

2.1. Normas básicas de programación

2.1.1. Normas obligatorias

Las siguientes normas son necesarias si queremos que nuestros programas compilen:

- Los identificadores diferencian entre mayúsculas y minúsculas.
- Deben comenzar con una letra, subrayado (_) o símbolo de dólar (\$).
- No puede usarse como identificador una palabra reservada.
- Los caracteres posteriores al primero pueden ser números.
- No existe longitud mínima.

Identificadores correctos	Identificadores incorrectos
contador	12contador
clase	class
_nombre	1245nombre
\$nombre	3

2.1.2. Normas recomendadas

Estas normas son una serie de convenios que se han adoptado en el mundo de los programadores Java:

- Los nombres de clases empiezan con mayúsculas.
- La primera letra de las variables y métodos con minúscula.
- Los nombres compuestos se unen con una letra mayúscula en el comienzo de cada palabra.
- Se usan nombres largos y significativos. Por ejemplo para una variable donde guardar un nombre de usuario, una buena idea sería denominarla "nombreUsuario". Sería un error denominarla u.

Siguiendo las recomendaciones	Sin seguirlas
ClasePersona	clasepersona
cuantos //atributo	Cuantos //atributo
saldoCuenta()	saldocuenta()
calcularInteresCuenta()	calcularinteres cuenta()

Para realizar comentarios en nuestros programas tenemos tres opciones:

```
// Comentario de una sola línea  
/* Comentario de  
varias líneas*/  
/** Comentario de documentación
```

de una o varias líneas*/

2.2. Tipos de datos y variables

La declaración de variables en Java tiene la siguiente sintaxis:

```
tipoDeDato nombreVariable;
```

```
tipoDeDato nombreVariable=valorInicial;
```

La asignación en Java se realiza con el símbolo '=':

```
variable = valor;
```

2.2.1. Enteros

Representan valores de tipo int y long (por defecto int). Si queremos especificar que sea long debemos ponerle una "l" o "L" al final del número.

Hay tres tipos: decimal, octal y hexadecimal.



Ejemplos:

```
int vueltas;
```

```
int contador = 0;
```

```
int a,b,c,d,e = 6;    // Sólo se inicializa la variable 'e'.
```

```
long iteraciones = 123456789L;
```

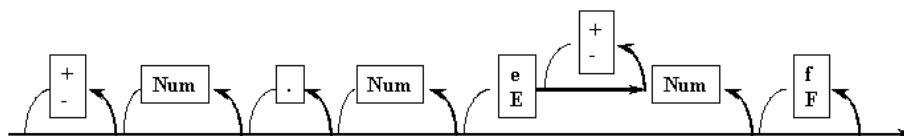
Tipo de dato	Decimal	Octal	Hexadecimal
int (32 bits)	27	077	0xDC00
long (64 bits)	35451096942L	04762123L	0x9823740149L

2.2.2. Reales

Representan números decimales con partes fraccionarias, es decir, valores de tipo float o double (por defecto double). Ejemplos:

```
634.3346    // Standard
```

```
6.343346e2  // Científica
```



Ejemplos:

```
double temperatura;
```

```
double temperatura = 23.65;    // notación estándar
```

```
double decimal1 = 345.23e12;    // notación científica
```

```
float decimal2 = 0.545;    // Mal.
```

```
float decimal2 = 0.545f;    // Correcto.
```

2.2.3. Booleanos

Como es habitual pueden tener los valores 'verdad' y 'falso'. Estos valores se corresponden con las palabras reservadas del lenguaje 'true' y 'false'.



Ejemplos:

- `boolean ok = true;`
- `boolean valido = false;`

2.2.4. Caracteres

Representan valores de tipo char (caracteres Unicode). A continuación se muestran los caracteres no imprimibles.

Secuencia de caracteres

Salida por pantalla

<code>\\</code>	Barra invertida
<code>\f</code>	Salto de página
<code>\b</code>	Retroceso
<code>\r</code>	Retorno de carro
<code>\t</code>	Tabulador horizontal
<code>\n</code>	Línea nueva
<code>\'</code>	Comillas simples
<code>\"</code>	Comillas dobles

Ejemplos:

```
char carac;  
char variableCaracter = 'A';  
char carac = 'P';  
char a, b, c = 'A';
```

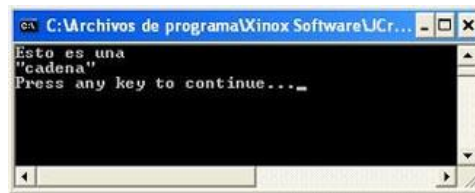
2.2.5. Cadenas

Representan objetos de tipo String. Son caracteres entre comillas dobles y se pueden incluir caracteres de escape.

