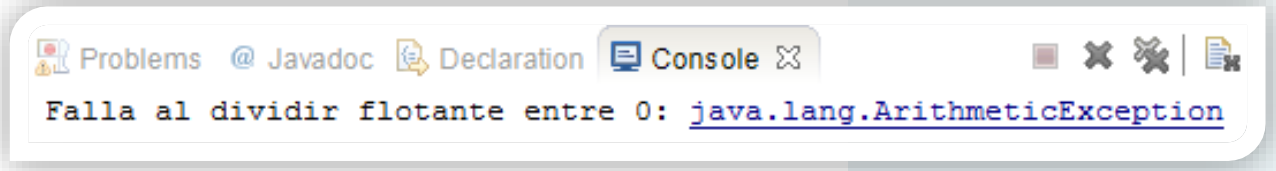
```java
public class Lanzar {

    public static void main(String[] args) {

        float cero=5, uno=0, div=0;

        try{
            if(uno!=0){
                div= cero/uno;
                System.out.println(cero + " / " + uno + "= " + div);
            }
            else{
                //se crea un objeto de la subclase ArithmeticException()
                // y se lanza por throw para manejar la excepcion
                throw new ArithmeticException();
            }
        }
        catch( ArithmeticException cero_real)
        {
            System.out.println("Falla al dividir flotante entre 0: " + cero_real);
        }
    }

}
```



Impresión de mensaje de la excepción

- El **objeto de la clase** acompañado de un mensaje propio.
- Con el **método *toString*** que devuelve el nombre de la clase que describe la excepción acompañado del mensaje asociado.
- Extrayendo únicamente el mensaje de la clase mediante el **método *getMessage***

Impresión de mensaje de la excepción

```
public class Cero {

    public static void main(String[] args) {

        int valor= 5;

        try{
            valor= valor/0;
        }
        catch(ArithmeticException cero)
        {
            System.out.println("Pegando excepcion: " + cero);
            System.out.println("Excepcion y descripcion: " + cero.toString());
            System.out.println("Mensaje de la excepcion: " + cero.getMessage());
        }
    }
}
```

método *toString*

Objeto de la clase

método *getMessage*

```
Pegando excepcion: java.lang.ArithmeticException: / by zero
Excepcion y descripcion: java.lang.ArithmeticException: / by zero
Mensaje de la excepcion: / by zero
```

Lectura del teclado

Cosa (Clase)	Definición	Q hace
InputStream (System.in)	`System.in.read()`	Lee bytes de teclado, archivo u otro dispositivo de entrada

Reader	`InputStreamReader isr;`
	`isr = new InputStreamReader(System.in);`
	
	`int c=0;`
	`c= isr.read();`
		Lee caracteres pero deben tomarse del flujo de entrada en bytes y convertirse.
		Da los caracteres sueltos si no se dice cuantos ni limite

BufferedReader	`InputStreamReader isr;`
	`isr = new InputStreamReader(System.in);`
	`BufferedReader br = new BufferedReader (isr);`
	
	`String texto = br.readLine();`
		Toma una línea completa de caracteres a partir del Reader anterior hasta el fin o salto

Excepción predefinida

- El compilador Java **obliga al programador a proporcionar el código de manejo o control de algunas de las excepciones predefinidas** por el lenguaje.
- Puede manejar dentro del método con try-catch o que sea manejado por otro con throws.
- Se presenta por ejemplo al leer de la entrada estándar.
- Revisar en la documentación del api si las clases o métodos a usar deben manejar alguna excepción.