Ejemplos:

```
String cadena = "Esto es una cadena literal";
```

Si mostramos por pantalla la cadena: `"Esto es una \n \"cadena\" "`. El resultado será:



2.3. Operadores

Tipos de operadores

Símbolos relacionados

Posfijos	[] . (params) expr++ expr--
Prefijos (D)	++expr --expr +expr -expr ~ !
Creación o conversión	new (tipo) expr
Multiplicación, División, Resto División	* / %
Suma	+ -
Desplazamiento de bits	<< >> >>>
Relacionales	< > <= >= instanceof
Igualdad	== !=
Y bits	&
O Exclusivo bits	^
O Inclusivo bits	
Y lógica	&&
O lógica	
Condicional	?:
Asignación (D)	= += -= *= /= %= >>= <<= >>>= &= ^= =

Reglas de aplicación de operadores

- Orden de evaluación de izquierda a derecha.
- Excepto en &&, || y ?: se evalúan todos los operandos antes de llevar a cabo la operación.
- Para cambiar la precedencia, utilizamos paréntesis.
- Para las cadenas, se pueden utilizar los operadores + y += para la concatenación.

Ejemplos:

// Declaraciones y Operadores Unarios.

int cont1 = 1, cont2 = 2;

```
cont1++;    //cont1=2
cont2--;    //cont2=1
```

//Operaciones de Incremento y de Comparación

```
int cont1 = 1, cont2 = 2;
```

```
cont1 += cont2;    // equivale a: cont1 = cont1 + cont2;
```

```
cont1 == cont2;    // false
```

```
cont1 != cont2;    // true
```

/* Descripción:Declaraciones de Tipos de Datos y Salidas por Pantalla */

```
public class App1
{
    public static void main( String args[] )
    {
        int i=1;
        long l=11;
        double d=1.5;
        float f=1.5f;
        boolean b=true;
        char c='a';
        String cadena="Esto es una \ncadena";

        System.out.println( "i:"+i );
        System.out.println( "l:"+l );
        System.out.println( "d:"+d );
        System.out.println( "f:"+f );
        System.out.println( "b:"+b );
        System.out.println( "c:"+c );
        System.out.println( "cadena:"+cadena );
    }
}
```

/* Descripción: ejemplo de utilización de operadores a nivel de bit */

```
public class Operadores
{
    public static void main( String args[] )
    {
        // DESPLAZAMIENTO BINARIO A LA DERECHA
        int entero=10; // en binario 10d= 1010
        // Realizo un desplazamiento de bits a la derecha de 2
        entero= entero>>2;
        // El resultado del desplazamiento es : 10
        System.out.println(entero);

        // DESPLAZAMIENTO BINARIO A LA IZQUIERDA
        // Realizo un desplazamiento de bits a la izquierda de 2
        entero= entero<<2;
        // El resultado del desplazamiento es : 1000
        System.out.println(entero);
    }
}
```

```
}  
}
```

2.4. Arrays

Un array es una estructura de programación que almacena un determinado número de elementos del mismo tipo. Para poder disponer y usar un array en Java hay que realizar dos pasos:

1. Declarar el array.
2. Reservar espacio para un número determinado de elementos mediante la instrucción **new**.

Una vez declarado un array, podremos acceder a él con la siguiente sintaxis:

```
int cuantos = numeros[3];
```

Para modificar una posición del array, por ejemplo la posición cero, utilizamos la siguiente sintaxis:

```
vocales[0] = 'a';
```

Ejemplos

```
int numeros[];           // Declaración del array  
numeros = new int[20];    // Reservar espacio en memoria
```

Otra forma:

```
int[] numeros = new int[20];
```

Matrices:

```
int tabla[ ][ ] = new int[10][3];  
tabla.length;      /* 10 */  
tabla[0].length;    /* 3 */
```

Más ejemplos:

```
char[] vocales = {'A', 'E', 'I', 'O', 'U'};  
String nombres[] = {"Juan", "Pepe", "Pedro", "Maria"};
```

equivale a:

```
String nombres[];  
nombres = new String[4];  
nombres[0] = new String( "Juan" );  
nombres[1] = new String( "Pepe" );  
nombres[2] = new String( "Pedro" );  
nombres[3] = new String( "María" );
```

```
/* Descripción: el siguiente código genera diez números aleatorios entre 0 y 15
y después solicita un número al usuario dentro de dicho rango. Por último informa
al usuario de si el número elegido esta en el array de los diez números aleatorios. */

import java.io.*;
import java.util.*;

public class Adivina
{
    public static void main(String args[]) throws IOException
    {
        BufferedReader entrada= new BufferedReader(new InputStreamReader(System.in));
        int i, num;
        int numeros[] = new int[10];
        boolean adivinado = false;
        Random rand = new Random();

        for (i=0;i<=9;i++)
        {
            numeros[i]=rand.nextInt(15);
        }

        //Ordenamos el array
        Arrays.sort(numeros);

        //Pedimos el número
        System.out.print("Introduzca el número (0-15):" );
        num = Integer.parseInt(entrada.readLine());

        //llamamos al método que se encuentra a continuación del main
        adivinado = adivinado(numeros, num);

        if (adivinado)
        {
            System.out.println("\nNúmero adivinado");
        }
        else
        {
            System.out.println("\nNúmero NO adivinado");
        }

        //Imprimimos todos los números
        System.out.print("\nTodos los números:");
        for (i=0; i<=9; i++)
        {
            System.out.print(" "+numeros[i]);
        }
    } // fin del main

    public static boolean adivinado(int numeros[], int num)
    {
        if (Arrays.binarySearch(numeros,num)>=0)
        {
            return true;
        }
        else
        {
            return false;
        }
    } // fin del método adivinado
}
```

2.5. Estructuras de control

2.5.1. IF

En todos los lenguajes de programación son muy importantes este tipo de sentencias.

Se utilizan para tomar decisiones en función de una condición que tiene un valor verdadero o falso. Suelen presentarse de forma anidada, es decir, un *if* contiene otro *if*.

Sintaxis del IF

```
if (expresión booleana)
    sentencias1;
```

Explicación del IF:

Si el valor de la expresión booleana (ej. contador==5), es verdadero entonces se ejecutarán las sentencias1.

Sintaxis del IF / ELSE

```
if (expresión booleana)
{
    sentencias1;
}
else
{
    sentencias2;
}
```

Explicación del IF / ELSE:

Si el valor de la expresión booleana es verdadero pasa a ejecutarse el bloque de sentencias1, sino pasará a ejecutarse el bloque de sentencias2.

```
/* Descripción: Ejemplo de utilización de sentencias if con condiciones de
verdadero y falso */
public class App3
{
    public static void main( String args[] )
    {
        int x=5, y=2;

        //igual
        if (x==5) System.out.println("x=5");
        //distinto
        if (x!=2) System.out.println("x!=2");
        //y lógico
        if (x==5 && y==2) System.out.println("x=5, y=2");
        //o lógico
        if (x==5 || y==1) System.out.println("x=5");
        //operación
        if (x+y>10)
        {
            System.out.println("x+y>10");
        }
        else
        {
            System.out.println("x+y<10");
        }
    }
}
```


2.5.2. SWITCH

Este tipo de sentencias se utilizan para decidir qué sentencias ejecutar, en función de un conjunto de valores de entrada. Solamente se puede utilizar con tipos básicos, es decir: **enteros**, **booleanos** y **caracteres**.