Excepción predefinida - IOException

- Al usar un método de lectura desde el teclado como read() no compilará y mandará un error hasta manejar la excepción **IOException.**
- Para manejar clase IOException debe importarse el paquete de entrada y salida:

```
import java.io.*      ó      import java.io.IOException
```

```
while( (c= isr.read()) != '\n' )
```

```
Unhandled exception type IOException
d()) != '\n' )
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Unhandled exception type IOException
```

System.in

- **Paquete:** `java.io`
- **Clase:** `InputStream`
- **Método:** `read()`
- **Devuelve:** **byte** leído como **int** en un rango de **0 a 255**

  -1 en caso de encontrar EOF (final de archivo)

read

```
public abstract int read()
                 throws IOException
```

Reads the next byte of data from the input stream. The value byte is returned as an `int` in the range `0` to `255`. If no byte is available because the end of the stream has been reached, the value `-1` is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

A subclass must provide an implementation of this method.

Returns:

the next byte of data, or `-1` if the end of the stream is reached.

Throws:

`IOException` - if an I/O error occurs.

System.in

```java
import java.io.*;

public class LecturaByte {

    public static void main(String[] args) throws IOException{
        // TODO Auto-generated method stub

        //throws IOException xq tratamos de leer con read
        int c=0;
        // se lee por bytes hasta encontrar el fin de línea
        while( (c = System.in.read() ) != '\n' )
        {
            //se convierte el decimal del codigo a caracter
            System.out.println( c );
            //System.out.println( (char)c );
        }
    }
}
```

```
hola
104
111
108
97
13
```

Haciendo un casting resulta en el carácter leído

```java
System.out.println( (char)c );
```

```
hola
h
o
l
a
```

InputStreamReader

- **Paquete:** `java.io`
- **Clase:** `InputStreamReader`
- **Método:** `read()`
- **Devuelve:** **char** leído como **int** en un rango de **0 a 65535 (Unicode)**

  -1 en caso de encontrar EOF (final de archivo)

read

```
public int read()
         throws IOException
```

Reads a single character.

Overrides:

`read` in class `Reader`

Returns:

The character read, or -1 if the end of the stream has been reached

Throws:

`IOException` - If an I/O error occurs

Hecho por Huicho :)

InputStreamReader

```java
import java.io.*;

public class LecturaChar {

    public static void main(String[] args) throws IOException{
        // TODO Auto-generated method stub

        //throws IOException xq tratamos de leer con read
        int c=0;
        InputStreamReader letra = new InputStreamReader(System.in);
        //se leen caracteres pero se guardan como enteros
        //se detiene al final de la linea
        while( (c= letra.read()) != '\n' )
        {
            System.out.println( c );
            //System.out.println( (char)c );
        }
    }
}
```

```
hola
104
111
108
97
13
```

Haciendo un casting resulta en el carácter leído

```java
System.out.println( (char)c );
```

```
hola
h
o
l
a
```

Excepción predefinida - IOException

Las excepciones de este tipo deben ser capturadas o se manda un error en tiempo de compilación.

Se maneja la excepción dentro del método donde se genera.

```
public static void main(String[] args) {

  int c=0;

  InputStreamReader isr = new InputStreamReader(System.in);

  try{
   while( (c= isr.read()) != '\n' )
      System.out.print( (char)c);
  }
  catch(IOException e)
  {
   System.out.println("El error al leer es: " + e);
  }
}
```

Si el método no captura la excepción debe especificar que puede lanzarla.

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

   int c=0;

   InputStreamReader isr = new InputStreamReader(System.in);

   while( (c= isr.read()) != '\n' )
    System.out.print( (char)c);
}
```

BufferedReader

- **Paquete:** `java.io`
- **Clase:** `BufferedReader`
- **Método:** `readLine()`
- **Devuelve:** **String** con el renglón leído sin carácter de terminación

readLine

```
public String readLine()
                throws IOException
```

Reads a line of text. A line is considered to be terminated by any one of a line feed ('\n'), a carriage return ('\r'), or a carriage return followed immediately by a linefeed.

Returns:

A String containing the contents of the line, not including any line-termination characters, or null if the end of the stream has been reached

Throws:

`IOException` - If an I/O error occurs

See Also:

`Files.readAllLines(java.nio.file.Path, java.nio.charset.Charset)`

BufferedReader