Sintaxis de switch

```
switch (valorDeVariable)
{
    case valor1: sentencias1;
        break;
    case valor2: sentencias2;
        break;
    case valor3: sentencias3;
        break;
    default: sentencias 4;
}
```

Se compara **valorDeVariable** con los valores de los bloques **case**, si coincide con alguno de los valores de los bloques **case**, pasa a ejecutarse dicho bloque. Si el valor de **valorDeVariable** no coincide con ninguno de los listados en las etiquetas **case**, se ejecuta el bloque asociado a la parte **default**. El bloque **default** es opcional, puede no aparecer, pero es aconsejable colocar un bloque **default** en cada **switch**, para cubrir todo el dominio de entrada.

```
/** Descripción: aplicación donde se muestra la utilización
 * de sentencias switch. Y también como manejar los datos
 * de entrada/salida del usuario.
 */
import java.io.*;

class Medallas
{
    /* Al main hay que añadirle el throws, por si se produce algún
    error de entrada/salida. */
    public static void main( String args[] ) throws IOException
    {
        String medalla=new String();
        //Objeto para leer una cadena del teclado
        BufferedReader entrada=
            new BufferedReader(new InputStreamReader(System.in));

        int posicion=0;

        String nombre=new String();

        System.out.print( "\nIntroduce tu nombre: " );
        // con este metodo leo por teclado una cadena de caracteres
        nombre = entrada.readLine();

        System.out.print( "\nIntroduce tu posicion: " );
        //conversión de cadena a número
        posicion = Integer.parseInt(entrada.readLine());

        switch (posicion)
        {
            case 1:
                medalla = "oro";
                break;
            case 2:
                medalla = "plata";
                break;
            case 3:
                medalla = "bronce";
                break;
            default:
                medalla = "sin medalla";
                break;
        }
    }
}
```

```
// si no ha quedado en el podium
if (medalla.equals("sin medalla"))
{
    System.out.println( "Lo sentimos pero no tienes medalla." );
}
else // tiene medalla
{
    System.out.println( "D. "+nombre+" ha conseguido la medalla de "+medalla );
}
}
}

/** Descripción: a continuación se muestra un ejemplo básico
 * de utilización del switch. El usuario debe modificar el valor de la
 * variable valorEntero, para ver como varia la salida por pantalla.
 */

import java.io.*;

class Switch1
{
    /* Al main hay que añadirle el throws, por si se produce algún
    error de entrada/salida. */
    public static void main( String args[] ) throws IOException
    {
        String mensaje="";
        int valorEntero=2;

        switch (valorEntero)
        {
            case 1: mensaje = "Uno";
                break;
            case 2: mensaje = "Dos";
                break;
            case 3: mensaje = "Tres";
                break;
            default: mensaje = "desconocido";
        }
        System.out.println( "El valor de mensaje es: "+mensaje );
    }
}
```

2.5.3. FOR

Sintaxis de bucle for

```
for (expr1 inicialización; expr2 test; expr3 incremento)
    { sentencias; }
```

A continuación se explican cada una de las tres partes de la sentencia for:

- **Expresión de inicialización:** En esta parte del bucle se inicializa la variable de la que va a depender la iteración. En el ejemplo del bucle for es la variable contador la que se inicializa.
- **Condición de parada o test:** En esta parte se evalúa una condición que determina si el bucle se ejecuta o no. En caso de que el resultado de la expresión sea true el bloque de código sentencias se ejecutará. En el ejemplo del bucle for el código se ejecutará, si el valor de contador es menor o igual que 12.
- **Incremento:** en esta parte se modifica el valor del que depende la condición de parada. En el ejemplo del bucle for se incrementa el valor de la variable contador
- **Sentencias:** Este es el bloque de sentencias que se ejecutarán en caso de que se satisfaga la condición del bucle for.

Ejemplo:

```
int contador;
for( contador=1; contador <= 12; contador++ )
{
    System.out.println("Iteración" + contador);
}
```

```
/* Descripción: varios ejemplos de utilización de bucles FOR*/

public class App4
{
    public static void main( String args[] )
    {
        int arrayint[] = new int[3];
        char arraychar[] = {'a', 'b', 'c'};

        //rellenamos el array de enteros
        for (int i=0; i<3; i++)
        {
            arrayint[i]=i;
        }

        //imprimimos su valor
        for (int i=0; i<3; i++)
        {
            System.out.println( "arrayint["+i+"]: "+arrayint[i] );
        }

        //incrementamos en 1 cada valor
        for (int i=0; i<3; i++)
        {
            arrayint[i]+=1;
            System.out.println( "arrayint["+i+"]: "+arrayint[i] );
        }
    }
}
```

```
//incrementamos en 1 cada letra
for (int i=0; i<3; i++)
{
    arraychar[i]+=1;
}

//imprimimos su valor
for (int i=0; i<3; i++)
{
    System.out.println( "arraychar["+i+"]: "+arraychar[i] );
}
}

/* Descripcion: ejemplo de bucle for que pinta por pantalla doce veces
un contador que va incrementando */

public class For1
{
    public static void main( String args[] )
    {
        int contador;
        for( contador=1; contador <= 12; contador++ )
        {
            System.out.println("Pasada =" + contador);
        }
    }
}
```

2.5.3. DO / WHILE

Sintaxis del bucle do / while (existen dos formas de uso posibles):

<pre>while (expresión booleana) { sentencias }</pre>	<pre>do { sentencias } while (expresión booleana);</pre>
<p>Si se cumple la expresión booleana pasa a ejecutarse el bloque de código sentencias.</p> <p>Este bloque será el que se ejecute cuando se cumpla la condición del bucle.</p>	<p>El bloque de código sentencias se ejecuta incondicionalmente una vez al entrar en el bucle (aún no se ha evaluado la condición).</p> <p>A continuación, y mientras se cumpla expresión booleana, se ejecuta de nuevo el bloque sentencias que hay a continuación del do.</p>

Ejemplos:

```
int contador=1;
while( contador <= 12 )
{
    System.out.println("Iteración" + contador);
    contador++;
}
```

```
/* Descripción: Ejemplo básico de utilización de DO-WHILE */
public class While1
{
    public static void main( String args[] )
    {
        int i=10;
        do
        {
            System.out.println( "i:"+i-- );
        } while (i>0);
    }
}
```

```
/* Descripción: Ejemplo básico de utilización de WHILE */
public class While2
{
    public static void main( String args[] )
    {
        int i=10;

        while (i>0)
            System.out.println( "i:"+i-- );
    }
}
```

2.5.5. OTROS

return

Termina la ejecución de un método y devuelve un valor del tipo de retorno.

```
return [expresion];
```

break

Permite salir de cualquier bloque. Se suele utilizar para interrumpir un bucle while, do o for, o bien una sentencia switch. Por defecto, sale del bucle más interno, pero con etiquetas, podemos hacer que salga de cualquier bucle anidado.

```
break [etiqueta];
```

Ejemplos:

```
uno: for(;;) // etiqueto esta sentencia, como "uno"
{
    dos: for(;;) // etiqueto esta sentencia como "dos"
    {
        System.out.println("Iteracion");
        break; // se saldría del bucle "dos"
    }
    break;
}
```

```
void f() {
    ...
    return;
}
```

```
double op(double x, double y) {  
    ...  
    return x * y / (x + y);  
}
```

```
int func(){  
    if( a == 0 ) return 1;  
    return 0; // es imprescindible porque retorna int  
}
```

```
/* Descripción: programa que ilustra el funcionamiento de  
   las sentencias de control de flujo BREAK. */  
  
public class Otros  
{  
    public static void main( String args[] )  
    {  
        uno: for(;;) // etiqueto esta sentencia, como "uno"  
        {  
            dos: for(;;) // etiqueto esta sentencia como "dos"  
            {  
                System.out.println("Iteracion");  
                break; // se saldría del bucle dos  
            }  
            break; // se saldría del bucle uno  
        }  
    }  
}
```

2.6. Excepciones en Java

2.6.1. Introducción

¿Qué es el mecanismo de las excepciones?

Java implementa excepciones para facilitar la construcción de código robusto. Cuando ocurre un error en un programa, el código que encuentra el error lanza una excepción, que se puede capturar y recuperarse de ella. Java proporciona muchas excepciones predefinidas.

Ejemplo de cómo capturar excepciones:

```
try
{
    sentencias Try;
}
catch( Exception e)
{
    sentencias Catch;
}
```

Sentencias Try: Si en estas sentencias se produce una excepción del estilo de la división por cero, la ejecución salta al bloque catch, donde es tratada como el programador desee. Uno de los tratamientos más habituales es informar al usuario del error.