```java
import java.io.*;

public class LecturaBuffer {

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

        //throws IOException xq tratamos de leer con readLine

        InputStreamReader letra = new InputStreamReader(System.in);
        BufferedReader entrada = new BufferedReader(letra);

        String Mensaje;
        int Entero=0;
        float Real=0;

        System.out.println("Escribe un mensaje a mostrar");
        //readLine recibe datos de tipo String
        Mensaje = entrada.readLine();
        System.out.println(Mensaje);

        System.out.println("Escribe un entero a mostrar");
        //convertir String a entero
        Entero = Integer.parseInt(entrada.readLine());
        System.out.println(Entero);

        System.out.println("Escribe un real a mostrar");
        //convertir String a float
        Real = Float.parseFloat(entrada.readLine());
        System.out.println(Real);
        System.out.printf("%.2f",Real);
    }
}
```

```
Escribe un mensaje a mostrar
Hola
Hola
Escribe un entero a mostrar
5
5
Escribe un real a mostrar
4.5789
4.5789
4.58
```

- Si se desea un valor numérico convertir la entrada usando el método del tipo de dato de referencia correspondiente.

```
Integer.parseInt();
```

```
Float.parseFloat();
```

Excepciones más frecuentes

Excepción	Descripción	Ejemplo
ArithmeticException	Dividir entre cero (***solo con int***) En float devuelve Infinity	`int valor=5;` `valor= valor/0;`
ArrayIndexOutOfBoundsException	Intento de acceso a un elemento de arreglo no declarado	`int arreglo[] = new int[]{1, 2, 3};` `arreglo[3]= 5;`

NumberFormatException	Convertir cadena que no tiene caracteres numéricos (***solo con int***)	`String mensaje= “ 123 “;`
		`int numero=0;`
		`numero= Integer.parseInt(mensaje);`

NullPointerException	Acceso a atributos o métodos nulos o no inicializados	`String s=null;` `s.equals(“casa”);`
NegativeArraySizeException Hecho por Huicho :)	Intento de creación de un vector con un número negativo de elementos	`float cuadritos[]= new float[-2];`

Excepciones definidas por el usuario

- Además de las excepciones predefinidas se pueden crear otras:
  - Deben heredar de Throwable o de Exception o de alguna de sus subclases.
  - Darle funcionalidad.
  - Definir el constructor de la superclase con un mensaje de entrada.

```
public class MiExcepcion extends ArithmeticException{

    //constructor
    public MiExcepcion(String mensaje)
    {
        super(mensaje);
    }
}
```

Excepciones definidas por el usuario

- Cualquier clase puede lanzar la excepción creada.
- Lanzamiento desde el método main o desde otro método:

```
throw new MiExcepcion("Creada para division flotante entre cero");
```

- La impresión al presentarse manda el nombre de la excepción y el mensaje escrito en el constructor:

```
MiExcepcion: Creada para division flotante entre cero
```

Excepciones definidas por el usuario

```
public class PruebaMiExcepcion {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        float cero= 4.5f, uno=0, div=0;

        try{
            if(uno!=0)
            {
                div= cero/uno;
                System.out.println(cero + " / " + uno + "= " + div);
            }
            else
            {
                //se crea un objeto de MiExcepcion y se lanza por throw pasando un mensaje
                throw new MiExcepcion("Creada para division flotante entre cero");
            }
        }
        catch( MiExcepcion cero_real)
        {
            System.out.println("Excepcion: " + cero_real);
        }
    }
}
```

Problems @ Javadoc Declaration Console

```
<terminated> PruebaMiExcepcion [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (07/04/2014 01:27:52)
Excepcion: MiExcepcion: Creada para division flotante entre cero
```

Referencias

- http://darkbyteblog.wordpress.com/2011/03/08/java-flujos-de-datos-entrada-y-salida-estandar/
- http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/archivos/teclado.htm
- http://unicode-table.com/en/