Sentencias Catch: Estas sentencias se ejecutan en caso se saltar una excepción, de modo que el programador puede informar al usuario del error producido, con un mensaje por pantalla.

```
/* Descripción: ejemplo de captura de varias excepciones y
de finally*/

public class EjExcepcionBien
{
    public static void main (String args[])
    {
        int i = 0;
        String cadenas[] = {
            "Cadena 1",
            "Cadena 2",
            "Cadena 3",
            "Cadena 4"
        };

        try {
            for (i=0; i<=4; i++) System.out.println(cadenas[i]);
        } catch( ArrayIndexOutOfBoundsException ae ) {
            System.out.println("\nError: Fuera del indice del array\n");
        } catch( Exception e ) {
            // Captura cualquier otra excepción
            System.out.println( e.toString() );
        } finally {
            System.out.println( "Esto se imprime siempre." );
        }
    }
}
```



```

}

/* Descripción: Ejemplo de tratamiento de excepciones con throws */

import java.io.*;
public class ExcepcionThrows {
    public static void main (String args[]) {
        double op1, op2, resd;
        //bucle infinito aunque se produzca una excepción
        while (true)
        {
            try {
                System.out.println("\n##### Nueva División #####");
                System.out.print("\nNumerador: ");
                op1 = capturanum();
                System.out.print("\nDenominador: ");
                op2 = capturanum();
                if (op2 == 0) throw new ArithmeticException();
                resd = op1 / op2;
                System.out.println( "\nResultado: " + Double.toString(resd) );
            } catch ( ArithmeticException ae ) {
                System.out.println("\nError aritmético: " + ae.toString());
            } catch( NumberFormatException nfe ) {
                System.out.println("\nError de formato numérico: " + nfe.toString());
            } catch( IOException ioe ) {
                System.out.println( "\nError de entrada/salida: " + ioe.toString() );
            } catch( Exception e ) {
                // Captura cualquier otra excepción
                System.out.println( e.toString() );
            }
        }
    }

    public static double capturanum() throws NumberFormatException, IOException
    {
        BufferedReader entrada=
            new BufferedReader(new InputStreamReader(System.in));

        String txt = entrada.readLine();
        double num = Double.parseDouble(txt);
        return num;
    }
}

```

2.6.2. Utilidad

- Cuando sucede un evento anormal en la ejecución de un programa y lo detiene decimos que se ha producido una excepción. Cualquier programa bien escrito debe ser capaz de tratar de manera inteligente las posibles excepciones que pueden ocurrir en su ejecución y de recuperarse, si es posible, de ellas.
- Con los mecanismos de recuperación ante excepciones construimos programas robustos y con capacidad de recuperación ante errores más o menos previsibles que se pueden producir en el momento en que se ejecuta el programa.
- En Java una excepción es un objeto que avisa que ha ocurrido alguna condición inusual. Existen muchos objetos de excepción predefinidos, y también podremos crear los nuestros propios.
- Cuando se capturan excepciones en Java las sentencias que pueden causar un error se deben insertar en un bloque formado por try y catch, a continuación, en catch debemos tratar esos posibles errores. También existe la posibilidad de lanzar una excepción cuando se produzca una determinada situación en la ejecución del programa, para hacerlo utilizaremos la instrucción throw.

2.6.3. Try

Define un bloque de código donde se puede generar una excepción. El bloque try va seguido inmediatamente de uno o más bloques catch y opcionalmente de una cláusula finally. Cuando se lanza una excepción el control sale del bloque try actual y pasa a un manejador catch apropiado.

Se pueden presentar dos situaciones diferentes a la hora de definir el bloque try:

1. Podemos tener más de una sentencia que generen excepciones, en cuyo caso podemos definir bloques individuales para tratarlos.
2. Podemos tener agrupadas en un mismo bloque try varias sentencias que puedan lanzar excepciones, con lo que habría que asociar múltiples controladores a ese bloque.

Será la decisión del programador utilizar una forma u otra de controlar las excepciones.

2.6.4. Catch

Define el bloque de sentencias que se ejecutarán cuando se haya producido una excepción en un bloque try.

La sintaxis general de la sentencia catch en Java es la siguiente:

```
catch( TipoExcepcion nombreVariable )
{
    // sentencias Java
}
```

Se pueden colocar sentencias catch sucesivas, cada una controlando una excepción diferente.

No debería intentarse capturar todas las excepciones con una sola cláusula ya que representa un uso demasiado general y podrían llegar muchas excepciones.

```
catch ( Exception e ) { . . . }    // Captura genérica.
```

2.6.5. Throw

La sentencia throw se ejecuta para indicar que ha ocurrido una excepción, lo que se suele denominar como lanzamiento de una excepción. La sentencia throw especifica el objeto que se lanzará. La forma general de la sentencia throw es:

```
throw ObjetoThrowable;
```

El flujo de la ejecución se detiene inmediatamente después de la sentencia throw, y nunca se llega a la sentencia siguiente, ya que el control sale del bloque try y pasa a un manejador catch cuyo tipo coincide con el del objeto. Si se encuentra, el control se transfiere a esa sentencia. Si no, se inspeccionan los siguientes bloques hasta que el gestor de excepciones más externo detiene el programa.

2.6.6. Throws

Con throws un método lista las excepciones que puede lanzar, y que no va a manejar. Esto sirve para que todos los métodos que lo llamen puedan colocar protecciones frente a esas excepciones.

Para la mayoría de las subclases de la clase Exception, el compilador Java obliga a declarar qué tipos de excepciones podrá lanzar un método.

Si un método lanza explícitamente una instancia de Exception o de sus subclases, se debe declarar su tipo con la sentencia throws. La declaración del método sigue ahora la sintaxis siguiente:

```
tipo NombreMetodo( argumentos ) throws excepciones
{. . .}
```

2.6.7. Definidas

En Java el usuario puede definir sus propias excepciones:

- A continuación se adjunta un programa que hace uso de una excepción definida por el usuario.
- El programa realiza la media de una serie de notas introducidas por el usuario, cuando este intenta poner una nota inferior a 0 o superior a 10, salta la excepción NotaMal.
- También se han tenido en cuenta otras posibles excepciones predefinidas como errores de formato numérico, errores de entrada salida, etc.
- Con este código se ha intentado proporcionar un ejemplo donde se utilice todo lo visto en cuanto a excepciones en Java.

```
/* Descripción: código ejemplo JAVA que utiliza todo lo visto en cuanto a
   excepciones Java, incluso excepciones definidas por el usuario.*/

import java.io.*;

public class Notas
{
    private static String notatxt="";

    public static void main( String args[] )
    {
        double media=0, total=0, notanum=0;
        int contador=0;
        BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
        while ( ! notatxt.equals("Z") )
        {
            try {
                System.out.print( "\nTeclee calificación (1-10), Z para terminar: " );
                notanum = capturatum();
                if (notanum < 0 || notanum > 10) throw new NotaMal();
                total += notanum;
                contador = contador + 1;
            } catch (NotaMal nm) {
                System.out.println( "\n" + nm.getMessage() );
            } catch( NumberFormatException nfe ) {
                if ( ! notatxt.equals("Z") )
                    System.out.println("\nError de formato numérico: " + nfe.toString());
            } catch( IOException ioe ) {
                System.out.println( "\nError de entrada/salida: " + ioe.toString() );
            } catch( Exception e ) {
                // Captura cualquier otra excepción
                System.out.println( e.toString() );
            }
        }
    }
}
```

```

    }
}

if ( contador != 0 ) {
    media = (double) total / contador;
    System.out.println( "\nEl promedio del grupo es: " + media );
}
else
    System.out.println( "\nNo se introdujeron calificaciones." );
}

public static double capturanum() throws NumberFormatException, IOException
{
    BufferedReader entrada=
        new BufferedReader(new InputStreamReader(System.in));

    notatxt = entrada.readLine().toUpperCase();
    double num = Double.parseDouble(notatxt);
    return num;
}
}

class NotaMal extends Exception {
    public NotaMal()
    {
        super( "Excepción definida por el usuario: NOTA INCORRECTA." );
    }
}
}

```

2.6.8. Comunes

Excepciones predefinidas más comunes:

ArithmeticException	Las excepciones aritméticas son típicamente el resultado de una división por 0.
NullPointerException	Se produce cuando se intenta acceder a una variable o método antes de ser definido.
ClassCastException	El intento de convertir un objeto a otra clase que no es válida.
NegativeArraySizeException	Puede ocurrir si se intenta definir el tamaño de un array con un número negativo.
ArrayIndexOutOfBoundsException	Se intenta acceder a un elemento de un array que está fuera de sus límites.
NoClassDefFoundException	Se referenció una clase que el sistema es incapaz de encontrar.